6 SOME ADDITIONAL COMPUTATIONAL ASPECTS

6.1 INTRODUCTION

In this chapter a few other aspects of linear programming problems are considered to make further improvement in computational efficiency over and above that obtained from the multiplex algorithm. One method is to minimise the popping in and out of variables of the basis and the other is to check for redundancy, if any, among the constraints so that they could be removed before the commencement of computation. The set of popping variables wastes computational effort as does a set of redundant constraints. The proposed methods attempt to minimise computational effort due to these two variants of the simplex procedure. It has been observed from some experiments conducted on linear programming problems that if the steepest ascent/descent direction in a selected basis is chosen for the search, the number of popping variables is minimised. The multiplex algorithm developed in Chapter 3 combined
with a modified optimality criterion has resulted in the minimisation of popping variables. Alternate selection procedures for the selection of variable using 'e' matrix has also resulted in the minimisation of popping variables. The 'e' matrix could also be used to detect redundant constraints, if any, which could be ignored for purposes of computation. The approach is illustrated with a number of examples.

6.2 THE PROBLEM

Medium to large scale linear programming problems on which the experiments were conducted were drawn from a library of industrial problems. None was especially constructed for test purposes. Problems having either minimisation or maximisation objectives with constraints of all types were chosen. In all these problems, a number of decision variables enters, leaves and reenters the basis, wasting computational effort. The optimality criterion used by the simplex procedure does not minimise the tendency of variables popping in and out of the basis.

Consider a m constraint and n variable linear programming problem. Theoretically speaking, if there are no popping variables, any linear programming problem
should converge to optimal solution in exactly $m$ iterations. This is only a theoretical venture but does not happen in practice particularly when simplex algorithm is used. This means, variables pop in and out of the basis a number of times increasing the number of iterations to more than the number of constraints. As far as the author is aware, no procedure has been suggested by any one so far to minimise the occurrence of this phenomenon. The argument in favour of the simplex procedure has always been that it works satisfactorily for any kind of linear programming problem. One should not be complacent with this aspect of the simplex procedure alone. The optimality criterion adopted by the simplex procedure does very little towards minimising the popping of variables of the basis. An optimality criterion is suggested to minimise the number of iterations while finding out optimal solution. The criterion is to search for the optimal solution in the direction of the greatest change (i.e. steepest ascent/descent). The intercept matrix developed in Chapter 3 is used to choose the direction of the largest change for the objective function.
6.3 MINIMIZATION OF POPPING VARIABLES

6.3.1 Criterion Employed

It is by no means true that the optimality criterion adopted for selecting an incoming vector yields a procedure which reaches an optimal solution in the smallest number of iterations. The ideal criterion would be one which assures that once a vector had been inserted into the basis, it would never have to be removed again; i.e., a minimum of basis changes would lead to the optimal basis; in fact no more than \( m \) (number of resource constraints) basis changes would ever be needed. Unfortunately, no criteria have been developed which will guarantee that once a vector has been inserted into the basis, it will never be removed. In the simplex method a vector may enter the basis and leave again. In fact this can happen several times [28].

The conventional optimality criterion uses the \( (z_j - c_j) \) value to identify the most promising variable. This is a very well founded, proven and largely used criterion, which works satisfactorily for all the linear programming problems. But it has an element of sluggishness in the search process to reach optimal solution. Several investigators have tried different
optimality criteria to improve computational efficiency [2, 3, 4, 5, 6, 9, 12, 19, 22, 28, 40, 46, 47]. As stated already one serious drawback of the optimality criterion employed by the simplex algorithm is that it allows variables to pop in and out of the basis. This leads to considerable wastage of computational effort due to a nonbasic variable becoming basic and again nonbasic and so on. There is no known algorithm by which this phenomenon could be completely arrested. However, the author has found while solving linear programming problems that the following optimality criterion acts as a deterrent to variables popping in and out of the basis. The optimality criterion which gives the greatest change in the objective function, was described long ago, but has been little used [40]. This procedure is referred to as the method of steepest ascent, since it gives the greatest possible change in the objective function in each iteration. In order to effect this, that column is chosen which after pivoting will give the greatest increase in the objective function at each iteration; it is that variable which gives maximum/minimum value depending on the objective in the following formula.

\[
\text{Max or Min } (z_j - c_j) \min_i \left[ \frac{(B^{-1}p)_i}{(B^{-1}P_j)_i}; (B^{-1}p)_i > 0 \right]
\]

(6.3.1)
Using (6.3.1) a set of $k$ ($k \leq m$) number of graded variables in the order of their promise, could be chosen to form the starting basis. The maximum rate of change and the maximum change in the objective function criteria were used in the multiplex algorithm. It was found by solving a number of linear programming problems that the greatest change criterion minimises the number of change of bases. This means saving in computational effort. Examples were worked out and the results are furnished.

