Karmarkar's Algorithm: Extensions and Implementation

Abdellah Salhi

Doctor of Philosophy

The University of Aston in Birmingham

September 1989

The University of Aston in Birmingham

Karmarkar's Algorithm: Extensions and Implementation

Abdellah Salhi

Doctor of Philosophy

September 1989

Summary

Linear Programming (LP) is a powerful decision making tool, extensively used in various economic activities. Its success is mainly due to the efficiency of the simplex method. In recent years, however, new techniques have emerged.

The present work is concerned with investigating one such technique, namely Karmarkar's algorithm and its variants, extending it to structured linear programming problems and efficiently implementing it, taking account of sparsity.

A review of recent work on the algorithm and early polynomial time methods for LP such as the ellipsoid and the simplicial algorithms is presented. The performance of the simplex method is also discussed.

One of the major developments in Karmarkar's algorithm is the discovery of dual variants. Duality allows the method to be simply extended to problems having an unknown optimum objective value and also to investigate postoptimality analysis. The study showed that postoptimality analysis is possible with Karmarkar's algorithm in the three cases considered (cost, right-hand side and rim).

Based on Ye and Kojima's dual variant, a specialized form of the algorithm for structured LP is presented together with computational results on various problems. The results show that inherent parallelism of some linear programming problems can be efficiently exploited with Karmarkar type algorithms. The advantages of decomposition are also discussed in a wider context (eg. lack of favourable structure).

Finally, an efficient implementation of a variant of the Karmarkar algorithm, which combines sparsity-preserving techniques for least squares, such as the nested dissection ordering algorithm and updating techniques is described. The performance of this implementation on realistic LP problems is reported.

Key Words: Linear Programming, Least Squares, Karmarkar's Algorithm, Duality, Partitioning.

In memory of
Mohand Tayeb Salhi

Acknowledgements

I wish to express my thanks to the following people:

Mr. George R Lindfield, my supervisor, for his support, guidance and being available whenever needed,

Miss Jacqueline M Archibald for her support, encouragement and friendship throughout this work,

Dr. Meng Hua for his advice and friendship,

Dr. Brian Gay for all his help,

all members of staff of the Department of Computer Science & Applied Mathematics, and Computer Services, especially Dr. Les Hazlewood, Neil Toye, Malcolm MacGregor, Dave Stops and Roy Parsons, for their technical help and advice,

all my fellow research students, especially Rob Thomson and Ian Hardy, for their stimulating ideas and lively debates.

Finally, I acknowledge the financial support provided by the Algerian Government.

List of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 A Brief History

Linear Programming (LP) developed from the Twentieth Century's need to solve problems of production management. Although known to Kantorovich (1939), it effectively started in 1947 with the design of the simplex method of G.B.Dantzig for solving optimum planning problems. A period of rapid developments and exciting discoveries in this new field followed and continue today. In the post-war era LP has provided a good framework for the analysis of classical economic theories such as the Walras mathematical model of economy and Leontief Input-Output model. It has also been successfully used to bring together different fields of pure mathematics such as convex sets theory, combinatorics and two-person game theory.

Before LP, various problems of production management were solved by a trial-and-error approach guided only by experience and intuition. Later, most of those problems were stated in terms of LP and systematically solved by the simplex method.

It is when combined with the computer that LP is most effective. The widespread use of LP is mainly due to this combination. Indeed, a large proportion of computing power, is devoted to solving present day large scale LP problems. This is well expressed in [Lovász, 1980]:

"If one would take statistics about which mathematical problem is using up most of the computer time in the world, then (not counting database handling problems like sorting and searching) the answer would be linear programming"

From the beginning, the simplex method was effective on almost any type of LP problem. Over the years it has been further polished and new variants of it have been developed by Gass, Lemke, Orchard-Hays and others (see Dantzig, 1963). Today, it is a practical and robust decision making tool, which stands on a firm theoretical basis.

In recent years, however, with the progress of complexity theory, most algorithms have come under scrutiny and their efficiency questioned. The simplex was no exception, and was shown to run in exponential time for an artificially built class of LP problems [Klee & Minty, 1972]. This important result encouraged the debate over the efficiency of the simplex and a crucial question arose: Is LP in the P-class or NP-class?

Before going any further, we ought to define some terminology borrowed from complexity theory; the definition may help to see how this theory contributes to understanding algorithms and evaluating their performance.

1.1.1 Algorithm and Problem Complexity

Usually, for a given problem, a range of algorithms may be used to solve it. As a random choice may not be suitable, it is useful to have some criteria for identifying a specific algorithm. These criteria are the amounts of CPU time and storage required to run a code of the algorithm on a computer [Lovász, 1984].

One of the main concerns of complexity theory is to find, for a given algorithm, a bound on its running time, i.e. its time complexity function, and a bound on the space requirement, i.e. its space complexity function. The time is usually the only factor considered. However, the theory can be extended to storage. In finding these bounds, the problem difficulty is also investigated. This allows us to separate problems into different complexity classes. Hence, algorithm complexity and problem complexity go hand in hand, although a distinction between them should be made. Algorithm complexity is the cost of a particular algorithm, while problem complexity is the minimal cost over all possible algorithms [Traub & Wozniakowski, 1982].

We have already mentioned two complexity classes: The P-class and the NP-class. The P-class, probably the most studied, contains problems for which a polynomial time algorithm has been found, on deterministic computers (like the ones we use in the real world). A polynomial time algorithm is one with a running time bound, (worst case complexity), which is a polynomial function of the length of the problem data (eg. 2n, n^3+n, etc...), or behaves asymptotically like one (eg. logn, nlogn, n^6logn, etc...) [Garey & Johnson, 1979; Kronsjö, 1985].

The NP-class contains problems for which a polynomial time algorithm can be found only on a non-deterministic computer. Non-deterministic computers are pure mathematical inventions. On real life computers only exponential time algorithms can be found for them. These algorithms have time bounds which are exponential functions, or behave similarly, (eg. e^n+n, 2^n, etc...). The hardest problems in the NP-class form the NP-Complete class. Intuitively, problems in the NP-class are of the form, 'determine whether a solution exists.' Their complementary problems are of the form, 'establish that there are no solutions'. They constitute the CO-NP-class, [Kronsjö, 1985].

As early as 1953, von Neumann made the distinction between polynomial and exponential time algorithms. However, it was not until 1965 that the class of problems solvable by polynomial algorithms, was identified (see Cook, 1983). This was due to Edmonds (1965) who first thought that exponential time computability approximately indicates how difficult a problem is. Consequently, he introduced the notions of "easy" and "hard" problems and "good" and "bad" algorithms.

In practical terms, this idea of classifying problems and algorithms is not totally justified. Indeed, many reliable and practical algorithms, such as the simplex method, are known to run in exponential time for some cases, and many good algorithms in theory are inefficient in practice (appropriate examples will be given in the next section). It is in this respect that the average run time, (average complexity), is relevant to understanding the behaviour of algorithms. However, average time bounds are more difficult to derive, as *a priori* probability distributions on the data must be postulated [Lovász, 1984].

Until recently, the LP problem was believed to be in the NP-complete group. It was thought that the discovery of a polynomial time algorithm for LP would bring an answer to the outstanding question of whether P=NP. As will be seen in the following sections, such an algorithm has been discovered, which shows that LP is in the P-class. However, a closer study of the problem's properties revealed that linear programming has the properties of the NP as well as the CO-NP groups. Because there is strong evidence that

NP≠CO-NP, LP can only be in one of them. Further studies supported the argument that LP is not a member of the NP-class, [Garey & Johnson, 1979; Kronsjö, 1985].

1.1.2 Developments in Linear Programming

The rules for pivot selection of the simplex algorithm are a decisive feature in its performance. Many new rules, were proposed [Bland, 1977], but soon problems for which those variants lacked efficiency were also constructed. Note that these problems have not been observed to occur in the real world, and were appropriately labelled pathological. However, there is a wide agreement that "well solved" problems are those for which polynomial time algorithms were found, [Garey & Johnson, 1979]. This is because exponential time algorithms are only intelligent variations on exhaustive search, which implies they are costly in terms of computing time. It was thus understood, at least from the theoretical point of view, that the LP problem was still not "well solved". The search for a polynomial time algorithm for LP continued, encouraged by the need to answer the theoretical question about the class of LP, and also by the thought, that an algorithm with a polynomial time worst case bound would increase the efficiency of managing operations beyond what the simplex provided so far.

In 1979 such a polynomial time algorithm was discovered by the Russian mathematician Leonid Genrikovitch Khachyan. The algorithm was designed primarily to recognize compatible systems of linear inequalities in polynomial time in the length of the data. The underlying idea is reminiscent of the binary search. The latter can be briefly described [Papadimitriou & Steiglitz, 1982] as follows. Suppose that an integer x is to be determined in the inteval [1, Z] by performing the test "Is $x > b$?", for some chosen value b. The obvious way is to take b in the middle of the interval, thus splitting it into two parts. The outcome of the test will allow us to drop one part of the interval, and continue the search in the remaining part. It is stopped when the final interval contains exactly one integer, which is x, and this happens after $n = \lceil \log (Z) \rceil$ tests. Similarly, the Khachyan algorithm strives to restrict the search for a solution to one part of the solution set and discard the other one. This is achieved by the use of ellipsoids whose volumes decrease at every iteration. The process will be explained later. It should already be mentioned that to apply the algorithm to the linear programming problem, the latter should be converted into a set of strict linear inequalities. Indeed, the equivalence of LP and strict linear inequalities is a key feature in the theory of Khachyan's algorithm. Khachyan (1979) established that LP problems defined in the set of integer numbers can be solved in O(n^6Logn) arithmetic operations.

Much work followed the discovery of the ellipsoid algorithm, and the bulk of it was aimed at producing practical implementations and codes. However, despite persistent efforts, no implementation of the algorithm seems to be as efficient as the simplex codes. In fact the algorithm performed better on Nonlinear Programming Problems (NLP). It became clear that in terms of practical value, the worst case bound is not very significant. As a consequence it was thought [Smale, 1983; Avis & Chvátal, 1978] that bounds on the average performance (average case bounds) of algorithms may be the key to understanding their behaviour. Consider, for instance, the number of pivot steps on average taken by the simplex or its variants on problems encountered in practice as well as randomly generated. This number must be more significant for practical purposes than that taken on a special class such as the Klee-Minty problems. Studies on the average performance of the simplex have been undertaken since the early 50's. A good account of the outcome from these studies may be found in [Shamir, 1987]. This issue will be further discussed later.

The search for other polynomial time algorithms for LP continued and in 1982 A. Ju Levin and Boris Yamnitsky showed that the role of ellipsoids in the Khachyan algorithm can be played by simplices. The algorithm was shown to be polynomial in the size of the input data, with a better bound than that of the ellipsoid algorithm. The simplicial algorithm was based on an early algorithm of Levin (1965). The 1982 version is characterized by an implementation of Levin's idea so that the algorithm runs in polynomial time.

With these discoveries, much interest has been paid to the non-combinatorial aspects of LP, on which the ellipsoid and the simplicial algorithms are based. The interior-point or nonlinear programming approach to LP is also not new. Brown and Koopmans, mentioned in [Charnes *et al.*, 1984], as early as 1951, considered the idea of moving through the polyhedron of the solution set rather than from vertex to vertex in search of the optimum, which is the way the simplex method proceeds. The Brown-Koopmans algorithm proceeds as follows: (a) Start with a point in the constraint set, (b) move in the direction of the objective functional vector until a constraint boundary is reached, (c) make a lateral move orthogonal to this direction staying inside the constraint set, (d) repeat the process until an approximate solution is reached.

The difficulty to maintain feasibility and the slow convergence near the boundaries constitute the major drawbacks of the Brown-Koopmans algorithm. Variants which aimed to guarantee the feasibility of the sequence of points generated, were developed using logarithmic potential and penalty functions [Fiacco & McCormick, 1968]. However, the increase in the size of the problem and the necessity to solve a sequence of nonlinear programming problems arising from the transformation of the original LP problem, made these variants uncompetitive with the standard simplex method.

1.1.3 A New Generation of Polynomial Time Algorithms

The basic idea of the Brown-Koopmans algorithm was considered again by Narendra Z. Karmarkar of AT&T Bell Labs and led to the development of yet another polynomial time algorithm for LP [Karmarkar, 1984a, 1984b]. The algorithm has worst case bound of $O(n^{3.5}Logn)$ and is of substantial improvement over the ellipsoid and simplicial algorithms. Karmarkar's innovation resides in the way feasibility is guaranteed after each iteration. The use of Projective Geometry and a logarithmic potential function to measure convergence and polynomial complexity is central to the algorithm. With this new technique, LP appears to be invariant under rescaling. In other words the change of the scale unit does not affect LP problems. Indeed, the algorithm is basically a rescaling process. This is the main feature of the Karmarkar algorithm and the new breed of related algorithms.

Karmarkar's algorithm works in a transformed space of the original LP problem. The process is an optimization over a sphere inscribed in a (unit) simplex. At each iteration a step is taken from the centre of the sphere in the direction of the negative projected gradient of a special objective function with optimum value zero, on the null space of the constraints matrix. The resulting point is guaranteed to be feasible by appropriate choice of a steplength α. At the end of each iteration a projective transformation (rescaling) is used to bring back the current point into the centre of the inscribed sphere, and the current simplex into itself. A minimum amount of reduction in the objective function and especially in the logarithmic potential function is guaranteed at each iteration. The process is then repeated.

Karmarkar's claim that the method may be up to 50 times faster than the standard simplex method caused a stir in the Mathematical Programming Community. However, this claim was not supported by any published experimental results. The first experimental results obtained outside the AT&T Bell Labs were not as good as expected. The first difficulties with the algorithm came from the computation of the projected gradient. It constitutes the bulk of the work needed at each iteration and is more costly than one iteration of the simplex method. For many it is a serious contender to simplex method as a standard way for solving LP problems. However, many aspects of LP such as duality, sensitivity analysis, sparsity exploitation, remain to be investigated in the frame of Karmarkar's algorithm and its performance evaluated on a wide range of problems before it can be fully adopted.

1.2 LP Problem: Statement, Notation and Terminology

A l'intention du novice, we would like to state the general linear programming problem and equivalent forms before going into the details of the present work. The notation will be consistently followed in subsequent chapters. Other forms and symbols will be defined when introduced.

1.2.1 The General Form

The general problem of linear programming is the search for the optimum (maximum, minimum) of a linear function of variables subject to linear relations (equations or inequalities) called constraints. Some constraints are specific to some or all variables: The non-negativity constraints ($x_j \geq 0$) and the non-positivity constraints ($x_j \leq 0$). Some or all variables can be arbitrary. It is, however, very common to impose *a priori* the condition of non-negativity on all variables in economic problems.

According to the above definition, the algebraic formulation of the general LP problem [Simonnard, 1966] is:

$$
(\mathrm{GLP}) = \begin{cases}
\text{min (or max)} \quad z = \sum_{j=1}^{n} c_j x_j & \\
\text{subject to} \quad \sum_{j=1}^{n} a_{ij} x_j \geq b_i, & i = 1, \dots, p, \\
\qquad\qquad \sum_{j=1}^{n} a_{ij} x_j = b_i, & i = p+1, \dots, m, \\
\qquad\qquad x_j \geq 0, & j = 1, \dots, q, \\
\qquad\qquad x_j \ \text{arbitrary}, & j = q+1, \dots, n, \\
\text{where } a_{ij},\ b_i,\ c_j,\ x_j \text{ and } z \in R, \text{ for } i = 1, \dots, m, & \\
\qquad\qquad \text{and } j = 1, \dots, n. &
\end{cases}
$$

1.2.2 Equivalent Formulations

The general LP problem can be put under more compact and easy to handle forms. These forms are equivalent.

The Canonical Form :

$$
\begin{array}{rl}
\text{Min } \mathbf{c}^T\mathbf{x} & \mathbf{c} \in R^{n_c} \\
\text{s.t. } A\mathbf{x} \geq \mathbf{b} & A \in R^{m_c X n_c},\ \mathbf{b} \in R^{m_c} \\
\mathbf{x} \geq 0 & \mathbf{x} \in R^{n_c}.
\end{array}
$$

The Standard Form :

$$
\begin{array}{rl}
\text{Min } \mathbf{c}^T\mathbf{x} & \mathbf{c} \in R^{n_s} \\
\text{s.t. } A\mathbf{x} = \mathbf{b} & A \in R^{m_s X n_s},\ \mathbf{b} \in R^{m_s} \\
\mathbf{x} \geq 0 & \mathbf{x} \in R^{n_s}.
\end{array}
$$

The Mixed Form :

$$\begin{array}{lll} \text{Min } \mathbf{c}^T\mathbf{x} & & \mathbf{c} \in R^{n_d} \\ \text{s.t. } A_1\mathbf{x} & \geq \mathbf{b}_1 & A_1 \in R^{m_1 x n_d},\ \mathbf{b}_1 \in R^{m_1} \\ A_2\mathbf{x} & = \mathbf{b}_2 & A_2 \in R^{m_2 x n_d},\ \mathbf{b}_2 \in R^{m_2} \\ \mathbf{x} & \geq 0 & \mathbf{x} \in R^{n_d}. \end{array}$$

To transform the general LP problem to any of the three equivalent forms, elementary operations and relations are used, such as:

* $\min f(\mathbf{x}) = -\max[-f(\mathbf{x})]$,
* if x is arbitrary then $x = x^+ - x^-$, where $x^+ = \max[0, x]$ and $x^- = \max[0, -x]$,
* $\{\mathbf{a}^T\mathbf{x} = \beta,\ \mathbf{a} \in R^n, \mathbf{x} \in R^n, \beta \in R\} \equiv \{\mathbf{a}^T\mathbf{x} \geq \beta$ and $-\mathbf{a}^T\mathbf{x} \leq -\beta\}$,
* $\mathbf{a}^T\mathbf{x} \geq \beta$ may be replaced by $\mathbf{a}^T\mathbf{x} + x_s = \beta$, $x_s \geq 0$, x_s is called a slack variable.

1.2.3 Terminology and Geometric Concepts

A *program* or a feasible solution is a set of values of the variables which satisfies all the constraints of the problem. An *optimal solution* is a finite solution which optimizes the *objective function*. It is also called *optimal program* or *optimal plan*. A set $K = \{\mathbf{x} \in R^n \mid \mathbf{a}^T\mathbf{x} = \beta,\ \mathbf{a} \in R^n, \beta \in R\}$ is called a *hyperplane*. The set $\{\ \mathbf{x} \in R^n \mid \mathbf{a}^T\mathbf{x} = \beta,\ \mathbf{a} \in R^n, \beta \in R\}$ defines a *half-space*. The intersection of a finite set of half-spaces in R^n forms a *polyhedral set* , a *polyhedron* or a *polytope*. If H is a hyperplane, K a polyhedron and $H \cap K = E = \varnothing$, then H is called a *supporting hyperplane* to K. It should also be noted that $H \cap (\text{Int. } K) = \varnothing$. The set $E = H \cap K$ is called a *face*. If $\dim(K) = n$ and $\dim(E) = k$, then:

If $k = n-1$ then E is a *facet*,

if $k = 1$ then E is an *edge*,

if $k = 0$ then E is a *vertex*.

A *relevant constraint* is one that corresponds to a supporting hyperplane, or equivalently there is a feasible point **x** in the solution set for which the constraint is *tight*, i.e. satisfied as equality. A (min) LP problem is *unbounded* if in the feasible set the objective function $\mathbf{c}^T\mathbf{x}$ is not bounded from below. If the feasible set is empty, the problem is *infeasible*.

1.3 The Simplex Method

Without going into details, the simplex method can be described as follows.

Usually two phases are needed; in phase I the feasibility or otherwise of the problem is established and a vertex of the domain of the LP problem is found, if there is any. Phase II generates a monotone path in the feasible set, in accordance with the objective function of the problem. The path stops at a vertex when no improvement in the objective function value is possible, or else at an unbounded edge in which case the problem is unbounded.

The generation of the path corresponds to moving from an extreme point to an adjacent one at each iteration of the process. This is done by changing one of the vectors of the current basis with a non-basic vector which becomes basic after pivoting. Thus, algebraically, moving from a vertex to an adjacent one corresponds to changing the current basis with an adjacent one. In an m by n LP problem, where $m < n$, a vertex is determined by m linearly independent tight constraints.

There are many variants of the simplex, and they can differ a great deal. The way the feasible starting point is found and the criteria for choosing the entering variables into the basis can be totally different from one variant to another. However, they all generate a monotone path which ends at an optimal solution to the LP problem, if it admits any.

1.3.1 Performance of the Simplex Method

Given the worst case time bound of the simplex and the shear volume of work spent on LP since the 40's, LP was suspected by many to be in the NP-class. As worst case bounds are the easiest to derive and fail to reflect practical experience, many researchers tried to study the average case behaviour of the simplex. One of the early statements on the matter is due to Dantzig (1963):

"For an m-equation problem with m different variables in the final basic set, the number of iterations may run anywhere from m as a minimum to 2m and rarely to 3m. The number is usually less than 3m/2 when there are less than 50 equations and 200 variables (to judge from empirical observations.) Some believe that for a randomly chosen problem with fixed m, the number of iterations grows in proportion to n."

The quote reflects results on the performance of the simplex prior to 1963.

Systematic studies of the performance of the simplex method on real life and randomly generated LP problems have been carried out by many and results can be found in [McCall, 1982; Ho & Loute, 1980; Goldfarb & Reid, 1977; Benichou *et al.*, 1977]. Over more than 30 years the experience accumulated on the behaviour of simplex is vast. However in the scientific literature this experience is not fully documented as most practitioners in industry and other areas do not keep or publish the results of their experiences. The published results usually concern newly discovered variants of the simplex applied to standard test problems. This is probably due to the fact that since the 50's the finiteness of the algorithm was accepted by all.

In general, from the results of experiments with the simplex method it is concluded that simplex runs as a polynomial time algorithm. A rough bound on the number of steps one would expect to find a feasible solution to a linear program using Phase I of the simplex is conjectured to be αm, where m is the number of equations and α is 2 to 3. For n large relative to m, the value of α grows slowly as in $\exp(\alpha) < \log_2(2+n/m)$. These statements made by Dantzig in 1979 after the discovery of the ellipsoid algorithm, can be regarded as a summary on the average performance of the simplex, as was pointed out by Shamir (1987). For some problems, however, the number of vertices in the path of the simplex is unreasonably high, and not reflected in the above conclusions. These problems arise from Set Partitioning and periodical or time-staged tasks (Staircase Problems) [Ho & Loute, 1980; Fourer, 1982].

At a meeting in London, Dantzig (1987) reported that a rule for which no exponential counter example is known, has been pointed out by Zadeh. The rule is: "Choose as entering column one with $c_j < 0$ which has entered the basis the least number of times so far." It is also reported in [Papadimitriou & Steiglitz, 1982, p.192].

1.3.2 Experiments on Randomly Generated Problems

Despite a wealth of experimental results, theorists are cautious about drawing conclusions when the test problems are few and unrealistic and the hardware may play an important role in the results. A different experimental approach was considered: Controlled or Monte Carlo experimenting. Experiments were on randomly generated problem data with respect to some predetermined distribution. The results are compared to analytical results obtained under the same probabilistic assumptions. This approach may not bring significant conclusions as methods of random generation of test problems may influence the results. However, large classes of problems may be considered and "realistic" problems may be designed.

The first such experiments due to Kuhn and Quandt, mentioned in [Avis & Chvátal, 1978], were conducted on nine different pivoting rules for the simplex method. The results were not very conclusive due to the special form of the problems (constraints matrix always square and problems of small size). Indeed all the pivoting rules had almost the same performance and even the random choice rule performed well.

In [Avis & Chvátal, 1978] a similar approach has been taken. The performance of Bland's first rule [Bland, 1977] has been investigated and compared to other pivoting rules. Bland's rule performed worse than Dantzig's largest possible improvement rule. From these experiments it appears that the simplex is linear in min(m, n), where m and n are the dimensions of the problem.

Experimentation with real world problems and randomly generated problems does not seem to bridge the gap between the practical efficiency of the simplex and its exponential worst case bound. Probabilistic analysis is a natural approach as classes of problems with different distributions of data may be considered and average behaviour evaluated. This approach was taken by many, (see Shamir, 1987). Important results may be found in [Orden, 1980; and Smale, 1983]. The results, however, present some disparity due to wide range of assumptions and variants of the simplex. A unified theory of the probabilistic approach to the behaviour of the simplex is needed as some models are more general than others.

1.4 The Ellipsoid Algorithm

In the spring of 1979 LP received much attention with the discovery of a polynomial time algorithm. Khachyan (1979), a Russian mathematician, developed an algorithm which has a polynomial worst case bound in the length of the data of the LP problem. The algorithm was a continuation to the work of Shor, Iudin and Nemirowskiy in the early 70's on the larger class of convex optimization problems. Shor (1977) showed that, for a convex programming problem, if an *a priori* bound could be given for the distance from an initial point to an optimal solution, then a sequence $\{E_k\}$ of decreasing ellipsoids could be constructed, each containing an optimal solution. The decrease of the volume of each E_k depends only on the dimension n of the solution space. Khachyan adapted this approach to the solution of systems of linear inequalities. He used the length of the original data of the problem to derive an *a priori* bound for the distance of a solution from the origin. He perturbed the right-hand side (RHS) of the linear inequalities to obtain a lower bound on the volume of the feasible region. These two bounds combined with the rate at which the ellipsoids were shrinking was enough to obtain a polynomial bound for the number of iterations necessary to find a solution, if the system has any.

1.4.1 Constructing Khachyan's Algorithm

Solving LP problems is no more difficult than solving sets of linear inequalities [Chvátal, 1983; Gács & Lovász, 1979, 1981; Khachyan, 1979]. The set of linear inequalities may be divided into two subsets: The weak linear inequalities and the strict linear inequalities. It is shown in [Papadimitriou & Steiglitz, 1982; Chvátal, 1983; Apsvall & Stone, 1980] that a set of linear strict inequalities can be constructed by perturbing the RHS of weak inequalities, and any solution to one system is also solution to the other.

Hence to solve LP in polynomial time it is enough to solve the equivalent system of linear strict inequalities in polynomial time. The ellipsoid algorithm was designed to meet these requirements. The method is basically similar to the binary search as was mentioned earlier. To make this process work when looking for the solution to a set of strict linear inequalities, lower and upper bounds on the set of solutions are needed. The bounds are set up as follows.

The set of solutions to a system of strict linear inequalities is a polyhedron K which can be unbounded or empty. Here, bounded and nonempty polyhedra are considered. Khachyan derived an upper bound to the set of solutions by considering the smallest ball (ellipsoid) containing it. The radius of the ball is defined by the length of the data of the system of linear inequalities. It has been shown [Edmonds, 1967], through the use of Cramer's rule, that the number of digits in any coordinate of a solution to a system of linear inequalities cannot exceed the total number of digits in the m(n+1) integers a_{ij} and b_i of the system. Consider the system

$$\sum_{j=1}^{n} a_{ij}x_j < b_i, \quad i = 1, \ldots, m. \qquad (1.1)$$

The length of its data is $L = mn + \lceil \mathrm{Log}\,|P| \rceil$, where P is the product of all entries of A and **b** different from zero, i.e. $P = \prod_{i=1}^{m} \left[\prod_{j=1}^{n} a_{ij} \right] \prod_{i=1}^{m} b_i$.

From what was said earlier, it can be written that: $\forall x_j \in K$, $-2^L \le x_j \le 2^L$. And geometrically, the polyhedron K can be enclosed in the ball $E_0 = \{ \mathbf{x} : \|\mathbf{x}\| \le 2^L \}$.

As a solution to a system of strict inequalities may not exist, it is crucial to know when the search has to be stopped. The lower bound in the interval of search is a minimum volume ellipsoid. An important lemma [Khachyan, 1979; Gács & Lovász, 1979; Apsvall & Stone, 1980] states that: If (1.1) has a solution, then the volume of its solution space inside the sphere $\|\mathbf{x}\|_2 \le 2^L$ is at least $2^{-(n+1)L}$.

At each step in the process of the binary search, one part of the solution space is discarded. To carry on the search, the remaining part of the previous ellipsoid which contains a solution, if there is one, is enclosed in a smaller ellipsoid.

Suppose that the ellipsoid $E = \{\mathbf{x} : (\mathbf{x} - \mathbf{x}^{(k)})^T B^{(k)} (\mathbf{x} - \mathbf{x}^{(k)}) \leq 1\}$ is cut by the hyperplane supporting the half space defined by one of the constraints in (1.1), say $\mathbf{a}_i^T \mathbf{x} < b_i$, for some i. If the centre $\mathbf{x}^{(k)}$ of E violates this constraint, i.e. $\mathbf{a}_i^T \mathbf{x}^{(k)} \geq b_i$, then define the ellipsoid $E' = \{\mathbf{x} : (\mathbf{x} - \mathbf{x}^{(k+1)})^T B^{(k+1)} (\mathbf{x} - \mathbf{x}^{(k+1)}) \leq 1\}$, where

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \frac{1}{n+1} \frac{B^{(k)}\mathbf{a}}{\sqrt{\mathbf{a}^T B^{(k)} \mathbf{a}}},$$

$$B^{(k+1)} = \frac{n^2}{n^2 - 1}\left(B^{(k)} - \frac{2}{n+1} \frac{\left(B^{(k)}\mathbf{a}\right)\left(B^{(k)}\mathbf{a}\right)^T}{\mathbf{a}^T B^{(k)} \mathbf{a}}\right)$$

It can be shown that E' contains the set of points defined by E and $\mathbf{a}_i^T \mathbf{x} < b_i$, for some i, and also has volume less than that of E, [Khachyan, 1979; Gács & Lovász, 1979; Apsvall & Stone, 1980]. A version of the ellipsoid algorithm based on this construction may be described as follows.

Algorithm 2.1

```
begin
    Initialization: $k = 0$, $L = mn + \lceil \mathrm{Log}\,|P| \rceil$, $\mathbf{x}^{(k)} = 0$, $B^{(k)} = n^2 2^{2L} I$.
    while ($\exists i \mid a_i^T \mathbf{x}^{(k)} \geq \beta_i$, $i \in \{1, ..., m\}$) and ($k \leq 16n(n+1)L$) do

        $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \frac{1}{n+1} \frac{B^{(k)}\mathbf{a}}{\sqrt{\mathbf{a}^T B^{(k)} \mathbf{a}}}$,
        $B^{(k+1)} = \frac{n^2}{n^2 - 1}\left(B^{(k)} - \frac{2}{n+1} \frac{\left(B^{(k)}\mathbf{a}\right)\left(B^{(k)}\mathbf{a}\right)^T}{\mathbf{a}^T B^{(k)} \mathbf{a}}\right)$

    endwhile
    if ($k > 16n(n+1)L$) then set of strict inequalities incompatible endif
end
```

1.4.2 Improvements to the Basic Algorithm

The volumes of ellipsoids generated by the algorithm are central to its convergence. It is therefore natural to think that if the size of the enclosing ellipsoid after a cut can be reduced then convergence is consequently improved. Along this line the following suggestions were made.

In the basic ellipsoid algorithm the cut which passes through the centre of E_k is used. The half ellipsoid $\{ \mathbf{x} \in E_k \mid \mathbf{a}^T\mathbf{x} \leq \mathbf{a}^T\mathbf{x}_k \}$ is retained in E_{k+1}. However, as E_{k+1} is only required to contain the smallest portion of E_k, i.e. $\{ \mathbf{x} \in E_k \mid \mathbf{a}^T\mathbf{x} \leq \beta \}$, then it is possible to obtain an ellipsoid of smaller volume using the deep cut $\mathbf{a}^T\mathbf{x} \leq \beta$. Shor and Gershovitch, cited by Bland *et al.* (1981), first thought of the deep cuts based of the idea that, if α is the distance of $\mathbf{x}_k$ to the half-space $\{\mathbf{x} \in R^n \mid \mathbf{a}^T\mathbf{x} \leq \beta\}$, then by computing α for each inequality and choosing one corresponding to the largest α guarantees the deepest cut. The distance α appears in the step, dilation and expansion parameters which characterize the ellipsoid method with deep cuts.

Deeper cuts than those obtained with a single constraint can be obtained by combining inequalities in (1.1). They are termed "surrogate cuts" by Goldfarb and Todd (1980). It was noticed that points satisfying (1.1) are not discarded (cut off) by inequalities of the form $\mathbf{u}^T A^T \mathbf{x} \leq \mathbf{u}^T\mathbf{b}$, when $\mathbf{u} \geq 0$. Hence, by considering a subset of linearly independent inequalities of (1.1), it is possible to compute $\mathbf{u}$ corresponding to the deepest cut. However, it is too expensive to do so as a quadratic programming problem arises in the process. In [Goldfarb & Todd, 1980; Bland *et al.*, 1981] it is recommended to use surrogate cuts obtained with two constraints at most.

Shor and Gershovitch, again, first thought of using the parallelism of constraints that may arise in the problem to solve. Two parallel constraints occurring in the problem may be used simultaneously to construct the new ellipsoid. This ellipsoid will be flat in the direction of the perpendicular to the constraints, as it is only required to contain the slice encompassed between them. Suppose that $\mathbf{a}^T\mathbf{x} \leq \beta$ and $-\mathbf{a}^T\mathbf{x} \leq -\lambda$, then $\{\mathbf{x} \in E_k \mid \lambda \leq$

$\mathbf{a}^T\mathbf{x} \leq \beta\} \subset E_{k+1}$. The step, dilation and expansion parameters corresponding to parallel cuts can be found in [Todd, 1982; König & Pallaschke, 1981].

1.4.3 Performance of the Ellipsoid Algorithm

From the literature considered, the general consensus is that the ellipsoid method is computationally inferior to simplex. Because of the large number of ellipsoids to be evaluated, together with the required high precision, the method seems too expensive to apply to nontrivial problems. If calculations are carried out with low precision it can practically solve linear inequality systems in up to 15 variables [Schrijver, 1986].

The few numerical results we encountered support this conclusion. An implementation of the algorithm by Halfin (1983) solved randomly generated linear programs with up to 50 variables in more than 24,000 iterations. König & Pallaschke (1981) reported on solving LP problems in 25 variables and 100 constraints in about 1,500 iterations. A detailed account of experiments with an APL code of the Khachyan algorithm is also found in [Bisshopp, 1981]. The experiments were on systems of linear strict inequalities with integer entries ranging from 5 variables and 10 inequalities to 20 variables and 40 inequalities. Some of the problems required up to 14,119 iterations.

1.5 Central Splitting and Simplicial Algorithms of Yamnitsky and Levin

As for the ellipsoid method, the idea behind the Central Splitting Algorithm (CSA) [Levin 1965], is also reminiscent of the binary search. CSA was aimed at finding an approximate solution within $\varepsilon \in R$ of the exact solution to the problem of minimizing a convex function f of n variables on a convex polytope K in the Euclidean space. It is assumed that f satisfies a Lipschitz condition, i.e. $| f(\mathbf{x}^{(1)}) - f(\mathbf{x}^{(2)}) | \leq c\rho(\mathbf{x}^{(1)}, \mathbf{x}^{(2)})$, for $\mathbf{x}^{(1)}$ and $\mathbf{x}^{(2)} \in K$ where $\rho(\mathbf{x}^{(1)}, \mathbf{x}^{(2)})$ is the distance between points $\mathbf{x}^{(1)}$ and $\mathbf{x}^{(2)}$, and c the Lipschitz constant which can be *a priori* determined. The "principal operation" of the process consists of the following: (a) Find a point $\mathbf{x}^*$ interior to K, (b) split K into two parts through $\mathbf{x}^*$, by a point, a line or a hyperplane depending on the dimension of K, (c) discard one part according to the relation $(\mathbf{x}_{\min} - \mathbf{x}^*, \mathrm{grad}[f(\mathbf{x}^*)]) < 0$.

