AN ALGORITHM FOR THE SOLUTION OF LINEAR PROGRAMMING PROBLEMS USING STEP-BY-STEP ADDITION OF CONSTRAINTS

MICHAEL ROBERT FENN

[redacted]

AN ALGORITHM FOR THE SOLUTION OF LINEAR PROGRAMMING PROBLEMS USING STEP-BY-STEP ADDITION OF CONSTRAINTS

by

Michael Robert Fenn
Lieutenant, United States Navy
B. S., Naval Academy, 1960

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
December 1967

ABSTRACT

As linear programming techniques find applications in more diverse fields, the problem of solution time becomes increasingly important. A variation of the revised simplex algorithm, in which the constraints are added in a step-by-step fashion, is investigated as a potentially faster solution technique. A computational procedure, coded for the IBM 360 computer, is developed to compare this algorithm with the standard two-phase revised simplex algorithm. A limited number of problems, including several randomly generated problems, is solved by each of the two methods. The resulting comparison of solution times indicates that a significant improvement is obtained by the use of the procedure of step-by-step addition of constraints.

TABLE OF CONTENTS

LIST OF TABLES

1. Introduction

Increased utilization of linear programming techniques has led to the formulation of problems of sufficient size to tax the storage capabilities of many computers now in use. The problem of storage capacity may be alleviated by auxiliary storage, with the accompanied requirement of access delays which increase solution time. Even with sufficient storage and a coded procedure large enough to accommodate a problem, the computation time may be such that solution costs approach a budget limitation. For example, the CEIR LP/90 program for the IBM 7090 computer, which can accommodate 512 constraints and an unlimited number of variables, or the Philco Corporation LP 2000 System for the S-2000 computer, which can accommodate 2500 constraints and an unlimited number of variables, might take several hours to solve a large problem at a cost of several thousand dollars.

Improvement in the solution time might be accomplished by improvement of the hardware or by an improvement in the mathematical procedure. The hardware improvements, which are becoming available in the newer computers such as the IBM 360, are increased computation speeds and faster access to external storage.

Solution procedure improvement has been attempted by several methods. Some attempts are being made to take advantage of special structuring of the problem. These techniques are directed toward partitioning or decomposing the problems into manageable sub-problems. Another approach to improved solution time is the Primal-Dual algorithm which was developed by Dantzig. [3]

This project has been directed towards another possible method for improving solution procedures. An appraisal is made of the step-by-step addition of constraints to the two-phase revised simplex procedure as a possible solution technique which would serve to improve solution times. This method, called here the "step-by-step addition of constraints (SSAC)," exploits the advantage of obtaining a rapid solution to a small sub-problem and then moving from one optimal solution to an optimal solution of a slightly larger sub-problem as the remaining constraint equations are added one at a time. In the earlier stages of the solution procedure, the size of the matrices used in the iteration procedure would be relatively small, compared to the complete problem. By exploiting the advantage of multiplying smaller-dimensioned matrices, with the attendant shorter computational times, a shorter overall solution time might be achieved in spite of the fact that a greater number of iterations would be required.

2. Notation

The notation used throughout this paper was chosen to correspond to the notation most frequently used in linear programming texts.

(1) Upper case letters represent matrices.

(2) Lower case letters represent column or row vectors.

(3) Subscripted lower case letters represent elements of row or column vectors.

(4) Tableau notation, as illustrated below, is similar to that used by Dantzig. [3]

z	w	x_5	x_1	x_2	x_3	x_4	b		
1			-2	0	-4	0	0		"z"
	1		-3	-4	-6	-1	-24		"w"
		1	3	4	6	1	24	$\theta = 24/6 = 4$	"R1"

The columns corresponding to basis vectors are indicated by a dot. The pivot column (or row in the case of the dual simplex algorithm) is indicated by an arrow. The pivot element, as determined by the appropriate minimum θ criterion, is indicated by circling the element, and the basis variable which is then to be driven out is indicated by circling the associated dot.

The tableau rows are referred to as the "z", "w" or Ri" row where i is an integer corresponding to the sequence in which the constraints appear. That is, "R1" refers to the first constraint, "R2" refers to the second constraint and so on.

Specific notation which will receive repeated use in this paper includes:

m number of constraint equations

n number of variables

A $m \times n$ matrix of coefficients of the constraint equations, having elements a_{ij}

B $m \times m$ matrix of basis vectors

B^{-1} inverse of the basis

P_j the m-dimensional vectors which make up the B matrix

x an n-dimensional column vector having elements x_j

c the n-element row vector of cost coefficients having elements c_j

b the m-element column vector of the right-hand side of the constraint equations (requirements vector), having elements b_i

3. Formulation of the Problem

The general linear programming problem is stated as:

Maximize

$$cx$$

subject to:

$$Ax = b,$$

and,

$$x \geq 0.$$

For the algorithm to be investigated, we define

$$z = cx.$$

We can then rewrite the problem as:

Maximize

$$z,$$

subject to:

$$Ax = b,$$

$$z - cx = 0,$$

and,

$$x \geq 0.$$

To obtain an identity matrix to begin the two-phase revised simplex procedure we add artificial variables $x_{n+1}, x_{n+2}, \ldots, x_{n+m}$ to the constraint equation with $x_{n+j} \geq 0$ for $j = 1,\ldots,m$.

If we define

$$w = -x_{n+1} - x_{n+2} - \ldots - x_{n+m}$$

and then maximize w, we will drive the artificial variables out of the basis and either obtain an initial basic feasible solution or an indication that the problem is infeasible. The process associated with maximizing w is usually referred to as Phase I.

During the second phase (Phase II), the problem is stated in the following form:

Maximize

$$z$$

subject to:

$$\begin{array}{lllllll}
a_{11}x_1 + a_{12}x_2 + \cdots\cdots + a_{1n}x_n & + x_{n+1} & & & & & = b_1 \\
a_{21}x_1 + a_{22}x_2 + \cdots\cdots + a_{2n}x_n & & + x_{n+2} & & & & = b_2 \\
\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots & & & & & & \\
a_{m1}x_1 + a_{m2}x_2 + \cdots\cdots + a_{mn}x_n & & & + x_{n+m} & & & = b_m \\
-c_1x_1 - c_2x_2 - \cdots\cdots - c_nx_n & & & & + z & & = 0 \\
 & & & x_{n+1}+x_{n+2}+\cdots+x_{n+m}+ & & w & = 0
\end{array}$$

and,

$$x_j \geq 0 \quad \text{for } j = 1, 2, \ldots, m.$$

Because of the w equation we do not have an identity matrix to use as a starting basis. Therefore, we subtract from that equation each of the other equations, as appropriate, to remove the $x_{n+1}, \ldots, x_{n+m}$ variables. We obtain the following form for the w equation:

$$w + a_{m+2,1}x_1 + a_{m+2,2}x_2 + \cdots + a_{m+2,n}x_n = b_{m+2}$$

where,

$$a_{m+2,j} = - \sum_{i=1}^{m} a_{ij} \quad j = 1, \ldots, n$$

and,

$$b_{m+2} = - \sum_{i=1}^{m} b_i$$

for those artificial variables, i, having non-zero prices. Our problem is now in a form such that we have an identity matrix to use as our starting basis.

Maximize

$$z,$$

subject to:

$$
\begin{array}{llllllll}
a_{11}x_1 + a_{12}x_2 + \cdots\cdots + a_{1n}x_n & + x_{n+1} & & & & & = b_1 \\
a_{21}x_1 + a_{22}x_2 + \cdots\cdots + a_{2n}x_n & & + x_{n+2} & & & & = b_2 \\
\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots & & & & & & \\
a_{m1}x_1 + a_{m2}x_2 + \cdots\cdots + a_{mn}x_n & & & + x_{n+m} & & & = b_m \\
-c_1x_1 - c_2\,x_2 - \cdots\cdots - c_n\,x_n & & & & + z & & = 0 \\
a_{m+2,1}x_1 + a_{m+2,2}x_2 + \cdots + a_{m+2,n}x_n & & & & & + w & = b_{m+2},
\end{array}
$$

and,

$$x \geq 0 \quad \text{for } j = 1,2,\ \ldots,\ n+m.$$

If we let $a_{m+1,j} = -c_j$, the problem resolves into the three matrices used for computations in the revised simplex procedure:

$$
A = \begin{bmatrix}
a_{11} & a_{12} & \cdots\cdots\cdots & a_{1n} \\
a_{21} & a_{22} & \cdots\cdots\cdots & a_{2n} \\
\cdot & \cdot & \cdots\cdots & \cdot \\
a_{m1} & a_{m2} & \cdots\cdots\cdots & a_{mn} \\
a_{m+1,1} & \cdots & \cdots\cdots\cdots & a_{m+1,n} \\
a_{m+2,1} & \cdots & \cdots\cdots\cdots & a_{m+2,n}
\end{bmatrix}
$$

$$
B^{-1} = \begin{bmatrix}
1 & 0 & \cdots\cdots & 0 \\
0 & 1 & & \cdot \\
\cdot & & \cdot & \cdot \\
0 & & & 1
\end{bmatrix}
$$, an identity matrix, and

$$b = \begin{bmatrix} b_1 \\ b_2 \\ \cdot \\ b_m \\ 0 \\ b_{m+2} \end{bmatrix} .$$

