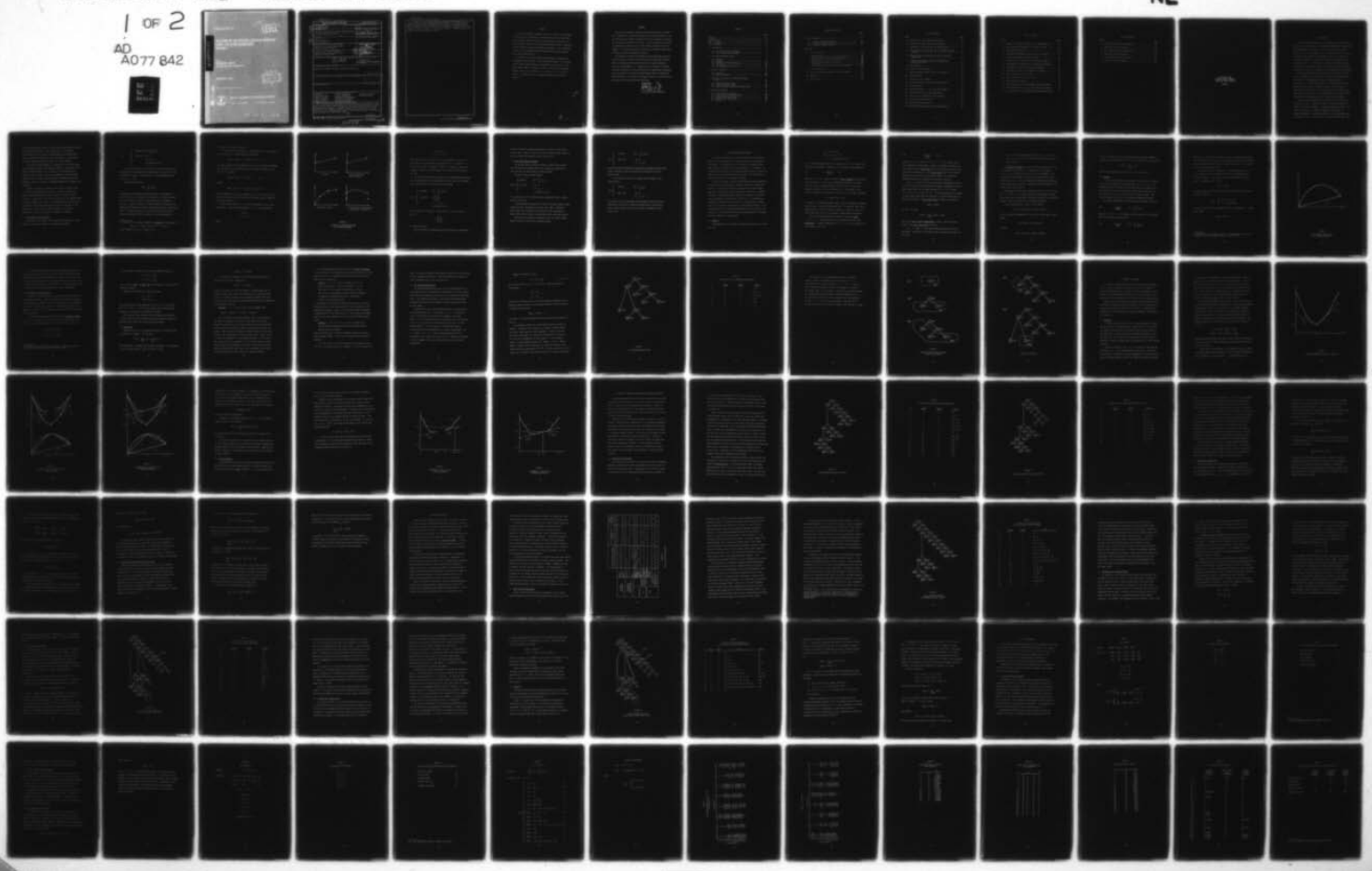
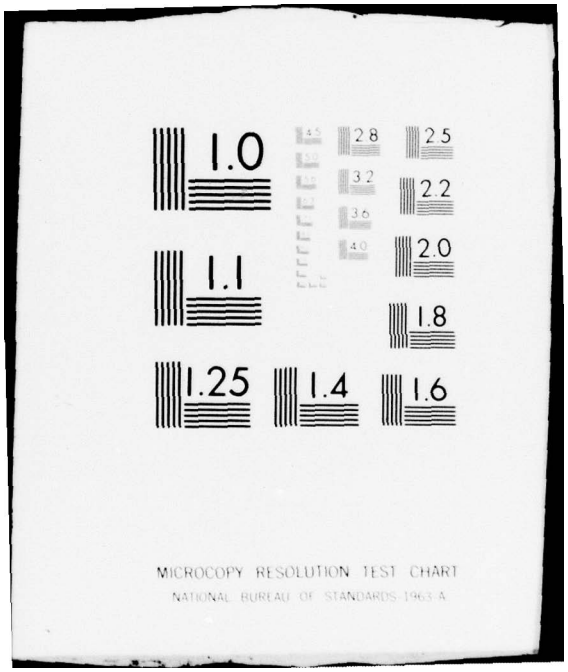
AD-A077 842 NAVAL SURFACE WEAPONS CENTER DAHLGREN LAB VA F/G 12/2
SOLUTION OF THE INTEGER CONCAVE PROGRAM USING THE ICON ALGORITH--ETC(U)
NOV 79 H W LOOMIS
UNCLASSIFIED NSWC/TR-3119 NL

1 OF 2
AD
A077 842




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

NSWC/DL TR-3119

LEVEL II

SOLUTION OF THE INTEGER CONCAVE PROGRAM USING THE ICON ALGORITHM VOLUME 1

by
HARLAN W. LOOMIS
Strategic Systems Department

NOVEMBER 1979

NAVAL SURFACE WEAPONS CENTER

Dahlgren, Virginia 22448 Silver Spring, Maryland 20910

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: NSWC/DL TR-3119	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): Solution of the Integer Concave Program Using the ICØN Algorithm, Volume 1. AD062012	5. TYPE OF REPORT & PERIOD COVERED: FINAL Rept.
6. PERFORMING ORG. REPORT NUMBER	
7. AUTHOR(s): Harlan W. Loomis	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Naval Surface Weapons Center (K30), Dahlgren, Virginia 22448	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 61152N, ZR0000 01, ZR01407, NWL/01K07
11. CONTROLLING OFFICE NAME AND ADDRESS: Naval Surface Weapons Center (K30), Dahlgren, Virginia 22448	12. REPORT DATE: 1 November 1979
	13. NUMBER OF PAGES: 145
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): 12 133	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

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

18. SUPPLEMENTARY NOTES

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

algorithm	integer programming	optimization theory
branch-and-bound	linear programming	systems analysis
computer program	mathematical programming	
concave function	nonlinear programming	
convex function	operations research	

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

The branch-and-bound technique has been the basis for algorithms to solve both the mixed integer linear program (having linear objective function, linear constraints, and integrality restrictions on some variables) and the concave nonlinear program (having a separable concave objective function and linear constraints). The subprograms for each of these branch-and-bound algorithms are linear programs with simple upper bounds.

(Continued)

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

UNCLASSIFIED

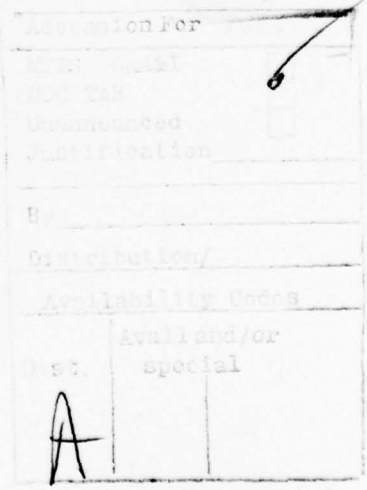
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

In Volume 1, a new branch-and-bound algorithm is presented for the composite mixed integer, concave nonlinear program. This integer concave (ICØN) algorithm has been implemented in the form of a computer program coded in FORTRAN. A guide to the use of the computer program, together with examples of its application, is included in Volume 1. Documentation of the computer program is included in Volume 2.

ABSTRACT

The branch-and-bound technique has been the basis for algorithms to solve both the mixed integer linear program (having linear objective function, linear constraints, and integrality restrictions on some variables) and the concave nonlinear program (having a separable concave objective function and linear constraints). The subprograms for each of these branch-and-bound algorithms are linear programs with simple upper bounds.

In Volume 1, a new branch-and-bound algorithm is presented for the composite mixed integer, concave nonlinear program. This integer concave (ICØN) algorithm has been implemented in the form of a computer program coded in FORTRAN. A guide to the use of the computer program, together with examples of its application, is included in Volume 1. Documentation of the computer program is included in Volume 2.



FOREWORD

Optimization problems involving integer-valued decision variables occur frequently in operations research and in systems analysis. Many cost-effectiveness analyses performed within the Department of Defense are optimization problems of this type. Specific applications related to amphibious operations occur in mine warfare, logistics, and fire support. This report presents a new method for solving a large class of integer nonlinear optimization problems.

The material presented here implements ideas developed during the author's program of studies in the Department of Operations Research, School of Engineering and Applied Science, The George Washington University, Washington, D. C. Support for this research in integer nonlinear optimization and the development of computational algorithms has been provided by the Naval Surface Weapons Center's Independent Research Program. The author is presently involved in surface warfare applications of this optimization technique.

RELEASED BY:

R. T. RYLAND, JR., Head
Strategic Systems Department

CONTENTS

CONTENTS (Continued)

LIST OF TABLES

LIST OF FIGURES

LIST OF EXHIBITS

SOLUTION OF THE INTEGER CONCAVE PROGRAM USING THE ICØN ALGORITHM

VOLUME 1

1. INTRODUCTION

The branch-and-bound method is a theoretical tool which has been used to solve various types of optimization problems and, in particular, to solve mathematical programs. An abstract optimization problem involves (i) a set of variables, (ii) a constraint set which consists of feasible combinations of the variables, and (iii) an objective function (a mathematical expression) which is to be optimized over the constraint set. In the case of a mathematical program, the set of variables is customarily finite and the constraint set is defined implicitly by a system of equations or other mathematical relationships.

The terminology "mathematical program" derives from the application of mathematics to the solution of economic problems involving the allocation ("programming") of scarce resources subject to constraints. The first such problems to be formulated were linear programs, having a linear objective function (to be minimized or maximized) and linear equations (either equalities or inequalities) defining the constraint set. The simplex algorithm of Danzig (1963) provides the solution method usually used for linear programs.

Mathematical programming now finds application in varied and diverse areas, with a significant number of these applications resulting in nonlinear programs. A nonlinear program can occur if the objective function or a constraint equation is nonlinear. An important example of a nonlinear constraint is the requirement that some variables must be integer-valued. Branch-and-bound algorithms

have been developed to solve the mixed integer linear program (having linear objective function, linear constraints and integrality restrictions) and the concave nonlinear program (having separable concave objective function and linear constraints). Each of these nonlinear programs extends the concept of a linear program.

The integer concave program is a composite of the mixed integer linear program and the concave nonlinear program. Application of the branch-and-bound method to the solution of this program was discussed in Loomis (1973b). This report presents a branch-and-bound algorithm for the integer concave program (the ICØN algorithm) and provides a guide to the use of a computer program which implements the algorithm.

Chapter 1 includes a statement of the integer concave program and other related mathematical programs. Chapter 2 describes the basic branch-and-bound method used to solve the integer concave program. A description of the options available in the ICØN algorithm is given in Chapters 3, 4 and 5. Test programs which illustrate the application of the ICØN algorithm are presented in Chapter 6. The preparation of input for the computer program is described in Appendices A, B and C using the test programs as examples.

1.1 The Integer Concave Program

Two forms of the integer concave program are considered in this section. The first of these is the mathematical program

$$(*) \qquad \begin{cases} \text{minimize} \quad f(x) = \sum_{j=1}^{n} f_j(x_j) \\ \text{such that} \quad Ax = b \\ \qquad\qquad 0 \le x \le u^o \\ \qquad\qquad x_j \text{ integer for } j \in J. \end{cases}$$

A is an $m \times n$ matrix ($m \le n$) and b is a vector in E^m with $b \ge 0$. The vector u^o in E^n represents an upper limit on x, where $u^o \ge 0$ but u^o need not be finite. The index set J is an arbitrary subset of $N = \{1, 2, \ldots, n\}$.

The objective function

$$f(x) = \sum_{j=1}^{n} f_j(x_j)$$

is assumed to be a separable concave function on the hypercube $[0, u^o]$; that is, f is the sum of functions f_j of a single variable with each function f_j being a concave function on the interval $[0, u_j^o]$. A concave function of a single variable is defined by the property that linear interpolation always underestimates the function value.[1] Further properties of concave functions can be found in Fiacco and McCormick (1968, Section 6.1). Selected examples of

---

[1] A function c(v) of a single variable v is concave on the interval I if

$$c(wv_1 + (1 - w)v_2) \ge w\, c(v_1) + (1 - w)\, c(v_2)$$

for all choices of $v_1, v_2 \in I$ and $0 \le w \le 1$.

functions f_j are shown in Figure 1.

In the statement of program (*), the equation Ax = b is synonymous with the system of m linear equality constraints

$$a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{in}x_n = b_i \quad (1 \leq i \leq m).$$

Constraints which involve inequalities ($\leq$ or $\geq$) are readily transformed into equalities by the addition of slack or surplus variables to the program. For example,

$$a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{in}x_n \leq b_i$$

becomes

$$a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{in}x_n + 1\, x_{n+1} = b_i$$

with the addition of the slack variable $x_{n+1} \geq 0$ to the program. (In the other constraints and in the objective function, x_{n+1} is added but with a coefficient of 0.)

A finite lower limit ℓ^o on x can also be included in the program formulation. Under the transformation $y = x - \ell^o$, the constraints

$$Ax = b$$
$$\ell^o \leq x \leq u^o$$

become

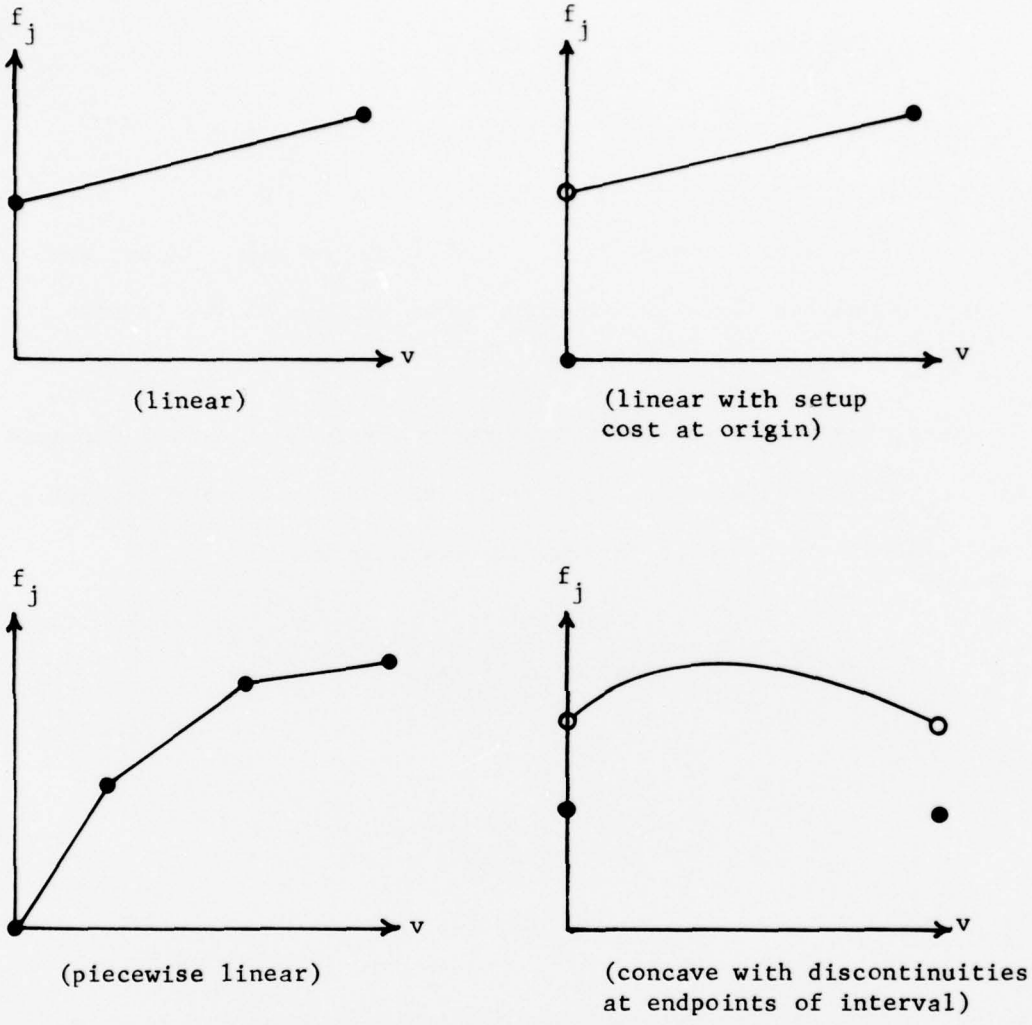


FIGURE 1

EXAMPLES OF CONCAVE FUNCTIONS
OF A SINGLE VARIABLE

$$Ay = b - A\ell^o$$

$$0 \leq y \leq u^o - \ell^o$$

which has the form of program (*). The conditions $b \geq 0$ and $u^o \geq 0$ in program (*) are replaced by conditions $b - A\ell^o \geq 0$ and $u^o \geq \ell^o$ when the lower limit constraint is added to the program. Also, each component function f_j is now required to be concave on the interval $[\ell_j^o, u_j^o]$.

The computer program which implements the ICØN algorithm performs the two transformations just mentioned, and thus solves the following (more general) form of the integer concave program:

$$\text{(ICP)} \quad \begin{cases} \text{minimize} & f(x) = \sum_{j=1}^{n} f_j(x_j) \\ \text{such that} & Ax \begin{bmatrix} \leq \\ = \\ \geq \end{bmatrix} b \\ & \ell^o \leq x \leq u^o \\ & x_j \text{ integer for } j \in J. \end{cases}$$

In the balance of this report, the notation $Ax = b$ will be used in place of

$$Ax \begin{bmatrix} \leq \\ = \\ \geq \end{bmatrix} b$$

to simplify notation.

Various additional transformations and techniques are frequently

useful for reducing problems encountered in practice to the form of program (ICP). Some of these may be found in Wagner (1969, Chapter 3) and in Garfinkel and Nemhauser (1972, Section 1.4).

1.2 Other Mathematical Programs

The integer concave program includes as special cases several other mathematical programs which occur commonly in applications. These may also be solved using the ICØN algorithm.

The mixed integer linear program

$$(\text{MILP}) \quad \begin{cases} \text{minimize} & f(x) = \sum_{j=1}^{n} c_j x_j \\ \text{such that} & Ax = b \\ & \ell^o \leq x \leq u^o \\ & x_j \text{ integer for } j \in J. \end{cases}$$

is a special case of the integer concave program which has a linear objective function.