In [Levin, 1965] the process is explicitly given for the case n=2. The polytope K being a polygon, the principal operation consists of finding $\mathbf{x}^*$ and cutting through it with a straight line. If $\mathrm{grad}[f(\mathbf{x}^*)] = 0$ then $\mathbf{x}^* = \mathbf{x}_{\min}$. Otherwise the points which do not satisfy the inequality $(\mathbf{x}_{\min} - \mathbf{x}^*, \mathrm{grad}[f(\mathbf{x}^*)]) < 0$ are discarded. The choice of point x^* is crucial to the convergence of the algorithm. The part of K, K_1, K_2 discarded at each iteration should be large enough to allow rapid contraction of the feasible region. In [Levin, 1965] it was recommended that the centre of gravity of K be chosen as $\mathbf{x}^*$. This is based on the fact that cutting a convex polygon of area σ, with a line passing through its centre of gravity results in two convex polygons each one having an area not less than $(4/9)\sigma$. It can be said that at iteration k, area of K_k is at most $(5/9)^k\sigma$, which suggests a geometric convergence of the process. It should be noticed that for "elongated" areas, the speed of convergence may not be geometric. However, approximating them with intervals, which is equivalent to reducing the dimension of the feasible region, can overcome the problem. The number of operations can be shown to grow only geometrically (note: geometric refers to the speed of convergence of geometric series.)

For $n \geq 3$ a geometric bound can be drawn on the number of principal operations for similar reasons as earlier. However, the gometric bound does not apply to the total work involved in a single step of the algorithm. In addition to the principal operation supplementary work is needed to determine and store information about the retained half of the feasible region at each iteration (vertices, faces and centre of gravity). The subsidiary work may grow exponentially with the number of iterations. This results from the fact that a random polyhedron may have an exponential number of vertices [Chvátal, 1983].

To alleviate the difficulties of the CSA, Levin suggested the following: If V is the volume of a polyhedron K of dimension n, then there is $\gamma_n \in R$ and an n-dimensional simplex S containing K, with volume not exceeding $\gamma_n V$. S is then used in subsequent splittings. At this point the similarity with the ellipsoid method is clear. The embedding operation may not be required at every step. A geometric decrease in the volume of subsequent simplices is guaranteed for similar reasons as earlier. In [Yamnitsky, 1982] it has been shown that the modified CSA runs in polynomial time in the length of the input data. The novelty was a procedure for enclosing a half simplex $1/2S_k$ inside a simplex S_{k+1} whose volume is less than $e^{-1/2(n+1)^2}$ the volume of S_k.

1.6 Research Needs and Objectives

Much of the work done in LP for over thirty years has been concerned with improving existing simplex variants and developing new ones. It is only in recent years that polynomial time algorithms became a topic of wide interest. This interest stems from LP being widely used on its own and as a building block in many optimization problems, and also from the general agreement that well solved problems are those for which polynomial time algorithms were found. Any improvement in solving LP will have a positive impact on related problems.

The discovery of the ellipsoid [Khachyan, 1979], the simplicial [Levin & Yamnitsky, 1982] and the projective algorithms ended the important debate over the complexity of LP, at least in the integer model of computation. However on the practical side, the gap remained. It is generally agreed, after investigation, that the ellipsoid and the simplicial algorithms, in practical terms, are inferior to simplex on most real life LP problems. Karmarkar's algorithm on the other hand is relatively new and has not been fully investigated. The algorithm is promising and may be a good alternative to the simplex method. However, much work remains to be done before definite conclusions can be drawn. The lack of experimental results in the original Karmarkar's paper and the conclusion of analysis of the algorithm by Charnes *et al.* (1984), Strang (1985) and others that the method is inherently slow, stressed the need for further investigation of the algorithm and its performance.

The overall objective of the present research has been to investigate some aspects of Karmarkar's algorithm such as the preponderance of least squares techniques in its efficient implementation, the optimum choice of the step size to take along the search direction, the retrieval of dual variables during the course of the algorithm and the exploitation of favourable structure of LP problems. Sparsity exploitation is undoubtedly the important issue in any efficient implementation of the algorithm. Advanced least squares techniques were, therefore, called upon to cut down the work needed in an iteration of the algorithm. The nested dissection ordering algorithm was used in conjunction with Cholesky method for least squares and also Givens rotations. Large LP problems were solved with this approach in realistic times.

Structured LP problems constitute an important class to which much work has been devoted in the frame of the simplex method leading to the design of elegant decomposition algorithms such as the Dantzig-Wolf algorithm, Rosen's partitioning algorithm and others. However, these algorithms never outclassed the standard simplex method. We thus considered the applicability of Karmarkar's algorithm in conjunction with some classical decomposition principles and also specialized it for structured LP problems. This led to a partitioning Karmarkar algorithm which performed better than the straight application of a variation on the dual Karmarkar algorithm of Ye and Kojima (1987).

An attempt to study postoptimality analysis in the frame of Karmarkar's algorithm was also made, encouraged by the availability of dual variables.

The polynomial time algorithms discussed in this chapter have a common feature which is a centring scheme, i.e. the algorithms strive to start a new iteration from the "centre" of the feasible region. A centre point, is defined as one which is sufficiently distant from the boundaries of the polytope [Sonnevend, 1985; Freund, 1988]. In this sense, under certain assumptions, Chebyshev points returned as solutions to the Chebyshev minimax problem are "centres" of the simplex containing the feasible region defined by the linear inequalities. By converting the LP problem into a Chebyshev one, we attempted to build an algorithm that generates a sequence of points of minimum deviation converging to the optimum solution of the original LP problem.

Chapter 2

The Projective Algorithm of Karmarkar: A Survey

2.1 Introduction

The algorithm of Karmarkar (1984a, 1984b) came as a result of the search for a method which has polynomial complexity like the ellipsoid and the simplicial methods but is practical like the simplex. It is related to classical interior point methods, but presents original features such as the use of projective geometry and a logarithmic potential function to measure convergence.

Going in the direction of the gradient is the first thing one thinks of when interior point methods are considered for linear programming. However, this will yield a substantial improvement in the objective function only if the current feasible point is at the centre of the polytope, i.e. sufficiently distant from all its boundaries. Consequently, for an iterative process to work with these ideas, it must alternate between centring the feasible point and taking a step in the gradient direction.

Classical interior methods of the Brown-Koopmans type have difficulties near the boundaries, precisely because they lack the centring step. The difficulties, usually, result in the loss of feasibility and slow convergence. On the other hand, Karmarkar's algorithm successfully combines the two steps and thus avoids the difficulties of the classical methods, as will be seen in the convergence analysis of the algorithm.

The centring process is performed by rescaling the feasible region at each iteration using a projective transformation. This results in approximating the optimization problem with a minimization over a sphere of known centre and radius. The minimization over a sphere is then solved by taking a step to its boundary along a projected gradient direction. The rescaling process combined with the step along the negative projected gradient is then repeated until optimality is achieved or the problem is recognized to be unbounded or infeasible.

2.2 The Projective Algorithm of Karmarkar

Consider the linear programming problem in standard form

$$\begin{aligned} \text{SLP}_\mathbf{x}: \quad & \text{Min } \mathbf{c}^T\mathbf{x} \\ & \text{s.t. } A\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \geq 0, \end{aligned}$$

where R^n is the n-dimensional Euclidean space; $\mathbf{x}, \mathbf{c} \in R^n$, $\mathbf{b} \in R^m$ and $A \in R^{m\times n}$. The original Karmarkar algorithm requires that the LP problem is expressed in a special form called the canonical form, which is

$$\begin{aligned} \text{PC}: \quad & \text{Min } \mathbf{c}^T\mathbf{x} \\ & \text{s.t. } A\mathbf{x} = 0 \\ & \mathbf{e}^T\mathbf{x} = 1 \\ & \mathbf{x} \geq 0 \end{aligned}$$

where $\mathbf{e}^T = (1, 1, ..., 1)$.

In addition, it is required that the minimum objective value is 0, and the value of the objective at any feasible and nonoptimal point is strictly positive. The question of converting $\text{SLP}_\mathbf{x}$ into PC will be treated in detail later.

The centring scheme of Karmarkar is based on a projective transformation defined by

$$T_x(\mathbf{x}) = \frac{D^{-1}\mathbf{x}}{\mathbf{e}^T D^{-1}\mathbf{x}} = \mathbf{x}',$$

and its inverse

$$T_x^{-1}(\mathbf{x}') = \frac{D\mathbf{x}'}{\mathbf{e}^T D\mathbf{x}'} = \mathbf{x},$$

where $D = \text{diag}(\mathbf{x}^{(k)})$, $\mathbf{x}^{(k)}$ being a point in the space of PC.

Transforming PC using T_x^{-1}, results in a nonlinear (fractional) programming problem with the objective function $\mathbf{c}^T D\mathbf{x}'/\mathbf{e}^T D\mathbf{x}'$. However, $\mathbf{e}^T D\mathbf{x}'$ being positive in the transformed feasible region and given that $\mathbf{c}^T\mathbf{x}$ has minimum zero, $\mathbf{c}^T D\mathbf{x}'/\mathbf{e}^T D\mathbf{x}'$ has also minimum zero. Thus, it can be approximated with $\mathbf{c}^T D\mathbf{x}'$. It follows that the transformed problem to be considered is

$$P_{x'}: \quad \text{Min} \quad \mathbf{c}^T D\mathbf{x}'$$
$$\text{s.t.} \quad AD\mathbf{x}' = 0$$
$$\mathbf{e}^T\mathbf{x}' = 1, \mathbf{x}' \geq 0,$$

which is of the required form PC.

This problem is an optimization over the intersection of the simplex $\Sigma = \{\mathbf{x}' \in R^{n+1}: \mathbf{x}' \geq 0, \ \Sigma x'_j = 1\}$, with the linear subspace $\Pi = \{\mathbf{x}' \in R^{n+1}: \mathbf{x}' \geq 0, A\mathbf{x}' = 0\}$. The centre of the simplex $\mathbf{x}_0'^T = (1/(n+1), 1/(n+1), ...)$, being a feasible point, a reduction in the objective function is likely to be achieved along the opposite direction of the projected gradient $\mathbf{p}$, starting from $\mathbf{x}_0'$. However, to insure feasibility after the move, Karmarkar considered the minimization over the largest inscribed sphere S_r in Σ, as an approximation to the minimization over the simplex Σ. This insures feasibility of the resulting point. The problem is written

$$P_{x'_S}: \quad \text{Min} \quad \mathbf{c}^T D\mathbf{x}'$$
$$\text{s.t.} \quad AD\mathbf{x}' = 0$$
$$\mathbf{e}^T\mathbf{x}' = 1$$
$$\| \mathbf{x}'-(e/n)\| \leq \alpha r$$
$$\mathbf{x}' \geq 0,$$

where $r = 1 / \sqrt{(n(n-1))}$ is the radius of S_r.

From the geometric point of view, there are 3 spaces involved: the space of the original problem, the space of the homogeneous form of the problem and its image resulting from the projective transformation. Call the last two spaces respectively x-space and x'-space. A sketch of the optimization process is as follows (see Fig 2.1).

Let $\mathbf{x}^{(k)}$ be a point in x-space. Applying the projective transformation T_x to $\mathbf{x}^{(k)}$ results in the centre of S_r in x'-space. A new point in x'-space would be $\mathbf{x}'^{(k)}$ at the boundary of S_r. This point is transformed back into the x-space by the inverse projective transformation T_x^{-1}, resulting in a point $\mathbf{x}^{(k+1)}$. It is easy to see that an improvement in the objective function of $P_{x'_S}$ is achieved in the direction of the projected gradient. The reduction in the objective function of PC is harder to see, when we know that the set of linear functions is not invariant under projective transformations. In this respect, Karmarkar introduced the logarithmic potential function $F(\mathbf{x}) = n\log \mathbf{c}^T\mathbf{x} - \Sigma_j \log(x_j)$, which is invariant under projective transformations. To see that, we write the potential function associated to the objective function in the transformed problem

$$F(\mathbf{x}') = \sum_j \log (\mathbf{c}^T D\mathbf{x}'/x'_j) \tag{2.2.1}$$

and in the x-space after applying inverse transformation to $\mathbf{x}'$

$$F(T_x^{-1}(\mathbf{x}')) = \sum_j \log (\mathbf{c}^T D\mathbf{x}'/x'_j) - \sum_j \log x_j. \tag{2.2.2}$$

Expressions (2.2.1) and (2.2.2) are similar except for a constant $-\sum_j \log x_j$.

Karmarkar proved that a positive constant reduction is achieved in the potential function associated with $\mathbf{c}^T D\mathbf{x}$, when moving from the centre of S_r to its boundary. From (2.2.1) and (2.2.2), this reduction corresponds to some reduction in the image of the potential function in x-space. It follows that

$$F(\mathbf{x}^{(k+1)}) \leq F(\mathbf{x}^{(k)}) - \delta, \tag{2.2.3}$$

where δ is a positive constant.

Theorem 2.1 (Karmarkar (1984b), Theorem 1):

An algorithm to solve PC that generates a sequence of points $\{\mathbf{x}^{(k)}\}$ satisfying (2.2.3) will find a feasible point $\mathbf{x}$ such that $\mathbf{c}^T\mathbf{x} / \mathbf{c}^T\mathbf{x}^{(0)} \leq 2^{-q}$ in $O(n(q+\log n))$ steps.

Proof:

From (2.2.3) we have $F(\mathbf{x}^{(k)}) \leq F(\mathbf{x}^{(0)}) - k\delta$, i.e.

$$n\log \mathbf{c}^T\mathbf{x}^{(k)} - \sum_j \log x^{(k)}_j \leq n\log \mathbf{c}^T\mathbf{x}^{(0)} - \sum_j \log x^{(0)}_j - k\delta,$$

or equivalently

$$n\log \mathbf{c}^T\mathbf{x}^{(k)} - n\log \mathbf{c}^T\mathbf{x}^{(0)} \leq \sum_j \log x^{(k)}_j - \sum_j \log x^{(0)}_j - k\delta.$$

As $x^{(k)}_j \leq 1$, from $\mathbf{e}^T\mathbf{x} = 1$, and $\mathbf{x}^{(0)} = (1/n, 1/n, ..., 1/n)$ we can write

$$n\log \mathbf{c}^T\mathbf{x}^{(k)} - n\log \mathbf{c}^T\mathbf{x}^{(0)} \leq n\log n - k\delta,$$

and

$$\log \mathbf{c}^T\mathbf{x}^{(k)} - \log \mathbf{c}^T\mathbf{x}^{(0)} \leq \log n - k\delta/n.$$

Thus for $k > n/\delta(q + \log n)$, $\mathbf{c}^T\mathbf{x} / \mathbf{c}^T\mathbf{x}_0 \leq 2^{-q}$ Q.E.D.

Algorithm 2.1

Karmarkar algorithm generates a sequence of points $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ..., \mathbf{x}^{(k)}, ...$ with the assumption that $\mathbf{x}^{(k)} \geq 0$, $k = 1, ...$ Assume also that an interior starting point $\mathbf{x}^{(0)}$ is available, and an arbitrarily small value ε is chosen, then the algorithm can be described in the following steps.

```
0- $k = 0$
1- Set $D = \mathrm{diag}(\mathbf{x}^{(k)})$ and $B = \begin{pmatrix} AD \\ \mathbf{e}^T \end{pmatrix}$
2- Project vector $D\mathbf{c}$ onto the null space of B to find $\mathbf{p} = HD\mathbf{c}$ where the projection
   matrix $H = I - B^T(BB^T)^{-1}B$
3- Normalize $\mathbf{p}$ and scale it by the radius $r = 1 / \sqrt{(n(n-1))}$ of $S_r$ to find the direction
   vector $\mathbf{p}' = r\frac{\mathbf{p}}{\|\mathbf{p}\|}$.
4- Compute a new feasible point in x'-space by taking a step of length $\alpha$ along $\mathbf{p}'$,
   starting from the centre $\mathbf{e}/n$ of $S_r$
       $\mathbf{x}' = \mathbf{e}/n - \alpha\mathbf{p}', \alpha \in (0, 1)$
5- Apply inverse transformation to x' to find a new point in x-space
       $\mathbf{x}^{(k+1)} = \frac{D\mathbf{x}'}{\mathbf{e}^T D\mathbf{x}'}$
6- Check for optimality
   if $\mathbf{c}^T\mathbf{x}^{(k+1)} / \mathbf{c}^T\mathbf{x}^{(0)} \leq \varepsilon$ then stop (optimum obtained)
   else $k = k + 1$, go to 1- endif
```

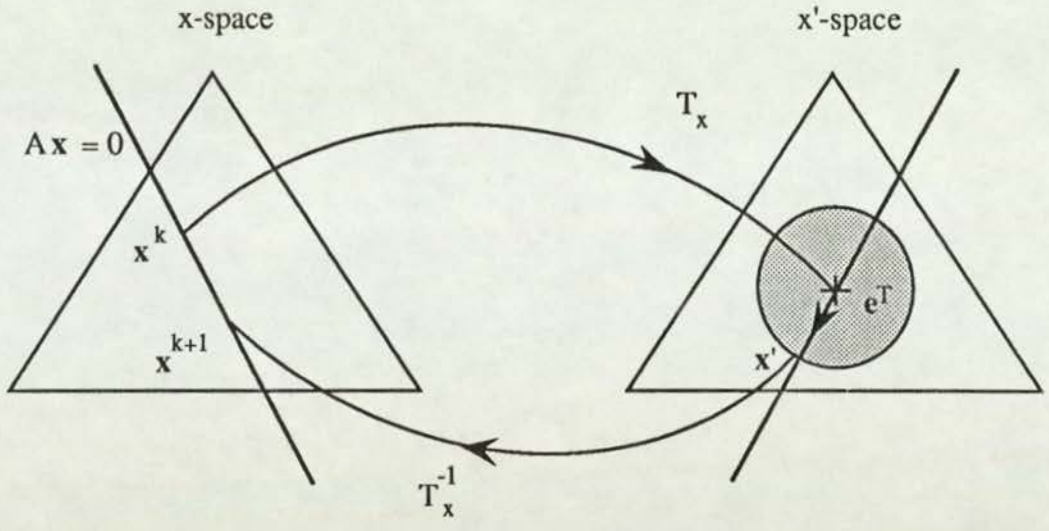


Fig 2.1 An Iteration of the Algorithm

Illustration:

Consider the problem

$$\begin{aligned} \text{Min} \quad & z = 2x_1 - x_2 \\ \text{s.t.} \quad & 3x_1 + x_2 = 4 \\ & x_1 \geq 0,\ x_2 \geq 0, \end{aligned}$$

whose optimum objective value is $z^* = -4$.

$$\begin{aligned} \text{Min} \quad & 2x_1 - x_2 - z^* x_3 \\ \text{s.t.} \quad & 3x_1 + x_2 - 4x_3 = 0 : \Omega \\ & x_1 + x_2 + x_3 = 1 : \Sigma \\ & x_1 \geq 0,\ x_2 \geq 0,\ x_3 \geq 0. \end{aligned}$$

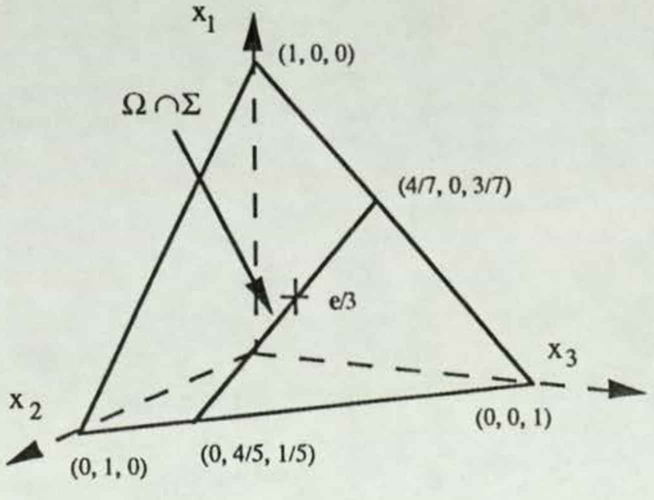


Fig 2.2 Feasible Polytope $\Omega \cap \Sigma$

The problem being under canonical form, the algorithm may be applied if a feasible interior point is available to start with. For this purpose, point $x_0 = e/3 = (1/3, 1/3, 1/3)^T$

is interior feasible, as it belongs to the line segment between points (0, 4/5, 1/5) and (4/7, 0, 3/7) in $\Omega \cap \Sigma$ which is the feasible region. It is also the centre of the simplex Σ as depicted in Fig 2.2.

The first iteration of the algorithm requires rescaling the feasible region, using the projective transformation $T(\mathbf{x}) = \mathbf{x}' = D^{-1}\mathbf{x} / \mathbf{e}^T D^{-1}\mathbf{x}$, and its inverse $T^{-1}(\mathbf{x}') = \mathbf{x} = D\mathbf{x}'/\mathbf{e}^T D\mathbf{x}'$, where $D = \text{diag}(\mathbf{x}_0) = \text{diag}(1/3, 1/3, 1/3)$. However, $\mathbf{x}_0$ being already at the centre of the simplex, the transformation is equivalent to an identity, which leaves the region as in Fig 2.2. The objective function, however, has changed. The transformed problem is

$$
\begin{aligned}
\text{Min} \quad & 2/3x'_1 - 1/3x'_2 - (z^*/3)x'_3 \\
\text{s.t.} \quad & x'_1 + 1/3x'_2 - 4/3x'_3 = 0 : \Omega \\
& x'_1 + x'_2 + x'_3 = 1 \qquad : \Sigma \\
& x'_1 \geq 0,\ x'_2 \geq 0,\ x'_3 \geq 0.
\end{aligned}
$$

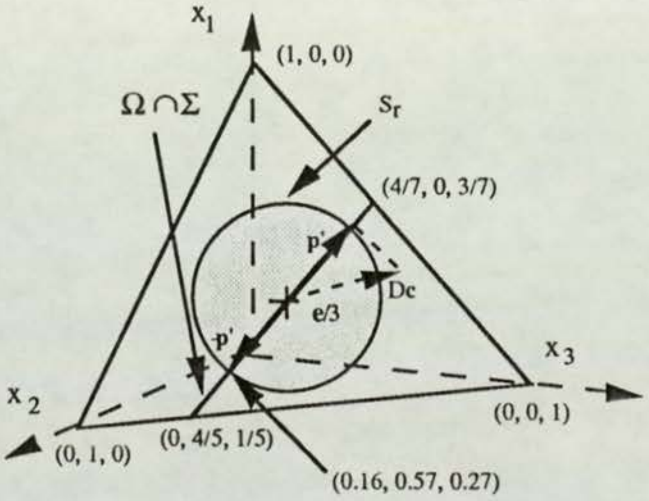


Fig 2.3 A Step Along the Projected Gradient

In step 2 of Karmarkar's algorithm the steepest descent direction in the transformed feasible region is found. The direction **-p'** is found by projecting the gradient Dc onto the polytope $\Omega \cap \Sigma$, i.e. multiplying the projection matrix H with vector Dc, and considering the negative of this vector (see Fig 2.3). It is along the negative gradient that the objective function decreases most rapidly.

In step 4 a move in the direction **-p'** is made. However, to guarantee feasibilty after the move, a sphere S_r of radius $\alpha[n(n-1)]^{-1/2}$ centered at **e**/3 is inscribed inside the triangle Σ of above figures, and the minimization is over $\Omega \cap \Sigma \cap S_r$ which is a sphere but of lower dimension. All the points of this lower dimension sphere are feasible. In Fig 2.3 it is the segment which is delimited by S_r in $\Omega \cap \Sigma$. Notice that $r = [n(n-1)]^{-1/2}$ is the radius of the largest sphere that can be inscribed in Σ. A fraction α of r is only taken to avoid infeasibility, with $0 < \alpha < 1$. The move is then of length αr, with $\alpha = 0.9$ and results in point (0.16, 0.57, 0.27). The point is not optimal as it does not reduce the objective function to zero. Note that if the move was long enough, we would have reached the optimum solution, which is the end point (0, 4/5, 1/5) of segment $\Omega \cap \Sigma$. However, this is so because the feasible region is a segment and the solution is one of its two vertices. In higher dimensions it would not be so easy to identify the solution.

Having obtained a new point, we can proceed with the next iteration. The problem is first transformed into

$$\begin{aligned} \text{Min } & 0.32x'_1 - 0.57x'_2 + 1.08x'_3 \\ \text{s.t. } & 0.48x'_1 + 0.57x'_2 - 1.08x'_3 = 0 : \Omega \\ & x'_1 + x'_2 + x'_3 = 1 : \Sigma \\ & x'_1 \geq 0, x'_2 \geq 0, x'_3 \geq 0. \end{aligned}$$

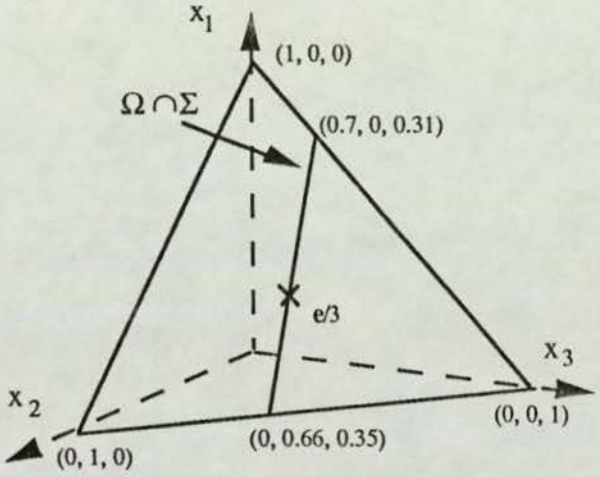


Fig 2.4 Rescaling of the Feasible Region

Again, the search direction is computed by multiplying the projection matrix H and the gradient of the objective function in above problem. The optimization process over $\Omega \cap \Sigma \cap S_r$ produces point (0.07, 0.59, 0.35) as depicted in Fig 2.5. When transformed back to the space of the canonical form, point (0.11, 3.56, 1.0) is obtained which is close to the optimal solution $x^* = (0.0, 4.0, 1.0)$. The corresponding objective function is $z = 0.67$, which is still much larger than zero. One more iteration is necessary to get a good approximation to the optimum solution.

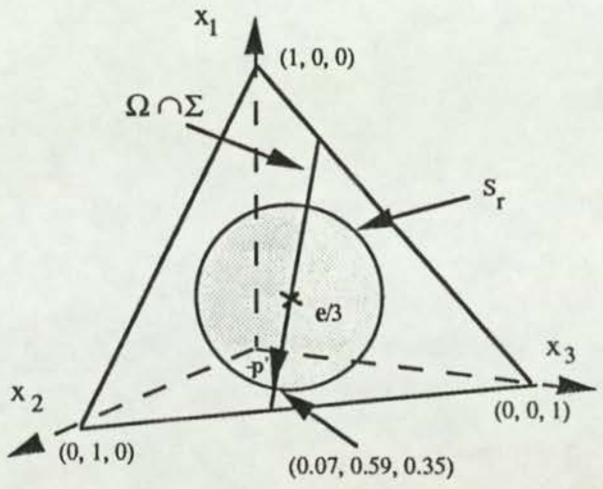


Fig 2.5 Result of Iteration 2

2.2.1 Algorithm Complexity

As was seen earlier the potential function is central to the convergence of the algorithm. At the end of iteration k a constant reduction must occur in the potential function. However, a constant reduction in the objective function is produced after O(nq) steps, (Theorem 2.1), for some natural number q.

It is known [Edmonds, 1965; Chvátal, 1983], that if the data of the linear programming problem are rational numbers then there exists $L \in N$, the set of natural numbers, such that all nonzero coordinates of the vertices of the feasible region belong to $[2^{-L}, 2^{+L}]$ and any numerical value of the problem can be described with L digits, i.e. a binary precision of L digits is sufficient. If we replace q with L then the algorithm requires O(nL) iterations to find a positive feasible point such that $\mathbf{c}^T\mathbf{x} < 2^{-L}$.

Consider the number of arithmetic operations per iteration: The computation of **p**', **x**' and $\mathbf{x}^{(k+1)}$ requires O(n) operations. Most of the work, however, is needed in the computation of **p** which requires $O(n^3)$ operations. This is because computing **p** is in general equivalent to solving a set of linear equations by Gaussian elimination. As each operation may take O(L) time, then an iteration of the algorithm has time bound of $O(n^3L)$. Thus, the whole algorithm takes, in the worst case, $O(n^4L^2)$ time to solve a LPP on the set of rationals Q.

In [Karmarkar, 1984b] it was shown that the time bound of the algorithm may be reduced using a rank-one updating of the diagonal matrix D at each iteration. $\sqrt{n}$ operations can be gained if one updates only those entries of D that have changed from iteration k to iteration k+1. The modified algorithm has time bound $O(n^{3.5}L^2)$ compared to $O(n^6L^2)$ for the ellipsoid algorithm.

Limited experience with medium scale problems [Meggido, 1986] shows that the improvement in the time bound of the modified algorithm does not appear in practice.

2.2.2 Transforming The Standard Form into Canonical Form

The applicability of Algorithm 2.1 is restricted by the assumptions that the LP problem is in canonical form and an interior feasible point to start with is available. As the conversion of the GLP into standard form (SLP_x) has already been dealt with in Chapter 1, the present section is concerned with converting LP problems in standard form into the required form and finding an interior feasible point. Three methods were suggested respectively by Karmarkar (1984a), Tomlin (1985) and Lustig(1985).

Method 1:

In Karmarkar (1984a) it was shown that a projective transformation can be used to get the required canonical form as follows.

Define the projective transformation T from R^n to R^{n+1} by

$$T(x_i) = x'_i = \frac{x_i x'_i}{1 + \sum_{j=1}^{n} \frac{x_j}{x_i}}$$

and $\qquad x'_{n+1} = 1 - \sum_j x'_i.$

T transforms $x \in R^n$ into the centre of the unit simplex in R^{n+1}. Its inverse is

$$T^{-1}(x'_i) = x_i = \frac{x_i x'_i}{x'_{n+1}}, \quad i = 1, \ldots, n,$$

defined on the unit simplex in R^{n+1}, $\Sigma = \{\mathbf{x}' \in R^n: \mathbf{x}' \geq 0,\ \Sigma_j x'_j = 1\}$. Replacing x_i in SLP_x results in problem

$$\begin{aligned} &\text{Min } \mathbf{c}^T D\mathbf{x}' \\ &\text{s.t. } AD\mathbf{x}' - x'_{n+1}\mathbf{b} = 0 \\ &\qquad \mathbf{e}^T\mathbf{x}' + x'_{n+1} = 1 \\ &\qquad (\mathbf{x}', x'_{n+1}) \geq 0, \end{aligned}$$

where $D = \text{diag}(x_i)$, $i = 1, ..., n$. The denominator in the objective function is discarded as the fractional programming problem has also optimum value zero (see section 2.1). The centre of the the simplex Σ is interior feasible.

Method 2:

The use of a projective transformation to convert SLP_x into homogeneous form can be dropped if we assume that an upper bound B on the sum of all entries of $\mathbf{x}$ is available [Tomlin, 1985; Turner, 1987], i.e. $\Sigma_j x_j \leq B$. In this case the conversion may proceed as follows.

Introduce variable $x_{n+1} = 1$ such that $A\mathbf{x} = x_{n+1}\,\mathbf{b}$, and adjoin constraint $x_{n+1} = 1$ to the problem. This leads to

$$\begin{aligned} &\text{Min } \mathbf{c}^T\mathbf{x} \\ &\text{s.t. } A\mathbf{x} - x_{n+1}\mathbf{b} = 0 \\ &\qquad x_{n+1} = 1 \\ &\qquad (\mathbf{x}, x_{n+1}) \geq 0. \end{aligned}$$

As $\Sigma_j x_j \leq B$, another slack variable can be added such that

$$\Sigma_j x_j + x_{n+2} = B, \text{ or}$$

$$\Sigma_j x_j + x_{n+1} + x_{n+2} = 1 * B = x_{n+1}B, \text{ as } x_{n+1} = 1.$$

Transferring the RHS into the left-hand side and factoring leads to

$$\Sigma_j x_j + (1-B)x_{n+1} + x_{n+2} = 0.$$

Scale the variables such that their sum is n: $\mathbf{x}' = n\mathbf{x}/B$. As two variables have been added so far, two elements are appended to the vector $\mathbf{c}$, i.e. $\mathbf{c}'^T = (\mathbf{c}^T, 0, 0)$. Assume that optimum objective value z^* is known, i.e. $\mathbf{c}^T\mathbf{x} = \mathbf{c}'^T\mathbf{x}' = z^*$. Then it is easy to transform the problem into one with target value zero. We have

$$\mathbf{c}'^T\mathbf{x}' - z^* = 0, \text{ and } \textstyle\sum_j x'_j = n + 2 = n'.$$

Hence

$$\begin{aligned} \mathbf{c}'^T\mathbf{x}' - z^* &= \mathbf{c}'^T\mathbf{x}' - (z^* / n')\textstyle\sum_j x'_j \\ &= (c'_1 - (z^* / n'))x'_1 + (c'_2 - (z^* / n'))x'_2 + \dots + (c'_{n+2} - (z^* / n'))x'_{n+2} \\ &= c''_1 x'_1 + c''_2 x'_2 + \dots + c''_{n'} x'_{n'}. \end{aligned}$$

The problem in canonical form is:

$$\text{Min } \mathbf{c}''^T\mathbf{x}'$$

subject to

$$\begin{pmatrix} A & -\mathbf{b} & 0 \\ 1 & (1-B) & 1 \end{pmatrix} \mathbf{x}' = 0$$

$$\mathbf{e}^T\mathbf{x}' = n'$$

$$\mathbf{x}' \geq 0.$$

Note that two variables and two constraints have been added and the optimization is over a simplex of sidelength 1, rather than 1/n for Σ. Although this approach allows $\mathbf{e}_{n+2}{}^T$ to be interior feasible the adjoined extra constraints and columns may destroy the original sparsity of the problem. Also, assuming that an upper bound on the sum of the variables is known can be restrictive for some problems.

Method 3:

Lustig (1985) and others suggested a simpler and more advantageous method for transforming LP problems in standard form into Karmarkar's canonical form. It consists in introducing an extra variable x_{n+1} attached to the right hand side $\mathbf{b}$. The LP problem is handled in the form

$$\begin{aligned} &\text{Min } \mathbf{c}^T\mathbf{x} - z^* x_{n+1} \\ &\text{s.t. } A\mathbf{x} - \mathbf{b}x_{n+1} = \mathbf{0} \\ &\qquad \mathbf{x}, x_{n+1} \geq 0, \end{aligned}$$

with z^* assumed to be at hand as well as an interior feasible point $\mathbf{x}^{(0)} \in R^n$. Applying the projective transformation of Karmarkar leads to

$$\begin{aligned} &\text{Min } \mathbf{c}D\mathbf{x}' - z^*x'_{n+1} \\ &\text{s.t. } AD\mathbf{x}' - \mathbf{b}x'_{n+1} = \mathbf{0} \\ &\qquad \mathbf{e}^T\mathbf{x}' + x'_{n+1} = 1 \\ &\qquad\qquad \mathbf{x}', x'_{n+1} \geq 0, \end{aligned}$$

where $D = \text{diag}(\mathbf{x}^{(0)})$, and $(\mathbf{x}', x'_{n+1}) \in R^{n+1}$. The solution in the original space is given by inverse transformation. Thus $\mathbf{x} = \mathbf{x}' / x'_{n+1}$.

The assumption that an interior feasible point is available may be dropped. Instead, the following feasibility problem can be solved.

$$\begin{aligned} &\text{Min } t \\ &\text{s.t. } A\mathbf{x} + (\mathbf{b} - A\mathbf{e}_n)t = \mathbf{b} \\ &\qquad\qquad \mathbf{x}, t \geq 0, \end{aligned}$$

for which $(\mathbf{e}^T_n, 1)^T$ is interior feasible and extremal $t = 0$. Algorithm 2.1 is thus directly applicable and will provide an interior feasible point to the original problem.

2.2.3 Solving Problems with unknown z*

So far it has been assumed that the optimum objective value of the problem in canonical form is 0, which imposes the condition that z^* of the original problem is known. The assumption is restrictive, as for most problems it is hard to estimate the optimum objective value before hand. To remove the assumption Karmarkar suggested the combination of the primal and the dual of the LP problem and use of the strong duality result, which says that if the LP problem is feasible and has finite optimum solution, then the primal and the dual have same optimum objective value. This can be expressed as $\mathbf{c}^T\mathbf{x}^* - \mathbf{b}^T\mathbf{u}^* = 0$, where $\mathbf{u}^*$ is the dual optimum solution.