One other criterion is that whichever column has the greatest projection on the objective function co-ordinate axis that column is the most promising. Thus picking that vector whose projection on the $Z$ axis is the longest, is the same as picking that variable whose $(z_j - c_j)$ is the largest. Since the length of a vector affects the length of its projection and since a vector's length is affected by initial matrix scaling, the projection of a normalised vector on the $Z$ axis will give a better measure of the prominence of the variable[19]. This may be used as a criterion to select a set of variables.
6.3.2 Alternate Selection Criterion

In the multiplex algorithm described in Chapter 3, the type of constraint was not considered while selecting a set of variables to enter the basis at start of each pass. In the event of a slack variable(s) leaving the basis which is often the case during the first pass, advantage can be taken of the type of constraint to enter only such variable(s) which will minimize tendency of variable(s) to pop in and out of basis. The following step by step procedure can be used to accomplish the above.

Step 1: The '0' matrix is scanned row by row and the positions of the minimum intercept irrespective of the type of constraints and the minimum intercept due to upper bound constraints are identified.

Step 2: If the minimum (min 1) in a row due to the upper bound constraints happens to be greater than the minimum (min 2) for the entire row, then min 2 is due to either equality or lower bound constraints. In the above, the following two cases may arise.
Step 5: Steps 3 and 4 are repeated until all the rows are covered. The collection of vectors corresponding to the kth rows will be the entering vectors and the ith columns will be the leaving vectors.

This type of selection policy reduces the popping in and out of variables when there are lower bound and upper bound constraints, particularly in case of problems with minimization objective.

6.3.3 Selection based on non-competing variables

The step by step selection procedure for the above method is described below.

Step 1: The $\theta$ matrix is scanned row after row and the position of the smallest intercept in each row is located.

Step 2: The first entering vector will be the most promising vector in the set, say, jth vector and let the corresponding row in the $\theta$ matrix be k. This is the entering vector in the place of a vector corresponding to the column in which the minimum intercept occurs in the $\theta$ matrix, say ith column.
Step 3: Examine whether any other row has positive entry in the ith column. All such rows as well as the kth row and the ith column are deleted.

Step 4: Steps 2 and 3 are repeated till all the rows are covered.

Step 5: The collection of vectors corresponding to the kth rows will be the entering vectors and the kth columns will be the leaving vectors.

While applying this criterion all the inequality constraints are changed to upper bound type in addition to the equality constraints, if any. First, the equality constraints alone are taken and the variables are selected. When no more variables can be introduced, other constraints are included and the entering variables are chosen. As before, when no more promising variable is present or variable cannot be introduced with a positive value, feasibility can be checked. Infeasibility, if any, can be removed using dual multiplex algorithm. This method reduces the number of variables popping in and out of the basis.
6.3.4 Method based on Resource Level

Before applying multiplex algorithm for any linear programming problem, the largest among the resources is identified. Check whether all other resources are less than half of the largest resource. If it is so, then the multiplex algorithm is applied. Otherwise, the constraint which has a resource greater than half of the biggest resource is replaced by a constraint obtained by subtracting that constraint from the constraint with the largest resource. Then the multiplex algorithm is applied. This also reduces the popping in and out tendency of the variables.

6.4 REDUNDANT CONSTRAINTS

A resource constraint is said to be redundant if it does not form part of the boundary of the constraint set. If such constraints are not deleted they waste lot of computational effort and thus contribute to inefficiency. As far as the author's knowledge goes, no proven step by step procedure is available to detect redundancy at the first instance. It is more an educated guess than a methodology founded on any formulation. It was observed while finding solution for some of the linear programming problems that a set of surplus variables always stays as
basic variables irrespective of changes in the other variables from being nonbasic to basic and vice versa[40]. Surplus variables behaving in this manner clearly indicate that no product could consume these resources completely and irrespective of the product mix, they are always surplus. Such redundant constraints at no stage become active constraint and they do not affect the optimal solution. It is a well established fact that additional constraints take more computations to obtain optimal solution. The 'θ' matrix developed in Chapter 3 may be utilised to detect simple redundancies.