A branch-and-bound algorithm for solving the mixed integer linear program was first developed by Land and Doig (1960). Subsequent modifications and refinements are reflected in the algorithms of Dakin (1965); Davis, Kendrick and Weitzman (1971); and Tomlin (1971).

Deleting the integrality requirements from the integer concave program results in the concave nonlinear program

$$
(CP)\quad \begin{cases} \text{minimize} & f(x) = \sum_{j=1}^{n} f_j(x_j) \\ \text{such that} & Ax = b \\ & \ell^o \le x \le u^o. \end{cases}
$$

A branch-and-bound algorithm for solving this program and other more general separable nonconvex programs was developed by Falk and Soland (1969).

A final special case of the integer concave program is the linear program

$$
(LP)\quad \begin{cases} \text{minimize} & f(x) = \sum_{j=1}^{n} c_j x_j \\ \text{such that} & Ax = b \\ & \ell^o \le x \le u^o. \end{cases}
$$

The solution of linear programs having bounded variables utilizes a modification of Dantzig's simplex algorithm which is discussed in Lasdon (1970, Section 6.3) and in Garfinkel and Nemhauser (1972, Section 2.8).

2. THE BRANCH-AND-BOUND METHOD

The branch-and-bound method for the integer concave program is based on the transformation of a single program into a sequence of subprograms. An essential part of the method is a dynamic scheme which monitors the selection of the subprograms to be examined. Associated with each stage of the branch-and-bound method are lower and upper bounds on the optimal objective function value which serve as the natural basis for a convergence criterion.

Section 2.1 presents the basic branch-and-bound method (method 1) for the integer concave program. Method 1 includes as options the selection of particular computational procedures and rules which, when they are specified, convert an abstract method into a workable algorithm. A bounding procedure which can be applied to the individual subprograms is presented in Section 2.2. The bounding of a subprogram is based upon the solution of a related linear program. Section 2.3 describes a constraint set partitioning procedure. Convergence properties of the branch-and-bound method are presented in Section 2.4. The final section gives an interpretation of the method from the standpoint of graph theory.

2.1 Method 1

Corresponding to the statement of program (ICP) given in Section 1.1, let

$$Q = \{x \mid Ax = b\}$$

$$H^o = \{x \mid \ell^o \leq x \leq u^o\}$$

$$X = \{x \mid x_j \text{ integer for } j \epsilon J\}.$$

Q is a linear polyhedron and H^o is a hypercube in E^n. In terms of this notation, an equivalent formulation for the integer concave program is

$$(M) \qquad \underset{x \epsilon M}{\text{minimize}} \quad f(x)$$

where $M = Q \cap H^o \cap X$. This is called the master program for the branch-and-bound method. M is the set of feasible solutions to the master program. When the optimal objective function value for the master program is finite, it will be denoted by f*. In addition, if this value is attained at some point in M, then the set of optimal solutions

$$M^* = \{x \epsilon M \mid f(x) = f^*\}$$

is nonempty. The branch-and-bound method seeks to determine the optimal solution value f* and an optimal solution $x^* \epsilon M^*$ whenever these exist. Additionally, in the event that M* is empty or f* does not exist, the method discovers this fact and terminates computations accordingly.

The branch-and-bound method proceeds in a series of stages $t = 1, 2, \ldots$. Each stage involves the examination of one or two subprograms. A typical subprogram i ($i = 1, 2, \ldots$) is an integer concave program of the form

$$(M^i) \qquad \underset{x \in M^i}{\text{minimize}} \qquad f(x)$$

where $M^i = Q \cap H^i \cap X$ and $H^i = \{x \mid \ell^i \leq x \leq u^i\}$ is a subset of H^o. The examination of subprogram i consists of applying a bounding procedure which yields a lower bound LB(i) on the optimal objective function value for subprogram i and an upper bound UB(i) on the optimal objective function value to the master program. Two alternative bounding procedures are presented in Sections 2.2 and 3.1.

Stage t = 1 consists of only one subprogram (subprogram 1) which serves to initiate the branch-and-bound method. Subprogram 1 has the same constraint set as the master program, $M^1 = Q \cap H^1 \cap X$ where $H^1 = H^o$ (i.e., $\ell^1 = \ell^o$ and $u^1 = u^o$). The bounding procedure is applied to obtain lower bound LB(1) on subprogram 1 and upper bound UB(1) on the master program. The best upper bound at stage 1 is defined by

$$BUB(1) = UB(1).$$

Let $N(1) = \{1\}$ and

$$BLB(1) = \min_{i \in N(1)} LB(i) = LB(1).$$

N(1) is the set of active subprograms at stage 1 and the quantity BLB(1) is the best lower bound at stage 1.

For $t \geq 2$, stage t of the branch-and-bound method will now be described. Assume that the following data (resulting from stage t-1) are given:

(i) $N(t-1)$, a nonempty subset of $\{1, 2, \ldots, 2t-3\}$;

(ii) $H^i = \{x \mid \ell^i \leq x \leq u^i\}$ and $LB(i)$ for $i \in N(t-1)$; and

(iii) $BUB(t-1)$.

A branching subprogram $k \in N(t-1)$ is selected using one of the rules presented in Section 4.1. Subprograms s and s+1 (where $s = 2t - 2$) are considered at this stage. The constraint set for subprogram k is partitioned to form the constraint sets for subprograms s and s+1. This is accomplished by splitting the hypercube H^k into hypercubes H^s and H^{s+1}. Constraint set partitioning is discussed further in Section 2.3.

Subprograms s and s+1 have constraint sets $M^s = Q \cap H^s \cap X$ and $M^{s+1} = Q \cap H^{s+1} \cap X$. The bounding procedure is applied to obtain lower bound $LB(s)$ on subprogram s and upper bound $UB(s)$ on the master program. $LB(s+1)$ and $UB(s+1)$ are determined similarly.

The best upper bound at stage t is the quantity

$$BUB(t) = \min \{BUB(t-1), UB(s), UB(s+1)\}.$$

The set of active subprograms $N(t)$ at stage t is now formed. First, define

$$I(t) = N(t-1) \setminus \{k\} \cup \{s, s+1\}$$

and then

$$N(t) = \{i \in I(t) \mid LB(i) < BUB(t)\}.$$

If N(t) is an empty set, then the branch-and-bound method terminates. Otherwise, the best lower bound at stage t is defined to be the quantity

$$BLB(t) = \min_{i \in N(t)} LB(i)$$

and the branch-and-bound method proceeds to the next stage.

2.2 Bounding

In the branch-and-bound method presented in Section 2.1, a bounding procedure is applied to each subprogram i (i = 1, 2, ...). The bounding procedure determines a lower bound LB(i) on the optimal objective function value for subprogram i and an upper bound UB(i) on the optimal objective function value for the master program. This section presents the first of two alternative bounding procedures.

The typical subprogram is an integer concave program of the form

$$(M^i) \qquad \underset{x \in M^i}{\text{minimize}} \qquad f(x) = \sum_{j=1}^{n} f_j(x_j)$$

where $M^i = Q \cap H^i \cap X$. The bounding procedure begins with the formulation of an auxiliary linear program

$$(R^i) \qquad \underset{x \in R^i}{\text{minimize}} \qquad g^i(x) = \sum_{j=1}^{n} g^i_j(x_j)$$

where $R^i = Q \cap H^i$ and, for each index j, g_j^i is the unique quasi-linear function[2] which agrees with f_j at the endpoints of the interval $[\ell_j^i, u_j^i]$. The relationship between g_j^i and f_j is depicted in Figure 2. In particular, g^i underestimates f on M^i.

The simplex procedure is applied to the linear program (R^i). If it is feasible, then let x^i denote a solution as determined by the simplex procedure. A lower bound on the optimal objective function value for subprogram i is given by

$$LB(i) = g^i(x^i).$$

An upper bound on the optimal objective function value for the master program is given by

$$UB(i) = \begin{cases} f(x^i), & \text{if } x^i \in M^i \\ \infty, & \text{if otherwise.} \end{cases}$$

If linear program (R^i) is infeasible, then so is subprogram i. In this case, define

$$LB(i) = UB(i) = \infty.$$

---

[2] A function q(v) of a single variable v is quasi-linear if there are constants m and b such that $q(v) = mv + b$.

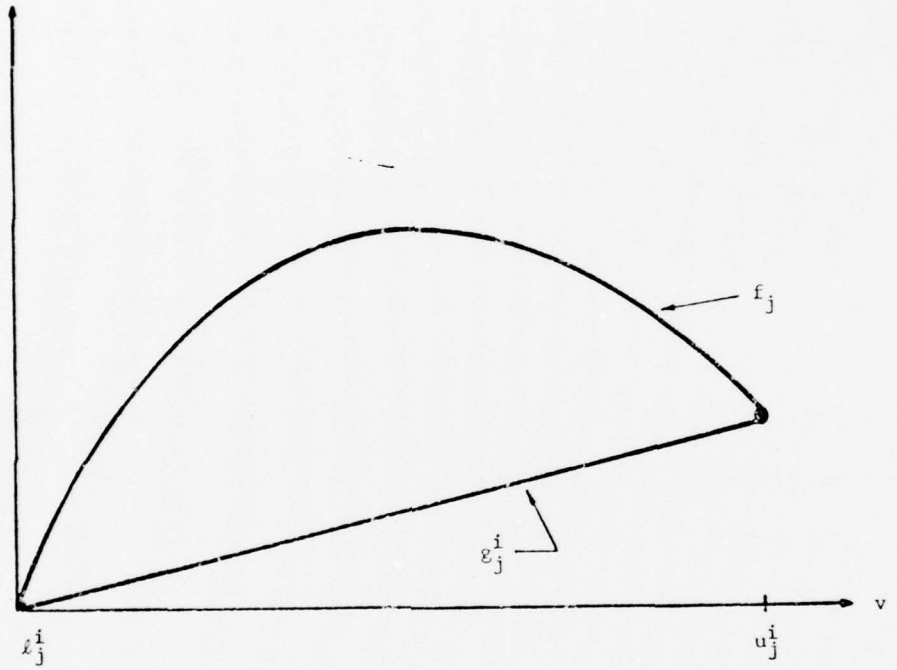


FIGURE 2

QUASI-LINEAR UNDERESTIMATE
OF A CONCAVE FUNCTION

For a mixed integer linear program, the bounding procedure just described agrees with the procedure used by Dakin (1965) and others. Subprogram i and the linear program (R^i) coincide in this case. For a concave nonlinear program, this bounding procedure agrees with that of Falk and Soland (1969). The condition $x^i \in M^i$ always holds for this case and each feasible subprogram yields a finite upper bound.

2.3 Constraint Set Partitioning

Essential to the generation of subprograms is the constraint set partitioning procedure. Upon the selection of a branching subprogram k, this procedure is applied to select constraint sets for subprograms s and s+1. To apply the constraint set partitioning procedure, the hypercube $H^k = \{x \mid \ell^k \leq x \leq u^k\}$ and the solution x^k to linear program (R^k) are required.

The partition begins with the selection of a branching variable x_{j_o} $(1 \leq j_o \leq n)$ using one of the rules presented in Sections 4.2 or 4.3. If the branching variable is an integer variable ($j_o \in J$), let

$$L = \{x \mid x_{j_o} \leq [x^k_{j_o}]\}$$

$$U = \{x \mid x_{j_o} \geq \langle x^k_{j_o} \rangle\}.$$[3]

[3] $[v]$ denotes the largest integer less than or equal to v, while $\langle v \rangle$ denotes the smallest integer greater than or equal to v.

If the branching variable is not an integer variable ($j_o \notin J$), let

$$L = \{x \mid x_{j_o} \leq x^k_{j_o}\}$$

$$U = \{x \mid x_{j_o} \geq x^k_{j_o}\}.$$

L and U are the lower and upper cuts corresponding to value $x^k_{j_o}$ of the branching variable.

Hypercubes H^s and H^{s+1} are formed by setting

$$H^s = H^k \cap L$$

$$H^{s+1} = H^k \cap U.$$

The lower and upper limits appearing in $H^s = \{x \mid \ell^s \leq x \leq u^s\}$ and $H^{s+1} = \{x \mid \ell^{s+1} \leq x \leq u^{s+1}\}$ thus agree with those of H^k with the exception of the limits on the branching variable.

This procedure for constraint set partitioning was introduced by Dakin (1965) for mixed integer linear programs and was adopted by Falk and Soland (1969) for concave nonlinear programs.

2.4 Convergence

Assume that the set M* of optimal solutions to the integer concave program is nonempty. In this case,

$$LB(1) \leq \min_{x \in M^1} f(x) = \min_{x \in M} f(x) = f^*$$

since subprogram 1 coincides with the master program. As a consequence of the definitions BLB(1) = LB(1) and BUB(1) = UB(1),

$$BLB(1) \leq f^* \leq BUB(1).$$

The convergence properties of the branch-and-bound method are based on the fact that, at <u>any</u> stage t,

$$BLB(t) \leq f^* \leq BUB(t).$$

Because the sequence of best lower bounds is nondecreasing and the sequence of best upper bounds is nonincreasing, the precision with which f* is determined improves as the method progresses from stage to stage. These facts about the branch-and-bound method are summarized in the following theorem.

<u>Theorem 1</u>. If $t \geq 2$, N(t-1) and N(t) are nonempty, then

$$BLB(t-1) \leq BLB(t) \leq f^* \leq BUB(t) \leq BUB(t-1).$$

The convergence of the method depends, first, on the ability of the method to establish a finite best upper bound and, second, to narrow the gap between the best lower bound and the best upper bound. The branch-and-bound method can therefore be regarded as having two phases. Phase 1 seeks to determine a feasible solution for the master program, thus establishing a finite best upper bound. Phase 2 seeks to verify the optimality of this feasible solution. In the course of this verification, the optimality of the feasible solution could be denied through the determination of another feasible solution which yields an improved best upper bound. The second phase thus seeks to verify the optimality of the current best feasible solution.

The branch-and-bound method is said to be finitely convergent if, for any integer concave program, the method terminates in a finite number of stages. The following theorem describes the conditions which occur at termination.

Theorem 2. If $t \geq 2$, $N(t-1)$ is nonempty and $N(t)$ is empty, then $BLB(t-1) < BUB(t-1)$ and $BUB(t) = f^*$. In this case, if i is an index for which $UB(i) = BUB(t)$ and x^i is the corresponding feasible solution determined by applying the bounding procedure to subprogram i, then x^i is an optimal solution.

At termination of the method, an optimal solution and the optimal solution value have both been computed. Finite convergence is not guaranteed in general. However, if the method employs a branching subprogram selection rule from Section 4.1 and a branching variable selection rule from Sections 4.2 or 4.3, then the following result holds.

Theorem 3. If $H^o = \{x \mid \ell^o \leq x \leq u^o\}$ is a compact set, then the branch-and-bound method for the integer concave program converges finitely.

Finite convergence for the concave nonlinear program is proved in Falk and Soland (1969). Their result is required for the proof of Theorem 3.

The results thus far have been made under the assumption that the set M^* of optimal solutions is nonempty. For the case that M^* is

empty, the branch-and-bound method depends exclusively on lower bounds for its convergence properties. Under the hypotheses of Theorem 3, finite termination also occurs for this case.

2.5 The Branch-and-Bound Tree

The branch-and-bound method has a structure which admits of an interpretation from the standpoint of graph theory. For this reason, graph theoretic terminology occurs often in the lexicon of branch-and-bound. The terminology used here is natural and self-explanatory. Definitions and other related aspects of graph theory may be found in Berge (1962).

Imbedded within the branch-and-bound method is a graph called the branch-and-bound tree. The indices $i = 1, 2, \ldots$ associated with the subprograms are regarded as the nodes of this graph. Stage $t = 1$ of the method begins with subprogram 1 which corresponds to the root node (node 1) of the branch-and-bound tree. At subsequent stages $t \geq 2$ of the method, a branching subprogram k is selected and two new subprograms are created, subprograms s and $s + 1$ (where $s = 2t - 2$). In the branch-and-bound tree, this corresponds to the creation of nodes s and $s + 1$ which are connected to node k by arcs. These arcs are denoted by (k,s) and $(k,s+1)$, respectively.

A path is a sequence of arcs

$$a_j = (i_j, i_{j+1}) \qquad j = 1, 2, \ldots$$

which connect nodes $i_1, i_2, \ldots$ in the tree. Along any path, the relationships

$$M^{i_1} \supset M^{i_2} \supset \ldots$$

$$R^{i_1} \supset R^{i_2} \supset \ldots$$

hold for the constraint sets of the corresponding subprograms and the auxiliary linear programs of the bounding procedure. Thus the respective lower bounds satisfy

$$LB(i_1) \leq LB(i_2) \leq \ldots$$

for either of the lower bounding procedures described in Sections 2.2 and 3.1.

A hypothetical example of a branch-and-bound tree is shown in Figure 3. Indicated in the figure are the upper and lower bounds UB(i)/LB(i) associated with each subprogram i. Table 1 gives the best upper bound BUB(t), the best lower bound BLB(t), and the set N(t) of active subprograms, for each stage t. In this example, phase 1 of the computation consists of stages 1, 2 and 3. Phase 2 begins at stage 4 with the discovery of a feasible solution to the master program at subprogram 6 ($x^6 \in M^6$). With the termination of the computation at stage 6, this feasible solution is found to be optimal.

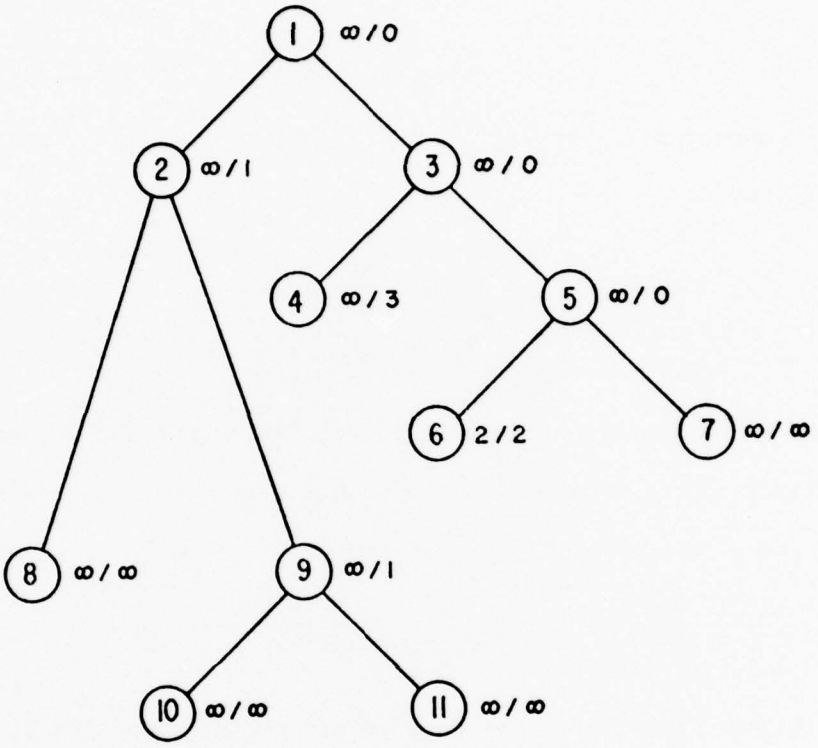


FIGURE 3

THE BRANCH-AND-BOUND TREE

TABLE 1

STATISTICS FOR THE BRANCH-AND-BOUND TREE

t	BUB(t)	BLB(t)	N(t)
1	∞	0	{1}
2	∞	0	{2, 3}
3	∞	0	{2, 4, 5}
4	2	1	{2}
5	2	1	{9}
6	2	--	{ }

The evolution of the branch-and-bound tree is described through a series of subgraphs G(t) for each stage t = 1, 2, ... The subgraph G(t) consists of nodes 1, 2, ... , 2t - 1. The set N(t) of active subprograms at stage t corresponds to a subset of the pendant nodes in this subgraph. Based on the preceeding example of a branch-and-bound tree, Figure 4 shows the subgraphs G(1), ... , G(5) associated with the first five stages and the corresponding sets N(1), ... , N(5) of active subprograms. Note that G(6) is the same as the entire tree shown in Figure 3 and that N(6) is empty.