For simplification in handling the matrices when coded in FORTRAN, they will be re-arranged so that the objective function and the modified w equation appear in the first two rows rather than the last two:

$$A = \begin{bmatrix} a_{m+1,1} & \cdots\cdots\cdots & a_{m+1,n} \\ a_{m+2,1} & \cdots\cdots\cdots & a_{m+2,n} \\ a_{11} & \cdots\cdots\cdots & a_{1n} \\ a_{21} & \cdots\cdots\cdots & a_{2n} \\ \cdot & \cdot\;\cdot\;\cdot\;\cdot\;\cdot\;\cdot & \cdot \\ a_{m1} & \cdots\cdots\cdots & a_{mn} \end{bmatrix}$$

$$B^{-1} = I, \text{ and } \quad b = \begin{bmatrix} 0 \\ b_{m+2} \\ b_1 \\ \cdot \\ b_m \end{bmatrix} .$$

In the revised simplex procedure, the original A matrix and b vector of the full starting tableau ("original" tableau) are used at each iteration with the inverse of the current basis matrix to determine certain unknown elements of the current tableau.

When a constraint is added, the original A matrix and b vector are changed. The size of each is increased by one row, and the A matrix is increased by one column. The basis matrix is therefore increased in size by the addition of a row vector, γ , corresponding to the elements of the new constraint which are in the current basis. It is, at the same time, increased by a column vector corresponding to a new artificial variable.

The new (m+1) x (m+1) basis is:

$$\bar{B} = \begin{bmatrix} B & 0 \\ \gamma & 1 \end{bmatrix}$$

By partitioning of the $\bar{B}$ matrix, the new inverse is readily obtained:

$$\bar{B}^{-1} = \begin{bmatrix} B^{-1} & 0 \\ -\gamma B^{-1} & 1 \end{bmatrix}$$

Upon adding a new constraint to a problem, one of three cases will result:

(1) the new constraint is satisfied;

(2) the new constraint is not satisfied and the value of the additional artificial variable is positive in the basic solution;

(3) the new constraint is not satisfied and the value of the additional artificial variable is negative in the basic solution.

In the first case, the new constraint has no effect and the optimal solution to the entire original problem has been obtained if no further constraints are to be added. The optimal value of z is not affected by the new constraint.

In the second case, the two-phase revised simplex procedure is used to first drive out the artificial variable and then to maximize z.

In the third case, if the new artificial variable is assigned a zero cost in the z equation, the dual simplex procedure can then be applied to the infeasible primal to obtain an optimal solution to the dual, and hence an optimal solution to the primal. The artificial variable is considered to be a legitimate variable of the original problem in the dual simplex approach. As a consequence, it never appears in the w equation. For convenience, we will carry along the "w" row of the tableau, for use when later constraints are added.

When a solution is obtained which satisfies the added constraint, the optimal solution to the original problem has been found if there are no new constraints to be added.

The addition of a new constraint to a linear programming will have the effect of either decreasing the previously obtained maximum solution or leaving it unchanged. That is, letting the subscript on z denote the number of constraints,

$$\max z_{m+1} \leq \max z_m .$$

Solution of a linear programming problem by step-by-step addition of constraints may result in one or more unbounded solutions to the subproblems if the initial constraints have fewer variables than are included in the objective function. This causes no difficulty, however, as the addition of one or more new constraints will serve to place bounds on the problem, unless the original problem is unbounded.

The method of step-by-step addition of constraints has the advantage that an infeasible solution at an early stage will determine that the original problem has no feasible solution, and no new constraints need be added. The solution procedure is then terminated.

4. Sample Problem

Consider, as an example, the problem:

maximize

$$z = 2x_1 + 4x_3$$

subject to:

$$3x_1 + 4x_2 + 6x_3 \leq 24$$

$$4x_1 + 3x_2 + 12x_3 \leq 24$$

$$x_1 + x_2 + 4x_3 = 8$$

and,

$$x_j \geq 0 \quad \text{for } j = 1, 2, 3.$$

Adding slack variables and rewriting the problem, we have

maximize

$$z,$$

subject to:

$$\begin{aligned}
z - 2x_1 \phantom{{}+4x_2} - 4x_3 &= 0 \\
3x_1 + 4x_2 + 6x_3 + x_4 &= 24 \\
4x_1 + 3x_2 + 12X_3 + x_5 &= 24 \\
x_1 + x_2 + 4x_3 &= 8
\end{aligned}$$

and,

$$x_j \geq 0 \quad \text{for } j = 1, .., 5$$

For our solution by the SSAC procedure, the modified w equation for the first sub-problem will be:

$$w - 3x_1 - 4x_2 - 6x_3 - x_4 = -24$$

because the original w equation is

$$w + x_6 = 0,$$

where x_6 is the artificial variable introduced into "R1".

The initial tableau will be:

	z	w	x_6	x_1	x_2	x_3	x_4	x_5	b
"z"	1			-2	0	-4	0	0	0
"w"		1		-3	-4	-6	-1	0	-24
"R1"			1	3	4	6	1	0	24

$\theta = 24/6 = 4$

In Phase I we will maximize w. We choose the most negative value in the modified w equation and pivot on the element which meets the minimum θ criterion according to the usual (primal) simplex procedure. Note that in the first tableau the first pivot will always be such as to drive out the first artificial variable.

We pivot using the product form of the inverse. [5] The η vector corresponding to the x_3 column will be

$$\eta = \begin{bmatrix} 4/6 \\ 6/6 \\ 1/6 \end{bmatrix} = \begin{bmatrix} 2/3 \\ 1 \\ 1/6 \end{bmatrix}$$

and,

$$E = \begin{bmatrix} 1 & 0 & 2/3 \\ 0 & 1 & 1 \\ 0 & 0 & 1/6 \end{bmatrix}$$

The new inverse is determined from $\bar{B}^{-1} = EB^{-1}$ as follows:

$$\bar{B}^{-1} = \begin{bmatrix} 1 & 0 & 2/3 \\ 0 & 1 & 1 \\ 0 & 0 & 1/6 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 2/3 \\ 0 & 0 & 0 \\ 0 & 0 & 1/6 \end{bmatrix}$$

The new tableau is now

z	w	x_6	x_1	x_2	x_3	x_4	x_5	b
1	0	2/3	0	8/3	0	2/3	0	16
0	1	1	0	0	0	0	0	0
0	0	1/6			1			4

where x_3 is now a basic variable; all values of $z_j - c_j \geq 0$, and we have achieved the "first" optimal solution.

To add a new constraint, the "original" tableau is augmented by one row and the column vector for x_7, where x_7 is the artificial variable associated with "R2". Ignoring the "w" row for the time being, the augmented "original" tableau will be

z	w	x_6	x_7	x_1	x_2	x_3	x_4	x_5	b
1				-2	0	-4	0	0	0
		1		3	4	6	1	0	24
			1	4	3	12	0	1	24

We compute the γ vector by determining from the previous solutions which vectors are in the basis. Observe that the coefficient vectors associated with z and w will always be in the basis since we are maximizing z and w. The first two elements of the γ vector will therefore always be zeros. The third element in this case will be the coefficient of x_3 in the new constraint.

$$\gamma = \begin{bmatrix} 0 & 0 & 12 \end{bmatrix}$$

The augmented inverse of the basis is determined by partitioning:

$$\bar{B}^{-1} = \begin{bmatrix} B^{-1} & 0 \\ -\gamma B^{-1} & 1 \end{bmatrix}$$

The product $-\gamma B^{-1}$, in this case, is

$$\begin{bmatrix} 0 & 0 & -12 \end{bmatrix} \begin{bmatrix} 1 & 0 & 2/3 \\ 0 & 1 & 1 \\ 0 & 0 & 1/6 \end{bmatrix} = \begin{bmatrix} 0 & 0 & -2 \end{bmatrix},$$

and the new inverse becomes

$$\bar{B}^{-1} = \begin{bmatrix} 1 & 0 & 2/3 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1/6 & 0 \\ 0 & 0 & -2 & 1 \end{bmatrix}.$$

Our next step is to determine if the value of the new artificial variable is positive or negative in the basic solution. We find

$$x_7 = 24 - 12x_3 = -24.$$

Because it is negative, our procedure tells us to assign $c_7 = 0$ to the new artificial variable, x_7, and use the dual simplex algorithm to drive out the artificial variable.

z	w	x_6	x_7	x_1	x_2	x_3	x_4	x_5	b
1	0	2/3	0	0	8/3	0	2/3	0	
0	1	1	0	0					
0	0	1/6	0	1/2					
0	0	-2	1	-2	-5	0	-2	1	-24

We pivot on the row having the most negative b_i element in the dual simplex method. As a consequence, the new artificial variable will be dropped as a basic variable.

