CHAPTER 2

ALGORITHMS TO SOLVE LINEAR PROGRAMMING PROBLEMS

2.1 INTRODUCTION

The solution to a Linear Programming Problem (LPP),

Optimize $Z = CX$.
subject to $AX \leq, = \geq P_0$ and $X \geq 0$

is obtained by using a tool called the Simplex Algorithm. An improved and efficient version of the Simplex Algorithm used for solving large scale LPP on a digital computer is the Revised Simplex Algorithm (Orchard-Hays 1968). The Revised Simplex Algorithm also makes use of the same basic principles of the Simplex Algorithm. But at each iteration, the entries in the entire table are not updated. The necessary information to move from one basic feasible solution to another is directly generated from the original set of matrix coefficients.

Klee and Minty (1972) have shown that it is possible to construct artificial problems for which the Simplex Algorithm would require the examination of a number of vertices which increase exponentially the size of the problem. The complexity of the Simplex Algorithm is, therefore, exponential.

In 1979, Khachian published the first polynomial algorithm for linear programming. Although this first polynomial method has many

important theoretical consequences, it has been proved in practice that this algorithm is hopelessly slow.

In 1984, Karmarkar proposed a new polynomial algorithm and claimed that this so-called projective algorithm could solve large scale linear programming problems a 100 times faster than the Simplex Algorithm. He has also shown that its worst running time is 0 ($n^{3.5}$ L^2), (n is the number of variables and L is the length of the input - the number of bits used to represent the problem data) and so it is better than that of Khachian's algorithm.

This chapter presents a brief outline of the existing algorithms that are useful in solving Linear Programming Problems and its variants.

2.2 THE REVISED SIMPLEX ALGORITHM

Actually, the birth of linear programming is usually identified with the development of the "Simplex Algorithm" developed in 1947 by Dantzig. For long it remained a major algorithm used in optimization problems. The Simplex Algorithm starts with a non-optimal basis and then updates the basis until it reaches the optimal basis. Each iteration of the method brings one variable (not in the current non optimal basis) into the basis and sends one variable out (in the current non optimal basis) of the basis. Unfortunately, the pricing rule does not guarantee that variables removed from the basis once will never reappear in the optimal basis. Therefore, each variable has to be kept alive during the pricing process. The Simplex Algorithm may be viewed as a procedure for systematically estimating the set of constraints that are binding at a solution. More specifically, if the current estimate is a feasible (but not-optimal) vertex, the next estimate is chosen as an adjacent feasible vertex at which the objective function has an improved value.

In 1953, Dantzig modified the simplex calculations so that the implementation of the Simplex Algorithm on a computer could be done more efficiently. This is called the Revised Simplex Algorithm.

2.2.1 The Procedure of the Algorithm

The Revised Simplex Algorithm saves both storage and computational time compared to the Simplex Algorithm. Unlike the original Simplex Algorithm, only the inverse of the current basis is maintained to generate the next inverse in the Revised Simplex Algorithm. All other quantities except X_B are computed from their definitions as and when necessary. Even though X_B can be computed from its definition, it is more economical to transform it at each stage.

Consider the LP model

$$\text{Maximize} \quad Z = C\,X$$

$$\text{subject to } A\,X = P_0\,; \quad X \geq 0$$

where

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

$$P_0 = \begin{bmatrix} b_1 \\ b_2 \\ . \\ b_i \\ . \\ . \\ b_m \end{bmatrix}, \quad X = \begin{bmatrix} x_1 \\ x_2 \\ . \\ x_j \\ . \\ . \\ x_n \end{bmatrix}$$

and $C = (c_1\ c_2\ \ldots\ c_j\ \ldots\ c_n)$

Let the columns corresponding to the matrix A be denoted by P_1, P_2, P_3, ... P_n where

$$P_1 = \begin{bmatrix} a_{11} \\ a_{21} \\ . \\ a_{i1} \\ . \\ . \\ a_{m1} \end{bmatrix}, \ldots, P_n = \begin{bmatrix} a_{1n} \\ a_{2n} \\ . \\ a_{in} \\ . \\ . \\ a_{mn} \end{bmatrix}$$

The various steps involved in the Revised Simplex procedure are as follows:

Step 1

i. Compute $Y = C_B B^{-1}$

ii. Calculate $z_j - c_j = YP_j - c_j$ for every non basic variable x_j

iii. If all $(z_j - c_j) \geq 0$, then the optimal solution is reached and is given by $X_B = B^{-1} P_0$ and $Z = C_B X_B$.

