5-1971

U. S. U. Mathematical Programming Package

R. Gary Goodwin
Utah State University

Follow this and additional works at: https://digitalcommons.usu.edu/gradreports

Part of the Applied Statistics Commons

Recommended Citation
https://digitalcommons.usu.edu/gradreports/1140

This Report is brought to you for free and open access by the Graduate Studies at DigitalCommons@USU. It has been accepted for inclusion in All Graduate Plan B and other Reports by an authorized administrator of DigitalCommons@USU. For more information, please contact dylan.burns@usu.edu.
U. S. U. Mathematical Programming Package

by

R. Gary Goodwin

A report submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

Applied Statistics

Plan B

Approved:

Utah State University
Logan, Utah

1971
ACKNOWLEDGMENTS

I would like to thank Dr. Bartell Jensen, chairman of my graduate committee, for his encouragement as I endeavored to complete this report and for his suggestions concerning a report topic; Dr. Rex Hurst, Head of the Applied Statistics and Computer Science Department, for his helpful suggestions; and Mr. E.C. Eastman for his help in clearing up many difficult points. Finally, I would like to thank my wife, Barbara, for her patience and for the many hours of time spent typing.

Ronald Gary Goodwin
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>IBM MPS PACKAGE</td>
<td>3</td>
</tr>
<tr>
<td>Pertinent Literature</td>
<td>3</td>
</tr>
<tr>
<td>Examples Illustrating Use</td>
<td>4</td>
</tr>
<tr>
<td>QUADRATIC PROGRAMMING COMPUTER CODE</td>
<td>17</td>
</tr>
<tr>
<td>Method</td>
<td>17</td>
</tr>
<tr>
<td>Usage</td>
<td>20</td>
</tr>
<tr>
<td>Limitations</td>
<td>22</td>
</tr>
<tr>
<td>Sample Problem</td>
<td>22</td>
</tr>
<tr>
<td>SHARE MATHEMATICAL PROGRAMMING Routines</td>
<td>33</td>
</tr>
<tr>
<td>Mixed Integer Programming Routine</td>
<td>33</td>
</tr>
<tr>
<td>Zero-One Integer Programming Routine</td>
<td>41</td>
</tr>
<tr>
<td>Transportation Code</td>
<td>45</td>
</tr>
<tr>
<td>BIBLIOGRAPHY</td>
<td>55</td>
</tr>
<tr>
<td>APPENDIXES</td>
<td>57</td>
</tr>
<tr>
<td>Appendix A. Documentation of Mixed Integer Programming Routine</td>
<td>58</td>
</tr>
<tr>
<td>Appendix B. Documentation of Zero-One Integer Programming Routine</td>
<td>74</td>
</tr>
<tr>
<td>Appendix C. Documentation of Transportation Code</td>
<td>89</td>
</tr>
</tbody>
</table>
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Contents of Fat, Dry Matter and Water in Milk, Butter and Cheese</td>
<td>23</td>
</tr>
<tr>
<td>2. Number of passengers desiring transportation on various routes and the income obtained</td>
<td>34</td>
</tr>
<tr>
<td>3. Number of passengers (in hundreds) per year each plane can carry on the various routes</td>
<td>34</td>
</tr>
<tr>
<td>4. Costs of one air craft per year for a given route (in thousands of dollars)</td>
<td>34</td>
</tr>
<tr>
<td>5. Cost in dollars for producing products 1-5 at plants 1-3</td>
<td>47</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Control program illustrating how to override the XDONFS demand</td>
<td>7</td>
</tr>
<tr>
<td>2. Sample problem output from TRACE procedure</td>
<td>8</td>
</tr>
<tr>
<td>3. Program setup to save basis of optimal solution</td>
<td>10</td>
</tr>
<tr>
<td>4. Program setup to insert the basis of a previous problem run</td>
<td>12</td>
</tr>
<tr>
<td>5. Program setup to find the optimal solution of a constraint system with a nonlinear separable function nonlinear in more than one variable</td>
<td>15</td>
</tr>
<tr>
<td>6. Listing of data input for milk production problem</td>
<td>26</td>
</tr>
<tr>
<td>7. Illustration of placement of quadratic coefficients within matrix system</td>
<td>29</td>
</tr>
<tr>
<td>8. Solution output for milk production problem</td>
<td>32</td>
</tr>
<tr>
<td>9. Input data for aircraft routing problem</td>
<td>36</td>
</tr>
<tr>
<td>10. Diagram illustrating row order and column order adhered to, to define program input to aircraft routing problem</td>
<td>39</td>
</tr>
<tr>
<td>11. Program output from aircraft routing problem</td>
<td>40</td>
</tr>
<tr>
<td>12. Input data to depot problem</td>
<td>44</td>
</tr>
<tr>
<td>13. Program output from depot problem</td>
<td>46</td>
</tr>
<tr>
<td>14. Program setup for production allocation problem</td>
<td>49</td>
</tr>
<tr>
<td>15a. Summary output from product allocation problem</td>
<td>51</td>
</tr>
<tr>
<td>15b. Program output of checkpoint tape summary, optimal solution and outstanding requirements</td>
<td>52</td>
</tr>
<tr>
<td>15c. Program output of work allocation by product and by plant</td>
<td>53</td>
</tr>
</tbody>
</table>
INTRODUCTION

Solving mathematical programming problems without the aid of electronic computers is very time consuming even for a mathematical model consisting of only a few constraints and variables. The objective of this report is to illustrate the use of mathematical programming computer routines available at Utah State University. Computer routines discussed are the International Business Machines (IBM) Mathematical Programming System (MPS) package, a quadratic programming routine, a zero-one integer programming routine, and a transportation code. The MPS package is a product of IBM; the quadratic programming routine was developed by the author; the last three routines were obtained from the IBM SHARE library.

Source decks, 80-80 lists, and execution lists are available at the Department of Applied Statistics and Computer Science, the Department of Economics, and the Utah State Computer Center of Utah State University (USU) for all the computer programs indicated above excluding the MPS package. Each source deck, complete with job control language and sample problem input is ready to run on the computer with the addition of a valid job card. The 80-80 list for each program is an exact duplicate of its corresponding deck. The execution list shows the output that is to result upon the execution of a given program and sample input deck. Within this report a discussion is prepared for each sample problem. First the problem is defined and the mathematical model for solution of the problem is developed. Second, explanation is provided on how the input cards are prepared and finally an interpretation of the program output is included.
The MPS package is stored on disk at the U.S.U. Computer Center. Sample problems are provided in this report which illustrate how to use the MPS package. Also included in the report are references to where other sample problems may be found which illustrate the use of the MPS package.

Even though the Job Control Language (JCL) cards included in the illustrations vary with computer installation and with time, these are included so that the user will have an idea of the JCL needed to run the programs.

Various words within this report require a blank character. Since a blank character within a word would be confusing, the symbol "\" is used in place of the blank character.
IBM MPS PACKAGE

Due to the enormity of the MPS Package, directions concerning its use are extensive. The IBM manual (1969a) is quite complete in providing the information required. However, without examples illustrating its use considerable time and effort would be required to learn how to use it, but with proper examples many of the features of the MPS package can be understood without extensive study. Although IBM (1969a) has some examples illustrating the use of the MPS package, more examples are needed.

Pertinent Literature

IBM (1969a) illustrates the general use of the MPS package with a sample optimization problem consisting of a linear objective function and linear constraints. Separable programming is also illustrated with a sample problem. In particular, the sample problem illustrating separable programming also illustrates the use of the parametric programming procedures. In particular, the use of procedures PARAROW, PARARHS, and PARAOBJ are illustrated. PARAROW provides the capability of performing parametric programming on a specified row; PARARHS provides the capability of paramaterizing the right-hand-side (RHS), and PARAOBJ provides parameterization of the objective function.

Beneke and Winterboer (1970) and Freeman and Lard (1970) also illustrate the use of the MPS package with some sample problems. These authors have examples illustrating the solution of a standard problem, the use of the parametric procedures, and the preparation of the input data. Beneke and Winterboer (1970) also have examples illustrating the use of multiple right-hand-side vectors, the use of multiple objective functions, the use of multiple bounds, and the use of REVISE to make
unconditional revisions and also to make conditional revisions on the
input problem. REVISE is a procedure which allows the user to make
revisions in the constraint system after finding the initial optimal
solution. This is helpful since the revised problem may be solved
using the basis of the original optimal solution as the starting point.
Also Beneke and Winterboer (1970) have a sample problem using the RANGE
procedure. With this sample problem is a discussion interpreting the
meaning of the output from RANGE.

IBM (1969b) provides instructions concerning the job control lan-
guage needed in executing the MPS package as well as the syntax rules
for the MPS control language. The MPS control language is similar to
other programming languages such as PL/I and FORTRAN, but is specifi-
cally oriented to mathematical programming. This language is composed
of eight types of statements: procedure call, data definition, move-
ment, arithmetic, logical, program flow, macro definition, and delimit-
ing. Each of these statement types are illustrated in the IBM (1969b)
manual.

**Examples Illustrating Use**

The following are sample problems which illustrate uses of the MPS
package which have not been brought out in the manuals thus far discus-
sed. Example one illustrates overriding demands and the use of TRACE.
Example two illustrates how to save the basis of a problem and use it
in succeeding problems. Example three illustrates how to set up a sep-
arable programming problem in which the nonlinear separable functions
are nonlinear in more than one variable. The major purpose of these
sample problems are to bring out the finer points that must be known
to accomplish the above specified tasks. For sample problems in
which a discussion is included on how the control program and input
data are set up refer to IBM (1969a), Beneke and Winterboer (1970) or

Example One

In attempting to solve a system of linear simultaneous equations
which represent the constraints of an optimizing problem if the solu-
tion is not feasible the demand XDONFS is set. With this demand set
the normal course of action is to execute the procedures STATUS,
SOLUTION, and EXIT in that order. A procedure which is of use when the
XDONFS demand is set is TRACE. TRACE procedure may be executed. The
following sample problem illustrates how to override the usual course of
action when XDONFS demand is det and how to use TRACE in the analysis of
a nonfeasible solution.

A compound which is produced by blending three inputs must meet
the following specifications:
Specific Weight ≤ 1.0
Flash Point ≥ 500 F
Acid Content ≤ 1% (by volume)
Abrasive Content ≤ 10% (by volume)

The three inputs are characterized as follows:

<table>
<thead>
<tr>
<th>Property</th>
<th>Input 1(X₁)</th>
<th>Input 2(X₂)</th>
<th>Input 3(X₃)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specific Weight</td>
<td>.90</td>
<td>1.10</td>
<td>0.99</td>
</tr>
<tr>
<td>Flash Point (degrees F)</td>
<td>600</td>
<td>400</td>
<td>475</td>
</tr>
<tr>
<td>Acid Content (by volume)%</td>
<td>2</td>
<td>0.5</td>
<td>1.0</td>
</tr>
<tr>
<td>Abrasive Content (by volume)%</td>
<td>8</td>
<td>12</td>
<td>11.0</td>
</tr>
</tbody>
</table>

¹This problem was used for illustrative purposes in computer science
class, OR-245 taken at Utah State University.
Given that the mixed characteristics are a convex combination or the characteristics and the cost per gallon of each of the inputs is $2, $1, and $3 respectively minimize

\[ Z_{\text{MIN}} = 2X_1 + X_2 + 3X_3 \]

subject to the following constraints:

\[ .9X_1 + 1.1X_2 + .99X_3 \leq 1.0 \]  \hspace{1cm} (1)

\[ 600X_1 + 400X_2 + 475X_3 \geq 500 \]  \hspace{1cm} (2)

\[ 2X_1 + .5X_2 + X_3 \leq 1 \]  \hspace{1cm} (3)

\[ 8X_1 + 12X_2 + 11X_3 \leq 10 \]  \hspace{1cm} (4)

Figure 1 contains a listing of the control program. The statement

\text{MOVADR}(XDONFS,INFEAS)

causes control in the case of a nonfeasible solution to pass to the statement labeled INFEAS. This causes the execution of the procedures TRACE, and EXIT overriding the usual course of action taken when the XDONFS demand is set. Output from the procedure TRACE provides the simplex multipliers, Zj − Cj values, for all rows in which the absolute value of the simplex multiplier is greater than or equal to one. The simplex multipliers indicate the sensitivity of the corresponding restraints. Figure 2 contains a listing of the output from TRACE. The simplex multiplier (PI value) for acid is 200. The absolute value of this PI value is much larger than the PI value for any other constraint, indicating that the acid constraint is much more sensitive than the other constraints. By relaxing this constraint slightly a feasible solution results. This is obvious since by changing the right-hand-side value of constraint (3) to two, using 100% of input 1 becomes a feasible solution.
PROGRAM
INITIALZ
MOVE(XDATA,'BLEND')
MOVE(XP3NAME,'EXAMPLE')
CONVERT
BCDOUT
SETUP('MIN')
MOVE(XOBJ,'ZMIN')
MOVE(XDONFS,INFEAS)
PRIMAL
SOLUTION
EXIT

INFEAS TRACE
SOLUTION
EXIT
PEND

Figure 1. Control program illustrating how to override the XDONFS demand.
<table>
<thead>
<tr>
<th>PI VALUE</th>
<th>INPUT1</th>
<th>INPUT2</th>
<th>INPUT3</th>
<th>FIRSTP</th>
<th>FIRSTB</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZVIN</td>
<td>N</td>
<td>2.00000</td>
<td>1.00000</td>
<td>1.00000</td>
<td>NONE</td>
</tr>
<tr>
<td>SPRT</td>
<td>L</td>
<td>0.50000</td>
<td>1.10000</td>
<td>1.10000</td>
<td>0.50000</td>
</tr>
<tr>
<td>FLASH</td>
<td>G</td>
<td>1.00000</td>
<td>600.00000</td>
<td>400.00000</td>
<td>475.00000</td>
</tr>
<tr>
<td>ACID</td>
<td>L</td>
<td>200.00000</td>
<td>2.00000</td>
<td>1.50000</td>
<td>1.00000</td>
</tr>
<tr>
<td>ABRASIVE</td>
<td>L</td>
<td>24.00000</td>
<td>1.000000</td>
<td>12.00000</td>
<td>11.00000</td>
</tr>
</tbody>
</table>

Figure 2. Sample problem output from TRACE procedure.
**Example Two**

Given a lengthy problem, much computer time can be saved in succeeding runs by saving the basis of the optimal solution of the original constraints and inserting this basis in future runs. The following example taken from Hadley (1964, p. 285) is used to illustrate this. The problem as stated here has been slightly modified.

The initial problem is to maximize

\[ Z_{\text{MAX}} = 0.25X_1 + X_2 \]

subject to the constraints

\[ 0.5X_1 + X_2 \leq 1.75 \]
\[ X_1 + 0.3X_2 \leq 1.5 \]

(5)

Figure 3 lists the users program. This listing as can be seen includes the MPS control program, job control cards, and the input data. The statements of major interest are (1) the MPS control statement

PUNCH ('LIST')

and (2) the job control statement

//EXEC.SYSPUNCH DD UNIT=SYSCP

The PUNCH procedure punches the basis of the optimal solution. The option LIST provides a list of the punched output. The job control statement initiates the punch for punched output. The punch unit specified will vary at different computer installations.

To illustrate how to use the saved basis from a previous run, consider the following modification of the constraint system (5).

\[ 0.5X_1 + X_2 + X_3 = 1.75 \]
\[ X_1 + 0.3X_2 \leq 1.5 \]
\[ -0.5X_1 + X_3 \leq 0.75 \]

(6)
Job Card
// EXEC MPS360
//COMP. SYSTN DD *

PROGRAM
INITIAL
MOVE(YRNAME, 'EXAMPLE')
MOVE(YRDATA, 'INTEGER')
CONVTR('CHECK', 'SOMETHING')
RCOUT
SETUP('MAX')
MOVE(YORJ, 'ZMAX')
MOVE(YRHS, 'FIRST')
PRIMAL
SOLUTION
PUNCH('LIST')
EXIT
END

// EXEC. SYSTN DD UNIT: SYSTN
// EXEC. SYSTN DD *

NAME INTEGER

DOTS
N ZMAX
L R101
L R102

COLUMNS
v1 ZMAX .25 R101 .5
v1 ROW2 1.0
v2 ZMAX 1.0 R101 1.0
v2 ROW2 .3

RHS
FIRST R101 1.75 ROW2 1.5

ENDATA
/
/

Figure 3. Program setup to save basis of optimal solution.
Figure 4 illustrates the use of the basis of the optimal solution of (5) as a starting solution to (6). The statement

MOVE(XDATA,'INTEGER')

brings in the data set headed with the name INTEGER and procedure CONVERT reduces the data set to packed binary and files it as the problem to be solved.

MOVE(DATA,'INSERT1')

brings in the data set headed with the name INSERT1 and procedure INSERT causes this data set to be inserted as part of the initial basis.

Example Three

IBM Manual (1969a) discussed the use of the separable programming feature of the MPS package and also provides an example. Although the example is good it does not illustrate how to prepare the input for a system of constraints in which the nonlinear separable functions are nonlinear in more than one variable.

A sample problem from Hadley (1964, p. 149), maximizes

\[ z_{\text{MAX}} = 3x_1 + 2x_2 \]

subject to the restraints

\[ 4x_1^2 + x_2^2 < 16 \]

\[ x_1, x_2 > 0 \]  

(7)
Job Card
// EXEC MPS360
//COMP.SYSIN DD *

PROGRAM
INITIAL7
MOVE(XDATA,'INTEGER')
MOVE(XPNNAME,'EXAMPLE')
CONVERT('CHECK','SUMMARY')
PCDOUT
SETUP('MAX')
"MOVE(XC','ZMAX')
MOVE(XPHS,'FIRSTB')
MOVE(XDATA,'INSERT1')
INSERT
PRIMAL
SOLUTION
EXIT
PEND

//EXEC.SYSIN DD *

NAME       INTEGER
ROWS
N   ZMAX
E   ROW1
L   ROW2
L   ROW3

COLUMNS:
   X1   ZMAX   .25   ROW1   .5
   X1   ROW2   1.0   ROW3  -.5
   X2   ZMAX   1.0   ROW1   1.0
   X2   ROW2   .3
   X3   ROW1   1.0   ROW3  -1.0

RHS
FIRSTB   ROW1   1.75   ROW2   1.5
FIRSTB   ROW3  -.75

Figure 4. Program set up to insert the basis of a previous problem run.
ENDATA
NAME INSERT1
XL X2 ROW1
ENDATA

Figure 4, Continued
contains a separable function nonlinear in variables $X_1$ and $X_2$.

This system of equations is approximately equivalent to the following: Maximize

$$Z_{\text{MAX}} = 3X_1 + 2.0X_2$$

subject to the restraints

$$0.5X_{11} + 0.5X_{12} + 0.5X_{13} + 0.5X_{14} = X_1$$
$$X_{21} + X_{22} + X_{23} + X_{24} = X_2$$

(8)

$$(X_{11} + 3X_{12} + 5X_{13} + 7X_{14}) + (X_{21} + 3X_{22} + 5X_{23} + 7X_{24}) \leq 16$$

$X_1, X_2 \geq 0$

$0 \leq X_{11}, X_{12}, X_{13}, X_{14}, X_{21}, X_{22}, X_{23}, X_{24} \leq 1$

with the added restraint that if $X_{i,j+1} > 0$ then all $X_{i,k}$ ($k=1,2...j$) must be equal to one.