Our pivot element is determined for $a_{4j} < 0$.

$$\theta = \text{Min}\left[-a_{1j}/a_{4j}\right] = \text{Min}\left[0,\ 8/15,\ 2/6\right] = 0.$$

The new inverse is determined by the product form of the inverse.

$$\eta = \begin{bmatrix} 0 \\ 0 \\ 1/4 \\ -1/2 \end{bmatrix} \qquad E = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1/4 \\ 0 & 0 & 0 & -1/2 \end{bmatrix}$$

The current tableau becomes:

z	w	x_6	x_7	x_1	x_2	x_3	x_4	x_5	b
1	0	2/3	0	0	8/3	0	2/3	0	
0	1	1	0				0		
0	0	-1/3	1/4	0	-7/12	1	-1/3	1/4	-2 ←
0	0	1	-1/2	1			1		12

$$\theta = \text{Min}\left[32/7,\ 2\right] = 2$$

Since the up-dated requirements vector, b, still has a negative component, the dual simplex algorithm must again be applied. Pivoting on the x_4 column gives:

z	w	x_6	x_7	x_1	x_2	x_3	x_4	x_5	b
1	0	0	1/2	0	3/2	2	0	1/2	12
0	1	1	0						0
0	0	1	-3/4				1		6
0	0	0	1/4	1					6

Since all elements of b are now non-negative, our solution is optimal. Notice that the "w" row is retained but not operated on in the use of the dual simplex approach.

Adding the third constraint, the augmented "original" tableau becomes

z	w	x_6	x_7	x_8	x_1	x_2	x_3	x_4	x_5	b
1					-2	0	-4	0	0	0
		1			3	4	6	1	0	24
			1		4	3	12	0	1	24
				1	1	1	4	0	0	8

Again, the "w" row is ignored until after we determine if the new artificial variable, x_8, is positive or negative in the basic solution. After bringing in the new constraint,

$$\gamma = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}$$

and,

$$-\gamma B^{-1} = \begin{bmatrix} 0 & 0 & 0 & -1/4 \end{bmatrix} .$$

The value of the new artificial variable, x_8, for constraint "R3", is

$$x_8 = 8 - x_1 = 2.$$

Since the value of x_8 is positive in the basic solution, we can use Phase I of the revised simplex procedure to move to the optimal solution for the complete "original" problem. The augmented "w" equation in the "original" tableau will be

z	w	x_6	x_7	x_8	x_1	x_2	x_3	x_4	x_5	b
	1				-4	-5	-10	-1	0	-32

and the current tableau becomes

z	w	x_6	x_7	x_8	x_1	x_2	x_3	x_4	x_5	b
1	0	0	1/2	0			2			
0	1	1	0	0	-1	-1	-4	0	0	
0	0	1	-3/4	0			-3			6
0	0	0	1/4	0			3			6
0	0	0	-1/4	1			1			2

$\theta = \text{Min}\left[6/3,\ 2/1\right] = 2$

Pivoting on the appropriate element of the column having the most negative value in the "w" row, we bring x_3 into the basis. The resulting tableau is then

z	w	x_6	x_7	x_8	x_1	x_2	x_3	x_4	x_5	b
1	0	0	1	-2	0					
0	1	1	-1	4	-1	0	0	0	-1	
0	0	1	-3/2	3	0					
0	0	0	1	-3	1					0
0	0	0	-1/4	1	0					

Since all of the coefficients of the "w" row are not non-negative, we again pivot on the column having the most negative element, or in the case of ties, the left-handed one of the tied columns. Pivoting on the x_1 column, we bring x_1 into the basis.

After this iteration, we find that all elements of the "w" row are zero and at the same time, all $z_j - c_j \geq 0$ so we have completed both Phase I and Phase II. The final tableau is:

z	w	x_6	x_7	x_8	x_1	x_2	x_3	x_4	x_5	b
1	0	0	1	-2	0	1	0	0	1	8
0	1	1	0	1	0	0	0	0	0	0
0	0	1	-3/2	3				1		12
0	0	0	1	-3	1					0
0	0	0	-1/4	1			1			2

We have now arrived at the optimal solution of the "original" problem:

$$z = 8$$

$$x_1 = 0$$

$$x_3 = 2$$

$$x_4 = 12$$

and all other $x_j = 0$.

5. Programming Technique

As a means of testing the feasibility of solving general linear programming problems by the SSAC method, the solution technique was coded in FORTRAN IVg for use on the IBM 360/67 computer.

One subroutine was designed which would carry out the solution procedure by either the two-phase revised simplex method or the dual simplex method as appropriate. This same subroutine was used to solve the problems by both the standard revised simplex procedure and by the SSAC procedure. By using the same subroutine for both methods, it was hoped that any bias which might result from programming technique could be avoided. A driving routine was designed which would first solve a problem by the revised simplex method and then re-solve the same problem using the SSAC method. The two sections of the program were then timed.[1] Sections of the program not germane to the method being investigated, such as the reading in and printing out of data, were not included in the timing. The number of iterations required for solution by each method was tabulated.

---

[1]The timing routine was developed by Lt E. A. Singer, a student at the Naval Postgraduate School.

6. Efficiency of the Algorithm

Several small problems, for which hand solutions could easily be obtained, were used for preliminary testing and debugging of the program. A number of larger problems were then solved to obtain a limited experimental verification of the new procedure. The problems were chosen from three categories of problem types; mixing problems, transportation problems, (including a network problem and a transshipment problem), and caterer problems. Results based on this preliminary comparison, as shown in Table I, were inconclusive.

In order to obtain the solutions to a large number of problems, and as a means of avoiding the considerable time and effort required to input data by hand, a routine was designed which would generate random problems. This routine utilized a random-number generator to generate elements for the A and b matrices. To insure the existence of a bounded optimal feasible solution, the problems were formulated as:

Maximize

$$z = cx$$

subject to:

$$Ax \geq b$$

and,

$$c_j \leq 0$$

$$x_j \geq 0 \qquad \text{for } j = 1, \ldots, n$$

$$a_i \geq 0$$

$$b_i \geq 0 \qquad \text{for } i = 1, \ldots, m$$

The problems were generated to have 70 variables, including slack variables, and 20 constraints. The distribution of the coefficients was uniform over the following intervals:

a_{ij} , uniform (0.1);

b_i , uniform (0,5);

c_j , uniform (-1,0).

A total of forty-six problems were solved using this method of problem generation. A tabulation of solution times for these random problems is given in Appendix I.

A comparison of the solution times of these forty-six problems shows that the method of step-by-step addition of constraints was faster in thirty-four cases. The mean solution time for the SSAC procedure was 3.94 seconds faster than the mean solution time by the revised simplex procedure. By applying an appropriate statistical test to the solution results, it was determined that, with 95% confidence, the mean difference in solution times for the two methods is not less than 2.27 seconds.[2] Therefore, it can be concluded that the method of SSAC is significantly faster than the revised simplex method.

---

[2] Detailed computations for the t-test and computation of lower confidence limits on the mean solution time difference are given in Appendix I.

TABLE I

SOLUTION TIME RESULTS OF PROBLEMS USED IN PRELIMINARY INVESTIGATION

Problem	Revised Simplex (RS)		Add. of Constr. (SSAC)		Time Ratio
	Iterations	Time (sec.)	Iterations	Time (sec.)	SSAC/ R.S.
Mixing Problems					
Waugh's Diet [4]	9	.106	15	.188	1.17
Gasoline Blend [1]	10	.766	29	1.446	1.88
Transportation Problems					
3 by 33 tableau	85	118.829	89	51.352	.43
7 by 7 tableau	41	6.398	38	4.071	.64
3 by 5 tableau	18	.852	22	.700	.82
3 by 4 tableau	13	.446	21	.479	1.07
Transshipment [3]	18	4.223	20	3.117	.74
Network Flow	17	3.318	24	2.507	.75
Caterer Problems					
Wardroom Napkin [1]	35	10.768	39	6.177	.57
Hadley Napkin [5]	10	.745	25	.918	1.23

7. Concluding Remarks

It is important to emphasize that the solution time is a function of the way the computer program is written. In the code employed, a full set of artificial variables was generated for each problem. An algorithm which takes advantage of existing slack variables for an initial feasible solution might well prove to be faster than the present program.