6.5 STEP BY STEP ALGORITHM

Step 1 : Select the entering and leaving variables using the 'θ' matrix.

Step 2 : If there are any unscored columns go to step 3, otherwise find the optimal solution.

Step 3 : Check whether the intercepts in the unscored column in each row is greater than the intercepts in the deleted rows and also check whether the sum of the pivot intercepts of the selected variables is greater than the intercept in the undeleted column. If both
these conditions are satisfied, then the constraint corresponding to the undeleted column is treated as redundant and is omitted and the optimal solution is obtained.

**Step 4:** Check, when the optimal solution is sought whether the detected redundancy is correct by substituting the current optimal solution in the constraints. If the optimal solution violates the redundant constraint, then update the solution using the provision to carry out sensitivity analysis.

6.6 **EXAMPLES**

6.6.1 **Minimising reentry of variables**

The frequency with which variables pop in and out of the basis is furnished in Table 6.1 for a four constraint six variable problem[4]. The linear programming model of the problem is given below.

Max \( Z = 0.4x_1 + 0.28x_2 + 0.32x_3 + 0.72x_4 + 0.64x_5 + 0.6x_6 \)

subject to

\[
\begin{align*}
0.01 & \quad 0.01 & \quad 0.01 & \quad 0.03 & \quad 0.03 & \quad 0.03 \\
0.02 & \quad 0 & \quad 0 & \quad 0.05 & \quad 0 & \quad 0 \\
0 & \quad 0.02 & \quad 0 & \quad 0 & \quad 0.05 & \quad 0 \\
0 & \quad 0 & \quad 0.03 & \quad 0 & \quad 0 & \quad 0.08
\end{align*}
\]

\[
\begin{align*}
x_1 & \quad x_2 & \quad x_3 & \quad x_4 & \quad x_5 & \quad x_6 \\
\end{align*}
\]

\[
\begin{align*}
850 \\
750 \\
100 \\
900
\end{align*}
\]

\( x_1, x_2, x_3, x_4, x_5, x_6 \geq 0 \)
It may be observed that this phenomenon depends on the criterion used as a decision rule to decide the entering vectors. On the average, it was observed that the greatest change in the objective criterion is more effective in minimising the phenomenon of variables popping in and out of the basis than the maximum rate of change (i.e., \( z_j - c_j \)) criterion as may be seen from the following frequency tables. This is where the advantage of the 'O' matrix and multiplex algorithm lies. Wherever the frequency of a variable is one, those variables when once enter the basis, never leave.

The frequency of a variable may be more than one depending on the intensity of its entry and reentry into the basis. This phenomenon not only accounts for why in most of the large scale real life problems, the optimal solution is obtained with a number of iterations far more than the number of constraints but also a measure of waste of computational effort. The update of any column more than once in the revised simplex procedure requires wasteful and unnecessary computational effort. The frequency with which variables enter and leave the basis for different problems are furnished in Tables 6.1, 6.2, 6.3 and 6.4 and the frequency diagrams are given in Figures 6.1, 6.2, 6.3 and 6.4.
Table 6.1 4 x 6 Problem

Frequency of the

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Decision variables</th>
<th>Slack variables</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$x_1$  $x_2$  $x_3$  $x_4$</td>
<td>$s_1$  $s_2$  $s_3$  $s_4$</td>
</tr>
<tr>
<td>Simplex</td>
<td>1  1  1  2  2  2</td>
<td>1  2  2  2</td>
</tr>
<tr>
<td>Multiplex - Criterion I</td>
<td>1  1  1  2  2  2</td>
<td>1  2  2  2</td>
</tr>
<tr>
<td>Multiplex - Criterion II</td>
<td>1  1  1  0  0  0</td>
<td>1  2  2  2</td>
</tr>
</tbody>
</table>
FIG. 6.1. 4 X 6 MAX PROBLEM