Figure 5 contains a list of the user's program to solve this problem. Note in the input data how each set of 'special variables' is set apart by the use of SEPORG cards (cards containing 'SEPORG'). The set of variables $X_{11}, X_{12}, X_{13},$ and $X_{14}$ are the 'special variables' related to $X_1$ and the set of variables $X_{21}, X_{22}, X_{23},$ and $X_{24}$ are the 'special variables' related to $X_2$. 
Job Card

// EXEC MPS360
// CMT. SYSTN DD *

PROGRAM
INITIALIZE
MOVE (YDATA, 'SEPARI')
MOVE (YPNAME, 'EXAMPLE')
CONVEXT ('CHECK', 'SUMMARY')
ECOUT
SETUP ('BOUND', 'SEBOUND', 'MAX')
MOVE (XOUT, 'ZMAX')
MOVE (XPLUS, 'BOUND')
PRIMAL
SOLUTION
EXIT
END

// EXIC SYSTN DD *

NAME. SEPAR
ROWS
N  ZMAX
L  ROW1
F  GRID1
E  GRID2
COLUMNS
X1  ZMAX  3.0  GRID1  -1.0
X2  ZMAX  2.0  GRID2  -1.0
EX1  'MARKER'  'SEPORG'
X11  ROW1  1.0  GRID1  .5
X12  ROW1  3.0  GRID1  .5
X13  ROW1  5.0  GRID1  .5
X14  ROW1  7.0  GRID1  .5
EX2  'MARKER'  'SEPORG'

Figure 5. Program setup to find the optimal solution of a constraint system with a nonlinear separable function nonlinear in more than one variable.
\begin{verbatim}
x21  ROW1   1.0  GRID2  1.0
x22  ROW1   3.0  GRID2  1.0
x23  ROW1   5.0  GRID2  1.0
x24  ROW1   7.0  GRID2  1.0
RHS
   POUND  ROW1   16.0
ENDATA
/
/

Figure 5. Continued
\end{verbatim}
QUADRATIC PROGRAMMING COMPUTER CODE

Method

This program uses a method developed by Wolfe (1959) and the two-phase simplex process. Wolfe's method solves the quadratic programming problem using the simplex algorithm by adding additional constraints to the original model and restricting the vectors that enter the basis.

Given the quadratic programming problem consisting of M rows, N columns, and K cost coefficients

\[ AX = b \]

maximize

\[ \text{ZMAX} = CX + X'FX. \]

Transform (7) into

\[ AX = b \]

\[ 2FX + A'Z - A'D + V + EU = -C' \]

\[ U \geq 0, \ X \geq 0 \]

maximize

\[ \text{ZMAX} = -\sum u_j \]

where \( E = \begin{bmatrix} \Delta_{j} \end{bmatrix} \) is a diagonal matrix whose elements are \( \Delta_{j} = \pm 1 \) and \( V \) is an N dimensions identity matrix. To determine the sign of \( \Delta_{j} \), let \( F_B \) contain the columns of \( F \) corresponding to the columns of \( A \) in \( B \) (the basis) and denote by \( f^j_B \) the jth row of \( F_B \). Then

\[ \Delta_{j} = \begin{cases} +1 \text{ if } -C_j - 2f^j_B X^B < 0 \\ -1 \text{ if } -C_j - 2f^j_B X^B > 0 \end{cases} \]

The simplex method is then used to reduce \( -\sum u_j \) to zero. Only one variation from the standard simplex procedure is introduced. If \( x_j > 0 \); then \( v_j \) is not allowed to enter the basis and vice versa.
For the system of equations as expressed in (9) the technical coefficients, quadratic coefficients and the negative of the cost coefficients are defined by the program input. Note that most likely some of the technical coefficients will be due to slack and surplus variables added to make all restraints appear as equalities. From these coefficients a restraint system in the form of (10) is constructed.

The new system of equations will consist of M+N rows and 2N+2M+K columns. The first N vectors are due to the original constraints; N vectors are due to the V matrix; 2M vectors are due to the Z and D vectors and K vectors are due to the U matrix. Row names 'ROWXX', where XX is a right adjusted integer such that $M+1 \leq XX \leq M+N$, are formed to define the N new constraints added. Also column names are formed to define the N+2M+K new columns added. The column names for the V-vectors are identical to the names of the corresponding X-vectors except for the fact that the column names for X-vectors start with 'X' and column names for V-vectors start with 'V'. The column names for the Z and D vectors are Z\Delta XX and DELXX where \Delta stands for a blank character and XX is as defined previously. The column names for the U-vectors are U\Delta XX. The values of XX start at one and go to M for the Z and D vectors and start at one and go to K for the U-vectors. How the coefficients for the V, Z, and D vectors are established is evident from constraint system (10). The coefficients for the U-vectors are equivalent to the negative of the corresponding cost coefficients. After the constraints are in the form of system (10), the right-hand-side value of each row is checked for a negative value. Any row in which the right-hand-side value is negative is multiplied through by a minus one.
The program uses the two-phase method. Initially artificial variables are generated for each row. In the first phase an attempt is made to arrive at a feasible solution by driving all the artificial variables out of the basis. Artificial variables are not allowed to remain in the basis, even at the zero level. After all the artificial variables are driven out of the basis, control is passed to the second phase which proceeds to find an optimal solution. The iterative process proceeds as follows. The variable to enter and the variable to leave the basis are determined. Then the activity level of each of the basis variables is determined for the new basis. The activity of the \( X \)-variables is multiplied by the activity of the corresponding \( V \)-variables. If \( X_jV_j \) is greater than or equal to .00001, then the variable which was originally determined to enter the basis is flagged and another variable is chosen to enter the basis. The new variable to enter the basis is determined using the general simplex techniques except no flagged variable may enter the basis. If there are no more vectors that qualify as candidates to enter the basis or if the program is in phase two of the simplex method and the value of the sum of the activity of the \( V \)-values is less than .00001, the program terminates. If the program terminates in phase one of the simplex method a message is written indicating the solution is infeasible. If the program terminates in phase two of the simplex method the value of the objective function and the activity of the program variables is printed out.
Usage

Multiple problems may be run with one loading of the program.

Each problem deck is made up as follows:

<table>
<thead>
<tr>
<th>Card Type</th>
<th>Number/problem</th>
<th>Card Columns</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>1</td>
<td>2-40</td>
<td>alphameric problem name</td>
</tr>
<tr>
<td>Row Identification Header</td>
<td>1</td>
<td>1-6</td>
<td>ROWID</td>
</tr>
<tr>
<td>Row Identification</td>
<td>Variable</td>
<td>14-18</td>
<td>row name</td>
</tr>
<tr>
<td>Matrix Header</td>
<td>1</td>
<td>1-6</td>
<td>MATRIX</td>
</tr>
<tr>
<td>Matrix Element (sorted by column)</td>
<td>Variable</td>
<td>8-12</td>
<td>column name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4-18</td>
<td>row name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>19</td>
<td>11-punch when value below is negative</td>
</tr>
<tr>
<td></td>
<td></td>
<td>20-30</td>
<td>element value including decimal</td>
</tr>
<tr>
<td>RHS Header</td>
<td>1</td>
<td>1-7</td>
<td>FIRSTAB</td>
</tr>
<tr>
<td>RHS Value</td>
<td>1</td>
<td>14-18</td>
<td>row name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>20-30</td>
<td>RHS Value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>31-40</td>
<td>scale factor for row (if used, decimal point must be included)</td>
</tr>
<tr>
<td>Problem Delimeter</td>
<td>1</td>
<td>1-3</td>
<td>EOF</td>
</tr>
</tbody>
</table>

2Row names "ROWXX" are reserved and must not be used by the user except where indicated in the future. XX is an integer value right adjusted in the field. The range of XX is 0+1-XX<4+1.

3All column names must be left justified and start with the character "X".
<table>
<thead>
<tr>
<th>Card Type</th>
<th>Number/problem</th>
<th>Card Columns</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quadratic Cost Coefficient</td>
<td>Variable</td>
<td>8-12</td>
<td>column name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>14-18</td>
<td>row name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>19-30</td>
<td>value</td>
</tr>
<tr>
<td>Header</td>
<td>1</td>
<td>1-4</td>
<td>COMPANY</td>
</tr>
<tr>
<td>Linear Cost Coefficient</td>
<td>Variable</td>
<td>14-18</td>
<td>row name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>20-30</td>
<td>the negative of the coefficient value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>31-40</td>
<td>Scale factor for row (if used, decimal point must be included)</td>
</tr>
<tr>
<td>Problem Delimiter</td>
<td>1</td>
<td>1-3</td>
<td>EOF</td>
</tr>
<tr>
<td>Print Control</td>
<td>1</td>
<td>5</td>
<td>0 or blank- problem matrix will not be printed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1- problem matrix will be printed</td>
</tr>
</tbody>
</table>

*These row names must be "ROWXX" where XX is an integer value right justified in the field and equivalent to 'M+j' where,

M is the number of constraints, and j is a number indicating the position of the column variable associated with the coefficient value.
Limitations

The program is dimensioned to solve a problem with as many as 100 rows and 150 columns. An equation to determine the number of rows and an equation to determine the number of columns that are required to solve a given problem were given previously. Also due to the automatic naming of the constraints added to (9), the program limits the size of (10) to 99 constraints and variables.

Sample Problem

Description

The sample problem that follows was taken from Boot (1964). The problem is concerned with the optimal use of milk in the Netherlands. Netherland farmers deliver their surplus milk to Landbouw Egalisatie Fonds (Agricultural Stabilization Fund) and the government pays the farmers a guaranteed price for their milk. "The Fund tries to get the maximum value for the milk by selling it for consumption directly, and by processing it into butter, cheese, condensed milk etc., and selling these products on the home and export market. The Fund desires to know how the quantity of milk available for use in the Netherlands...should be split up between milk for consumption, butter and cheese; and what the prices...should then be to maximize revenue or, what amounts of the same, to minimize the subsidy of the government to the farmers."

Milk is a mixture of water, fat, and other ingredients which we will call dry matter. Table 1 shows the composition of milk and milk products.
Table 1. Contents of Fat, Dry Matter and Water in Milk, Butter and Cheese (Boot, 1964, p. 153)

<table>
<thead>
<tr>
<th></th>
<th>Fat</th>
<th>Dry matter</th>
<th>Water</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Milk</td>
<td>3.8</td>
<td>8.6</td>
<td>87.6</td>
<td>100</td>
</tr>
<tr>
<td>Milk for Consumption</td>
<td>2.6</td>
<td>8.6</td>
<td>88.8</td>
<td>100</td>
</tr>
<tr>
<td>Butter</td>
<td>80.0</td>
<td>2.0</td>
<td>18.0</td>
<td>100</td>
</tr>
<tr>
<td>Fat Cheese</td>
<td>30.6</td>
<td>29.7</td>
<td>39.7</td>
<td>100</td>
</tr>
<tr>
<td>40+ Cheese</td>
<td>24.5</td>
<td>37.1</td>
<td>38.4</td>
<td>100</td>
</tr>
</tbody>
</table>

The amount of fat and dry matter produced during the year amounts to 119,000 tons and 251,000 tons respectively. This data results in the following two constraints on the production of milk, butter, and cheese. Writing $P_1$, $P_2$, $P_3$, and $P_4$ for the quantities in 1000 tons produced of milk, butter, fat cheese, and 40+ cheese, respectively, we have:

\[
.026 P_1 + .800 P_2 + .036 P_3 + .245 P_4 \leq 119
\]

\[
.086 P_1 + .020 P_2 + .297 P_3 + .371 P_4 \leq 251
\]

The demand curves for milk, butter, fat cheese and 40+ cheese were determined to be:

\[
P_1 = 1.2338 X_1 + 2139
\]

\[
P_2 = 0.0203 X_2 + 135
\]

\[
P_3 = 0.0136 X_3 + 0.0015 X_4 + 103
\]

\[
P_4 = 0.016 X_3 - 0.027 X_4 + 19
\]

where $X_1$, $X_2$, $X_3$, and $X_4$ are the prices in guilders per ton of milk, butter, fat cheese, and 40+ cheese respectively and the P-values are
the demands in units of 1000 tons. The linear demand functions are justified since the main concern is with small deviations from the equilibrium values.

To accommodate for the fact that large price rises are unsatisfactory from the social point of view, the following constraint is added:

\[ 0.0163 \, X_1 + 0.0003 \, X_2 + 0.0006 \, X_3 + 0.0002 \, X_4 = 10 \quad (13) \]

Since the constraints are to be formulated in terms of prices, equations (12) must be substituted into equations (11) giving the restraints:

\[-0.0321 \, X_1 - 0.0162 \, X_2 - 0.0038 \, X_3 - 0.0002 \, X_4 \leq -80.5 \]
\[-0.1061 \, X_1 - 0.0004 \, X_2 - 0.0034 \, X_3 - 0.0001 \, X_4 \leq 26.6 \quad (14) \]
\[ 1.2338 \, X_1 \leq 2139 \]
\[ 0.0203 \, X_2 \leq 135 \]
\[ 0.0136 \, X_3 - 0.0015 \, X_4 \leq 103 \]
\[ -0.0016 \, X_3 - 0.0027 \, X_4 \leq 19 \]

The function to be maximized, \( \Sigma X_i P_i \), can be given as a quadratic function in \( X_i \), using (12):

\[-1.2338 \, X_1^2 - 0.0203 \, X_2^2 - 0.0136 \, X_3^2 - 0.0027 \, X_4^2 + 0.0031 \, X_3 \, X_4 \]
\[+ 2139 \, X_1 + 135 \, X_2 + 103 \, X_3 + 19 \, X_4 \quad (15) \]

Thus our final model consists of equations (13), (14), and (15).

We desire to maximize (15) subject to the restraints of (13) and (14).
Data Input

All constraints must be in the form of equalities when input to the program. Adding slack and surplus variables, constraint system (14) becomes:

\[ +0.0321X_1 +0.0162X_2 +0.0038X_3 +0.0002X_4 -X_5 = 80.5 \]
\[ -0.1061X_1 -0.0004X_2 -0.0034X_3 -0.0006X_4 +X_6 = 26.6 \]  
\[ 1.2338X_1 \quad +X_7 = 2139 \]
\[ +X_8 = 135 \]
\[ 0.0136X_3 -0.015X_4 +X_9 = 103 \]
\[ -0.0016X_3 -0.0027X_4 +X_{10} = 19 \]

Figure 6 is a listing of the data input to the computer program. Input card MILK1 is the title card. The information punched in columns 2-42 of this card will appear at the beginning of the program output listing. Input card MILK2 is a header card. This card indicates that the "Row Identification" cards follow.

Input cards MILK3 to MILK7 are "Row Identification" cards. One "Row Identification" card is included for each equation of (16) except the second equation and for equation (13). A card does not need to be included for the second equation of (16) since this constraint does not need to be included in our mathematical model. This becomes obvious by observing the corresponding constraint of constraint set (14) and realizing that all X-variables must be greater than zero. The identifying name for the first equation of (16) is ROW1, for the third ROW3, etc. to ROW6 for the 6th equation. The identifying name for equation (13) is ROW7.

Input card MILK9 is a header card which indicates that the "Matrix Element" cards follow. Input cards MILK9 to MILK28 are "Matrix Element" cards. One card is included for each technical part of at least the
### OPTIMAL USE OF MILK IN THE NETHERLANDS

<table>
<thead>
<tr>
<th>ROW ID</th>
<th>MILK1</th>
<th>MILK2</th>
<th>MILK3</th>
<th>MILK4</th>
<th>MILK5</th>
<th>MILK6</th>
<th>MILK7</th>
<th>MILK8</th>
</tr>
</thead>
<tbody>
<tr>
<td>RO11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RO13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RO44</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RO45</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RO46</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RO47</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### MATRIX

<table>
<thead>
<tr>
<th>X1</th>
<th>RO31</th>
<th>.0521</th>
</tr>
</thead>
<tbody>
<tr>
<td>X1</td>
<td>RO33</td>
<td>1.2338</td>
</tr>
<tr>
<td>X1</td>
<td>RO47</td>
<td>.0163</td>
</tr>
<tr>
<td>X2</td>
<td>RO41</td>
<td>.0162</td>
</tr>
<tr>
<td>X2</td>
<td>RO44</td>
<td>.0205</td>
</tr>
<tr>
<td>X2</td>
<td>RO47</td>
<td>.0093</td>
</tr>
<tr>
<td>X3</td>
<td>RO41</td>
<td>.0038</td>
</tr>
<tr>
<td>X3</td>
<td>RO45</td>
<td>.0136</td>
</tr>
<tr>
<td>X3</td>
<td>RO46</td>
<td>-.0016</td>
</tr>
<tr>
<td>X3</td>
<td>RO47</td>
<td>.0006</td>
</tr>
<tr>
<td>X4</td>
<td>RO41</td>
<td>.0002</td>
</tr>
<tr>
<td>X4</td>
<td>RO45</td>
<td>-.0015</td>
</tr>
<tr>
<td>X4</td>
<td>RO46</td>
<td>.0027</td>
</tr>
<tr>
<td>X4</td>
<td>RO47</td>
<td>.0002</td>
</tr>
<tr>
<td>X5</td>
<td>RO41</td>
<td>-1.0</td>
</tr>
<tr>
<td>X7</td>
<td>RO43</td>
<td>1.0</td>
</tr>
<tr>
<td>X8</td>
<td>RO44</td>
<td>1.0</td>
</tr>
</tbody>
</table>

*Figure 6. Listing of data input for milk production problem.*
<table>
<thead>
<tr>
<th>X9</th>
<th>ROW5</th>
<th>1.0</th>
<th>MILK27</th>
</tr>
</thead>
<tbody>
<tr>
<td>X10</td>
<td>ROW6</td>
<td>1.0</td>
<td>MILK28</td>
</tr>
</tbody>
</table>

**First B**

| ROW1 | 80.5 | MILK29 |
| ROW3 | 2139.0 | .01 | MILK30 |
| ROW4 | 135.0 | .1 | MILK31 |
| ROW5 | 103.0 | .1 | MILK32 |
| ROW6 | 19.0 | MILK33 |
| ROW7 | 10.0 | MILK34 |

**ECF**

| X1  | ROA 7 -1.2338 | MILK35 |
| X2  | ROX 8 -.0203  | MILK36 |
| X3  | ROY 9 -.0156  | MILK37 |
| X4  | ROY 9 .00155  | MILK38 |
| X3  | ROW10 .00155  | MILK39 |
| X4  | ROW10 -.0027  | MILK40 |

**SOCO**

| ROW 7-2139.0 | .01 | MILK41 |
| ROW 8-135.0  | .1 | MILK42 |
| ROW 9-103.0  | .1 | MILK43 |
| ROW10-19.0   | MILK44 |

**EOF**

- blank card

Figure 6. Continued
coefficient. Input cards containing technical coefficients pertaining to the same variable must be grouped together. Notice all input cards pertaining to column $X_1$ are together, all cards pertaining to $X_2$ are together, etc.