Round-off error and exponent underflow can greatly affect the solution technique. The use of the product form of the inverse in the pivoting operation relieves this situation somewhat. In the experimental algorithm a routine was employed which set any element having an absolute value less than .0001 equal to zero. A better method might be to use double precision mode for computations and then allow values smaller than .0001 to be carried along in the solution.

By using a single subroutine for the simplex iteration procedure in both methods of problem solution, it was hoped that any inconsistency due to programming technique could be kept to a minimum. That is, necessary computations for the iteration procedure were carried out in the same sequence for both methods of problem solution.

It is recognized that the problems selected are not necessarily a representative sampling of linear programming problems. The manual input problems were selected primarily because they were large enough to allow the step-by-step addition of constraints to be demonstrated, yet small enough to handle conveniently as data inputs. The randomly generated problems were considered to be of a size which was large enough to effectively test the procedure, subject to available computer time.

The results of this limited number of tests indicate that the SSAC method is worthy of further investigation. This further effort could be directed in one of several areas. The present method should be applied to a large number of more diverse problems in order to obtain a better data base for verification of the results already obtained, and to determine more accurately the advantage of this method over the revised simplex procedure. At the same time, it would be possible to determine some bounds of effectiveness of this procedure as to the size and structure of problems.

Modification of the step-by-step procedure might be attempted to take advantage of an existing basis in the original problem so as to require generation of fewer artificial variables. An attempt might also be made to modify the step-by-step addition of constraints procedure for application to the primal-dual algorithm.

8. Bibliography

1. Byers, A. L. "A Study of Some Aspects of Linear Programming and An Approximate Solution of a Classical Weight Loading Problem." Thesis: Department of Chemical and Petroleum Engineering. University of Kansas, 1963.

2. Charnes, A., W. W. Cooper, and A. Henderson. An Introduction to Linear Programming. New York: John Wiley and Sons, Inc., 1953.

3. Dantzig, G. B. Linear Programming and Extensions. Princeton: Princeton University Press, 1963.

4. Gass, S. I. Linear Programming Methods and Applications. New York: McGraw-Hill Book Co., Inc., 1958.

5. Hadley, G. Linear Programming. Reading, Mass: Addison-Wesley Publishing Co., Inc., 1962.

APPENDIX I.

Statistical Testing of Random Problem Solutions

If it can be assumed that the solution times obtained by each method form two normal distributions, we can test to determine if the mean solution time by the addition of constraints method, μ_{SSAC} is less than the mean solution time by the revised simplex method, μ_{RS}.

Letting X_i be the solution time obtained by the revised simplex procedure, and Y_i the solution time obtained by the addition of constraints procedure for problem i, a pairwise comparison of the results of the two methods can be made.

Consider the hypothesis that the mean solution time difference, $\mu = \mu_{RS} - \mu_{SSAC}$, is non-positive. That is,

$$H_0: \quad \mu \leq 0,$$

with the alternative hypothesis,

$$H_1: \quad \mu > 0 .$$

To show that the method of addition of constraints is faster, we must be able to reject H_0.

If we form a t statistic for n = 46 samples,

$$t = \frac{\bar{D}\sqrt{n}}{S}$$

where,

$$\bar{D} = \frac{1}{n} \sum_{i=1}^{n} (X_i - Y_i)$$

and,

$$S^2 = \frac{1}{n-1} \sum_{i=1}^{n} (D_i - \bar{D})^2 \quad ,$$

and then compare it against tabulated values of the cumulative t distribution, we would reject the hypothesis if

$$t \geq t_{\alpha, n-1} .$$

In this case we choose a $100(1-\alpha) = 95\%$ confidence level. The computation of the t statistic for $n = 46$, $S = 6.74$ gives,

$$t = \frac{3.94 \sqrt{46}}{6.74} \doteq 3.97 \quad .$$

The tabulated value for $t_{(0.05),45} = 1.68$. Since $t > t_{\alpha,n-1}$ we reject the hypothesis that the revised simplex method is faster.

A lower confidence limit on the mean difference in solution times can be determined from the expression

$$\text{Prob}\left\{\frac{(\bar{D} - \mu_L)\sqrt{n}}{S} \leqslant t_\alpha\right\} = 1 - \alpha .$$

Since with 95% confidence

$$\frac{(\bar{D} - \mu_L)\sqrt{n}}{S} \leqslant t_\alpha$$

we can solve for the lower limit on μ,

$$\bar{D} - \frac{ts}{\sqrt{n}} \leq \mu_L \cdot$$

For the given data,

$$3.94 - \frac{(1.68)(6.74)}{(6.785)} \leq \mu_L$$

or,

$$2.27 \leq \mu_L .$$

So we can say with 95% confidence that the mean difference in solution times is no less than 2.27 seconds. That is, the mean solution time obtained by the step-by-step addition of constraints is at least 2.27 seconds less than the solution time obtained by the revised simplex procedure.