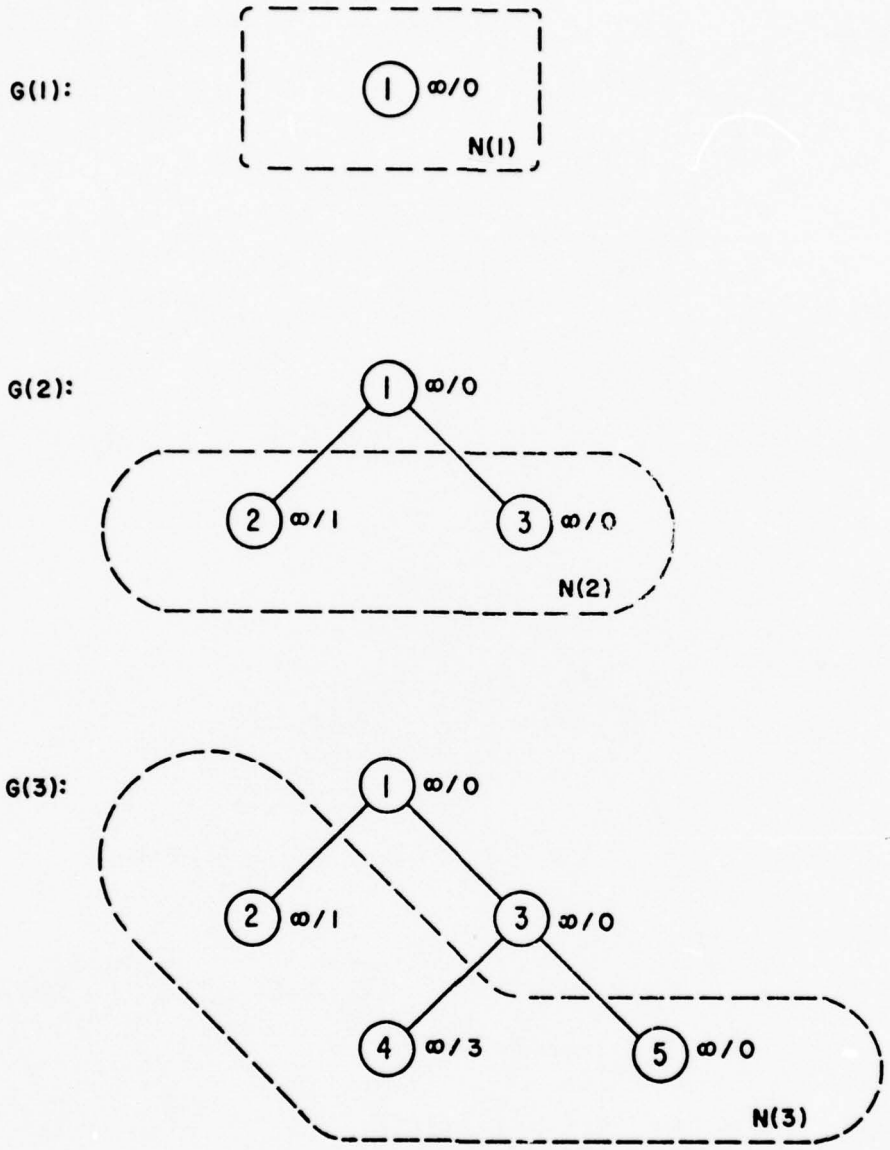


FIGURE 4

SUBGRAPHS ASSOCIATED WITH EACH
STAGE OF THE METHOD

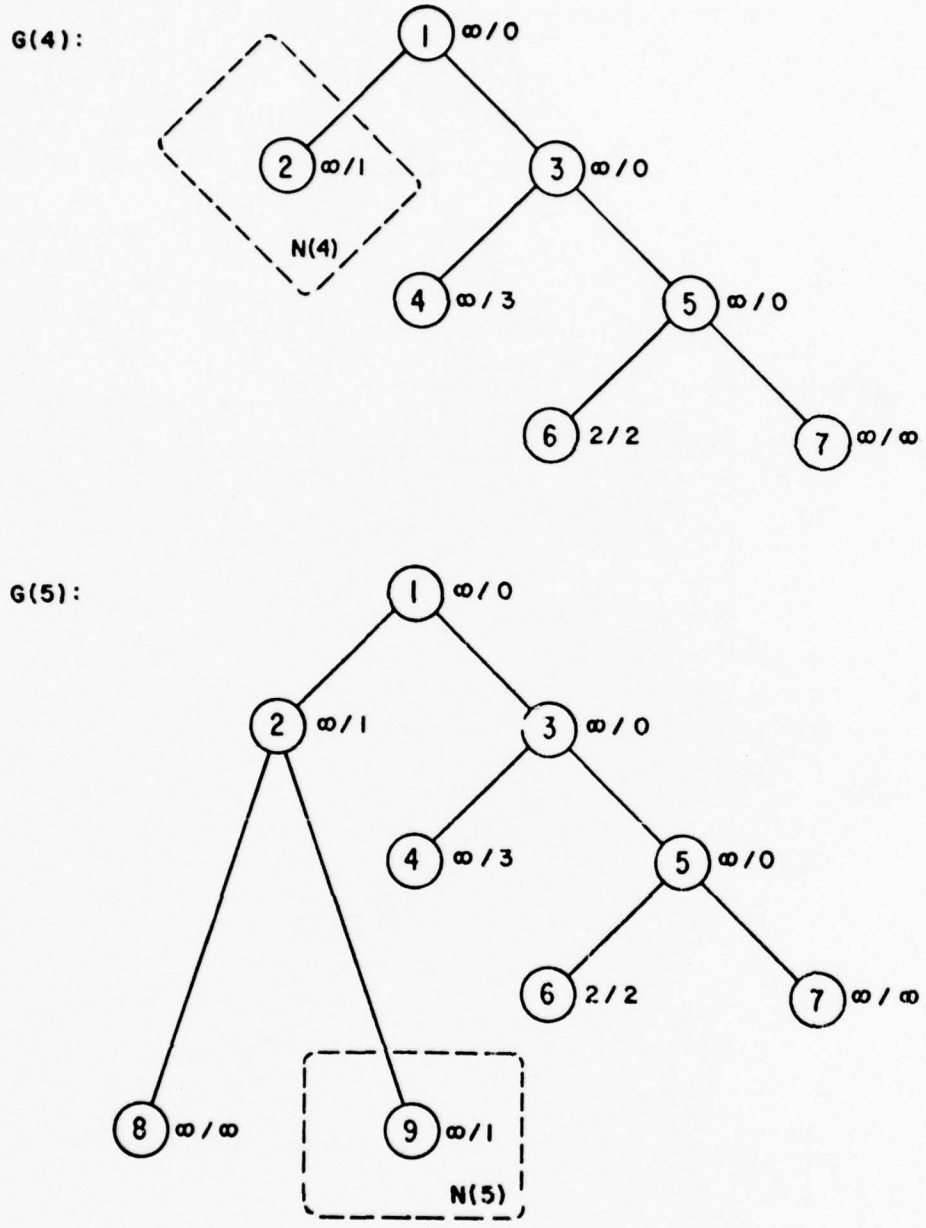


FIGURE 4 (Continued)

3. PENALTY TECHNIQUES

An alternate bounding procedure for the integer concave program uses a system of "penalties" from which stronger lower bounds for individual subprograms may be computed. Penalty techniques have been applied in branch-and-bound algorithms for the mixed integer linear program, but their application to the integer concave program is new.

Application of the constraint set partitioning procedure together with linear programming sensitivity analysis to derive penalties and stronger lower bounds is described in Section 3.1. Constraint set tightening, a second application of penalty techniques, is described in Section 3.2.

3.1 Bounding

The second bounding procedure for a typical subprogram i begins with the solution of auxiliary linear program (R^i) as in Section 2.2. The optimal solution value to the linear program is then tested for its sensitivity to small deviations of individual variables x_j from the values x_j^i. This results in the derivation of a lower bound for subprogram i which is stronger than that produced by the first bounding procedure.

Selecting a variable x_j ($1 \leq j \leq n$), the affect of adjoining the constraint $x_j = v$ to the linear program is considered. From sensitivity analysis techniques of linear programming, the optimal solution value for linear program (R^i) with this constraint adjoined (regarded

as a function of the parameter v) is convex and piecewise linear. An example of this function is denoted by $G^i(v)$ in Figure 5. The left and right derivatives of $G^i(v)$ at $v = x_j^i$ are denoted by $m_0(j)$ and $m_1(j)$, and are obtainable from the final tableau for linear program (R^i). These derivatives provide a means for constructing a simple function, consisting of the two dashed linear segments shown in Figure 5, which underestimates $G^i(v)$ over all values v.

Consider now the hypothetical case that, at some future stage of the branch-and-bound method, subprogram i is selected as the branching subprogram and variable x_j is selected as the branching variable. Figures 6 and 7 apply to the cases of x_j an integer or noninteger variable (respectively). The quasi-linear underestimate for f_j is modified as indicated for the linear programs (R^s) and (R^{s+1}). The optimal solution value functions $G^s(v)$ and $G^{s+1}(v)$ are related to $G^i(v)$ by

$$G^s(v) = G^i(v) + g_j^s(v) - g_j^i(v)$$

$$G^{s+1}(v) = G^i(v) + g_j^{s+1}(v) - g_j^i(v).$$

Corresponding relationships hold between simple linear underestimates for $G^s(v)$ and $G^{s+1}(v)$ (having slopes $\bar{m}_0(j)$ and $\bar{m}_1(j)$) and the underestimate for $G^i(v)$.

A lower bound for subprogram s is the minimum value $p_0(j)$ along the linear segment having slope $\bar{m}_0(j)$. This is simply the minimum of the two endpoints of the line segment. A lower bound $p_1(j)$ for

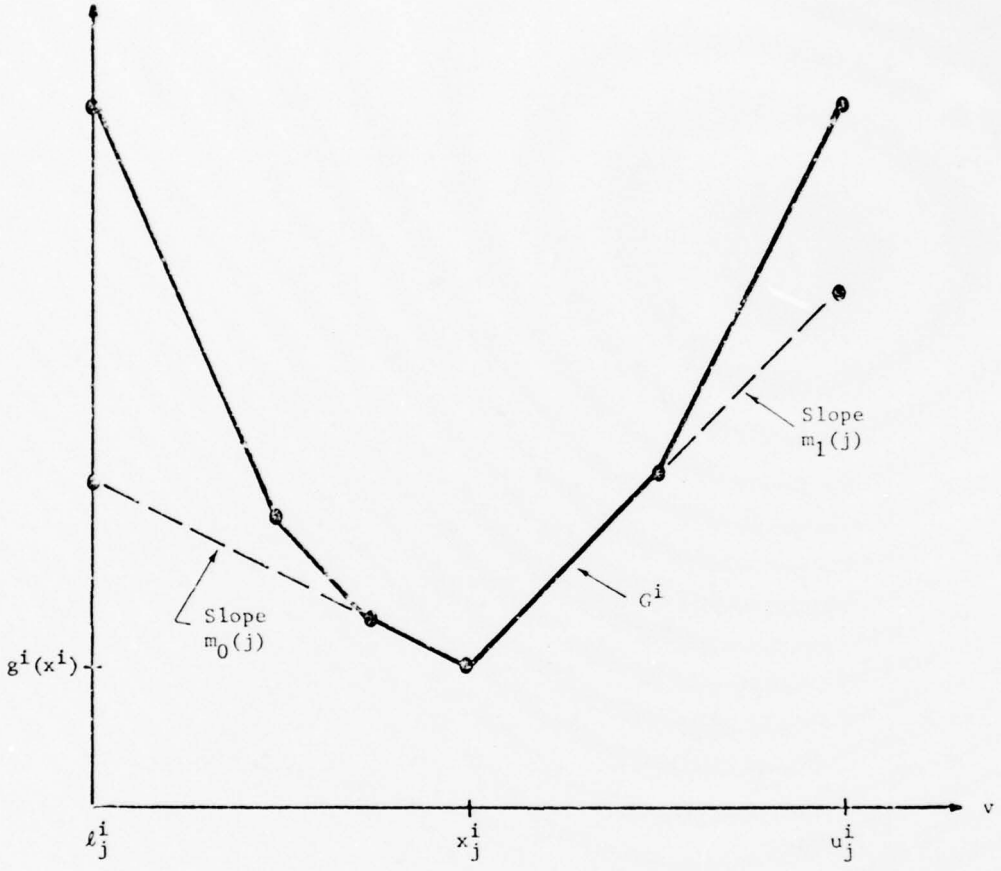


FIGURE 5

LINEAR PROGRAMMING SENSITIVITY ANALYSIS

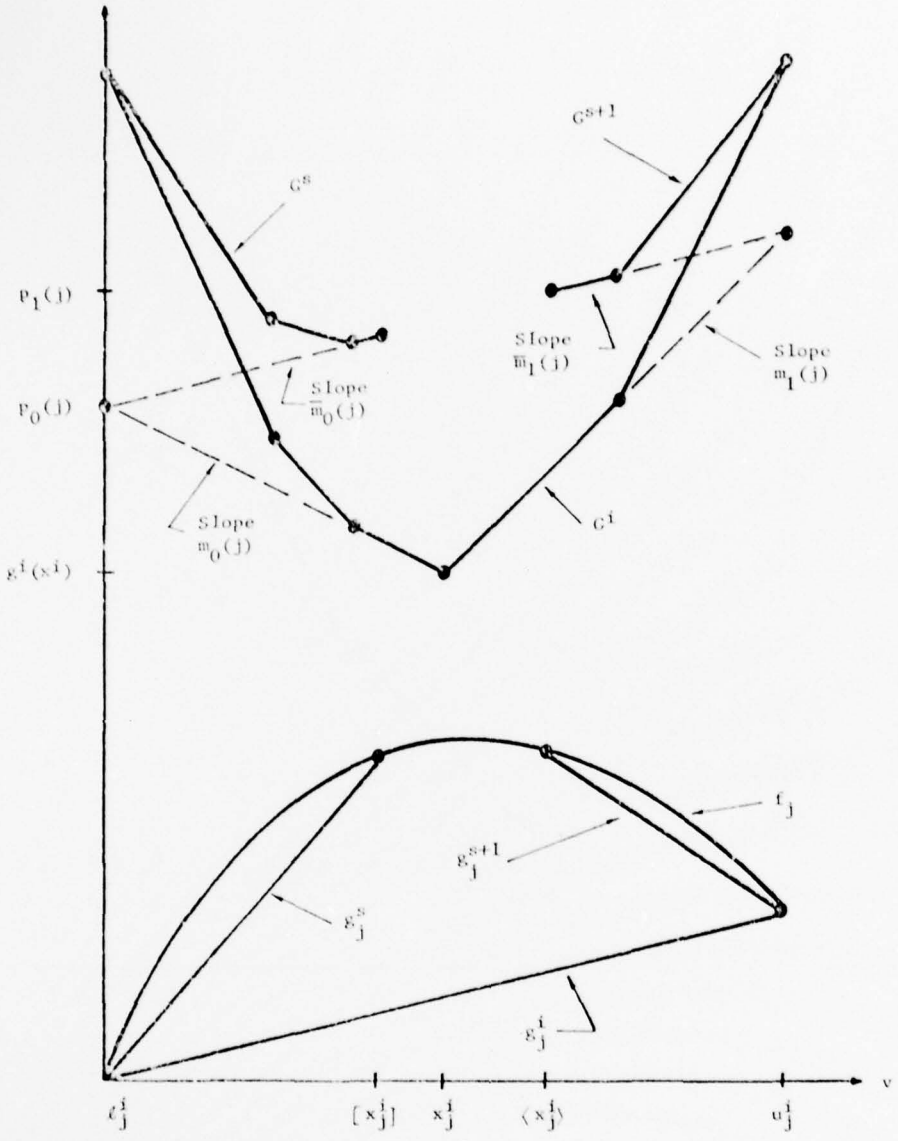


FIGURE 6

DETERMINATION OF PENALTIES FOR AN INTEGER VARIABLE

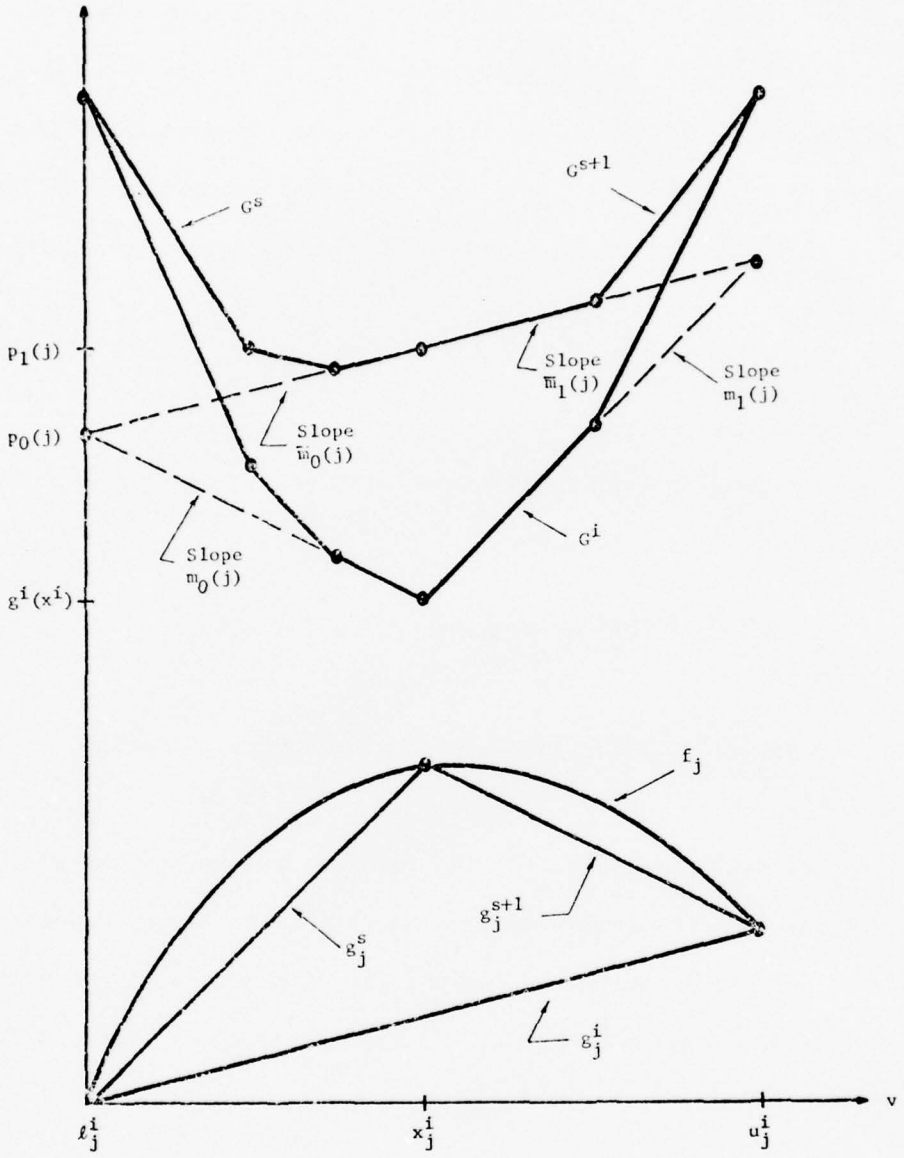


FIGURE 7

DETERMINATION OF PENALTIES FOR A
NONINTEGER VARIABLE

subprogram s+1 is defined similarly. In comparison to the lower bound produced by the first bounding procedure, the quantities $p_0(j)$ and $p_1(j)$ reflect the penalties which would occur if variable x_j were selected as the branching variable. Since the constraint sets for subprograms s and s+1 contain the set for subprogram i, the quantity

$$\min \{p_0(j), p_1(j)\}$$

is a lower bound for subprogram i.