Input card MILK29 is a header card indicating that the "RHS Value" cards follow. Input cards MILK30 to MILK35 are "RHS Value" cards. One card is included for each right-hand-side value. Scale factors are included for the 3rd, 4th, and 5th equations of (16). Input card MILK36 is a delimeter card indicating the end of the "RHS Value" cards.

Input cards MILK37 to MILK42 are "Quadratic Cost Coefficient" cards. The coefficients of the quadratic and cross product terms must be punched on these cards. The objective function for the sample problem is given in equation (15). Figure 7 illustrates the positioning of the cost coefficients for quadratic and cross product terms within the matrix system of the sample problem. Only the upper left hand portion of the matrix is shown. The first six rows contain the matrix coefficients corresponding to equations (16) and (13). These six equations contain ten variables; therefore row names ROW7 through ROW16 are reserved for the extra ten equations which are built internally. Since the cards containing the technical coefficients for $X_1$ are placed first in the "Matrix Element" cards, the cost coefficient corresponding to the quadratic term $(X_1)^2$ must be referenced by ROW7; Since the cards containing the technical coefficients for $X_2$ are placed second, the cost coefficient corresponding to the quadratic term $(X_2)^2$ must be referenced by ROW8; likewise the cost coefficients for $(X_3)^2$ and $(X_4)^2$ would be
<table>
<thead>
<tr>
<th>ROW1</th>
<th>X1</th>
<th>X2</th>
<th>X3</th>
<th>X4</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROW2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROW3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROW4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROW5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROW6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROW7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROW 7</td>
<td>(X1)²</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROW 8</td>
<td></td>
<td>(X2)²</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROW 9</td>
<td></td>
<td></td>
<td>(X₃)²</td>
<td>X₃X₄</td>
</tr>
<tr>
<td>ROW10</td>
<td></td>
<td></td>
<td>X₃X₄</td>
<td>(X₄)²</td>
</tr>
</tbody>
</table>

Figure 7. Illustration of placement of quadratic coefficients within matrix system. This represents only the upper left hand corner of the matrix.
referenced by ROWA9 and ROW10 respectively. The coefficient being referenced by ROWA9 and the other half by ROW10. Input card MILK43 is a header card which indicates that the "Linear Cost Coefficient" cards follow. Input cards MILK30 to MILK34 are "Linear Cost Coefficient" cards. These cards contain the cost coefficients corresponding to the linear terms in the objective function. The row names that must be used to reference the cost coefficients of the linear terms are determined in the same manner that the row names were determined for the coefficients of the quadratic and cross product terms; therefore the cost coefficient corresponding to the linear term $X_1$ would be referenced by the row name ROWA7; $X_2$ would be referenced by ROWA8, etc. Scale factors are included for the constraints developed in the program that are referenced by the row names ROWA7, ROWA8, and ROWA9.

Input card MILK48 is a delimiter card that marks the finish of the "Linear Cost Coefficient" cards. Input card MILK49 is the "Print Control" card. Since column 5 was left blank, the problem matrix will not be printed on the computer listing.

**Program Output**

Items included in the output are the problem name, a statistical summary of the number of rows etc., a listing of the problem restraints, an iteration log and an optimal solution report. As previously indicated (page 21 of this report) the listing of problem restraints may be suppressed by the user. When the problem restraints are listed they are shown after scaling has taken place. Coefficients for the first row stored internally are printed first including it's right-hand-side value, then for the second row etc. Each row is labeled
at the left hand side of the page. Column headings are also included. The row coefficients are listed twelve values per line. If more than one line is required to list the row coefficients, then the column headings for the additional lines are required. The column headings for these additional lines are listed immediately underneath the column headings for the first line of coefficients. A listing of the optimal solution report is shown in Figure 8. The variables of interest are $X_1$, $X_2$, $X_3$, and $X_4$. The price levels for $X_1$, $X_2$, $X_3$, and $X_4$ that provide the optimal solution are 418, 3467, 2736, and 2513 guilders per ton respectively. The optimal value of the objective function is 1135 million guilders. The price levels for milk, butter, fat cheese, and 40+ cheese ($X_1$, $X_2$, $X_3$, and $X_4$) as shown by Boot (1964, p. 161) with $K=0$ are 423, 3454, 2760, and 2988 guilders per ton respectively and the value of the objective function is 1143 million guilders. The major discrepancy in our results is in the price on 40+ cheese. Upon inserting the price activities indicated by Boot back into the restraint system it was found that these values violate the social restraint (11) by the amount of .184. Changing the social restraint to fit Boots indicated results, the price levels for $X_1$, $X_2$, $X_3$, and $X_4$ for the optimal solution were 427, 3440, 2767, and 2589 guilders per ton respectively and the value of the objective function was 1148 million guilders.
ITERATION 27 OBJ FN 0.0

<table>
<thead>
<tr>
<th>BASIS VAR</th>
<th>AMOUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>X3</td>
<td>2736.153800</td>
</tr>
<tr>
<td>X7</td>
<td>1624.123535</td>
</tr>
<tr>
<td>X8</td>
<td>64.603561</td>
</tr>
<tr>
<td>X9</td>
<td>69.557999</td>
</tr>
<tr>
<td>X10</td>
<td>15.592270</td>
</tr>
<tr>
<td>Z1</td>
<td>418.117676</td>
</tr>
<tr>
<td>X1</td>
<td>1676.677734</td>
</tr>
<tr>
<td>X2</td>
<td>3467.802734</td>
</tr>
<tr>
<td>DEL 6</td>
<td>71231.437500</td>
</tr>
<tr>
<td>X4</td>
<td>2513.166260</td>
</tr>
<tr>
<td>V5</td>
<td>1676.677979</td>
</tr>
<tr>
<td>V7</td>
<td>0.0</td>
</tr>
<tr>
<td>Z1 3</td>
<td>0.0</td>
</tr>
<tr>
<td>V9</td>
<td>0.0</td>
</tr>
<tr>
<td>V10</td>
<td>0.0</td>
</tr>
</tbody>
</table>

**The value of the Objective Function is**

1134711.00000

Figure B. Solution output from milk production problem.
SHARE MATHEMATICAL PROGRAMMING ROUTINES

Three mathematical programming computer routines were obtained from the IBM share library: one to solve the classical transportation problem, one to solve the mixed integer linear programming problem, and one to solve the zero-one integer programming problem. Following is a sample problem for each computer routine illustrating the preparation of the data input and the interpretation of program output. Program documentation for each of the programs is contained in the appendixes.

**Mixed Integer Programming Routine**

This routine uses a branch bound algorithm based on the Doig and Land (1960) method to solve pure or mixed integer programming problems. Shareshian (1969) suggests to the users of this program that they change the dimension statements in the program to correspond with each problem they solve with the program. To save on compile time the dimensions were set so that the program required slightly less than 102 K^5 bytes of core storage; then an object deck was prepared. Using these program dimensions, the maximum problem which can be run consists of 40 rows and 40 columns. The second dimension of the array SAVTAB is set at 242. This value is placed in columns 2-4 of the first data card.

**Sample Problem**

This sample problem is taken from Vajda (1962, p. 41-45). The sample problem has been changed so that any given plane can fly on only one route and the planes are allowed to fly with vacancies.

^K as used here is equivalent to 1024.
Tables 2, 3, and 4 provide the fictitious data for a year period needed to formulate the problem.

**Table 2. Number of passengers desiring transportation on various routes and the income obtained**

<table>
<thead>
<tr>
<th>Route</th>
<th>Numbers (in hundreds)</th>
<th>Income (per hundred passengers)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>320</td>
<td>$15,000</td>
</tr>
<tr>
<td>2</td>
<td>165</td>
<td>$15,000</td>
</tr>
<tr>
<td>3</td>
<td>190</td>
<td>$8,000</td>
</tr>
</tbody>
</table>

**Table 3. Number of passengers (in hundreds) per year each plane can carry on the various routes**

<table>
<thead>
<tr>
<th>Route</th>
<th>Type of Aircraft</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>18</td>
</tr>
<tr>
<td>3</td>
<td>0(^a)</td>
</tr>
<tr>
<td>Total Number Available</td>
<td>15</td>
</tr>
</tbody>
</table>

\(^a\) Aircraft of that type do not fly on that route.

**Table 4. Costs of one aircraft per year for a given route (In thousands of dollars)**

<table>
<thead>
<tr>
<th>Route</th>
<th>Type of Aircraft</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>3</td>
<td>(^a) (^b)</td>
</tr>
</tbody>
</table>

\(^b\) Aircraft of that type are not allowed on the route.
Consider the cost of turning a passenger away as equivalent to the
cost of a vacancy and the cost of these to be equivalent to the in-
come which would have been obtained had a passenger ridden. The ob-
jective is to minimize the cost of transporting the passengers.

Let $X_{ij}$ be the number of aircraft of type $j$ routed to route $i$. Let $X_i$ be the number of passengers turned away on route $i$ and $Y_i$ be the number of vacancies not filled on route $i$. The mathematical model becomes, minimize:

$$ZMIN = 12X_{11} + 13X_{12} + 12X_{21} + 13X_{22} + 15X_{23} + 11X_{32} + 14X_{33} + 15X_i + 15X_2 + 8X_3 + 15Y_1 + 15Y_2 + 8Y_3$$

subject to the constraints:

$$20X_{11} + 15X_{12} + X_i - Y_1 = 320$$
$$18X_{21} + 13X_{22} + 10X_{23} + X_2 - Y_2 = 165$$
$$14X_{32} + 8X_{33} + X_3 - Y_3 = 190$$
$$X_{11} + X_{21} \leq 15$$
$$X_{12} + X_{22} + X_{32} \leq 14$$
$$X_{23} + X_{33} \leq 18$$

All variables must be $\geq 0$ and variable $X_{11}$, $X_{12}$, $X_{21}$, $X_{22}$, $X_{23}$, $X_{32}$, $X_{33}$ must take on integer values.

Data Input Preparation

Figure 9 contains a listing of the input data for the sample problem. Shareshain (1969, section 3-c) pages 66-69 in the appendix of this report gives instructions for preparing the input data. The first input card DATA1 is the "SAVTAB Dimension Card." The value 242 is punched in columns 2-4. This card will remain the same for all problem runs as long as the dimension of SAVTAB remains unchanged in the
<table>
<thead>
<tr>
<th>Aircraft</th>
<th>12.0</th>
<th>13.0</th>
<th>13.0</th>
<th>15.0</th>
<th>15.0</th>
<th>15.0</th>
<th>15.0</th>
<th>14.0</th>
<th>18.0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td></td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>

*Figure 9. Input data for aircraft routing problem.*
program. Input card DATA2 is the "Number of Problems Card." Since only one problem is to be solved a "1" is punched in column four. Input card DATA3 is the "Problem Identification Card." A "1" was punched in column one so that the program output would start at the top of the page. Columns 2–52 may contain any information which the user desires to put to identify his problem. Input card DATA4 is the "Optional Output Card." Since printout was not desired of the initial tableau or the tableau of the final continuous solution, this card was left blank. Input card DATA5 is the "Objective Function Upper Bound Card." The value of 906 is placed in columns one to ten. A decimal point is included, therefore, alignment of the number is not necessary. The upper bound was determined by finding an arbitrary feasible solution to the constraint system. Note that a feasible solution is to allocate 15 type 1 airplanes to route 1, five type 2 and 10 type 3 airplanes to route 2, and eight type 3 airplanes to route 3. This solution will mean that 20 passengers are turned away on route 1, thus giving a value of 906 for the objective function. Input card DATA6 is the "Problem Dimensions Card." Referring back to the equation (17) defining the sample problem, we note that there is the objective function and six constraint equations: therefore the sample problem contains seven rows. Also, it may be noted that the system of equations (17) contain 13 variables and one right-hand-side vector; therefore, the sample problem contains 14 columns. Also note that seven of the variables must be integer. Input cards DATA7 and DATA8 are the "Variables Upper Bound Card(s)." The upper bounds for the integer variables were determined for each variable by setting all other variables to zero and solving for that particular variable. For example, an
upper bound for $X_{11}$ can be found by setting $X_{21}$ to zero in the forth
constraint equation of (17) and solving for $X_{11}$. One field must be al-
located for each variable whether an upper bound is specified or not.
The second card DATA8 is blank since no upper bounds need to be
specified for the continuous variables. Input card DATA9 is the
"Constraint Type Card(s)." Ordering the constraint equations (17) from
top to bottom, the first three constraint equations are equality type
constraints and the last three constraint equations are less than or
equal type constraints; therefore, the first three fields are left
blank and the next three fields contain "-1". Input card DATA10 is
the "Right-Hand-Side Card(s)." Input card DATA11 is the "Matrix Type
Card." A "1" is punched in column four since the "Matrix Data Cards"
are punched in "Packed Format." Input cards DATA12 to DATA19 are the
"Matrix Data Cards." Figure 10 illustrates the row and column order
used to prepare these cards.

Consistency is a must in preparing the bound, constraint, right-
hand-side, and matrix cards. The constraint that is referenced n_th
with respect to constraint type ($\leq$, $\geq$, or =) must also be referenced
n_th with respect to the right-hand-side values, and must be the (n+1)_th
row referenced (the objective function being considered as the first
row in the matrix system) in the matrix input. If the upper bound of
a given variable is implied (by it's position on the bounds input cards)
to be in a given column, then the coefficients of that variable for the
objective function and for the succeeding rows must be stated or im-
plied by their position on the matrix input cards to be in that column.

Sample Problem Output

Figure 11 contains a listing of the program output. The initial
tableau and/or the final tableau may be printed out by providing the
Figure 10. Diagram illustrating row order and column order adhered to, to define program input to aircraft routing problem.
CONTINUOUS SOLUTION

OBJECTIVE FUNCTION 610.5000000 AT ITERATION 7

14.99999999 1.33333330 0.0 0.0 15.5000000 12.6666666 1.4999999 0.0
0.0 2.6666666 0.0 0.0 0.0

TOLERANCE SET AT 999.99999999 AT ITERATION 7

OBJECTIVE FUNCTION 879.0000000 AT ITERATION 13

15.0000000 2.0000000 0.0 0.0 16.0000000 12.0000000 2.0000000 0.0
5.0000000 5.9999999 10.0000000 0.0 0.0

OBJECTIVE FUNCTION 790.9999948 AT ITERATION 16

15.0000000 1.0000000 0.0 0.0 16.0000000 12.0000000 2.0000000 4.9999952
5.0000000 5.9999999 0.0 0.0 0.0

OPTIMALITY ESTABLISHED AT ITERATION 67

END OF PROBLEM

Figure 11. Program output from aircraft routing problem.
proper parameters on the "Optional Output Card." The objective function along with the activity of the variables is printed out when a continuous linear programming solution is reached, when the first mixed integer solution is reached, and subsequently whenever a better mixed integer solution is reached. The activity level of each variable is printed in the same order that the variables assumed in the program input: that is, the activity level for \( X_{11} \) is listed first the activity level for \( X_{12} \) is listed second etc. The last objective function value and activity values listed are the appropriate values for the optimal solution. This is true for all problems except for the case when the problem has no feasible solution. When there is no feasible solution this fact is plainly shown on the program listing.

Zero-One Integer Programming Routine (DZIP1)

DZIP1 solves the linear programming problem with the added restriction that the variables may take on only values of zero or one. The program is developed to minimize the objective function. A direct tree search method is employed to arrive at the optimal solution.

Sample Problem

Five towns are connected by a network of roads in the following way:

Town A is connected with B, C, and E
Town B is connected with A and C
Town C is connected with A, B, and D
Town D is connected with C and E

Each town should be served by a depot situated either in it or in a town which is connected with it by a road. Where should the depots be located if it is required that the number of depots be minimal? Vajda (1962, p. 95)

Stating the problem mathematically, we wish to minimize the objective function:
ZMIN = A + B + C + D + E \tag{18}

subject to the constraints

\begin{align*}
A + B + C & + E \geq 1 \\
A + B + C & \geq 1 \\
A + B + C + D & \geq 1 \\
C + D + E & \geq 1 \\
A + D + E & \geq 1
\end{align*} \tag{19}

with the added restriction that all the variables must take on the value of zero or one. Since all variables are restricted to zero or one, the first constraint of (19) will force at least one depot to be built such that town A is serviced by a depot. The second constraint will see that town B is serviced by a depot etc. to town E.

**Sample Problem Input**

DZIP1 requires the constraint equations to be expressed in the form

\[ A y \leq b \]

where \( A \) is the matrix containing the technical coefficients, \( y \) is the vector of variables, and \( b \) is the right-hand-side vector. Therefore all the constraints of (19) must be changed to \( \leq \) type constraints by multiplying them by -1 giving

\begin{align*}
- A - B - C & \leq -1 \\
- A - B - C & \leq -1 \\
- A - B - C - D & \leq -1 \\
- C - D - E & \leq -1 \\
- A & \leq -1 \\
- D & \leq -1 \\
- E & \leq -1
\end{align*} \tag{20}
Figure 12 is a listing of the input data for the depot problem Section 5 and 6 or Lemke and Snierberg (1968) pages 85-86 of the appendix of this report provide instructions for preparing the input data. Except for KTOL, KINCP, and KAPUB the reason for the parameters on the first two cards are easily explained by these sections. The parameter KTOL allows a tolerance limit to be specified on the objective function. The optimal value arrived at by DZIP1 will be within the specified tolerance of the true optimal solution. The field for the value of KTOL was left blank; therefore the program will continue until the exact optimal solution is reached. It is recommended that Gomory cuts be used (that KINCP be equal to one). According to Lemke and Snierberg (1968) this procedure reduces the number of possible solutions that need to be considered by a factor of three to ten. If the user does not wish to determine an upper bound (KAPUB) for the objective function, he may leave the card field allocated for this parameter blank and the program will set the upper bound. The upper bound set will be

\[ E_j C_j + 1 \]

where \( C_j \) is the \( j \text{th} \) cost coefficient of the objective function.

On input card DEPOT3, \( KJR(1) \) is set equal to one allowing for the print-out of the input data. Since no intermittent print-out of the input data is desired of allowed data card DEPOT4 is left blank. Input card DEPOT5 contains the cost coefficients. Note from equation (18) that the cost coefficient is equal to one for each variable.

Lemke and Snierberg (1968, p. 18) in explaining how to get intermittent print-out state "This option is only available for \( KJR(1) \) through \( KJR(4) \)," but it should have read \( KJR(2) \) instead of \( KJR(1) \).
Figure 12. Input data to depot problem.
Input card DEPOT6 contains the right-hand-side values. Note from constraint equations (20) that all the right-hand-side values are "-1". The technical coefficients are contained on input cards DEPOT7 to DEPOT11. One card is included for each constraint. Notice that the forth field is left blank on input card DATA7. This is due to the fact that the variable D does not appear in the first constraint equation of (20). Likewise on the other input cards DEPOT8 to DEPOT11 fields are left blank where the coefficient for the variable is zero.