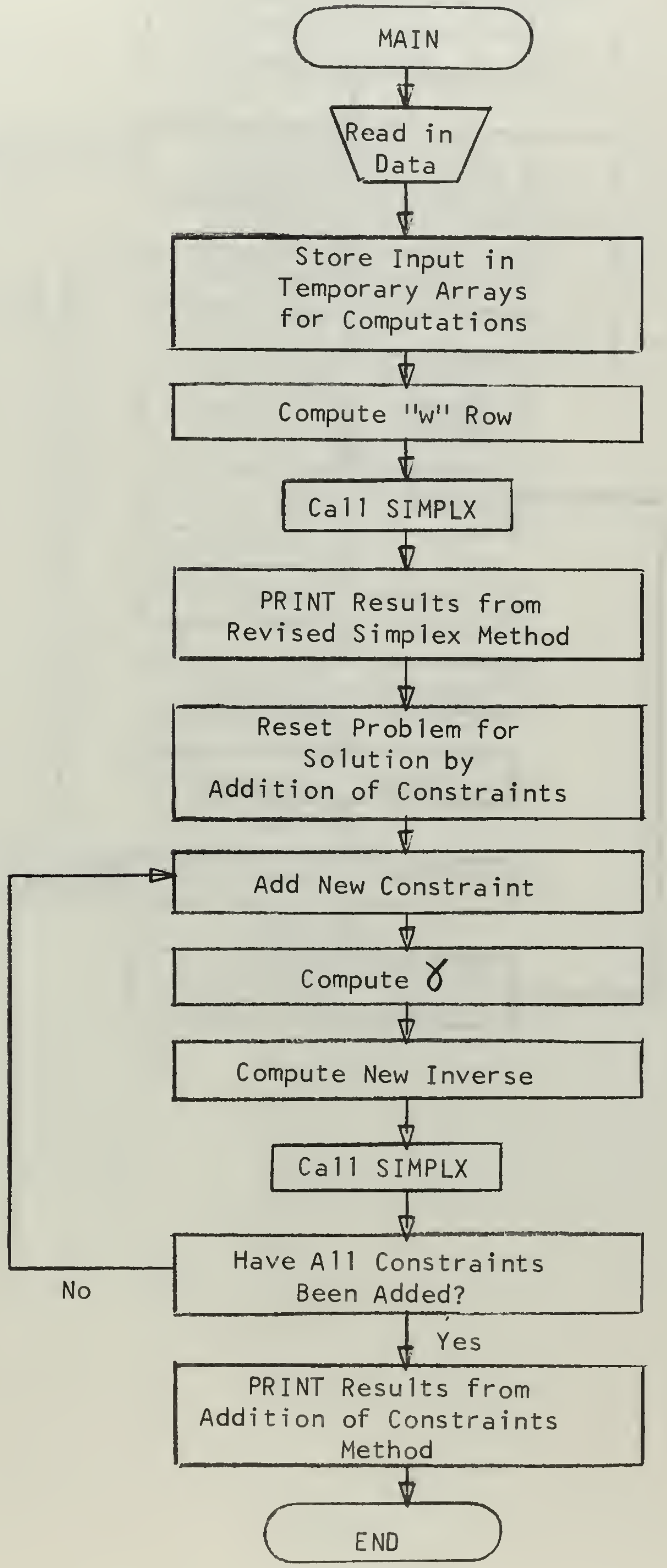
TABLE II

SOLUTION TIME RESULTS OF RANDOMLY GENERATED PROBLEMS

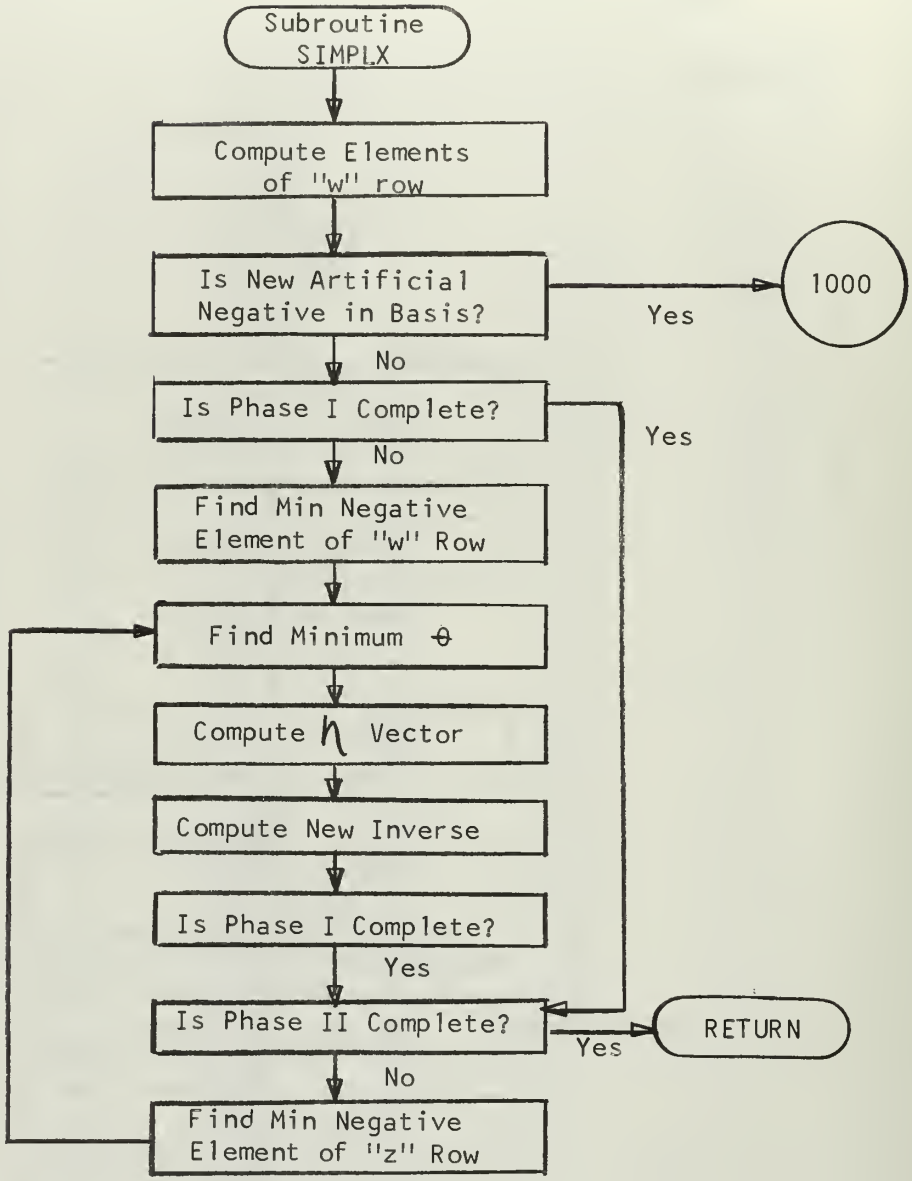
Prob. No.	RS Time (X_i) (sec.)	RS Iter.	SSAC Time (Y_i) (sec.)	SSAC Iter.	$X_i - Y_i$	$(X_i - Y_i)^2$
1	23.90	68	5.11	25	18.79	220.5
2	18.19	51	7.08	56	11.11	51.4
3	14.97	42	10.39	71	4.58	0.4
4	28.25	82	18.34	101	9.91	35.6
5	20.21	58	16.66	92	3.55	0.2
6	23.31	64	9.88	64	13.43	90.1
7	27.24	79	25.23	179	2.01	3.7
8	12.10	33	5.14	24	6.96	9.1
9	13.47	37	17.63	93	-4.16	65.6
10	13.47	37	5.56	38	7.91	15.8
11	13.13	36	18.50	113	-5.02	80.3
12	14.50	40	5.69	38	8.81	23.7
13	9.72	26	14.68	80	-4.96	79.2
14	13.48	37	7.68	59	5.80	3.5
15	13.14	36	10.10	91	3.04	0.8
16	10.76	29	18.43	93	-7.67	134.8
17	15.53	43	15.41	90	0.12	14.6
18	19.62	56	7.29	48	12.33	70.4
19	24.38	69	12.05	65	12.33	70.4
20	19.16	54	10.30	51	8.86	24.2
21	22.23	63	8.68	56	13.55	92.4
22	21.93	62	10.83	60	11.10	51.3
23	26.35	75	19.69	87	6.66	7.4
24	23.65	67	7.69	45	15.96	144.5
25	14.40	43	5.22	38	9.18	27.5
26	9.72	28	12.20	90	-2.48	41.2
27	14.73	44	7.02	52	7.71	14.2
28	10.34	30	10.93	70	-0.59	20.5
29	11.58	34	14.32	97	-2.74	44.6
30	11.62	34	9.06	58	2.56	1.9
31	12.51	35	11.81	81	0.70	10.5
32	12.17	34	5.43	32	6.74	7.8
33	13.16	37	5.44	38	7.72	14.3
34	11.83	33	5.55	40	6.28	5.5
35	12.18	34	17.25	93	-5.07	81.2
36	12.51	35	16.03	73	-3.52	55.7
37	11.18	31	14.30	101	-3.12	49.8
38	15.18	43	15.07	90	0.11	14.7
39	9.24	27	7.90	54	1.34	6.8
40	11.98	36	12.72	81	-0.74	21.9
41	11.06	33	9.99	74	1.07	8.2
42	11.41	34	7.86	62	3.55	0.2
43	13.52	41	6.88	57	6.64	7.3
44	11.36	34	8.63	51	2.73	1.5
45	11.68	35	24.94	142	-13.26	295.8
46	9.53	28	8.16	58	1.37	6.6

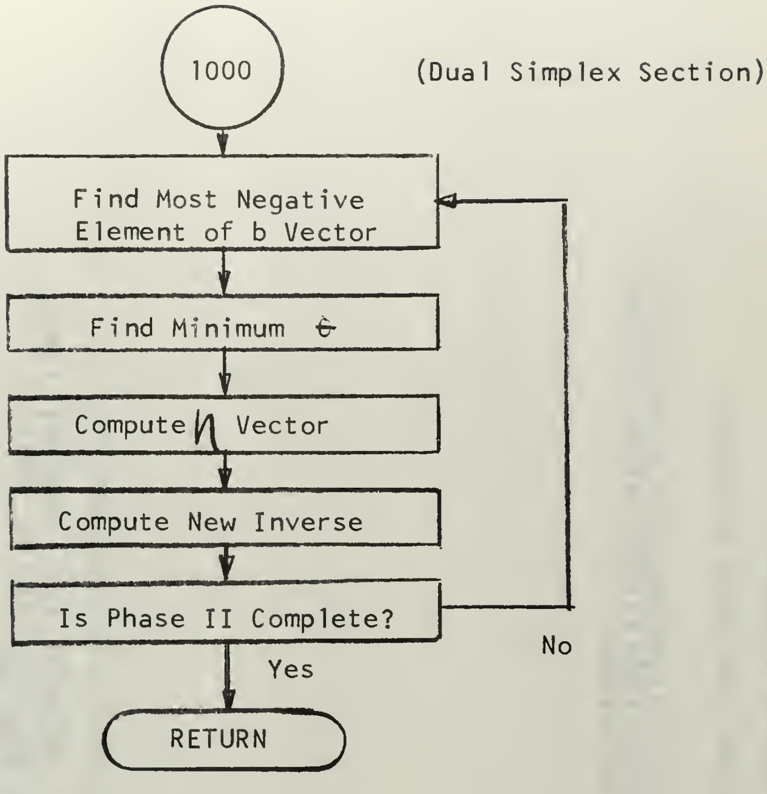
APPENDIX II.