a. SIMPLEX

b. MULTIPLEX CRITERION I

MULTIPLEX CRITERION II
### Table 6.2 8 x 16 Problem

Frequency of the

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Decision variables</th>
<th>Slack variables</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$x_1$ $x_2$ $x_3$ $x_4$ $x_5$ $x_6$ $x_7$ $x_8$ $x_9$ $x_{10}$ $x_{11}$ $x_{12}$ $x_{13}$ $x_{14}$ $x_{15}$ $x_{16}$</td>
<td>$s_1$ $s_2$ $s_3$ $s_4$ $s_5$ $s_6$ $s_7$ $s_8$</td>
</tr>
<tr>
<td>Simplex</td>
<td>1 1 1 2 1 1 0 1 0 0 0 2 1 0 0 0</td>
<td>2 2 2 2 2 3 4 2</td>
</tr>
<tr>
<td>Multiplex -</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Criterion I</td>
<td>1 1 1 0 1 1 0 1 0 0 0 0 1 0 0 0</td>
<td>2 2 2 2 2 3 2 2</td>
</tr>
<tr>
<td>Multiplex -</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Criterion II</td>
<td>1 1 1 0 1 1 0 1 0 0 0 0 1 0 0 0</td>
<td>2 2 2 2 2 3 2 2</td>
</tr>
</tbody>
</table>
FIG. 6.2. 8X16 MAX PROBLEM
### Table 6.3 17 x 31 (min) problem

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Decision variable</th>
<th>Slack variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simplex</td>
<td>1 1 1 1 1 1 1 1 3</td>
<td>1 1 1 1 1 1 1 1 0 0 2 2 2 2 0 0 0 0 2 0 0 2 3 1 2 2 2 3</td>
</tr>
<tr>
<td>Multiplex - Criterion I</td>
<td>1 1 1 1 1 1 1 1 1</td>
<td>1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 2 2 1</td>
</tr>
<tr>
<td>Multiplex - Criterion II</td>
<td>1 1 1 1 1 1 1 1 1</td>
<td>1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 2 2 1</td>
</tr>
</tbody>
</table>
FIG. 6.3. 17X31 MIN PROBLEM
<table>
<thead>
<tr>
<th>Criterion</th>
<th>Decision variable</th>
<th>Slack variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simplex</td>
<td>2 2 2 1 1 1 3 0 0</td>
<td>s_1 s_2 s_3</td>
</tr>
<tr>
<td>Multiplex - Criterion I</td>
<td>2 2 2 1 1 1 2 2 1</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>Multiplex - Criterion II</td>
<td>2 2 2 1 1 1 2 2 1 1</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
</tbody>
</table>

Table 6.4 17 x 31 (max) problem

Frequency of the
FIG. 6.4. 17X31 MAX PROBLEM
6.6.2 Inference from the frequency diagram

Computational effort may be wasted in the simplex algorithm due to variables entering, leaving and reentering the basis. The multiplex algorithm proposed in this thesis minimises the waste of computational effort due to variables popping in and out of the basis. This is evident from the Figures 6.2 and 6.3 that the optimal path chosen by the multiplex algorithm leads to optimal solution in exactly the same number of iterations as the number of constraints.

It may be observed that the multiplex algorithm completely eliminates variables popping in and out of the basis (Figure 6.3b) whereas the simplex algorithm is not able to prevent this phenomenon (Figure 6.3b) for the 17 constraints, 31 variables problem. Figures (6.4a) and (6.4b) reveal the same with a little graceful degradation of the multiplex algorithm.

Figures (6.2a) and (6.2b) drawn for another example also show the superiority of the multiplex algorithm over the simplex. It also establishes that if there are m constraints in a linear programming problem, the optimal solution could be obtained exactly in m iterations which hitherto has been a far fetched idea. It may be observed
from the various frequency diagrams that the multiplex algorithm does not entertain, in general, variables popping in and out of bases.

6.6.3 Example to illustrate alternate selection criterion

Minimize \( Z = 2x_1 + 3x_2 \) \[29\]
subject to
\[
\begin{align*}
2x_1 + 3x_2 &\leq 30 \\
x_1 + 2x_2 &\geq 10 \\
x_1 - x_2 &\geq 0 \\
x_1 &\geq 5 \\
x_2 &\geq 0
\end{align*}
\]

Pass 1: The '\( \delta \)' matrix formed for the above problem is

\[
[\delta] = \begin{bmatrix}
2 & x_1 & 15 & 10 & [0] & 5 \\
3 & x_2 & 10 & [5] & - & -
\end{bmatrix}
\]

As per the original selection procedure
i. \( x_1 \) enters and \( s_3 \) leaves
ii. \( x_2 \) enters and \( s_2 \) leaves

But the solution to the above problem is
Hence, it requires one more pass to reach the optimal solution. But, if the alternate selection criterion is used, the entering and leaving vectors are given as below.