Considering all possible choices of j ($1 \leq j \leq n$), the penalty analysis leads to the definition

$$LB(i) = \max_{1 \leq j \leq n} \min \{p_0(j), p_1(j)\}.$$

The upper bound UB(i) for this bounding procedure is defined as in Section 2.2.

The bounding procedure for the integer concave program presented in this section agrees with the procedure used by Davis, Kendrick and Weitzman (1971) for the mixed integer linear program. Application of penalty techniques to the integer concave program was first indicated in Loomis (1973b).

3.2 Limit Tightening

The algorithm for the concave nonlinear program developed by Falk and Soland uses the slopes $m_0(j)$ and $m_1(j)$ to tighten the lower and upper limits ℓ_j^i and u_j^i on <u>each</u> variable x_j. In the algorithm for the

integer concave program presented here, the slopes $\overline{m}_0(j)$ and $\overline{m}_1(j)$ can be used for the same purpose.

Examples of limit tightening for the integer concave program are shown in Figure 8 (for an integer variable) and in Figure 9 (for a noninteger variable). The procedure involves the computation of the intercept of the line having slope $\overline{m}_0(j)$ (or $\overline{m}_1(j)$) with the horizontal line corresponding to the current best upper bound BUB(t). In the example of Figure 8, the upper bound u_j^i is replaced by $(u_j^i)'$. Such limit tightening is justified by the fact that no point y of the set $M^i \cap \{x \mid x_j \geq (u_j^i)'\}$ can provide an improvement on the current best upper bound since

$$f^* \leq \text{BUB}(t) \leq g^i(y) \leq f(y).$$

It should be noted that limit tightening can be effective only during phase 2 of the branch-and-bound method when BUB(t) is finite. Limit tightening is most naturally employed at the same time as the bounding procedure described in Section 3.1.

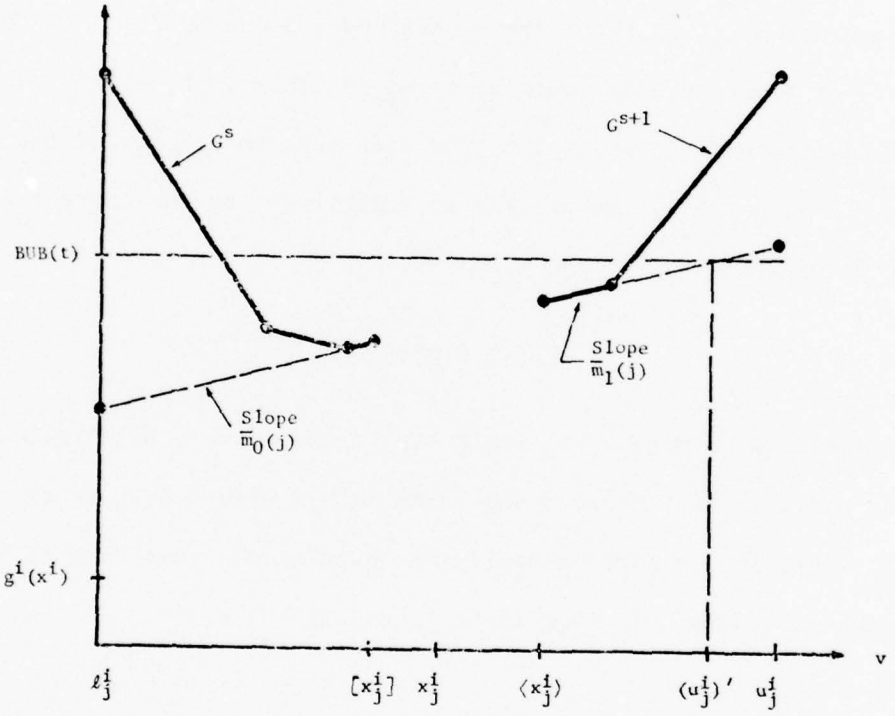


FIGURE 8

TIGHTENING OF LIMITS FOR AN
INTEGER VARIABLE

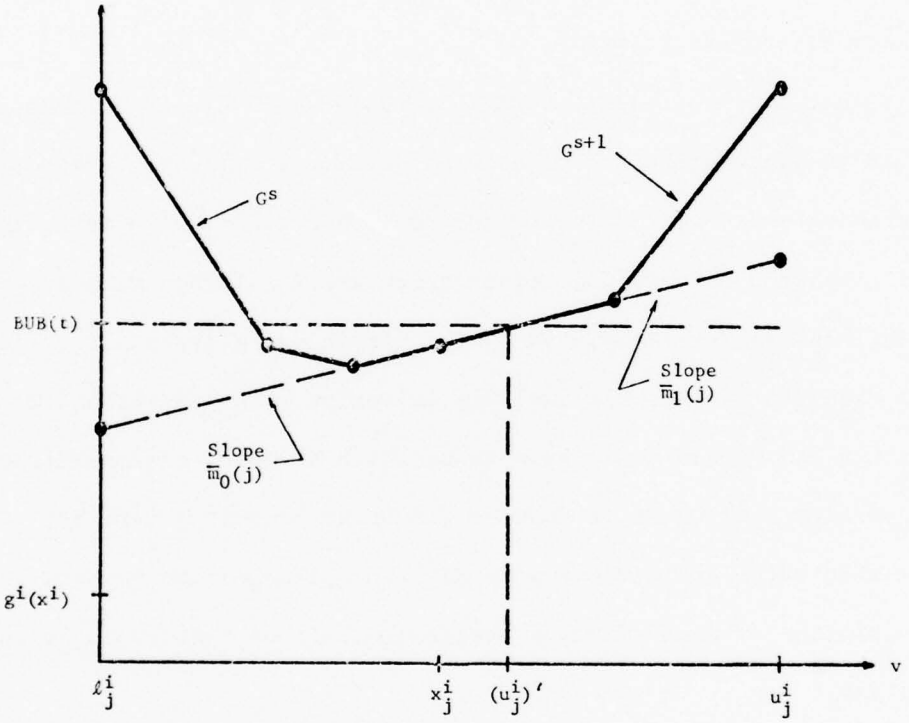


FIGURE 9

TIGHTENING OF LIMITS FOR A
NONINTEGER VARIABLE

4. BRANCHING SUBPROGRAM AND BRANCHING VARIABLE SELECTION

A branching subprogram selection rule is required in the branch-and-bound method of Section 2.1. This rule serves to determine the order in which the active subprograms are processed. Section 4.1 presents the characteristics of two branching subprogram selection rules, the LIFO and priority rules, which may be used in the solution of integer concave programs.

Constraint set partitioning, discussed in Sections 2.1 and 2.3, serves to dichotomize the constraint set for a branching subprogram, generating constraint sets for two new subprograms while excluding from further consideration points which are irrelevant with respect to the master program solution. The partitioning procedure utilized here requires a branching variable selection rule. Section 4.2 presents the maxmin and maxmax rules, both of which are based upon the penalty techniques of Chapter 3. Other branching variable selection rules are discussed in Section 4.3 which may be used in the solution of mixed integer linear programs or concave nonlinear programs.

4.1 LIFO and Priority Rules

Two alternative rules which may be applied to select a branching subprogram from among the active subprograms are the LIFO (last in/first out) and priority rules. Application of the LIFO rule results in the selection of the subprogram which was the last one added to

the set of active subprograms. The priority rule results in the selection of an active subprogram for which the corresponding lower bound is the least. A family of branching subprogram selection rules which span the gap between the LIFO and priority rules is discussed in Section 5.4.

Figure 10 gives a typical example of the branch-and-bound tree structure which results from the LIFO rule. Each node in this tree corresponds to a subprogram in the branch-and-bound method. Indicated in the figure are the upper and lower bounds UB(i)/LB(i) associated with the i-th subprogram. Table 2 gives the corresponding statistical data: the best upper bound BUB(t), the best lower bound BLB(t), and the set N(t) of active subprograms for each stage t. Figure 11 shows the same example solved using the priority rule, with Table 3 giving the corresponding statistical data. In this example, the branch-and-bound computation resulting from the LIFO rule examines some subprograms which have no counterpart in the computation resulting from the priority rule. Otherwise, the two computations differ only in the sequence in which the subprograms are examined.

The subprograms which are examined initially using the LIFO rule correspond to the nodes on a single path of the branch-and-bound tree called the principal path. This path terminates when a branching subprogram gives rise to a pair of subprograms having lower bounds greater than or equal to the current best upper bound. The LIFO rule then selects the active subprogram for which the corresponding node is

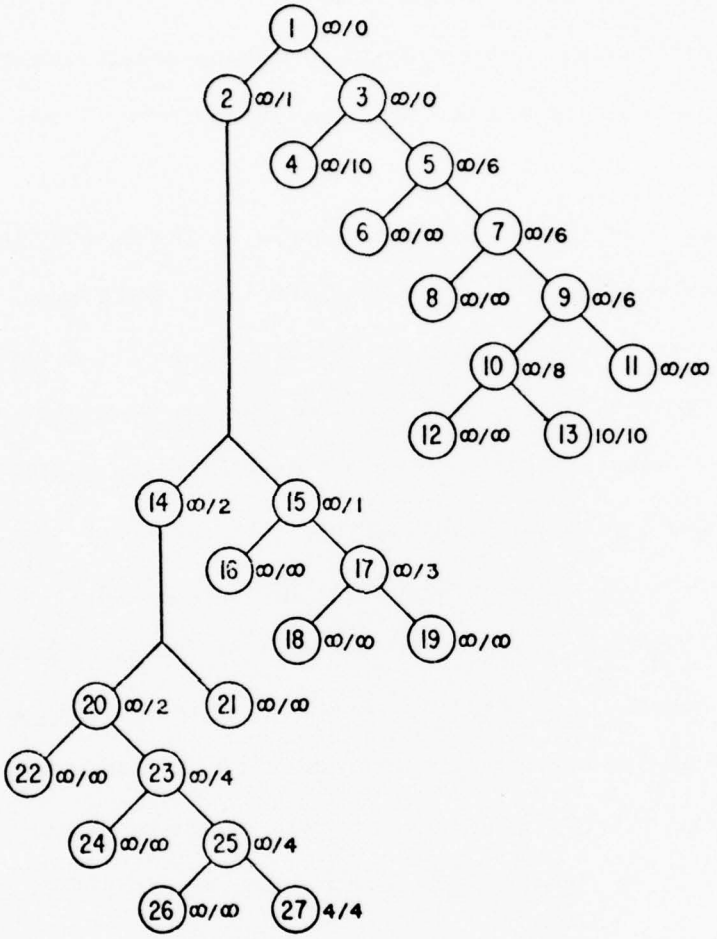


FIGURE 10

TREE STRUCTURE FOR THE LIFO RULE

TABLE 2

STATISTICS FOR THE TREE WITH THE LIFO RULE

t	BUB(t)	BLB(t)	N(t)
1	∞	0	{1}
2	∞	0	{2, 3}
3	∞	1	{2, 4, 5}
4	∞	1	{2, 4, 7}
5	∞	1	{2, 4, 9}
6	∞	1	{2, 4, 10}
7	10	1	{2}
8	10	1	{14, 15}
9	10	2	{14, 17}
10	10	2	{14}
11	10	2	{20}
12	10	4	{23}
13	10	4	{25}
14	4	--	{ }

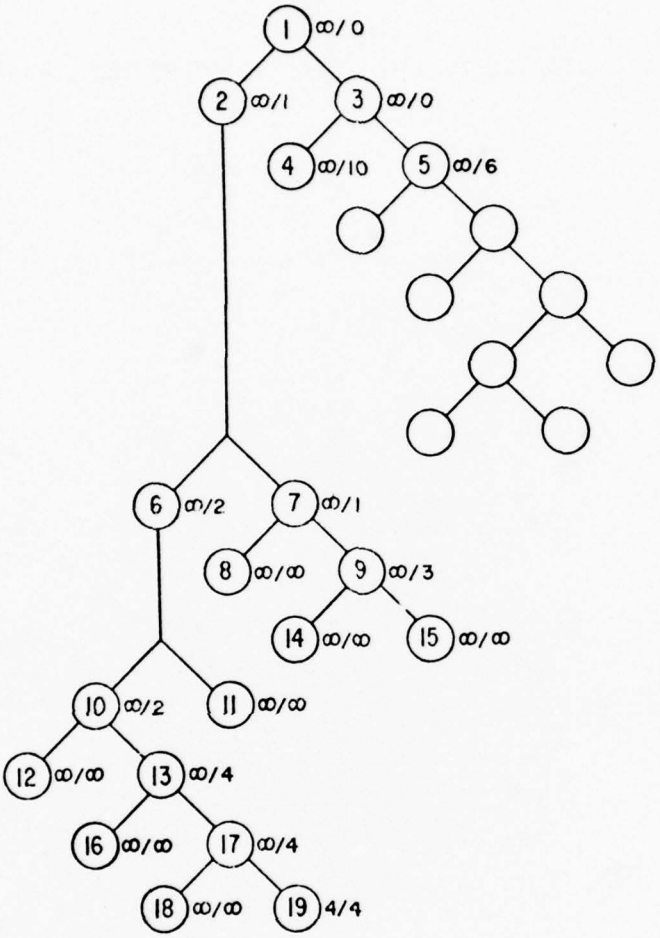


FIGURE 11

TREE STRUCTURE FOR THE PRIORITY RULE

TABLE 3

STATISTICS FOR THE TREE WITH THE PRIORITY RULE

t	BUB(t)	BLB(t)	N(t)
1	∞	0	{1}
2	∞	0	{2, 3}
3	∞	1	{2, 4, 5}
4	∞	1	{4, 5, 6, 7}
5	∞	2	{4, 5, 6, 9}
6	∞	2	{4, 5, 9, 10}
7	∞	3	{4, 5, 9, 13}
8	∞	4	{4, 5, 13}
9	∞	4	{4, 5, 17}
10	4	--	{ }

nearest to the tip of the principal path. The offshoot which emanates from this node is then pursued until no further active subprograms are generated and the path terminates. The LIFO rule then selects another offshoot from the principal path and the process is repeated.

The priority rule is more flexible in its approach to the selection of the branching subprogram than is the LIFO rule. Under the priority rule, the node corresponding to the branching subprogram (the active subprogram with least lower bound) at one stage need not be on the same path as the node corresponding to the branching node at the preceding stage. The priority rule thus leads to the simultaneous exploration of all paths within the branch-and-bound tree. The path containing the node for which the corresponding subprogram yields an optimal solution may be the only path which is examined completely.

The LIFO rule was first used in a branch-and-bound algorithm for the mixed integer linear program by Dakin (1965). The priority rule was used both in the algorithm of Davis, Kendrick and Weitzman (1971) for the mixed integer linear program and in the algorithm of Falk and Soland (1969) for the concave nonlinear program.

4.2 Maxmin and Maxmax Rules

Upon the selection of a branching subprogram k, a constraint set partitioning procedure is applied to determine constraint sets for subprograms s and s + 1. This procedure begins with the selection of a branching variable x_{j_o} $(1 \leq j_o \leq n)$ as described in Section 2.3.

Two alternative rules which may be applied to select the branching variable are the maxmin and maxmax rules. These rules employ the penalty techniques of Chapter 3 to make an optimal selection. Consequently, the maxmin or maxmax branching variable selection rules may be used quite efficiently in conjunction with the bounding procedure of Section 3.1.

In the notation of Section 3.1, the maxmin rule results in the selection of a variable x_{j_o} for which

$$\max_{1 \le j \le n} \min \{p_0(j), p_1(j)\}$$

is attained. This expression is the same as the lower bound LB(k) for branching subprogram k. The maxmax rule results in the selection of a variable x_{j_o} for which

$$\max_{1 \le j \le n} \max \{p_0(j), p_1(j)\}$$

is attained. This second expression, which is larger than LB(k), involves the same basic quantities as those used in the computation of LB(k) but in a different logical combination. Consequently, branching variable selection using either the maxmin or maxmax rules may be done with little additional effort at the time that the lower bounding procedure is applied.

For both the maxmin and maxmax rules, the penalties $p_0(j_o)$ and $p_1(j_o)$ are related to the branching subprogram k and to subprograms s and s + 1 as follows:

$$g^k(x^k) \leq p_0(j_o) \leq g^s(x^s) \leq LB(s)$$

$$g^k(x^k) \leq p_1(j_o) \leq g^{s+1}(x^{s+1}) \leq LB(s+1).$$

By selecting the branching variable x_{j_o} such that

$$\min \{p_0(j), p_1(j)\}$$

is at a maximum for $j = j_o$, the maxmin rule guarantees the largest possible increase for <u>both</u> LB(s) and LB(s+1). On the other hand, by selecting the branching variable x_{j_o} such that

$$\max \{p_0(j), p_1(j)\}$$

is at a maximum for $j = j_o$, the maxmax rule guarantees the largest possible increase in <u>one</u> of LB(s) or LB(s+1) but guarantees nothing with respect to the other lower bound.

The following modified maxmax rule is a compromise which attempts to avoid the inability of the maxmax rule to guarantee an increase in both LB(s) and LB(s+1). The modified maxmax rule results in the

selection of a variable x_{j_o} for which

$$\max_{j \in Z} \max \{p_0(j), p_1(j)\}$$

is attained, where

$$Z = \{j \in N \mid \min \{p_0(j), p_1(j)\} > g^k(x^k)\} .$$

The maxmin branching variable selection rule was first used in a branch-and-bound algorithm for the mixed integer liner program by Davis, Kendrick and Weitzman (1971). The maxmax rule was first used for the mixed integer linear program by Dakin (1975). With the development of penalty techniques described in Chapter 3, the maxmin and maxmax rules may now be applied to solve the integer concave program.