Flow Diagrams of the Computer Program



2. Simplex and Dual Simplex Iteration Subprogram




1000
(Dual Simplex Section)
Find Most Negative Element of b Vector
Find Minimum θ
Compute η Vector
Compute New Inverse
Is Phase II Complete?
No
Yes
RETURN

APPENDIX III.

FORTRAN Listing of the Computer Program

```
C
C     MAIN PROGRAM -- SERVES AS A DRIVING ROUTINE FOR BOTH METHODS
C
      DIMENSION RUN(15)
      COMMON/BLOCKA/A(52,150),XA(52,150),E(52,52),XE(52,52),BINV(52,52),
     1B(52),XB(52),XNU(52),JH(52),XBINV(52,52),THETA(150),MTEMP,NTEMP,
     2MROW,NCOL,GAMMA(100),JTYPE,JTERT/BLOCKB/MTYPE
      Z=RNG(0)
    5 DO 10 I=1,52
      DO 10 J=1,150
      A(I,J)=0.
   10 XA(I,J)=0.
      JTYPE=0
      MTYPE=0
      CALL CLOCK0
      JTERT=0
      INDX5=0
      DO 15 I=1,52
      JH(I)=0
      B(I)=0.
      XB(I)=0.
      XNU(I)=0.
      DO 15 J=1,52
      E(I,J)=0.
      XE(I,J)=0.
   15 BINV(I,J)=0.
      READ(5,20)MROW,NCOL,(RUN(I),I=1,15)
   20 FORMAT(2I5,15A4)
C
C     INPUT NOTE...MROW= NO. OF ROWS PLUS TWO, BECAUSE COST COEFFICIENTS
C                  ARE ENTERED AS ROW ONE AND THE ARTIFICIAL VARIABLE COST
C                  COEFFICIENTS ARE ENTERED AS ROW TWO.
C
      IF(MROW)23,22,25
   22 STOP
   23 L=1-4*MROW
      Z=RNG(L)
      CALL PSEUDO
      GO TO 41
   25 READ(5,35)(A(1,J),J=1,NCOL)
      DO 26 J=1,NCOL
   26 A(1,J)=-A(1,J)
      DO 30 I=3,MROW
   30 READ(5,35)(A(I,J),J=1,NCOL)
   35 FORMAT(7F10.5)
```

```
      READ(5,35)(B(I),I=3,MROW)
      WRITE(6,40) RUN
   40 FORMAT(1H1,28X,15A4)
   41 WRITE(6,45)(A(1,J),J=1,NCOL)
   45 FORMAT(///10X,'COST COEFFICIENTS'//(10X,8F13.6))
      DO 55 I=3,MROW
      K=I-2
      WRITE(6,50)K,(A(I,J),J=1,NCOL)
   50 FORMAT(//3X,'ROW',I3,1X,8E13.6/(10X,8E13.6))
   55 WRITE(6,60)K,B(I)
   60 FORMAT(1H ,2X,'B(',I2,')  ',F13.6)
C
C     GENERATE AN MROW * MROW IDENTITY MATRIX
C
      DO 65 I=1,MROW
   65 E(I,I)=1.
C
C     STORE INPUT INFORMATION IN TEMPORARY ARRAYS FOR COMPUTATIONS
C
      DO 70 I=1,MROW
      XB(I)=B(I)
      DO 70 J=1,NCOL
   70 XA(I,J)=A(I,J)
      DO 71 I=1,MROW
      DO 71 J=1,MROW
      BINV(I,J)=E(I,J)
   71 XE(I,J)=E(I,J)
      CALL CLOCK1(0,TIMEX)
C
C     COMPUTE 'W(J) - D(J)' ROW
C
      DO 75 J=1,NCOL
      DO 75 I=3,MROW
      XA(2,J)=XA(2,J)-XA(I,J)
   75 A(2,J)=XA(2,J)
      DO 80 I=3,MROW
   80 XB(2)=XB(2)-XB(I)
      B(2)=XB(2)
      MTEMP=MROW
      NTEMP=NCOL
C
      CALL SIMPLX
C
      CALL CLOCK1(-1,TIMEX)
      WRITE(6,775)
```

```
775 FORMAT(1H1,22X,'SOLUTION BY REVISED SIMPLEX METHOD')
 81 WRITE(6,740)RUN
740 FORMAT(1H0,28X,15A4)
    WRITE(6,773)TIMEX
773 FORMAT(1H0,34X,'ELAPSED TIME',F15.3,'  MICROSECONDS')
    WRITE(6,82)JTERT
 82 FORMAT(1H0,34X,'NUMBER OF ITERATIONS REQUIRED',I10)
    WRITE(6,85)XB(1)
 85 FORMAT(///34X,'MAXIMUM COST OF OBJECTIVE FUNCTION IS ',E13.6////
   145X,'BASIS VECTORS AND COEFFICIENTS'///37X,'VECTOR',10X,
   2'COEFFICIENT (X-ZERO COMPONENT)'//)
    DO 95 I=3,MROW
    IF(JH(I))95,95,90
 90 WRITE(6,100)JH(I),XB(I)
 95 CONTINUE
100 FORMAT(37X,'P(',I2,')',19X,E13.6)
    IF(INDX5)200,200,5
C
C  RESET PROBLEM FOR SOLUTION BY ADDITION OF CONSTRAINTS
C
200 DO 500 I=1,MROW
    DO 500 J=1,MROW
    XE(I,J)=E(I,J)
    XA(I,J)=0.
500 BINV(I,J)=E(I,J)
    MTYPE=-1
    JTERT=0
    DO 501 I=1,NCOL
501 JH(I)=0
    CALL CLOCK1(0,TIMEX)
    DO 505 J=1,NCOL
505 XA(1,J)=A(1,J)
    XB(1)=B(1)
    NTEMP=NCOL
    DO 575 MTEMP=3,MROW
C
C  ADD NEW CONSTRAINT
C
520 DO 525 J=1,NCOL
    XA(MTEMP,J)=A(MTEMP,J)
525 XA(2,J)=0.
    DO 531 J=1,NCOL
    DO 530 I=3,MTEMP
530 XA(2,J)=XA(2,J)- A(I,J)
531 A(2,J)=XA(2,J)
```

```
      XB(2)=0.
      DO 535 I=3,MTEMP
  535 XB(2)=XB(2)- B(I)
      B(2)=XB(2)
      MM1=MTEMP-1
C
C  COMPUTE GAMMA
C
      DO 545 J=1,NCOL
  545 GAMMA(J)=0.
      DO 550 J=1,MM1
      IF(JH(J))550,550,546
  546 GAMMA(J)=XA(MTEMP,JH(J))
  550 CONTINUE
C
C  COMPUTE NEW B-INVERSE
C
      DO 565 J=1,MM1
      DO 555 I=1,MM1
  555 BINV(MTEMP,J)=BINV(MTEMP,J)-GAMMA(I)*BINV(I,J)
      IF(ABS(BINV(MTEMP,J))-(.0001))560,560,565
  560 BINV(MTEMP,J)=0.
  565 CONTINUE
      BINV(MTEMP,MTEMP)=1.
      DO 570 I=1,MM1
  570 BINV(I,MTEMP)=0.
C
      CALL SIMPLEX
C
      IF(MTYPE)575,575,576
  575 CONTINUE
  576 CALL CLOCK1(-1,TIMEX)
      WRITE(6,776)
  776 FORMAT(1H1,20X,'SOLUTION BY ADDITION OF CONSTRAINTS METHOD')
      INDX5=1
      GO TO 81
      END
C
C
C
C
C
C
C
C
```

```
      SUBROUTINE SIMPLX
      COMMON/BLOCKA/A(52,150),XA(52,150),F(52,52),XF(52,52),BINV(52,52),
     1B(52),XB(52),XNU(52),JH(52),XBINV(52,52),THETA(150),MTEMP,NTEMP,
     2MROW,NCOL,GAMMA(100),JTYPE,JTERT/BLOCKB/MTYPE
C
C  L=2  IF PROBLEM IS IN PHASE ONE
C  L=1  IF PROBLEM IS IN PHASE TWO
C
C
C  COMPUTE ELEMENTS OF XA(2,J)
C
      INDX3=0
      L=2
    1 ITERT=0
    5 DO    6 J=1,NTEMP
    6 XA(L,J)=0.
      DO 887 J=1,NTEMP
      DO    7 I=1,MTEMP
    7 XA(L,J)=XA(L,J)+BINV(L,I)*A(I,J)
      IF(ABS(XA(L,J))-(.0001 ))886,886,887
  886 XA(L,J)=0.
  887 CONTINUE
      JTYPE=JTYPE+1
      XB(MTEMP)=0.
      DO 888 J=1,MTEMP
  888 XB(MTEMP)=XB(MTEMP)+BINV(MTEMP,J)*B(J)
C
C  IF XB(MTEMP) IS NEGATIVE, ASSIGN ZERO PRICE TO THE NEW ARTIFICIAL
C  AND DRIVE IT OUT WITH DUAL SIMPLEX
C
      IF(JTYPE)850,850,840
  840 IF(XB(MTEMP))845,850,850
  845 KVECT=MTEMP
      BINV(2,MTEMP)=1.
      L=1
      GO TO 1040
  850 CONTINUE
C
C  DETERMINE MINIMUM NEGATIVE VALUE OF XA(L,J)
C
    8 JVECT=0
      XAMIN=0.
      DO 20 J=1,NTEMP
      IF(XA(L,J))10,20,20
   10 IF(XA(L,J)-XAMIN)15,20,20
```

```
   15 XAMIN=XA(L,J)
      JVECT=J
   20 CONTINUE
      IF(JVECT)200,200,25
C
C  COMPUTE ELEMENTS OF XA(I,JVECT) AND XB(I)
C
   25 DO 30 I=1,MTEMP
      DO 30 J=1,MTEMP
      XA(I,JVECT)=0.
   30 XB(I)=0.
      DO 835 I=1,MTEMP
      DO  35 J=1,MTEMP
      XA(I,JVECT)=XA(I,JVECT)+BINV(I,J)*A(J,JVECT)
   35 XB(I)=XB(I)+BINV(I,J)*B(J)
      IF(ABS(XA(I,JVECT))-(.0001 ))830,830,831
  830 XA(I,JVECT)=0.
  831 IF(ABS(XB(I))-(.0001 ))832,832,835
  832 XB(I)=0.
  835 CONTINUE
C
C  COMPUTE VALUES OF THETA AND DETERMINE MINIMUM THETA ROW
C
      DO  40 I=3,MTEMP
   40 THETA(I)=0.
      TMIN=99999.
      INDX1=0
      DO  60 I=3,MTEMP
      IF(XA(I,JVECT)) 60, 60, 44
   44 IF(XB(I))60,45,45
   45 THETA(I)=XB(I)/XA(I,JVECT)
      IF(THETA(I)-TMIN) 50, 60, 60
   50 NUVECT=I
      TMIN=THETA(I)
   60 CONTINUE
      IF(TMIN-99999.) 52,750, 52
   52 DO 55 K=1,MTEMP
      IF(JH(K)-JVECT)55,54,55
   54 JH(K)=0
   55 CONTINUE
      JH(NUVECT)=JVECT
C
C  COMPUTE ETA VECTOR FOR PRODUCT FORM OF THE INVERSE
C
   65 TEMPXA=XA(NUVECT,JVECT)
```

```
      DO   70 I=1,MTEMP
   70 XNU(I)=-XA(I,JVECT)/XA(NUVECT,JVECT)
      XNU(NUVECT)=1./TEMPXA
      DO   75 I=1,MTEMP
      DO   75 J=1,MTEMP
   75 XE(I,J)=E(I,J)
      DO   80 I=1,MTEMP
   80 XE(I,NUVECT)=XNU(I)
      ITERT=ITERT+1
C
C  COMPUTE NEW B-INVERSE
      IF(ITERT-300)85,85,760
   85 DO   90 I=1,MTEMP