The combination of the primal and dual under standard form leads to the following minimization problem

$$\begin{array}{ll}
\text{Min } t & \\
A\mathbf{x} + \mathbf{y} + (\mathbf{b} - A\mathbf{x}_0 - \mathbf{y}_0)t & = \mathbf{b} \\
A^T\mathbf{u} - \mathbf{v} + (\mathbf{c} - A^T\mathbf{u}_0 + \mathbf{v}_0)t & = \mathbf{c} \\
\mathbf{c}^T\mathbf{x} - \mathbf{b}^T\mathbf{u} + (-\mathbf{c}^T\mathbf{x}_0 + \mathbf{b}^T\mathbf{u}_0)t & = 0 \\
(\mathbf{x}, \mathbf{u}, \mathbf{y}, \mathbf{v}, t) \geq 0, &
\end{array}$$

where t is an artificial variable, driven to zero in the Phase I Karmarkar problem. The problem is in standard form, thus any of the strategies discussed earlier can be used to convert it into canonical form. Note that $\mathbf{e} \in R^{2(m+n)+1}$ is interior feasible and that the dual solutions are also found. However, a disadvantage of this approach is the considerable increase in the size of the problem

A different approach also suggested by Karmarkar (1984a) is the sliding objective function technique. The technique consists in having a lower and an upper bound for the optimum objective function value, i.e. $\ell \leq z^* \leq u$. Trial lower and upper bounds are then set up as $\ell' = \ell + 1/3(u - \ell)$ and $u' = u + 2/3(u - \ell)$. If the potential function is not improved by a constant value δ then ℓ' is lower than the optimum value. If the optimum of the objective value drops below u', then u is set to u' and new trial values are determined. Karmarkar showed that, when the algorithm is equipped with the sliding objective function technique, it retains its polynomial complexity. The method has been used by Lustig (1985) and Nickels *et al.* (1985) in their implementations of the Karmarkar algorithm. However, it has a disadvantage: it may be hard to set up appropriate values for ℓ and u.

These techniques for relaxing the requirement of known objective value are not satisfactory because of the effect they may have on the problem (growth of the size of problem in the first method) or the assumptions they are based upon (lower and upper bound for z* in the second method). In the next section variants of the Karmarkar algorithm which handle LP problems under milder assumptions will be reviewed.

2.3 Recent Developments in Karmarkar-Type Algorithms

The practical use of the Karmarkar algorithm is made difficult by the assumptions mentioned earlier but also by the need for accurate computation of the search direction and the use of a constant steplength throughout the algorithm. Recently, strategies which relax these assumptions were developed. Linear transformations [Vanderbei *et al.*, 1986; Kortanek & Shi, 1987], in other words different scalings were investigated. More classical interior point methods such as Newton methods [Vial & De Ghellinck, 1986] and Barrier methods [Gill *et al.*, 1986] which were originally intended for nonlinear optimization were also reinvestigated. Many researchers [Turner, 1986; Dennis *et al.*, 1986; Shanno & Marsten, 1988] considered using approximate rather than accurate directions to reduce work in step 2 of Algorithm 2.1, as it accounts for much of the work needed in an iteration, ($O(n^3L)$ arithmetic operations). In the following we discuss relaxed forms of the Karmarkar algorithm and alternatives which are substantially different from it.

2.3.1 Alternative Search Directions

We have already mentioned Karmarkar's suggestion of using an approximate direction **p** at iteration k+1 obtained by a rank-one updating of $(BB^T)^{-1}$ in the expression of **p**. This reduces the overall complexity of the algorithm by $O(\sqrt{n})$ operations. Shanno (1988) takes a similar approach and uses a Fletcher-Powell rank-one update of a Cholesky factorization of BB^T. The Fletcher-Powell update [Gill *et al.*, 1981] is based on the observation that if $BB^T = LL^T$, L a lower triangular matrix, and if only few elements of L change in each iteration then it is possible to compute a good approximation $L'L'^T$ to it using rank-one updates rather than a fresh factorization.

In [Dennis *et al.*, 1986; Turner, 1987] a similar idea is considered. Their approach is to approximate direction **p** by use of a nonsingular approximation D' to the diagonal matrix D. Thus, again rank-one updating of the factorization of BB^T is possible. The adopted updating strategy is that used in the classical variable-metric algorithm, i.e. the BFGS updating method [Gill *et al.*, 1981].

Recall that the expression of the direction vector is $\mathbf{p} = [I - B^T(BB^T)^{-1}B]D\mathbf{c}$. The proposed approximation is $\mathbf{p}' = D^{-1}D'[I - B'^T(B'B'^T)^{-1}B']D'^T\mathbf{c}$, where

$$B' = \begin{bmatrix} AD' \\ \mathbf{e}^T D^{-1} D' \end{bmatrix} = BD^{-1}D'.$$

Note that $\mathbf{p} = \mathbf{p}'$ when $D = D'$. The vector $-\mathbf{p}'$ is a feasible direction for the linear programming problem PC as it can be shown to be in the null space of B, and a descent direction for the potential function (2.2.1), (Theorem 4.2 of Turner, 1987).

The best performance of the variable-metric variant of the Karmarkar algorithm [Turner, 1987] was observed for the approximation obtained from

$$D'_{k+1} = D'_k + \frac{\left(\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)} - D'^{(k)}\mathbf{v}\right)\mathbf{v}^T}{\mathbf{v}^T\mathbf{v}}$$

for some $\mathbf{v} \in R^n$.

In the course of the algorithm a restarting strategy may be taken when the updates do not lead to much improvement in the reduction of the objective function value. The current point is then considered as the starting one and the matrix BB^T is refactored. Turner showed that the algorithm retains polynomial complexity.

The correspondence between null space projections and the concept of a reduced-gradient vector was investigated recently by Shanno and Marsten (1988) leading to a reduced-gradient variant of the Karmarkar algorithm. However, as will be seen, this method did not perform well in practice.

The inexact projections used by Goldfarb and Mehrotra (1988a, 1988b) were inspired by the following observation. The projected steepest direction descent $\mathbf{p}$ used in the Karmarkar algorithm belongs to a cone of acceptable directions in the null space of B. Thus any of the directions in the cone can be selected and used in the optimization over a ball subproblem solved at each iteration of the algorithm. Exact computation of $\mathbf{p}$ is therefore relaxed. Goldfarb and Mehrotra perform this selection by solving approximately the least squares problem arising when computing $\mathbf{p}$. Their algorithm retains the polynomial complexity of Algorithm 2.1.

2.3.2 Alternative Rescaling

One of the most interesting variants of the Karmarkar algorithm is that proposed by Vanderbei *et al.* (1986). The original feature in this variant is the use of a different rescaling based on a linear transformation as compared to the projective transformation of all the variants discussed so far. The algorithm works in the positive orthant rather than the simplex and handles LP problems in the standard form SLP_x. The algorithm, seems to be similar to that briefly described by Dikin (1967).

The idea, as in the Karmarkar algorithm, is to guarantee a "good bite" by taking a step in the direction of the projected gradient always from the centre of the feasible polytope. This is achieved by using a centring scheme based on changing units in the variables in every iteration. Given a feasible interior point $\mathbf{x}^{(0)}$, the linear transformation is

$$T_x(\mathbf{x}) = D^{-1}\mathbf{x} = \mathbf{x}',$$

where $D = \mathrm{diag}(x_1, x_2, \ldots, x_n)$. Thus, $T_x(\mathbf{x}^{(0)}) = \mathbf{x}'^{(0)} = \mathbf{e}$,

$$T_x^{-1}(\mathbf{x}') = D\,\mathbf{x}' = \mathbf{x},$$

and the transformed LP problem is

$$\begin{array}{lll} \mathrm{SLP}_{x'}: & \text{Min} & \mathbf{c}^T D\mathbf{x}' \\ & \text{s.t.} & AD\mathbf{x}' = \mathbf{b} \\ & & \mathbf{x}' \geq 0, \end{array}$$

for which $\mathbf{x}'^{(0)} = \mathbf{e}$ is a feasible point. The next iterate $\mathbf{x}'^{(1)}$ is obtained by taking a step in the direction $\mathbf{p}$ of the projection of $\mathbf{c}' = D\mathbf{c}$ onto the null space of AD, i.e.

$$\mathbf{x}'^{(1)} = \mathbf{e} - \alpha\mathbf{p} \,/\, \max_j\{\mathbf{p}_j\}, \text{ where } \alpha \in (0,1).$$

The convergence of the algorithm was established under the following assumptions.

1) The problem is bounded and feasible.

2) The problem is primal nondegenerate.

3) the problem is dual nondegenerate.

Vanderbei *et al.* observed that, in practice, the algorithm works equally well on problems not satisfying these assumptions.

Algorithm 2.2 : The Linear Scaling Algorithm

```
Initialization
    $k = 0$
    $\mathbf{x}^{(k)} = \mathbf{x}^{(0)}$
    $D = \text{diag}(x_j^{(k)}), j = 1, ..., n$
    $\alpha \in (0, 1)$
    $\mathbf{p} = H_k D\mathbf{c}$, where $H_k = I - (AD)^T[(AD)(AD)^T]^{-1}(AD)$
begin
    while $\mathbf{p} \neq 0$ do
        $\mathbf{p}' = \mathbf{p} / \max_j\{p_j>0\}$
        $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \alpha D\mathbf{p}'$
        $k = k+1$
        $D = \text{diag}(x_j^{(k)}), j = 1, ..., n$
        $\mathbf{p} = H_k D\mathbf{c}$
    endwhile
end
```

A similar approach is also taken by Cavalier & Schall (1987). They raised the problem of staying in the flat $\{A\mathbf{x} = \mathbf{b}\}$ when taking a projective step. This problem is related to finding a direction vector $\mathbf{d}$ such that $A\mathbf{d} = 0$. On computational grounds one has $A\mathbf{x} = \varepsilon$. Thus, if the new iterate is $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha\theta\mathbf{d}$, where $\alpha, \theta \in R$, then $A(\mathbf{x}^{(k)}+\alpha\theta\mathbf{d}) = \mathbf{b}+\alpha\theta\varepsilon$, hence the propagation of errors. There is also the problem of ill-conditioning of the matrix $(AD)(AD)^T$ to invert when computing $\mathbf{p}$. Cavalier and Schall (1987) attempted to alleviate these difficulties by devising an algorithm for inequality constrained LP problems. Slack variables are then added to put the problem into standard form and only the slack variables are concerned with the rescaling. This is based on the idea that inequalities are easier to satisfy than equations.

The linear rescaling algorithm is also evident in [Barnes, 1986]. Barnes considers the LP problem in standard form and its dual. Given a feasible point $\mathbf{y}$ to the primal and a scalar $\omega \in]0, 1[$, the ellipsoid

$$\sum_{j=1}^{n} \frac{(x_j-y_j)^2}{y_j^2} \le \omega$$

is in the positive orthant. Solving the problem

$$\begin{aligned} &\text{Min } \mathbf{c}^T\mathbf{x} \\ &\text{s.t. } A\mathbf{x} = \mathbf{b} \\ &\sum_{j=1}^{n} \frac{(x_j-y_j)^2}{y_j^2} \le \omega \end{aligned}$$

leads to the point $\mathbf{x}$ such that $\mathbf{c}^T\mathbf{x} < \mathbf{c}^T\mathbf{y}$. An iterative process is then constructed as follows: If $\mathbf{x}^{(0)} > 0$ and $A\mathbf{x}^{(0)} = \mathbf{b}$ then after iteration k where $\mathbf{x}^{(k)}$ is calculated, set $D = \text{diag}(x_j^{(k)})$, $j = 1, ..., n$ and find $\mathbf{x}^{(k+1)} > 0$ from the relation

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \omega \frac{D^2(c - A^T\lambda_k)}{\left\| D(c - A^T\lambda_k) \right\|},$$

where $\lambda_k = (AD^2A^T)^{-1}AD^2\mathbf{c}$ is a dual feasible solution.

Kortanek and Shi (1987) suggested a hybrid method based on the above algorithm and a purification procedure to obtain the dual basic optimal solution.

Affine variants of the Karmarkar algorithm developed by Vanderbei *et al.* (1986), Cavalier and Schall (1987) and Kortanek and Shi (1987) drop the assumption that the optimum objective value is at hand, although at the expense of polynomial complexity. It should be stressed that the loss of the polynomial complexity of affine variants is not solely due to the relaxation of this assumption, but rather because they are basically different from the projective algorithm. We recall that they use a simple translation of the current iterate to $\mathbf{e}^T$ rather than projective geometry and work in the positive orthant rather than the simplex.

Gay (1987), Ye and Kojima (1987) and Todd and Burrell (1986) described variants of Karmarkar's algorithm which do not require *a priori* knowledge of optimum objective value z*. Improving lower bounds on z* were used in these algorithms, based on the generation of dual variables. The way dual variables are generated will be treated in a subsequent chapter. Anstreicher (1986b) and more recently Nemirowskiy (1988) versions of the Karmarkar algorithm which also do not require the knowledge of z^*. It is important to mention that these variants retain the polynomial complexity of the Karmarkar algorithm.

While all algorithms described here solve linear problems, Anstreicher's version solves the fractional programming problem resulting from the transformation of PC using the inverse of T_x given earlier. A similar approach has also been taken by Padberg (1986). The idea is to make the Karmarkar algorithm a monotonic process regarding the value of z, i.e. at each iteration $z_{k+1} > z_k$. Recall that in the original projective algorithm improvement by a constant at each iteration is guaranteed for the potential function and not the objective function.

2.3.3 A Barrier Function Approach to LP

An important aspect of the Karmarkar algorithm is that of maintaining feasibility after each step and insuring reduction in the objective function value monitored by the use of a logarithmic potential function. The idea is reminiscent of the barrier and penalty functions approach in nonlinear programming, due to Frish (1955) and championed by Fiacco & McCormick (1968). Gill *et al.* (1985) suggested using this approach to LP and developed a class of projected Newton algorithms, in which Karmarkar's algorithm is a particular case [Fletcher, 1986]. LP problems are handled by being transformed into a nonlinear programming problem of the form

$$\begin{aligned} \text{BAP:}\quad & \text{Min } \Phi(x) = \mathbf{c}^T\mathbf{x} - \mu\textstyle\sum_j \text{Log } x_j \\ & \text{s.t.}\quad A\mathbf{x} = \mathbf{b}, \\ & \qquad \mathbf{x} \geq 0, \\ & \qquad \mu > 0,\ \mu \text{ is the barrier parameter.} \end{aligned}$$

The algorithm proceeds from a feasible point $\mathbf{x} > 0$ following the Newton direction $\mathbf{d} = (\nabla^2\Phi(\mathbf{x}))^{-1}\nabla\Phi(\mathbf{x})$ projected onto Ker(A) to get a feasible point $\mathbf{y}$ (i.e. $A\mathbf{y} = \mathbf{b}$). More explicitly the Newton search direction $\mathbf{d}$ is obtained as the solution of a quadratic programming problem which is the minimization of a quadratic approximation of $\Phi(\mathbf{x})$ under feasibility constraints. This amounts to problem

BQP: $\quad$ Min $\nabla\Phi(\mathbf{x})\mathbf{d} + 1/2\mathbf{d}^T\nabla^2\Phi(\mathbf{x})\mathbf{d}$

s.t. $\quad \mathbf{Ad} = 0,$

where $\nabla\Phi(\mathbf{x}) = \mathbf{c} - \mu D^{-1}\mathbf{e}$ is the gradient of $\Phi(\mathbf{x})$, $\nabla^2\Phi(\mathbf{x}) = \mu D^{-2}$ the Hessian and $D = \text{diag}(x_1, x_2, ..., x_n)$, $\mathbf{x}$ being a feasible point to BAP.

The solution of BQP gives $\mathbf{d} = \mathbf{x} - 1/\mu\ D^2(\mathbf{c} - A^T)$, where λ is the solution to the normal equations $AD^2A^T\lambda = AD^2\mathbf{c}$, and the barrier parameter is chosen as $\mu = \mathbf{x}^TD(\mathbf{c} - A^T\lambda)$ [Fletcher, 1986]. Vector $\mathbf{d}$ is a descent direction as the Hessian is positive definite when $\mathbf{x} > 0$. Thus a step of length α along $\mathbf{d}$ results into point $\mathbf{y} = \mathbf{x} + \alpha\mathbf{d}$ such that $\Phi(\mathbf{y}) < \Phi(\mathbf{x})$. Hence an iterative process can be constructed. Gill *et al.* showed that the projected Newton barrier method, for some parameter μ generates a path parallel to that followed by the projective algorithm. For $\mu=0$ the barrier method is similar to the linear rescaling algorithm of Vanderbei *et al.* (1986).

2.3.4 Newton Methods for LP

By incorporating the objective function of a LP problem as a constraint in a parametrized feasibility problem, LP can be handled as a linear system without combining the primal and dual problems. De Ghellinck and Vial (1986) proposed a polynomial Newton method for linear systems which can be used at most n times (n is the dimension of the problem space) to solve LP problems. Assume that after incorporating the objective function into the constraints set as $zx_0 - \sum_j c_jx_j = 0, j = 1, ..., n$ the following problem is obtained

PLP: $\quad \mathbf{Ax} = 0, \quad A \in R^{m\mathrm{x}(n+1)}$

$\mathbf{x} \geq 0, \quad x_0 = 1$ if b_i is replaced by $-a_{i0}$, $i=1, ..., m$.

De Ghellinck and Vial consider a related problem to PLP, i.e.

PLP': $\quad \mathbf{Ax} = 0, \quad A \in R^{m\mathrm{x}(n+1)}$

$x_j \geq 0 \quad x_0 \neq 0, j = 0, 1, ..., n.$

Solutions to PLP' are directions rather than points.

A feasible point to PLP' can be found by driving to zero the following potential function:

$$\psi(\mathbf{x}) = \frac{A\mathbf{x}}{\mathbf{e}^T\mathbf{x}}.$$

The idea is to keep the numerator constant or small and increase the denominator. This is done by taking a step $\mathbf{p} \in R^{n+1}$, from the current iterate $\mathbf{x} \in R_+^{n+1}$, i.e. $\mathbf{x} = \mathbf{x} + \mathbf{p}$, such that $\mathbf{x} + \mathbf{p} \geq 0$. Consequently, the potential function is written as

$$\psi(\mathbf{x}) = \frac{A(\mathbf{x}+\mathbf{p})}{\mathbf{e}^T(\mathbf{x}+\mathbf{p})}.$$

To keep the numerator constant, De Ghellinck and Vial suggest imposing on the new iterate the condition $A\mathbf{p} = 0$. This insures that $A(\mathbf{x} + \mathbf{p}) = A\mathbf{x}$. The problem of reducing the potential function can be formulated as a linear programming problem whose objective is the denominator $\mathbf{e}^T(\mathbf{x} + \mathbf{p})$. Explicitly, the problem is

$$\text{Max} \sum_{j=0}^{n}(x_j+p_j)$$

$$\text{s.t.} \quad A\mathbf{p} = 0$$

$$\mathbf{x} + \mathbf{p} \geq 0,$$

where $\mathbf{p}$ is the variable, i.e. the optimization is with respect to $\mathbf{p}$, $\mathbf{x}$ being a parameter such that $\mathbf{x} > 0$. However, the above problem is just as difficult to solve as the original one. Using the geometric mean and the arithmetic mean inequality, i.e.

$$\left(\prod_{j=0}^{n} x_j\right)^{1/_{n+1}} \leq \frac{1}{n+1}\sum_{j=0}^{n} x_j,$$

the following nonlinear programming problem is considered instead.

$$\text{Max} \prod_{j=0}^{n}(x_j+p_j)$$

$$\text{s.t.} \quad A\mathbf{p} = 0,$$

$$\mathbf{x} + \mathbf{p} \geq 0.$$

The advantage of this problem is that the objective is zero whenever $x_j + p_j = 0$ for some j and the nonnegativity constraint is implicitly taken dealt with by the maximization process. If any direction $\mathbf{p}$ gives a large value to the objective of the nonlinear problem then it also forces the quantity $\psi(\mathbf{x})$ to zero, which solves the problem, [De Ghellinck &

Vial, 1986]. Thus, to insure a monotonic increase in the objective function value, the strict inequalities $\mathbf{x} + \mathbf{p} > 0$ are considered in a problem equivalent to the one above, i.e.

PLP":

$$\text{Max } F(\mathbf{p}) = \sum_{j=0}^{n} \log(x_j+p_j)$$

$$\text{s.t.} \qquad A\mathbf{p} = 0.$$

$$\mathbf{x} + \mathbf{p} > 0.$$

$F(\mathbf{p})$ is concave and a Newton method can be applied to solve PLP", which amounts to solving a quadratic programming problem to get the search direction $\mathbf{p}$. Meggido (1986) argues that the nonnegativity constraints may be totally removed.

The Newton algorithm of De Ghellinck and Vial works in the positive orthant of R^{n+1} and consists of only one phase in which both the feasibility and optimality problems are solved. It is interesting to note that the algorithm generates points exterior to $A\mathbf{x} = 0$, i.e. infeasible; feasibility and optimality are attained simultaneously. It is, however, considered as an interior point method, related to Karmarkar's algorithm.

Iri and Imai (1986) also suggested a Newton-like method for LP different from the Karmarkar algorithm in that projective geometry is not used and it has superlinear convergence while the Karmarkar algorithm is only of linear convergence [Charnes *et al.*, 1984]. The method uses a Newton search direction to minimize a special barrier function free of a barrier parameter, and related to the potential function of Karmarkar. The problem considered is {Min $\mathbf{c}^T\mathbf{x}$, s.t. $A\mathbf{x} \geq 0$} and the corresponding barrier function is

$$F(\mathbf{x}) = \frac{(\mathbf{c}^T\mathbf{x})^{m+1}}{\prod_{i=1}^{m}(A_i\mathbf{x}-b_i)}.$$

The minimization of $F(\mathbf{x})$ is over the domain of feasibility $K=\{\mathbf{x}\in R^n \mid A\mathbf{x} - \mathbf{b} \geq 0\}$. Iri and Imai prove the convexity of $F(\mathbf{x})$ over K to justify the choice of the Newton method. It is assumed that the problem admits a solution, the target objective value is zero and an interior starting point is available. Iri and Imai analysed the effect of line search in the behaviour of the algorithm, as compared to taking standard steps along the chosen direction. It appears that when the line search is used their algorithm converges quadratically. However, it is not known whether it has a polynomial worst case bound.

2.4 Computational Experience

The lack of experimental results in [Karmarkar, 1984a, 1984b] generated a lot of interest in the computational side of the projective algorithm. Most of the modifications discussed in the previous sections were supported by computational experience, which although limited and not conclusive gives nevertheless good insight in the practicality of the projective algorithm and related variants. In this section some of the significant numerical results obtained with implementations of the projective algorithm will be reviewed.

Tomlin (1985) solved a set of test problems among which are the 7 nontrivial LP problems listed in Table 2.1.

Problem	Rows	Columns	Slacks
AFIRO	27	32	19
ADLITTLE	52	97	41
SHARE2B	96	79	83
ISRAEL	174	142	174
BRANDY	220	249	54
E226	223	282	190
BANDM	305	472	0

Table 2.1 Problems Statistics

His implementation is characterized by the use of constant steplengths. He also investigated the use of Givens rotations in computing the projected gradient. His most efficient code, with α set to 0.99, performed slightly better than Ketron's WHIZARD assembly language simplex code only on AFIRO. On the remaining problems of Table 1, it was slower (2 to 10 times) than WHIZARD.

The test set solved by Lustig (1985) included the first five problems of Table 1. Lustig's code of a version of the Karmarkar algorithm, using LSQR subroutine of Paige and Saunders (1982) to solve the least squares problem arising in the computation of the projected gradient, performed poorly on all the problems (10 to 115 times slower than the simplex code MINOS 5.0).

Gill *et al.* (1986) implemented a projected Newton barrier method of section 2.3.3. Their code was tested on 14 nontrivial problems including those in Table 1. Three problems (Degen 1, Degen 2 and Degen 3) are highly degenerate. They tested the code against WHIZARD and MINOS 5.0. For 3 of the problems listed above the barrier method was slower than MINOS 5.0, (2 to 5 times slower). On the other problems the barrier method and the simplex codes were comparable. The degenerate problems were solved with WHIZARD. The barrier method was approximately 2 times slower than WHIZARD.

Turner (1987) solved 8 problems among which are the first five problems of Table 1. The tests were against a code of the original Karmarkar algorithm. Periodic restarts were used in the variable-metric algorithm of Turner to reduce the number of factorizations and rank-one updates to approximate D at each step. From the results of Turner it appears that the number of iterations to get a solution is inversely proportional to the number of factorizations. The code of the variable-metric performed slightly better than that of the original Karmarkar algorithm, except for AFIRO on which it was 1.5 to 2.5 times slower.

Ye and Kojima (1987) presented limited experimental results with a variant of the Karmarkar algorithm which works on the standard LP problem, with no *a priori* knowledge of the optimum objective value. Dual variables were generated at each iteration and from their results the dual solution converges faster than the primal.

Shanno and Marsten (1988) implemented two versions of a reduced gradient algorithm, with exact and inexact-projections, within the framework of the XMP simplex code. Three problems of small and medium size were solved with the two variants of the reduced gradient Karmarkar method and the XMP code of the simplex. The reduced gradient codes took more iterations than the simplex code to solve all the problems. But the interesting result concerned the behaviour of the inexact-projection reduced gradient code: it took less iterations than the version with exact projections on all the three problems. On the whole, however, Shanno and Marsten did not think that a direct implementation of the reduced gradient variant would be competitive with the simplex or even with the original Karmarkar's algorithm. In Shanno (1988) a version of the Karmarkar algorithm was implemented with the Fletcher-Powell rank-one update of a known factorization LL^T of BB^T. He solved randomly generated LP problems with $\mathbf{c}^T\mathbf{x}^* = 0$ and such that $A\mathbf{e} = 0$. The code for the original Karmarkar algorithm with a Cholesky factorization at each step required less iterations but the number of updates (n x (number of iterations)) was higher than in the modified version.

In Nemirowskiy (1988) experimental results were presented on 16 problems, 12 of which were randomly generated. He implemented the original Karmarkar algorithm and a variant which does not require *a priori* knowledge of optimum objective value and an admissible plan (i.e. an interior feasible point). This implies that the variant works in one phase. Variations on the stopping rule and the way $(BB^T)^{-1}$ was dealt with were also considered. The two algorithms performed in the same order of efficiency with a slight advantage for his variant. Nemirowskiy reported that a simplex code performed "badly" on four of the problems.

Vanderbei, Meketon and Freedman (1986) implemented their affine variant of the Karmarkar algorithm and reported encouraging results on small dense problems. The affine variant was competitive with the revised simplex method. An implementation of the affine variant was also carried out by Cavalier and Schall (1987) and was found to be 2 to 3 times faster than Fortran subroutine ZX4LP based on simplex. The test problems were randomly generated and are of medium size. In Monma and Morton (1987) extensive numerical results obtained with a Fortran 77 code of a dual affine variant of Karmarkar's method were presented. 31 test problems including those in Table 1 were solved. Their code was tested against MINOS 5.0. On 26 of the problems the dual affine variant outperformed MINOS 5.0 (1.28 to 10.80 times faster). Four of the five problems on which the dual affine variant did not perform so well are included in Table 1. On problem ISRAEL, MINOS 5.0 was 5.84 times faster. The problem ISRAEL has three very dense columns and several dense rows. This, probably, explains why the Karmarkar variant performed badly.

Ferris and Philpott (1988) studied the performance of the Karmarkar algorithm and its rescaling variant on small to medium size problems. A line search to obtain steplength α and the sliding objective technique were investigated. Householder transformations and Givens rotations were used in solving the least squares problems to obtain the projected gradient search direction. The tests were against the MPSX code of the simplex. On average the simplex code performed better than the codes of Karmarkar algorithm variants. The point made however was that the choice of the steplength and techniques for solving the least squares problem greatly influences the behaviour of any code of the algorithm. In Nickels *et al.* (1985) and Schönlein (1986) a similar approach was also taken. Their Fortran IV code of a version of the Karmarkar with no *a priori* knowledge of z^* was tested against two MP-codes: The Marsten Code and the APEX IV simplex based package of Control Data, on eight small to medium size problems. The Karmarkar based program was faster than the Marsten Code on most problems (1.8 to 12 times faster) except for one problem for which it was over two times slower. Note that the Marsten Code is not a commercial package. APEX code, however, was 1.2 to 81 times faster than that of the Karmarkar algorithm. Schreck (1986) also implemented Karmarkar's algorithm with QR decomposition to find the search direction. He solved problems in canonical form obtained by primal-dual combination suggested in [Karmarkar, 1984a]. The tests were against APEX. The latter was uniformly better than Schreck's codes K11C and K11D (which differ only in the stopping rule) except for one (10x10)-Maximum Matching Problem solved in the same time by APEX and K11D. On other problems APEX largely outperformed K11C and K11D (>> 100 times faster). The bad performance of the Karmarkar algorithm in Schreck's implementation is due to the growth of problem size due to the primal-dual combination.

Goldfarb and Mehrotra (1988b) presented limited results with a code of their relaxed version of the projective algorithm on the first three problems of Table 1. Emphasis was put on the role of subroutine CGLS [Paige & Saunders, 1982] used when computing approximately the direction of search, and the effect of rescaling the data. From their results it appears that scaling has no substantial effect on the results. But the success of their method is dependent on CGLS, i.e on the solution of the least squares problem. Problems AFIRO and ADLITTLE were solved in acceptable iteration counts. On the other hand, the performance of their method on SHARE2B was disappointing. This lack of robustness was justified by the ill-conditioning of the least squares problem deriving from SHARE2B.

			Implementations of Karmarkar Algorithm by:							
	Simplex		Tomlin		Lustig		Gill et al.		Monma et al.	
Problems	Iter	CPU(s)	Iter	CPU(s)	Iter	CPU(s)	Iter	CPU(s)	Iter	CPU(s)
AFIRO	6	0.5	17	0.40	14	0.8	19	0.4	22	0.23
ADLITTLE	98-126	1.0-1.1	24	1.87	29	12.3	26	1.0	22	0.95
SHARE2B	91-121	1.0-1.4	23	2.80	21	67.4	23	1.4	32	2.98
ISRAEL	231-338	3.7-4.2	30	55.07	33	636.0	41	15.9	41	96.65
BRANDY	292-377	4.1-5.9	33	17.23	35	215.0	28	6.4	34	16.32
E226	471-572	7.5-7.9	37	31.66	59	644.0	37	8.5	42	18.73
BANDM	392-534	6.4-10.	47	33.13	55	771.0	33	7.9	31	15.60

Table 2.2 Comparative Results between 4 Implementations of Karmarkar Related Algorithms and MINOS 5.0 Simplex Code

2.5 Conclusions

From the numerical results briefly reviewed some interesting aspects of the Karmarkar and related algorithms can be highlighted.

1) The number of iterations required by the Karmarkar algorithm and its variants is in general low, and grows slowly with the problem size. This conclusion does not contradict Karmarkar's claim that the number of iterations required by his algorithm, in general, is O(Logn). This constitutes the most attractive feature of the algorithm.

2) Some implementations of the algorithm outperformed the simplex on realistic LP problems. But the difference in CPU time was never large enough to impose the technique for adoption as the standard way for solving LP problems. Indeed many of the results reviewed earlier speak in favour of the simplex, although the comparisons were

not very meaningful as simplex based packages have been developed and refined for years, while Karmarkar based codes are still experimental products.

3) The work in an iteration of the Karmarkar algorithm is substantial. All current variants of the algorithm are dependent on the solution, in every iteration, of a least squares problem that can be expensive. Speeding up the convergence of the algorithm, therefore, is limited by existing technology for least squares problems.

Chapter 3

Computation of the Projected Gradient and the Steplength

3.1 Introduction

After describing Karmarkar's algorithm and related work, we are faced with two major problems for its efficient implementation, namely the computation of the search direction **p** and the optimum choice of the steplength α to take along it. The present chapter is therefore in two parts. In the first part we look at the least squares (LSQ) problem and find out what is in store that can be used in the implementation of Karmarkar's algorithm. The second part is devoted to investigating the choice of the steplength.

Very few implementations discussed in Chapter 2 do not consider solving a LSQ problem when computing the projection matrix in the main step of the algorithm. In [Tomlin, 1985], it has been argued that the efficiency of the projective algorithm is limited by the technology for solving LSQ problems. Efficient solution of the LSQ problem is also relevant to our algorithm based on generating a sequence of Chebyshev points, treated in Chapter 5, and to different implementations of the projective algorithm considered in Chapter 7. Based on these arguments we found it necessary, at least for completeness, to review some important results concerning the LSQ problem. Other related information will be given as appendices and referred to when necessary.

3.2 The Linear Least Squares Problem

The method of least squares has many applications. In statistics it is used to identify and estimate parameters, in engineering it is used for curve fitting and data smoothing. In numerical analysis it is used as an "extension" to the well known Gaussian Elimination to overdetermined (underdetermined) systems of linear equations.

The use of LSQ can be traced back three thousand years, to Chinese mathematicians [Longley, 1984]. It is, however, credited to Gauss. Methods for LSQ problems predate computers, although the development of efficient algorithms with sparsity and numerical stability considerations are recent and strongly linked to the availability of digital computers.

3.2.1 LSQ Problem and Normal Equations

The LSQ problem is to minimize the norm of the residual vector $\mathbf{r} = \mathbf{b} - A\mathbf{x}$ of a system of linear equations $A\mathbf{x} = \mathbf{b}$. Although any norm may be used, it is generally the Euclidean norm which is considered. The LSQ problem is formulated as follows

$$\min_{\mathbf{x}} \|\mathbf{b} - A\mathbf{x}\|_2 \qquad (3.2.1)$$

The LSQ problem and the normal equations are almost inseparable. They are naturally derived as follow.

The residual vector $\mathbf{r}$ must be orthogonal to the column space of A (or space of A^T). This condition is expressed as $\mathbf{r} \in \ker(A^T)$, i.e. $A^T\mathbf{r} = 0$ or $A^T(\mathbf{b} - A\mathbf{x}) = 0$, which leads to the normal equations $A^TA\mathbf{x} = A^T\mathbf{b}$. The cross-product A^TA is a positive definite matrix.

3.2.2 Data Characteristics and Algorithm Performance

Many techniques are available to solve the LSQ problem. The diversity in the ways LSQ problems may be approached is due to the different characteristics these problems may have. Two important characteristics of the data of the problems are the ill-conditioning of the matrix A and its sparsity, which make techniques for LSQ differ in their numerical properties and execution time. Any successful method takes account of these two features. However, despite continuous research for powerful methods, large scale and ill-conditioned problems remain difficult and expensive to solve.

Numerical stability of a technique applied to large ill-conditioned LSQ problems is achieved, when possible, mainly by scaling and preconditioning the data of the problems. Sparsity preservation and exploitation is achieved by appropriate matrix decomposition and ordering.

3.2.3 Numerical Stability and Condition Number

Every operation performed during computation, gives rise to some error. This error may decay or grow in subsequent calculations. In some cases, errors grow so large that the computed result is totally redundant. A procedure leading to such results is labelled numerically unstable. However, some problems are inherently unstable or ill-conditioned, which may cause bad performance of most procedures on such problems regardless of the precautions taken [Kronsjö, 1987].

Instability of a solution and ill-conditioning of the corresponding problem may be monitored by perturbing the data and measuring the effect this has on the solution. In the case of a system of linear equations $A\mathbf{x} = \mathbf{b}$, where A is square and non-singular, the measure of instability is contained in the quantity $||A||.||A^{-1}||$ defined as the condition number, Cond(A). As $||I|| = 1$, for every subordinate norm, and $I = AA^{-1}$ then $1 = ||I|| \leq ||A||\,||A^{-1}|| = \text{Cond}(A)$. Thus $\text{Cond}(A) \geq 1$ for any matrix.

The condition number indicates the maximum effect of perturbations in A and **b** on the exact solution of $A\mathbf{x} = \mathbf{b}$. If Cond(A) is "large", the exact solution may be substantially changed by even small changes in the data. A is often said to be ill-conditioned. If Cond(A) is "small", A is said to be well-conditioned.