Otherwise, let $z_k - c_k = \min_j \{z_j - c_j : z_j - c_j < 0\}$, $j = 1,2 \ldots n$

If there is a tie, the one with the smallest index is selected (Bland 1977).

Step 2

i. Calculate $\alpha^k = B^{-1} P_k$

ii. If $\alpha^k \leq 0$, then the problem has unbounded solution.

Otherwise, let $\dfrac{(B^{-1}P_0)_r}{\alpha_r^k} = \min_i \left\{ \dfrac{(B^{-1}P_0)_i}{\alpha_i^k} : \alpha_i^k > 0 \right\}$,

$i = 1,2 \ldots m$

If there is a tie, the one with the smallest index is chosen (Bland 1977).

Step 3

Construct the η vector for x_k by using the following formula

$$\eta = \frac{1}{\alpha_r{}^k} \begin{bmatrix} -\alpha_1{}^k \\ -\alpha_2{}^k \\ -\alpha_3{}^k \\ \cdots \\ \cdots \\ -\alpha_{r-1}{}^k \\ 1 \\ -\alpha_{r+1}{}^k \\ \cdots \\ \cdots \\ -\alpha_m{}^k \end{bmatrix}$$

Step 4

Use the η vector to construct the E matrix which is a transformation matrix having (m-1) unit vectors and only one non unit vector η in place of the r^{th} column.

Step 5

i. Update the basis inverse by using the formula $B^{-1}{}_{next} = EB^{-1}$
ii. Set $B^{-1} = B^{-1}{}_{next}$
iii. Goto step 1

2.2.2 Forward and Backward Transformation

If x_k is to enter the basis at iteration t, then the computation of the vector $\alpha^k = B^{-1} P_k$ is performed sequentially via.

$B^{-1}{}_t P_k = E_{t-1} (E_{t-2} (E_{t-3}(...(E_2(E_1P_k))...)))$, by first premultiplying P_k by E_1, next premultiplying the resulting vector by E_2 and so forth. This operation is called a forward transformation (FTRAN), because it corresponds to a forward scan of the eta file.

A computation of the form $C_B B^{-1}$ is performed sequentially via.

$C_B B^{-1}{}_t = ((...(((C_B E_{t-1})E_{t-2})E_{t-3})...)E_2)E_1$ by first postmultiplying C_B by E_{t-1}, next postmultiplying the result by E_{t-2}, and so forth. This operation is called a backward transformation (BTRAN), because it corresponds to a backward scan of the eta file (Bazaraa *et al* 1990).

2.3 VARIANTS OF THE SIMPLEX ALGORITHM

The Revised Simplex is a well founded algorithm to solve linear programming problems. That is, a better feasible solution is obtained from the current feasible solution by changing one vector in the basis at a time. This algorithm adopts a technique to utilize the computational results of the previous iteration in the current iteration so that an improvement in the solution can be obtained. But the number of iterations required for most large scale problems to converge to optimal solution can be depressingly large, and consequently, the amount of computer time spent is quite substantial. As a result, a great deal of effort and thought has been put in to develop techniques, whose purpose is to reduce the computational efforts taken in solving large scale linear programming problems.

A few of the algorithms which are expected to reduce the computational effort are discussed briefly in this section.

Chipman (1953), Dickson and Frederick (1960), Paranjape (1965), Graves *et al* (1963) and Orchard-Hays (1968) have proposed a number of modified algorithms such as starting initial basis, crashing techniques, selecting multiple column, method for replacing of two basic variables at a time and a suboptimizing to reduce the computational effort.

The Simplex Algorithm allows considerable flexibility in the choice of the pivot element on each iteration and so dozens of variants have been

developed (see for example, Dantzig 1963, Jeroslow 1973, Harris 1973, Murty 1976, Bland 1977, Goldfarb and Reid 1977, Fukuda 1982, Clausen 1987, Klafszky *et al* 1989 and Zhang 1991).