      DO   90 J=1,MTEMP
   90 XBINV(I,J)=0.
      DO 895 I=1,MTEMP
      DO 895 J=1,MTEMP
      DO   95 K=1,MTEMP
   95 XBINV(I,J)=XBINV(I,J)+XE(I,K)*BINV(K,J)
      IF(ABS(XBINV(I,J))-(.0001 ))894,894,895
  894 XBINV(I,J)=0.
  895 CONTINUE
      DO 100 I=1,MTEMP
      DO 100 J=1,MTEMP
  100 BINV(I,J)=XBINV(I,J)
      GO TO 5
  200 IF(L-1)400,400,2200
 2200 DO 201 J=1,NTEMP
      IF(XA(2,J))201,201,700
  201 CONTINUE
  202 IF(XB(2))205,205,700
  205 L=1
      GO TO 5
  400 IF(MTEMP-MROW)500,401,500
  401 IF(INDX3)403,403,1000
  403 DO 406 I=1,MTEMP
      DO 406 J=1,MTEMP
  406 XA(I,J)=0.
      DO 990 I=1,MTEMP
      DO 990 J=1,NTEMP
      DO 410 K=1,MTEMP
  410 XA(I,J)=XA(I,J)+BINV(I,K)*A(K,J)
      IF(ABS(XA(I,J))-(.0001 ))989,989,990
  989 XA(I,J)=0.
  990 CONTINUE
```

```
      DO 415 I=1,MTEMP
  415 XB(I)=0.
      DO 985 I=1,MTEMP
      DO 420 J=1,MTEMP
  420 XB(I)=XB(I)+BINV(I,J)*B(J)
      IF(ABS(XB(I))-(.0001 ))984,984,985
  984 XB(I)=0.
  985 CONTINUE
  500 JTERT=JTERT+ITERT
      RETURN
C
C  DUAL SIMPLEX SECTION
C
 1000 BMIN=0.
      KVECT=0
      DO 1010 I=1,MTEMP
 1010 XB(I)=0.
      DO 1020 I=1,MTEMP
      DO 1015 J=1,MTEMP
 1015 XB(I)=XB(I)+BINV(I,J)*B(J)
      IF(ABS(XB(I))-(.0001 ))1016,1016,1020
 1016 XB(I)=0.
 1020 CONTINUE
C
C  DETERMINE MOST NEGATIVE XB(I)
C
      DO 1035 I=3,MTEMP
      IF(XB(I))1025,1035,1035
 1025 IF(XB(I)-BMIN)1030,1035,1035
 1030 BMIN=XB(I)
      KVECT=I
 1035 CONTINUE
      IF(KVECT)205,205,1040
C
C  COMPUTE ELEMENTS OF XA(KVECT,J)
C
 1040 DO 1045 J=1,NTEMP
 1045 XA(KVECT,J)=0.
      DO 1060 J=1,NTEMP
      DO 1050 I=1,MTEMP
 1050 XA(KVECT,J)=XA(KVECT,J)+BINV(KVECT,I)*A(I,J)
      IF(ABS(XA(KVECT,J))-(.0001 ))1055,1055,1060
 1055 XA(KVECT,J)=0.
 1060 CONTINUE
C
```

```
C   COMPUTE VALUES OF THETA AND DETERMINE MINIMUM THETA COLUMN
C
      DO 1065 I=1,NTEMP
 1065 THETA(I)=0.
      TMIN=99999.
      INDX5=0
      DO 1085 J=1,NTEMP
      IF(XA(KVECT,J))1070,1085,1085
 1070 IF(XA(1,J))1085,1074,1075
 1074 THETA(J)=0.
      GO TO 1076
 1075 THETA(J)=-XA(1,J)/XA(KVECT,J)
 1076 IF(THETA(J)-TMIN)1080,1085,1085
 1080 NUVECT=J
      TMIN=THETA(J)
 1085 CONTINUE
      IF(TMIN-99999.)1086,700,1086
 1086 DO 1088 I=1,MTEMP
      IF(JH(I)-NUVECT)1088,1087,1088
 1087 JH(I)=0
 1088 CONTINUE
      JH(KVECT)=NUVECT
C
C   COMPUTE ELEMENTS OF MINIMUM THETA COLUMN
C
      DO 1200 I=1,MTEMP
 1200 XA(I,NUVECT)=0.
      DO 1215 I=1,MTEMP
      DO 1205 J=1,MTEMP
 1205 XA(I,NUVECT)=XA(I,NUVECT)+BINV(I,J)*A(J,NUVECT)
      IF(ABS(XA(I,NUVECT))-(.0001))1210,1210,1215
 1210 XA(I,NUVECT)=0.
 1215 CONTINUE
C
C   COMPUTE ETA VECTOR FOR PRODUCT FORM OF THE INVERSE
C
 1089 TEMPXA=XA(KVECT,NUVECT)
      DO 1090 I=1,MTEMP
 1090 XNU(I)=-XA(I,NUVECT)/XA(KVECT,NUVECT)
      XNU(KVECT)=1./TEMPXA
      DO 1095 I=1,MTEMP
      DO 1095 J=1,MTEMP
 1095 XE(I,J)=E(I,J)
      DO 1100 I=1,MTEMP
 1100 XE(I,KVECT)=XNU(I)
```

```
      ITERT=ITERT+1
      IF(ITERT-300)1105,1105,760
C
C   COMPUTE NEW B-INVERSE
C
 1105 DO 1110 I=1,MTEMP
      DO 1110 J=1,MTEMP
 1110 XBINV(I,J)=0.
      DO 1120 I=1,MTEMP
      DO 1120 J=1,MTEMP
      DO 1115 K=1,MTEMP
 1115 XBINV(I,J)=XBINV(I,J)+XE(I,K)*BINV(K,J)
      IF(ABS(XBINV(I,J))-(.0001 ))1116,1116,1120
 1116 XBINV(I,J)=0.
 1120 CONTINUE
      DO 1125 I=1,MTEMP
      DO 1125 J=1,MTEMP
 1125 BINV(I,J)=XBINV(I,J)
C
C   COMPUTE NEW ELEMENTS OF XA(L,J)
C
      DO 1130 J=1,NTEMP
 1130 XA(L,J)=0.
      DO 1140 J=1,MTEMP
      DO 1135 I=1,MTEMP
 1135 XA(L,J)=XA(L,J)+BINV(L,I)*A(I,J)
      IF(ABS(XA(L,J))-(.0001 ))1136,1136,1140
 1136 XA(L,J)=0.
 1140 CONTINUE
 1150 INDXJ4=0
      DO 1160 J=1,NTEMP
      IF(XA(1,J))700,1160,1160
 1160 CONTINUE
      GO TO 1000
  700 IF(MTEMP-MROW)500,705,705
  705 CALL ERR8R(4)
      MTYPE=1
      GO TO 400
  750 IF(MTEMP-MROW)500,755,755
  755 CALL ERR8R(5)
      GO TO 400
  760 CALL ERR8R(6)
      GO TO 400
      END
C
```

```
      SUBROUTINE PSEUDO
      COMMON/BLOCKA/A(52,150),XA(52,150),F(52,52),XE(52,52),BINV(52,52),
     1B(52),XB(52),XNU(52),JH(52),XBINV(52,52),THETA(150),MTEMP,NTEMP,
     2MROW,NCOL,GAMMA(100),JTYPE,JTERT/BLOCKB/MTYPE
C   GENERATE A RANDOM PROBLEM WITH 50 CONSTRAINTS, 100 VARIABLES AND
C   50 SLACKS.
      DO 5 I=1,52
      DO 5 J=1,150
    5 A(I,J)=0.
      DO 20 J=1,100
      A(1,J)=RNG(1)
      IF(A(1,J)-.001)10,10,20
   10 A(1,J)=0.
   20 CONTINUE
      DO 30 I=3,52
      B(I)=RNG(1)*5.
      IF(B(I)-.01 )25,25,30
   25 B(I)=C.
   30 CONTINUE
      DO 40 I=3,52
      DO 40 J=1,100
      A(I,J)=RNG(1)
      IF(A(I,J)-.001)35,35,40
   35 A(I,J)=0.
   40 CONTINUE
      DO 60 I=1,50
   60 A(I+2,I+100)=-1.
      MROW=52
      NCOL=150
      RETURN
      END
      FUNCTION RNG(N)
      NR=N
      IF(NR)10,10,20
   10 IX=30517
      NR=NR+1
   20 DO 50 I=1,NR
      IY=IX*65539
      IF(IY)5,5,6
    5 IY=IY+2147483647+1
    6 RNG=IY
      RNG=RNG*.4656613E-9
   50 IX=IY
      RETURN
      END
```

```
C
C
      SUBROUTINE ERROR(KK)
      COMMON/BLOCKB/MTYPE
      IF(KK-4)2,70,62
   62 IF(KK-7)63,72,2
   70 WRITE(6,71)
   71 FORMAT(1HO,20HNO FEASIBLE SOLUTION )
      GO TO 2
   63 IF(KK-5)2,64,67
   64 IF(MTYPE)2,65,65
   65 WRITE(6,66)
   66 FORMAT(1HO,27HNO PIVOT, INFINITE SOLUTION )
      GO TO 2
   67 WRITE(6,68)
   68 FORMAT(1HO,25HITERATION LIMIT EXCEEDED )
      GO TO 2
   72 WRITE(6,73)
   73 FORMAT(1HO,22HILLEGAL INPUT QUANTITY )
    2 RETURN
      END
C
C
C
C
C
C
C
C
      SUBROUTINE CLOCKO
      DIMENSION T(2,9)
      CALL GETIME (N,M)
      TIMEH=N
      TIMEH=TIMEH/100.0
      TIMEM=M
      TIMEM=TIMEM*26
      DO 5 J=1,9
       T(1,J)=TIMEH
    5 T(2,J)=TIMEM
      RETURN
      ENTRY CLOCK1 (KEY,TIME)
      I=1
      GO TO 99
      ENTRY CLOCK2 (KEY,TIME)
      I=2
```

```
      GO TO 99
      ENTRY CLOCK3 (KEY,TIME)
      I=3
      GO TO 99
      ENTRY CLOCK4 (KEY,TIME)
      I=4
      GO TO 99
      ENTRY CLOCK5 (KEY,TIME)
      I=5
      GO TO 99
      ENTRY CLOCK6 (KEY,TIME)
      I=6
      GO TO 99
      ENTRY CLOCK7 (KEY,TIME)
      I=7
      GO TO 99
      ENTRY CLOCK8 (KEY,TIME)
      I=8
      GO TO 99
      ENTRY CLOCK9 (KEY,TIME)
      I=9
   99 CALL GETIME (N,M)
      TIMEH=N
      TIMEH=TIMEH/100.0
      TIMEM=M
      TIMEM=TIMEM*26
      IF (KEY)6,7,8
    6 TIME=TIMEM-T(2,I)
      RETURN
    7 T(1,I)=TIMEH
      T(2,I)=TIMEM
      TIME=0.0
      RETURN
    8 TIME=TIMEH-T(1,I)
      RETURN
      END
C
C
C
C
C
C
C
C
C
```

```
C
C
GETIME    START
          SAVE   (14,12)
          USING  GETIME,12
          LR     12,15
          ST     13,TEMP
          LA     13,SAVE
          L      2,0(1,0)
          L      3,4(1,0)
          TIME   BIN
          ST     0,0(2,0)
          TIME   TU
          ST     0,0(3,0)
          L      13,TEMP
          RETURN   (14,12),T,RC=0
TEMP      DS     F
SAVE      DS     18F
          END
C
C
C
C
C
C
C
C
C
C
C
C
C
C
C
C
C
C
C
C
C
C
C
C
C
C
```