<table>
<thead>
<tr>
<th>2</th>
<th>15</th>
<th>10</th>
<th>0</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>10</td>
<td>5</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

i. \( x_1 \) enters and \( s_2 \) leaves.

The optimum solution is obtained at the end of first pass.

6.6.4 Example to illustrate the method of selection based on non-competing variables for the same resource

Maximize \( Z = 10x_1 + 5x_2 + 4x_3 \)

subject to

\[
2x_1 + 2x_2 - x_3 \leq 10 \\
x_1 + 2x_2 + x_3 \leq 8 \\
-x_1 + 3x_2 + 6x_3 \leq 60
\]
The solution for the above problem is

\[ Z = 68 \]
\[ x_1 = 6 \]
\[ x_3 = 2 \]
\[ s_3 = 54 \]

The selection based on normal 'e' matrix is

<table>
<thead>
<tr>
<th></th>
<th>( s_1 )</th>
<th>( s_2 )</th>
<th>( s_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>-10 ( x_1 )</td>
<td>5</td>
<td>8</td>
<td>-</td>
</tr>
<tr>
<td>-5 ( x_2 )</td>
<td>5</td>
<td>4</td>
<td>20</td>
</tr>
<tr>
<td>-4 ( x_3 )</td>
<td>-</td>
<td>8</td>
<td>10</td>
</tr>
</tbody>
</table>

i. \( x_1 \) enters \( s_1 \) leaves

ii. \( x_2 \) enters \( s_2 \) leaves

Whereas the selection based on noncompeting variable for the same resource is

<table>
<thead>
<tr>
<th></th>
<th>( s_1 )</th>
<th>( s_2 )</th>
<th>( s_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>-10 ( x_1 )</td>
<td>5</td>
<td>8</td>
<td>-</td>
</tr>
<tr>
<td>-5 ( x_2 )</td>
<td>5</td>
<td>4</td>
<td>20</td>
</tr>
<tr>
<td>-4 ( x_3 )</td>
<td>-</td>
<td>8</td>
<td>10</td>
</tr>
</tbody>
</table>

i. \( x_1 \) enters \( s_1 \) leaves

ii. \( x_3 \) enters \( s_2 \) leaves
Once \( x_1 \) is selected \( x_2 \) is ignored because \( x_2 \) also requires some resource from the first constraint which has been allotted for \( x_1 \). This gives the optimal solution at the end of first pass.

### 6.6.5 Example to show the selection based on resource level

Maximize \( Z = 3x_1 + 2x_2 + 5x_3 \)
subject to

\[
\begin{align*}
3x_1 + 2x_3 &\leq 460 \\
x_1 + 2x_2 + x_3 &\leq 430 \\
x_1 + 4x_2 &\leq 420
\end{align*}
\]

The solution for the above problem is

\[
\begin{align*}
Z &= 1350 \\
x_2 &= 100 \\
x_3 &= 230 \\
s_3 &= 20
\end{align*}
\]

The selection based on '0' matrix is

<table>
<thead>
<tr>
<th></th>
<th>( s_1 )</th>
<th>( s_2 )</th>
<th>( s_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_1 )</td>
<td>( \frac{460}{3} )</td>
<td>430</td>
<td>420</td>
</tr>
<tr>
<td>( x_2 )</td>
<td>-</td>
<td>215</td>
<td>105</td>
</tr>
<tr>
<td>( x_3 )</td>
<td>230</td>
<td>430</td>
<td>-</td>
</tr>
</tbody>
</table>

i. \( x_3 \) enters and \( s_1 \) leaves

ii. \( x_2 \) enters and \( s_3 \) leaves
This shows that an additional pass to bring $s_3$ in place of $s_2$ is needed.