4.3 Other Branching Variable Selection Rules

Additional branching variable selection rules may be employed to solve the special cases of the integer concave program described in Section 1.2. For the mixed integer linear program, alternative rules are the most noninteger rule and the weighted noninteger rule. For the concave nonlinear program, an alternative rule is the conventional rule of Falk and Soland (1969). These rules do not require the penalty techniques of Chapter 3. Consequently, these rules may be used in conjunction with the bounding procedure of either Section 2.2 or Section 3.1.

In the case of the mixed integer linear program, let

$$Z_1^k = \{j \in J \mid x_j^k \text{ is noninteger}\}$$

where x^k is the solution to the auxiliary linear program for branching subprogram k. The most noninteger rule results in the selection of a variable x_{j_o} for which

$$\max_{j \in Z_1^k} \min \{x_j^k - [x_j^k], \langle x_j^k \rangle - x_j^k\}$$

is attained. The weighted noninteger rule results in the selection of a variable x_{j_o} for which

$$\max_{j \in Z_1^k} |c_j| \min \{x_j^k - [x_j^k], \langle x_j^k \rangle - x_j^k\}$$

is attained. The weighted noninteger rule is an attempt to reflect the sensitivity of the objective function to changes in the variable values. The most noninteger and weighted noninteger rules provide a direct approach to branching variable selection, involving a small number of computational steps, as opposed to the detailed analysis which forms the basis for the maxmin and maxmax rules.

In the case of the concave nonlinear program, let

$$Z_2^k = \{j \in N \mid f_j(x_j^k) - g_j^k(x_j^k) > 0\}$$

where x^k is the solution to the auxiliary linear program for branching subprogram k. The conventional rule of Falk and Soland (1969) results in the selection of a variable x_{j_o} for which

$$\max_{j \in Z_2^k} f_j(x_j^k) - g_j^k(x_j^k)$$

is attained. This rule provides a direct approach to branching variable selection, involving a small number of computational steps, as opposed to the detailed analysis which forms the basis for the maxmin and maxmax rules for the concave nonlinear program.

5. SOLUTION STRATEGIES

A branch-and-bound method, such as method 1 described in Section 2.1, includes optional computational procedures and rules. These are: the bounding procedure, the branching subprogram selection rule, and the constraint set partitioning rule. In turn, the constraint set partitioning rule is equivalent to a branching variable selection rule and an option to perform limit tightening. A particular choice of these procedures and rules is called a solution strategy. A branch-and-bound method together with a solution strategy thus results in a branch-and-bound algorithm which can be applied to solve an integer concave program.

The various possible solution strategies result in algorithms having widely different convergence properties. Consequently, a solution strategy is usually chosen so as to optimize some measure of effectiveness associated with the branch-and-bound computation. Examples of this are: to minimize the number of stages required to complete the branch-and-bound computation; to minimize the number of stages required to determine an initial feasible solution; or to minimize the maximum number of active nodes over all stages.

Solution strategies consisting of various branching subprogram selection and branching variable selection rules for the mixed integer linear program have been evaluated by Mitra (1973). For the integer concave program considered here, the evaluation of the

effectiveness of solution strategies is extended to include the limit tightening option and the availability of an initial best upper bound.

The basic solution strategies for the integer concave program are discussed in Section 5.1 with consideration being given to the qualitative impact of the strategy on the convergence of the algorithm. Important examples of basic solution strategies are the LIFO/maxmax and priority/maxmin strategies. Section 5.2 discusses the formation of a composite solution strategy from two different basic strategies which are appropriate to the two different phases of the branch-and-bound method. The primary example of such a composite strategy is to use the LIFO/maxmax strategy during phase 1 and the priority/maxmin strategy during phase 2.

A new branch-and-bound method, the variable best upper bound method, has been developed for the solution of integer concave programs. This method serves to improve the performance of phase 1 computations when a composite solution strategy is employed. Its effectiveness is derived from the efficiencies which accrue when an initial estimate is available for the optimal objective function value f^*, as discussed in Section 5.3. The variable best upper bound method includes a means for generating and refining such an estimate during phase 1 of the branch-and-bound method. This is the subject of Sections 5.4 and 5.5.

5.1 Basic Solution Strategies

Figure 12 shows the basic solution strategies for the integer concave program as well as solution strategies which apply only to the

			Integer Concave Program												Mixed Integer Linear Program								Concave Nonlinear Program			
			1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
Bounding Procedure		Penalty	X	X	X	X	X	X	X	X	X	X	X	X												
		Normal													X	X	X	X	X	X	X	X	X	X	X	X
Branching Subprogram Selection Rule		Priority	X	X	X	X	X	X							X	X	X	X					X	X		
		LIFO							X	X	X	X	X	X					X	X	X	X			X	X
Constraint Set Partitioning Rule	Branching Variable Selection Rule	Maxmin	X	X					X	X																
		Maxmax			X	X					X	X														
		Modified Maxmax					X	X					X	X												
		Most Noninteger													X	X			X	X						
		Weighted Noninteger															X	X			X	X				
		Falk and Soland																					X	X	X	X
	Limit Tightening Option	Yes	X		X		X		X		X		X		X		X		X		X		X		X	
		No		X		X		X		X		X		X		X		X		X		X		X		X

FIGURE 12

BASIC SOLUTION STRATEGIES

particular cases of the mixed integer linear program and the concave nonlinear program. The bounding procedures shown include the normal lower bounding procedure described in Section 2.2, and the lower bounding procedure based upon the penalty techniques described in Section 3.1. The branching subprogram selection rules shown in Figure 12 include the LIFO and priority rules of Section 4.1. The branching variable selection rules include: the maxmin, maxmax, and modified maxmax rules of Section 4.2 (which apply to the integer concave program); the most noninteger and weighted noninteger rules of Section 4.3 (which apply only to the mixed integer linear program); and the conventional rule of Falk and Soland described in Section 4.3 (which applies only to the concave nonlinear program). Limit tightening, as described in Section 3.2, may be employed as an option.

Two combinations of branching subprogram selection/branching variable selection rules which are commonly employed in solution strategies are the priority/maxmin rules and the LIFO/maxmax rules. As stated in Section 4.2, the maxmin or maxmax rules may be used quite efficiently in conjunction with the system of stronger lower bounds based upon penalty techniques. This combination is adopted in the solution strategies shown in Figure 12. The priority/maxmin solution strategy (with limit tightening) was used for the mixed integer linear program by Davis, Kendrick and Weitzman (1971). The LIFO/maxmax solution strategy (but with the normal bounding procedure) was used for the mixed integer linear program by Dakin (1965).

Consider first the priority/maxmin solution strategy. As the branching subprogram k, the priority rule selects an active subprogram for which the corresponding lower bound is the least. This leads to the increase of the best lower bound as rapidly as possible. The maxmin branching variable selection rule, on the other hand, generates new subprograms s and s+1 such that the corresponding lower bounds increase as much as possible in comparison with the lower bound for subprogram k. The priority and maxmin rules are thus complementary and result in a solution strategy which emphasizes the convergence of the best lower bound.

An example of the branch-and-bound tree structure which results from the LIFO/maxmax solution strategy is given in Figure 13. The corresponding statistical data is shown in Table 4. As the branching subprogram k, the LIFO rule selects the subprogram which was the last to be added to the set of active subprograms. The maxmax branching variable selection rule generates new subprograms s and s+1 such that <u>one</u> of the corresponding lower bounds, say LB(s), increases as much as possible but guarantees nothing with respect to the amount of increase for the other lower bound, LB(s+1). Under the LIFO rule, subprogram s+1 would be the branching subprogram at the next stage.[4] That is, the

[4] If $p_0(j_o) < p_1(j_o)$, it is LB(s+1) which is guaranteed to increase as much as possible. In this case, subprogram s is selected as the branching subprogram at the next stage. This is accomplished by listing subprograms s and s+1 in the set of active subprograms in <u>reverse order</u>.

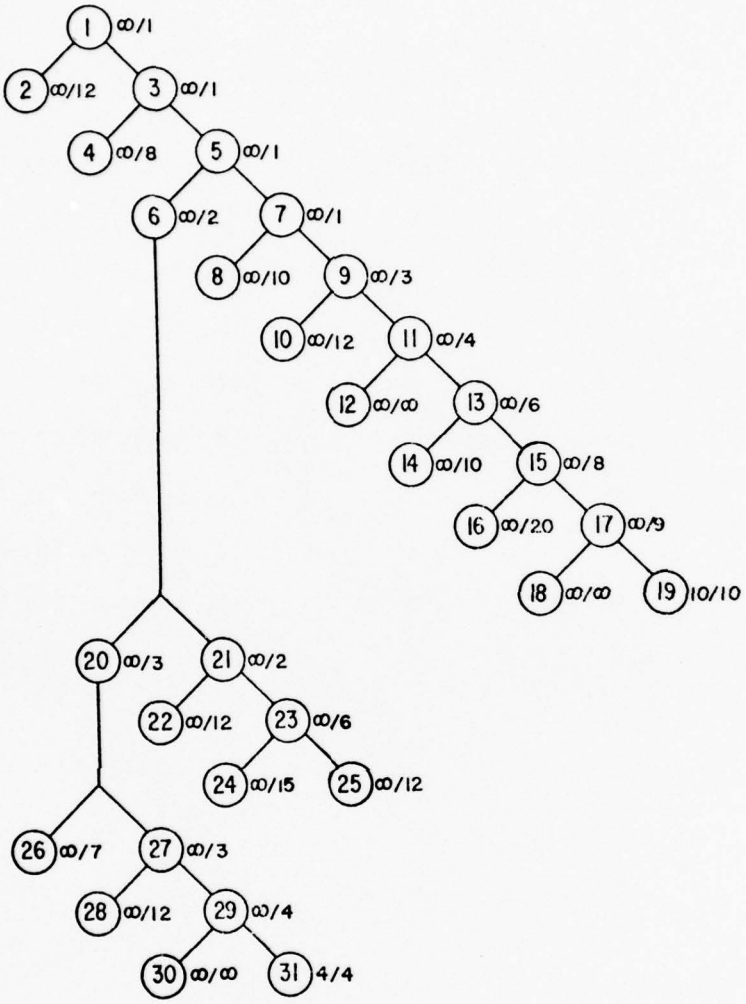


FIGURE 13

TREE STRUCTURE FOR THE
LIFO/MAXMAX SOLUTION STRATEGY

TABLE 4

STATISTICS FOR THE TREE WITH
THE LIFO/MAXMAX SOLUTION STRATEGY

t	BUB(t)	BLB(t)	N(t)
1	∞	1	{1}
2	∞	1	{2, 3}
3	∞	1	{2, 4, 5}
4	∞	1	{2, 4, 6, 7}
5	∞	2	{2, 4, 6, 8, 9}
6	∞	2	{2, 4, 6, 8, 10, 11}
7	∞	2	{2, 4, 6, 8, 10, 13}
8	∞	2	{2, 4, 6, 8, 10, 14, 15}
9	∞	2	{2, 4, 6, 8, 10, 14, 16, 17}
10	10	2	{4, 6}
11	10	2	{4, 20, 21}
12	10	3	{4, 20, 23}
13	10	3	{4, 20}
14	10	3	{4, 26, 27}
15	10	4	{4, 26, 29}
16	4	--	{ }

subprogram which has the larger lower bound remains in the set of active subprograms while the subprogram which has the smaller lower bound becomes the branching subprogram at the next stage. Considering the subprograms which correspond to nodes on the principal path of the tree, the sequence of lower bounds increases slowly. In the event that the bounding procedure yields a feasible solution along the principal path, this feasible solution may be optimal or near optimal, and the initial finite best upper bound resulting from this feasible solution would be small. As the active subprograms generated along the principal path correspond to larger lower bounds, many active subprograms could be eliminated with the determination of this best upper bound. The LIFO and maxmax rules are thus complimentary and result in a solution strategy which emphasizes the convergence of the best upper bound.

5.2 Strategies for Different Phases

The separation of the branch-and-bound method into two phases was discussed in Section 2.4. Phase 1 of the method seeks to determine an initial feasible solution, thereby establishing a finite best upper bound. Phase 2 seeks to verify the optimality of the current best feasible solution, an objective which is attained when the set of active subprograms becomes empty. Individual solution strategies differ with respect to their efficiency in accomplishing these phase 1/phase 2 objectives. For example, the LIFO/maxmax solution strategy is best suited

to the phase 1 objective but is inefficient with respect to the number of stages required to attain the phase 2 objective. The priority/maxmin solution strategy, on the other hand, is better suited to phase 2 than to phase 1.

A composite solution strategy results when different solution strategies are employed for the different phases of the branch-and-bound method. An example of such a composite is the use of the LIFO/maxmax solution strategy during phase 1 and the priority/maxmin solution strategy during phase 2. This example is discussed further below as an illustration of the techniques required to implement a composite strategy.

Assume that the branch-and-bound method begins computations with the LIFO/maxmax solution strategy and the lower bounding procedure which uses the stronger lower bound associated with the penalty techniques of Section 3.1. For each subprogram k during phase 1, apply the maxmax branching variable selection rule to determine a branching variable and the associated cuts L_1^k, U_1^k. Simultaneously, apply the maxmin branching variable selection rule to determine a branching variable and the associated cuts L_2^k, U_2^k. If an active subprogram k is selected as the branching subprogram during phase 1, define

$$H^s = H^k \cap L_1^k$$

$$H^{s+1} = H^k \cap U_1^k.$$

When phase 1 ends, the LIFO/maxmax solution strategy is replaced by the priority/maxmin solution strategy. The Lower bounding procedure is unchanged with this replacement. During phase 2, the maxmin branching variable selection rule and the determination of cuts L_2^k, U_2^k is continued but the application of the maxmax rule is no longer required. If an active subprogram k is selected as the branching subprogram during phase 2, define

$$H^s = H^k \cap L_2^k$$

$$H^{s+1} = H^k \cap U_2^k.$$

The LIFO/maxmax and priority/maxmin solution strategies are thus used in the particular phase for which each is best suited. The resulting composite solution strategy has convergence properties superior to those of either the LIFO/maxmax or priority/maxmin strategies individually. These properties are attained at the expense of having to make branching variable selections according to two rules during phase 1. For the example considered here, however, the additional computation required to perform two selections is negligible because of the similarities between the maxmax and maxmin rule.

The limit tightening option should be mentioned in connection with composite solution strategies. The procedures presented in Section 3.2 for limit tightening depend upon the presence of

a finite best upper bound for their implementation. As a consequence, limit tightening is only done during phase 2 of the branch-and-bound method.

5.3 Initial Best Upper Bound

In the application of any solution strategy, there are instances where the initial feasible solution is far from optimal. This can occur even for the LIFO/maxmax solution strategy. In such a case, the corresponding initial finite best upper bound may then be so large that relatively few active subprograms are eliminated through the comparison of lower bounds with the best upper bound. This effect contributes to the occurrence of lengthy branch-and-bound computations.

The effect can be alleviated when an initial estimate is available for the optimal objective function value f^*. If BUB(0) is the given estimate, modify the branch-and-bound method by setting

$$BUB(1) = \min \{BUB(0), UB(1)\}$$

at stage 1. Figure 14 presents the same example as was considered in Figure 13 (an application of the LIFO/maxmax solution strategy), but with an initial estimate BUB(0) = 5. Table 5 presents the corresponding statistical data. With this initial best upper bound, the method terminates a node when the corresponding lower bound exceeds BUB(0). The isovalue contour which corresponds to the lower bound value 5 (i.e., LB = 5) is shown in Figure 14. The subgraph consisting

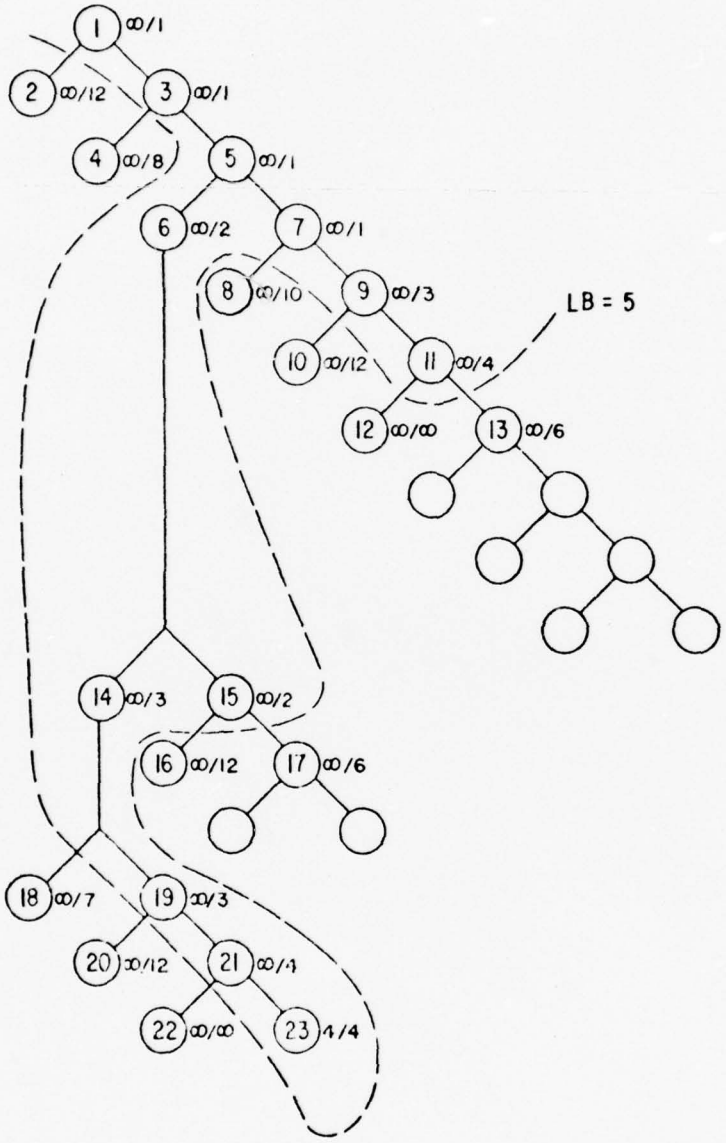


FIGURE 14

TREE STRUCTURE WITH A
GIVEN INITIAL BEST UPPER BOUND

TABLE 5

STATISTICS FOR THE TREE WITH
A GIVEN INITIAL BEST UPPER BOUND

t	BUB(t)	BLB(t)	N(t)
1	5	1	{1}
2	5	1	{3}
3	5	1	{5}
4	5	1	{6, 7}
5	5	2	{6, 9}
6	5	2	{6, 11}
7	5	2	{6}
8	5	2	{14, 15}
9	5	3	{14}
10	5	3	{19}
11	5	4	{21}
12	4	--	{ }

of the nodes examined under this method is bounded by the contour LB = 5 in the sense that each node of the subgraph is the terminal node of an arc emanating from within this contour. The availability of a good initial upper bound thus leads to the examination of fewer nodes. In particular, when the LIFO/maxmax solution strategy is employed, this provides a natural mechanism for limiting the tree search and assuring that the initial feasible solution will be near optimal.

In the case of an initial estimate BUB(0) being used which is less than f^*, no feasible solution to the master program will be found. Termination of the branch-and-bound computation with no feasible solution being found should be interpreted as implying that $f^* > BUB(0)$. In this case, computations should be repeated with a larger value of BUB(0).

The use of an upper limit on the optimal objective function value f^* to facilitate branch-and-bound computations was advocated by Land and Doig (1960) for the case of a mixed integer linear program.