3.2.4 Scaling and Preconditioning

Scaling and preconditioning are means for improving the condition of a matrix. The basic purpose of scaling is to make the variables of the scaled problem have the same magnitude and order unity in the solution region. The common way of scaling a matrix is by multiplying its rows or columns by a factor so that all entries are about size 1. Scaling is not always easy and satisfactory. Indeed, there is no automatic way of satisfactorily scaling any matrix.

Preconditioning is another way of scaling. It is aimed at reducing the condition number of a matrix by multiplying it by suitable matrix. Preconditioners are also viewed as accelerators of convergence. Among the techniques for LSQ, the iterative ones are most dependent on preconditioning [Gill *et al.*, 1981].

3.2.5 Sparsity

Sparsity is the characteristic of a matrix with "many" zero entries. Although, It is difficult to exactly define a sparse matrix, we call a matrix sparse when it is profitable to exploit its zeros [George & Liu, 1981].

Sparsity exploitation is aimed at reducing the CPU time and storage requirements of procedures for matrix computation. This is justified by the redundancy of the following operations.

If a is nonzero then: $0 . a = 0, 0 + a = a, 0 / a = 0, 0 . 0 = 0$.

In other words, it is unnecessary to allocate any computing power to these operations as the results are obvious. Also there is no need to store the zero elements.

One of the main problems in solving sparse systems is that when the matrix is factored, it suffers fill-in, i.e., nonzeros are created as a consequence of the factorization. Thus, sparsity tends to be destroyed. In the case of the normal equations, for example, the Cholesky factor L has more nonzeros than the lower part of A^TA. However, it has been observed that a judicious reordering of the matrix rows and columns can drastically reduce fill-in. Such a reordering is practically embodied in a permutation matrix, which is defined as follows.

A permutation matrix P is a square matrix whose columns are some permutation of those of the identity matrix. Matrix P is orthogonal, i.e. $P^TP = I$. (See Appendix C for details about ordering algorithms.)

Sparse matrix technology was founded by Ralph Willoughby of I.B.M. in the 60's [Duff, 1981]. Since then it dominated the design of efficient software in numerical computations of large systems. In the following sections and subsequent chapters, the relevance of sparsity considerations for the solution of the LSQ problem and the implementation of the projective algorithm will be underlined.

3.2.6 Solving the Least Squares Problem

As mentioned earlier there are many techniques for solving LSQ problems. The choice of a technique may be determined by two main criteria: Numerical stability and Sparsity exploitation (i.e. cost). Unfortunately, no single technique completely fulfils these criteria, as problems differ a great deal in the condition of their data and their sizes. For small scale problems, even when they are ill-conditioned, most techniques can be successfully applied. However, when the problems are large, the choice of a suitable technique becomes crucial.

Techniques for LSQ problems may be divided into two categories:

- Direct and
- Iterative.

3.2.6.1 Direct Methods

a) Cholesky Factorization Technique

The Cholesky method is a symmetric variant of the Gaussian elimination tailored to symmetric positive definite (SPD) matrices. Suppose we have to solve the system $Ax = \mathbf{b}$, where A is SPD matrix, using the Cholesky method. Then a triangular factorization of A is obtained such that $A = LL^T$, where L is a lower triangular matrix (see Appendix B for computation details of L). The system at hand may be written as

$$LL^T\mathbf{x} = \mathbf{b}.$$

Put $\quad L^T\mathbf{x} = \mathbf{y},$

and solve $\quad L\mathbf{y} = \mathbf{b}$

by a forward substitution. Then solve $L^T\mathbf{x} = \mathbf{y}$ by a back substitution to obtain $\mathbf{x}$.

The algorithmic form of the Cholesky method applied to $A\mathbf{x} = \mathbf{b}$, with ordering brought into play is given below [Heath, 1984].

Algorithm 3.0

```
1- Find a permutation matrix P.
2- Factorize $P^TA^TAP$ to find a sparse Cholesky factor L.
3- Solve $L^T\mathbf{z} = P^TA^T\mathbf{b}$.
4- Solve $L\mathbf{y} = \mathbf{z}$.
5- Restore original order: $\mathbf{x} = P\mathbf{y}$.
```

b) Advantages/Disadvantages of Cholesky Method

Speed and widespread availability of Cholesky method are among its major assets.

- For dense problems with m >> n, the number of arithmetic operations needed is approximately half of that taken by direct methods
- For sparse problems, there are excellent software packages available on the market such as YMSP, SPARSPAK, MA27.

On the other hand many people find it unreliable for the following reasons.

- Numerical difficulties may originate from problems such as:
    * Potential loss of information when explicitly computing the cross-product A^TA and $A^T\mathbf{b}$. This arises from the truncation of the numerical values resulting from the products due to limited precision on the computer.
    * The condition number of A^TA is the square of that of A, hence an accurate solution to the LSQ problem at hand may be difficult to get if not impossible, especially when A is already poorly conditioned.

- A suitable accuracy requires high working precision which results in an increase in storage requirement for very large problems.
- Explicitly computing the cross-product A^TA destroys the sparsity of the original problem.

c) Orthogonal Methods

The basic idea is to avoid explicit formation of the cross-product A^TA, by computing its Cholesky factor R directly from A. This can be done by orthogonal factorization. An orthogonal matrix Q of order m is one which satisfies the relation $Q^TQ = I$. Such a matrix is used to reduce A and b into the following forms:

$$QA = \begin{bmatrix} R \\ 0 \end{bmatrix}, \text{ and } Q\mathbf{b} = \begin{bmatrix} \mathbf{c} \\ \mathbf{d} \end{bmatrix}, \qquad (3.2.2)$$

where **c** is of order n, and **d** of order (m-n) and R is triangular (nxn)-matrix.

Based on the property of Q, it can be written

$$A^TA = A^TIA = A^TQ^TQA = \begin{bmatrix} R^T 0 \end{bmatrix} \begin{bmatrix} R \\ 0 \end{bmatrix} = R^TR.$$

This shows that R is the Cholesky factor of A^TA.

Three main methods are available for computing the reductions (3.2.2):

- Gram-Schmidt Orthogonalization, [Longley,1984]
- Householder Reflections, [Kronsjö, 1987]
- Givens Rotations, [Golub & Van Loan, 1983]

d) Advantages/Disadvantages of Orthogonalization Methods

Gram-Schmidt and Householder methods reduce A to triangular form by annihilating the subdiagonal elements in an entire column at each step. The methods are effective for dense matrices. For sparse matrices, they present some drawbacks: While a column is annihilated, nonzero elements are eventually created where there was a zero before in the remaining columns not yet zeroed by the orthogonalization process. These nonzero elements will be zeroed later, but they must be stored for the mean time, which increases storage requirements [Heath, 1984].

The Givens method reduces A to triangular form by annihilating subdiagonal elements in a row, unlike the two other methods. It is, however, similar to them, although it presents the advantage of introducing zeros more selectively [Heath, 1984]. The order in which the rows are reduced does not affect the zeroing process with respect to the correctness of the result, but may be used to advantage to preserve sparsity in the factor R. It should be said that Givens algorithm takes approximately twice the time needed by Gram-Schmidt and Householder algorithms. But another version of the algorithm known as the "Fast Givens Rotations", has been discovered by Gentleman (1973). The performance of this second version is not superior to the original algorithm, in practice. However, many people favour the use of the Givens algorithm due to the flexibility of the zeroing process [Heath, 1984]. In general the three methods present the same disadvantage: Even if A and R are sparse, it is unlikely that the orthogonal matrix Q will be particularly sparse. In section 3.6 we will see how this problem can be alleviated by using a partitioning algorithm based on QR methods.

3.2.6.2 Iterative Methods for Least Squares

Iterative methods may be good alternatives to direct methods for some large sparse LSQ problems. One of their advantages, (and that of all iterative processes for any class of problems, like the projective algorithm itself), is the possibility to stop the iterative process when an approximate solution to the problem at hand is reached. This obviously is not possible with direct methods. Another advantage is also the difficulty of obtaining an accurate solution with direct methods, for some problems. In this respect, iterative methods are more suitable, as accuracy may be monitored.

A simple iterative scheme, based on the normal equations, referred to in [Golub, 1965], is as follows.

Let $\mathbf{x}^{(0)}$ be an arbitrary vector, then solve $(A^TA + \alpha I)\mathbf{x}^{(k+1)} = A^Tb + \alpha\mathbf{x}^{(k)}$. The convergence of this process can be proved, when $\alpha > 0$ and the spectral radius of $\alpha(A^TA + \alpha I)^{-1}$ is less than 1. Its implementation can be done using orthogonal transformations.

a) Conjugate Gradient Methods

Conjugate gradient methods are popular because of their robustness and stability for large problems. They are called upon to replace direct methods, when these are not viable because of the size or density of the problems matrices. Conjugate gradient methods refer to a wide class of optimization algorithms which generate search directions without storing a matrix [Gill *et al.*, 1984]. There are two types of conjugate gradient methods: The linear and the nonlinear methods. For our purposes, linear conjugate gradient methods will be considered.

Originally, conjugate gradient methods were designed to solve, iteratively, positive definite systems of linear equations. The iterative process uses the relation $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k\mathbf{p}_k$,where α_k is a non-negative scalar called steplength, and $\mathbf{p}_k$ a vector direction of search. The vector $\mathbf{p}_k$ is obtained as follows:

If the positive definite system to be solved is $Q\mathbf{x} = -\mathbf{c}$, the direction of search can be computed as $\mathbf{p}_{k+1} = -(Q\mathbf{x}^{(k+1)} + \mathbf{c}) + \beta_k\mathbf{p}_k$, with

$$\beta_k = \frac{\mathbf{g}_{k+1}^T Q\mathbf{p}_k}{\mathbf{p}_k^T Q\mathbf{p}_k},$$

where $\mathbf{g}_k = Q\mathbf{x}^{(k)} + \mathbf{c}$. The steplength α_k is evaluated with the formula:

$$\alpha_k = -\frac{\mathbf{g}_k^T\mathbf{p}_k}{\mathbf{p}_k^T Q\mathbf{p}_k}.$$

b) LSQR Algorithm of Paige &Saunders

LSQR algorithm was designed to solve nonsymmetrical systems of linear equations, LSQ problems and damped LSQ problems of the form:

$$\min_{\mathbf{x}} \left\| \begin{pmatrix} A \\ \lambda I \end{pmatrix} \mathbf{x} - \begin{pmatrix} \mathbf{b} \\ 0 \end{pmatrix} \right\|_2 \qquad (3.2.3)$$

where λ is a scalar. The algorithm was intended to solve large and sparse problems. It is based on the algorithm of Golub and Kahan, cited by Paige and Saunders (1982), to reduce matrix A to a lower bidiagonal form. However, this algorithm is itself a variation on the Lanczos process (or tridiagonalization) for symmetric matrices. The solution to (3.2.3) satisfies the symmetric system

$$\begin{bmatrix} I & A \\ A^T & -\lambda^2 I \end{bmatrix} \begin{bmatrix} \mathbf{r} \\ \mathbf{x} \end{bmatrix} = \begin{bmatrix} \mathbf{b} \\ 0 \end{bmatrix}, \qquad (3.2.4)$$

where $\mathbf{r} = \mathbf{b} - A\mathbf{x}$. Note that (3.2.4) is a symmetric system. Hence, application of the Lanczos process is possible and leads to the forms

$$\begin{bmatrix} I & B_k \\ B_k^T & -\lambda^2 I \end{bmatrix} \begin{bmatrix} \mathbf{t}_{k+1} \\ \mathbf{y}_k \end{bmatrix} = \begin{bmatrix} \beta_1 \mathbf{e}_1 \\ 0 \end{bmatrix}, \qquad (3.2.5)$$

$$\begin{bmatrix} \mathbf{r}_k \\ \mathbf{x}_k \end{bmatrix} = \begin{bmatrix} U_{k+1} & 0 \\ 0 & V_k \end{bmatrix} \cdot \begin{bmatrix} \mathbf{t}_{k+1} \\ \mathbf{y}_k \end{bmatrix}, \qquad (3.2.6)$$

where B_k is (k+1)xk and lower bidiagonal and $\mathbf{y}_k$ is the solution of the damped least squares problem

$$\min_{\mathbf{y}_k} \left\| \begin{pmatrix} B_k \\ \lambda I \end{pmatrix} \mathbf{y}_k - \begin{pmatrix} \beta_1 \\ 0 \end{pmatrix} \right\|_2 .$$

Orthogonal transformations may then be used to reliably solve it.

The algorithm LSQR is analytically equivalent to conjugate gradient methods. It generates a sequence of approximations $\{\mathbf{x}_k\}$ such that the residual norm $\| \mathbf{r}_k \|$ is monotonically reduced. Paige and Saunders (1982) claim that it is numerically more reliable than the standard conjugate gradient methods, in various circumstances. In our implementation of the projective algorithm, LSQR is also used.

3.2.7 An Updating Algorithm for Least Squares

Updating methods are an important feature of LSQ problems. In the real world, problem data, most of the time, are incomplete. Often new observations are made after the problem has been already solved. It is, therefore, crucial to be able to incorporate the effects of these observations into the solution without having to solve it *de novo*. In our case, however, the usefulness of such techniques is mainly concerned with efficient exploitation of the sparsity of the problem. When the problem matrix is sparse except for few rows, it is attractive to discard the nonsparse rows, which will certainly cause severe fill in the Cholesky factor, solve the resulting incomplete problem, then update its solution taking account of the removed rows. In the following we shall present an updating algorithm, due to Heath (1981), which we use in our implementations of variants of the projective method. The algorithm of Heath (see Appendix D), however, involves storing an orthogonal matrix which can be very large. In the following we shall present an analogous algorithm which does not require explicitly the orthogonal matrix. It will be shown that the solutions returned by both algorithms are equivalent.

Consider the partitioning of A and $\mathbf{b}$ of (3.2.1) into $A = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix}$, and $\mathbf{b} = \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{bmatrix}$.

Then (3.2.1) can be written as

$$\min_{\mathbf{x}} \left\| \begin{pmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{pmatrix} - \begin{pmatrix} A_1 \\ A_2 \end{pmatrix} \mathbf{x} \right\|_2 \qquad (3.2.7)$$

Let $\mathbf{r}1(\mathbf{x}) = \mathbf{b}_1 - A_1\mathbf{x}$ and $\mathbf{r}_2(\mathbf{x}) = \mathbf{b}_2 - A_2\mathbf{x}$ and solve the incomplete problem

$$\min_{\mathbf{y}} \|\mathbf{b}_1 - A_1\mathbf{y}\|_2 \qquad (3.2.8)$$

in variable $\mathbf{y}$, using orthogonal factorization

$$A_1 = Q^T \begin{bmatrix} R \\ 0 \end{bmatrix} \quad \text{and} \quad \mathbf{b}_1 = \begin{bmatrix} \mathbf{c} \\ \mathbf{d} \end{bmatrix}.$$

If $\mathbf{z}$ is the effect of the removed rows on the solution $\mathbf{x}$ to (3.2.1), then $\mathbf{x} = \mathbf{y} + \mathbf{z}$. This leads to the following form of Heath's algorithm.

Algorithm 3.1

1- Solve the sparse problem (3.2.8) using orthogonal factorization or Cholesky method to obtain $\mathbf{y} = R^{-1}\mathbf{c}$.

2- Compute $F = A_2R^{-1}$.

3- Compute $\mathbf{r}_2(\mathbf{y}) = \mathbf{b}_2 - A_2\mathbf{y}$.

4- Solve $(I + FF^T)\mathbf{u} = \mathbf{r}_2(\mathbf{y})$.

5- Compute $\mathbf{z} = R^{-1}F^T\mathbf{u}$.

6- $\mathbf{x} = \mathbf{y} + \mathbf{z}$.

Lemma 3.1: The solutions returned by Algorithm 3.1 and Algorithm D (see Appendix D) are equivalent.

Proof: First we consider the solution returned by Algorithm D, i.e. $\mathbf{x} = \mathbf{y} + \mathbf{z}$ where $\mathbf{z} = R^{-1}U^TL^{-1}\mathbf{r}_2(\mathbf{y})$, U being the orthogonal matrix used in the factorization of $[A_2R^{-1} \quad I]^T$ and L its Cholesky factor. We have

$$U\begin{bmatrix} R^{-1}A_2^T \\ I \end{bmatrix} = \begin{bmatrix} L^T \\ 0 \end{bmatrix}, \tag{3.2.9}$$

or equivalently $UR^{-1}A_2^T = L^T$. Thus $R^{-1}A_2^T = U^TL^T$ and $U^T = R^{-T}A_2^TL^{-T}$. Substituting U^T in the expression of $\mathbf{z}$ we obtain

$$\mathbf{z} = R^{-1}(R^{-T}A_2^TL^{-T})L^{-1}\mathbf{r}_2(\mathbf{y})$$

Replace $\mathbf{z}$ by its expression in the solution to the full problem to get

$$\mathbf{x} = \mathbf{y} + R^{-1}(R^{-T}A_2^TL^{-T})L^{-1}\mathbf{r}_2(\mathbf{y}). \tag{3.2.10}$$

Now consider the solution returned by Algorithm 3.1, i.e., $\mathbf{x} = \mathbf{y} + \mathbf{z}$, where $\mathbf{z} = R^{-1}F^T\mathbf{u} = R^{-1}R^{-T}A_2^T\mathbf{u}$, $\mathbf{u}$ being the solution to the symmetric positive definite system $(I + FF^T)\mathbf{u} = \mathbf{r}_2(\mathbf{y})$. Assume that $\mathbf{u}$ is obtained by the Cholesky method, the Cholesky factor of $(I + FF^T)$ is similar to that obtained in the orthogonal factorization (3.2.9) as one can see from

$$\left(I + FF^T\right) = \left(FF^T + I\right) = (F \quad I)\begin{bmatrix} F^T \\ I \end{bmatrix} = \begin{bmatrix} A_2R^{-T} & I \end{bmatrix}\begin{bmatrix} R^{-1}A_2^T \\ I \end{bmatrix} = LL^T.$$

Thus $\mathbf{u}$ can be written $\mathbf{u} = L^{-T}L^{-1}\mathbf{r}_2(\mathbf{y})$, and substituting it into the expression of $\mathbf{z}$ we obtain

$$\mathbf{z} = R^{-1}R^{-T}A_2^TL^{-T}L^{-1}\mathbf{r}_2(\mathbf{y})$$

Replace $\mathbf{z}$ by its expression in the solution to the full problem to get

$$\mathbf{x} = \mathbf{y} + R^{-1}(R^{-T}A_2^TL^{-T})L^{-1}\mathbf{r}_2(\mathbf{y}). \tag{3.2.11}$$

Expressions (3.2.10) and (3.2.11) of the updated solutions givens by Algorithm 3.1 and Algorithm D are similar. Q.E.D.

3.2.8 Exploiting the Sparsity of The Right-Hand Side

Algorithm 3.1 is useful not only in preserving the sparsity of the problem matrix, but also to exploit that of the RHS. Exploiting any sparsity in the RHS is tedious and expensive [Duff *et al.*, 1986]. However, when the sparsity is favourably distributed, i.e. the vector $\mathbf{b}$ presents large sequences of zeros, then exploiting it is worthwhile. We are mainly interested in the situation where the nonzeros in the RHS are only few and correspond to some relatively full rows in the problem matrix. Thus the partitioning of the problem as in (3.2.7) is most appropriate, i.e. A_2 contains the full rows and $\mathbf{b}_2$ the nonzeros of the RHS. In this case Algorithm 3.1 is dominated by the orthogonal factorization carried out in step 1-. The forward substitution $\mathbf{y} = R^{-1}\mathbf{c}$ is totally discarded, as the solution to the incomplete LSQ problem (3.2.8) when $\mathbf{b}_1 = \mathbf{0}$, is obviously $\mathbf{y} = \mathbf{0}$. The solution to the full problem is, therefore, reduced to the updating vector $\mathbf{z}$. Thus $\mathbf{x} = \mathbf{z}$. When matrix A_2 has only few rows, applying Algorithm 3.1 and exploiting the sparsity of the RHS is very attractive. We shall see in Chapter 5 the relevance of this algorithm to our work.

3.3 Optimum Choice of Steplength α

In all interior-point methods the choice of the step size is crucial and may greatly influence their convergence. One of the main problems related to the step size is keeping feasible, as it was noted in the Brown-Koopmans algorithm, cited by Charnes *et al.* (1984). In Karmarkar type algorithms things have not much changed and the choice of the steplength remains crucial, although keeping feasible is mainly dealt with by considering at each iteration a trivial optimization problem, i.e., minimizing a linear functional over a sphere. Obviously, the minimum is at the surface of the sphere. However, the radius of the sphere must be defined in conjunction with the size of the step along the negative projected gradient of the linear functional to be minimized (that is the transformed objective function of the LP problem). This results into problem $P_{x'_s}$ of Chapter 2.

Karmarkar (1984a) recommended a small step size ($\alpha = 1/4$) mainly to maintain polynomial complexity of the algorithm, a better approximation of the objective function in the transformed space and numerical stability. However practical experience supported the use of a much larger α than 1/4. Moreover, using a constant step size for every iteration is counter-productive, because at each iteration a different minimization problem is solved.

In the following, aspects of the choice of α are considered and suggestions for an optimum choice are made.

3.3.1 Constant Steplength

Theorem 4 of Karmarkar (1984a) states that if $\mathbf{x}^* = \mathbf{a}_0 - \alpha\mathbf{p}'$ minimizes the transformed objective function then the value of the transformed potential function F' is such that $F'(\mathbf{x}^*) \le F'(\mathbf{a}_0) - \delta$, where $\mathbf{a}_0$ is the centre of the inscribed sphere in the unit simplex and δ a constant defined as

$$\delta(n) = \alpha - \frac{\alpha^2}{2} - \frac{\alpha^2 n}{(n-1)\left[1 - \alpha\sqrt{\frac{n}{n-1}}\right]}.$$

Thus, $\lim_{n\to\infty} \delta(n) = \alpha - \alpha^2/2 - \alpha^2/(1-\alpha)$,

and $\lim_{n\to\infty} \delta(n) \geq 1/8$ if $\alpha = 1/4$.

The constant δ is a bound on the decrease in the potential function. Plotting δ against α, (Fig 3.1), clearly shows that the optimum reduction δ is obtained for $\alpha = 0.25$.

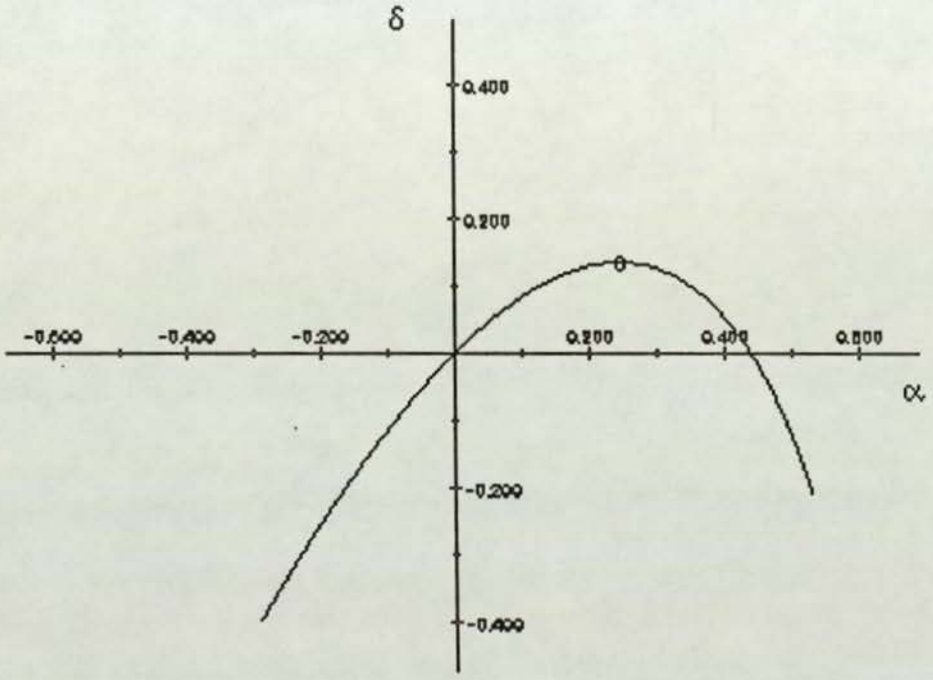


Fig 3.1 Maximum Decrease in Potential Function Occurs for $\alpha \approx 0.2453$.

Hence the use of such a constant as steplength is justified. However, in practice, values of α closer to 1 result in a faster convergence [Shetty & Ben Daya, 1985; Tomlin, 1985; Lustig, 1985]. To explain this phenomenon, let us put aside for a while, the reasons why α is chosen equal to 1/4, and consider the decrease in the objective function in the transformed space, i.e.,

$$\delta(\alpha) = \frac{\mathbf{c}^T D a_0}{\mathbf{e}^T D a_0} - \frac{\mathbf{c}^T D\left(a_0 - \alpha \mathbf{c}^T D \mathbf{p}'\right)}{\mathbf{e}^T D\left(a_0 - \alpha \mathbf{c}^T D \mathbf{p}'\right)}$$

After some manipulations, it can be seen that the function has a curve similar to that represented in Fig 3.2. It appears that large values of α lead to large decreases in the transformed objective function.

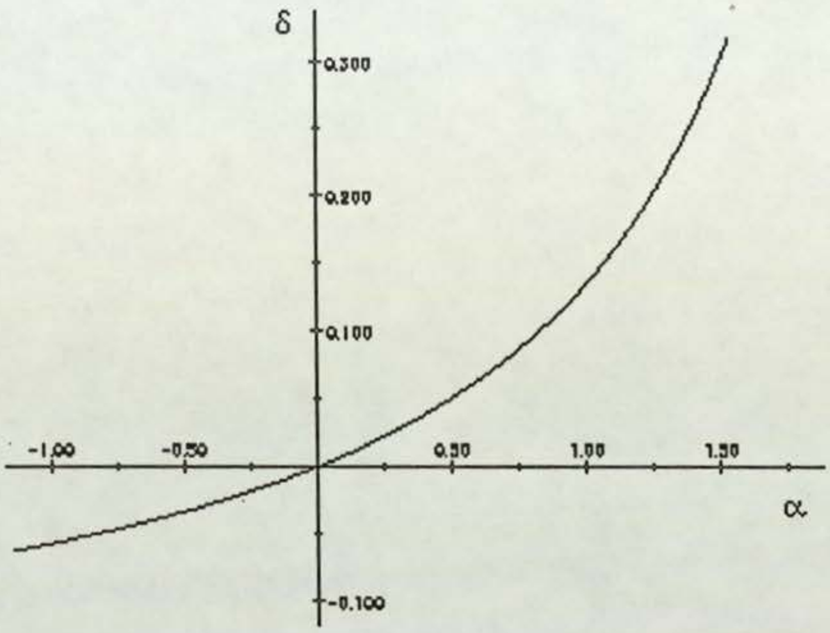


Fig 3.2 Decrease in $\mathbf{c}^T D\mathbf{x}'/\mathbf{e}^T D\mathbf{x}'$ After Taking a Step of Length α

Now we take a similar approach to investigate the conditions under which α will lead to a decrease in the objective function value, i.e. $z^{(k+1)} < z^{(k)}$, of problem PC of Chapter 2. First of all the step should not lead to a non-feasible point. This is achieved by imposing on α the condition, directly extracted from $\mathbf{x}' = 1/n\ \mathbf{e} - \alpha\mathbf{p}'$, and represented as $\alpha \leq \min \gamma$, where $\gamma \in \Gamma = \{1/(np'_j),\ p'_j > 0,\ j = 1, ..., n\}$. It can be shown that to guarantee a decrease in the objective function value, α may be required to take a negative value or a value larger than 1 [Tomlin, 1985]. From the relations

$$z = \mathbf{c}^T\mathbf{x},$$

$$\mathbf{x}' = 1/n\ \mathbf{e} - \alpha\mathbf{p}'$$

and

$$\mathbf{x}^{(k+1)} = \frac{D\mathbf{x}'}{\mathbf{e}^T D\mathbf{x}'},$$

one can write

$$\mathbf{x}^{(k+1)} = \frac{D(\mathbf{e}-n\alpha\mathbf{p}')}{\mathbf{e}^T D(\mathbf{e}-n\alpha\mathbf{p}')},$$

and

$$z^{(k+1)} = \frac{z^{(k)} - \alpha n \sum_j c_j x_j^{(k)} \mathbf{p}'_j}{1 - \alpha n \sum_j x_j^{(k)} \mathbf{p}'_j}.$$

Put

$$s = n \sum_j c_j x_j^{(k)} \mathbf{p}'_j \quad \text{and} \quad r = n \sum_j x_j^{(k)} \mathbf{p}'_j.$$

Then

$$z^{(k+1)} = \frac{z^{(k)} - \alpha s}{1 - \alpha r}.$$

Thus

$$z^{(k)} - z^{(k+1)} = \alpha\left(s - rz^{(k+1)}\right). \tag{3.3.1}$$

Adding $\alpha r z^{(k)}$ and substituting it from the right hand side of (3.2.1) will yield the relation

$$z^{(k)} - z^{(k+1)} = \frac{\alpha s - \alpha r z^{(k)}}{1 - \alpha r} = \frac{s - rz^{(k)}}{\frac{1}{\alpha} - r}, \tag{3.3.2}$$

from which the conditions α should satisfy to guarantee $z^{(k)} > z^{(k+1)}$, can be derived as follows.

$$z^{(k)} - z^{(k+1)} > 0 \;\Rightarrow\; \frac{s - rz^{(k)}}{\frac{1}{\alpha} - r} > 0,$$

thus we have the cases

$$1)\; s - rz^{(k)} > 0 \Rightarrow \frac{1}{\alpha} - r > 0,$$

$$* \; r > 0 \Rightarrow \frac{1}{r} > \alpha > 0,$$

$$* \; r < 0 \Rightarrow \alpha \text{ may be negative,}$$

and

$$2)\ s - rz^{(k)} < 0 \Rightarrow \frac{1}{\alpha} - r < 0,$$

$* \; r > 0 \Rightarrow \alpha$ may take negative values,

$* \; r < 0 \Rightarrow \alpha < 0.$

The use of a negative α may be necessary in those situations where a constant positive steplength does not allow a "steady" convergence towards the solution, in the last iterations. These situations were termed oscillations. In our experiments, we came across with some problems for which the algorithm failed to converge to the solution, although inspection of preliminary iterations showed reduction in the objective function value. However, many factors may justify this loss of robustness, amongst which are 1) the ill-conditioning of the problem matrix in the transformed space, and 2) the use of a constant α when a variable one is more appropriate.

3.3.2 Variable Steplength

The projective algorithm basically works in a simplex. The optimization over a sphere required at each iteration is a device to keep feasibility. However, it is possible to move beyond the sphere boundary as long as the resulting point is inside the simplex. Going beyond the boundaries of the simplex will result in some entry of the new point being negative. Thus, to guarantee the validity of the move, it is enough to assure that the smallest element of $\mathbf{x}^{(k+1)}$ is not less than a certain value β_j which is set arbitrarily small. As in the previous section, we consider the relations

$$\mathbf{x}^{(k+1)} = \frac{D\mathbf{x}'}{\mathbf{e}^T D\mathbf{x}'},$$

and $$\mathbf{x}' = \mathbf{a}_0 - \alpha\mathbf{p}'.$$

Denote γ the steplength that allows $\min_j x_j^{(k+1)} = \beta_j$, i.e.

$$\min_j \left(\frac{D(\mathbf{a}_0 - \gamma_{min}\mathbf{p}')}{\mathbf{e}^T D(\mathbf{a}_0 - \gamma_{min}\mathbf{p}')} \right) = \beta_j,$$

or

$$\left(\frac{D(a_0 - \gamma_{min} p')}{e^T D(a_0 - \gamma_{min} p')}\right) = \beta \ .$$

Thus

$$\min_j D(a_0 - \gamma_{min} p') = \beta_j e^T D(a_0 - \gamma_{min} p'),$$

or

$$Da_0 - \gamma_{min} Dp' = e^T Da_0 \beta \ \ - \gamma_{min} e^T Dp' \beta \ ,$$

and

$$Da_0 - e^T Da_0 \beta \ \ = \gamma_{min}\left[Dp' - e^T Dp' \beta \ \right].$$

Optimum step γ_{min} is the minimum of elements obtained from one by one division $\ominus$ of the components of the vectors in the left hand and right hand sides of the above relation. Thus

$$\gamma_{min} = \min_j \ (Da_0 - \beta e^T Da_0) \ \ominus \ (Dp' - \beta e^T Dp')$$

$$\gamma_{min} = \min_j \frac{\frac{1}{n}\left(x_j^{(k)} - \beta_j\right)}{\left(x_j^{(k)} p'_j - \beta_j \sum_{j=1}^{n} x_j^{(k)} p'_j\right)}.$$

At each iteration β_j is set to a small arbitrary value, γ_{min} is evaluated as above and α set to γ_{min}.

Finally, we describe a method [Lustig, 1985] for choosing a steplength in every iteration based on identifying variables which undergo a substantial change during the progress of the algorithm. It has been observed that, as iterations progress the change in variables that are null at the solution is very large compared to those that are not null. It is thus profitable to speed up the zeroing of the variables that will eventually converge to zero. The ratio test $\beta = \min_j \{1/p'_j, p'_j > 0, j = 1, ..., n\}$, may be used to identify such variables. Lustig (1985), suggests a further test on the identified variable, say x_j, which is $(x_j/(p'_j/x_j)) < \varepsilon$, in order to relate the variables in the space of PC of Chapter 2. If x_j satisfies the condition, α is set to β. Else it is set to $\rho\beta$, where $\rho = .99, \ .95,$ or $.90$. This approach was used in our experiments and seems to work well.

Chapter 4

Duality and Postoptimality Analysis

4.1 Availability of dual solutions

The basic concept of duality is that every linear programming problem (called the primal) has an associated problem, called its dual, such that a solution to it is provided whenever a solution to the original problem is found. Thus, whenever a linear programming problem is solved we actually get the solution to two problems. The primal-dual relationship is important in many respects. It is extensively used in the design of many variants of the simplex and also in the proofs of theoretical results.

One of the early criticisms of the Karmarkar (1984a) algorithm is that it does not generate dual solutions. It was thought to be a primal method only. However, this state of affairs did not last long, as many researchers realized that the computation of the projected gradient in the main step of the algorithm provided, in certain instances, values [Fieldhouse & Tromans, 1985; Lustig, 1985], which converged to the dual optimum solution [Todd & Burrell, 1986; Ye & Kojima, 1987; Gay, 1987].

To see that consider the linear programming problem in standard form and its dual

$$\begin{array}{llll} \text{(P)} & \min \mathbf{c}^T\mathbf{x} & \text{(D)} & \max \mathbf{y}^T\mathbf{b} \\ & \text{s.t. } A\mathbf{x} = \mathbf{b} & & \text{s.t. } \mathbf{y}^T A \le \mathbf{c} \\ & \quad \mathbf{x} \ge 0 & & \end{array} \qquad (4.1.1)$$

The problem in the transformed space is

$$\begin{array}{l} \min \mathbf{c}'^T\mathbf{x}' \\ \text{s.t. } A'\mathbf{x}' = \mathbf{0} \\ \quad \mathbf{e}^T\mathbf{x}' = 1,\ \mathbf{x}' \ge 0, \end{array} \qquad (4.1.2)$$

where $\mathbf{c}' = (\mathbf{c}^T D, -z)$, $A' = (AD, -\mathbf{b})$ and $D = \mathrm{diag}\,(\mathbf{x})$, $\mathbf{x}$ being a feasible point to the primal problem in (4.1.1). Let us write this problem and its dual in a more compact form.

$$\begin{array}{llll} (P') & \min\ \mathbf{c}'^T\mathbf{x}' & (D') & \max\ \mathbf{b}'^T\mathbf{y} \\ & \text{s.t.}\ \ B\mathbf{x}' = \mathbf{b}' & & \text{s.t.}\ \ B^T\mathbf{y} \le \mathbf{c}' \\ & \qquad \mathbf{x}' \ge 0 & & \end{array} \qquad (4.1.3)$$

where $B = \begin{pmatrix} A' \\ \mathbf{e}^T \end{pmatrix}$. If the solution to the primal is nondegenerate, from complementary slackness the dual variables $\mathbf{y}$ satisfy the dual constraints as equalities, i.e.

$$B^T\mathbf{y} = \mathbf{c}' \qquad (4.1.4)$$

Now recall the expression of the projected gradient in Karmarkar's algorithm

$$\mathbf{p} = (I - B^T(BB^T)^{-1}B)D\mathbf{c},$$

and write it as $\mathbf{p} = D\mathbf{c} - B^T\pi$, where $\pi = (BB^T)^{-1}BD\mathbf{c}$. Clearly π is the solution to the system $B^T\pi = \mathbf{c}'$, in the sense of least squares. If B^T is full rank then the solution to this system is the same as that of (4.1.4). Thus, vector π is dual feasible to (D').