Based on the resource level the problem will be

Maximize \[ Z = 3x_1 + 2x_2 + 5x_3 \]
subject to
\[
\begin{align*}
3x_1 + 2x_3 & \leq 460 \\
2x_1 - 2x_2 + x_3 & \geq 30 \\
2x_1 - 4x_2 + 2x_3 & \geq 40
\end{align*}
\]

\[ i.e., \quad \begin{bmatrix} 3 & 0 & 2 \\ -2 & 2 & -1 \\ -2 & 4 & -2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \leq \begin{bmatrix} 460 \\ -30 \\ -40 \end{bmatrix} \]

The selection based on '$e'$ matrix is

<table>
<thead>
<tr>
<th></th>
<th>$s_1$</th>
<th>$s_2$</th>
<th>$s_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>$\frac{460}{3}$</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$x_2$</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$x_3$</td>
<td>$230$</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

i. $x_3$ enters and $s_1$ leaves

In the second pass $x_2$ enters and $s_2$ leaves. Though both the methods take two passes, in the second one, the number of basic changes as well as the multiply/divide operations are less compared to the first.
6.6.6 Detecting Redundant Constraints

It is a well established fact that every additional constraint in a linear programming problem increases the computational effort. If a constraint is redundant, it consumes extra computations which is a sheer waste. Any redundant constraint should be detected and removed before the commencement of the computations. The 'G' matrix developed in Chapter 3 along with the algorithm developed in section 6.4 of this chapter may be used to detect simple redundant constraints.

Max \( Z = 2.3x_1 + 1.6x_2 + 1.9x_3 + 1.4x_4 \)
subject to
\[
\begin{align*}
2x_1 + 1.5x_2 + 2x_3 + 1.5x_4 &\leq 100 \\
x_1 + x_2 + x_3 + x_4 &\leq 30 \\
x_1 + x_3 &\leq 20 \\
x_2 + x_4 &\leq 20
\end{align*}
\]

optimal solution for the above problem is

\[
\begin{align*}
s_1 &= 45 \\
x_2 &= 10 \\
x_1 &= 20 \\
s_4 &= 10
\end{align*}
\]
\[ \begin{array}{c|cccc} & s_1 & s_2 & s_3 & s_4 \\ \hline x_1 & 50 & 30 & 20 & - \\ x_2 & 66.67 & 30 & - & 20 \\ x_3 & 50 & 30 & 20 & - \\ x_4 & 66.67 & 30 & - & 20 \\ \end{array} \]

As per the selection

i. \( x_1 \) enters \( s_3 \) leaves

ii. \( x_2 \) enters \( s_4 \) leaves

Columns corresponding to variables \( s_1 \) and \( s_2 \) are not scored out and the intercepts in these columns are greater than the intercepts in the unscored columns.

The addition of pivot elements of the selected column is 40 and this is greater than the elements in column 2 but less than column 1. Hence constraint 1 is treated as the redundant constraint. Then the constraints are

\[
\begin{align*}
 x_1 + x_2 + x_3 + x_4 & \leq 30 \\
 x_1 + x_3 & \leq 20 \\
 x_2 + x_4 & \leq 20 
\end{align*}
\]
Using the resource level property, the constraints are modified as

\[
\begin{align*}
x_1 + x_2 + x_3 + x_4 & \leq 30 \\
x_2 + x_4 & \geq 10 \\
x_1 + x_3 & \geq 10
\end{align*}
\]

In this method, \(x_1\) enters and \(s_2\) leaves in the first pass and \(x_2\) enters and \(s_3\) leaves in the second pass leading to the optimal solution in two basic changes, compared to the three basic changes using normal 'θ' matrix selection procedure.

Thus the suggested algorithm is helpful in finding out the redundant constraint at the beginning itself.

6.7 CYCLING

It is generally assumed that the objective function value increases/decreases at each iteration and the final solution is optimal when termination takes place after a finite number of iterations. Unfortunately, the convergence property of the simplex procedure has not been established and the objective function actually stalls for several iterations in many linear optimisation problems[6,13,17].
This anomaly occurs when the solution is degenerate, i.e., when the values of one or more variables become zero. At this point there is no assurance that the value of the objective function will improve. It is then possible that the simplex iterations will enter a loop which will repeat the same sequence of iterations without ever reaching the optimal solution. This problem is said to have the property of cycling [8,13].