Sample Problem Output

Figure 13 is a listing of the program output. The output indicates that nine iterations were required to insure that the optimal solution had been reached, but that the solution was printed at the third iteration, we see that the optimal value of the objective function is two and that the "non-zero components of X" are one and three. This implies that the minimum number of depots that need to be built to service towns A, B, C, D, and E is two and that the minimum requirement can be met by building a depot in towns A and C.

Transportation Code

According to Kahan (1968) this routine uses the stepping stone method to solve the classical transportation problem. The program is designed to deal with all cases of assigning resources to requirements, even when the total resources and requirements are unequal, and further to restart a partially completed run from a checkpoint record on tape.

Sample Problem

Hillier and Liberman (1968) discuss a sample problem which can be formulated as a transportation problem. The problem is as follows:

A company has excess production time at three of its plants. To
**START OF CLEANUP AT ITERATION NO. 1**

**OUTPUT OF PROBLEM VARIABLES**

**SUBOPTIMAL FEASIBLE SOLUTION AFTER ITERATIVE STEP NO.**

**OBJECTIVE FUNCTION:**

**NON-ZERO COMPONENTS OF X:**

**OPTIMAL SOLUTION ASCERTAINED AFTER ITERATIVE STEP NO.**

**SOLUTION WAS PRINTED AFTER ITERATIVE STEP NO.**

**NUMBER OF POINTS INVESTIGATED:**

**FRACTION OF POINTS INVESTIGATED**

**NO. OF TIMES LOWER BOUNDS, INCREMENTS EFFECTIVE:**

**NO. OF TIMES PREFERRED ROW SELECTED:**

**INFEASIBILITY BACKUPS AND CANCELLATIONS**

**FIGURE 13.** Program output from test problem.
take up this time the company decides to produce five new products. The production effort to produce one unit of product is essentially the same for all five products. The excess production capacity is 40, 60, and 70 items per unit of time at plants 1, 2, and 3 respectively. Marketing research has determined that the consumer demand will be 30, 40, 50, 40, and 60 items per unit of time for products 1, 2, 3, 4, and 5 respectively. Due to special equipment needs plant 3 is unable to produce product 5. The cost for producing each of the products at each of the plants is shown in Table 5.

Table 5. Cost in dollars for producing products 1-5 at plants 1-3

<table>
<thead>
<tr>
<th>Plants (source)</th>
<th>Products (Sinks)</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>20</td>
<td>19</td>
<td>14</td>
<td>21</td>
<td>16</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>15</td>
<td>20</td>
<td>13</td>
<td>19</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>18</td>
<td>15</td>
<td>18</td>
<td>20</td>
<td>MC</td>
</tr>
</tbody>
</table>

MC is a value sufficiently large to insure that product 5 will not be produced at plant 3.

The objective is to allocate production of the five products to the three plants in a manner to minimize cost.

This problem can be formulated as a transportation problem by referring to the plants as the sources, the products as the destinations or sinks, and the product costs as the transportation costs. To test the capabilities of the transportation code, values for the production capacity and consumer demand were chosen which would cause a degenerate condition.
Sample Problem Input

The transportation code requires that the input data be stored on a master file for program execution. Figure 14 shows the job control cards and input cards to execute the sample problem. Unit 12 is used as a scratch file and is utilized by the routine (TOTAPE) that generates the master file and by the routine (BKAC) which solves the transportation problem. The master file is generated on unit 13 in job step one and passed on to job step two. Unit 14 is used to generate the checkpoint tape. For a large problem where the completed solution may not result on the first computer run, this data set should be kept for future computer runs. To keep the checkpoint data set a disposition of "NEW,KEEP" must be indicated on data definition (DD) card FT14F001.

The following equations will be helpful in determining the block sizes to specify on the DD cards. The minimum block size for unit 13 must be the greater of 20 + 4(M+N) and 12 + 8(LBLOCK) where M is equal to the number of sources, N is equal to the number of sinks and LBLOCK is equal to the number of cost matrix entries stored per record. The value of LBLOCK is arbitrary; it is mainly up to the taste of the user. The main idea when determining block size is to keep the number of records to a minimum and the amount of core used within the bounds of computer storage. The minimum blocksize for units 12 and 14 are 16 + 8(LBLOCK) and 20 + 16(M+N+1) + 8(N+1) respectively.

Kahan (1968) pages 112-113 in the appendix of this report gives instructions for preparing the input data used to generate the master file. MSPREP1 to MSPREP8 of Figure 14 are the input cards to TOTAPE.
// Standard OS job card
// TOTAPE EXEC PRTCLG
// ICDP.SYSIN DD *

Object deck (TOTAPE) to generate master file.

// GO.FT12FOOl DD UNIT=2314,DSN=FILE,DISP=(,PASS),SPACE=(CYL,(1,1)), 1
// DCR=(RECFM=VS,PKSIZE=250)
// GO.FT13FOOl DD UNIT=2314,DSN=TESTDATA,DISP=(,PASS),SPACE=(CYL,(1,1)),1
// DCR=(RECFM=VS,PKSIZE=500)
// GO.SYSIN DD *

    3 5 15
    40 60 70
    30 40 50
    20 1 1 19 1 2 14 1 3 21 1 4
    16 1 5 15 2 1 20 2 2 13 2 2
    19 2 4 16 2 5 18 3 1 15 3 2
    18 3 3 20 3 4 99999 3 5

// BKACE EXEC PRTCLG
// ICDP.SYSIN DD *

Object deck (BKACE) to solve transportation problem.

// GO.FT12FOOl DD UNIT=2314,DSN=FILE,DISP=(,PASS)
// GO.FT13FOOl DD UNIT=2314,DSN=TESTDATA,DISP=OLD,
// DCR=(RECFM=VS,PKSIZE=500)
// GO.FT14FOOl DD UNIT=2314,SPACE=(CYL,(1,1)),DCR=(RECFM=VS,PKSIZE=250)
// GO.SYSIN DD *

99999 1 CONTROL

*CFE PLANT PRODUCTS 999999
/
/*

Figure 14. Program setup for production allocation problem.
for the given sample problem. All input data here discussed must be	right justified in its field. Input card MSPREP1 is the "Control
Card." Recall that the sample problem consisted of three sources and
five sinks. Since the sample problem is small the cost matrix entries
(15 in all) were stored in one record. Input card MSPREP2 is the
"Source Data Cards." For the sample problem excess production capacity
at each of the plants are the sources. Input card MSPREP3 is the
"Sink Data Cards." Consumers demand for a product represents a sink.
Input cards MSPRSP to MSPREP7 contain the "Cost Data Cards." The
cost matrix entries are shown in Table 5. The cost matrix entry for
source 3 sink 5 on input card MSPRSP7 is set to an arbitrary large
number 99999 to insure that product 5 is not produced at plant 3. In-
put card MSPREP9 is the "End-of-file Card." Kahan (1968) page 97
in the appendix of this report provides the instructions for preparing
the input data to BKAC. Since the work file is generated in job step
one and the sample problem is to be solved from scratch, column one
of input card CONTROL contains '3'. Columns 6-8 were left blank since
no checkpoint tape from a previous run was used. A '1' is placed in
column 9 of card CONTROL because we desire optimization to take place
even though the resources of the problem are not sufficient to satisfy
the requirements. Columns 25-26 of card HEADTM contain '60'; sixty
lines of output per page will leave a margin of three spaces at the top
and bottom of the page.

Sample Problem Output

Figures 15a, 15b, and 15c contain the program output from the
sample problem. Notice in Figure 15b that the hash cost for the last
iteration is significantly different from the optimal solution of
$2590. This difference is due to the fact that there are insufficient
<table>
<thead>
<tr>
<th>PROBLEM SUMMARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUMBER OF PLANTS</td>
</tr>
<tr>
<td>NUMBER OF PRODUCTS</td>
</tr>
<tr>
<td>NUMBER OF RECORDS OF COSTS DATA</td>
</tr>
<tr>
<td>TOTAL RESOURCES</td>
</tr>
<tr>
<td>TOTAL REQUIREMENTS</td>
</tr>
<tr>
<td>UNUSED RESOURCES</td>
</tr>
</tbody>
</table>

Figure 15a. Summary output from product allocation problem.
CHECKPOINT RECORDS

<table>
<thead>
<tr>
<th>CHECKPOINT NUMBER</th>
<th>COMPLETER SCANS</th>
<th>ITERATION COUNT</th>
<th>HASH COST</th>
<th>SOME REQUIREMENTS ARE NOT SATISFIED</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0.51002864E+08</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>5</td>
<td>0.50002496E+08</td>
<td></td>
</tr>
</tbody>
</table>

OPTIMIZATION HAS BEEN COMPLETED

HOWEVER, IT HAS NOT BEEN POSSIBLE TO SATISFY ALL SPECIFIED REQUIREMENTS

THE COST OF THE LISTED ALLOCATIONS AFTER 5 ITERATIONS IS $2590

THE FOLLOWING REQUIREMENTS ARE OUTSTANDING

<table>
<thead>
<tr>
<th>PRODUCT</th>
<th>ORDER</th>
<th>DEFICIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>5</td>
<td>60</td>
<td>10</td>
</tr>
<tr>
<td>TOTAL</td>
<td></td>
<td>50</td>
</tr>
</tbody>
</table>

Figure 15b. Program output of checkpoint tape summary, optimal solution and outstanding requirements.
### The Allocation of Work

#### By Product

<table>
<thead>
<tr>
<th>PRODUCT</th>
<th>ORDER</th>
<th>DEFICIT</th>
<th>PLANT</th>
<th>AMOUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>30</td>
<td>0</td>
<td>3</td>
<td>30</td>
</tr>
<tr>
<td>2</td>
<td>40</td>
<td>0</td>
<td>3</td>
<td>40</td>
</tr>
<tr>
<td>3</td>
<td>50</td>
<td>0</td>
<td>2</td>
<td>50</td>
</tr>
<tr>
<td>4</td>
<td>40</td>
<td>40</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>60</td>
<td>10</td>
<td>1</td>
<td>40</td>
</tr>
</tbody>
</table>

#### By Plant

<table>
<thead>
<tr>
<th>PLANT</th>
<th>TOTAL</th>
<th>SURPLUS</th>
<th>PRODUCT</th>
<th>AMOUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>40</td>
<td>0</td>
<td>5</td>
<td>40</td>
</tr>
<tr>
<td>2</td>
<td>60</td>
<td>0</td>
<td>3</td>
<td>50</td>
</tr>
<tr>
<td>3</td>
<td>70</td>
<td>0</td>
<td>1</td>
<td>30</td>
</tr>
</tbody>
</table>

*Figure 15c. Program output of work allocation by product and by plant.*
products produced to fill the demand. The transportation algorithm requires that supply equal demand. Since supply was less than demand fictitious supplies were generated to compensate. Figure 15b indicates that supply was deficit by 40 items of product 4 and 10 items of product 5. Recall that the fictious cost for fictious allocations was $999999; therefore the fictious cost is $49999950. Adding to this cost the real production cost of $2590 we get $50002496. The difference in the answers is due to the fact that accuracy of single precision arithmetic in the IBM-360 is only to seven significant digits. Figure 15c shows the allocation of work by plant and also by product. The optimal allocation of products to plants is for plant 1 to produce 40 items of product 5, plant 2 to produce 50 items of product 3 and 10 items of product 5, and plant 3 to produce 30 items of product 1 and 40 items of product 2. Hillier and Lieberman (1968, p. 193) verify this solution.
BIBLIOGRAPHY

In order to make the literature cited more obtainable, included in parenthesis at the end of each of the bibliography entries is information stating where the written work may be found. IBM manuals may be ordered through the secretary of the USU Computer Science Department. Concerning other written works below, if a note is not included indicating explicitly where it can be found then the publisher is the nearest source or else the written work is included in the appendix of this work.


APPENDIXES
Appendix A

Documentation of Mixed Integer Programming Routine

BRANCH AND BOUND MIXED INTEGER PROGRAMMING (BBMIP)

by

R. Shareshian

November 1969
Branch and Bound Mixed Integer Programming

BBMIP

TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td></td>
<td>60</td>
</tr>
<tr>
<td>Deck Key</td>
<td></td>
<td>61</td>
</tr>
<tr>
<td>User Information</td>
<td></td>
<td>61</td>
</tr>
<tr>
<td>Section 1</td>
<td>The Algorithm</td>
<td>61</td>
</tr>
<tr>
<td>Section 2</td>
<td>The Program</td>
<td>63</td>
</tr>
<tr>
<td>Section 3</td>
<td>User's Guide</td>
<td>65</td>
</tr>
<tr>
<td></td>
<td>a) The Program Deck</td>
<td>65</td>
</tr>
<tr>
<td></td>
<td>b) Optimal Use of Available Memory</td>
<td>65</td>
</tr>
<tr>
<td></td>
<td>c) Program Input</td>
<td>66</td>
</tr>
<tr>
<td></td>
<td>d) Program Output</td>
<td>69</td>
</tr>
<tr>
<td></td>
<td>e) Executing a Machine Run</td>
<td>71</td>
</tr>
<tr>
<td></td>
<td>f) Program Limitations</td>
<td>72</td>
</tr>
<tr>
<td>Section 4</td>
<td>References</td>
<td>73</td>
</tr>
</tbody>
</table>
Branch and Bound Mixed Integer Programming

BRMIP

ABSTRACT

The program employs a branch and bound algorithm based upon the Land and Diog method to solve mixed integer programming problems of limited size. However, the program may also be used to solve pure integer problems. The linear programming minimization problem is first solved without regard to integrality constraints. From this point the program proceeds as if to enumerate the set of all possible mixed integer solutions by constraining each integer variable singly and in turn to an integer value within its range. A dual simplex LP algorithm is used as a bound-establishing mechanism immediately after each integer variable is constrained. Large sub-sets of possible solutions, corresponding to continuations of partial sequence of integer-constrained integer variables, may be eliminated from consideration once it has been demonstrated that they must be inferior to the "best" feasible solution obtained to that point. When the total set has been exhausted, the best feasible solution is optimal.

The program was written in the Fortran IV Language for the IBM 360 under OS/360. Problem size is limited by the amount of core memory available. The LP routine is double precision.
Branch and Bound Mixed Integer Programming

BBMIP

USER INFORMATION

Section 1: The Algorithm

The mixed integer program will solve the following type of problem:

\[
\begin{align*}
\min \; & \mathbf{c}^T \mathbf{x} + \mathbf{\bar{c}}^T \mathbf{y} \\
\text{subject to} \; & \mathbf{A} \mathbf{x} + \mathbf{\bar{A}} \mathbf{y} \leq \mathbf{b} \\
& \mathbf{z}_i \text{ integral} \\
& \mathbf{z}_i, \mathbf{y}_i \geq 0
\end{align*}
\]

where \( \mathbf{c} \) and \( \mathbf{x} \) are vectors of \( n_1 \) components, \( \mathbf{\bar{c}} \) and \( \mathbf{y} \) are vectors of \( n - n_1 \) components, and \( \mathbf{b} \) is a vector of \( m \) components. \( \mathbf{A} \) is an \( m \times n_1 \) matrix and \( \mathbf{\bar{A}} \) is an \( m \times (n - n_1) \) matrix. Although not explicitly stated, it is required that each of the \( z_i \) have an upper bound, establishing the range of the variable.

The algorithm is based upon the Land and Doig method. A discussion of the algorithm follows.

Conceivably, one could enumerate all possible solutions to the mixed integer problem. Simultaneously constraining each of the \( z_i \) to an integer value within its range would yield a linear program whose solution would be a candidate for the optimal mixed integer solution. If the upper bound on each of the \( z_i \) is \( u_i \) (\( u_i \) integral), then the total number of such problems would be \( \sum u_i(u_i + 1) \), and an optimal mixed integer solution would be that one with lowest associated objective function. Unless \( n_1 \) and the \( u_i \) are so small as to make the original problem trivial, such an approach would be prohibitive and one seeks ways to avoid considering some of these resultant problems. Presumably, some of the problems do not have feasible solutions because of the integer value assigned to the \( z_i \). Let us consider the \( z_i \) one by one and in order, rather than simultaneously. Constrain the first integer variable to an integer level within its range. This yields a linear program which we proceed to solve. Assuming a feasible solution exists, the first variable is held at its designated value and the second variable is constrained in like manner. We proceed in this fashion alternately constraining another variable and solving the resultant program, until either we arrive at a feasible solution having constrained all the \( z_i \), or the integer choices for the variables thus far constrained do not admit a feasible solution. In the first case, we have a candidate for the optimal solution to the original problem. In the second case it makes no sense to proceed, since a linear program obtained by adding a constraint to a "non-feasible" linear program must also be non-feasible. In either case, we make a new choice for the integer value of
the latest constrained variable and proceed as before. If we have exhausted the range of the latest constrained variable, we "back-track" to the next latest constrained variable, make a new choice and proceed as before. When we have exhausted the range of the first constrained variable the procedure terminates and the solution with lowest associated objective function is an optimal solution to the mixed integer problem. In all likelihood, this procedure would cut down considerably the number of linear programs examined as compared to "pure" enumeration.

To continue the analysis, one may eliminate additional resultant problems by making use of information available from the solution of the "feasible" linear programs. First, we notice that as we proceed in a forward direction, the objective function cannot decrease. In effect, we have established a lower bound on the optimal solution to the (partially) constrained problem immediately following each decision point. Once a feasible solution is obtained, it represents an upper bound on the optimal solution to the original problem. Therefore, at any point in the procedure, if the objective function for the (partially) constrained problem equals or exceeds that for the current "best" feasible solution for the original problem, it is unnecessary to examine continuations. (At this point we mention two related items: 1. To begin with, one could use an arbitrary initial value as an "upper bound" on the objective function, to be replaced as soon as a feasible solution is obtained; 2. If a dual simplex LP algorithm is employed, the test for "dominance" may be made after each pivot, rather than after the LP has terminated.) Secondly, consider a point in the procedure where we wish to constrain \( x_k \) to an integer value, where in the immediately preceding LP solution it had a value \( x_k^f \). We note that the integer value for \( x_k \) which results in the smallest increase in the objective function in the succeeding LP is either \( \lfloor x_k^f \rfloor \) or \( \lceil x_k^f \rceil + 1 \), and the farther we proceed from \( x_k^f \) the "worse" the resulting objective function. Therefore, if we have established that constraining \( x_k \) to \( \lfloor x_k^f \rfloor + \ell \) (for \( \ell \) a positive integer) produces an increase in the objective function which eliminates any continuation as a candidate for the solution to the mixed integer problem, we may conclude that constraining \( x_k^f \) to \( \lfloor x_k^f \rfloor + \ell \) for \( \ell = \ell + 1, \ell + 2, \ldots, u_k \) need not be considered. Similarly, this discussion applies to integer values less than \( x_k^f \) also.

The ideas discussed in the previous paragraphs form the basis for the algorithm and the expectation that it might be moderately successful in the solution of some mixed integer problems. The next section contains a step by step description of the program which implements the algorithm.
Section 2 The Program

The program was written with the following ideas in mind, not necessarily in order of importance: to demonstrate the practicability of the solution method for problems of limited size; to provide a comparison method for those experimenting in the area of mixed integer problems; to produce actual solutions for those users who are fortunate enough to have "real" mixed integer problems whose "degree of complexity" falls within the effective range of the program. Execution takes place entirely within the memory of the computer.