5.4 Variable Best Upper Bounds

As noted in Section 5.3, an initial feasible solution which is far from optimal can occur no matter which solution strategy is used. For the case of a composite solution strategy, this occurrence has the effect of transferring the major burden of the branch-and-bound computation to phase 2. In the particular case of a composite

solution strategy which uses the LIFO/maxmax strategy during phase 1 and the priority/maxmin strategy during phase 2, the priority/maxmin strategy must then serve to determine better feasible solutions in addition to the usual phase 2 objective of verifying the optimality of the current best feasible solution. The determination of feasible solutions is, of course, the task for which the priority/maxmin strategy is least suited. On the other hand, phase 2 computations are facilitated when the intial feasible solution determined during phase 1 is near optimal. A new method is described in this section which can achieve this result.

The variable best upper bound method is intended for use during phase 1. This method serves to limit the tree search for the case where no initial estimate of f^* is available. The basis for the method is the treatment of the best upper bound as a free parameter. After selecting an initial value for the best upper bound, the subgraph bounded by the corresponding isovalue contour is explored. If a feasible solution is encountered, the method enters phase 2; otherwise, a larger value for the best upper bound is selected and the subgraph bounded by the corresponding contour is explored. This process is repeated until an initial feasible solution is encountered.

The efficient exploration of the successive subgraphs requires that two sets of active subprograms be maintained. At stage t, N(t) is the set of subprograms which would be considered to be active if the best upper bound were $+\infty$. N'(t) is the subset of N(t) consisting

of those subprograms which are active in the subgraph being examined. In the event that N'(t) becomes empty at stage t, a new best upper bound is established by setting

$$BUB(t) = BLB(t) + \Delta$$

$$N'(t) = \{i \in N(t) \mid LB(i) < BUB(t)\}$$

where the increment Δ is assumed to be an input to the method. N'(t) thus contains those subprograms of N(t) which lie in the band between two consecutive contours.

Figure 15 presents an example of the structure of the branch-and-bound tree which results from the variable best upper bound method having $\Delta = 2$. Table 6 gives the corresponding statistical data. In this particular case, the initial feasible solution is in fact optimal.

5.5 Method 2

The variable best upper bound method (method 2) is stated here using the terminology and notation established in Section 2.1 for the basic branch-and-bound method (method 1).

Stage t = 1 consists of only one subprogram (subprogram 1) which serves to initiate phase 1 of the branch-and-bound method. Subprogram 1 has the same constraint set as the master program, $M^1 = Q \cap H^1 \cap X$ where $H^1 = H^o$. The bounding procedure of the phase 1 solution strategy is applied to obtain lower bound LB(1) on

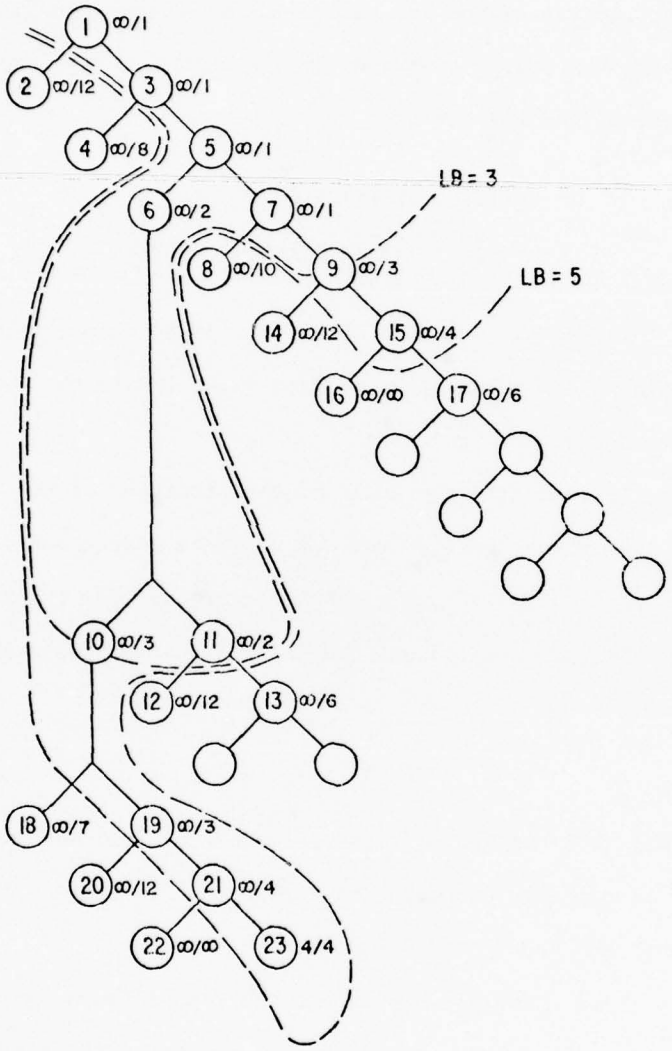


FIGURE 15

TREE STRUCTURE FOR THE
VARIABLE BEST UPPER BOUND METHOD

TABLE 6

STATISTICS FOR THE TREE WITH
THE VARIABLE BEST UPPER BOUND METHOD

t	BUB(t)	BLB(t)	N(t)	N'(t)
1	3	1	{1}	{1}
2	3	1	{2, 3}	{3}
3	3	1	{2, 4, 5}	{5}
4	3	1	{2, 4, 6, 7}	{6, 7}
5	3	2	{2, 4, 6, 8, 9}	{6}
6	3	2	{2, 4, 8, 9, 10, 11}	{11}
7	5	3	{2, 4, 8, 9, 10, 12, 13}	{9, 10}
8	5	3	{2, 4, 8, 10, 12, 13, 14, 15}	{10, 15}
9	5	3	{2, 4, 8, 10, 12, 13, 14, 17}	{10}
10	5	3	{2, 4, 8, 12, 13, 14, 17, 18, 19}	{19}
11	5	4	{2, 4, 8, 12, 13, 14, 17, 18, 20, 21}	{21}
12	4	--	{ }	{ }

subprogram 1 and upper bound UB(1) on the master program. If $UB(1) < \infty$ occurs, the branch-and-bound method enters phase 2 and the balance of the computation (for stage t=1 and subsequent stages) is the same as in method 1 but employing the phase 2 solution strategy. So assume that the case $UB(1) = \infty$ occurs. Let $N(1) = N'(1) = \{1\}$ and

$$BLB(1) = \min_{i \in N(1)} LB(i) = LB(1)$$

$$BUB(1) = BLB(1) + \Delta.$$

For $t \geq 2$, stage t of the branch-and-bound method will now be described. Assume that the following data (resulting from stage t-1) are given:

(i) $N(t-1)$ and $N'(t-1)$, nonempty subsets of $\{1, 2, \ldots, 2t - 3\}$ with $N'(t) \subset N(t)$;

(ii) $H^i = \{x \mid \ell^i \leq x \leq u^i\}$ and $LB(i)$ for $i \in N(t-1)$; and

(iii) $BUB(t-1)$.

A branching subprogram $k \in N'(t-1)$ is selected using the branching subprogram selection rule of the phase 1 solution strategy. Subprograms s and s+1 (where $s = 2t - 2$) are considered at this stage. The constraint set for subprogram k is partioned to form the constraint sets for subprograms s and s+1, accomplished by splitting hypercube H^k into hypercubes H^s and H^{s+1}.

Subprograms s and s+1 have constraint sets $M^s = Q \cap H^s \cap X$ and $M^{s+1} = Q \cap H^{s+1} \cap X$. The bounding procedure is applied to obtain lower bound LB(s) on subprogram s and upper bound UB(s) on the master program. LB(s+1) and UB(s+1) are determined similarly. If either $UB(s) < \infty$ or $UB(s+1) < \infty$ occurs, the branch-and-bound method enters phase 2 and the balance of the computation (for stage $t \geq 2$ and subsequent stages) is the same as in method 1 but employing the phase 2 solution strategy. So assume that the case $UB(s) = UB(s+1) = \infty$ occurs.

Form the index sets

$$I(t) = N(t-1) \setminus \{k\} \cup \{s, s+1\}$$

$$N(t) = \{i \in I(t) \mid LB(i) < \infty\}$$

$$N'(t) = \{i \in N(t) \mid LB(i) < BUB(t-1)\}.$$

The best lower bound at stage t is

$$BLB(t) = \min_{i \in N(t)} LB(i).$$

If $N'(t)$ is not empty, the best upper bound at stage t is $BUB(t) = BUB(t-1)$. If $N'(t)$ is empty,

$$BUB(t) = BLB(t) + \Delta$$

and redefine

$$N'(t) = \{i \in N(t) \mid LB(i) < BUB(t)\}.$$

The branch-and-bound method then proceeds to the next stage.

6. TEST PROGRAMS

This chapter presents sample integer concave programs of the types which may be solved using the branch-and-bound method. The complete formulation and solution of these programs is presented so that they may be used to test the operation of the ICØN algorithm. The preparation of input for the ICØN algorithm is described in Appendices A, B, and C using the test programs as examples.

Computational experience with the ICØN algorithm is summarized briefly for each type of program--concave nonlinear, mixed integer linear, and integer concave.

6.1 A Concave Nonlinear Program

Test program 1 is a concave nonlinear program obtained from Falk and Soland (1971). The formulation and solution of this test program are shown in Tables 7 and 8. Application of the branch-and-bound method to test program 1, using the priority/maxmin solution strategy and limit tightening, results in the statistics shown in Table 9. For any concave nonlinear program, phase 2 begins with the first subprogram. Three subprograms are examined for test program 1, with the solution being obtained at subprogram 3. The maximum size of the set of active nodes (the "branch-and-bound list") is 1.

Computational experience with concave nonlinear programs has included the solution of programs from Falk and Soland (1971) and

TABLE 7

TEST PROGRAM 1

minimize $\quad -130x_1 + f_2(x_2) - 160x_3 + f_4(x_4)$

such that
$$10x_1 + 10x_2 + 10x_3 + 10x_4 \leq 150$$
$$7x_1 + 5x_2 + 3x_3 + 2x_4 \leq 100$$
$$3x_1 + 5x_2 + 10x_3 + 15x_4 \leq 100$$

$$0 \leq x_1 \leq 100$$
$$0 \leq x_2 \leq 25$$
$$0 \leq x_3 \leq 100$$
$$0 \leq x_4 \leq 25$$

where

$$f_2(x_2) = \begin{cases} 0 & , \text{ if } x_2 = 0 \\ 2000 - 130x_2 - 10x_2^2 & , \text{ if } x_2 > 0 \end{cases}$$

$$f_4(x_4) = \begin{cases} 0 & , \text{ if } x_4 = 0 \\ 2000 - 200x_4 - 18x_4^2 & , \text{ if } x_4 > 0 \end{cases}$$

TABLE 8

SOLUTION TO TEST PROGRAM 1

$f^* = -2200$

$x_1 = 0$

$x_2 = 15$

$x_3 = 0$

$x_4 = 0$

TABLE 9

BRANCH-AND-BOUND STATISTICS FOR TEST PROGRAM 1

Node Enter Phase 2	1
Solution Node	3
Nodes Examined	3
Maximum List Size	1
Simplex Iterations[5]	9

[5]The first subprogram requires 5 simplex iterations.

Taha (1973). Additional concave nonlinear programs, of the type given in Section 6.3, have been solved using the ICØN algorithm.

6.2 A Mixed Integer Linear Program

Test program 2 is a mixed integer linear program obtained from Haldi (1964). The formulation and solution of this test program are shown in Tables 10 and 11. Application of the branch-and-bound method to test program 2, using the priority/maxmin solution strategy and limit tightening, results in the statistics shown in Table 12.

Computational experience with mixed integer linear programs has included the solution of programs from Haldi (1964), Garfinkel and Nemhauser (1972), and Giglio and Wagner (1964). Additional mixed integer linear programs, of the type given in Section 6.3, have been solved using the ICØN algorithm.

6.3 An Integer Concave Program

Test program 3 is an integer concave program obtained from Loomis (1973a). The formulation of this test program, as shown in Tables 13 through 17, includes both a separable concave objective function and integer-valued variables. An alternative (and equivalent) formulation as a mixed integer linear program is obtained by replacing condition (iv) in Table 13 with

$$\text{(iv)}' \quad x_j \text{ integer for } 1 \leq j \leq 24$$

and by defining

$$f_j(x_j) = c_j x_j$$

for $17 \leq j \leq 24$. A concave nonlinear program is obtained from test program 3 by deleting condition (iv) entirely. The solutions to test program 3, the equivalent mixed integer linear program, and the related concave nonlinear program are shown in Table 18. Application of the branch-and-bound method to these programs, using the LIFO/maxmax solution strategy and limit tightening, results in the statistics shown in Table 19.

TABLE 10

TEST PROGRAM 2

maximize $x_3 + x_4 + x_5$

such that

$$2x_1 + 3x_2 + x_3 + 2x_4 + 2x_5 \leq 18$$

$$3x_1 + 2x_2 + 2x_3 + x_4 + 2x_5 \leq 15$$

$$-6x_1 + x_3 \leq 0$$

$$-7x_2 + x_4 \leq 0$$

$$0 \leq x_1 \leq 1$$

$$0 \leq x_2 \leq 1$$

$$0 \leq x_3 \leq 6$$

$$0 \leq x_4 \leq 7$$

$$0 \leq x_5 \leq 9$$

x_j integer for $1 \leq j \leq 5$

TABLE 11

SOLUTION TO TEST PROGRAM 2

$$f^* = 7$$

$$x_1 = 0$$

$$x_2 = 1$$

$$x_3 = 0$$

$$x_4 = 7$$

$$x_5 = 0$$

TABLE 12

BRANCH-AND-BOUND STATISTICS FOR TEST PROGRAM 2

Node Enter Phase 2	6
Solution Node	32
Nodes Examined	33
Maximum List Size	3
Simplex Iterations[6]	67

[6]The first subprogram requires 5 simplex iterations.

TABLE 13

TEST PROGRAM 3

minimize $\sum_{j=1}^{16} c_j x_j + \sum_{j=17}^{24} f_j(x_j)$

such that (i) $Ax \geq b$

$$
\begin{cases}
x_4 + x_5 \leq 3 \\
x_6 - 2x_7 + x_8 = 0 \\
x_2 + x_3 \geq 2 \\
x_7 - x_6 \geq 0 \\
x_9 - x_{10} + x_{11} \geq 0 \\
-x_9 + 4x_{10} - x_{11} \geq 0 \\
x_6 + x_7 + x_8 + x_9 + x_{10} + x_{11} \geq 3 \\
\text{(ii)} \quad 19x_{17} - x_1 - x_2 - x_3 \geq 0 \\
3x_{18} - x_4 - x_5 \geq 0 \\
9x_{19} - x_9 - x_{10} - x_{11} \geq 0 \\
18x_{20} - x_6 - x_7 - x_8 - x_9 - x_{10} - x_{11} \geq 0 \\
4x_{21} - x_{13} \geq 0 \\
12x_{22} - x_{14} \geq 0 \\
16x_{23} - x_{13} - x_{14} \geq 0 \\
31x_{24} - x_{12} - x_{13} - x_{14} - x_{15} - x_{16} \geq 0
\end{cases}
$$

TABLE 13 (Continued)

(iii) $\ell^o \leq x \leq u^o$

(iv) x_j integer for $1 \leq j \leq 16$

where

$$f_j(x_j) = \begin{cases} 0 \text{ , if } x_j = 0 \\ c_j, \text{ if } x_j > 0 \end{cases}$$

TABLE 14

CONSTRAINT MATRIX A = $[a_{ij}]$ FOR
TEST PROGRAM 3

Column Number j \ Row Number i	1	2	3	4	5	6	7	8
1	0	0	.00741	0	0	0	0	0
2	0	0	.00400	0	0	0	0	.073
3	0	0	0	0	0	0	0	.526
4	0	0	.11420	40.664	.2719	.39174	0	3.789
5	0	0	.06441	31.598	.2865	.32166	0	1.891
6	-.0644	-.3344	.00754	11.100	.0927	.06931	-.251	.400
7	.9258	1.0550	.38303	36.958	1.1107	.45466	26.755	11.875
8	0	0	0	0	0	0	0	0
9	.1154	.1612	.09924	21.934	.2083	.27466	12.250	5.145
10	.3922	.0154	.18658	5.744	.2214	0	0	20.692
11	.6738	.7411	.13567	16.684	.9351	.28256	9.163	1.907
12	.0886	0	.01190	0	.0733	.02106	4.687	0
13	1.0278	.9476	.14200	9.329	.7313	.31088	23.326	2.138
14	.2955	.5430	.03058	2.932	.2530	.07237	5.365	.202
15	.0381	.0502	0	.464	0	0	3.830	.580
16	.0461	0	0	2.100	0	.03359	0	1.495

TABLE 14 (Continued)

Column Number j	Row Number i 9	10	11	12	13	14	15	16
1	.0615	0	0	0	.1371	.01521	0	0
2	.0225	0	0	0	.0933	0	0	0
3	0	0	0	.0742	.1888	.01446	0	0
4	0	0	0	0	.2622	0	0	0
5	0	0	0	0	.2643	.10047	0	0
6	.1466	.0379	.18941	-2.8200	-1.0364	-.71932	-.16689	.00601
7	.7422	.4718	.10079	2.4223	2.0501	.71301	.26709	.30637
8	0	0	0	0	0	0	0	0
9	.0682	.5904	0	0	.6843	0	0	.39258
10	1.2945	0	0	.9652	0	.46116	0	0
11	.2847	0	.21799	.5351	.7933	.15437	0	0
12	.0224	1.1783	.63822	.7381	.6234	.40417	.15283	.29269
13	.8700	2.1150	.57995	1.1132	.3276	.41676	.86077	.26108
14	.1890	.3723	.07318	.2031	.0837	.12291	.19143	.03371
15	.0705	.3664	.22251	.6653	0	.34865	.35038	.14164
16	0	.3969	.24297	.6081	0	.40708	.22731	.12580

TABLE 15

RIGHT-HAND-SIDE VECTOR FOR
TEST PROGRAM 3

i	b_i
1	3.9415
2	3.8398
3	1.09980
4	138.190
5	3.8663
6	0.01841
7	77.595
8	29.070
9	2.6440
10	5.0324
11	2.13960
12	7.1200
13	4.7010
14	0.03371
15	3.90961
16	1.96236

TABLE 16

LIMITS ON THE VARIABLES FOR
TEST PROGRAM 3

j	ℓ_j^o	u_j^o
1	4	11
2	0	4
3	0	4
4	0	3
5	0	3
6	0	2
7	0	3
8	0	4
9	0	4
10	0	2
11	0	3
12	0	3
13	0	4
14	0	12
15	0	7
16	0	5
17	0	1
18	0	1
19	0	1
20	0	1
21	0	1
22	0	1
23	0	1
24	0	1

TABLE 17

COST DATA FOR TEST PROGRAM 3

j	c_j
1	22.9
2	33.2
3	12.5
4	11.9
5	14.9
6	41.0
7	260.3
8	82.0
9	101.4
10	167.1
11	138.9
12	54.9
13	90.6
14	15.9
15	10.2
16	7.4
17	30.2
18	10.9
19	83.3
20	4.2
21	62.8
22	11.8
23	57.6
24	4.2

TABLE 18

SOLUTION TO TEST PROGRAM 3

	Concave Nonlinear Program	Mixed Integer Linear Program	Integer Concave Program
$f^* =$	958.048	974.3	974.3
$x_1 =$	4	4	4
$x_2 =$	0	0	0
$x_3 =$	2	2	2
$x_4 =$	3	3	3
$x_5 =$	0	0	0
$x_6 =$	0.00445491	0	0
$x_7 =$	1	1	1
$x_8 =$	1.99555	2	2
$x_9 =$	0	0	0
$x_{10} =$	0	0	0
$x_{11} =$	0	0	0
$x_{12} =$	0	0	0
$x_{13} =$	0	0	0
$x_{14} =$	11.2654	12	12
$x_{15} =$	4.56966	5	5
$x_{16} =$	5	5	5
$x_{17} =$	0.315789	1	0.315789
$x_{18} =$	1	1	1
$x_{19} =$	0	0	0
$x_{20} =$	0.166667	1	0.166667
$x_{21} =$	0	0	0
$x_{22} =$	0.938785	1	1
$x_{23} =$	0.704088	1	0.750000
$x_{24} =$	0.672099	1	0.709677

TABLE 19

BRANCH-AND-BOUND STATISTICS FOR TEST PROGRAM 3

	Concave Nonlinear Program	Mixed Integer Linear Program	Integer Concave Program
Node Enter Phase 2	1	31	43
Solution Node	2	73	167
Nodes Examined	5	79	225
Maximum List Size	1	13	17
Simplex Iterations[7]	78	367	1018

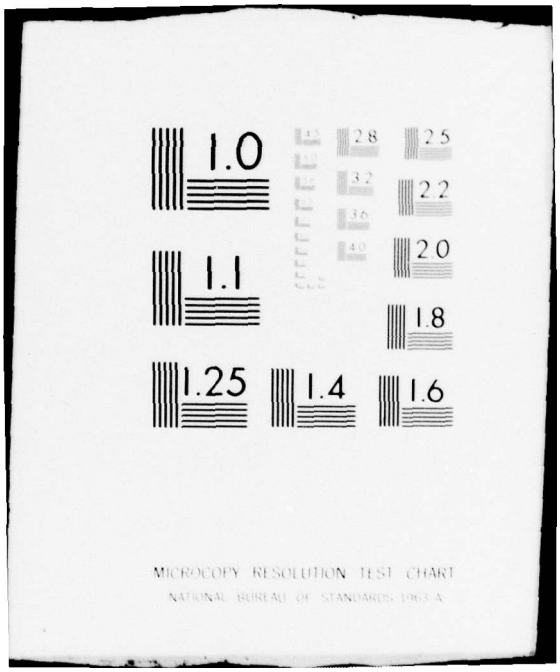
[7] The first subprogram requires 68 simplex iterations.