Based on this observation Todd and Burrell (1986) designed a dual variant of the Karmarkar algorithm which generates dual solutions and uses them to deal with LP problems whose optimum objective values are not at hand. Ye and Kojima (1987) and Gay (1987) also devised similar dual projective algorithms. In the following these dual variants will be presented and their interrelationship studied. Their role in the use of the Karmarkar algorithm for postoptimality analysis [Salhi & Lindfield, 1988], will also be discussed.

4.2 Extending Karmarkar's Algorithm to Problems with Unknown z*

The variant of Todd and Burrell was aimed at removing the restrictive assumption made in the original Karmarkar's algorithm that the optimum objective value z^* of the problem we wish to solve must be known. The idea is to use the dual variables generated during the course of the algorithm to find ever better lower bounds on z^*. The algorithm works as follows.

Consider again problem (4.1.2) with unknown minimum value z^*. Let z be an estimate of z^* which we update at each iteration by substituting for $\mathbf{c}'$ in (4.1.2), $\mathbf{c}' - z\mathbf{e}$. To choose a good estimate the dual variables can be identified and used to compute the dual objective value which is a lower bound on z^* from duality theory. The dual problem of (4.1.2) is

$$\max\ z \tag{4.2.1}$$

$$\text{s.t.}\quad A'^T\mathbf{y} + \mathbf{e}z \le \mathbf{c}'.$$

Todd and Burrell suggest that z be chosen as

$$z = \min_j \{(\mathbf{c}' - A'^T\mathbf{y})_j\} \tag{4.2.2}$$

This choice is justified because it guarantees dual feasibility of $(\mathbf{y}, z)$. It remains to find $\mathbf{y}$. Such a vector can be found according to the observation made earlier. Explicitly, it is given by

$$\mathbf{y} = (A'A'^T)^{-1}A'\mathbf{c}' \tag{4.2.3}$$

The starting dual vector $(\mathbf{y}^{(0)}, z^{(0)})$ is found as above, assuming that a feasible interior point $\mathbf{x}'^{(0)}$ to the primal is found. To update the dual variables after iteration k, $\mathbf{y}^{(k+1)}$ is computed as the solution to $A'^T\mathbf{y}^{(k+1)} = \mathbf{c}'(z^{(k)})$, where $z^{(k)}$ is the kth estimate to z^*. Let $z^{(k+1)} = \min_j \{(\mathbf{c}' - A'^T\mathbf{y}^{(k+1)})_j\}$. If $z^{(k+1)} \le z^{(k)}$, then no improvement to the lower bound on z^* is made. Thus $z^{(k)}$ is kept as next estimate, i.e. $z^{(k+1)} = z^{(k)}$. If, on the other hand, $z^{(k+1)} > z^{(k)}$, then $z^{(k+1)}$ is the new estimate and vector $\mathbf{y}^{(k+1)}$ is recomputed as the solution to the linear system $A'^T\mathbf{y}^{(k+1)} = \mathbf{c}'(z^{(k+1)})$.

For computational purposes, the projection matrix $P_B = I - B^T(BB^T)^{-1}B$ can be written equivalently as $P_B = PP_{A'}$, where $P = I - \mathbf{e}\mathbf{e}^T/n$. This is valid since $A'\mathbf{e} = 0$. The advantage of this form of the projection matrix is that matrix B with a full row of ones is not used. The original sparsity of A, if there is any, is kept except for the extra column due to $\mathbf{b}$, introduced when forming A'. The projection vector, therefore, is given by

$$\mathbf{p} = P_B\mathbf{c}'(z^{(k+1)}) = PP_{A'}\mathbf{c}'(z^{(k+1)}) = P[\ \mathbf{c}'(z^{(k+1)}) - A'^T\mathbf{y}^{(k+1)})].$$

Todd and Burrell (1986) showed that, assuming A' a mxn-matrix has rank m, $(\mathbf{y}, z)$ is feasible for problem (4.2.1). Along the direction $-\mathbf{p} / \|\mathbf{p}\|$, a constant reduction of 1/5 in the potential function of Karmarkar described in Chapter 2, is guaranteed for $\alpha = 1/3$. Thus an algorithm with polynomial complexity can be built. Although, Todd & Burrell showed that their algorithm retains polynomial complexity, its original form does not seem to be efficient in practice. The choice of steplength $\alpha = 1/3$, is only aimed at achieving polynomial complexity; in practice, values close to 1 are more suitable for reducing the number of iterations. Also, after some iterations, entries of $\mathbf{x}^{(k)}$ get close to zero, especially in degenerate cases. As $D = \text{diag}\ (\mathbf{x}^{(k)})$, $A' = AD$ becomes ill-conditioned, and may be rank deficient. In this case, the dual vector $(\mathbf{y}, z)$ may not be feasible for problem (4.2.1). The search direction $\mathbf{p}$ is, thus, not good enough, which prevents the convergence of the algorithm, as a consequence.

The variant of Gay (1987) is similar to that of Todd and Burrell. However, some differences may be pointed out. For instance, no assumption is made regarding the rank of A', in Gay's method. Also, the way the bound on the optimum objective value is updated, is different from that of Todd and Burrell's algorithm.

Consider the problem (4.1.1), where (P) has a nonempty and bounded feasible region, i.e. it has at least one solution $\mathbf{x}^*$. Denote the optimum objective value of (P), $z^* = -\mathbf{c}^T\mathbf{x}^*$ and define $\mathbf{u}$ and $\mathbf{v}$ by

$$\mathbf{u} = P_{A'}\begin{bmatrix} Dc \\ 0 \end{bmatrix}, \quad \text{and} \quad \mathbf{v} = P_{A'}\begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

Let $\qquad d_j(z) = (\mathbf{u} + z\mathbf{v})_j$

and $\qquad d(z) = \min_j \{d_j \text{ for } j = 1, 2, ..., n+1\}$.

Based on an observation by Anstreicher (1986b), that after rescaling the feasible region of (P) using Karmarkar's projective transformation, d(z) can be written as

$$d(z) = \min((\mathbf{u} + z\mathbf{v})^T\mathbf{x}', \mathbf{x}' \geq 0 \text{ and } \mathbf{e}^T\mathbf{x}' = 1)$$

and for any $\mathbf{x}'$ such that $A'\mathbf{x}' = 0$ we have

$$(\mathbf{u} + z\mathbf{v})^T\mathbf{x}' = \begin{bmatrix} Dc \\ z \end{bmatrix}^T \mathbf{x}'. \tag{4.2.4}$$

As $\mathbf{x}' \geq 0$, $d(z)$ increases monotonically with z. The right-hand side of (4.2.4) being precisely the objective function of (4.1.2) which is zero at the optimum, i.e.

$$\begin{bmatrix} Dc \\ z^* \end{bmatrix}^T \mathbf{x}'^* = 0,$$

then $d(z^*) \leq 0$, (at the optimum). From this observation the adjustment of z can be done as follows.

If for the current $z^{(k)}$ we have $d(z^{(k)}) \leq 0$, then no change is made to the value of $z^{(k)}$. However, if $d(z^{(k)}) > 0$ then we can find $z^{(k+1)}$ such

$$z^{(k+1)} \leq z^{(k)} \text{ and } d(z^{(k+1)}) = 0.$$

The value is $z^{(k+1)} = z^{(k)} + \delta z^{(k)}$, where $\delta z^{(k)} = \max_j \{-d_j(z)/v_j$, for all j such that $v_j > 0\}$. Accordingly to the updating of the estimate $z^{(k)}$, the dual solutions at iteration k+1 are updated as follows.

$$\mathbf{y}^{(k+1)} = -(A'^T)^{\dagger}\mathbf{c}'(z^{(k+1)}), \text{ when } z^{(k+1)} < z^{(k)}$$

where $(A'^T)^{\dagger}$ is the pseudoinverse of A'^T and $\mathbf{c}'(z^{(k+1)}) = [Dc, z^{(k+1)}]$. If, on the other hand, $z^{(k+1)} = z^{(k)}$, then $\mathbf{y}^{(k+1)} = \mathbf{y}^{(k)}$, i.e. no updating is necessary. The use of the pseudoinverse allows us to find $\mathbf{y}$ of minimum length regardless of the column rank of A'^T. The solution vector $\mathbf{y}$ is unique when A'^T is full rank. Note that this constitutes the main difference between the variant of Gay and that of Todd and Burrell.

Now, we may describe an algorithm which is basically that of Todd and Burrell augmented with Gay's approach to updating the dual variables.

Algorithm 4.1

Assume we have the current approximation $\mathbf{x}^{(k)}$ to the primal solution, and $(\mathbf{y}^{(k)}, z^{(k)})$ to the dual solution, and ε an arbitrary small value, then

```
1- Compute $\mathbf{u}$, $\mathbf{v}$ and $d(z^{(k)}) = \min_j \{(\mathbf{u} + z^{(k)}\mathbf{v})_j$ for $j = 1, 2, ..., n+1\}$
2- **if** $d(z^{(k)}) \leq 0$ **then**
```

```
        $y^{(k+1)} = y^{(k)}$
        $z^{(k+1)} = z^{(k)}$
        go to 4-
    else
        find $z^{(k+1)}$ such that $z^{(k+1)} > z^{(k)}$ and $d(z^{(k)}) = 0$.
    endif.
3- Compute $y^{(k+1)}$ from
        $y^{(k+1)} = -(A'^T)^\dagger c'(z^{(k+1)})$,
    where
        $c'(z^{(k+1)}) = c' - z^{(k+1)}e$.
4- Compute p from
        $p^{(k)} = -P(u + z^{(k)}v)$
5- Compute
        $x'^{(k+1)} = x^{(k)} + \alpha\, p^{(k)} / \|p^{(k)}\|$.
6- Compute
        $x^{(k+1)} = x'^{(k+1)} / e^T x'^{(k+1)}$.
7- If $\mathrm{abs}(c^T x^{(k+1)} - z^{(k+1)}) < \varepsilon$ then stop.
    else $k = k+1$ go to 1-, endif.
```

In step 5-, the suggested value of 1/3 by Todd and Burrell does not seem to be appropriate as it causes slow convergence in practice. They also suggested the use of a linesearch of the potential function. It is not clear how this can be done and no numerical experience is provided which supports this claim. On the other hand, the blocking technique proposed by Lustig (1985) and given at the end of Chapter 3, may be an alternative. In the following numerical results, this technique has been used.

4.2.1 Computational Experience

We implemented Algorithm 4.1 and solved a set of ill-conditioned LP problems whose constraints are based on a submatrix of the Hilbert matrix. The problems have also been considered by Roos (1985). They will be described in described in Chapter 7 where further experiments are carried on them. The primal solution to these problems is $\mathbf{x}^* = (1, 1, ..., 1)^T$ and its dual $\mathbf{y}^* = (2, 1, ..., 1)^T$, as specified in the following tables under the heading "True Solution".

Todd-Burrell-Gay's Variant		H01ADF	True Solution	
Dual	Primal	Primal	Dual	Primal
2.0172	0.9993	1.0000	2.0	1.0
0.8023	1.0847	1.0000	1.0	1.0
1.6723	0.7115	1.0000	1.0	1.0
0.1226	1.3733	1.0000	1.0	1.0
1.3880	0.8327	1.0000	1.0	1.0
Iterations:	8	7		

Table 4.1 5x5 Hilbert Type LP Problem Solved With Todd, Burrell and Gay's Variant

Todd-Burrell-Gay's Variant		H01ADF	True Solution	
Dual	Primal	Primal	Dual	Primal
2.0102	0.9971	1.0000	2.0	1.0
0.9039	1.0336	1.0000	1.0	1.0
1.2117	0.8991	1.0000	1.0	1.0
0.9741	1.0656	0.9997	1.0	1.0
0.9035	1.0651	1.0013	1.0	1.0
0.8219	0.9997	0.9967	1.0	1.0
0.8591	0.9501	1.0050	1.0	1.0
1.0810	0.9423	0.9955	1.0	1.0
1.9049	0.9828	1.0022	1.0	1.0
0.3273	1.0655	0.9995	1.0	1.0
Iterations:	9	22		

Table 4.2 10x10 Hilbert Type LP Problem Solved With Todd, Burrell and Gay's Variant

Todd-Burrell-Gay's Variant		H01ADF	True Solution	
Dual	Primal	Primal	Dual	Primal
1.9932	0.9985	1.0000	2.0	1.0
1.0732	1.0174	1.0000	1.0	1.0
0.8047	0.9507	1.0000	1.0	1.0
1.0610	1.0210	0.9949	1.0	1.0
1.2038	1.0419	1.0352	1.0	1.0
1.0080	1.0094	0.8433	1.0	1.0
0.9071	0.9819	1.4791	1.0	1.0
0.9261	0.9752	0.0000	1.0	1.0
0.9547	0.9814	2.3625	1.0	1.0
0.9747	0.9906	0.0000	1.0	1.0
0.9909	0.9979	0.9117	1.0	1.0
1.0061	1.0031	1.9910	1.0	1.0
1.0217	1.0071	0.0000	1.0	1.0
1.0360	1.0112	1.4577	1.0	1.0
1.0476	1.0151	0.9156	1.0	1.0
1.0524	1.0177	0.0000	1.0	1.0
1.0461	1.0167	0.0000	1.0	1.0
1.0233	1.0092	0.0000	1.0	1.0
0.9756	0.9919	0.0000	1.0	1.0
0.8922	0.9616	0.0000	1.0	1.0
Iterations:	10	40		

Table 4.3 20x20 Hilbert Type LP Problem Solved With Todd, Burrell and Gay's Variant

From the results of Table 4.1 to 4.3, our implementation of the dual Karmarkar Algorithm 4.1 seems to return more accurate approximate primal solutions than the Nag simplex based subroutine H01ADF when the problem is large. These results also compare favourably with those reported by Roos [1985]. They are relatively more accurate and obtained in less iterations. In addition, the dual solutions are provided.

4.3 Dual Algorithm of Ye and Kojima

The dual Karmarkar algorithm of Ye and Kojima (1987) is similar to the version of Gay (1987). The algorithm is applicable under the same assumptions, i.e. the feasible region is nonempty and bounded and the primal solution is non-negative. However, there are subtle differences in the way bounds on the objective function value are found in Ye and Kojima's variant. Their approach is developed on the dual of the original problem, i.e. (D) rather than (D') and seems to lead to better bounds. In the following we shall describe their method and suggest a practical procedure for finding bounds on z*. We shall also show that the updating of the bounds is equivalent to a one-dimensional LP problem.

Consider the pair of primal and dual LP problems (4.1.1) and the vector

$$\mathbf{y}(z)^T = (A'^T)^{\dagger}\mathbf{c}'(z) \tag{4.3.1}$$

which is similar to that given by (4.2.3). This can be expressed as

$$\mathbf{y}(z)^T = \mathbf{y}_2^T + z\mathbf{y}_1^T, \tag{4.3.2}$$

where

$$\mathbf{y}_1^T = (A'^T)^{\dagger}(\mathbf{0}, -1)^T$$

$$\mathbf{y}_2^T = (A'^T)^{\dagger}(D\mathbf{c}, 0)^T.$$

The vector $\mathbf{y}(z)$ is dual feasible if $A^T\mathbf{y}(z) \le \mathbf{c}^T$, which represents the constraints of the dual problem (D). Under the assumption that $\mathbf{x}$ is positive, D is positive definite. Thus $DA^T\mathbf{y}(z) \le D\mathbf{c}^T$. If $\mathbf{y}(z)$ also satisfies $z \le \mathbf{b}^T\mathbf{y}(z)$, then

$$\begin{pmatrix} A^T\mathbf{y}(z) \le \mathbf{c}^T \\ z \le \mathbf{b}^T\mathbf{y}(z) \end{pmatrix} \Leftrightarrow A'^T\mathbf{y}(z) \le \mathbf{c}'^T(z). \tag{4.3.3}$$

From equivalence (4.3.4), $\mathbf{y}(z)$ is dual feasible and $\mathbf{b}^T\mathbf{y}(z)$ is a lower bound for the minimal objective value of the original problem, but no less than the current objective value z at point $\mathbf{y}(z)$.

Assume that iteration k has been completed. To find $\mathbf{y}^{(k+1)}$ and $z^{(k+1)}$, it is crucial to find the best z, say z', that guarantees

$$\mathbf{c}'^T(z) - A'^T\mathbf{y}(z) \ge 0, \tag{4.3.4}$$

which can be written using expression (4.3.2) of $\mathbf{y}(z)$ as

$$\mathbf{c}'^T(z) - A'^T\mathbf{y}_2 - zA'^T\mathbf{y}_1 \ge 0,$$

or
$$(\mathbf{c}^T D, -z) - A'^T\mathbf{y}_2 - zA'^T\mathbf{y}_1 \ge 0.$$

By splitting the vector between brackets into the sum of two vectors, the above inequality can be written as

$$(\mathbf{c}^T D, 0) - A'^T \mathbf{y}_2 + z[(\mathbf{0}, -1) - A'^T \mathbf{y}_1] \geq 0,$$

which can be expressed in the form $\mathbf{s} + z\mathbf{r} \geq 0$, where

$$\mathbf{s} = \begin{pmatrix} D\mathbf{c} \\ 0 \end{pmatrix} - \begin{pmatrix} DA^T \\ -\mathbf{b}^T \end{pmatrix} \mathbf{y}_2$$

and

$$\mathbf{r} = \begin{pmatrix} 0 \\ -1 \end{pmatrix} - \begin{pmatrix} DA^T \\ -\mathbf{b}^T \end{pmatrix} \mathbf{y}_1.$$

z' is the supremum of the set of ratios $-\mathbf{s} / \mathbf{r}$, such that $\mathbf{s} + z\mathbf{r} \geq 0$. More explicitly, z' is chosen among elements of the set

$$Z = \left\{ \left(- \frac{\left[\mathbf{c} - A^T \mathbf{y}_2\right]_j}{\left[-A^T \mathbf{y}_1\right]_j} \right)_{j = 1, \ldots, n-1}, \ - \frac{\mathbf{b}^T \mathbf{y}_2}{-1 + \mathbf{b}^T \mathbf{y}_1} \right\}.$$

In Ye and Kojima (1987), this set was not explicitly given. They suggested that a one-dimensional search over Z, to find a good lower bound z' on the optimum objective z*, may be used. However, the procedure which is crucial for the convergence of the algorithm, was not clearly stated. We developed such a procedure and it is described below.

4.3.1 Improved Lower Bound on z*

The elements of Z divide the one-dimensional space R into half-spaces, in the intersection of which may be found a value z' such that $z' \leq z^*$. Obviously this intersection may be empty ; in this case no value z' satisfying all the constraints $\mathbf{s}+\mathbf{r}z \geq 0$ exists. The half-spaces may be represented as follows (assuming that $\mathbf{r}(j) \neq 0$, for j = 1, ..., n).

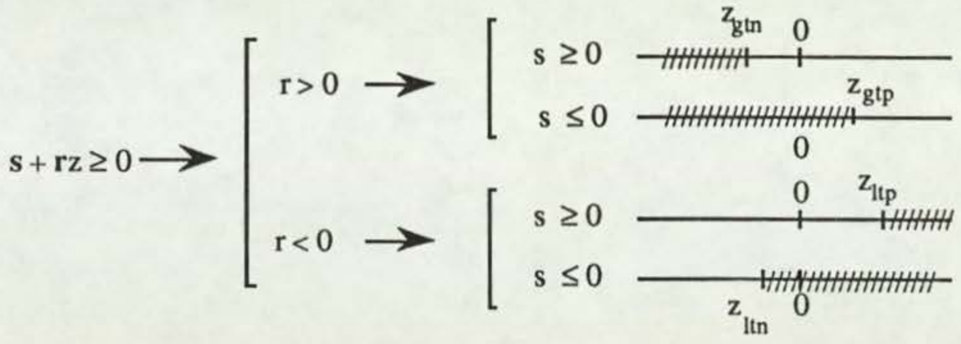


Fig 4.1 The Half-Spaces Defined by the Ratios s(j)/r(j)

(r and s are scalar entries of **r** and s)

The above diagram shows the different cases that may arise. When all four case are present it is likely that their intersection is empty, i.e. a value z' that bounds z* is not provided by the ratios. However, this seldom happens. On the other hand, when their intersection is a non-empty interval, its upper bound is the value z' we are looking for. The ratios z_{gtn}, z_{gtp}, z_{ltn} and z_{ltp} are determined by the non-redundant inequalities among $\mathbf{s} + \mathbf{r}z \geq 0$. By non-redundant inequality we mean the one that gives for each case the best ratio. For instance if ratio1 > ratio2 > ratio3... then ratio1 is taken as z_{gtn} and the inequality that produced it is non-redundant. Explicit procedures for determining z' are given in Appendix E.

Now, we can look at the problem from a different point of view. We need to find the upper bound (maximum element) of a one-dimensional interval defined by a set of constraints $\mathbf{s} + \mathbf{r}z \geq 0$. This, obviously, is a one-dimensional linear programming problem which can be formulated as

$$\max z$$

$$\text{s.t. } \mathbf{s} + \mathbf{r}z \geq 0.$$

Todd (1988a) also mentions this approach but considers a two-dimensional linear programming problem. It is not clear how advantageous (or otherwise) it may be if updating the estimate z' was carried out by solving a one-dimensional LP problem. In our experiments (Chapter 6 and 7) the technique we have described was used.

Algorithm 4.2: Algorithm of Ye and Kojima

Initially, set $\mathbf{y}^{(0)} = 0$, $\mathbf{x}^{(0)}$ primal feasible and $z^{(0)}$ a lower bound on the optimum objective value. An arbitrary small value ε is pre-set. At iteration k do:

```
0- if $(\mathbf{c}^T\mathbf{x}^{(k+1)} - z^{(k+1)}) \le \varepsilon$ then stop, Optimum solution obtained; else go to 1-.
1- Set $D = \mathrm{diag}(\mathbf{x}^{(k)})$.
2- Set  $\mathbf{y}_1^T = (A'^T)^\dagger[\mathbf{0}, -1]$,
        $\mathbf{y}_2^T = (A'^T)^\dagger[\mathbf{c}D, 0]$,
        $\mathbf{c}'(z) = [\mathbf{c}D, -z]$,
        $\mathbf{y}(z) = \mathbf{y}_2 + z\mathbf{y}_1$.
3- Find z' such that $z' = \sup \mathbf{Z}$,
   where $\mathbf{Z} = \{ z \mid \mathbf{c}'(z) - \mathbf{y}(z)A' \ge 0 \}$.
4- if $z^{(k)} < \mathbf{y}(z')\mathbf{b}$ then
     $\mathbf{y}^{(k+1)} = \mathbf{y}(z')$,
     $z^{(k+1)} = \mathbf{y}(z')\mathbf{b}$.
   else
     $\mathbf{y}^{(k+1)} = \mathbf{y}^{(k)}$,
     $z^{(k+1)} = z^{(k)}$.
   endif
   $\mathbf{p}(z^{(k+1)}) = \left(\mathbf{c}'(z^{(k+1)}) - \mathbf{y}(z^{(k+1)})A'\right)^T - \frac{\mathbf{c}^T\mathbf{x}^{(k)} - z^{(k+1)}}{n+1}\mathbf{e}$.
5- Find point $\mathbf{x}'^{(k+1)}$ in the transformed space as follows
     $\mathbf{x}'^{(k+1)} = \mathbf{e} - \beta\frac{\mathbf{p}(z^{(k+1)})}{\|\mathbf{p}(z^{(k+1)})\|}$,
   where $\beta \in [0.27, 0.36]$.
6- Transform back to original space
```

$$x^{(k+1)} = \frac{Dx'^{(k+1)}}{x'^{(k+1)}_{n+1}}.$$

7- k = k+1, repeat from 0-

4.3.2 Experiments with Algorithm 4.2

The same problems solved earlier with Algorithm 4.1 were solved with Ye and Kojima's algorithm. The results recorded in the following tables show some improvements over those obtained with Todd, Burrel and Gay's variant; they are closer to the true solution, especially on the smaller problems. They also compare favourably with the results returned by the Nag subroutine H01ADF.

Algorithm 4.1 and Algorithm 4.2 were coded in Fortran 77 and run on a VAX 8650, in single precision.

Ye-Kojima's Variant		H01ADF	True Solution	
Dual	Primal	Primal	Dual	Primal
1.9984	1.0000	1.0000	2.0	1.0
1.0127	1.0027	1.0000	1.0	1.0
0.9674	0.9961	1.0000	1.0	1.0
1.0330	1.0082	1.0000	1.0	1.0
0.9885	0.9953	1.0000	1.0	1.0
Iterations:	8	7		

Table 4.4 5x5 Hilbert Type LP Problem Solved With Ye and Kojima's Variant

Ye-Kojima's Variant		H01ADF	True Solution	
Dual	Primal	Primal	Dual	Primal
1.9891	1.0005	1.0000	2.0	1.0
1.0450	1.0045	1.0000	1.0	1.0
1.0454	0.9601	1.0000	1.0	1.0
0.8415	1.0519	0.9997	1.0	1.0
0.8959	1.0314	1.0013	1.0	1.0
1.0310	0.9787	0.9967	1.0	1.0
1.1748	0.9574	1.0050	1.0	1.0
1.2173	0.9726	0.9955	1.0	1.0
1.0130	1.0058	1.0022	1.0	1.0
0.7402	1.0382	0.9995	1.0	1.0
Iterations:	9	22		

Table 4.5 10x10 Hilbert Type LP Problem Solved With Ye and Kojima's Variant

Ye-Kojima's Variant		H01ADF	True Solution	
Dual	Primal	Primal	Dual	Primal
1.9917	0.9604	1.0000	2.0	1.0
1.0668	1.3118	1.0000	1.0	1.0
0.8852	0.4570	1.0000	1.0	1.0
0.9763	0.8338	0.9949	1.0	1.0
1.0487	1.2791	1.0352	1.0	1.0
1.0707	1.3251	0.8433	1.0	1.0
1.0560	1.1605	1.4791	1.0	1.0
1.0239	1.0034	0.0000	1.0	1.0
0.9885	0.9086	2.3625	1.0	1.0
0.9578	0.8672	0.0000	1.0	1.0
0.9360	0.8622	0.9117	1.0	1.0
0.9255	0.8801	1.9910	1.0	1.0
0.9279	0.9108	0.0000	1.0	1.0
0.9447	0.9470	1.4577	1.0	1.0
0.9788	0.9835	0.9156	1.0	1.0
1.0341	1.0165	0.0000	1.0	1.0
1.1155	1.0436	0.0000	1.0	1.0
1.2236	1.0636	0.0000	1.0	1.0
1.3046	1.0759	0.0000	1.0	1.0
0.5422	1.0807	0.0000	1.0	1.0
Iterations:	9	40		

Table 4.6 20x20 Hilbert Type LP Problem Solved With Ye and Kojima's Variant

4.4 Postoptimality Analysis via Karmarkar's Algorithm: Introduction

Postoptimality analysis is concerned with the sensitivity of the solution of a problem to changes in the original data of the problem. The whole idea can formally be presented as follows:

Given a problem and its solution obtained using some algorithm, a crucial question would be: How stable the solution set is to perturbations in the data of the problem? Such question is most relevant when linear programming is extensively used in real world applications, as was expressed by Gal (1979):

> "An objection frequently heard to more extensive dissemination of the theories of Linear Programming in the practical field has been that the data which are available in practice are at once too inexact and too unreliable to provide the basis for the application of 'exact' procedures like Linear Programming."

Before going into the details of efficient updating of the solution when changes occur in some component of the problem, it is important to know whether updating is necessary. This is equivalent to checking the optimality of the solution of the original problem to the modified one. In this respect it may be useful to briefly recall how this question is dealt with in the simplex method.

After solving a linear programming problem using the simplex, the last tableau holds vital information about the problem such as primal and dual solutions, basis inverse and primal-dual relationships in general. When a change occurs in some parameter of the problem, it is possible to revise the tableau taking account of the change, then apply the optimality test of the simplex (dual simplex) to decide whether the solution is still optimal or updating it is necessary. If c_j were changed to c'_j in a maximization problem for instance, it is enough to compute $\delta c'_j$, the corresponding reduced cost. If $\delta c'_j \geq 0$, then the solution at hand is still optimal. Otherwise updating it is necessary and can be done by carrying the simplex iterative process from the revised last tableau.

When using Karmarkar's algorithm it is hard to check whether the solution of the original problem remains optimal after a change has occurred in some parameter without making the assumption that z'* of the modified problem is at hand. This is due to the fact that the optimality criterion of the Karmarkar algorithm is based on the gap between the primal and dual objective values. The approach of the simplex is not readily usable precisely because its optimality test is different from that of Karmarkar's method. However, a primal-dual relationship, known as the complementary slackness conditions, can be used to check the optimality of a point for a LP problem regardless of the method used to obtain it. The complementary slackness conditions can be formulated as follows [Chvátal, 1983, p.63].

Consider the pair of primal and dual LP probems in the simple symmetric form:

$$\begin{array}{llll} \text{Primal:} & \text{Max } \mathbf{c}^T\mathbf{x} & \text{Dual:} & \text{Min } \mathbf{b}^T\mathbf{y} \\ & \text{s.t. } A\mathbf{x} \le \mathbf{b} & & \text{s.t. } \mathbf{y}^T A \ge \mathbf{c} \\ & \mathbf{x} \ge 0, & & \mathbf{y} \ge 0. \end{array} \qquad (4.4.1)$$

A feasible solution $x_1^*, x_2^*, ..., x_n^*$ of the primal in (4.1.1) is optimal if and only if there are numbers $y_1^*, y_2^*, ..., y_m^*$ such that

$$\begin{cases} \sum_{i=1}^{m} a_{ij} y_i^* = c_j, \text{ whenever } x_j^* > 0 \\ y_i^* = 0, \text{ whenever } \sum_{j=1}^{n} a_{ij} x_j^* < b_i \end{cases} \qquad (4.4.2)$$

and such that

$$\begin{cases} \sum_{i=1}^{m} a_{ij} y_i^* \ge c_j, \text{ for all } j = 1, 2, \ldots, n \\ y_i^* \ge 0, \text{ for all } i = 1, 2, \ldots, m. \end{cases} \qquad (4.4.3)$$

In the following we shall present two ways for finding whether the solution point $\mathbf{x}^*$ remains optimum after a modification has occured in some parameter of the original problem. The first approach makes the assumption that z'^* of the modified problem is known, while the second one does not make this assumption and is based on the complementary slackness conditions. Postoptimal analysis will be studied when discrete changes occur in the entries of the cost vector, the right-hand side and the rim of the problem. We shall see how the updating process can be carried out, when necessary, using Karmarkar algorithm, possibly without considering the perturbed problem as a completely new one.

4.4.1 Perturbations in the Cost Vector

If a change δc_j occurs in entry c_j of $\mathbf{c}$ then the new entry c'_j is written as $c'_j = c_j + \delta c_j$. The solved LP problem in S form can be written as

$$\begin{aligned} S': \quad & \min \ \mathbf{c}^T\mathbf{x} - z^* \\ & \text{s.t.} \ \ A\mathbf{x} - \mathbf{b} = 0, \\ & \qquad (\mathbf{x}, 1) \geq 0. \end{aligned}$$

Its solution is

$$(\mathbf{x}^{*T}, 1), \text{ and } \mathbf{c}^T\mathbf{x}^* - z^* = 0.$$

It is clear that changes in $\mathbf{c}$ do not affect the feasibility of $\mathbf{x}^*$. However, it may not be optimal. Knowing that in S' the target objective value is zero, two cases may arise in respect with optimality of $\mathbf{x}^*$.

Assuming that ε is pre-set to some arbitrary small value, then

case 1: If $\mathbf{c}'^T\mathbf{x}^* - z'^* \leq \varepsilon$, then $\mathbf{x}^*$ is optimal solution to the modified problem.

case 2: If $\mathbf{c}'^T\mathbf{x}^* - z'^* > \varepsilon$, then $\mathbf{x}^*$ is feasible but not optimal. To update the solution of the modified problem we carry on the Karmarkar solving process from the state at which $\mathbf{x}^*$ is a solution to the unmodified problem.

This approach is not realistic because of the assumption that z'^*, the optimum objective value of the modified problem, is at hand. A better approach may be based on the complementary slackness conditions given earlier. Let $\mathbf{x}^*$ be the solution of the primal problem in (4.4.1). To decide whether $\mathbf{x}^*$ is still optimal after c_j becomes c'_j, the system (4.4.2) is set up and solved. If it admits a solution $\mathbf{y}^*$ satisfying (4.4.3) then $\mathbf{x}^*$ is still optimum. Otherwise it is necessary to update it. This can be done as in case 2 above.

Let us show through an example how the sensitivity of the solution to changes in the cost vector may be studied.

<u>Example 4.1</u>

Max $z = 2x_1 + 3x_2 + x_3$

s.t. $x_1 + x_2 + x_3 \leq 3$

$x_1 + 4x_2 + 7x_3 \leq 9$

$x_1 \geq 0, x_2 \geq 0, x_3 \geq 0,$

whose solution is $\mathbf{x}^* = (1, 2, 0)$ and $z^* = 8$.

Assume that cost c_3 becomes $c'_3 = 3$. Is $\mathbf{x}^*$ still optimal? To find out we set up the system (4.4.2), i.e.

$$y_1^* + y_2^* = 2$$

$$y_1^* + 4\,y_2^* = 3.$$

The solution to this system is $\mathbf{y}^* = (5/3, 1/3)$, which satisfies the conditions (4.4.3), especially $y_1^* + 7y_2^* = 4 \geq c'_3 = 3$. Thus $\mathbf{x}^*$ remains optimal solution.

Now, an interesting thing to know is how far c_3 can be increased without affecting the optimality of $\mathbf{x}^*$. To find out, it suffices to consider c_3 as a variable in the condition $y_1^* + 7y_2^* \geq c_3$, which leads to $c_3 \leq 4$. Thus, any value of $c_3 \leq 4$ will not affect the optimality of $\mathbf{x}^*$ and the optimum objective value, as $x_3 = 0$. If, on the other hand, $c'_3 = 7$ for instance, $\mathbf{x}^*$ is no longer optimum solution as conditions (4.4.3) are not satisfied. However, the point remains feasible and can be used as a starting point for Karmarkar's algorithm. The new optimum is $\mathbf{x}'^* = (2, 0, 1)$ and $z'^* = 11$.

Let us now consider a change in an entry of **c** corresponding to a basic optimum solution, say c_2. In this case system (4.4.2) will read

$$y_1^* + y_2^* = c_2$$

$$y_1^* + 4\,y_2^* = 3.$$

Using Cramer's rule to solve the above system, we obtain $y_1^* = 4/3c_2 - 1$ and $y_2^* = 1 - c_2/3$. For $\mathbf{x}^*$ to remain optimum, the values y_1^* and y_2^* should satisfy conditions (4.4.3). These conditions are

$$4/3c_2 - 1 \geq 0$$

$$1 - c_2/3 \geq 0,$$

from which the range [3/4, 3] of c_2 is derived.