2.3.1 Crash Algorithm

When a linear programming problem has both lower and upper bound inequalities, the introduction of slack variables, in all the inequality constraints makes the starting solution infeasible. In order to find out the solution for the problems mentioned above, either artificial variable technique or the two phase technique is employed. Attempting to achieve a feasible solution, from a solution that is infeasible at start is known as crashing (Beale 1969). This involves making a sequence of iterations, omitting the step of finding out the entering variable using the (z_j-c_j) criterion. Instead each variable is considered simply in turn. One variant is as follows : Start with a basis consisting of slacks on all inequality constraints and artificials on all others. Make one pass through the matrix, performing a forward transformation on each structural variable in turn and making it basic without creating any new infeasibilities if this

a) removes an artificial variable from the basis
b) reduces the number of infeasibilities and
c) has a negative reduced cost

If any of these conditions holds, then update B^{-1} matrix and repeat the step for the next variable. Otherwise abandon this variable and try the next one (Beale 1969).

2.3.2 The Multiple Column Selection Algorithm

A related improvement over the crash algorithm is known as the multiple column selection algorithm (Beale 1969). The step by step procedure of this method is as follows.

Step 1 : The (z_j-c_j) values of all the variables are computed. The solution is checked for optimality. If it is non-optimal then go to Step 2. If the solution is optimal verify feasibility. In case of infeasibility, invoke the Dual Simplex procedure to remove the infeasibility. The process is terminated when the solution is feasible.

Step 2 : Two or more promising vectors are selected.

Step 3 : All the selected columns and P_0 are updated by premultiplying them by B^{-1}.

Step 4 : The ratio test is performed on all of them to determine the entering vector which makes the greatest change in the objective function. The ratio test is defined as follows:

$$ß = \min_j (z_j-c_j)\, ß_j$$

$$\text{where } ß_j = \min_i \left[\frac{(B^{-1}P_0)_i}{(B^{-1}P_j)_i} \quad : \quad (B^{-1}P_j)_i > 0 \right]$$

$$i = 1, 2, \ldots, m$$

Step 5 : Update the B^{-1} matrix and the other selected columns. Return to Step 4 to see whether another variable can be gainfully introduced. Continue the procedure until no further progress is possible with the selected variables and then go to Step 1.

2.3.3 Suboptimization

Unlike the multiple column selection method, the variables which become nonbasic during a pass, may be considered subsequently for pivoting in the same pass. This procedure is known as suboptimization (Aronofsky 1969), since one is effectively solving a subprogram containing a very limited number of columns (non basic variables) by the straight Simplex Algorithm. But it is doubtful whether complete optimization is worthwhile, since a lot of time may be wasted with variables popping in and out of the basis.

Thus the crashing techniques, the multiple column selection procedure and the suboptimization procedure all select a subset from the nonbasic part of the matrix which can be exchanged for a set of basic vectors. The order in which the vectors of the subset enter the basis and the choice of the pivot rows however, are not determined by a single set of criteria. It is this lack of fixity which has led to the emergence of a wide variety of algorithms.

2.4 FRISCH'S MULTIPLEX METHOD

Considering all the methods discussed so far, it can be stated that the improvement in the solution will be obtained by moving from one vertex to another vertex of the constrained set. The Multiplex Method (Frisch, 1955) selects more than one variable at a time to construct a basis. This procedure cuts across the feasible region in search of a vertex unlike the Simplex Procedure which religiously sticks to the periphery of the constrained set. The step by step procedure of this method is described below.

Consider a linear programming problem of n+m variables which satisfies m linearly independent equations. Let there be n degrees of

freedom. Let $X_u, X_v, \ldots X_w$ be a basic set of variables. The equations are expressed in terms of a basic set

$$X_j = b_{j0} + \sum_{k=u,v,\ldots w} b_{jk}X_k,\ j = 1,2,\ldots n+m$$

where the b_{j0} and b_{jk} are constants.

Consider the linear preference function

$$f = p_0 + \sum_{k=u,v,\ldots w} p_k X_k$$

Step 1 : An initial point is determined which satisfies the equations.

Step 2 : A movement in the preference direction is defined as $X_k = X_k^0 + \lambda p_k$, $k = u,v,\ldots w$ where X_k^0 are the coordinates of the initial point, $\lambda \geq 0$.