AD-A077 842 NAVAL SURFACE WEAPONS CENTER DAHLGREN LAB VA F/G 12/2
SOLUTION OF THE INTEGER CONCAVE PROGRAM USING THE ICON ALGORITH--ETC(U)
NOV 79 H W LOOMIS

UNCLASSIFIED NSWC/TR-3119 NL

2 OF 2
AD A077 842

END
DATE FILMED
1-80
DDC


1.0
1.1
1.25
1.4
1.6
2.8
2.5
3.2
2.2
3.6
2.0
4.0
1.8
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

APPENDIX A

INPUT DESCRIPTION

The input data for the ICØN branch-and-bound algorithm are described in this appendix. This data consists of the following seven basic items:

(i) Control cards,

(ii) Constraint matrix and right-hand-side vector,

(iii) Lower and upper limits on the variables,

(iv) Cost data,

(v) Lists of integer/concave variables,

(vi) User input, and

(vii) Initial feasible basis for the first subprogram.

Each data item will be described in detail below. The input data are accepted by the ICØN algorithm in the sequence indicated, with data items (vi) and (vii) being optional depending upon the requirements of a particular program being solved. All input data is normally read from tape unit 5. However, data item (ii) can be read from a separate tape unit (tape unit 1) if desired by the user. Appendix C contains the input data for the three test programs.

CONTROL CARDS

The four control cards are required input for each program. These specify the program dimensions, the solution method to be used, and program identification information.

Card 1 (Format 16I5)

The first ten parameters must be specified by the user. The last six parameters need not be specified unless a value other than the default value is desired.

Variable	Card Columns	Description
1. N	1-5	Number of variables (excluding slack, surplus or other variables which are automatically added to the program as required)
2. M	6-10	Number of constraints (excluding the lower and upper limit constraints on the variables)
3. ITYPE	11-15	Program type (0 = integer concave; 1 = mixed integer linear; 2 = concave nonlinear; 3 = linear)
4. NSTRAT	16-20	Number of solution strategies to be used (1 = one strategy to be used in both phases; 2 = two strategies, the first to be used in phase 1 and the second in phase 2)
5. NØDRL1	21-25	Node selection rule for the first solution strategy (0 = priority rule; 1 = LIFO rule)
6. NBVRL1	26-30	Branching variable selection rule for the first solution strategy (0 = maxmin rule; 1 = maxmax rule; 2 = modified maxmax rule; 3 = most noninteger rule for mixed integer linear programs; 4 = weighted noninteger rule for mixed

Variable	Card Columns	Description
		integer linear programs; 5 = conventional rule for concave programs)
7. NTITE1	31-35	Limit tightening option for the first solution strategy (0 = tighten limits before solving a subprogram; 1 = leave the limits alone)
8. NØDRL2	36-40	Node selection rule for the second solution strategy
9. NBVRL2	41-45	Branching variable selection rule for the second solution strategy
10. NTITE2	46-50	Limit tightening option for the second solution strategy
11. MXLIST	51-55	Maximum number of nodes allowed in the branch-and-bound list (at most 1000; default value is 1000)
12. LISTØP	56-60	Branch-and-bound list option (0 = include the basis inverse in the list; 1 = *do not* include the basis inverse but rather use the basis reinversion feature to regenerate it)
13. ITAPE	61-65	Input tape for the constraint matrix and right-hand-side vector (1 or 5; default value is 5)
14. IFB	66-70	Initial feasible basis option (0 = not used; 1 = used)
15. MXITER	71-75	Maximum number of pivots allowed for any one subprogram (default value is 1000)
16. MBINV	76-80	Number of pivots between basis reinversions (0 = basis reinversion feature not used)

Card 2 (Format 3I5)

The three parameters on this card need not be specified unless a value other than the default value is desired.

Variable	Card Columns	Description
1. IØUTPT	1-5	Output level option (0 = minimum output; 1 = summary output for each node; 2 = variable limits and costs for each node; 3 = sensitivity analysis information, limit tightening information, and the branch-and-bound list; 4 = complete linear program solution at each node; 5 = the basis inverse corresponding to the linear program solution)
2. ITRACE	6-10	Program trace information (0 = no trace; 1 = partial trace of subroutine calling sequence; 2 = complete trace)
3. MSTART	11-15	Job restart option (0 = begin problem from input data; 1 = begin problem from restart tapes 7 and 8)

Card 3 (Format 5E12.0)

The five parameters on this card need not be specified unless a value other than the default value is desired.

Variable	Card Columns	Description
1. TIME1	1-12	Maximum job execution time in seconds, the time at which restart tapes 9 and 10 are prepared (default value is 180 seconds)
2. TØL1	13-24	Tolerance for comparison of objective function values (default value is 10^{-11})
3. TØL2	25-36	Tolerance for comparison of variable values (default value is 10^{-11})

Variable	Card Columns	Description
4. UNØT	37-48	Initial best upper bound (default value is 10^{100})
5. PCBUB	49-60	Variable best upper bound solution method option (if PCBUB = 0, the variable best upper bound method is not to be used; otherwise, the method is to be used with PCBUB as the quantity for incrementing the best upper bound during phase 1)

Card 4 (Format 10A8)

An 80 character alphanumeric identifier for the program is specified next. This is read into the array

(ALPHA(I), I = 1,10)

under the format specified.

CONSTRAINT MATRIX AND RIGHT-HAND-SIDE VECTOR

The MxN constraint matrix and the M component right-hand-side vector are required input for each program. If parameter ITAPE on the first control card has the value 1, this information is read from tape unit 1; otherwise, it is assumed that this information follows immediately after the control cards on tape unit 5. For large programs, the constraint matrix and right-hand-side vector can be very bulky. The input tape option is useful in such a situation since, with ITAPE = 1, this data need not be provided as input in card form but can be read from a previously prepared tape.

For large programs, the constraint matrix is often sparse, having many entries which are zero. Consequently, the input associated with the constraint matrix can be greatly reduced if only the nonzero entries need be specified. Also, it is noted that, among the nonzero entries, certain values recur often and that input can be further reduced by establishing a table of constants in which each such value is listed only once.

The constraints of the program are assumed to be ordered as follows: less than or equal, equality, and greater than or equal constraints.

Card 1 (Format 3I5)

Variable	Card Columns	Description
1. M1	1-5	The number of less than or equal constraints in the program
2. M2	6-10	The number of equality constraints in the program
3. M3	11-15	The number of greater than or equal constraints in the program

Card(s) 2 (Format 16I5)

The number of nonzero entries in each column of the constraint matrix is specified next. This information is read into the array

(NZ(J), J = 1,N)

under the format specified. Multiple input cards are needed if N exceeds 16, with subsequent cards having the same format. An auxiliary parameter, NSUM, is computed to be the sum of the values NZ(J) for J = 1,N.

Card(s) 3 (Format 16I5)

For each column of the constraint matrix, the nonzero entries are specified as follows: an indicator of the row in which the entry occurs, and a pointer to the appropriate value in the table of constants.

For the J-th column, there are NZ(J) nonzero entries. Let IR(K) and IA(K) for K = 1,NZ(J) denote the row of the K-th nonzero entry and the pointer to the table of constants. The elements of the arrays IR and IA alternate on the input cards and are read in the sequence

(IR(K), IA(K), K = 1,NZ(J))

under the format specified. Multiple input cards are needed if NZ(J) exceeds 8, with subsequent cards having the same format.

This information is repeated for each column J = 1,N.

Card 4 (Format I5)

The input for the table of constants is preceded by the following parameter:

Variable	Card Columns	Description
1. NTC	1-5	Number of entries in the table of constants

Card(s) 5 (Format 6E12.0)

The table of constants is specified next. This information is read into the array

(TC(K), K = 1,NTC)

under the format indicated. Multiple input cards are needed if NTC exceeds 6, with subsequent cards having the same format.

Card(s) 6 (Format 6E12.0)

The right-hand-side vector is specified next. This information is read into the array

(BØRIG(I), I = 1,M)

under the format indicated. Multiple input cards are needed if M exceeds 6, with subsequent cards having the same format.

LOWER AND UPPER LIMITS ON THE VARIABLES

The lower and upper limits on the program variables are assigned initial values of 0 and $+\infty$ (respectively) by the program. The following inputs permit the assignment of initial values other than the default values. Cards 1 and 4 are mandatory inputs.

Card 1 (Format I5)

Variable	Card Columns	Description
1. NDN	1-5	The number of lower limits to be assigned values other than the default value 0

If NDN is not zero, the following two cards are provided as input.

Card(s) 2 (Format 16I5)

A list of the variables for which the lower limits are to be assigned values other than the default value 0 is given next. This information is read into the array

(NV(K), K = 1,NDN)

under the format indicated. Multiple input cards are needed if NDN exceeds 16, with subsequent cards having the same format.

Card(s) 3 (Format 6E12.0)

A list of the lower limits corresponding to the variables specified on card(s) 2 is given next. This information is read into the array

(V(K), K = 1,NDN)

under the format indicated. Multiple input cards are needed if NDN exceeds 6, with subsequent cards having the same format.

Card 4 (Format I5)

Variable	Card Columns	Description
1. NUP	1-5	The number of upper limits to be assigned values other than the default value + ∞

If NUP is not zero, the following two cards are provided as input.

Card(s) 5 (Format 16I5)

A list of the variables for which the upper limits are to be assigned values other than the default value + ∞ is given next. This information is read into the array

(NV(K), K = 1,NUP)

under the format indicated. Multiple input cards are needed if NUP exceeds 16, with subsequent cards having the same format.

Card(s) 6 (Format 6E12.0)

A list of the limits corresponding to the variables specified on card(s) 5 is given next. This information is read into the array

(V(K), K = 1,NUP)

under the format indicated. Multiple input cards are needed if NUP exceeds 6, with subsequent cards having the same format.

COST DATA

The cost data (objective function coefficients) for those variables which enter linearly into the objective function are the next inputs. For variables which enter nonlinearly into the objective function, the appropriate cost data is obtained by calling the user supplied subroutine GETØBJ.

Card(s) 1 (Format 6E12.0)

The cost data is read into the array

(C2(J), J = 1,N)

under the format indicated. Multiple input cards are needed if N exceeds 6, with subsequent cards having the same format.

For variables J which enter nonlinearly into the objective function, the corresponding value C2(J) should be set to zero or left blank.

LISTS OF INTEGER/CONCAVE VARIABLES

A list of the integer variables and/or a list of the concave variables are the next inputs. For an integer concave program (ITYPE = 0), both lists are required input. Only the list of integer variables is required for a mixed integer linear program (ITYPE = 1) and only the list of concave variables is required for a concave nonlinear program (ITYPE = 2). Neither list is required for a linear program (ITYPE = 3).

Card 1 (Format I5)

Variable	Card Columns	Description
1. NINT	1-5	Number of integer variables

Card(s) 2 (Format 16I5)

The list of the integer variables is read into the array

(NV(K), K = 1,NINT)

under the format indicated. Multiple input cards are needed if NINT exceeds 16, with subsequent cards having the same format.

Card 3 (Format I5)

Variable	Card Columns	Description
1. NCC	1-5	Number of concave variables

Card(s) 4 (Format 16I5)

The list of concave variables is read into the array

(NV(K), K = 1,NCC)

under the format indicated. Multiple input cards are needed if NCC exceeds 16, with subsequent cards having the same format.

USER INPUT

For integer concave and concave programs (ITYPE = 0 or 2), the user supplied subroutine GETØBJ computes the nonlinear components appearing in the objective function. The user supplied subroutine READIN accepts any input which may be required to facilitate this computation. This user input immediately follows the lists of integer/concave variables in the input stream.

INITIAL FEASIBLE BASIS FOR THE FIRST SUBPROGRAM

In the integer concave program, the first subprogram requires substantially more effort to solve than the subsequent subprograms. The first subprogram may involve a two phase linear program beginning from an initial basis which consists of slack and artificial variables, unlike subsequent subprograms which are solved much more rapidly because computations begin from a near optimal basis. For a large integer concave program involving many variables and/or constraints, the first subprogram may require substantial computer time for its solution.

Having solved an integer concave program, a related program with similar data can be solved more quickly using the initial feasible basis option, which is signalled by setting IFB = 1 on the first control card. An initial (dual or primal) feasible basis for the first subprogram is then included in the program input. A basis inversion feature is used to commence the solution of the first subprogram from this basis.

Even on the first attempt to solve an integer concave program, the initial feasible basis option can be used profitably if a feasible solution is known.

Card(s) 1 (Format 16I5)

A list of the variables which constitute the initial feasible basis is read into the array

(IBV(I), I = 1,M)

under the format indicated. Multiple input cards are needed if M exceeds 16, with subsequent cards having the same format.

Card 2 (Format I5)

Variable	Card Columns	Description
1. NUP	1-5	The number of nonbasic variables initially at upper bound

Card(s) 3 (Format 16I5)

If NUP is nonzero, the list of nonbasic variables initially at upper bound is required input. This list is read into the array

(NV(K), K = 1,NUP)

under the format indicated. Multiple input cards are needed if NUP exceeds 16, with subsequent cards having the same format.

APPENDIX B

USER SUBROUTINES

This appendix describes the three routines which must be coded by the user for each integer concave program to be solved. These are:

(i) Program MAIN,

(ii) Subroutine READIN, and

(iii) Subroutine GETØBJ.

Appendix C contains the actual routines used in the solution of the three test programs.

Program MAIN serves three basic functions. The first of these is the specification of the ten tape units used by the ICØN algorithm:

Tape Unit	Description
1	Alternate input tape for constraint matrix and right-hand-side vector
2	Random access mass storage used to save integer numbers/arrays in the branch-and-bound list
3	Random access mass storage used to save floating point numbers/arrays in the branch-and-bound list
4	Scratch tape
5	Input tape
6	*Output tape*
7	Job restart input tape created at the end of a previous job
8	Job restart input tape
9	Job restart output tape created in the event that the maximum job execution time (specified on the third control card) is reached by this job
10	Job restart output tape

In the CDC 6700 computer system, these tape units are in reality areas of mass storage. Information can be placed in or retrieved from these

areas of mass storage by means of the read and write instructions formerly used for peripheral tape devices.

The second basic function served by the program MAIN is the allocation of core storage to the two basic arrays (arrays IF and F) used in the ICØN algorithm. These arrays are in turn subdivided into the various arrays required in the branch-and-bound computation. The user must assign fixed dimensions to arrays IF and F commensurate with the size of the program or programs to be solved. The parameters NI and NF are assigned values equal to these dimensions. The ICØN algorithm determines the minimum dimensions required to solve a given program. In the event that either of the assigned dimensions fails to meet the minimum required value, a diagnostic will be printed and execution will be terminated.

The final function served by program MAIN is to transfer control of the computation to the branch-and-bound algorithm. This is accomplished by calling subroutine ICØN, the master subroutine in the algorithm, and by passing the locations of arrays IF and F together with parameters NI and NF to the subroutine.

Subroutines READIN and GETØBJ are required only for integer concave and concave nonlinear programs. These subroutines together provide the mechanism by which the nonlinear components $f_j(x_j)$ appearing in the program objective function are computed. Subroutine READIN is called once, just before the branch-and-bound algorithm commences. This subroutine serves to read in any input data which may be required in the

computation of the functions $f_j(x_j)$ and also serves to perform any supporting computations or manipulation of data which need be done only once. Subroutine GETØBJ is called repeatedly in the course of the branch-and-bound algorithm. It serves to compute the value $f_j(x_j)$ for any particular choices of j and x_j. Note that the index j used in subroutines READIN and GETØBJ refers to the j-th nonlinear component of the objective function. Any data which is to be passed from subroutine READIN to subroutine GETØBJ should be placed in labeled common storage.

APPENDIX C

INPUT AND USER SUBROUTINES FOR THE TEST PROGRAMS

The user subroutines and input for the three test programs are presented in this appendix. This provides an illustration of the discussions given in Appendices A and B.

The user supplied subroutines for test program 1 are shown in Exhibit 1. In program MAIN, lines 1-3 and 8-11 serve to specify the ten tape units required together with the amount of core storage to be allocated to the associated buffer areas. Subroutine BUFSHAR, a system subroutine available in the CDC 6700 computer system, enables the assignment of a single buffer area to two or more tape units. Lines 4-7 in program MAIN allocate storage to the arrays IF and F, which are located in blank common for test program 1. Control is transferred to the branch-and-bound algorithm at line 13. Subroutines READIN and GETØBJ are coded for the general concave nonlinear program in which the functions $f_j(x_j)$ are quadratic for $x_j \neq 0$ and are zero for $x_j = 0$:

$$f_j(x_j) = \begin{cases} 0 & , \text{ if } x_j = 0 \\ C1(j) + C2(j) \cdot x_j + C3(j) \cdot x_j^2 & , \text{ if } x_j \neq 0. \end{cases}$$

Subroutine READIN accepts as input the number of nonlinear components of the objective function (NCC) and the coefficients of the quadratic terms (C1(j), C2(j), and C3(j) for j = 1,NCC). This data is passed to subroutine GETØBJ using labeled common storage. Note that, in the computation of $f_j(x_j)$ in subroutine GETØBJ, the branch-and-bound input parameter TØL2 (located in CØMMØN/P1/) is used as a tolerance to test if $x_j = 0$.