4.4.2 Perturbations in the Right-Hand Side

Consider the linear programming problem

$$S: \qquad \max\ \mathbf{c}^T\mathbf{x}$$
$$\text{s.t.}\ \ A\mathbf{x} = \mathbf{b},$$
$$\mathbf{x} \geq 0,$$

and let J be an optimal basis for S. Let $\delta\mathbf{b}$ be a perturbation of $\mathbf{b}$ and assume that $\delta\mathbf{b}$ is small enough so that

$$(A_J)^{-1}(\mathbf{b} + \delta\mathbf{b}) \geq 0.$$

This means that J is also an optimal basis for the perturbed problem

$$S\delta: \qquad \max\ \mathbf{c}^T\mathbf{x}$$
$$\text{s.t.}\ \ A\mathbf{x} = \mathbf{b} + \delta\mathbf{b},$$
$$\mathbf{x} \geq 0.$$

The optimal basic solution of $S\delta$ is

$$\mathbf{x}_J = (A_J)^{-1}\mathbf{b} + (A_J)^{-1}\delta\mathbf{b}$$

$$x_j = 0, \qquad j \notin J$$

$$z^* = \mathbf{b}^T\mathbf{y}^* + \delta\mathbf{b}^T\mathbf{y}^*,$$

where $\mathbf{y}^*$ is the dual optimal solution. Thus the variation of the optimal value of the objective function of problem (S), for a variation $\delta\mathbf{b}$ of the right-hand side $\mathbf{b}$ small enough for the optimal basis to remain the same, is $\delta\mathbf{b}^T\mathbf{y}^*$.

If a change δb_i occurs in entry b_i of $\mathbf{b}$ then the new entry, say b'_i is written as $b'_i = b_i + \delta b_i$. The solved LP problem in S form can be written as S' of the previous section, whose optimum primal and dual solutions respectively are $\mathbf{x}^*$ and $\mathbf{y}^*$. The modified problem with new RHS $\mathbf{b}'$ admits $\mathbf{y}^*$ as a feasible solution. Concerning its optimality as for the perturbations in $\mathbf{c}$, two cases arise.

Assume that ε is pre-set to some arbitrarily small value, then

case 1: If $\mathbf{b}'^T\mathbf{y}^* - z'^* \leq \varepsilon$, then $\mathbf{y}^*$ is dual optimal solution to the modified problem.

case 2: If $\mathbf{b}'^T\mathbf{y}^* - z'^* > \varepsilon$, $\mathbf{y}^*$ is dual feasible but not optimal. To update the solution of the modified problem we carry on the Karmarkar algorithm from the state at which $\mathbf{y}^*$ is solution to the unmodified problem.

Here again, the assumption that z'^* of the modified problem is available is not realistic. We drop this assumption and adopt the same approach taken to study the sensitivity of the solution to changes in the cost vector. When a change occurs in the RHS of the original problem the dual solution returned by the dual Karmarkar algorithm remains feasible. Its optimality, however, is not guaranteed.

Consider again Example 4.1 in which $b_1 = 3$ is changed to $b'_1 = 4$. The dual of the modified form can be written as

$$\begin{aligned}
&\text{Max} - 4y_1 - 9y_2 \\
&\text{s.t.} \quad -y_1 - y_2 \leq -2 \\
&\qquad\;\; -y_1 - 4y_2 \leq -3 \\
&\qquad\;\; -y_1 - 7y_2 \leq -1 \\
&\qquad\;\; y_1 \geq 0,\ y_2 \geq 0.
\end{aligned}$$

The system (4.4.2) of the complementary slackness conditions for the dual will read

$$- x^*_1 - x^*_2 - x^*_3 = -4$$
$$- x^*_1 - 4x^*_2 - 7x^*_3 = -9$$
$$x^*_3 = 0$$

or equivalently

$$x^*_1 + x^*_2 = 4$$
$$x^*_1 + 4x^*_2 = 9$$
$$x^*_3 = 0.$$

The solution to the system is $\mathbf{x}^* = (7/3, 5/3, 0)$ which satisfies conditions (4.4.3). Thus the dual solution remains optimal.

Sensitivity analysis to changes in the RHS can be studied as for the cost vector, but working with the dual. When the dual solution is no longer optimal, it can be used to start the Karmarkar algorithm and find the new optimum solution.

4.4.3 Perturbations in the Rim

We consider simultaneous changes in both the cost vector and the right-hand side of the LP problem already solved. This is a complex case that can be shown to be tractable but under a restrictive assumption. W assume that z'^*, the optimum objective value of the modified problem, is available. The rim of the modified problem is $\mathbf{c}' = \mathbf{c} + \delta\mathbf{c}$ and $\mathbf{b}' = \mathbf{b} + \delta\mathbf{b}$. Three cases arise.

case 1: If $\mathbf{c}'^T\mathbf{x}^* - z'^* \leq \varepsilon$, then $\mathbf{x}^*$ is primal optimal solution to the modified problem.

case 2: If $\mathbf{b}'^T\mathbf{y}^* - z'^* \leq \varepsilon$, then $\mathbf{y}^*$ is dual optimal solution to the modified problem.

case 3: If $\mathbf{c}'^T\mathbf{x}^* - z'^* > \varepsilon$ and $\mathbf{b}'^T\mathbf{y}^* - z'^* > \varepsilon$, then $\mathbf{x}^*$ and $\mathbf{y}^*$ are not optimal and may not be feasible for the modified problem.

The problem has the form

$$\min \ \mathbf{c}'^T\mathbf{x}' - z'^*$$
$$\text{s.t.} \quad A\mathbf{x}' - \mathbf{b}' = 0,$$

$(\mathbf{x}', 1) \geq 0.$

If a feasible point is available then the projective algorithm may be carried out from Phase II. Otherwise we can get a feasible point using the primal optimal solution already at hand, i.e. $\mathbf{x}^*$.

The system of constraints of above problem may be represented as follows.

$$[A_B,\ A_s]\begin{bmatrix}\mathbf{x}'_B \\ \mathbf{x}'_s\end{bmatrix} = \mathbf{b}',$$

or

$$[A_B,\ I_s]\begin{bmatrix}\mathbf{x}'_B \\ \mathbf{x}'_s\end{bmatrix} = \mathbf{b}',$$

from which we get $\mathbf{x}'_s = \mathbf{b}' - A_B\mathbf{x}'_B$. If $\mathbf{x}'_s \geq 0$ then $\mathbf{x}'$ is primal feasible. Hence the primal algorithm can proceed from Phase II. If any entry of $\mathbf{x}'_s$, say $\mathbf{x}'_{sj}$, is negative, then it is possible to represent it as the difference of two nonnegative variables, i.e. $\mathbf{x}'_{sj} = \mathbf{x}'_{sj_1} - \mathbf{x}'_{sj_2}$, where $\mathbf{x}'_{sj_1} \geq 0$ and $\mathbf{x}'_{sj_2} \geq 0$. A feasible point is thus obtained at the expense of an extra column. The projective algorithm can proceed from Phase II.

4.5 Summary

In this chapter we have considered variants of the Karmarkar algorithm which provide dual solutions and use them to deal with problems for which the optimum objective value is not known. Three algorithms were considered. All of them are basically similar. The dual variables are generated in the same way as well as the bounds on the optimum objective value. The differences are in the way the convergence of these algorithms was established. This led to different suggested steps to take along the negative projected gradient. In practice these steps were found inappropriate. A longer step was used in our experiments. At least from the theoretical point of view, the variant of Ye and Kojima is superior to the others; in Todd (1988a) it is shown that it generates better bounds on z*. It also works under milder assumptions and seems to work on any type of LP problems. The algorithm will be further discussed in Chapter 6, and used for decomposition.

One of the purposes of this investigation of dual Karmarkar algorithms is to find out whether postoptimality analysis is possible. We have shown that the sensitivity of the solution to changes in the cost vector, the right-hand side and the rim can be studied using Karmarkar type algorithms. Updating the solution, after a perturbation has occurred in these components of the LP problem, is possible but may be expensive. On the small problems we experimented with, the updating of the solution often took as many iterations as was necessary to find it in the first place. For problems of this size the simplex takes no more than one or two steps to find the new solution. However, our experiments are limited and not conclusive. Further investigation of postoptimality analysis is necessary.

Chapter 5

A Centring Scheme Based on Chebyshev Points

5.1 Introduction

The present chapter is concerned with investigating the possible use of the Chebyshev problem as a centring scheme in the design of a new algorithm for linear programming. Centring schemes were successfully used in interior point algorithms for LP discussed in chapters 1 and 2. We recall that the benefits of such strategies are keeping feasibility and speeding convergence by allowing large steps to be taken along the gradient direction.

Our approach is similar to that of Levin (1965), used in the simplicial algorithm: After a Chebyshev point is found, a simplex containing the feasible region is split through the point, using the objective function as the cutting hyperplane. A Chebyshev point of the new system of inequalities is found and the operation is repeated, and so on. The convergence of such a process is established, and numerical results are reported.

5.2 The Chebyshev Problem

Given a set of m linear inequalities in n variables ($n < m$), the Chebyshev problem is that of finding point $\mathbf{x}^*$ in R^n which is equidistant from a set of n+1 hyperplanes among the m inequalities. Such a point is called a Chebyshev point.

Consider a system of linear inequalities

$$\eta_i(x) = a_{i1}x_1 + \ldots + a_{in}x_n + b_i \le 0, \quad i = 1, \ldots, m > n. \tag{5.1}$$

A Chebyshev point $\mathbf{x}^*$ is a solution to the system of linear inequalities if

$$\max_{1\le i\le m} \eta_i(x^*) = \min_{x} \max_{1\le i\le m} \eta_i(x) = L,$$

where L is called the deviation.

A system of linear inequalities is solvable if and only if $L \leq 0$. In this case the Chebyshev point is unique. The point lies inside the solution set (feasible region) of the system of inequalities, but also inside the largest simplex defined by any (n+1) inequalities at equal distance from its sides. This holds for consistent systems. Thus we assume throughout, that the LP problems to be considered have bounded and nonempty feasible regions.

5.3 Converting the Chebyshev Problem into a LP Problem

It is well known that the Chebyshev problem can be solved as a LP problem [Zukhovitsky & Avdeyeva, 1966]. The conversion is based on introducing a variable for the deviation in the set of linear inequalities and optimizing that variable as the objective function over the polytope defined by the linear inequalities. Following this idea, introduce variable x_{n+1} in every inequality of (5.1) and rewrite the system as

$$\eta_i(x) \equiv a_{i1}x_1 + \ldots + a_{in}x_n + x_{n+1} + b_i \leq 0 \quad (i = 1, \ldots, m > n).$$

The linear programming formulation of the Chebyshev problem is thus

$$\text{CLP:} \quad \begin{aligned} &\min \; x_{n+1} \\ &\text{s.t. } a_{i1}x_1 + \ldots + a_{in}x_n + x_{n+1} + b_i \leq 0 \quad (i = 1, \ldots, m). \end{aligned}$$

Any standard LP technique can be applied to CLP. The solution $\mathbf{x}^* = (x_1, x_2, \ldots, x_n)^T$ is the Chebyshev point and $x_{n+1} = L$ its deviation.

5.4 Equivalence of CP and LP

In this section we will look at the LP problem as a Chebyshev problem or a sequence of Chebyshev problems whose solutions converge to that of the LP.

Consider the problem

$$\text{SLP:} \quad \min \mathbf{c}^T\mathbf{x}$$
$$\text{s.t} \quad A\mathbf{x} \le \mathbf{b}$$
$$\mathbf{x} \ge 0.$$

A feasible point to SLP may be found by solving the Chebyshev problem

$$\min_{\mathbf{x}} \max_{i} \eta_i(\mathbf{x}) = L,$$

$$\text{for} \quad \eta(\mathbf{x}) \equiv A\mathbf{x} - \mathbf{b} \le 0.$$

To find the optimum solution to SLP first assume that the optimum objective value of the original SLP is known, i.e. $\mathbf{c}^T\mathbf{x}^* = z^*$. This equation may be written as two inequalities in the following manner

$$\left\{\mathbf{c}^T\mathbf{x} = z^*\right\} \equiv \begin{cases} \mathbf{c}^T\mathbf{x} \le z^* \\ -\mathbf{c}^T\mathbf{x} \le -z^*. \end{cases}$$

They are then incorporated in the set of constraints of the LP problem. In other words, we transform the LP problem into a set of linear inequalities. The Chebyshev problem corresponding to these inequalities can be written as follows

$$\min_{\mathbf{x}} \max_{i} \eta_i(\mathbf{x}) = L,$$

for

$$\eta(\mathbf{x}) \equiv A\mathbf{x} - \mathbf{b} \le 0,$$

and

$$\eta_{m+1}(\mathbf{x}) \equiv \mathbf{c}^T\mathbf{x} - z^* \le 0$$
$$\eta_{m+2}(\mathbf{x}) \equiv -\mathbf{c}^T\mathbf{x} + z^* \le 0.$$

Now we consider the corresponding LP problem which is

$$\min \quad x_{n+1}$$

s.t.

$$\begin{pmatrix} A & 1 \end{pmatrix}\begin{pmatrix} \mathbf{x} \\ x_{n+1} \end{pmatrix} - \mathbf{b} \leq 0,$$

$$\begin{pmatrix} \mathbf{c}^T & 0 \end{pmatrix}\begin{pmatrix} \mathbf{x} \\ x_{n+1} \end{pmatrix} - z^* \leq 0,$$

$$\begin{pmatrix} -\mathbf{c}^T & 0 \end{pmatrix}\begin{pmatrix} \mathbf{x} \\ x_{n+1} \end{pmatrix} + z^* \leq 0,$$

$\mathbf{x} \geq 0$, x_{n+1} unconstrained.

The advantage of reducing the LP problem into above form resides in its very sparse cost vector, as $z = x_{n+1}$ is the linear form to be minimized. The sparsity of the cost vector results in large savings when solving the least squares problem arising in the computation of the projected gradient using the Karmarkar algorithm. The solution to the Chebyshev problem is that of the original LP problem. This corresponds to a deviation equal to zero as the cut with the objective function hyperplane goes through the solution. Hence the feasible region is reduced to one single point, i.e. $\mathbf{x}^*$.

5.4.1 Numerical Results

Consider the following small problem solved both as a normal LP problem and under the Chebyshev form using Algorithm 4.1. The paths generated by the algorithm are different for the two forms of the same problem.

Form1:

$$\begin{array}{ll} \text{Max} & 10x_1 + 12x_2 \\ \text{s.t} & 2x_1 + 3x_2 \leq 1500 \\ & 3x_1 + 2x_2 \leq 1500 \\ & x_1 + x_2 \leq 600 \\ & x_1 \geq 0, x_2 \geq 0. \end{array}$$

The Chebyshev form of the problem is

Form2:

$$\begin{array}{ll} \text{Min} & x_3 \\ \text{s.t -} & 2x_1 - 3x_2 + x_3 \geq -1500 \\ & -3x_1 - 2x_2 + x_3 \geq -1500 \\ & -x_1 - x_2 + x_3 \geq -600 \\ & -10x_1 - 12x_2 \geq -6600 \\ & x_1 \geq 0, x_2 \geq 0. \end{array}$$

Iter	Solution of Form 1		Solution of form 2		
	x_1	x_2	x_1	x_2	x_3
1	245.5143	245.5143	299.9756	300.0193	299.4940
2	187.8610	374.0975	260.0384	333.3012	21.4343
3	300.2600	298.8201	300.1660	299.8616	0.6035
4	299.1131	300.5751	299.9087	300.0760	4.9205-2
5	299.9968	299.9864	300.0004	299.9996	1.4475-3
6	299.9890	299.9983	299.9997	300.0001	1.1259-4
7	299.9946	299.9945	300.0000	299.9999	3.4651-6
8	299.9945	299.9946			
9	299.9946	299.9946			
10	299.9999	299.9999			

Table 5.1 Paths Generated by Algorithm 4.1 for Two Different Forms of the Same Problem. The First Point in Both Cases is Obtained in Phase I of the Algorithm.

We report here some results on a set of small problems solved as simple LP problems and as Chebyshev problems. The same version of the Karmarkar algorithm has been used.

Problems	LP Form	Chebyshev Form
Beale's Prob.	10	12
Prob. 2	6	6*
Prob. 3	8	8
Prob. 4	10	11
Prob. 5	12	9*
Prob. 6	10*	9
Prob. 7	8	10
Klee-Minty Prob.	24	22*
Prob.9	4	6
Prob. 10	8	9

Table 5.2 Iteration Count for a Variant of Karmarkar's Algorithm on a Set of LP Problems in 2 Different Forms.

Note: (*) refers to the failure of the algorithm to stop despite converging at some stage to a good approximate solution.

5.4.2 LP Problems with Unknown Optimum Objective Value

Usually, the optimum objective value of LP problems is not known. In the following we will show that a sequence of Chebyshev points converges to the solution of the LP problem at hand without the assumption that z* is available.

Consider again SLP with nonempty feasible domain and nondegenerate optimum solution. A Chebyshev point $\mathbf{x}_c$ exists for the set of constraints and can be obtained by solving CLP. The objective value of the LP problem is then evaluated at $\mathbf{x}_c$. Thus $\mathbf{c}^T\mathbf{x}_c = z_c^*$.

If we augment the set of constraints of the Chebyshev problem with the constraints $\mathbf{c}^T\mathbf{x} \leq z_c^*$, this corresponds to cutting the feasible region of the Chebyshev problem through $\mathbf{x}_c$. The volume of the polyhedron defined by the LP constraints is reduced by a value which is not zero. The process is then repeated after updating at each iteration the value of z_c. After a finite number of iterations (cycles) the Chebyshev point is close to $\mathbf{x}^*$, the optimum solution of the LP problem at hand. Hence, when the deviation is zero, the feasible region of the CLP and the original LP problem are reduced to one point. This is the optimum solution $\mathbf{x}_c^*$, and the optimum objective value is $z^* = z_c^* = \mathbf{c}^T\mathbf{x}_c$.

5.5 Convergence of a Sequence of Chebyshev Points

In any simplex in R^n, i.e. a simplex with n+1 sides, it is possible to inscribe a n-sphere of radius $\rho > 0$, whose centre is a Chebyshev point. Cutting through the centre leads to another simplex which contains half of the previous one. Its Chebyshev point is at distance ρ' from the cutting plane. This distance is the deviation of the Chebyshev point and the radius of the largest sphere inscribed in the remaining part of the simplex.

To show that the sequence of Chebyshev points converges to the solution of the LP problem it is enough to show that the sequence converges to a vertex of the simplex after a finite number of cuts. This is equivalent to show that after each cut a decrease occurs in the absolute value of the deviation, i.e. the radius of the new sphere is less than that of the current one. For simplicity we consider a regular simplex in the plane, Fig.5.1.

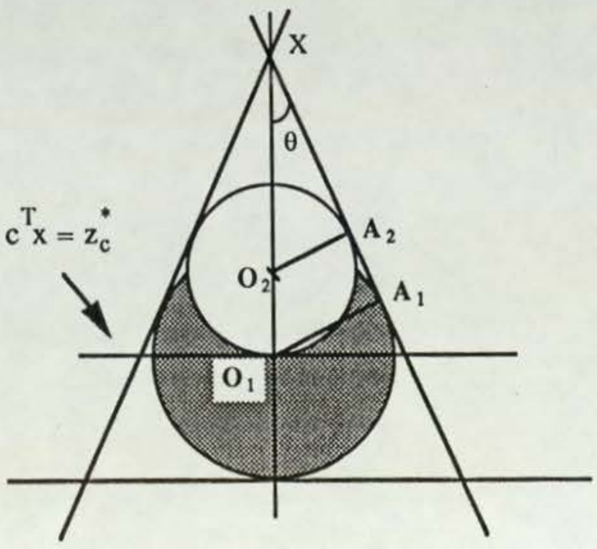


Fig 5.1 Centring Process of the Chebyshev Approach

$$\sin\theta = O_2A_2 / XO_2 = O_1A_1 / XO_1,$$

$$XO_1 = XO_2 + O_2O_1,$$

$$XO_2.\sin\theta + O_2O_1.\sin\theta = O_1A_1,$$

thus

$$O_2A_2 + O_2O_1.\sin\theta = O_1A_1,$$

and

$$O_2A_2 = O_1A_1 - O_2O_1.\sin\theta.$$

O_1A_1 is the radius of the current sphere, and O_2A_2 that of the new sphere. As $O_2O_1 = O_2A_2$, then $\rho_1 = \rho_2\,(1+ \sin\theta)$. Moving from centre O_1 to centre O_2 results into decreasing by $\delta = O_2O_1.\sin\theta$ the radius of the current sphere. The finiteness of the algorithm depends on the sign of δ. The latter can be shown to be strictly positive as follows.

From the assumption that an n-simplex is defined by the inequalities, it follows that $\sin\theta > 0$. Centres O_1 and O_2 are obtained at successive cycles k and k+1. They are optimum solutions of two LP problems which differ only in one entry of their RHS. Denote them $\mathbf{x}^{(k)}$ and $\mathbf{x}^{(k+1)}$, then

$$O_2O_1 = \left\| \mathbf{x}^{(k)} - \mathbf{x}^{(k+1)} \right\| = \sqrt{\sum_{j=1}^{n}\left(x_j^{(k)} - x_j^{(k+1)}\right)^2}\,.$$

As $\mathbf{x}^{(k)}$ and $\mathbf{x}^{(k+1)}$ are basic solutions, they can be expressed, using Cramer's rule, as follows

$$\mathbf{x}^{(k)} = \frac{\Delta^{(k)}}{\Delta} \quad \text{and} \quad \mathbf{x}^{(k+1)} = \frac{\Delta^{(k+1)}}{\Delta}.$$

The matrix of the prcblem remains identical from cycle to cycle; we can assume that Δ, the determinant of the optimum basis remains the same as well, without loss of generality. But $\Delta \neq 0$ always holds. $\Delta^{(k)}$ and $\Delta^{(k+1)}$, however, are different from cycle to cycle, as they involve the RHS which has one entry changing when updating the objective function value in the constraint $\mathbf{c}^T\mathbf{x} \leq z_c^*$. We can then write

$$O_2O_1 = \sqrt{\sum_{j=1}^{n}\left(\frac{\Delta_j^{(k)} - \Delta_j^{(k+1)}}{\Delta}\right)^2}\ .$$

$\Delta_j^{(k)} \neq 0$ and $\Delta_j^{(k+1)} \neq 0$ from the assumption that the feasible region is nonemty and $\forall j \in \{1, 2, ..., n\}$, $\forall k$, $\exists\ (h = j)$ such that $\Delta_h^{(k)} - \Delta_h^{(k+1)} \neq 0$. Thus

$$O_2O_1 > 0 \text{ and } \delta = O_2O_1.\sin\theta > 0.$$

5.5.1 An Algorithm for LP

We describe our algorithm based on the generation of a sequence of Chebyshev points converging to optimum solution of LP problems with bounded nonempty feasible domaine. Nondegeneracy is also assumed. The optimum objective value may not be known.

Algorithm 5.1

```
1- Transform the LP problem into a minimax Chebyshev problem.
2- Set up the corresponding LP problem.
3- Solve the LP problem using Karmarkar's algorithm to get a Chebyshev point $x_c^*$
     and its deviation L.
4- if $|L| = 0$ then
     $x^* = x_c^*$, optimum attained, stop.
   else
     go to 5-.
   endif
5- Cut through $x_c^*$ by augmenting the Chebyshev problem with the constraint
     $c^Tx \leq c^Tx_c^*$.
6- Repeat from 2-.
```

5.6 Computational Considerations: Improvements to Algorithm 5.1

Each iteration of Algorithm 5.1 is rather a cycle as it involves a linear programming problem which is as large as the original one. It looks as Algorithm 5.1 cannot be as efficient as solving the problem directly without passing through transforming it into a Chebyshev problem. However, there are some obvious improvements one can think of. These improvements follow.

Improvement one

One of the improvements which can be thought of stems from the geometric interpretation of the Chebyshev problem. It consists in reducing the number of constraints in the LP formulation of the Chebyshev minimax problem.

As was noticed earlier an n-simplex in R^n, defined by n+1 of the LP problem constraints, contains the feasible region. These n+1 constraints can be identified by the fact that all of them present the same deviation L to the Chebyshev point. Thus, after the Chebyshev point has been found, it is possible to remove the constraints with deviation larger than L. The cut through the point is carried out, and a new Chebyshev point is found. However, feasibility is no longer guaranteed. Thus, if any of the removed constraints is not satisfied, it is reintroduced in the reduced problem and a new Chebyshev point is found. The problem is again reduced and so on. This approach may be very profitable when $m >> n$.

Improvement two

Another improvement would be to take advantage of the fact that a Chebyshev point is a "centre". Thus, a great decrease in the objective function may result from moving in the direction of steepest descent towards a boundary, starting from a Chebyshev point. The resulting point is then used in the cutting process. In the following we shall describe a procedure for finding the steepest descent direction.

As in Zoutendjik (1960) a feasible direction can be found by solving a linear problem derived from CLP. Such a problem can be formulated as

$$\begin{aligned} &\min\ d_{n+1} \\ &\text{s.t}\ \eta_i(d) \equiv a_{i1}d_1 + \ldots + a_{in}d_n + d_{n+1} \leq b_i\ , \quad i \in I, \\ &\qquad\qquad\qquad\qquad |\, d_j \,| \leq 1, \quad j = 1, \ldots, n+1, \end{aligned}$$

where I is the set of indices of all constraints with deviations negative and larger than -L, and **d** the direction.

When **d** is found a step of length α is taken from $\mathbf{x}_c^*$ along **d** resulting into

$$\mathbf{x} = \mathbf{x}_c^* + \alpha\mathbf{d},$$

where α is chosen, [Zukhovitsky & Avdeyeva, 1966], as the smallest positive value among

$$\frac{-\eta_1\left(x_c^*\right)}{\sum_{j=1}^{n} a_{1j}d_j}, \ldots, \frac{-\eta_m\left(x_c^*\right)}{\sum_{j=1}^{n} a_{mj}d_j}.$$

These improvements embedded in Algorithm 5.1 result in the following algorithm.

Algorithm 5.2

1- Transform LP problem into a Chebyshev problem.

2- Set up the corresponding LP problem.

3- Solve the resulting LP problem using Karmarkar's algorithm to get a Chebyshev point $\mathbf{x}_c^*$ and its deviation L.

4- **if** $|L| = 0$, **then** $\mathbf{x}^* = \mathbf{x}_c^*$, problem solved, **stop.**

else go to 5-, **endif**

5- Select the constraints which constitute the sides of the simplex containing the feasible region (n+1 inequalities for which L is the same), and set up reduced problem.

6- Find a feasible direction vector **d**.

7- Find α and compute $\mathbf{x} = \mathbf{x}_c^* + \alpha\mathbf{d}$.

8- Cut through **x** if it is feasible for the overall problem.

9- If the point is not feasible, introduce the violated constraints into the reduced problem and cut through the previous Chebyshev point.

10- go to 3-.

5.7 Numerical Example

We consider the problem

$$\begin{array}{ll} \max & x_2 \\ \text{s.t.} & x_1 - x_2 \geq 1 \\ & -2x_1 - x_2 \geq -14 \\ & -x_1 - x_2 \geq -8 \\ & -0.5x_1 + x_2 \geq -3 \\ & x_1 \geq 0, x_2 \geq 0. \end{array}$$

To apply the algorithm described earlier, we set up the Chebyshev problem

$$\begin{array}{ll} \min & -x_3 \\ \text{s.t.} & x_1 - x_2 - x_3 \geq 1 \\ & -2x_1 - x_2 - x_3 \geq -14 \\ & -x_1 - x_2 - x_3 \geq -8 \\ & -0.5x_1 + x_2 - x_3 \geq -3 \\ & x_1 \geq 0, x_2 \geq 0, x_3 \geq 0, \end{array}$$

where $-x_3$ is the deviation. The solution to this problem using the projective algorithm is (4.50, 3.44). Given the position of the point in the feasible region, Fig 5.1, it is interesting to take a move along a feasible direction to get a better point through which we will do a cut.

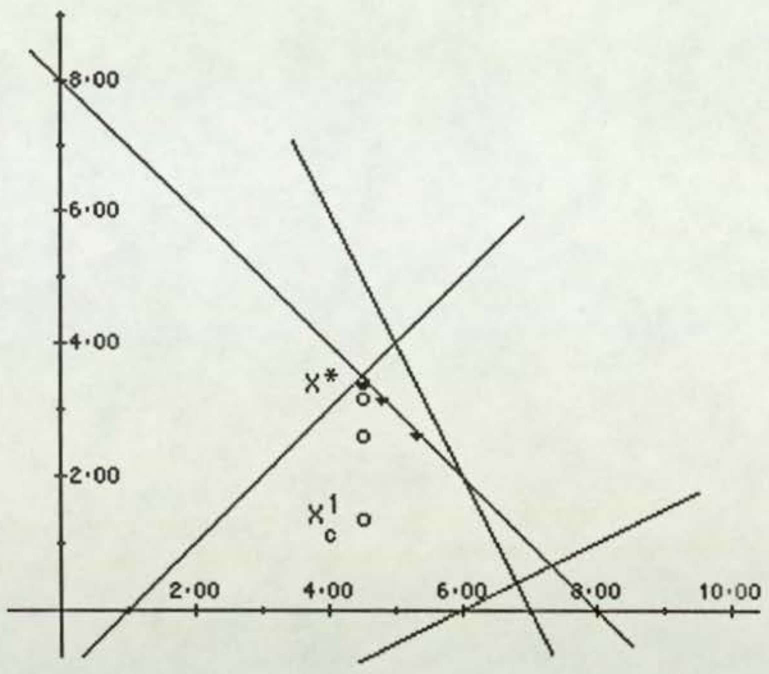


Fig 5.2 Sequence of Chebyshev Points (circles) Converging to **x***

Such a feasible direction is obtained as in Zoutendjik (1960) algorithm by solving the linear programming problem

$$\min \quad d_2$$
$$\text{s.t.} \quad |\, d_j \,| \leq 1, \qquad j = 1, 2,$$

where **d** is the direction vector. The vector direction in this case is **d** = (0.70, 1.00).

To find a new feasible point (dark points in Fig 5.2), a step α is taken along **d**. The steplength is obtained using the procedure described in section 5.7.

Chebyshev points	Interior feasible points	α
(4.50, 1.37)	(5.37, 2.63)	1.25
(4.50, 2.63)	(4.86, 3.14)	0.52
(4.50, 3.14)	(4.65, 3.35)	0.21
(4.50, 3.35)	(4.56, 3.44)	0.09
(4.50, 3.44)	-	-

Table 5.3 Points Returned by Algorithm 5.2 Partially Represented in Fig 5.2.

5.8 Conclusion

Algorithm 5.2 seems to work but is tedious and inefficient. However, the idea of using the Chebyshev problem as a centring scheme in a new interior point algorithm similar to the simplicial algorithm of Levin is viable and, I believe, may lead to an elegant method. The major criticism of Algorithm 5.2 is made regarding the work involved in every cycle. Indeed, any cycle corresponds to solving a problem larger than the original one, after transformation into a Chebyshev problem. Each cycle will take as many iterations as Karmarkar's algorithm would take to solve the original problem, although the iterations are cheaper due to the fact that the problem changes little from cycle to cycle. It may thus be possible to exploit this characteristic to cut down the work of the overall algorithm. The time limitation does not allow us to pursue this perspective, however, the procedure for postoptimality in the case of changes in the right-hand side presented in Chapter 4, and updating techniques for least squares, may be useful.

Chapter 6

Karmarkar Type Algorithms and Decomposition for Linear Programming

6.1 Introduction

The development of efficient optimization techniques for large structured linear programs is of major significance in economic planning, engineering and management science. An extensive literature exists on decomposition, which shows the magnitude of the effort devoted to the subject (for an excellent review see Geoffrion, 1970). Initially the idea of decomposition, as suggested by Dantzig and Wolfe (1960), was an extension of the use of the simplex method to solve large and structured LP problems. With implementations of this idea large LP problems arising in the oil industry, Government etc... were successfully solved. However, the decomposition algorithm, its variants and many other methods based on different ideas, never outclassed the standard simplex in terms of labour involved (when these large problems can be handled by the simplex). Commenting on staircase structured LP problems, Gear *et al.*, cited in [Fourer, 1982] say:

> "Today we know only how to solve it as we would any linear programming problem; but this type of problem requires more work to solve than does the average problem of the same size. However, there should be some way to take advantage of its simple structure."

Decomposition is usually called upon only when the problem cannot be handled by the standard simplex method, because of its size. Otherwise the standard simplex is preferred to decomposition, as it is easier to implement and involves less CPU time. However, following Gear *et al.*, decomposition should not only allow the solution of large problems under storage constraints but also in competitive times. Indeed, it should be viewed, primarily, as a means for mass exploitation of sparsity, when favourable structure is present. It is also a fact that today CPU time is more of a scarce resource than storage.

The relative "inefficiency" of decomposition algorithms so far developed, may be due to their tight relationship with the simplex algorithm. It is, therefore, worthwhile investigating decomposition in conjunction with interior point methods.

In the following we shall investigate the applicability of interior point methods, of Karmarkar type, coupled with classical decomposition principals [Rosen, 1964; Grigoriadis & Ritter, 1969] to structured and even unstructured LP problems. However, emphasis will be on the specialization of a dual Karmarkar algorithm to block-diagonal LP problems. A new partitioning algorithm for linear programming will be presented, supported with experimental results on randomly and non-randomly generated structured problems.

6.2 Structured LP Problems

Structure is an important attribute of large scale linear programming. Large scale programs almost always have distinctive structure beside convexity and linearity properties. There are many types of structure. However, the commonest and most important are multidivisional, combinatorial, dynamic and stochastic structures. We shall be interested in multidivisional problems which consist of interrelated subsystems to be optimized (Geoffrion, 1970). The subsystems can be modules of an engineering system, reservoirs in a water resources system, departments or divisions of an organization, production units of an industry, or sectors of an economy. The interrelation between the subsystems is represented with the so called linking constraints or variables.

The commonest structure in large LP problems is the block-angular structure, (see Fig.6.1). In standard form, such structured LP problems can be written as:

$$\begin{aligned} \min\ & \mathbf{c}_0^T\mathbf{x}_0 + \sum_{i=1}^{r} \mathbf{c}_i^T\mathbf{x}_i \\ \text{s.t. } & A_0\mathbf{x}_0 + \sum_{i=1}^{r} A_i\mathbf{x}_i = \mathbf{b}_0, \\ & B_i\mathbf{x}_i = \mathbf{b}_i, \\ & \mathbf{x}_0, \mathbf{x}_i \geq 0, \\ & i = 1, \ldots., r. \end{aligned} \tag{6.2.1}$$

The dual to the above problem is

$$\begin{aligned} \max\ & \mathbf{b}_0^T\mathbf{y}_0 + \sum_{i=1}^{r} \mathbf{b}_i^T\mathbf{y}_i \\ \text{s.t. } & A_0^T\mathbf{y}_0 \leq \mathbf{c}_0, \\ & A_i^T\mathbf{y}_0 + B_i^T\mathbf{y}_i \leq \mathbf{c}_i, \\ & \mathbf{y}_0, \mathbf{y}_i \text{ unrestricted}, \\ & i = 1, \ldots., r. \end{aligned} \tag{6.2.2}$$

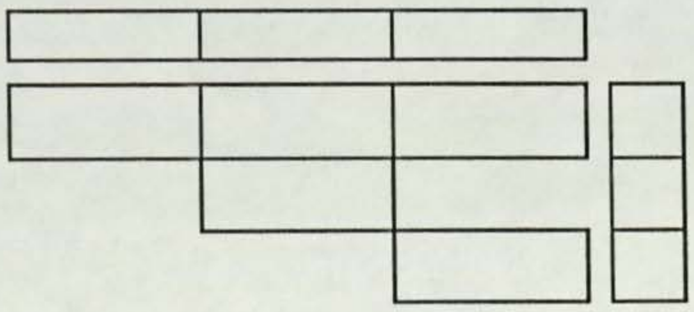


Fig 6.1 Diagram of a 2-Block LP Problem

The general block-diagonal LP problem with linking constraints and linking variables, in standard form is show below.

$$\min\ \mathbf{c}_0^T\mathbf{y} + \sum_{i=1}^{r}\mathbf{c}_i^T\mathbf{x}_i \qquad (6.2.3)$$
$$\text{s.t. } D_0\mathbf{y} + \sum_{i=1}^{r}A_i\mathbf{x}_i = \mathbf{b}_0,$$
$$D_i\,\mathbf{y} + B_i\mathbf{x}_i = \mathbf{b}_i,$$
$$\mathbf{y}, \mathbf{x}_i \geq 0,$$
$$i = 1, \ldots, r.$$

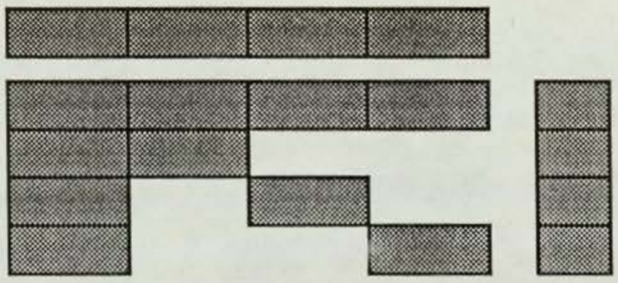


Fig 6.2 General Block Diagonal LP Problem

Another common structured LP problem is the staircase or time-staged problem. It is different from the above case in that every two successive blocks are linked by a set of variables (in the primal form) or constraints (in the dual form). Such a problem in standard form can be written as:

$$\min\ \sum_{i=1}^{r+1}c_i^Tx_i \qquad (6.2.4)$$
$$\text{s.t. } A_i\,x_i + B_ix_{i+1} = b_i$$
$$x_i \geq 0,$$
$$i = 1, \ldots, r.$$

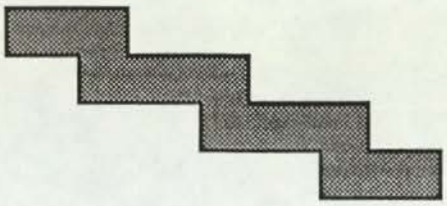


Fig 6.3 Staircase Structure

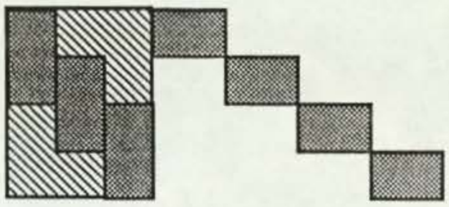


Fig 6.4 Equivalent Block Angular Structure

The staircase structure is amenable to block-diagonal form (Fig 6.4). Solving this type of LP problems will be considered at the end of this chapter.