The program employs a double precision dual simplex linear programming algorithm (not product form) hereinafter referred to as the LP. The tableau is carried in compact Tucker [2], [3] form: the initial number of rows equals the number of problem constraints plus one; the initial number of columns equals the number of true variables plus one. Whenever a zero-constrained slack variable becomes non-basic, it is removed from the problem, resulting in a reduction by one of the number of columns in the tableau. Zero-constrained slack variables arise from two sources: equality constraints in the initial tableau; constraining a basic integer variable to an integer value (see Step 3 below). The number of rows in the tableau remains constant throughout.

Intermediate tableaus corresponding to decision points are stored in upper memory. The program takes advantage of the fact that as each integer variable is constrained, the number of columns in the tableau is reduced by one.

**Step 1** Carry out an LP on the initial tableau. Print the solution. Check to see if all integer variables are integer valued. If so, the problem is terminated; if not, set the initial tolerance for the problem. (Tolerance is defined as the value below which the objective function must stay in order for a continuation of the current sequence of integer-constrained integer variables to be considered as a candidate for the mixed integer solution. Note that the objective function value at the continuous solution represents an absolute lower bound for the mixed integer solution.) Set to 1 the index of the integer variable being constrained.

**Step 2** Choose from those integer variables which are non-basic in the current tableau the one with highest coefficient in the objective function (shadow price.) (The program makes use of the fact that the shadow price represents an underestimate of the increase in the objective function associated with constraining the non-basic integer variable to 1.) If no non-basic integer variable exists, go to Step 3. Otherwise, store the current tableau and constrain the variable chosen to zero. This is done simply by removing the corresponding column from the tableau. (A non-basic variable is constrained to a non-zero integer value by adding the product of this value with each element in the corresponding column to the constant column of the tableau. The corresponding column is then removed from the tableau.) Go to Step 6.

**Step 3** Store the current tableau. Consider all integer variables $z_i$ which are basic in the current tableau (there must be at least one) with value $x_i^*$. For each $z_i$ determine the
absolute difference between the increase in the objective function associated with the initial LP pivot step when \( x_k \) is constrained to \( [x_k^f] \) and when \( x_k \) is constrained to \( [x_k^f] + 1 \). Choose as the integer variable to be constrained that \( x_k \) for which this difference is a maximum and constrain it to the value yielding the smaller increase. The actual constraining is accomplished by adding the integer value to the constant column of the row corresponding to the variable, and then stipulating that the row corresponds to a zero-constrained slack variable. Carry out an LP. If the objective function stays within the tolerance go to Step 5; otherwise go to Step 4.

Step 4 If the current integer variable was constrained to \([x_k^f]\) record the fact that constraining it to values \([x_k^f] - k \leq 2, \ldots)\) within its range need not be considered. Conversely, if \( x_k \) was set to \([x_k^f] + 1\) make note that values \([x_k^f] + 1 \leq k\) need not be considered. Go to Step 7.

Step 5 Test the constrained variable index. If it is equal to \( n_1 \), the number of integer variables in the problem, go to Step 8. Otherwise increase it by one and go to Step 2.

Step 6 Decrease the constrained variable index by one and test it.

Step 6A If it is zero go to Step 2. Otherwise go to Step 7.

Step 7 Determine for the integer variable corresponding to the current value of the index whether its range has been exhausted (explicitly or implicitly) on neither, on one or on both sides of its current value. If it has been exhausted on both sides, go to Step 6. If the variable to be constrained has been exhausted on one side, constrain it to the unexhausted integer value closest to its current value in the proper direction. If the range is unexhausted on either side, determine in which direction to go using the method employed in Step 3, and proceed as for only that side open. (Note that the range of an integer variable which was non-basic when constrained is immediately exhausted from below.) Carry out an LP. If the objective function stays within the tolerance go to Step 5. Otherwise, note that the range of the current variable is exhausted in the direction in which its current value lies from its original value (see Step 4). Go to Step 7.

Step 8 A better feasible mixed integer solution has been obtained. Print the solution. Replace the tolerance by the objective function value. Go to Step 6A.

Step 9 For the current tolerance, all ranges of all the integer variables have been exhausted. If at least one feasible mixed integer solution has been obtained, the last printed solution is an optimal solution to the mixed integer problem and the problem is terminated. Otherwise, the tolerance is increased, the continuous solution tableau is restored, the index of constrained integer variables is set to one, and control goes to Step 2.

If the program is terminated abnormally, the last printed feasible mixed integer solution (if any) is the best obtained.
Section 3  User's Guide

a) The Program Deck [Cards numbered BBMIPbb1 to BBMIP683]

The program is written in the Fortran IV Language for the IBM/360 Operating System. The Distributed deck contains the source program [numbered BBMIPbb1 to BBMIP683] followed by a sample input deck [numbered DATAbbb1 to DATAbbb14] which corresponds to the sample problem which appears later in this section.

b) Optimal Use of Available Memory

Because the program uses core memory for storage of intermediate tableaus (and aside from any considerations as to the efficiency of the algorithm), the subset of mixed integer problems whose solution may be attempted will be restricted in terms of problem size. In order to expand this subset as much as possible, we include a description of how the user may guarantee as little "wasted" memory as possible. This effort consists of making the program array sizes concur exactly with the dimensions of the problem being solved. All such arrays have been included in the first ten (COMMON) specification statements of the source program. Preceding each statement is a COMMENT card giving the exact form of the arrays involved. For the distributed deck, the actual cards correspond to the Sample Problem.

Some of the COMMENT cards require elaborations. The first three COMMON cards designate double precision arrays, which must be positioned on double word boundaries. Following the instructions in the corresponding COMMENT cards guarantees that this condition will be met.

The last array, SAVTAB, is used to store the intermediate tableaus and normally accounts for the major part of core memory required by the program. The formula in the corresponding COMMENT card stipulates the value which is required for the second dimension of SAVTAB if all intermediate tableaus are saved during program execution. However, in cases where such a value for the second dimension causes a compiler error due to insufficient core memory, the user should stipulate the largest second dimension which produces an error free compile and attempt to run the program. This is because a second (slower) solution method exists which involves saving only single columns of intermediate tableaus rather than entire tableaus.

The user communicates to the program via an input parameter (see SAVTAB Dimension Card below) the value with which the second dimension of SAVTAB has been compiled. The program then determines whether the problem will fit and which solution method to use (the faster one will be used if possible) on the basis of this information and proceeds accordingly. The determination is made subsequent to finding the continuous solution for three reasons: 1. The continuous solution may be the mixed integer solution, in which case there is no question of fit; 2. The amount of core required by SAVTAB (regardless of which solution method is used) is a function of the number of columns in the continuous solution tableau, which
may be less than that for the initial tableau because the problem contained equality constraints (see page 5).
3. The amount of core required by SAVTAB is the second solution method if the second solution method is used is a function of the number of non-basic integer variables in the continuous solution. In any case, if the problem will not fit under either solution method, the program prints a message to that effect.

c) Program Input

In the following sections, the various cards comprising the input deck are described. For each card, the line giving the verbal description of the format contains the actual Fortran format used in the program, enclosed in parentheses.

Unless otherwise stated, a blank field is interpreted as a zero value. All input is read from logical input unit 5. However, in the following discussion, the input unit is assumed to be the card reader. A listing of each card follows, in order of appearance in the input deck. In the headings for the detailed descriptions of each card, reference is made to the corresponding card(s) in the sample problem deck distributed with the program.

Identification Card
Optional Output Card.
Objective Function Upper Bound Card
Problem Dimension Card
Variables Upper Bound Card(s)
Constraint Type Card(s)
Right Hand Side Card(s)
Matrix Type Card
Matrix Data Cards

SAVTab Dimension Card [DATAbbb1]

This set of cards appears once for each problem to be solved during a run.

Number of Problems Card [DATAbbb2]

This card specifies the number of problems which the program is to solve in one machine run. The format is one 4-position integer field (14) occupying card columns 1-4. The value must be right justified in the field. Note that the user must follow the instructions under "Optimal Use of Available Memory" in such a manner as to allow for the solution of all the problems in a multiple problem machine run.
Problem Identification Card [DATAbb3]

This card should be used to identify the problem which follows. The format is one 55-position alphanemic field (65H) occupying card columns 1-55. The character in column 1 serves as a carriage control character for the printer and should be a 1 if the user wishes the problem output to begin on a new printer page.

Optional Output Card [DATAbb4]

This card allows the user the option of printing out his initial tableau and his continuous solution tableau. The format is two 4-position integer fields (2I4) occupying card columns 1-4 and 5-8. If the user wishes to print out the initial tableau for the problem, field one should contain a 1; otherwise it should be blank. If the user wishes to print out the final tableau for the continuous solution the second field should contain a 1; otherwise it should be blank. It should be noted here that the initial tableau differs from the input matrix in a number of respects. First, the program automatically introduces a non-negative slack variable corresponding to each row of the input matrix except the first. The row then appears in the initial tableau as a negative representation of the corresponding slack variable. Secondly, in the case of problem variables with negative coefficients in the objective function, the problem automatically introduces their complement variables (see under Variable Upper Bound Card(s)). Finally, the column corresponding to the right hand side appears first.

Objective Function Upper Bound Card [DATAbb5]

The purpose of this card is to supply the program with an objective function upper bound if one exists, or alternatively, a guess at the upper bound expressed in terms of a decimal fraction of the absolute value of the objective function at the continuous solution. The format is two 10-position decimal fields (2F10.0), occupying card columns 1-9 and 11-20. The first field should contain the upper bound if one is known and must be blank otherwise. This option may be employed when the user knows of a feasible mixed integer solution but does not know whether it is optimal. In the absence of a known upper bound, the user may have some idea of how close to the continuous solution the mixed integer solution is. The second field allows for the program to make use of this information. As an example, if the user expects the addition of the integrality constraints to increase the continuous solution objective function by no more than 25\% of its absolute value, this field should contain the value .25. The program uses this value in establishing a tolerance for the objective function. For the example, if Z* is the value of the continuous solution objective function, then the tolerance would be Z*+.25\|Z*|. If no solution within this tolerance is found, the program starts over, increasing the tolerance by the increment .25\|Z*|, and so on. If this field is left blank or contains a zero, the program automatically uses the value .1 as the incremental factor.
Problem Dimensions Card [DATAbb6]

The purpose of this card is to supply the program with the mixed integer problem dimensions. The format is three 4-position integer fields (314) occupying card columns 1-4, 5-8, and 9-12. Field one must contain the integer M, where M is the number of rows of the input matrix, including the objective function. Field two must contain the integer N, where N is the number of columns including the right hand side. Field three must contain the integer %NIVR, where %NIVR is the number of integer variables in the problem. The program, when subsequently reading in the input matrix, assumes the integer variables to be the first %NIVR variables from the left; i.e., no allowance is made for interspersing integer and non-integer variables. All three values on this card must be right-justified in the field.

Variables Upper Bound Card(s) [DATAbb7]

The purpose of this card is to supply the program with an upper bound for each of the variables in the problem. For the integer variables it is absolutely necessary that the user supply an (integer) upper bound since this determines the range of the variable. For the non-integer variables, it is only necessary to provide an upper bound where the variable has a negative coefficient in the objective function. The reason for this is that the program requires an initial dual feasible tableau and achieves it in the case of a problem which is not dual feasible by introducing the complements of those variables with negative coefficients. Of course, if the user wishes to restrict a non-integer variable from above, he may do so regardless of dual feasibility. The format of this card is seven 10-position decimal fields (7F10.0) occupying card columns 1-10, 11-20, 21-30, etc. One field must be allowed for each variable in the problem. When punching an integer value, the decimal point may be omitted if the value is right-justified in the field.

Constraint Type Card(s) [DATAbb8]

The purpose of this card is to indicate the type of each problem constraint as shown below.

<table>
<thead>
<tr>
<th>Constraint type</th>
<th>Indicator</th>
</tr>
</thead>
<tbody>
<tr>
<td>F(x,y) ≥ R.H.S.</td>
<td>+1</td>
</tr>
<tr>
<td>F(x,y) = R.H.S.</td>
<td>0</td>
</tr>
<tr>
<td>F(x,y) ≤ R.H.S.</td>
<td>-1</td>
</tr>
</tbody>
</table>

The format for this card is twenty 4-position integer fields (2014) occupying card columns 1-4, 5-8, 9-12, etc. Indicators should be right-justified in the fields. One field must be allowed for each constraint.

Right Hand Side Card(s) [DATAbb9]

The purpose of this card is to specify the value of the right hand side of each constraint. The format for this card is seven 10-position decimal field (7F10.0) occupying card columns 1-10, 11-20, 21-30, etc. When punching
an integer value, the decimal point may be omitted if the number is right-justified in the field. When punching a non-integer value, the decimal point must not be omitted. One field must be allowed for each constraint, but no allowance is made for a constraint term in the objective function.

**Matrix Type Card** [DATAbb10]

This card indicates the form in which the input matrix (which appears on the following cards) is punched. Two forms are allowed: packed and unpacked. The format for the card is one 4-position integer field (I4) occupying card columns 1-4. If the data is packed, the field should contain a one; if unpacked, it should contain a zero. The two forms for the matrix data are described in the next section. The matrix is composed of the objective function and the constraints of the problem, excluding the right hand side.

**Matrix Data Cards** [DATAbb11 to DATAbb14]

Whichever format is used, the matrix is read in one row at a time beginning with the objective function and continuing until the last constraint. The constraints are assumed to be ordered corresponding to the information contained in the Constraint Type Card(s).

**Packed Format.** This format should be used if the input matrix is sparse, i.e., many zero coefficients. The format is interspersed and consists of seven 3-position integer fields occupying card columns 1-3, 11-13, 21-23, etc., and seven 7-position decimal fields occupying card columns 4-10, 14-20, 24-30, etc. (7(13,F7.0)).

Each pair of fields corresponds to one element of the row being input. The first field contains the column number and the second field contains the value of the coefficient. Each row must begin with a new card. The program assumes it has encountered the end of a row if and only if the column field of the next element is blank. That is, all fields on a card which is not the last card for a row must be used, and a row cannot end with a completely filled card. In the latter case, a blank card must be inserted behind the last card.

**Unpacked Format.** This format consists of seven 10-position decimal fields occupying card columns 1-10, 11-20, 21-30, etc. One field must be allowed for each element of the row being input. Each row must begin with a new card and may end with a full card.

d) **Program Output**

Except for problem identification, the program output is described under section headings comprised of the actual output text. Where the text contains a variable field, the field size will be indicated by a series of 1's in the case of an integer variable, and a series of X's with a decimal point in the case of a decimal variable. The description of output associated with "normal" problem conditions will appear in the approximate order in which the output occurs. This will be followed by a description of output associated with abnormal problem conditions. All output appears on logical output unit 6. However, in the following discussion, the output unit is assumed to be the printer.
Problem Identification

A 53-character problem identification field provided by the user is printed as soon as it is read.

Initial Tableau (See Optional Output Card)

This message, if present, introduces the initial linear programming tableau for the problem. The tableau is set up by the program from the information in the Problem Dimension Card and the Matrix Data Cards, and includes row and column identifiers. Initially, column identifiers represent true program variables and row identifiers represent slack variables. Row identifiers with value zero correspond to equality constraints and represent slack variables which are to be constrained to zero. Complementation of a true variable (for purposes of dual feasibility) is indicated by the addition of 1000 to the corresponding column identifiers. The format for the first part of the tableau consists of as many lines as required to print the \( N \) column identifiers at a rate of 19 per line. Each identifier is right-justified in a 6-position field. Following this are the objective function and constraint coefficients at a rate of 8 per line with each coefficient occupying a 13 character field, 7 characters of which appear after a decimal point. The row identifier corresponding to each row is printed to the far right of the last line of the row. A line is skipped between rows of the tableau.

Continuous Solution

This line appears prior to the output associated with the solution to the linear programming problem disregarding integrality constraints.

Final Tableau  See Optional Output Card

This message (if present) introduces the tableau corresponding to the continuous linear programming solution. The format for the tableau is the same as that described under the heading Initial Tableau. The final tableau may have up to \( k \) fewer columns than the initial tableau where \( k \) is the number of equality constraints in the problem.

Objective Function = XXXXXX.XXXXXX At Iteration I I I I I

This message appears when the continuous solution is obtained and subsequently when a current best mixed integer solution is obtained. The decimal variable part is the value of the objective function at that solution. The integer variable part gives the cumulative number of linear programming pivots which have been carried out up to the point at which the message appears. The message precedes the vector of all true problem variables corresponding to that solution. The format for the problem variable values is 8 13-position decimal fields per line (8F13.7).

Tolerance Set At: XXXXXX.XXXXXX At Iteration I I I I I

This line appears immediately after the printout of the continuous solution, and subsequently whenever the program has determined that no feasible mixed integer
solution exists within the current objective function tolerance (see Objective Function Upper Bound Card). The message signals that the program will now go back and look for mixed integer solutions within the tolerance indicated by the decimal variable part. The integer variable part gives the cumulative number of LP pivots up to that point.

**Optimality Established At Iteration 11111**

This message appears when the problem is terminated under normal conditions, and indicates that the last solution printed represents an optimal mixed integer solution to the problem. The variable part gives the cumulative number of LP pivots at termination.

**End of Problem**

This message is printed whenever the problem is terminated for any reason. Under normal conditions it indicates that the last solution printed represents an optimal mixed integer solution to the problem. Under abnormal conditions, the reason for the termination precedes this message. These conditions and messages are described below.

**Problem Not Feasible (Abnormal)**

This message appears when the program has determined that no feasible solution exists to the continuous LP problem (exclusive of integrality constraints).

**Continuous Solution is Integer Solution (Abnormal)**

This message appears when the program determines that the continuous solution has all integer variables equal to integer values, i.e., the continuous solution is the optimal mixed integer solution.

**Problem Too Big For Machine Size (Abnormal)**

This message appears when the program has determined that memory space is not sufficient to allow for solution of the problem.

e) Executing a Machine Run

After the source deck has been altered as described under Optimal Use of Available Memory, insert the proper OS/360 control cards and the data deck (see Program Input). The program may then be compiled and executed using Operating System/360. The program assumes input to be read from logical unit 5 and output to be written on logical unit 6.

The program will execute serially each problem in the data deck with resultant output as described under Program Output. When all problems have been executed, control is returned to the system by a CALL EXIT statement.
f) Program Limitations

Problem execution times will vary directly as the values of $M$, $H$, $NKIVP$, and $U_2$, the upper bounds on $R_i$. Execution time will naturally vary according to the object machine used. Probably the greatest single factor in determining whether the optimal solution is obtained in a reasonable amount of execution time will be the proximity of the mixed integer solution value to the continuous solution value. Execution times for some sample problems appear under Some Sample Results.

The program employs a double precision LP. We note that during LP calculations tableau elements whose magnitude falls below that of the variable $ADELT$ are considered to be zero. This value is initialized at .0005 in the distributed program (see the source deck).
Section 4  References


5. K. Spielberg and R. Shareshian, On Integer and Mixed Integer Programming, Reports 1 and 2, New York Scientific Center, IBM.