Degeneracy and the consequent cycling has been plaguing the computational effort in linear programming problems for some time although this phenomenon has never made it impossible to reach an optimal solution since no practical problem has ever cycled. However, Charne's perturbation method [13] rescues from entering into the vicious circle of cycling. Consider the linear programming problem suggested by Beale [28].

$$\text{Max } Z = 0.75x_1 - 20x_2 + 0.5x_3 - 6x_4$$

subject to

$$0.25x_1 - 8x_2 + x_3 + 9x_4 \leq 0$$
$$0.5x_1 - 12x_2 - 0.5x_3 + 3x_4 \leq 0$$
$$x_3 \leq 1$$

$$x_1, x_2, x_3, x_4 \geq 0$$
The optimal solution of the above problem is

\[ s_1 = 0.75 \]
\[ x_1 = 1 \]
\[ x_2 = 1 \]

The various steps of the simplex algorithm are given in Table 6.5 which illustrates the cycling process.

It may be observed that the basis for iteration zero and six are identically the same.

Table 6.5 EXAMPLE OF CYCLING

<table>
<thead>
<tr>
<th>Iteration</th>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( x_3 )</th>
<th>( x_4 )</th>
<th>( s_1 )</th>
<th>( s_2 )</th>
<th>( s_3 )</th>
<th>Basic variables</th>
<th>( Z )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-0.75</td>
<td>20</td>
<td>-0.5</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>( s_1, s_2, s_3 )</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>-4</td>
<td>-3.5</td>
<td>33</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>( x_1, s_2, s_3 )</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>-2</td>
<td>18</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>( x_1, x_2, s_3 )</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0.25</td>
<td>0</td>
<td>0</td>
<td>-3</td>
<td>-2</td>
<td>3</td>
<td>0</td>
<td>( x_3, x_2, s_3 )</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>-0.5</td>
<td>16</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>( x_3, x_4, s_3 )</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>-1.75</td>
<td>44</td>
<td>0.5</td>
<td>0</td>
<td>0</td>
<td>-2</td>
<td>0</td>
<td>( s_1, x_4, s_3 )</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>-0.75</td>
<td>20</td>
<td>-0.5</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>( s_1, s_2, s_3 )</td>
<td>0</td>
</tr>
</tbody>
</table>

The various steps of the multiplex algorithm are given below.
i. 'Θ' matrix

<table>
<thead>
<tr>
<th></th>
<th>s₁</th>
<th>s₂</th>
<th>s₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>x₁</td>
<td>0</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>x₂</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>x₃</td>
<td>-</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>x₄</td>
<td>0</td>
<td>0</td>
<td>-</td>
</tr>
</tbody>
</table>

ii. Selection of variables.

As per the rules given in Chapter 3, the minimum positions are as marked in the table. Hence as per the selection

- x₁ enters s₂ leaves
- x₃ enters s₃ leaves
- x₄ enters s₁ leaves

iii. Update $M^{-1}$ matrix.

When the $M^{-1}$ matrix is updated at the time of entering, $x_1$ and $x_3$ are promising. But $x_4$ turns out to be nonpromising giving the optimal solution at the end of first pass. This is given in Table 6.6.

Table 6.6

<table>
<thead>
<tr>
<th>End of pass</th>
<th>x₁</th>
<th>x₂</th>
<th>x₃</th>
<th>x₄</th>
<th>s₁</th>
<th>s₂</th>
<th>s₃</th>
<th>Basic variables</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-0.75</td>
<td>20</td>
<td>-0.5</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>s₁, s₂, s₃</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>10.5</td>
<td>0</td>
<td>1.67</td>
<td>1.25</td>
<td>s₁, x₁, x₃</td>
<td>1.25</td>
</tr>
</tbody>
</table>
Thus the multiplex algorithm gives the optimal solution in a single pass. The phenomenon of cycling seems to depend on the initial basis with which one commences to search for optimal solution. If an advantageous basis is chosen, then cycling does no longer plague computations. This is another advantage of the proposed multiplex algorithm.

6.8 CONCLUSIONS

In this chapter, an attempt has been made to illustrate further uses of the 'θ' matrix developed in Chapter 3, to minimise variables popping in and out of the basis, to illustrate few modified selection procedures, to minimise popping tendency and to detect simple redundant constraint in the linear programming model. Computational experiences show that the greatest change optimality criterion minimises waste of computational effort and thus improves the efficiency of computation. The 'θ' matrix may also be used to detect simple redundant constraints which when removed before the commencement of computation may lead to additional savings. The problem of cycling has also been overcome by the multiplex algorithm. In the following chapter, the programs developed for the revised simplex and the multiplex algorithms and experiences in developing the software are explained.