6.3 A Decomposition Algorithm Using Karmarkar's Method

The main purpose of decomposition is to exploit the inherent parallelism of block-angular structured problem, using the relative independence of the subproblems. This independence is apparent in the formulation of LP problems as:

$$\begin{array}{ll} \text{Opt} & c_0^T x_0 + \mathbf{c}_1^T \mathbf{x}_1 + \ldots + \mathbf{c}_r^T \mathbf{x}_r \\ \text{s.t.} & \mathbf{x} \in K_0 \cap K_1 \cap \ldots \cap K_r, \end{array}$$

where Opt can be min or max and K_i, i = 0, ..., r, is the convex polyhedron, feasible domain of the ith subproblem.

In (6.2.1) the subproblems that are to be considered have the form

$$\begin{array}{ll} \min & \left(\mathbf{c}_i^T - \mathbf{y}_0^T A_i\right) \mathbf{x}_i \\ \text{s.t.} & B_i \mathbf{x}_i = \mathbf{b}_i, \quad i = 1, \ldots, r, \\ & \mathbf{x}_i \geq 0. \end{array} \qquad (6.2.5)$$

The corresponding duals are

$$\max \ \mathbf{b}_i^T \mathbf{y}_i$$
$$\text{s.t.} \quad B_i^T \mathbf{y}_i \le \mathbf{c}_i - A_i^T \mathbf{y}_0, \quad i = 1, \ldots, r. \qquad (6.2.6)$$

Assume that an initial feasible vector $\mathbf{y}_0^0$ is at hand, for which every subproblem (6.2.5) has a feasible solution, i.e., there exists $\mathbf{y}_0^0$ such that

$$B_i^T \mathbf{y}_i^0 \le \mathbf{c}_i - A_i^T \mathbf{y}_0^0.$$

This assumption is not very restrictive as it is always possible to guess a feasible solution from the real world interpretation of the problem. However, a systematic way for providing such a point would be to solve a Phase I problem using Karmarkar algorithm. In fact this assumption is also made in the original projective method (see Chapter 2).

If we solve each subproblem (6.2.6) using the dual Karmarkar algorithm given in Chapter 4, the dual solutions $\mathbf{y}_i^0$ will be obtained. The optimum objective value of (6.2.5) is $z_i^0 = \mathbf{b}_i^T \mathbf{y}_i^0$. The objective value of the complete problem is

$$z^0 = \mathbf{b}_0^T \mathbf{y}_0^0 + \sum_{i=1}^{r} \mathbf{b}_i^T \mathbf{y}_i^0.$$

From, the duality theorem, z^0 is a lower bound on the optimum objective value of the complete problem.

To see whether the solution obtained is feasible or optimal to the complete problem, we set up an equivalent problem based on the linking constraints and the nonnegativity constraints only. The constraints forming the separate blocks are satisfied.

Consider the partitioning of matrices B_i into B_{i1} and B_{i2}. B_{i1} contains the basic columns of B_i as per solution of subproblems (6.2.6), and B_{i2} contains the nonbasic columns. Accordingly, we partition other matrices and vectors involved; i.e.

$$\mathbf{c}_i^T = \left(\mathbf{c}_{i1}^T \quad \mathbf{c}_{i2}^T \right),$$

$$A_i = \begin{pmatrix} A_{i1} & A_{i2} \end{pmatrix},$$
$$\mathbf{x}_i^T = \begin{pmatrix} \mathbf{x}_{i1}^T & \mathbf{x}_{i2}^T \end{pmatrix}.$$

It follows that

$$\mathbf{x}_i^0 = B_{i1}^{-1}\mathbf{b}_i, \text{ and } \mathbf{y}_i^0 = \left(B_{i1}^{-1}\right)^T\left(\mathbf{c}_{i1} - A_{i1}^T\mathbf{y}_0^0\right).$$

We have also from the partitioning of ith block

$$\begin{pmatrix} B_{i1} & B_{i2} \end{pmatrix}\begin{pmatrix} \mathbf{x}_{i1} \\ \mathbf{x}_{i2} \end{pmatrix} = \mathbf{b}_i,$$

thus

$$\mathbf{x}_{i1} = \mathbf{x}_{i1}^0 - B_{i1}^{-1}B_{i2}\mathbf{x}_{i2} \tag{6.2.7}$$

The objective function of the original problem is

$$\mathbf{c}_i^T = \mathbf{c}_{i1}^T\mathbf{x}_{i1} + \mathbf{c}_{i2}^T\mathbf{x}_{i2} \tag{6.2.8}$$

Replacing $\mathbf{x}_{i1}$ in (6.2.8) by expression (6.2.7) we obtain

$$\mathbf{c}_i^T\mathbf{x}_i = \mathbf{c}_{i1}^T\mathbf{x}_{i1}^0 + \left(\mathbf{c}_{i2}^T - \left(B_{i1}^{-1}B_{i2}\right)^T\mathbf{c}_{i1}\right)\mathbf{x}_{i2}. \tag{6.2.9}$$

Similarly, the linking constraints can be written as

$$A_0\mathbf{x}_0 + \sum_{i=1}^{r}\left[A_{i1}\mathbf{x}_{i1} + A_{i2}\mathbf{x}_{i2}\right] = \mathbf{b}_0. \tag{6.2.10}$$

Substituting $\mathbf{x}_{i1}$ in (6.2.10) by its expression (6.2.7) we obtain

$$A_0\mathbf{x}_0 + \sum_{i=1}^{r}\left[A_{i2} - A_{i1}B_{i1}^{-1}B_{i1}\right]\mathbf{x}_{i2} = \mathbf{b}_0 - \sum_{i=1}^{r}A_{i1}\mathbf{x}_{i1}^0. \tag{6.2.11}$$

The reduced problem is

$$\min \mathbf{c}_0^T\mathbf{x}_0 + \sum_{i=1}^{r}\left(\mathbf{c}_{i2}^T - \left(B_{i1}^{-1}B_{i2}\right)^T\mathbf{c}_{i1}\right)\mathbf{x}_{i2} \qquad (6.2.12)$$

s.t. (6.2.11) and $\mathbf{x}_{i2} \geq 0$.

Algorithm 6.1 (based on the algorithm of Rosen (1964))

1- Find a feasible vector $\mathbf{y}_0 = \mathbf{y}_0^0$

2- Using the Karmarkar dual version, solve each subproblem (6.2.6). This provides the solution to subproblems (6.2.5)

3- Set up the reduced problem (6.2.12) based on the objective function of the original problem linking and nonnegativity constraints.

4- Solve the reduced problem using Karmarkar's algorithm to get $\mathbf{x}_{i2}*$ and $\mathbf{y}_0^1$ as a by-product.

if (6.2.12) is not feasible **then** the dual (6.2.2) of the original problem has an infinite solution **endif**.

5- Compute $\mathbf{x}_{i1}$ from (6.2.7) with $\mathbf{x}_{i2}$ replaced by $\mathbf{x}_{i2}*$.

if $\mathbf{x}_{i1} \geq 0$ **then** $(\mathbf{x}_{i1}, \mathbf{x}_{i2}*)^T$ is optimum solution .

else replace $\mathbf{y}_0^0$ by $\mathbf{y}_0^1$ and repeat from 2- **endif**

The validity of this algorithm derives from that of the Karmarkar algorithm and Rosen's scheme. A more general form is considered in the next section.

6.4 A General Form of Algorithm 6.1

In this section we shall consider the structured LP problem of the general form (6.2.3) which presents linking constraints as well as linking variables. Based on the idea of variable reduction of Algorithm 6.1, it is possible to derive a decomposition algorithm to which Karmarkar's algorithm can be coupled in a straightforward manner. The algorithm is basically that of Grigoriadis and Ritter (1969).

Assume that we partition the matrices B_i in (6.2.3) into two parts B_{i1} and B_{i2}, with B_{i1} being nonsingular with rank m_i. The vectors $\mathbf{x}_i$ and $\mathbf{c}_i$ being partitioned accordingly, we can write

$$\mathbf{x}_{i1} = B_{i1}^{-1}\mathbf{b}_i - B_{i1}^{-1}B_{i2}\mathbf{x}_{i2} - B_{i1}^{-1}D_i\mathbf{y}. \tag{6.4.1}$$

Replacing $\mathbf{x}_{i1}$ with its expression (6.4.1) in the objective function and the linking constraints of (6.2.3), we form the following reduced problem

$$\min\ \phi + \lambda_0^T\mathbf{y} + \sum_{i=1}^{r}\psi_i^T\mathbf{x}_{i2} \tag{6.4.2}$$

$$\text{s.t.}\qquad H_0\mathbf{y} + \sum_{i=1}^{r}H_i\mathbf{x}_{i2} = \mathbf{b}$$

$$\mathbf{y} \geq \mathbf{0},\ \mathbf{x}_{i2} \geq \mathbf{0}.$$

The matrices H_0 and H_1 are easily derived, as well as the vectors in the objective function. They are expressed as follows

$$\phi = \sum_{i=1}^{r}\mathbf{c}_{i1}^T B_{i1}^{-1}\mathbf{b}_i \tag{6.4.3}$$

$$\psi_i = \mathbf{c}_{i2}^T - \mathbf{c}_{i1}^T B_{i1}^{-1}B_{i2} \tag{6.4.4}$$

$$\lambda_0 = \mathbf{c}_0^T - \sum_{i=1}^{r}\mathbf{c}_{i1}^T B_{i1}^{-1}D_i \tag{6.4.5}$$

$$H_0 = D_0 - \sum_{i=1}^{r}A_{i1}B_{i1}^{-1}D_i \tag{6.4.6}$$

$$H_i = A_{i2} - A_{i1}B_{i1}^{-1}B_{i2} \tag{6.4.7}$$

$$\mathbf{b} = \mathbf{b}_0 - \sum_{i=1}^{r}A_{i1}B_{i1}^{-1}\mathbf{b}_i \tag{6.4.8}$$

Problem (6.4.2) can be solved using Karmarkar's algorithm. Its optimal solution is $\mathbf{x}_{i2}^*$. The solution to the original problem is $(\mathbf{x}^*_{i1}, \mathbf{x}_{i2}^*, \mathbf{y}^*)$, obtained by computing $\mathbf{x}^*_{i1}$

from the relation (6.4.1). If $\mathbf{x}^*_{i1} \geq \mathbf{0}$ then the optimum solution to the complete problem is obtained. If among $\mathbf{x}^*_{i1}$ entries there are negative ones, then the optimum solution is not yet reached. We shall, later, outline a procedure for dealing with negative elements in $\mathbf{x}^*_{i1}$.

Theorem 6.1 [Grigoriadis & Ritter, 1969] $(\mathbf{x}^*_{i1}, \mathbf{x}_{i2}^*, \mathbf{y}^*)$, $i = 1, \ldots, r$, is an optimal solution to (6.2.3) if and only if $\mathbf{x}^*_{i1} \geq \mathbf{0}$.

Proof: The condition is necessary as the solution should satisfy the nonnegativity constraints of (6.2.3). It is sufficient because from (6.4.1), the reduced problem (6.4.2) with the additional constraint (6.4.9) is equivalent to (6.2.3). Thus $(\mathbf{x}^*_{i1}, \mathbf{x}_{i2}^*, \mathbf{y}^*)$ is optimal solution to (6.2.3) if $\mathbf{x}^*_{i1} \geq \mathbf{0}$. QED.

Suppose that for some i, $(\mathbf{x}^*_{i1})_j$ is a negative component of $\mathbf{x}^*_{i1}$. The idea is to make sure that this element is enforced to be nonnegative in the reduced problem. From relation (6.4.1), $(\mathbf{x}^*_{i1})_j$ is given by equation (6.4.9) enforced to be nonnegative.

$$\left(\mathbf{x}^*_{i1}\right)_j = \left(\mathrm{B}_{i1}^{-1}\right)_j \mathbf{b}_i - \left(\mathrm{B}_{i1}^{-1}\mathrm{B}_{i2}\right)_j \mathbf{x}^*_{i2} - \left(\mathrm{B}_{i1}^{-1}\mathrm{D}_i\right)_j \mathbf{y}^* \geq 0. \qquad (6.4.9)$$

The subscript j is an index to the rows of the matrices and vectors involved in the computation of the negative component.

Constraint (6.4.9) is then added to reduced problem (6.4.2) and a new cycle is carried out.

Example 6.1

Let solve the following structured problem using the above generalization of Algorithm 6.1

$$\begin{aligned}
\text{Min}\quad & -x_1 - x_2 - 2x_3 - 3x_4 \\
\text{s.t.}\quad & x_1 + 2x_2 + 2x_3 + x_4 \le 40 \\
& x_1 + x_2 + 4x_3 + 2x_4 \le 50 \\
& x_1 + 3x_2 \le 30 \\
& 2x_1 + x_2 \le 20 \\
& x_3 \le 10 \\
& x_4 \le 10 \\
& x_3 + x_4 \le 15 \\
& x_1, x_2, x_3, x_4 \ge 0.
\end{aligned}$$

After adding slack variables, we proceed to the following partitioning of the problem.

c:	1	1	0	0	2	1	0	0	0	0	0	**b**	
A:	1	2	0	0	2	1	0	0	0	1	0	40	$\mathbf{b}_0$
	1	1	0	0	4	2	0	0	0	0	1	50	
B_{1j}:	1	3	1	0								30	$\mathbf{b}_1$
	2	1	0	1								20	
B_{2j}:					1	0	0	1	0	0	0	10	$\mathbf{b}_2$
					0	1	1	0	0	0	0	10	
					1	1	0	0	1	0	0	15	
										D_i			

From above partitioning, we have

$$B_{11} = \begin{pmatrix} 1 & 3 \\ 2 & 1 \end{pmatrix}, \text{ and } B_{21} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix}.$$

The submatrices A_{11}, A_{12}, A_{21}, A_{22}, B_{12}, B_{22}, D_0, D_1, D_2 and the vectors $\mathbf{c}_{11}$, $\mathbf{c}_{12}$, $\mathbf{c}_{21}$, $\mathbf{c}_{22}$, $\mathbf{c}_0$, $\mathbf{b}_0$, $\mathbf{b}_1$, $\mathbf{b}_2$ are also easily read. We compute the quantities given by (6.4.3) - (6.4.8) and set up the reduced problem (6.4.2) as follows

$$\begin{array}{llr}
\text{Max } 49 & -0.2(x_{12})_1 - 0.4(x_{12})_2 + (x_{22})_1 - 3(x_{22})_2 + 0y_1 + 0y_2 & \\
\text{s.t.} & -0.6(x_{12})_1 - 0.2(x_{12})_2 - (x_{22})_1 - (x_{22})_2 + y_1 & = -7 \\
 & -0.2(x_{12})_1 - 0.4(x_{12})_2 - 2(x_{22})_1 - 2(x_{22})_2 + y_2 & = -14 \\
 & (x_{12})_j, (x_{22})_j, y_1, y_2 \geq 0. &
\end{array}$$

The problem is solved by Karmarkar algorithm and seems to be unbounded. However an intermediate feasible solution indicates, when replaced in (6.4.1) that one of the solutions corresponding to the original problem is negative. The variable is forced to be nonnegative using equation (6.4.9). This leads to the new reduced problem problem

$$\begin{array}{llr}
\text{Max } 49 & -0.2(x_{12})_1 - 0.4(x_{12})_2 + (x_{22})_1 - 3(x_{22})_2 + 0y_1 + 0y_2 + 0y_3 & \\
\text{s.t.} & -0.6(x_{12})_1 - 0.2(x_{12})_2 - (x_{22})_1 - (x_{22})_2 + y_1 & = -7 \\
 & -0.2(x_{12})_1 - 0.4(x_{12})_2 - 2(x_{22})_1 - 2(x_{22})_2 + y_2 & = -14 \\
 & -(x_{22})_1 + (x_{22})_2 + y_3 & = -5 \\
 & (x_{12})_j, (x_{22})_j, y_1, y_2, y_3 \geq 0. &
\end{array}$$

The solution to this problem is (0, 0, 6, 1, 0, 0, 0). Replacing, accordingly in (6.4.1) gives the solution $x_1 = 6$, $x_2 = 8$, $x_3 = 4$, $x_4 = 10$, $x_5 = 0$, $z^* = -52$. The reduced problem takes about ten iterations of the Karmarkar algorithm.

6.5 Dual Karmarkar Algorithm for Block-Angular LP Problems

We have shown that Karmarkar algorithm can be adapted to decomposition schemes for linear programming. In the following we will present a partitioning algorithm for block-angular LP problems. The dual version of the Karmarkar algorithm is used as described in chapter 5. However, the pseudoinverse will not be used to compute the dual solution. Instead, a QR or Cholesky factorization will be used as it is more appropriate for block-angular matrices as well as decomposition for general least squares problems (see Chapter 3).

The idea behind this partitioning scheme is that at each iteration the computation of dual solutions is carried out on a modular basis. That is, for each block the corresponding dual solutions are computed separately, then updated by considering the effect of the linking variables. Computation of the direction, primal solution and lower bound on the minimum objective is also made in parallel.

Consider the following block angular LP problem in standard form.

$$\begin{aligned} \min\ & \sum_{i=1}^{r-1} \mathbf{c}'^T_i \mathbf{x}_i + \mathbf{c}'_r \mathbf{x}_r \\ \text{s.t.}\ & B'_i \mathbf{x}_i + A'_i \mathbf{x}_r = \mathbf{b}_i, \\ & \mathbf{x}_i \geq 0, \mathbf{x}_r \geq 0, \\ & i = 1, \ldots, r-1 . \end{aligned} \qquad (6.5.1)$$

The blocks B'_i are $(m_i x n_i)$-matrices and A'_i are $(m_i x n_r)$-matrices; $\mathbf{c}'_i$ and $\mathbf{x}_i$ are n_i-vectors, and $\mathbf{b}_i$ are m_i-vectors, $i = 1, ..., r$.

Karmarkar's canonical form of (6.5.1) is

$$\begin{aligned} \min\ & \sum_{i=1}^{r-1} (D_i \mathbf{c}'_i)^T \mathbf{x}'_i + (D_r \mathbf{c}'_r)^T \mathbf{x}'_r - z x'_{n_r+1} \\ \text{s.t.}\ & B'_i D_i \mathbf{x}_i + A'_i D_r \mathbf{x}_r - \mathbf{b}_i x'_{n_r+1} = 0, \\ & \mathbf{x}'_i \geq 0, \mathbf{x}'_r \geq 0, x'_{n_r+1}, \\ & i = 1, \ldots, r-1 . \end{aligned} \qquad (6.5.1')$$

Put $\mathbf{c}_i = D_i \mathbf{c}_i'$, $\mathbf{c}_r = [D_r \mathbf{c}_r', -z]$, $B_i = B_i' D_i$ and $A_i = [A_i' D_r, -\mathbf{b}_i]$ and write (6.5.1') as

$$\begin{aligned} \min\ & \sum_{i=1}^{r-1} \mathbf{c}_i^T \mathbf{x}'_i + \mathbf{c}_r \mathbf{x}'_r \\ \text{s.t.}\ & B_i \mathbf{x}'_i + A_i \mathbf{x}'_r = 0, \\ & \mathbf{x}'_i \geq 0, \mathbf{x}'_r \geq 0, \\ & i = 1, \ldots, r-1 . \end{aligned} \qquad (6.5.2)$$

Problem (6.5.2) is block angular and homogeneous.

Based on the natural structure of (6.5.1), and thus of (6.5.2), the dual variables $\mathbf{y}$ are partitioned as follows.

$$D = (D_1, D_2, \ldots, D_r),$$
$$\mathbf{y}^T = (\mathbf{y}_1^T, \mathbf{y}_2^T, \ldots, \mathbf{y}_{r-1}^T).$$

6.5.1 Computing the Dual Solutions

The computation of the projected gradient in the dual Karmarkar algorithm (Algorithm 4.2, Chapter 4) is carried out through the computation of the dual variables with the use of pseudoinverse as

$$\mathbf{y}^T = (A')^{\dagger}\mathbf{c}', \tag{6.5.3}$$

where A' and **c**' refer to the matrix and cost vector of the problem in canonical form. In this section we shall present an efficient algorithm for computing $\mathbf{y}^T$ when A' and **c**' correspond to the matrix and cost vector of of problem (6.5.2), i.e. A' is block angular. The procedure is basically an extension of Algorithm 3.1 for structured least squares.

Clearly, equation (6.5.3) , gives the solution to the least squares problem

$$\min_{\mathbf{y}} \| A'^T\mathbf{y} - \mathbf{c}' \|_2. \tag{6.5.4}$$

However, solving large structured systems using the pseudoinverse is time consuming. Cholesky decomposition and orthogonal factorization are more appropriate, as will be seen later.

The problem to solve is depicted in (6.5.5).

$$\begin{pmatrix} B_1^T & & \\ & \ddots & \\ & & B_{r-1}^T \\ A_1^T & \ldots & A_{r-1}^T \end{pmatrix} \begin{pmatrix} \mathbf{y}_1 \\ \vdots \\ \\ \mathbf{y}_{r-1} \end{pmatrix} = \begin{pmatrix} \mathbf{c}_1 \\ \vdots \\ \mathbf{c}_{r-1} \\ \mathbf{c}_r \end{pmatrix} \tag{6.5.5}$$

$$A'^T \quad \cdot \quad \mathbf{y} \quad = \quad \mathbf{c}'.$$

Assume that the problem is weakly linked, i.e. only few linking variables are present in the problem, then consider the incomplete problem (6.5.3) based on (6.5.2) from which we remove equations $(A_1^T, A_2^T, \ldots, A_{r-1}^T)(\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_{r-1})^T = \mathbf{c}_r$.

$$\underbrace{\begin{pmatrix} B_1^T & & \\ & \ddots & \\ & & B_{r-1}^T \end{pmatrix}}_{B^T} \cdot \underbrace{\begin{pmatrix} \mathbf{y}'_1 \\ \vdots \\ \mathbf{y}'_{r-1} \end{pmatrix}}_{\mathbf{y}'} = \underbrace{\begin{pmatrix} \mathbf{c}_1 \\ \vdots \\ \mathbf{c}_{r-1} \end{pmatrix}}_{\mathbf{c}'} . \tag{6.5.6}$$

We solve separately the subsystems (6.5.4) in the sense of least squares.

$$B_i^T \mathbf{y}'_i = \mathbf{c}_i, \quad i = 1, \ldots, r-1 \tag{6.5.7}$$

The QR factorization leads for each subproblem to

$$QB_i^T = \begin{bmatrix} R_i \\ 0 \end{bmatrix}, \text{ and } \quad Q\mathbf{c}_i = \begin{bmatrix} \mathbf{s}_i \\ \mathbf{d}_i \end{bmatrix}.$$

A forward substitution delivers $\mathbf{y}'_i = R_i^{-1}\mathbf{s}_i$.

By computing the separate QR factors, we have actually also computed the QR factor of the block diagonal matrix B^T, as depicted below.

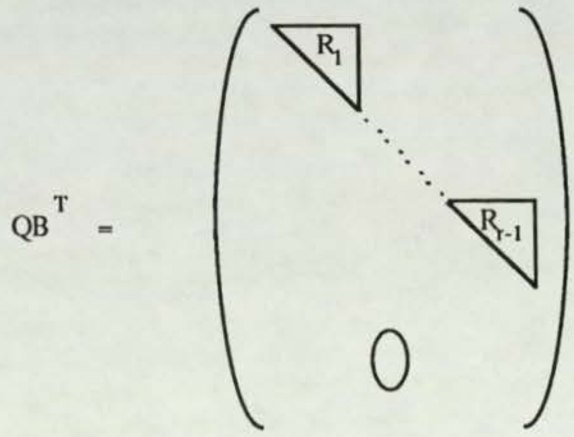


Fig 6.5 QR factor of the block-diagonal matrix B^T

Now, applying Algorithm 4.1 of Chapter 4, we update the solution **y'** by taking account of the removed rows ($A_1^T, A_2^T, \ldots, A_{r-1}^T$). However, given the structure of the problem matrix, it is better to apply a parallel version of Algorithm 4.1, which we propose here.

Algorithm 6.2

1- Solve each subproblem (6.5.7) using Cholesky method or QR factorization.

2- Compute

$$F = \left(A_1^T R_1^{-1} \; A_2^T R_2^{-1} \; \ldots \; A_{r-1}^T R_{r-1}^{-1}\right) = \left(F_1 \; F_2 \; \ldots \; F_{r-1}\right)$$

3- Compute

$$\mathbf{r}_2(\mathbf{y}') = \mathbf{c}_r - \left(A_1^T \mathbf{y}'_1 + A_2^T \mathbf{y}'_2 + \ldots + A_{r-1}^T \mathbf{y}'_{r-1}\right) = \mathbf{c}_r - \sum_{i=1}^{r-1} A_i^T \mathbf{y}'_i.$$

4- Solve $(I + FF^T)\mathbf{u} = \mathbf{r}_2(\mathbf{y}')$, i.e. compute

$$\mathbf{u} = \left(I + \sum_{i=1}^{r-1} F_i F_i^T\right)^{-1} \mathbf{r}_2(\mathbf{y}').$$

5- Compute

$$\mathbf{y}''_i = R_i^{-1} F_i^T \mathbf{u} = R_i^{-1}\left(A_i^T R_i^{-1}\right)^T \mathbf{u}.$$

6- $\mathbf{y} = \mathbf{y}' + \mathbf{y}''$.

The computation of **u** dominates the updating algorithm as it involves the solution of a square system. However, FF^T is a $(m_r x m_r)$-matrix, and assuming that the problem is weakly linked, computing **u** should not be expensive.

Note: Having the QR factor of B^T and orthogonal matrix Q available, they can be used to solve problems with the same matrix and a different RHS.

6.5.2 Search Direction and Lower Bound on z*

The direction of search involves the dual solutions provided by Algorithm 6.2. According to the partitioning considered earlier, it can be computed in a modular way as follows.

$$\mathbf{p}^T = (\mathbf{p}_1^T, \mathbf{p}_2^T, ..., \mathbf{p}_r^T),$$

where

$$\mathbf{p}_i = \left(\mathbf{c}_i - \mathbf{y}_i^T B_i\right)^T - \frac{\mathbf{c}^T\mathbf{x} - z}{n+1}\mathbf{e}_{n_i}, \quad i = 1, \ldots, r-1, \tag{6.5.8}$$

$$\mathbf{p}_r = \left(\mathbf{c}_r - \sum_{i=1}^{r-1}\mathbf{y}_i^T A_i\right)^T - \frac{\mathbf{c}^T\mathbf{x} - z}{n+1}\mathbf{e}_{n_r}. \tag{6.5.8'}$$

The norm of **p** is

$$\|\mathbf{p}\| = \sqrt{\sum_{i=1}^{r}\sum_{j=1}^{n_i} p_{ij}^2}\,. \tag{6.5.9}$$

A lower bound on the minimum objective value of (6.5.2) is provided by $\mathbf{y}^T\mathbf{b}$, from the duality theorem of linear programming. Let

$$\mathbf{U}^T = (A')^{\dagger}(0, -1)^T \tag{6.5.10}$$

$$\mathbf{V}^T = (A')^{\dagger}(\mathbf{c}, 0)^T \tag{6.5.11}$$

where -1 and zero are substituted to the last entry of the cost vector

$$\mathbf{c}^T = (\mathbf{c}_1^T, \mathbf{c}_2^T, ..., \mathbf{c}_r^T D_r, -z),$$

of (6.5.2). Vectors **U** and **V** can be computed using Algorithm 6.2 without factorizing B^T again as its QR factor is available from the computation of **y**. Now the dual vector **y** can be written as

$$\mathbf{y} = \mathbf{U} + z\mathbf{V} \tag{6.5.12}$$

or

$$\mathbf{y}_i = \mathbf{U}_i + z\mathbf{V}_i, \quad i=1, ..., r-1 \tag{6.5.13}$$

y being the dual solution at iteration k, say. At iteration k+1, the new dual vector we are looking for should primarily satisfy the constraints of the dual problem corresponding to (6.5.2) and allow a good improvement in the dual objective value. However, as in the dual version of the projective algorithm given in Chapter 4, we start first by determining the values of z that guarantee feasibility of **y**, from (6.5.10), then choose as z' the one that constitutes the best bound on the minimum objective value.

$$Z = \{ z \in R : \mathbf{c} - \mathbf{y}A' \geq 0\} = \{-U_i / V_i,\ i = 1, ..., m\}. \qquad (6.5.14)$$

Because of the parallel nature of the algorithm, we can write for each $\mathbf{y}_i$,

$$Z_i = \{-U_{ij} / V_{ij},\ j = 1, ..., m_i;\quad i = 1, ..., r-1\}. \qquad (6.5.14')$$

Thus $\quad Z = \cup_i Z_i,\ i = 1, ..., r-1$.

If $Z = \varnothing$ then the best bound is taken as $z' = -\infty$. Otherwise $z' = \text{Sup } Z$, (see Chapter 4).

Having a lower bound z' on z*, and the corresponding dual feasible solution $\mathbf{y}(z')$ given by (6.5.2), the new dual solutions and objective value are $\mathbf{y}^{(k+1)}$ and $z^{(k+1)}$ obtained as follows.

if $z^{(k)} < \mathbf{y}(z')\mathbf{b}$ **then** $\mathbf{y}^{(k+1)} = \mathbf{y}(z')$ **and** $z^{(k+1)} = \mathbf{y}(z')\mathbf{b}$,

else $\mathbf{y}^{(k+1)} = \mathbf{y}^{(k)}$ **and** $z^{(k+1)} = z^{(k)}$ **endif.** (6.5.15)

We have shown in this section and the previous one that the inherent parallelism of block-angular LP problems may be exploited in the different compartments of a dual version of Karmarkar algorithm. In the next section we shall present such an algorithm.

6.5.3 A New Partitioning Algorithm for LP

Algorithm 6.3

0- Initialization: Set $\mathbf{x}^{(0)}$ to an interior feasible point, $\mathbf{y}^{(0)}$ to 0, $z^{(0)}$ to a lower bound on optimum objective value z* and ε to an arbitrarily small value.

1- **if** $\left(\sum_{i=1}^{r} \mathbf{c}_i^T \mathbf{x}_i^{(k+1)} - z^{(k+1)}\right) \leq \varepsilon$ **then stop.**

2- $D_i = \mathbf{x}_i^{(k)},\ i = 1, ..., r.$

3- Compute **y**, **U** and **V** applying Algorithm 6.2.

4- Find z' from (6.5.14), and $\mathbf{y}^{(k+1)}$ and $z^{(k+1)}$ from (6.5.15).

5- Compute the solution in the transformed space as

$$x'_i = e_{n_i} - \alpha \frac{p_i}{\|p\|}, \; i = 1, \ldots, r,$$

where α is found by one of the techniques described in Chapter 4.

6- Compute primal solution in the original space as

$$x_i^{(k+1)} = \frac{x'_i}{x'_{rn_r}}, \; i = 1, \ldots, r.$$

7- $k = k + 1$, go to 1-.

6.6 Extending the Partitioning Algorithm to Staircase Structure

We have seen in section 6.2 that staircase structure is amenable to block-angular structure (Fig 6.4). However, mass sparsity remains in the linking block after transformation. In this section we will look at the way Algorithm 6.3 can be extended to exploit this sparsity .

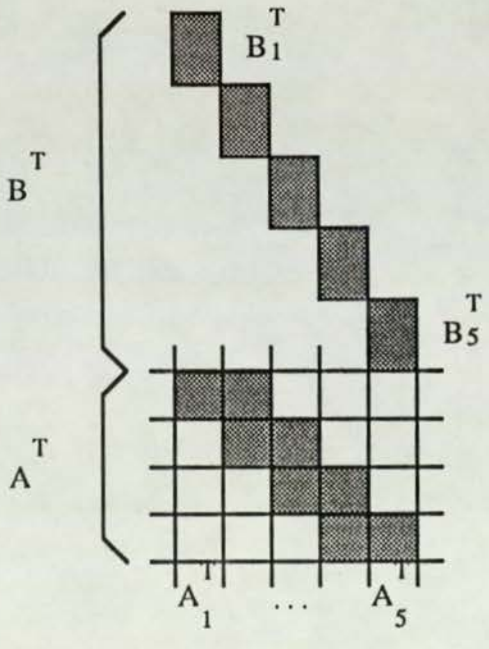


Fig 6.6 Partitioning of the Linking Block of the Matrix Derived from a 5-Stage Staircase LP Problem

The obvious approach is to further partition the linking block in order to isolate the zero sub-blocks. Consider the system (6.5.5) whose matrix has the structure of Fig 6.4. The matrix in the case of a 5-block problem is represented in Fig 6.6.

We can see that any submatrix A_i^T has at most two non-zero blocks and at least one non-zero block. When the original staircase problem has a high number q of stages, then every submatrix A_i^T has between q-2 and q-3 zero blocks. For instance, if q = 10, the number of zero blocks is 7 or 8.

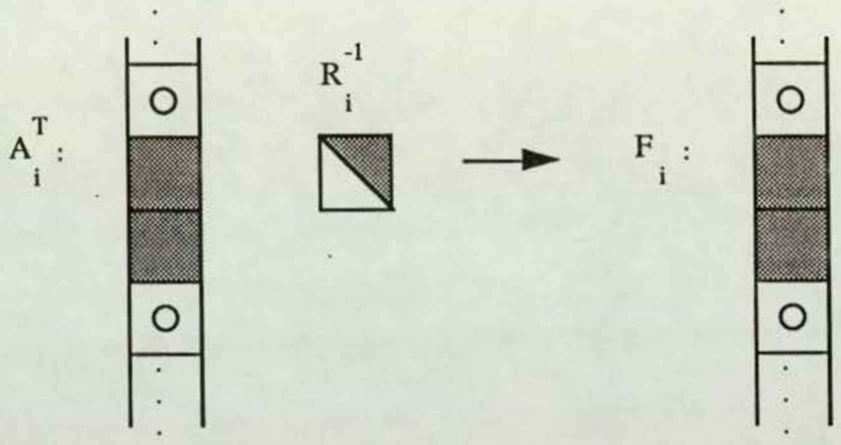


Fig 6.7 Structures of A_i^T and F_i.