Appendix E

Documentation of Zero-One Integer Programming Routine
Direct Search Zero-One Integer Programming 1

DZIPI

Carlton E. Lemke
Kurt Spielberg

May 1966

Direct all inquiries to:
Kurt Spielberg
IBM New York Scientific Center
590 Madison Avenue
New York, N.Y. 10022

TABLE OF CONTENTS

Section 1  Purpose . . . . . . . 77
Section 2  Restrictions . . . . . 78
Section 3  Survey of Method and Expected performance . . . . . 79
Section 4  Discussion of Algorithm . 80
Section 5  Input Parameters . . . . 85
Section 6  Data Deck, Operating Instructions . . . . . . 86
Section 8  References . . . . . 100
Direct Search Zero-One Integer Programming 1

DZIP1

ABSTRACT

The program DZIP1 is intended to solve minimization problems of the linear programming type, subject to the additional constraints that all the variables may take on the values 0 or 1 only. The program uses a direct tree search method developed by the authors. Results so far indicate that the method used is considerably faster, perhaps by a factor of 2 to 5, than the well known additive algorithm of E. Balas. A number of problems of 30 to 45 variables and a similar number of constraints have been solved optimally in about 5-30 minutes on the IBM 360 Model 40 and approximately .5 to 5 minutes on the IBM 7090.

The program is set up to treat problems of as much as 150 variables and 50 constraints. It can easily be modified to accommodate problems of similar overall size (total number of matrix entries) but different structure by simple changes of dimension statements.

Reasonably good suboptimal solutions have been obtained for problems of as many as 28 constraints and 89 variables. In the case of problems of this size, however, the program exhibits the usual behavior of integer programming algorithms. The results obtained depend in an unpredictable fashion on seemingly innocuous changes in input data and algorithm. An attempt has been made, therefore, to provide the user with a number of options for changing the strategy of the algorithm and for influencing the initial direction of the search.

The program was written in FORTRAN for the IBM 360 Model 40 and has been constrained somewhat by restrictions imposed by the available version of the compiler. This should, however, cause no difficulties to the user.
Direct Search Zero-One Integer Programming I

DZIP1

USER INFORMATION

Section 1 Purpose

DZIP1 is intended to solve problems of the form:

\[
\begin{align*}
\text{min} & \quad c^T \cdot y \\
\text{subject to} & \quad A \cdot y \leq b \\
& \quad (\text{or} \quad A \cdot y + s = b) \\
& \quad y_1 = 0 \text{ or } 1 \\
& \quad (s_i \geq 0)
\end{align*}
\]

Here, \( c \) (the cost vector) and \( y \) (the vector of variables \( y_1 \)) are vectors of \( n \) components, \( b \) (the right-hand side vector) and \( s \) (the slack vector, which is introduced solely for purposes of the algorithm and does not appear in the input) are vectors of \( m \) components. \( A \) is an \( m \) by \( n \) matrix.

The user supplies \( c, b, A, m, n \) and certain control parameters which will be described later. The components of \( c, b \) and \( A \) are integers unrestricted in sign. For purposes of the algorithm it is convenient to have the cost coefficients non-negative and in order of increasing magnitude. The program, therefore, initially transforms the data into this form (to the "program variables"). The output is, however, given in terms of the "problem variables" (and, optionally,
also in terms of the program variables). These transformations are simple and will not be described in detail, as the program listing will be quite adequate in this respect.

Section 2 Restrictions

a) The program is currently designed for problems up to
\[ m = 50 \quad \text{and} \quad n = 150. \]
To change the program for use with problems of maximal size \( M \) by \( N \), change the arguments of the dimension statements from 50 to \( M \), from 150 to \( N \) and from 151 to \( N+1 \). Also change the first FORTRAN instruction from \( K\)\(\text{TEST}\) = 150 to \( K\)\(\text{TEST}\) = \( N \).

b) The computations are almost entirely performed in the fixed point mode and, essentially, consist of additions and subtractions only. There should be no overflow difficulties unless numbers become too large. Throughout the program the number \( 10^7 \) has been, arbitrarily, taken to be "large" and should not be exceeded in the normal course of operations. The user will, most likely, not encounter any difficulties if the various coefficients are scaled so as to be smaller than \( 10^5 \) in magnitude.
Section 3  Survey of Method and Expected Performance

The method has been developed by the authors and will be described briefly in the next section and, probably, in more detail in a paper submitted to the journal of ORSA. It is a direct search method with carefully planned recording scheme and diagnostic and prognostic tests for the search procedure.

Work in this area was stimulated recently by a paper of E. Balas\(^1\). Other authors in the area have been F. Glover\(^2\), A. M. Geoffrion\(^3\), and R. J. Freeman\(^4\). A convenient survey of the various methods for integer programming will be found in a paper of M. L. Balinski\(^5\).

A number of numerical tests seem to indicate that the algorithms used for this program is considerably superior to that originally proposed by E. Balas. Running times have been approximately 3-4 times shorter than for the Balas scheme. For a detailed discussion of computational experience - see Section 7.

With the current program, the user should have little difficulty, barring the presence of unexpected programming errors, in solving optimally problems involving up to 30 or 40 variables. He will find that the optimal solution, or at least a good suboptimal feasible solution, is attained quite rapidly and that the major portion of the computing time is usually taken up in ascertaining the optimality of the solution (in the "cleanup stage").

In the case of larger problems, suboptimal feasible solutions are sometimes obtained rapidly and sometimes not. The search for feasibility is, at least without linear PROGRAMming techniques, difficult. The user may find that problems which prove intractable to one approach will yield readily to another. For this reason an attempt has been made to provide the user with the option of altering strategy and direction of search by means of input parameters.

The authors solicit communications about the performance of the program to possibly guide them in their future work in this area.
Section 4  Discussion of Algorithm

a) General Description
The algorithm usually starts at the zero-point (all components of \( y \) are zero) and proceeds by adding one component of 1 at each forward step. When one realizes that any permissible forward step leads to an unremovable infeasibility or, at best, to a feasible solution not better than the best solution obtained so far, one takes "a backward" step and proceeds in a new direction.

A sequence vector, \( S \), records which steps have been taken so far. E.g., \( S = (0,1,7,9) \) means that \( y_1 = y_7 = y_9 = 1 \), all other \( y_i \) being zero. (The leading zero was introduced for convenience in programming.) A state vector, \( \Sigma \), of \( n \) components records which forward steps are permitted \((\eta_j = 1)\) or not \((\eta_j \leq 0)\). Whenever it is ascertained that a branch is not permissible, the corresponding component, \( \eta_j \), is set to zero. On forward steps each non-positive component of \( \eta \) is reduced by one; on backward steps these components are increased by one. This procedure ensures that the state vector is always up to date. (A simpler procedure is possible but has not been incorporated in the current algorithm.)

During the algorithm the slack vector \( s \) (initially set to b) is updated. E.g., \( y_j \rightarrow 1 \) leads to \( s \rightarrow s - a_j \), \( a_j \) being the \( j \)th column of matrix \( A \). Feasibility is attained whenever \( s \geq 0 \) for all \( i \).

In the initial phase the input data are transformed such that \( (c_j) \) is a non-decreasing sequence. This implies that the objective function \( z = c^T y \) is non-decreasing on forward steps. As a consequence, whenever feasibility is attained or whenever all permissible \( j \) are such that \( z + c_j \) exceeds the current ceiling \( z \) (the best value of the objective function attained so far), the search in the currently pursued direction can be abandoned; a backward step is taken.

Permissible branches (forward steps) are also "cancelled" whenever a scan of the \( \eta \) matrix indicates that pursuit of the considered branch could not possibly lead to feasibility. \( (\eta_j \) is set to zero).

The search is exhaustive, but considerable effort is expended to detect possibilities of "cancellation" of portions of the tree.

b) Preferred Variables
Let \( \Pi \) be the set of permissible indices.
\[
\Pi = \{ j \mid \eta_j = 1, \exists j \leq j \leq j \}
\]
(For programming purposes these indices are usually collected in a block of storage, \( \Pi ) \). A subset \( P \) of \( \Pi \) is said to be a "preferred" set whenever, at a point \( \gamma \) (iteration count) or \( k \) (count of non-zero components), it is ascertained that one need merely consider the indices \( j \in P \) as candidates for a forward step. (The superscript \( \gamma \) will be put in parentheses.)

Subsets \( P \) can be constructed from special types of Gomory cuts.
\[
\sum_{j \in P} a_{ij} \cdot y_j \leq b_i
\]
be the inequality corresponding to row \( i \) of the matrix. Let \( s < 0 \), i.e., let row \( i \) currently be "infeasible". This implies that some of the \( a_{ij} \) are negative. (Otherwise we would have recognized, earlier, infeasibility and would have initiated a backward step).
The Gomory cut derived from 2) would be (with $\lambda = -$)

$$
\sum_{j \in C} y_j \geq 1 \quad \forall \gamma = \{ j \in C \mid a_{ij} < 0 \}
$$

Whenever $|s_i| > |a_{ij}|$ (for some $j$ such that $a_{ij} < 0$), however, we can derive a stronger cut by eliminating the variable $y_j$ from 3) and replacing $|s_i|$ by $|s_i| - |a_{ij}|$.

This follows from the fact that an inequality 2) can be combined with a suitable zero-one inequality $y_j \leq 1$ to imply a resultant inequality $y_j + \sum_{j \in C} a_{ij} y_j \leq s_i + 1$.

(E.g., $-4y_1 + 5y_2 - y_3 = 2y_4 \leq -5$ and $y_3 \leq 1$ imply $-4y_1 + 5y_2 - 2y_4 \leq -4$). When this process is exhausted one can apply Gomory reduction to obtain a final inequality of type 3). Example: $-4y_1 + 5y_2 - y_3 - 2y_4 \leq -5$ implies $y_1 + y_3 + y_4 \geq 1$ by Gomory reduction, but implies either $y_1 \leq 1$ or $y_3 + y_4 \geq 1$.

(Evidently the reduction procedure can be used in various ways. We shall denote by the term "complete reduction" the procedure in which elements $a_{ij} < 0$, if possible, are eliminated in order of increasing magnitude. This ensures that with each feasible row we associate a corresponding cut 3) with minimal number of elements $y_j$. In the above example we would obtain $y_1 \leq 1$.

At point $k$ we scan all $i$ for which $s_i < 0$ and determine by complete reduction the corresponding sets $P(i)$. We then associate with point $k$ the row $i$ with minimal $P(i)$ as the "preferred row" $i_*$. Evidently, when at point $k$, we need only examine the branches in $P(i_*)$.

The introduction of this procedure reduces the number of points to be considered substantially, perhaps by a factor from 3 to 10.

c) Data and Parameters Available During Iteration $y$

Mnemonics used in the program are given in square brackets. $i$ and $j$ refer to row $i$ and column $j$ of the matrix.

$y$ [KIT]- iteration count.

$k$ [KSC]- "distance" of point $y^{(y)}$ from origin, number of non-zero components, in $y^{(y)}$.

$j_f$ [JF]- first index $j$ to be considered (or "permissibility") in $n$.

$j_f(k)$ [KFRST]- vector of $k$ "leftmost" indices.

$j_b$ [JL]- last index to be considered.

$j_e(k)$ [KLST]- vector of $k$ "rightmost" indices.

$z$ [KJS]- current value of objective function.

$z^*$ [KZSTR]- current ceiling (upper bound) for objective function.

$\lambda(k)$ [KZLB]- current lower bound for objective function.

(Computed only when input parameter KINCP is one.)

$s(i)$ [KYS]- current right hand side (slack) vector of $m$ components.

$A(i,j)$ [KARR]- ($m$ by $n$) matrix of coefficients $a_{ij}$.

$c(j)$ [KCJ]- cost vector of $n$ components $c_j$.

$n(j)$ [KCNTV]- state (continuation) vector of $n$ components $n_j$.

$n_j = 1 \rightarrow$ branch $j$ is permissible

$n_j = 0 \rightarrow$ branch $j$ is not permissible

$\beta = 1$ means branch $j$ is permissible.
\[ \sigma(k) \quad [\text{SEQ}] \quad \text{sequence vector which records the non-zero components of } \gamma^k. \]

\( \Pi \quad [\text{RPERM}] \quad \text{set of permissible indices (branches).} \)

\( P \quad [\text{KPREP}] \quad \text{set of preferred indices, } P \subseteq \Pi. \)

\( a \quad [\text{KFWRD}] \quad \text{parameter which indicates whether point } k \text{ has been reached on a forward (} a=1 \text{) or on a backward (} a=0 \text{) step.} \)

\( \gamma c \quad [\text{KTVL}] \quad \text{input parameter which specifies the iteration at which the cleanup phase is deemed to begin.} \)

\[ \gamma \text{c} \quad \text{(start of new strategy).} \]

\( \delta \quad [\text{KAPT}] \quad \delta = 0 \quad \text{for } \gamma \leq \gamma c \]

\[ \delta = 1 \quad \text{for } \gamma > \gamma c \]

\( \zeta \quad [\text{KTOL}] \quad \text{input parameter specifying that the search may be abandoned when } z \text{ cannot become less than } z^* - \zeta. \)

\( i_0 \quad [\text{IP}] \quad \text{row index for preferred row.} \)

\( i_0(k) \quad [\text{KIPR}] \quad \text{vector of row indices for preferred rows.} \)

\( u_1 \quad [\text{KMAXP}] \quad \text{vector of } m \text{ entries } u_{1i} = \max \{a_{ij} | a_{ij} \geq 0, j \in J \} \)

\( u_2 \quad [\text{KMAXN}] \quad \text{vector of } u_{2i} = \max \{a_{ij} | a_{ij} < 0, j \in J \}. \)

\( \text{KINC} \quad \text{input parameter, } 1(0) \ldots \text{lower bound (not) computed from Gomory cuts.} \)

\( \delta \quad [\text{KINCR}] \quad \text{increment to objective function.} \)

\[ * \quad j = \{j | j_f \leq j \leq j_k \} \]

\[ d) \quad \text{Outline of Algorithm} \]

\[ \text{The following is not complete. The omitted details should be easily detectable from the Fortran listing.} \]

\[ \text{If not stated otherwise, the steps described below are to be considered consecutive.} \]

\[ \text{Superscripts } \gamma \text{ can be omitted. The iteration count } \gamma \text{ has, aside from comparison with } \gamma_{c}\text{, no influence on the algorithms.} \]

\[ 1. \quad \text{Initialization:} \]

\[ 1.1 \quad \text{data transformed such that } c_{j+1} \geq c_j \geq 0. \]

\[ 1.2 \quad z \text{ set to } a\text{-priori bound specified in input, or to } \sum_{j=1}^{n} c_j + 1 \]

\[ 1.3 \quad n \quad j = 1, \ldots n \quad j_f = 1 \quad j_k = n \quad a = 1 \]

\[ 1.4 \quad k = \gamma = \delta = 0 \quad (\delta \text{ will be changed to } 1 \text{ when } \gamma \text{ reaches } \gamma_{c}\text{ }) \]

\[ 2. \quad 2.1 \quad k \ightarrow k + 1 \quad \gamma \rightarrow \gamma + 1 \]

\[ \text{If } s_i \geq 0, \text{ all } i, \text{ go to 4.} \]

\[ \text{If some } s_i < 0: \]

\[ \text{Update } j_k: \quad j_k \rightarrow j_k^{*}, \text{ largest } j \leq j_k \]

\[ \text{such that } n_j = 1 \text{ and } z + c_j + \zeta < z \]

\[ \text{Test (after each decrease in } j_k\text{): } j_k < j_f ? \]

\[ \text{Yes, go to 3 (backward step).} \]

\[ \text{No, continue with next } j. \]

\[ \text{70} \]
2.2 – Save \( j \_f(k) \)
\( \alpha = 1 \) (after forward step), go to 2.3
\( \alpha = 0 \) (after backward step), go to 2.4

2.3 – update \( j \_f, \ j \_j \rightarrow j \_j \), smallest \( j \) \( \in J \)
such that \( \eta_j = 1, \ J = \{ j \_f, j \_j \} \)
If \( j \_f \) exists, save \( j \_f \) in \( j \_f(k) \), go to 2.4.
If \( j \_f \) does not exist, go to 3.

2.4 – construct \( \Pi(k) \) (set of permissible branches) from:
(i) \( (\alpha = 1) \ldots \Pi(k-1), \ J \) and \( \eta \).
(ii) \( (\alpha = 0) \ldots \Pi(k-1), \ J \) and \( \eta \).
If \( \Pi(k) \) is empty, go to 3.

2.5 – when \( \alpha = 1 \), set \( \lambda(k) \) equal to \( \lambda(k-1) \).
when \( \alpha = 0 \), \( \lambda(k) \) has been set previously.

2.6 – Infeasibility test:
If, for any \( i \), \( t_1 = s_1 - \sum_{j \in J(i)} a_{ij} < 0 \), go to 3.
\( J_1(i) = \{ j \mid j \in J, \ \eta_j = 1, \ a_{ij} < 0 \} \)
(Problem cannot become feasible by additional branching).

2.7 – Cancellation test:
(i) For all \( i \), test: \( t_1 < v_1 \) ?
No, consider next \( i \)
(After last \( i \), go to 2.8)
(ii) For all \( j \) \( \notin J \): \( t_1 - a_{ij} < 0 ? \)
No: take next \( j \) (after last \( j \), proceed to next \( i \), eventually to 2.8)

(iii) Yes, set \( \eta_j = 0 \), replace corresponding element of \( \Pi \) by 0 (cancel \( j \)).
For all \( i \) recompute \( t_1 \).
If \( t_1 < 0 \), for any \( i \), go to 3.
Otherwise go to (i).

During 2.7, the \( v_1 \) are changed after each cancellation.

2.8 – Readjust \( \Pi \) by omitting zero element \( \eta \).
If \( \Pi \) is empty, go to 3.
If \( \alpha = 1 \) (after forward step), go to 2.12.

2.9 – \( \alpha = 0 \), set \( i_\# \) from \( i_\#(k) \).
If \( i_\# = 0 \) (\( P(i_\#) \) is empty), go to 3.
Otherwise, determine \( P(i_\#) \) (set of preferred variables) by complete reduction from \( i_\# \).
If \( P \) is empty, go to 3.

2.10 – Determine forward step \( j \) by Balas value criterion applied to \( j \in \Pi(i_\#) \).
\( i.e. \) for each \( j \in \Pi(i_\#) \) compute
\( v(j) = 0 + \sum_{i \in j} (s_i - a_{ij}) \)
\( T_1 = \{ i \mid s_i - a_{ij} < 0 \} \)
Set \( j \) equal to that \( j \) which maximizes \( v(j) \).
\( v(j) = 0 \) implies feasibility at next point.

