7 SOFTWARE TO IMPLEMENT THE ALGORITHMS

7.1 INTRODUCTION

In the previous chapters, a few algorithms have been developed to solve linear programming problems and some numerical examples were used for illustration. But, real life linear programming problems will not be so trivial and problems of large size cannot be solved by manual methods. Hence, computer codes have been developed for the multiplex algorithm. Computational efficiency of new algorithms can be established only when they are compared with existing algorithms. In order to compare the new algorithms with the revised simplex algorithm, computer codes were developed for revised simplex and multiplex algorithms. In the following paragraphs, the various subroutines developed for this purpose and their functions are described. The concept of modular programming has been adopted in order to facilitate the implementation of this algorithm on systems with a small main memory.
7.2 SUBROUTINES AND THEIR FUNCTIONS

7.2.1 Subroutines of the Multiplex Algorithm

The program of the multiplex algorithm consists of the following subroutines: SIZES, SIZES, MVS, MVSS, and FINAL. The functions of each of the subroutines are described below.

i. SIZES: Using this subroutine, all the promising variables for a particular pass are identified. The working of this subroutine is described below.

   a. \((z_j - c_j)\) coefficients for all the decision variables are computed and the promising variables are arranged in the ascending/descending order of their magnitude depending upon the objective.

   b. If no more promising variable is there in a pass, then the feasibility condition is checked in the main program.

ii. ENTVAR: This routine selects a set of \(k\) promising vectors \((1 \leq k \leq m)\) amongst the \(L\) number of promising vectors found in the routine SIZES. It preserves the property of linear independence among the selected vectors.
a. The column vectors of all the L promising variables are updated and it is stored as AA matrix.

b. The 'Θ' matrix is constructed for all the promising variables using the relationship,

\[ \text{THETA}(I,J) = \frac{BB(J)}{AA(J,I)} \]

provided \( BB(J) \geq 0 \) and \( AA(J,I) > 0 \)

\( I = 1, 2, \ldots L \)

\( J = 1, 2, \ldots m \)

c. According to the criterion used i.e. either the maximum rate of change criterion or the maximum change criterion of the objective function, a set of \( k \) variables among the \( L \) promising variables is selected by identifying the minimum value in each row.

iii. MATINV: Using this subroutine, the \( M^{-1} \) matrix is updated as and when a variable is introduced. The structure of the \( M^{-1} \) matrix is

\[
\begin{bmatrix}
1 & & & & \frac{C_B}{B^{-1}} \\
0 & & & & B^{-1}
\end{bmatrix}
\]

\((m+1) \times (m+1)\)
Before a variable is introduced into the basis, it checks whether that variable is still promising and also the pivot element is positive. Only then the \( M^{-1} \) matrix is updated.

Depending upon the method adopted, the condition for entering a variable into the basis varies.

The number of basis changes made and the variables that are in the basis are kept track of by this routine.

The solution vector \( B^{-1}P_0 \) is also obtained.

\[ \textbf{iv. DUALS : The DUALS subroutine is used to remove infeasibility, if any, in the basis.} \]

a. It identifies the vectors with negative entries in the rows in which infeasibility occurs.

b. It forms the dual '\( \Theta \)' matrix, the entries of which are computed as

\[ \theta_{ij} = \frac{z_j - C_j}{(B^{-1}P_j)_i} \]  \hspace{1cm} (7.2.2)  

and \( (B^{-1}P_j)_i < 0 \)

\( j = 1, 2, \ldots, n \)

\( i = \text{the row in which infeasibility occurs.} \)
c. The dual '0' matrix selects a set of variables to enter the basis such that the introduction of these variables removes infeasible variables from the basis.

v. FINAL : This routine outputs the results in a convenient form.

The flow chart for the multiplex algorithm is given in Figure 7.1.

7.2.2 Subroutines of the Revised Simplex Algorithm

The conventional two-phase technique is employed to develop the code for this algorithm. This consists of ZJSIMP, BNTSIM, MATSIM and FINALS subroutines. The functions of each of the subroutines are described below.

i. ZJSIMP : This routine identifies the most promising vector from the set of nonbasic variables by computing the \((z_j - c_j)\) coefficients for all the nonbasic variables. During Phase I, the objective is to minimise

\[ \sum_{i=1}^{k_1 + k_2} R_i \]
START

Input the matrices \( A, P, \) and \( C \)

Form a unit matrix, \( M \)

Compute \( z_{-c} \) for all the variables

Is solution optimal?

Yes

Solve for \( X_B \) and \( Z \)

No

Form the \( \alpha \) matrix using the column corresponding to the promising variables

Construct the '0' matrix using the \( \alpha \) matrix for the promising variables

Select the group of entering variables using matrix

Update the \( M \) matrix by entering the selected promising variable

Is the solution feasible?

Yes

Remove infeasibility using either dual simplex or dual multiplex procedure

No

Output results

Stop

Figure 7.1 Flow Chart of the Multiplex Algorithm
where \(k_1\) and \(k_2\) are the number of lower bound and equality constraints respectively. Under optimal conditions it outputs the results and terminates.

ii. **ENTSIM**: The leaving vector is identified by this routine. The entering variable column is updated and the solution vector is found.

iii. **MATSIM**: It updates and inverts the \(M\) matrix.

iv. **FINALS**: This routine outputs the result in a convenient form.

7.2.3 **Subroutines for the Multiplex Bounded Algorithm**

The same number of subroutines as in the case of multiplex algorithm is used here also with slight modifications in the subroutines. The modifications are:

i. Two additional columns are added to the '\(\theta\)' matrix to accommodate the lower and upper bound of the decision variables and the selection of variables is made as explained in Chapter 4.

ii. The \(M^{-1}\) matrix, solution vector and the constraint matrix are suitably modified depending upon whether a particular variable is entering at its bounds or at an intermediate value.

iii. To remove infeasibility, if any, a version of the dual multiplex algorithm for bounded variable problem is used.
7.2.4 Subroutines for the Simplex Bounded Algorithm

The subroutines for the simplex algorithm are suitably modified to accommodate the bounded variables. The modifications include:

i. Selection of leaving variable by alternate criterion.

ii. Updating of $M^{-1}$ matrix, solution vector and the constraint matrix, depending upon whether a particular variable enters at its bounds or at an intermediate value.

7.3 EXPERIENCES IN PROGRAMMING

Programs were written in Fortran IV for all the algorithms to suit IBM System 370/155-II. Same problem run at different points in time took different processing time. This may be due to such phenomena as 'cycle stealing', 'swapping', 'page fault', 'thrashing', etc., which occur in multiprogramming mode of operation. The efficiency of a program can be increased by removing redundant statements and also reducing the number of repetitive operations. It can be done not only in the arithmetic statements but also in the performance of DO loops. Not only the program but also the compiler
performance is responsible for the increase or decrease in the execution time of a program.

7.4 CONCLUSION

Subroutines developed for solving problems using multiplex algorithm have been discussed. A few large size problems were tried and the results are tabulated in the Chapter 5. Though the multiplex algorithm takes less number of multiply/divide operations compared to simplex algorithm, the savings in processing time do not commensurate because of unnecessary checks made in the algorithm before a multiply/divide operation is performed. Hence, if the nonzero elements alone are stored in the memory and the multiply/divide operations can be done directly which in turn will reduce the computation time and increase the saving obtained using the multiplex algorithm.