The input for test program 1 is shown in Exhibit 2. The first four cards are the control cards. The constraint matrix and right-hand-side vector appear on cards 5-13. The lower and upper limits on the variables are specified on cards 14-17. Cost data for the variables which enter linearly into the objective function appear on card 18. The list of the concave variables in the program is given on cards 19 and 20. The last three cards are the input for user subroutine READIN.

The user supplied subroutines for test program 2 are shown in Exhibit 3. Program MAIN for test program 2 is in its simplest form. Subroutines READIN and GETØBJ are dummy subroutines since these are not required for a mixed integer linear program. The input data for test program 2 are shown in Exhibit 4. Originally a maximization problem, test program 2 has been converted to a minimization problem by changing the sign of the cost data appearing on card 19. Note that, in the case of a concave or integer concave program, the conversion from maximization to minimization would require a corresponding change in sign of $f_j(x_j)$ computed by subroutine GETØBJ.

The user supplied subroutines and input for test program 3 are shown in Exhibits 5 and 6. The user subroutines differ from those of test program 1 only in the dimensions assigned to arrays.

EXHIBIT 1

USER SUBROUTINES FOR TEST PROGRAM 1

```
      PROGRAM MAIN (INPUT,OUTPUT,TAPE5=INPUT,TAPE6=OUTPUT,                MAIN0001
     1             TAPE1,TAPE2=500,TAPE3=2000,TAPE4=0,                    MAIN0002
     2             TAPE7,TAPE8,TAPE9=0,TAPE10=0)                          MAIN0003
C ARRAYS IF AND F ARE DIMENSIONED BY THE USER TO SUIT HIS PROGRAM.        MAIN0004
      COMMON IF(1070),F(2165)                                             MAIN0005
      NI=1070                                                             MAIN0006
      NF=2165                                                             MAIN0007
C COMMON BUFFERS FOR TAPES 1 AND 4, 7 AND 9, 8 AND 10.                    MAIN0008
      CALL BUFSHAR (1,4)                                                  MAIN0009
      CALL BUFSHAR (7,9)                                                  MAIN0010
      CALL BUFSHAR (8,10)                                                 MAIN0011
C TRANSFER CONTROL TO THE BRANCH-AND-BOUND ALGORITHM.                     MAIN0012
      CALL ICON (IF,F,NI,NF)                                              MAIN0013
      CALL EXIT                                                           MAIN0014
      END                                                                 MAIN0015

      SUBROUTINE READIN                                                   READ0001
C QUADRATIC FUNCTION WITH SETUP COST AT THE ORIGIN.                       READ0002
      COMMON/INPUT/NCC,C1(2),C2(2),C3(2)                                  READ0003
      READ(5,1000)NCC                                                     READ0004
      WRITE(6,1002)NCC                                                    READ0005
      DO100J=1,NCC                                                        READ0006
      READ(5,1001)C1(J),C2(J),C3(J)                                       READ0007
  100 WRITE(6,1003)J,C1(J),C2(J),C3(J)                                    READ0008
      RETURN                                                              READ0009
 1000 FORMAT(I5)                                                          READ0010
 1001 FORMAT(3E12.0)                                                      READ0011
 1002 FORMAT(42H0USER INPUT FOR BRANCH-AND-BOUND ALGORITHM/                READ0012
     1       30H0NUMBER OF CONCAVE VARIABLES =,I5/                        READ0013
     2       1HO,3X,2HCC,14X,2HC1,15X,2HC2,15X,2HC3/1X,8HVARIABLE/        READ0014
     3       2X,6HNUMBER//)                                               READ0015
 1003 FORMAT(3X,I5,3X,3(E15.6,2X))                                        READ0016
      END                                                                 READ0017

      SUBROUTINE GETOBJ (J,X,OBJ)                                         GETO0001
C QUADRATIC FUNCTION WITH SETUP COST AT THE ORIGIN.                       GETO0002
      COMMON/P1/N,M,ITYPE,NSTRAT,NODRL1,NBVRL1,NTITE1,NODRL2,NBVRL2,      GETO0003
     1          NTITE2,MXLIST,LISTOP,ITAPE,IFB,MXITER,MBINV,IOUTPT,       GETO0004
     2          ITRACE,MSTART,TIME1,TOL1,TOL2,PCBUB,ALPHA(10)             GETO0005
      COMMON/INPUT/NCC,C1(2),C2(2),C3(2)                                  GETO0006
      IF(X.GT.TOL2)GOTO100                                                GETO0007
      OBJ=0.0                                                             GETO0008
      RETURN                                                              GETO0009
  100 OBJ=C1(J) + X*(C2(J) + X*C3(J))                                     GETO0010
      RETURN                                                              GETO0011
      END                                                                 GETO0012
```

EXHIBIT 2

INPUT DATA FOR TEST PROGRAM 1

```
4    3    2    1    0    0    0
3    0    0
      90        .0001        .0001          0          0
TEST PROGRAM 1. CONCAVE NONLINEAR PROGRAM
3    0    0
3    3    3    3
1    5    2    4    3    2
1    5    2    3    3    3
1    5    2    2    3    5
1    5    2    1    3    6
6
        2            3            5          7         10         15
      150          100          100
0
4
1    2    3    4
      100           25          100         25
     -130            0         -160          0
2
2    4
2
     2000         -130          -10
     2000         -200          -18
```

EXHIBIT 3

USER SUBROUTINES FOR TEST PROGRAM 2

```
      PROGRAM MAIN (INPUT,OUTPUT,TAPE5=INPUT,TAPE6=OUTPUT,TAPE1,TAPE2,   MAIN0001
     1             TAPE3,TAPE4,TAPE7,TAPE8,TAPE9,TAPE10)                 MAIN0002
C ARRAYS IF AND F ARE DIMENSIONED BY THE USER TO SUIT HIS PROGRAM.       MAIN0003
      DIMENSION IF(1082),F(2183)                                         MAIN0004
      NI=1082                                                            MAIN0005
      NF=2183                                                            MAIN0006
C TRANSFER CONTROL TO THE BRANCH-AND-BOUND ALGORITHM.                    MAIN0007
      CALL ICON (IF,F,NI,NF)                                             MAIN0008
      CALL EXIT                                                          MAIN0009
      END                                                                MAIN0010

      SUBROUTINE READIN                                                  READ0001
C THIS SUBROUTINE IS NOT CALLED FOR A MIXED INTEGER LINEAR PROGRAM.      READ0002
      RETURN                                                             READ0003
      END                                                                READ0004

      SUBROUTINE GETOBJ (J,X,OBJ)                                        GET00001
C THIS SUBROUTINE IS NOT CALLED FOR A MIXED INTEGER LINEAR PROGRAM.      GET00002
      RETURN                                                             GET00003
      END                                                                GET00004
```

EXHIBIT 4

INPUT DATA FOR TEST PROGRAM 2

```
    5    4    1    1    0    0    0
    3    0    0
         90        .0001        .0001              0              0
TEST PROGRAM 2. MIXED INTEGER LINEAR PROGRAM
    4    0    0
    3    3    3    3    2
    1    2    2    3    3    4
    1    3    2    2    4    5
    1    1    2    2    3    1
    1    2    2    1    4    1
    1    2    2    2
    5
           1              2              3             -6             -7
          18             15              0              0
    0
    5
    1    2    3    4    5
           1              1              6              7              9
           0              0             -1             -1             -1
    5
    1    2    3    4    5
```

EXHIBIT 5

USER SUBROUTINES FOR TEST PROGRAM 3

```
      PROGRAM MAIN (INPUT,OUTPUT,TAPE5=INPUT,TAPE6=OUTPUT,                MAIN0001
     1              TAPE1,TAPE2=500,TAPE3=2000,TAPE4=0,                   MAIN0002
     2              TAPE7,TAPE8,TAPE9=0,TAPE10=0)                         MAIN0003
C ARRAYS IF AND F ARE DIMENSIONED BY THE USER TO SUIT HIS PROGRAM.        MAIN0004
      COMMON IF(1775),F(4360)                                             MAIN0005
      NI=1775                                                             MAIN0006
      NF=4360                                                             MAIN0007
C COMMON BUFFERS FOR TAPES 1 AND 4, 7 AND 9, 8 AND 10.                    MAIN0008
      CALL BUFSHAR (1,4)                                                  MAIN0009
      CALL BUFSHAR (7,9)                                                  MAIN0010
      CALL BUFSHAR (8,10)                                                 MAIN0011
C TRANSFER CONTROL TO THE BRANCH-AND-BOUND ALGORITHM.                     MAIN0012
      CALL ICON (IF,F,NI,NF)                                              MAIN0013
      CALL EXIT                                                           MAIN0014
      END                                                                 MAIN0015

      SUBROUTINE READIN                                                   READ0001
C QUADRATIC FUNCTION WITH SETUP COST AT THE ORIGIN.                       READ0002
      COMMON/INPUT/NCC,C1(8),C2(8),C3(8)                                  READ0003
      READ(5,1000)NCC                                                     READ0004
      WRITE(6,1002)NCC                                                    READ0005
      DO100J=1,NCC                                                        READ0006
      READ(5,1001)C1(J),C2(J),C3(J)                                       READ0007
  100 WRITE(6,1003)J,C1(J),C2(J),C3(J)                                    READ0008
      RETURN                                                              READ0009
 1000 FORMAT(I5)                                                          READ0010
 1001 FORMAT(3E12.0)                                                      READ0011
 1002 FORMAT(42H0USER INPUT FOR BRANCH-AND-BOUND ALGORITHM/                READ0012
     1       30H0NUMBER OF CONCAVE VARIABLES =,I5/                        READ0013
     2       1H0,3X,2HCC,14X,2HC1,15X,2HC2,15X,2HC3/1X,8HVARIABLE/        READ0014
     3       2X,6HNUMBER//)                                               READ0015
 1003 FORMAT(3X,I5,3X,3(E15.6,2X))                                        READ0016
      END                                                                 READ0017

      SUBROUTINE GETOBJ (J,X,OBJ)                                         GET00001
C QUADRATIC FUNCTION WITH SETUP COST AT THE ORIGIN.                       GET00002
      COMMON/P1/N,M,ITYPE,NSTRAT,NODRL1,NBVRL1,NTITE1,NODRL2,NBVRL2,      GET00003
     1          NTITE2,MXLIST,LISTOP,ITAPE,IFB,MXITER,MBINV,ICUTPT,       GET00004
     2          ITRACE,MSTART,TIME1,TOL1,TOL2,PCBUB,ALPHA(10)             GET00005
      COMMON/INPUT/NCC,C1(8),C2(8),C3(8)                                  GET00006
      IF(X.GT.TOL2)GOTO.00                                                GET00007
      OBJ=0.0                                                             GET00008
      RETURN                                                              GET00009
  100 OBJ=C1(J) + X*(C2(J) + X*C3(J))                                     GET00010
      RETURN                                                              GET00011
      END                                                                 GET00012
```

EXHIBIT 6

INPUT DATA FOR TEST PROGRAM 3

```
 24   31    0    1    1    1    0
  1    0    0
          90          .0001          .0001               0          25
TEST PROGRAM 3. INTEGER CONCAVE PROGRAM
   1    1   29
   5    6    6    8    9   20   20    3   17   14   18   14   19   19   13   11
   1    1    1    1    1    1    1    1
   5   19   11   83   15  120   16  132   24  160
   5   20   10   70   11   84   15  121   19  159   24  160
  10   71   14  110   15  122   16  133   19  159   24  160
   1  159    5   21    6   31    7   42    8   52   10   72   15  123   25  160
   1  159    5   22    6   32    7   43    8   53   10   73   15  124   16  134
  25  160
   2  159    3    1    4   11    5   23    6   33    7   44    8   54    9   62
  10   74   11   85   12   94   13  102   14  111   15  125   16  135   17  144
  18  151   20  160   23  159   27  160
   2  162    3    2    4   12    5   24    6   34    7   45    8   55    9   63
  10   75   11   86   12   95   13  103   14  112   15  126   16  136   17  145
  18  152   20  159   23  159   27  160
   2  159   23  159   27  160
   3    3    4   13    5   25    6   35    7   46    8   56    9   64   10   76
  11   87   12   96   15  127   18  153   21  159   22  160   23  159   26  160
  27  160
   3    4    4   14    5   26    6   36    7   47   10   77   11   88   14  113
  16  137   21  160   22  161   23  159   26  160   27  160
   3    5    4   15    5   27    6   37    7   48    8   57    9   65   10   78
  11   89   13  104   14  114   15  128   16  138   21  159   22  160   23  159
  26  160   27  160
   3    6    5   28    7   49    8   50    9   66   11   90   12   97   13  105
  14  115   15  129   16  139   17  140   18  154   31  160
   3    7    4   16    5   29    6   38    7   50    8   59    9   67   10   79
  11   91   12   98   13  106   14  116   15  130   16  140   17  147   18  155
  28  160   30  160   31  160
   3    8    4   17    5   30    6   39    7   51    8   60    9   68   10   80
  11   92   12   99   13  107   14  117   15  131   16  141   17  148   18  156
  29  160   30  160   31  160
   3    9    4   18    6   40    9   69   10   81   11   93   12  100   13  108
  14  118   16  142   17  149   18  157   31  160
   3   10    6   41    8   61   10   82   12  101   13  109   14  119   16  143
  17  150   18  158   31  160
  24  163
  25  164
  26  165
  27  166
  28  161
  29  167
  30  168
  31  169
 169
      -.0644        .9258        .1154        .3922        .6738        .0886
     1.0278        .2955        .0381        .0461       -.3344       1.0550
      .1612        .0154        .7411        .9476        .5430        .0502
     .00741       .00400       .11420       .06441       .00754       .38303
```

EXHIBIT 6 (Continued)

```
   .09924       .18658       .13567       .01190       .14200       .03058
  40.664       31.598       11.100       36.958       21.934        5.744
  16.584        9.329        2.932         .464        2.100         .2719
   .2865         .0927       1.1107        .2083        .2214         .9351
   .0733         .7313        .2530        .39174       .32166        .06931
  .45466        .27466       .28256       .02106       .31088        .07237
  .03359        -.251       26.755       12.250        9.163         4.687
  23.326        9.365        3.830         .073         .526         3.789
   1.891         .400       11.875        5.145       20.692         1.907
   2.138         .202         .580        1.495        .0515         .0225
   .1466         .7422        .0682       1.2945        .2847         .0224
   .8700         .1890        .0705        .0379        .4718         .5304
  1.1783        2.1150        .3723        .3664        .3969         .18941
  .10079        .21799       .93822       .57995       .07318        .22251
  .24297        .0742       -2.8200      2.4223        .9552         .5351
   .7351        1.1132        .2031        .6653        .6081         .1371
   .0933         .1588        .2622        .2643       -1.0364       2.0501
   .6843         .7933        .6234        .3276        .0837         .01521
  .01446        .10047       -.71932      .71301       .45116        .15437
  .40417        .41676       .12291       .34865       .40708       -.16689
  .26709        .15283       .86077       .19143       .35038        .22731
  .00601        .30637       .39258       .29269       .26108        .03371
  .14164        .12580          1            -1           4            -2
      19            3            9            18           12           16
      31
       3            0         3.9415       3.8398       1.09380      138.190
   3.8663       0.01841      77.595       29.070        2.6440        5.0324
  2.13960        7.1200       4.7010       0.03371      3.90961       1.96236
       2            0            0            0            3            0
       0            0            0            0            0            0
       0
1
1
       4
24
 1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16
17  18  19  20  21  22  23  24
      11            4            4            3            3            2
       3            4            4            2            3            3
       4           12            7            5            1            1
       1            1            1            1            1            1
    22.9         33.2         12.5         11.9         14.9         41.0
   260.3         82.0        101.4        167.1        138.9         54.9
    90.6         15.9         10.2          7.4            0            0
       0            0            0            0            0            0
16
 1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16
 8
17  18  19  20  21  22  23  24
 8
    30.2            0            0
    10.9            0            0
    83.3            0            0
     4.2            0            0
    62.5            0            0
    11.8            0            0
    57.6            0            0
     4.2            0            0
```

1. Claude Berge, The Theory of Graphs, John Wiley and Sons, Inc., New York, 1962.

2. R. J. Dakin, "A Tree-Search Algorithm for Mixed Integer Programming Problems," Computer Journal 8(1965)250-255.

3. George B. Dantzig, Linear Programming and Extensions, Princeton University Press, Princeton, New Jersey, 1963.

4. Ronald E. Davis, David A. Kendrick and Martin Weitzman, "A Branch-and-Bound Algorithm for Zero-One Mixed Integer Programming Problems," Operations Research 19(1971)1036-1044.

5. James E. Falk and Richard M. Soland, "An Algorithm for Separable Nonconvex Programming Problems," Management Science 15(1969)550-569.

6. James E. Falk and Richard M. Soland, Outline of the Short Course: Optimization by Branch-and-Bound, Research Analysis Corporation, McLean, Virginia, 14 April 1971.

7. Anthony V. Fiacco and Garth P. McCormick, Nonlinear Programming: Sequential Unconstrained Minimization Techniques, John Wiley and Sons, Inc., New York, 1968.

8. Robert S. Garfinkel and George L. Nemhauser, Integer Programming, John Wiley and Sons, Inc., New York, 1972.

9. Richard J. Giglio and Harvey M. Wagner, "Approximate Solution to the Three-Machine Scheduling Problem," Operations Research 12(1964)305-324.

10. John Haldi, 25 Integer Programming Test Problems (Working Paper No. 43), Graduate School of Business, Stanford University, December 1964.

11. A. H. Land and A. G. Doig, "An Automatic Method of Solving Discrete Programming Problems," Econometrica 28(1960)497-520.

12. Leon S. Lasdon, Optimization Theory for Large Systems, The MacMillan Company, London, 1970.

13. Harlan W. Loomis, Nonlinear Methods Used in the Fire Support Study (TR-2862), Naval Weapons Laboratory, Dahlgren, Virginia, April 1973.

14. Harlan W. Loomis, Concave Nonlinear Optimization Under Constraints with Integer Valued Variables (TR-3044), Naval Weapons Laboratory, Dahlgren, Virginia, November 1973.

15. G. Mitra, "Investigation of Some Branch and Bound Strategies for the Solution of Mixed Integer Linear Programs," Mathematical Programming 4(1973)155-170.

16. Hamdy A. Taha, "Concave Minimization Over a Convex Polyhedron," Naval Research Logistics Quarterly 20(1973)533-548.

17. J. A. Tomlin, "An Improved Branch-and-Bound Method for Integer Programming," Operations Research 19(1971)1070-1075.

18. Harvey M. Wagner, Principles of Operations Research with Applications to Managerial Decisions, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1969.