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	20
2.	Library Naval Postgraduate School Monterey, California 93940	2
3.	Chief of Naval Operations (OP-96) Department of the Navy Washington, D. C. 20350	1
4.	Professor Rex H. Shudde (Thesis Advisor) Operations Analysis Curriculum Naval Postgraduate School Monterey, California	1
5.	Professor Alan McMasters Operations Analysis Curriculum Naval Postgraduate School Monterey, California	1
6.	Lt. Michael R. Fenn 2820 Dellwood Drive Kokomo, Indiana 46901	1

DOCUMENT CONTROL DATA - R&D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)* NAVAL POSTGRADUATE SCHOOL Monterey, California 93940	2a. REPORT SECURITY CLASSIFICATION Unclassified 2b. GROUP

3. REPORT TITLE
An Algorithm for the Solution of Linear Programming Problems Using Step-by-Step Addition of Constraints

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*
Masters Thesis

5. AUTHOR(S) *(Last name, first name, initial)*
FENN, Michael R.

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
December 1967	52	6

8a. CONTRACT OR GRANT NO. b. PROJECT NO. c. d.	9a. ORIGINATOR'S REPORT NUMBER(S) 9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*

10. AVAILABILITY/LIMITATION NOTICES
[redacted]

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
	Chief of Naval Operations (OP-96) Department of the Navy Washington, D. C. 20350

13. ABSTRACT

As linear programming techniques find applications in more diverse fields, the problem of solution time becomes increasingly important. A variation of the revised simplex algorithm, in which the constraints are added in a step-by-step fashion, is investigated as a potentially faster solution technique. A computational procedure, coded for the IBM 360 computer, is developed to compare this algorithm with the standard two-phase revised simplex algorithm. A limited number of problems, including several randomly generated problems, is solved by each of the two methods. The resulting comparison of solution times indicates that a significant improvement is obtained by the use of the procedure of step-by-step addition of constraints.

DD FORM 1 JAN 64 1473

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Linear Programming						
Computer Algorithm for Linear Programming						

DD FORM 1 NOV 65 1473 (BACK)