Variables not in the basic set are given by

$$X_j = X_j^0 + \lambda d_j,\ j = 1,2,\ldots n+m$$

where $d_j = \sum_{k=u,v,\ldots w} b_{jk}p_k,\ j = 1,2,\ldots n+m$

Step 3 : The largest permissible value in the parameter λ is calculated. This gives the optimal solution. Otherwise make the new point as the initial point and repeat Step 2.

This method provides an alternative to the Simplex Algorithm and might also shorten computation time considerably by shrewd guessing.

At each point, selection of the number of variables which are independent (columns) is arbitrary and hence the number of variables plays an important role in identifying the optimal solution.

2.5 KHACHIAN'S ELLIPSOID ALGORITHM

Today there is a general agreement among researchers that an algorithm is a practically useful tool for solving a computational problem only if its complexity grows polynomially with respect to the size of the input.

The complexity of the Simplex Algorithm is, therefore, exponential when confronted with the exponential worst case behaviour of the Simplex Algorithm. The question of whether a polynomial time algorithm for solving linear programming problems exists, remained open until 1979, when it was answered in the affirmative by Khachian.

In 1979, Khachian proposed a polynomial time algorithm for determining a solution (if one exists) to the open set of linear inequalities $S = [x:Ax \leq b]$, where A is m x n matrix, $m,n \geq 2$, and where A and b have all integer components. This algorithm either finds a solution in S, if one exists, or else concludes that S is empty.

The principle of the algorithm is similar to that which underlies the catching of fish in a net, over such a large region so that some of what is wanted must be inside, when the volume of the net is gradually decreased. When the volume is sufficiently reduced it becomes obvious whether or not any fish has been caught (Berresford 1980). Similar to this, a large ellipsoid is formed at the beginning. Depending upon the discrepancies in the solution vector the size of the ellipsoid is gradually decreased. When there is no discrepancy in the solution vector, the optimal solution is reached. This algorithm requires $O(mn^3L)$ arithmetic operations in the worst case, and each operation is performed to a precision of O(L) bits where $L = \log_2$ (largest absolute value of the determinant of any square submatrix of A) + $\log_2$ (max C_i) + $\log_2$ (max b_i) + $\log_2$ (m+n).

Although initially it was thought that the ellipsoid algorithm might be as fast in practice as the Simplex Algorithm, these hopes have not been realized, due to following difficulties.

i. The number of iterations tends to be very large
ii. The computation associated with each iteration is costlier than a simplex iteration because sparse matrix techniques are not applicable.

2.6 KARMARKAR'S INTERIOR POINT ALGORITHM

The appearance of Karmarkar's Algorithm in 1984 for linear programming generated much excitement in the mathematical community. Also known as the projective transformation method, Karmarkar's Algorithm was the first polynomial-time linear programming algorithm to compete viably with the Simplex Algorithm on real-world problems. Like the ellipsoid algorithm, Karmarkar's Algorithm almost completely ignores the combinatorial structure of linear programming.

One may recall that the polynomial-time Ellipsoid Algorithm generates a sequence of points outside the feasible region of the linear program. The ellipsoid algorithm is an exterior-point method. Simplex Algorithm generates vertices in the feasible region. Unlike the Simplex Algorithm, Karmarkar's Algorithm is an interior-point method. It generates a sequence of points inside the feasible region whose costs approach the optimal cost. In the end, it jumps to a vertex of no greater cost, which is then optimal.

This algorithm addresses linear programming problems of the form

Minimize CX
Subject to $AX = 0$
$e^T X = 1, X \geq 0$

where A is m x n matrix of rank m,n ≥ 2, C and A are all integers, e is a row vector of n ones, and with the following, two assumptions hold good:

i. The point X_0 = (1/n, 1/n, ... 1/n) is the starting feasible solution.
ii. The optimal objective value of problem is zero.

Karmarkar (1984) has proved that his algorithm's running time is a polynomial function of the problem size event of the worst case. He showed that if n is the number of variables in a problem and L is the number of bits used to represent the problem data, the theoretical worst case running time is $0(n^{3.5}L^2)$. In other words as the problem size increases, the running time tends to be a constant multiple of $n^{3.5}L^2$. This is substantially better than the ellipsoid algorithm's worst case running time. Two major difficulties found in the original algorithm are that

i. it requires prior knowledge of the exact minimal objective value which is usually unknown in advance and
ii. it does not generate the optimal dual solution which is significant in sensitivity analysis.