2.11 – Forward step:
\( z(k+1) = \bar{z} \)
\( \tau(j) = 0 \)
\( \tau(j) \rightarrow \bar{z} = 1 \) if \( j \in J \) such that \( \eta_j \leq 0 \)
\( z \rightarrow z = c_{j}^{-} \)
for all \(i: s_i \rightarrow \bar{s}_i \bar{a}_{ji}\)
\(n = 1\)
go to 2.1

2.12-a=1 (after forward step)
If \(\gamma < \gamma_{c1}\) determine \(\bar{J}\) by Balas value test
applied to \(\gamma_{c1}\) (see 2.10).
Then, go to 2.14
If \(\gamma \geq \gamma_{c1}\), go to 2.13

2.13- Determine \(i_a\) by complete reduction on all
"infeasible" rows \(s_i < 0\). Use is made of \(\nu_2\).
Set \(i_a(k) = i_a\).
If \(\text{KINCP} = 1\), determine a lower bound \(v(k) = z + \bar{\delta}\),
where \(\bar{\delta}\) is the largest increase obtained by one
Gomory cut (several cuts are tried, at least one
for each infeasible row).
(Note: When several rows are tied as candidates
for \(i_a\), i.e., when they have equal numbers of
negative \(a_{ij}\) after complete reduction, the
preferred row is taken as the first candidate
row which has \(a_{ij} < 0\) for \(j = \bar{J}\))
Go to 2.10.

2.14- Determine \(i_a\) by complete reduction on all
"infeasible" rows \(s_i < 0\), such that \(a_{ij} \bar{a}_{ji} < 0\),
with \(j = \bar{J}\) considered not permissible.
In ties, take first \(i\). Set \(i_a(k) = i_a\).
In contrast to 2.13, \(P(i_a)\) may be empty. In that
case, \(i_a\) is set to 0. (Implemented somewhat
differently in the current program).
Go to 2.11.

3. Backward Step:
3.1 - Set \(\bar{J} = \eta(k)\)
\(k \rightarrow k-2\)
If \(k < 0\), go to 5 (termination)
3.2 - \(z \rightarrow z - c_j\)
Restore \(v_{ij}\) to original \(v_{ij}\)
\(s_i \rightarrow s_i + a_{ij}\)
\(j_f\) set to \(j_f (k+1)\)
\(j_g\) set to \(j_g (k+1)\) new \(J\)
\(\eta(j) \rightarrow \eta(j) + 1\) for \(j \in J\), \(\eta_j \leq 0\)
\(n = 0\)
Go to 2.

Record new solution.
Go to 3.

5. Optimality ascertained or no feasible solution found
Proceed to read in data for new problem.
If not stated otherwise, the statements below correspond to parameter settings of 1 (or positive integers). The option is bypassed by, or the opposite holds for, a setting of 0 (or "blank" on input card).

\( \varepsilon \) [K10L] - tolerance. Search abandoned (in current direction) when \( z > z^* + \varepsilon \).

KORD - input data already such that \( c_j + 1 \leq c_j \geq 0 \).

KUPB - program is terminated after KUPB iterative steps
(KUPB = 0 -> KUPB is set to 500000)

KINCP - lower bounds are computed by means of Gomory cuts.

KOUTP = 0 - output in terms of problem variables.
= 1 - output in terms of program variables.
= -1 - output in terms of both variables.

KAPLB - a-priori lower bound (not utilized in current version of program).

KAPUB - a-priori upper bound, used as initial value of ceiling \( z^* \).

KITCL - iteration number at which a changeover is made from first to second strategy.

First strategy: the next branch \( \overline{J} \) is taken from all \( j \in \mathcal{F} \) before the preferred set \( \mathcal{P} \) is chosen.

Second strategy: the preferred set is chosen first; then the branch \( \overline{J} \) is determined from \( j \in \mathcal{P} \).

A limited set of sample problems seems to point to superiority of the strategy \#2. Hence one may wish to set KITCL = 1 (0 or blank is taken to mean 1).

KSTR1 - full cancellation test (recommended; see 2.7 in algorithm)
KSTR1 = 0 -> bypass of 2.7.

KSTR2 = 0 -> p (as many as 7) restarts of search into new direction.
KSTR2 = p -> p restarts of search into new direction.

If \( p > 0 \), the third data card should contain a list of \( p(\overline{J}) \) iteration numbers.

When such an iteration count is reached before a feasible solution has been found, the search restarts at zero with all previously tried starting branches prohibited. After attainment of a feasible, suboptimal, solution, the search starts once again with all branches readmitted.

This option is intended for large problems where choice of the initial branch may seriously affect the performance of the algorithm. In one 89 variable problem, one choice of the first step led to no feasible solution after 140000 (\#) iterations, whereas another gave feasible solutions after about 100 iterations.

KSTR3 = p - a starting point of \( p \) components is supplied.

If \( p > 0 \), the first two (or three, if KSTR2 > 0) cards should be followed by the components of the starting point (7 to a card).

KWR(1), KWR(2), ..., KWR(7)

Control parameters for writing. In the current version of the program, feasible suboptimal solutions are written unconditionally. (In a former version, they were written conditional on KWR(1). The reader may yet discern a program parameter KSTR1 which is a remnant of this version and has not been removed).
There is also an unconditional printout of the translation vector KTRAN which provides the correspondence between program and problem variables. (Except for complementation, which is indicated by negativity in block KUBJ).

Each KWR(1) refers to a different set of output parameters and variables. However, KWR(2) = 0 implies that KWR(3) and KWR(4) are treated as 0 also.

KWR(1) - output of input data.
KWR(2) - output of sequence vector and of certain other parameters (such as objective function, ceiling, iteration count, etc.) after each iteration.

The user will hardly have occasion to use the other parameters KWR(3) to KWR(7). If he should wish to use them, however, he should consult the Fortran listing.

Provision has been made for intermittent printout. When such output is desired, the parameters KWR(1) of interest should be set to negative integers -p and a corresponding set of parameters KPR(1) should be set to negative integers -q. (This option is available only for KWR(1) through KWR(4).)

Printout is then started at iteration p and repeated in intervals of q iterations.

E.g.: KWR(3) = -5, KPR(3) = -2
printout at iteration no. 5, 7, 9, 11, ...

Section 6 Data Deck, Operating Instructions

All cards contain as many as 7 integers in the 7 fields, 1-10, 11-20, ..., 61-70.

Numbers are to be right justified. Data in a particular section below always start a new card.

1. 1 card: KM, KN, KTOL, KORD, KUP8, KINCP
   KM ... m, number of constraints
   KN ... n, number of variables.
   For other parameters see section 4.5
   (input parameters)

2. 1 card: KOUTP, KAPLB, KAPUB, KITCL, KSTR1, KSTR2, KSTR3

3. - Only if KSTR2 > 0. 1 card, see section 5.

4. - Only if KSTR3 > 0. As many cards as are required to put in a starting point y in for the iterative procedure. The cards contain the non-zero components of y.
   E.g., y = (0, 1, 1, 0, 0, 1) would be given by the 3 numbers 2, 3, 6, KSTR3 would be = 3.
   Note: The y are assumed to be "program" variables (i.e., the variables after the transformation which ensures c ≥ 0).

5. 1 card: KWR(1), KWR(2), ..., KWR(7)

6. 1 card: KPR(1), KPR(2), ..., KPR(7)

7. - Cost coefficients c

8. - Right hand sides b

9. - Matrixe coefficients a

Any new row 1 starts on a new card.
Note: The problem must be of the form \( \min (c^T y | A y \geq b) \).

An equality constraint must be (wastefully, to be sure) expressed by two constraints.

E.g., \( \max \{2y_1 - 3y_2 \mid y_1 + y_2 = 1\} \)

becomes:

\[ \min \{-2y_1 + 3y_2 \mid y_1 + y_2 \leq 1, -y_1 -y_2 \leq -1\} \]

Operating Instructions

DZIP1 is a standard Fortran IV program. It has been tested without difficulties on both system 360 and the 7090. Some slight attention might have to be given to the operating system at an installation and to the conceivable restrictions of a particular compiler. The program uses standard input and output.
Section 8 References

1. E. Balas, An Additive Algorithm for Solving Linear Programs with Zero-One Variables. 


4. R. J. Freeman, Computational Experience with the Balas Integer Programming Algorithm. 
   Rand Corporation Report AD622767.


Appendix C

Documentation of Transportation Code
FORTRAN TRANSPORTATION CODE

by

Basil C. Kahan

November 1968
## PROGRAM BRIEF

This FORTRAN program solves the transportation problem on System/360 computers possessing the FORTRAN G or H compiler. A stepping-stone algorithm is employed, using cross-reference tables retained in core. The costs matrix is scanned using a technique whereby only valid entries in the costs matrix are considered. This method should prove more efficient than complete column scans in problems where the costs matrix is less than 50% active, and considerable time will be saved when the matrix is very sparse.

With the exception of intermediate total costs which are calculated at suitable intervals, integer arithmetic is used to obviate rounding errors. This limits the overall size of numbers for a meaningful problem to the maximum which can be stored in a 32-bit word. Two-byte integer form is used for reference table arrays.

The minimum configuration is a 128K 360 model 40 using disk for program and data storage, and a single tape drive for checkpointing.

It is not possible to give a formula for execution time, which is a function of problem size, severity of constraints and the equipment available. However, execution times indicate speeds of execution comparable with those for existing programs, attenuated by a factor due to FORTRAN I/OCS routines. Execution time for the sample problem on a 360/40 was about 90 seconds for 102 iterations; this includes preparation of a work file on disk and the output section of the program. However, this time was excessively high because, for demonstration purposes only, a blocking factor of 30 was specified for costs records. The suggested factor of 400, which allows full track records on a 2311, reduces I/O wait time to a minimum. When the whole program and data was core resident, which is feasible for small problems, the execution time was almost halved.
DETAILED PROGRAM DESCRIPTION

1. INTRODUCTION

The Transportation Problem has many applications in modern industry where goods or services must be allocated in an optimum manner to minimise costs or maximise profit. The problem arises because although there are usually many allocations, an allocation obtained by inspection is rarely optimum.

By supplying relevant data to the computer, a feasible allocation can be obtained. The program improves the allocation by successive iterations until no further optimisation can be achieved.

The Transportation Problem can be phrased as follows:

A number of consumers are to be allocated specified quantities of a product by a number of suppliers. Given the requirements of each consumer, the stock of each supplier, and the cost of transporting one item from each supplier to each consumer, fill the orders at minimum cost.

2. MATHEMATICAL FORMULATION

The Transportation Problem can be stated mathematically as follows:

Subject to the conditions

(i) \[ \sum_{i=1}^{M} x_{ij} \leq S_j \quad \text{for all } j, \quad 0 \leq j \leq M \]

(ii) \[ \sum_{j=1}^{N} x_{ij} = P_i \quad \text{for all } i, \quad 0 \leq i \leq N \]

and (iii) \[ x_{ij} \geq 0 \quad \text{for all } i \text{ and } j \]

find values \( x_{ij} \) to minimise

\[ \sum_{i=1}^{M} \sum_{j=1}^{N} c_{ij} x_{ij} \]

where \( x_{ij} = \) Quantity supplied to consumer \( j \) from source \( i \)
\( S_j = \) Quantity available at source \( j \)
\( P_i = \) Quantity required by consumer \( i \)
\( c_{ij} = \) Cost of transporting one unit from source \( i \) to consumer \( j \).

To facilitate solution of the problem the concept of "slack" is introduced. Condition (i) implies that not all the stock at source \( i \) will be used. If this slack is treated as an extra consumer with zero costs and unspecified requirements condition (i) becomes

\[ \sum_{j=1}^{N+1} x_{ij} = S_j \]

where \( x_{i,N+1} \) is the slack at source \( i \).

(Although Transportation Problems are often proposed on the assumption that the resources will be just sufficient to fill the requirements, it is unlikely in practice that this will occur when many sources and requirements exist. Further, the slack concept can assist in scheduling the amount of resources, because by over-specifying the available supplies it is possible to extrapolate optimum resources when these are subject to variation).

Further in many practical cases a large percentage of supplier - consumer pairs are either infeasible or uneconomic. The constraints of such a limitation can result in there being no possible solution even though total resources are greater than total requirements. By incorporating a fictitious supplier with capacity sufficient to satisfy all requirements (at an uneconomically high cost) a mathematical solution can be obtained. This technique is also valuable in the early stages of optimisation.

The problem can now be stated:

Subject to conditions

(i) \[ \sum_{i=1}^{M+1} x_{ij} = S_j \quad \text{for all } j, \quad 1 \leq i \leq M + 1 \]

(ii) \[ \sum_{i=1}^{M+1} x_{ij} = P_i \quad \text{for all } i, \quad 1 \leq j \leq N \]

(iii) \[ x_{ij} \geq 0 \quad \text{for all } i \text{ and } j \]

find values \( x_{ij} \) to minimise

\[ \sum_{i=1}^{M+1} \sum_{j=1}^{N} c_{ij} x_{ij} \]
3. **TECHNIQUE FOR SOLUTION**

The problem is solved using the stepping-stone method, which is iterative in operation. At each stage a feasible allocation of non-negative values \(x_{ij}\), called a "basis", satisfying all constraints, but not generally at minimum cost, is modified to reduce cost. Iterations are performed until a unique optimum cost and an associated optimum allocation are found. When the user specifies optimisation and the resources are insufficient, the program assigns available resources in an optimum manner, and notes the specific requirements outstanding.

The basis, containing \((M+N+1)\) elements, where \(M\) is the number of suppliers and \(N\) the number of consumers, can be considered as a linked graph consisting of vertical and horizontal lines joining the positions of elements in the matrix of costs.

To avoid blind searching for basis elements, comprehensive reference tables are built up. These are established at the beginning of solution and updated after each iteration.

At each iteration, evaluators \(u_j\), \(v_i\) are found for the current basis, one \(u_j\) for each row and one \(v_i\) for each column, which satisfy the linear equation

\[ u_i + v_j = c_{ij} \]

for each cell of the basis.

Basis elements in the slack column have zero cost \(c_{ij}\), a zero column evaluator \(v_j\) and thus a zero row evaluator \(u_i\). Basis elements assigned from the fictitious source have zero row evaluator \(u_j\); but the column evaluator \(v_i\) is normally non-zero.

Thus a path from slack can be solved in turn to appropriate \(u_i\) and \(v_j\) by systematic elimination.

When the evaluators are determined the cost matrix elements are scanned to determine values \(w_{ij}\), where

\[ w_{ij} = u_i + v_j - c_{ij} \]

Such values must be zero for basis elements and they indicate the improvement in the cost function which will be realised by the introduction of one unit of element \(i, j\) into the solution. The object of successive scans is to locate as quickly as possible that element which potentially yields the "best" improvement in overall cost. Details of the scanning method are presented in the description of the GETWILL routine. The optimisation process terminates when no positive \(w_{ij}\) element exists in the whole system.

4. **ROUTINES**

**MAIN**

The MAIN routine is basically an initialising and calling program which is used to access three main logical instruction sequences. These deal respectively with input, optimisation and output, and are all currently kept in core, but the logic has been organised so that these sections may be overlaid with a minimum of program modification.

By allocating large arrays in storage and specifying problem sizes in this routine all subroutine calls reference array sizes indirectly; this facility allows the programmer to change the dimensions of arrays if they are outside the standard option with the minimum of effort.

**SUBROUTINES OF MAIN**

1. **BEGIN**

Subroutine BEGIN handles the necessary writing of working data onto peripheral devices and allows for restarting problems which have been terminated before final optimisation by reading a checkpoint tape to obtain details of the basis. When a problem is being solved from scratch, with no checkpoint available, BEGIN calls SETUP.

The program totals the available resources and requirements and terminates with a "RESOURCES INSUFFICIENT" message if the requirements are too great.

However, the User can specify a branch round this terminating condition in which case the program will optimise as far as possible using a fictitious source to satisfy the lack of availability.

In all executions of the program (whether resources are sufficient or not) a fictitious source is automatically introduced with a user specified uneconomically high cost in order to speed up the initial stages of optimisation.
2. SETUP

Subroutine SETUP performs two functions. It sets up an initial basis and establishes reference tables.

The initial basis is set by allocating the cheapest supplier available in turn to the consumers in numerical order until all requirements are satisfied. The basis elements are stored in an array BASIS, which has (M+N+1) rows and 4 columns. BASIS (K, 1) and BASIS (K, 2) contain respectively the row and column numbers of the Kth element.

The reference tables BASIS (K, 3) and BASIS (K, 4) give respectively the reference numbers of elements in the same row and column as the Kth element, following the rule that if there is no other element in a given row or column the reference is to the element itself.

Further reference tables ROWTAG (I) and COLTAG (J) give respectively the reference numbers of an element in row I and column J for quick entry to any part of the basis.

The allocation and cost associated with element K are stored in X(K) and C(K) respectively.

The routine also sets up appropriate fictitious and slack assignments to balance all problems where resources and requirements are unequal.

This routine with BEGIN forms the input phase of the program.

3. TREES

Subroutine TREES determines evaluators U(I), V(J) such that U(I)=V(J)=C(K), where BASIS (K, 1)=I and BASIS (K, 2)=J.

First the row number of the slack element referenced by COLTAG (N1) is found. Since this element is in slack, the associated U(I), V(N1) will be zero. By means of the table entries BASIS (K, 3) another element in row I is found and its column evaluator V(J) is calculated from V(J)=C(K)-U(I). Another element in column J is found and its row evaluator U(I) is found using U(I)=C(K)-V(J). No tests are carried out to find out whether an evaluator has been solved previously; this is bound to happen using this method because an element K which is the only one in a row or column refers to itself in BASIS (K, 3) or BASIS (K, 4).

After (M + N + 1) successive steps of solving for both U(I) and V(J) via the reference tables each evaluator will have been solved twice, but this apparent duplication of effort should more than offset the extra time (and instructions) involved in the testing necessary at each stage to ensure a single solution process for each evaluator.

4. GETWIJ

At each iteration a new element is introduced, replacing another element so that the overall cost will be reduced. Subroutine GETWIJ finds the element to which an allocation of one unit would result in the largest saving. For each resource/requirement pair, there exists a value w_ij=ui-vj<ci_j, which behaves like a potential function. It indicates by how much the cost will reduce per unit allocation introduced at (i, j). Clearly, for all basis elements w_ij=0, and the best element to be introduced into the basis is the one with the largest positive w_ij value, although the actual reduction in cost depends on how much of this element can be introduced into the basis. When the matrix is too large to be contained completely in core, and the time taken to read in all the costs block by block from work file to find the largest w_ij would be prohibitive, the first block of costs with associated row and column references is read into core and the largest w_ij stored.

Then, the next block is read in and the largest w_ij, compared with the previous best w_ij. If the previous w_ij is the larger, the associated element is chosen as the new basis element. Otherwise, further blocks are read in until the greatest w_ij of one block is less than that of the previous block, whose associated element is then chosen as the one to be introduced into the basis. By this method matrix scanning ceases as soon as the potentially best element in a given part of the matrix has been located. This dynamic form of scanning a large matrix has been found to produce the optimum solution in fewer iterations than a fixed length scan.

When no further positive w_ij exists, a switch is set to transfer control to the output routine.
in large transportation problems, the running time may exceed the available computer time, therefore a checkpoint tape records the current basis and associated reference data at suitable intervals by calling subroutine OUTLAY. This checkpoint solution can then be submitted as an initial basis for further optimisation. Further, the user can stipulate the number of matrix scans required should he wish to terminate the execution at a stage prior to final optimisation, and can alter the frequency of checkpoints by specifying a code in the initial control card.