2.6.1 Some Variants of the Algorithm

Since the introduction of the polynomial-time algorithm for Linear Programming by Karmarkar (1984), there has been considerable interest in its use, and in developing strategies for modifying the algorithm to achieve greater computational efficiency. For example, Todd and Burrell (1986) have proposed a polynomial method that involves the use of dual variables in Karmarkar's linear programming canonical form. Their method generates not only a sequence of interior feasible solutions, but also a sequence of objective lower bounds that converge to the minimal objective value.

In order to achieve both polynomial complexity and practical efficiency in overcoming the two difficulties mentioned above in Karmarkar's original algorithm, Ye and Kojima (1987) have combined the polynomial method of using dual variables (Todd *et al* 1986) and the cutting objective technique (Ye 1985) and developed a polynomial variant of Karmarkar's Algorithm in the standard form. Vaidya (1987) has presented an algorithm for the linear programming problems which requires O $((m+n)n^2 + (m+n)^{1.5}$ n)L) arithmetic operations in the worst case, and it is adequate to perform each arithmetic operation to a precision of O(L) bits. This algorithm is faster than Karmarkar's Algorithm by a factor of $\sqrt{m}$ for all values of m and n. Renegar (1988) has provided an algorithm for linear programming which requires $0(m^{1.5}n^2L)$ arithmetic operation. Venkaiah (1989) has also presented a simple but efficient algorithm for solving linear programming problems. In this algorithm, the projection matrix is calculated once throughout the computation unlike in Karmarkar's Algorithm where the projection matrix is computed at each and every iteration. But the algorithm path followed in this is the same as that of Karmarkar's Algorithm.

Since Karmarkar (1984) published his projective algorithm for the solution of the linear programming problem, a wide variety of interior point methods for linear programming has been proposed. Lustig (1991) has developed a new insight into deriving feasible points using an interior point method to solve linear programming problems.

2.6.2 Some Remarks on the Convergence

In this section the convergence of Karmarkar's Algorithm is analysed by solving some linear programming problems.

The number of iterations taken for solving a linear programming problem is mainly dominated by the step length 'α' value (Singh, *et al* 1994) Karmarkar suggests a formula to find out 'α' which is (n-1)/3n.

For example, solving the following linear programming problem with $\alpha = 1/6$

Max $Z = 4x_1 + 10x_2$

subject to

$$2x_1 + x_2 \leq 50$$
$$2x_1 + 5x_2 \leq 100$$
$$2x_1 + 3x_2 \leq 90$$
$$x_1, x_2 \geq 0$$

The optimal solution is obtained in the 202^{nd} iteration. But select α value which will be larger than this, but is less than one according to Karmarkar. With $\alpha = 0.75$, the optimum solution is obtained in the 83^{rd} iteration.

The Table 2.1 gives the relationship between 'α'-value and the number of iterations obtained by solving various size LP problems.

Table 2.1: Relationship between 'α' value and the number of iterations in the Karmarkar's Algorithm

Size of A matrix	$\alpha = 0.25$	$\alpha = 0.75$	$\alpha = 1.0$
2 x 4	159	53	47
3 x 6	217	74	54
4 x 8	221	78	58
5 x 10	263	89	68
6 x 12	294	103	77

It is clear that the number of iterations is decreased when 'α' value is increased. By choosing suitably large value for 'α' the number of iterations is considerably reduced, and this is clear from the above table.

2.7 CONCLUSION

In this chapter, various algorithms available for solving linear programming problems have been highlighted. Several attempts have been made to enter two or more promising vectors into the basis and it has been pointed out that these methods suffer from one defect or the other. These defects can be described thus :

i. Multiple column selection is made on heuristics and lacks formalism

ii. An arbitrary selection of columns may lead to a singular basis matrix.

iii. Even if the basis matrix is non-singular, the solution may be infeasible.

iv. The selection of starting basic variables is arbitrary and hence depending on the choice, the number of iterations may increase or decrease.

Since 1955, they had been using Multiplex method to solve linger programming problems. In 1984 Sakthivel refined this method so that it could be used much more effectively in solving the problems overcoming the drawbacks mentioned above. This algorithm brings into the basis more than one variable in each pass. In the next chapter the computational aspects of the Multiplex Algorithm and its variants are discussed.