5. OUTLAY

At the beginning of each new scan of the costs matrix, GETWIJ tests the number of iterations since the last checkpoint and, if sufficient, calls OUTLAY where the total cost of the current allocations is calculated in floating point exponent form and printed out with the number of iterations performed. The choice of this form of total, rather than the “exact” integer cost is because during the initial stages of solution a number of fictitious high cost entries may be present in the basis and these could cause integer overflow which would result in misleading total costs. When fictitious allocations are present an appropriate message is printed with the checkpoint information.

6. DSOLVE

The introduction of a new element into the existing basis causes the formation of a loop with some of the existing basis elements. Subroutine DSOLVE traces out this loop starting with a column movement from the new element recording the reference numbers of loop elements as they occur in a directed scan of the basis, (using the reference tables) and deletes those elements which continued scanning shows to be non-loop in character. The end of the loop is recognised when an element of the D array is found to be in the same row as the element being introduced into the basis.

7. MODIFY

When the new element is introduced into the basis, the allocations of the existing basis elements must be altered to restore the row and column totals to the correct values.

Subroutine MODIFY scans the odd entries in the array D, looped elements, D, to determine which has the lowest associated allocation. This element will leave the basis, its reference is stored in LOST and its allocation is stored in THETA. The element LOST is replaced in the basis by the new element with an associated allocation of THETA.

The allocation associated with the odd elements of the array D are all reduced by the amount THETA and those associated with the even elements of D are increased by THETA.

The reference tables are modified to incorporate the new element and remove references to the one removed.

Following this routine control is returned to the TREES routine for the next cycle of optimisation.

The routines TREES, GETWIJ, OUTLAY, DSOLVE and MODIFY form the optimising part of the program.
8. OUTPUT

Subroutine OUTPUT, which prints the final allocation in two forms, is the third main section of the program. The producers are listed in numerical order with their availabilities, unused stock and the relevant consumers and amounts supplied to them.

Secondly, the consumers are listed in numerical order with their relevant suppliers and allocations listed alongside.

The consumers supplied by each producer are sorted into numerical order before printing. Similarly, the suppliers of each consumer are sorted into numerical order before the second part of the output is printed.

If there are any outstanding requirements when OUTPUT is called, an exception report precedes the final allocation. The presence of outstanding requirements indicates one of three possible conditions,

(i) The fictitiously high cost is not sufficiently high for optimisation to remove it from the basis.

(ii) The constraints on the solution do not permit a feasible allocation of resources.

(iii) The user has specified that optimisation should take place even though resources are insufficient.

When the program is terminated after a predetermined number of matrix scans, optimisation being incomplete, the presence of outstanding requirements does not necessarily indicate one of the above-mentioned conditions.

At the end of this routine the true cost of the allocations made (excluding fictitious assignments) is computed in integer form. When the output allocations are infeasible due to the presence of a fictitious assignment, this value will differ from the hash total cost calculated at checkpoint time. (For mathematical consistency this hash total must decrease during optimisation, but the cost of actual assignments may increase as optimisation replaces fictitious assignments with true allocations.)

INPUT FORMAT

A FORTRAN binary master file must be supplied in the following format:-

1st record

\[ M, N, NCOSTS, (S(i), i=1, M), (P(j), j=1, N) \]

where

- \( M \) is the number of sources
- \( N \) is the number of sinks
- \( S(i) \) is the availability at source \( i \),
- \( P(j) \) is the requirement of sink \( j \), and
- \( NCOSTS \) is the number of records of cost data which follow.

\( M, N, NCOSTS, S \) and \( P \) are all 4-byte integers.

This is followed by NCOSTS records of the form

\[ LBLOCK, (C1(L), IXREF(L), JXREF(L), L=i, LBLOCK) \]

where IXREF(L), JXREF(L) are respectively source and sink reference numbers, in 2-byte integer form, and C1(L) is the associated cost, in 4-byte integer form.

A source/sink pair which is not included in one of these records is assumed to be prohibited. The entries must be sorted in ascending column order.

Each of the records in the master file is read as a list in the program and therefore requires RECFM=V in the job control. BLKSIZE must be at least equal to the length in bytes of the largest record plus 8 bytes for its control field. This will be the greater of

\[ 20 + 4(M+N) \] : length of first record plus control field

and

\[ 12 + 8(LBLOCK) \] : length of subsequent records plus control field.
CONTROL CARD

This card must contain the following information:

cc
1 A single digit indicating whether a work file prepared from the master file is available, and if a previous checkpoint is to be picked up.

0 indicates that the work file is to be written on to disk and the problem is to be solved from scratch.

1 indicates that the work file is to be written and that a partial solution is available on a previous checkpoint tape.

2 indicates that the work file is already on disk and a partial solution can be read off a checkpoint tape.

3 indicates that the work file is already on disk and the problem is to be solved from scratch.

2-5 A right-justified integer of up to 4 digits stating how many complete scans of the cost matrix are to be made. If it is intended to solve the problem completely in a single run it is suggested that this field be filled with 4 9's.

If this field is left blank, execution will be terminated after an initial basis has been set up. An output report will be printed, and the basis is written onto the checkpoint tape.

6-8 A three digit integer indicating which checkpoint from the previous run is to provide the starting basis.

9 A single digit indicating whether optimisation is to take place, even if the resources are not sufficient to satisfy the requirements. A zero or blank indicates that execution should be terminated if resources are insufficient, any other numeric character will cause an initial basis to be set up and optimisation to take place.

If the resources are sufficient to satisfy the requirements, a numeric entry other than zero in cc 9 will cause the optimisation process to be terminated as soon as a feasible allocation of resources is recognised.

10-13 A right-justified integer of up to four digits indicating the minimum number of iterations to be performed between checkpoints. If this field is left blank (or zero) the program assumes a value of 100.

HEADING CARD

This card enables the user to insert headings of his own choice into the output report to make it more meaningful.

cc
1-8 A left-justified name for the goods being allocated, e.g. COAL, GRAIN, OIL.

9-16 A right-justified name for the source of the goods, e.g. PRODUCER, FACTORY, MILL.

17-24 A right-justified name for the sinks supplied with the goods, e.g. CONSUMER, RETAILER, CUSTOMER.

25-26 A two digit integer stating the maximum number of lines to be printed on a single page of the stationery used for the output report.

27-36 A ten digit integer giving the fictitious cost assigned to any fictitious allocations which may be introduced into the basis. This figure must be greater than the highest cost in the costs matrix.

NOTE:

Some of the literal output produced by the program adds a final 5 to entry, and this can lead to output such as WORK 55.
CHECKPOINT TAPE

At suitable intervals during the run a checkpoint of the current stage of solution is taken on tape. This consists of a single record in FORTRAN binary form

NCOST1, IT, INHASH, X, C, BASIS, ROWTAG, COLTAG

where

NCOST1 is the total number of records of costs data plus unity
IT is the count of completed iterations
INHASH is the total requirement plus unity and is the availability of the fictitious source
X is the array of allocations in the basis
C is the array of associated unit costs
BASIS is the entire BASIS array
ROWTAG is the reference table of row entry points
COLTAG is the reference table of column entry points

These records are written sequentially each time OUTLAY is called, always at the beginning of a costs matrix scan, depending on the number of iterations completed since the last checkpoint was taken. This interval is subject to user specification with a default option of 100 iterations.

DESCRIPTION OF OUTPUT

TRANSUB/360 is designed to deal with all cases of assigning resources to requirements, even when the total resources and requirements are unequal, and further to restart a partially completed run from a checkpoint record on tape. Therefore a flexible set of output information has been supplied so that the user can analyze the run as simply as possible. Results have been presented to make them readable for management, highlighting exceptions where they occur. Thus the output from different runs can vary considerably in content.

1. SUBROUTINE BEGIN

For a run starting from scratch the initial output consists of a problem summary, which lists,

1. Number of sources, the words sources and sinks are replaced by the names supplied by the user in the heading card.
2. Number of sinks,
3. Number of records of costs data,
4. Total resources,
5. Total requirements,
6. Unused resources,

If the unused resources are negative this summary is followed by the message

RESOURCES INSUFFICIENT

enclosed in a box of asterisks.

In such cases a feasible solution of the transportation problem does not exist and the run terminates unless the user has specified that the stop option should be overridden. When this option is specified the program will optimise the allocation of available resources and the box of asterisks is enlarged to include the further message

STOP OPTION OVERRIDDEN
OPTIMISATION CONTINUES

When the run is a restart from a checkpoint, the output from this routine is the message

OPTIMISATION CONTINUING FROM CHECKPOINT NUMBER N
enclosed in a box of asterisks, where N is the number of the check-
point used as starting basis.

At the end of this routine, whenever optimisation is to continue,
headings for checkpoint records are printed on a new page.

2. SUBROUTINE OUTLAY

When the requirement for a checkpoint is recognised in GETW1J,
subroutine OUTLAY is called. A checkpoint is taken of all details
necessary for a restart and a message is printed on a line consisting of

1. Checkpoint number,

2. Number of completed scans of costs matrix,

3. Iteration count,

4. Hash cost. (This is expressed in floating point exponent form
   and is a true cost when the solution is feasible. However, if fictitious allocations are present this
   figure is inflated by the artificially high cost and is not a true guide to the cost of allocations. If
   the hash cost does not decrease monotonically as the iteration count increases, the process is out of
   control).

5. When the current basis contains fictitious allocations a further
   message

   SOME REQUIREMENTS ARE NOT SATISFIED

   is also printed at the right hand end of the line.

Checkpoints are always taken at the beginning and end of a run.

3. SUBROUTINE OUTPUT

When the run is completed the reason for termination is printed in
a box of asterisks at the beginning of a new page.

There are two main classes of message:

(a) When the program has optimised as much as possible the line

   OPTIMISATION HAS BEEN COMPLETED

   is printed.

If some requirements are still outstanding a further line

HOWEVER, IT HAS NOT BEEN POSSIBLE TO SATISFY ALL REQUIREMENTS
   is printed.

Further, if some resources are still unallocated whilst requirements
are outstanding a further line

AND THE CONSTRAINTS DO NOT ALLOW ALLOCATION OF ALL RESOURCES

ends this message.

(b) When the program is terminated by user instruction, either
after a given number of costs matrix scans or when the check-
point first recognised a feasible solution, the message consists of
the single line

OPTIMISATION PROCESS TERMINATED BY USER INSTRUCTION AFTER N SCANS
   OF COST MATRIX

   where N is either the user specified number of scans, or the
   number of scans to achieve the first feasible checkpoited
   solution.

The next output line presents the true cost of the feasible
allocations, (suppressing fictitious assignments where necessary) in
the form

THE COST OF THE LISTED ALLOCATIONS AFTER N ITERATIONS IS E POUNDS

   where N is the iteration count and POUNDS is the true cost
   in integer form.

Where the solution has either outstanding requirements or
unallocated resources are present in the basis an appropriate
exception report appears, starting at the head of a new page
in each case.

When requirements are outstanding, the report heading is

THE FOLLOWING REQUIREMENTS ARE OUTSTANDING

Sinkname  ORDER  DEFICIT  (where sinkname is obtained
   from the user supplied heading
card)

followed by records giving the sink number and the amount in
deficit, each on a separate line with double spacing.

The final record is

\[
\text{TOTAL} \quad N
\]

where \( N \) is the total deficit.

For unallocated resources, the report heading is

THE FOLLOWING RESOURCES HAVE NOT BEEN ALLOCATED

Source name \( \text{SURPLUS} \) (where source name is obtained from the user supplied heading card)

followed by records giving the source number and the amount of surplus, each on a separate line with double spacing.

The final record is

\[
\text{TOTAL} \quad N
\]

where \( N \) is the total surplus.

These exception reports highlight cases where surpluses are building up at a supply point and cannot be used to alleviate shortages at given delivery points.

The final output consists of two detailed reports, both starting on a new page with the heading

THE ALLOCATIONS OF XXX

where XXX is the commodity name specified by the user in the heading card.

The two reports, which deal respectively with allocation from source to sink and to sink from source, have a second line

BY XXX

where XXX is respectively the user-specified source name or sink name.

A further heading, which is repeated at the head of each new output page has respective forms

\[
\text{Sourcename TOTAL SURPLUS, followed by, sinkname AMOUNT (repeated 5 times)}
\]

\[
\text{Sinkname TOTAL DEFICIT, followed by, sourcename AMOUNT (repeated 5 times)}
\]

For the first report a separate record follows for each source in numeric order detailing the total availability and surplus and allocations to the sinks, sorted in ascending order of sink number. If there are more than five allocations from any source the record is continued single spaced until allocations from the source have been specified. There is always a blank line between the end of one record and the beginning of the next. Zero allocations are suppressed, except when a source fills no requirements in which case the first allocation is of amount zero to sink zero.

The second report following an analogous form interchanging the roles of sources and sinks.

The final message is a single centred line.

END OF JOB

These cards contain the unit cost and associated source and sink references of the five elements in the costs matrix. Up to 4 entries can appear on one card.

- **cc**

1 - 7  Unit cost, (The cost of transporting 1 unit from source i to sink j).

8 - 12 The associated source reference.

13 - 17 The associated sink reference.

Columns 18 - 34, 35 - 51 and 52 - 68 may contain further similar entries.

Column 72 contains a single digit indicating how many cost entries appear on the card.

5. End-of-file card.

This card contains a 9 in column 72 and must be otherwise blank. The end-of-file card must follow the last costs data card.

To run the sample problem, the user must first compile, link-edit and execute the supplied file-preparing program, using file 3 of the supplied tape as data, before running TRANSUB/360. Details of Job Control records necessary for running the sample problem are given in the Operating Instruction section.

---

**MAGNETIC TAPE KEY**

This volume contains three Files and three Tape Mt's arranged as follows:

- **File 1** FORTRAN Source Deck - TRANSUB/360 VERSION 2.0
  - EBCDIC
  - Ident. BKAC in cc 73-76
  - Sequence 0010 to 7650 inclusive in cc 77-80 at intervals of 10; 765 cards
  - 765 card images blocked 25 per record
  - 31 records of 2000 characters
  - T/M

- **File 2** FORTRAN Source Deck - A program to prepare input dataset from spread cards
  - EBCDIC
  - Ident. TOTAPE in cc 73-78
  - Sequence 01 to 64 inclusive in cc 79-80; 64 cards
  - 64 card images blocked 25 per record
  - 3 records of 2000 characters
  - T/M

- **File 3** Input data cards - for processing by TOTAPE before use in TRANSUB/360
  - EBCDIC
  - Ident. AB in cc 74-75, LP in cc 79-80
  - Sequence 010-910 inclusive in cc 76-78 at intervals of 10; 81 cards
  - 81 card images blocked 25 per record
  - 4 records of 2000 characters
  - T/M
PROGRAM MODIFICATION

The DIMENSION statements in MAIN restrict the number of sources to 500, and the number of sinks to 1500 and the maximum length of a cost data record to 400 entries. If any of these limits is to be exceeded, the DIMENSION statements and the initialisation statement I NSIZE=400 must be changed. No alteration is required in any other routine, as all array dimensions in subroutines are defined at object-time through subroutine arguments.

The current version of TRANSUB/360 assumes that the costs matrix is less than 50% active and stores row and column references to save the time that would be spent in scanning and testing prohibited allocations. When the matrix is more than 50% active the GETWJ and SETUP routines will execute more quickly if complete columns of costs are examined in which case the arrays IXREF and JXREF can be omitted and column scans indexed directly.

TRANSUB/360 can be overlaid without any modification to the source deck. The program divides into 3 overlays, input, optimisation and output. Overlaying releases more than 11K bytes of core for additional data storage. Detailed instructions for overlaying appear in the Operating Instructions section.

If a user requires the solution to problems whose size always lies within certain fixed dimensions, execution time may be improved by employing fixed array sizes throughout all routines. The arrays should be placed in COMMON, and all subroutine arguments removed.

SAMPLE PROBLEM

The FORTRAN program which prepares the master file is designed to accept data from spread cards and can be used to create the master file for a problem of any size.

The program requires 5 types of data cards:

1. Control Card.

This card must be the first data card, and contains 3 fields.

cc

1 - 5 The number of sources.
6 - 10 The number of sinks.
11 - 15 The maximum number of cost matrix entries which TRANSUB/360 can accept in a single record.

2. Source data cards.

These cards follow the control card and contain the source availabilities, each as a six digit integer, spread 12 to a card, in cc 1 - 72.

3. Sink data cards.

The sink data cards have the same format as the source data cards.

These cards contain the unit cost and associated source and sink references of the live elements in the casts matrix. Up to 4 entries can appear on one card.

cc

1 - 7 Unit cost. (The cost of transporting 1 unit from source i to sink j).

8 - 12 The associated source reference.

13 - 17 The associated sink reference.

Columns 18 - 34, 35 - 51 and 52 - 68 may contain further similar entries.

Column 72 contains a single digit indicating how many cost entries appear on the card.

5. End-of-file card.

This card contains a 9 in column 72 and must be otherwise blank. The end-of-file card must follow the last costs data card.

To run the sample problem, the user must first compile, link-edit and execute the supplied file-preparing program, using file 3 of the supplied tape as data, before running TRANSUB/360. Details of Job Control records necessary for running the sample problem are given in the Operating Instruction section.

OVERLAYING

TRANSUB/360 can be overlaid without modification to the program, which divides into 3 overlay segments:

1. Input, consisting of Subroutines BEGIN and SETUP

2. Optimisation, consisting of Subroutines TREES, GETWIJ, OUTLAY, DSOLVE and MODIFY

3. Output, consisting of Subroutine OUTPUT.

The root segment consists of the calling program MAIN. Implementation of these overlays releases more than 11K bytes of core storage for additional data storage and approximately another 50K may be added to the sources plus sinks total. Figure 2 shows how to compile, link-edit, overlay and execute TRANSUB/360.

ACKNOWLEDGEMENTS

The algorithm is based on a technique described in the following paper:


This program was written with the assistance of Alec Cassells, an IBM scholar. I am pleased to record my appreciation of the effort which he made, particularly for his contribution to writing the original output routine; implementing many of the modifications in this second version and his help in producing this documentation.

Basil C. Kahan.
VITA

Ronald Gary Goodwin

Candidate for the degree of

Master of Science

Report: Utah State University Mathematical Programming Package

Major Field: Applied Statistics

Biographical Information:

Personal Data: Born at Idaho Falls, Idaho, September 15, 1940 son of Ronald G. and Inez Arave Goodwin; married Barbara Johnson August 27, 1965; three children—Wendy K., Melanie Sue, and Ronald J.

Education: Attended elementary school in Taylor, Idaho; graduated from Shelley High School in 1958; received the Bachelor of Science degree from Brigham Young University, with major in Statistics and Minor in Mathematic, in 1966; did graduate work in Applied Statistics at Utah State University, 1970-71; completed requirements for Master of Science degree, specializing in Applied Statistics and Computer Science, at Utah State University in 1971.