Let every submatrix A_i^T be partitioned into q-1 blocks, A_{i1}^T, A_{i2}^T, ..., A_{iq-1}^T, if the original staircase LP problem has q stages. As depicted in Fig 6.7, F_i has the same structure as A_i^T. Thus, F_i can be written as $(F_{i1}^T \ F_{i2}^T \ \ldots \ F_{iq-1}^T)^T$. The submatrices F_{ii-1} and F_{ii} are the nonzero blocks. Because F_1 and F_{q-1} have each only one nonzero block, then F_{i0} and F_{iq} do not exist. According to this partitioning Step 2 of Algorithm 6.2 can proceed as follows.

```
for i = 1, ..., q do
for j = 1, ..., q-1 do
      if j = i -1 or j = i then
            $F_{ij} = A_{ij}^T R_i^{-1}$
      else
            $F_{ij} = 0$
      endif
```

The general matrix F is depicted below.

$$F = \begin{pmatrix} F_{11} & F_{21} & \dots & 0 \\ 0 & F_{22} & & 0 \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ 0 & & \dots & F_{q\text{-}1q\text{-}1} \end{pmatrix}$$

Step 3 of Algorithm 6.2 can, in the same way as above, be optimized by considering only the nonzero blocks of A_i^T when computing $\mathbf{r}_2(\mathbf{y}')$.

6.7 Computational Experience and Conclusion

In this section we report some numerical results on the performance of the partitioning algorithm on two sets of LP problems also solved with a code of the dual projective algorithm presented in Chapter 4. The tests were carried out in MATLAB on a Macintosh SE/30 as well as in CTRLC on a VAX 8650. The times recorded for a comparison purpose were given, in seconds, by the function ETIME of MATLAB.

6.7.1 Tests on Random Generated Structured LP Problems

The test problems for Algorithm 6.3 were based on those used by Mangasarian (1981). These problems were generated as follows. The matrix A was fully dense with random elements a_{ij} uniformly distributed in the interval [-100, 100]. The right hand side was chosen such that

$$b_i = \begin{cases} \sum_{j=1}^{n} a_{ij} & \text{if } a_{ij} > 0, \\ -1 + 2\sum_{j=1}^{n} a_{ij} & \text{if } \sum_{j=1}^{n} a_{ij} \le 0, i = 1, \dots, m \end{cases}$$

and the cost vector such that

$$c_j = \sum_{i \in J} a_{ij}, \ \text{where } J = \left\{ i \mid \sum_{j=1}^{n} a_{ij} > 0 \right\}, \ j = 1, \ldots, n.$$

b and **c** so chosen make point **e** primal optimal.

In our tests the above problems constitute blocks linked with a set of columns also randomly generated, to form the structured problems. However, vectors **b** and **c** of the structured problems comply with their above definitions, with the linking columns taken into account. Thus point **e** is primal optimal. A sample randomly generated 2-block LP problem is given in Fig 6.8.

The results of these tests are recorded in Table 6.1. The two last columns under the headings Etime/P and Etime/NP correspond respectively to the performance of the partitioning algorithm (marked P) and that of Algorithm 4.2 which is a non-partitioning algorithm (marked NP).

Problems	Rows	Cols	Blocks	Links	Etime/P	Etime/NP	Time Ratio
Mang4	18	35	4	2+1	31.12	110.37	3.55
	18	36	4	3+1	33.08	113.77	3.44
	18	37	4	4+1	35.08	119.73	3.41
Mang6	29	57	6	2+1	55.83	418.58	7.45
	29	59	6	4+1	59.10	437.23	7.40
	29	61	6	6+1	63.09	545.90	8.65
Mang8	40	76	8	2+1	89.60	928.87	10.37
	40	78	8	4+1	95.47	972.85	10.19
	42	85	8	6+1	144.08	1232.30	8.55
	42	87	8	8+1	156.92	1259.40	8.03
Mang10	43	82	10	2+1	95.75	1127.20	11.77
	43	84	10	4+1	102.43	1150.20	11.23
	43	86	10	6+1	110.67	1235.10	11.16
	43	88	10	8+1	119.52	1299.20	10.87
	43	88	10	10+1	129.62	1677.30	12.94

Table 6.1 Numerical Results on Randomly Generated Problems

```
c:  105.35 166.58 174.31 279.19 120.83 | 261.10 321.04 |   b:
A:   21.13   0.87  45.24               |  61.34  65.37 | 193.97
      8.23  80.96  80.74               |  27.48  48.98 | 246.42
     75.98  84.74  48.31               |  88.06  77.40 | 374.52
                          96.26  74.69 |  26.12  11.67 | 208.75
                          99.33   3.77 |  24.02  62.49 | 189.63
                          83.60  42.36 |  34.04  55.10 | 215.11
                                        linking  Cols.
```

Fig 6.8 A Sample Randomly Generated Problem with
Entries of A Uniformly Distributed in [0, 100]

6.7.2 Tests on Non-Randomly Generated LP Problems

Problems which are not randomly generated were also solved with Algorithm 6.3 and Algorithm 4.2. Xnut1, Xnut2 and HL221 are text book problems. Little4 and Big8 are constructed by us using text book problems for each block and linking them by additional variables. Big8 is a 8-block problem, also solved as a 2-block and 4-block problem. AutoCo1 and AutoCo2 are variants of the same problem given in ICL 1900 Series, LP Mark3 User Guide, 1973. We noticed that the problem has favourable structure after reordering its constraints. Depending on the reordering it can be solved as a 2-block or a 4-block problem. The results of the tests are recorded in Table 6.2.

Problems	Rows	Cols	Blocks	Links	Etime/P	Iter.	Etime/NP	Iter.	Time Ratio
Xnut1	5	11	2	2+1	30.80	11	29.43	11	0.96
Xnut2	4	10	2	1+1	15.31	7	17.60	8	1.25
HL221	8	18	3	2+1	44.10	10	56.92	10	1.29
Little4	12	24	4	2+1	60.17	11	117.15	11	1.95
AutoCo1	32	69	2	7+1	733.05	15	2468.98	16	3.37
AutoCo2	33	71	4	12+1	451.35	15	2468.98	16	5.47
			2	2+1	201.12			16	6.13
Big8	25	53	4	2+1	181.20	16	1231.90	16	6.80
			8	2+1	202.47			16	6.08

Table 6.2 Results from Partitioning (P) and Nonpartitioning (NP) Algorithms
on Nonrandomly Generated Problems

6.7.3 Conclusion

It appears from these results that the partitioning algorithm does cut down the overall work of the dual Karmarkar algorithm. The partitioning algorithm is between 3 to 12 times faster than Karmarkar's dual variant, except for the 3 first problems of Table 6.2. This is shown in the last columns of Table 6.1 and Table 6.2 under the heading "Time Ratio". The bad performance of the partitioning algorithm on the small problems is justified by their large density; the work involved in the updating of the dual solutions is substantial. This work becomes negligible only when the density of the problem is low, which is a characteristic of large structured problems. The updating process also depends on the number of linking columns. When this number is large, the accuracy of the solution suffers and the updating process becomes costly. Indeed, it was assumed that the blocks are weakly linked. It still remains to know how the link affects the overall work involved and in what proportions.

Note also that a simplex based decomposition algorithm would not outperform the standard simplex on problems of the same size as the ones used in our tests. Indeed, the standard simplex is most of the time more effective, in terms of CPU time, than any of its decomposition variants even on very large problems, [Fourer, 1982]. Our partitioning algorithm, on the other hand, will perform even better, on large structured problems, than the straight Karmarkar's algorithm.

An important fact that should be mentioned is that the algorithm is not fundamentally a decomposition one in the sense that no linear programming subproblems are solved. Indeed, the partitioning is oriented towards the least squares problem which is solved to find the dual variables. The theoretical complexity of the partitioning algorithm is, therefore, that of Algorithm 4.2. A more fundamental partitioning is considered in Algorithm 6.1 and its general form, for which we showed that Karmarkar's dual algorithm is valid as a solution strategy. However, it is not clear how advantageous this approach is. Recently, Todd (1988b) attempted to build such an algorithm based on the

Dantzig-Wolfe decomposition principal. He concluded that the method is unlikely to provide substantial improvements to the solution of structured LP.

We end up this chapter by showing that, at least from the theoretical point of view, interior point methods may be advantageous in the context of decomposition.

The concept of volume is at the basis of interior point algorithms. In Karmarkar's method it is present in the form of a potential function, and "through" it reduction in the objective function is achieved. More specifically, the potential function is equivalent to the volume of an ellipsoid generated in the dual space at each iteration (Ye, 1987). As the potential function is important for the convergence of the algorithm, so is the volume of the ellipsoid, because optimality is achieved at the same time for both the primal problem and its dual. It follows that starting with a smaller volume can be advantageous. First, as in the case of the ellipsoid algorithm, it takes less iterations to reduce a smaller volume ellipsoid to near zero; secondly, the work involved is proportional to the size of the problem and hence the volume of the starting ellipsoid.

We know that the volume grows exponentially with space dimension [Le Tellier, 1984]. To avoid starting with a large volume ellipsoid, it is profitable to work in lower dimensional spaces. This can be achieved by decomposing the problem into subproblems defined in lower dimensional spaces.

Consider the linear functional $F(\mathbf{x}) = \mathbf{c}^T\mathbf{x}$, such that

$$\mathbf{c}^T\mathbf{x} = c_1x_1 + \ldots + c_{n1}x_{n1} + c_{n1+1}x_{n1+1} + \ldots + c_nx_n,$$

which can be seen as the sum of two functionals $f_1(\mathbf{x}_1)$ and $f_2(\mathbf{x}_2)$ whose parameters are obtained from the partitioning of the vectors **c** and **x** as follows

$$\mathbf{c} = \begin{bmatrix} \mathbf{c}_1 \\ \mathbf{c}_2 \end{bmatrix}, \quad \text{and } \mathbf{x} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix},$$

where $\mathbf{x} \in R^n$, $\mathbf{x}_1 \in R^{n1}$, $\mathbf{x}_2 \in R^{n2}$ and $n_2 = n - n_1$. Let us write

$$f_1(\mathbf{x}_1) = \mathbf{c}_1^T\mathbf{x}_1 \text{ and } f_2(\mathbf{x}_2) = \mathbf{c}_2^T\mathbf{x}_2.$$

The logarithmic potential function of Karmarkar as applied to F, f_1 and f_2 leads to:

$$P(\mathbf{x}) = \sum_{i=1}^{n} \ln\left(\frac{\mathbf{c}^T\mathbf{x}}{x_i}\right),$$

$$p_1(\mathbf{x}_1) = \sum_{i=1}^{n_1} \ln\left(\frac{\mathbf{c}_1^T\mathbf{x}_1}{x_i}\right),$$

and

$$p_2(\mathbf{x}_2) = \sum_{i=n_1+1}^{n} \ln\left(\frac{\mathbf{c}_2^T\mathbf{x}_2}{x_i}\right).$$

These potential functions will arise when we consider an LP with objective function F(x) and its subproblems with objective functions f_1 and f_2, if we partition the problem into two. One way of seeing the advantage of decomposition is to show that the potential function of the full problem is actually larger than the sum of the potential functions corresponding to the subproblems.

Lemma 6.1 : $p_1(\mathbf{x}_1) + p_2(\mathbf{x}_2) < P(\mathbf{x})$.

Proof:

$$p_1(\mathbf{x}_1) + p_2(\mathbf{x}_2) = n_1 \ln\left(\mathbf{c}_1^T\mathbf{x}_1\right) + (n - n_1)\ln\left(\mathbf{c}_2^T\mathbf{x}_2\right) - \sum_{i=1}^{n}\ln(x_i),$$

$$P(\mathbf{x}) = \sum_{i=1}^{n}\ln\left(\mathbf{c}^T\mathbf{x}\right) - \sum_{i=1}^{n}\ln(x_i),$$

$$= \left(n - n_1 + n_1\right)\ln\left(\mathbf{c}^T\mathbf{x}\right) - \sum_{i=1}^{n}\ln(x_i),$$

$$= n_1 \ln\left(\mathbf{c}^T\mathbf{x}\right) + (n - n_1)\ln\left(\mathbf{c}^T\mathbf{x}\right) - \sum_{i=1}^{n}\ln(x_i).$$

As $\mathbf{c}^T\mathbf{x} > 0$, $\mathbf{c}_1{}^T\mathbf{x}_1 > 0$ and $\mathbf{c}_2{}^T\mathbf{x}_2 > 0$, because the problems are solved in the transformed space and their objective functions must be positive then

$$\ln(\mathbf{c}_1{}^T\mathbf{x}_1 + \mathbf{c}_2{}^T\mathbf{x}_2) > \ln(\mathbf{c}_1{}^T\mathbf{x}_1)$$

and

$$\ln(\mathbf{c}_1{}^T\mathbf{x}_1 + \mathbf{c}_2{}^T\mathbf{x}_2) > \ln(\mathbf{c}_2{}^T\mathbf{x}_2).$$

Thus

$$n_1 \ln\left(\mathbf{c}^T\mathbf{x}\right) + (n - n_1) \ln\left(\mathbf{c}^T\mathbf{x}\right) > n_1 \ln\left(\mathbf{c}_1^T\mathbf{x}_1\right) + (n - n_1) \ln\left(\mathbf{c}_2^T\mathbf{x}_2\right).$$

It is then clear that $p_1(\mathbf{x}_1) + p_2(\mathbf{x}_2) < P(\mathbf{x})$. Q.E.D.

Chapter 7

Implementation of the Projective Algorithm and Computational Experience

7.1 Introduction

In Chapter 6 we have shown how Karmarkar's algorithm can be adapted to make use of favourable structures (block-angular and staircase). In those situations whole blocks or submatrices are sparse. This made it possible to specialize Karmarkar's algorithm for such structures and exploit the sparsity blockwise. However, sparsity often occurs in less regular patterns.

In this chapter we shall describe how a variant of the projective algorithm was implemented to solve Klee-Minty and Hilbert type problems as well as real world LP problems. We shall examine the performance of the algorithm in conjunction with the form in which the problem is handled and the technology for least squares described in Chapter 4. Issues related to the coding of the algorithm in Fortran 77 such as data structures and input data (MPS format) will be discussed.

Two implementations of the algorithm are mainly considered: LPKAR1 and LPKAR2. In LPKAR1 it is assumed that the optimum objective value z^* of the problem is *a priori* known while in LPKAR2 the assumption is dropped. The canonical form in which the problem is handled differs for both cases. The first code works on a canonical form we suggest, in which the objective function is included as a constraint with z^* as its right-hand side. It works in a single phase and is a primal only method. The second code, LPKAR2 works on canonical form 3 described in Chapter 3. It is a two-phase method and generates dual solutions.

Aspects of sparsity exploitation such as ordering and partitioning will be discussed and numerical results obtained using LPKAR1 and LPKAR2 will be presented.

7.2 A Variant of the Karmarkar Algorithm

The original algorithm of Karmarkar (1984a, b) with the indications he gave to set up the problem in canonical form was implemented and did not perform as efficiently as was thought. The difficulties encountered were partly due to inflation of problem size after primal-dual combination to put the problem in the required form, and also to ill-conditioning in the projection matrix which involves inverting a cross-product matrix of the form BB^T. However, this implementation provided valuable insights to the properties of the algorithm and its behaviour. The steplength, for instance does not have to be 1/4 as suggested by Karmarkar to insure convergence. Indeed, values closer to 1.0 and even larger, as was seen in Chapter 3, greatly improve the speed of convergence. We also noted that the number of iterations is generally low, which confirmed Karmarkar's claim. To the light of these observations and experience, we present a variant of the algorithm on which our codes were based.

Algorithm 7.1

The following algorithm handles problems in standard form, i.e. $\{\mathbf{x} \in R^n \mid \min \mathbf{c}^T\mathbf{x}, A\mathbf{x} = \mathbf{b}, \mathbf{x} \geq 0\}$. Assume that an interior feasible point is at hand, then

1- Transform problem into the form

$$\begin{aligned} &\min \mathbf{c}'^T\mathbf{x}' \\ &\text{s.t. } A'\mathbf{x}' = 0, \\ &\qquad \mathbf{x}' \geq 0, \end{aligned}$$

where $\mathbf{c}'^T = [\mathbf{c}^T, -z]$, $A' = [A, -\mathbf{b}]$ and $\mathbf{x}'^T = [\mathbf{x}^T, \ 1]$.

2- Initialization
$k = 0$, $\varepsilon = 1.0E\text{-}06$, $z = M$, where M is a large value, $D = \text{diag}(\mathbf{x}^{(0)}, 1)$.

```
3- if $\mathbf{c}'^T\mathbf{x}^{(k)} < \varepsilon$ stop

4- Compute $\mathbf{y} = (DA'^T)^{\dagger}D\mathbf{c}'$

5- Compute $\mathbf{p} = D\mathbf{c}' - (DA'^T)\mathbf{y} - (\mathbf{c}^T\mathbf{x}^{(k)}/n)\mathbf{e}$

6- Normalize $\mathbf{p}$, i.e. $\mathbf{p}' = \mathbf{p}/\,||\mathbf{p}||$

7- $\mathbf{x}'^{(k+1)} = \mathbf{e} - \alpha\mathbf{p}'$, where $\alpha$ is the steplength

8- Compute $\mathbf{x}^{(k+1)} = \dfrac{D\mathbf{x}'^{(k+1)}}{\mathbf{e}^T D\mathbf{x}'^{(k+1)}}$

9- Compute $\mathbf{x}^{(k+1)} = \dfrac{\mathbf{x}^{(k+1)}}{x_n^{(k+1)}}$

10- $D = \mathrm{diag}(\mathbf{x}^{(k+1)})$, $\mathbf{c}'^T = [\mathbf{c}^T, -\mathbf{c}^T\mathbf{x}^{(k+1)}]$, $k = k + 1$, go to 3-
```

Algorithm 7.1 differs from the original Karmarkar's algorithm and the variant described by Lustig (1985) in the way the projection matrix and the search direction are computed. This approach is more suitable as the sparsity of the original problem is only slightly disturbed by adjoining the column corresponding to the right-hand side. When optimum objective value z* is available, it can be shown that Algorithm 7.1 retains the polynomial complexity of Karmarkar's algorithm. On the other hand, if z* is not supplied, then updates of z, i.e. $\mathbf{c}^T\mathbf{x}^{(k+1)}$, after each iteration can be used instead. However, while it is possible to establish that $\mathbf{c}^T\mathbf{x}^{(k+1)} < \mathbf{c}^T\mathbf{x}^{(k)}$, which implies $\mathbf{p}'$ is a descent direction, it is difficult to show whether the algorithm is polynomial in time. To make sure that Algorithm 7.1 has polynomial complexity while dealing with unknown optimum objective value, the strategy that finds ever better lower bounds on z*, already presented in Chapters 4 and 6, can be used.

7.3 Implementations of Algorithm 7.1

LPKAR1 and LPKAR2 are two different ways of applying Algorithm 7.1 to a linear programming problem depending on assumptions made and information available about the problem. In LPKAR1 sparsity-exploitation is the central issue. This will involve symbolic factorization, ordering and updating techniques. In LPKAR2 we investigate the possibility of solving LP problems without supplying z* and by using the Moore-Penrose pseudo-inverse to solve the least squares of step 4 in Algorithm 7.1.

First, let us look at the form under which the problem is handled by LPKAR1.

Assuming that z* is available, it is possible to transform the original problem in standard form into the following equivalent form accepted by Algorithm 7.1.

$$\begin{aligned} &\min \lambda \\ &\text{s.t. } \mathbf{Ax} - \mathbf{b} - (\mathbf{Ae} - \mathbf{b})\lambda = 0 \\ &\qquad \mathbf{c}^T\mathbf{x} - z^* \quad + \quad 0\lambda = 0 \qquad\qquad (7.3.1) \\ &\qquad\qquad\qquad \mathbf{x}, \lambda \geq 0, \end{aligned}$$

where λ is an artificial variable.

This problem is in R^{n+2} and admits $\mathbf{e}_{n+2}$ as an interior feasible point. Algorithm 7.1 is readily applicable. One advantage this form of the problem offers is that the optimum solution is obtained in one phase. Indeed, when λ is reduced to zero, the resulting point $\mathbf{x}^*$ satisfies the constraints in the original space, i.e. R^n, and also the extra constraint $\mathbf{c}^T\mathbf{x} - z^* = 0$. It follows that $\mathbf{x}^*$ is optimum solution.

The other advantage of above canonical form is that the objective vector is totally sparse except for one entry corresponding to the artificial variable λ. In LPKAR1 this sparsity is used to reduce the work in an iteration of Algorithm 7.1. The way this is brought to effect will be shown later. Note that LPKAR1 is a primal only method as the original algorithm of Karmarkar.

In the case of LPKAR2, the problem is handled under canonical form 3 presented in Chapter 2.

7.3.1 Details of LPKAR1

LPKAR1 uses Cholesky method to deal with the least squares problem of step 4 in Algorithm 7.1 augmented with sparsity preservation steps comprising the Nested Dissection Ordering algorithm of George (1982), and a version of the updating technique for least squares of Heath (1984), described in Algorithm 3.1. Symbolic factorization is also used to set up appropriate data structures. With these steps added, Algorithm 7.1 can be described as follows.

Algorithm 7.2

Assume that a feasible point $\mathbf{x}^{(0)}$ is available and that the problem is under canonical form (7.3.1) accepted by Algorithm 7.1.

1- Initialization

$k = 0$, $\varepsilon = 1.0E\text{-}06$, $z = M$, where M is a large value, $D = \mathrm{diag}(\mathbf{x}^{(0)}, 1)$.

2- **if** $\mathbf{c}'^T\mathbf{x}^{(k)} < \varepsilon$ **stop**

3- Compute **y** as follows

a) Remove the full rows of matrix DA'^T

b) Find symbolic representation or adjacency structure of $A'D^2A'^T$

c) Find a permutation matrix P using the Nested Dissection Ordering Algorithm (see Appendix C)

d) Find a symbolic factorization of $PA'D^2A'^TP^T$, i.e. find the non-zero structure of the Cholesky factor L of the cross-product

e) Fill the structure with the actual numerical values by applying Cholesky or Givens method

f) Apply a forward and a back substitution to get the solution $\mathbf{y}'$ to the incomplete least squares problem

g) Apply inverse ordering to get incomplete solution in the original ordering

h) Add effect of the removed rows to the solution **y**' by updating it using Algorithm 7.1 resulting in **y**

4- Compute $\mathbf{p} = \mathrm{D}\mathbf{c}' - (\mathrm{DA'}^T)\mathbf{y} - (\mathbf{c}'^T\mathbf{x}'^{(k)}/n)\mathbf{e}$

5- Normalize **p**, i.e. $\mathbf{p}' = \mathbf{p}/\,\|\mathbf{p}\|$

6- $\mathbf{x}'^{(k+1)} = \mathbf{e} - \alpha\mathbf{p}'$, where α is the steplength

7- Compute $\mathbf{x}^{(k+1)} = \dfrac{\mathrm{D}\mathbf{x}'^{(k+1)}}{\mathbf{e}^T\mathrm{D}\mathbf{x}'^{(k+1)}}$

8- Compute $\mathbf{x}^{(k+1)} = \dfrac{\mathbf{x}^{(k+1)}}{x_n^{(k+1)}}$

9- $\mathrm{D} = \mathrm{diag}(\mathbf{x}^{(k+1)})$, $\mathbf{c}'^T = [\mathbf{c}^T, -\mathbf{c}^T\mathbf{x}^{(k+1)}]$, $k = k + 1$, **go to 2-**

Although sparse techniques such as updating and ordering were dealt with respectively in Chapter 3 and Appendix C, it remains to clarify the procedures involved in steps b, d, e and f of above algorithm and also the data structures used in its FORTRAN 77 code.

7.3.1.1 Adjacency Structure of $A'D^2A'^T$

Ordering algorithms are graph algorithms known to be sensitive to the way the graphs are represented. In our case, to proceed with the reordering of the cross-product $A'D^2A'^T$ and set up the data structures for the Cholesky factor, it is essential to efficiently store its nonzero structure and retrieve adjacency relations. Thus, the adjacency structure of a matrix is the representation of its graph.

Let G(x, E) be a graph with N nodes. The adjacency list of a node $x \in X$ is a list containing all adjacent nodes to x and the structure of G is the set of such lists for all its nodes. The implementation of the structure is done by storing the adjacency lists sequentially in a one dimensional array ADJNCY along with an index vector XADJ of length N+1 containing pointers to the beginning of the lists in ADJNCY. The extra entry XADJ(N+1) points to next available location in ADJNCY [George & Liu, 1981].

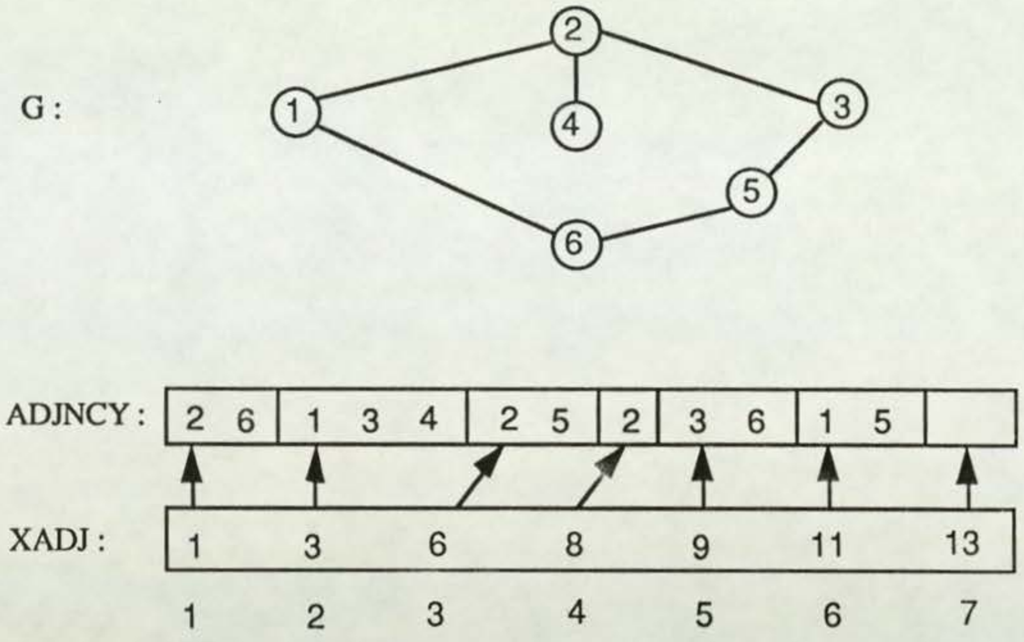


Fig 7.1 Adjacency Structure of a Graph

The attractive feature of this approach is that the structure of $(A'D)(A'D)^T$ is found without explicitly forming the cross-product.

7.3.1.2 Symbolic Factorization and Storage Scheme

After applying the nested dissection ordering algorithm, a permutation matrix P is returned which will help reduce fill-in during the factorization process of $PA'D^2A'^TP^T$. However, before proceeding with the actual numerical factorization, a simulation of it, or symbolic factorization is carried out to set up the data structures to contain the Cholesky factor in sparse form. The advantage of this approach is that the data structures are static; thus set up once for all, as the structure of the matrix does not change from iteration to iteration. Note that at this stage the numerical values of the Cholesky factor are not explicitly computed.

The data structures returned by the symbolic factorization are a sparse storage scheme known as the compressed scheme of Sherman, cited in [George & Liu, 1981]. The scheme has a main one-dimensional storage array LNZ which will contain all nonzero entries in the lower triangular factor of $PA'D^2A'^TP^T$ column-wise, an INTEGER vector NZSUB which will hold the row subscripts of the nonzeros, and an index vector XLNZ whose entries are pointers to the beginning of nonzeros in each column in LNZ. In addition, an index vector XNZSUB is also used to hold pointers to the start of row subscripts in NZSUB for each column. The diagonal elements are stored separately in vector DIAG.

7.3.2 Input Data for Codes of Algorithm 7.2

Real world problems usually are stored in MPS format which is standard in industry. The format, mainly, consists of three sections: constraints type, constraints entries stored column-wise including the cost vector and the right-hand side. Other sections may be added such as bounds on variables and free constraints.

Example 7.1:

```
NAME          PROB1
ROWS
 N  FOB00001
 G  ROW00001
 G  ROW00002
 G  ROW00003
COLUMNS
    COL00001  FOB00001     -5.000000  ROW00001     -2.000000
    COL00001  ROW00002     -4.000000  ROW00003     -3.000000
    COL00002  FOB00001     -4.000000  ROW00001     -3.000000
    COL00002  ROW00002     -1.000000  ROW00003     -4.000000
    COL00003  FOB00001     -3.000000  ROW00001     -1.000000
    COL00003  ROW00002     -2.000000  ROW00003     -2.000000
RHS
    RHS       ROW00001     -5.000000  ROW00002    -11.000000
    RHS       ROW00003     -8.000000
ENDATA
```

The problem under MPS format is read into a one-dimensional array ALIST of length NZ, which is a column-wise storage of the problem matrix. Slack variables are added according to the type of constraints encountered as well as the two columns, **-b** and -(Ae - **b**) required by the canonical form. ALIST is accompanied with two INTEGER vectors, ICOL and IT, with lengths NZ and N+1, N being the number of total variables in the canonical form. ICOL contains the row subscript of each nonzero in ALIST, while IT contains pointers to the beginning of each column.

Our implementation requires that we repeatedly form the matrix $A'D^2A'^T$ as D changes from iteration to iteration. Thus, to avoid searching for the rows of A' in ALIST, we preferred to store again the matrix row-wise. This may seem inefficient regarding space. However, it makes sense from the time point of view. Consequently, we have another trio of vectors RA(NZ), IA(NZ) and NA(M+1) containing matrix A' row-wise. These arrays are filled in once only by performing a fast sparse-matrix transposition after ALIST, ICOL and IT have been constructed.

7.3.3 Computational Experience

Problem	Original Form		Canonical Form				z*
	Rows	Cols	Rows	Cols	Nonzeros	Density	
Chvtl1	16	11	17	28	142	29.83	-14021.04
Chvtl2	17	13	18	32	114	19.79	-273382.1
Alfaut	38	33	39	72	301	10.72	-12233742
RandD	39	15	40	56	396	17.68	-9474.4845
Scsd1	77	760	78	762	3268	5.43	8.666667
Scagr7	129	140	130	187	782	3.22	-2331390
Scsd6	147	1350	148	1352	5824	2.91	50.50000
Sc205	205	203	206	319	911	1.39	-52.20206
Sctap1	300	480	301	662	2688	1.35	1412.250
Scfxm1	330	457	331	602	3203	1.61	18416.76
Scagr25	471	500	472	673	2852	0.90	-14753433

Table 7.1 Test Problems Statistics

LPKAR1 was tested on the problems listed in Table 7.1 whose origins are as follows:

Chvtl1 and Chvtl2, respectively, are a farm planning LP model and a case study in forestry described in Chvátal (1983). Alfaut and RandD were borrowed from ICL LP3 manual (1973). The remaining problems are standard test problems supplied to us by Dr.

Etienne Loute of the Catholic University of Louvain, Belgium and described in Ho and Loute (1981).

The results reported below (Table 7.2 through 7.5) concern the performance of Algorithm 7.1 in conjunction with the nested dissection ordering algorithm and the updating algorithm for least squares. Four versions of LPKAR1 were run on all the test problems. The versions differ in the ways sparsity is exploited. Four cases arise:

Case 1 : Ordering and partitioning were not implemented in LPKAR1 (Table 7.2).

Case 2 : The nested dissection ordering algorithm was implemented, but no partitioning was considered (Table 7.3).

Case 3 : The partitioning or updating Algorithm 7.1 was implemented, but no ordering was performed (Tables 7.4).

Case 4 : Both ordering and partitioning were implemented in LPKAR1 (Table 7.5).

Beside the CPU time (in sec.) and the number of iterations taken by the four versions of LPKAR1 on all the test problems, a column containing the number of nonzeros in the Cholesky factor for each problem is included. This column, with the heading "R nonzeros", clearly shows advantages and disadvantages of both ordering and updating techniques.

Problems	R Nonzeros	Iterations	CPU(s)
Chvtl1	136	10	0.23
Chvtl2	153	9	0.20
Alfaut	741	16	1.94
RandD	780	14	2.32
Scsd1	3003	13	66.21
Scagr7	7232	20	35.25
Scsd6	10878	15	264.41
Sc205	20914	23	157.25
Sctap1	45150	28	669.00
Scfxm1	54519	25	797.36
Scagr25	94244	29	1948.77

Table 7.2 Performance of LPKAR1 (Case 1)

Problems	R Nonzeros	Iterations	CPU(s)
Chvtl1	136	10	0.26
Chvtl2	153	9	0.23
Alfaut	741	16	2.12
RandD	780	14	2.26
Scsd1	1390	12	64.29
Scagr7	6230	18	30.25
Scsd6	3167	14	240.45
Sc205	20317	22	157.95
Sctap1	45150	27	672.69
Scfxm1	54047	25	839.53
Scagr25	79994	25	1404.74

Table 7.3 Performance of LPKAR1 (Case 2)

Problems	R Nonzeros	Iterations	CPU(s)
Chvtl1	136	10	0.24
Chvtl2	124	9	0.16
Alfaut	196	16	1.16
RandD	771	15	2.13
Scsd1	1408	13	42.05
Scagr7	1250	18	9.56
Scsd6	2779	14	217.32
Sc205	1574	22	17.92
Sctap1	8286	28	153.56
Scfxm1	12075	25	204.31
Scagr25	4922	28	177.99

Table 7.4 Performance of LPKAR1 (Case 3)

Problems	R Nonzeros	Iterations	CPU(s)
Chvtl1	136	10	0.24
Chvtl2	61	9	0.14
Alfaut	104	17	1.25
RandD	690	14	1.88
Scsd1	1393	12	49.52
Scagr7	1116	19	10.59
Scsd6	3119	14	219.35
Sc205	1507	22	19.39
Sctap1	3736	27	128.90
Scfxm1	6812	26	180.13
Scagr25	4848	25	170.59

Table 7.5 Performance of LPKAR1 (Case 4)

In these experiments, the potential function as well as the objective function λ of the canonical form (7.3.1) are monitored for some of the problems of Table 7.1. These functions are represented in the graphs below. The potential function is the logarithmic function (2.2.1) of Karmarkar.

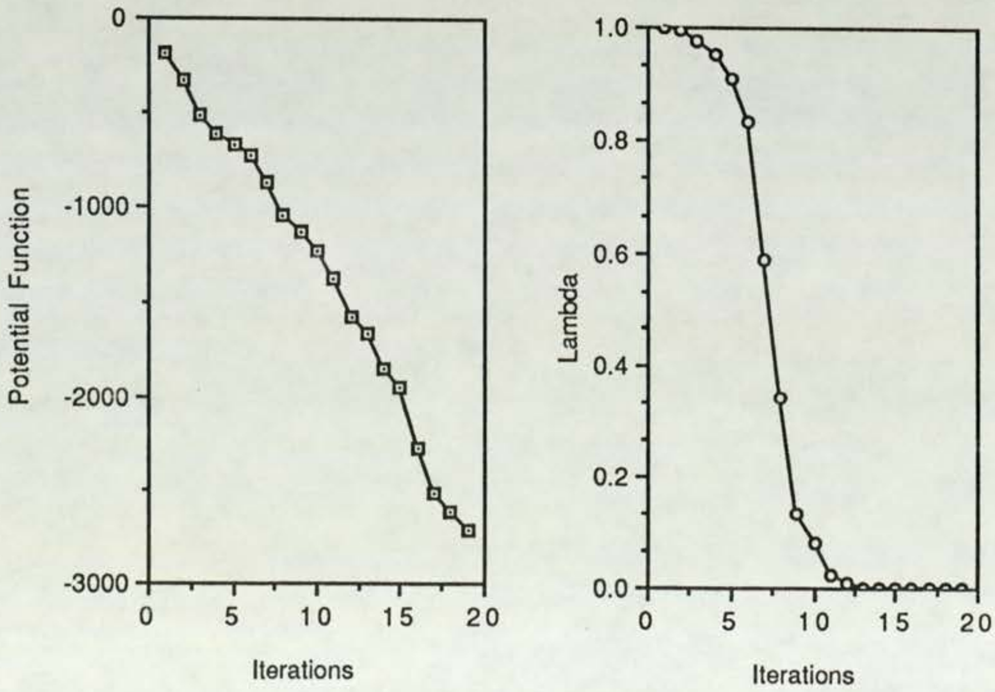


Fig 7.2 Decrease in the Potential and Objective Functions for Problem Scagr7

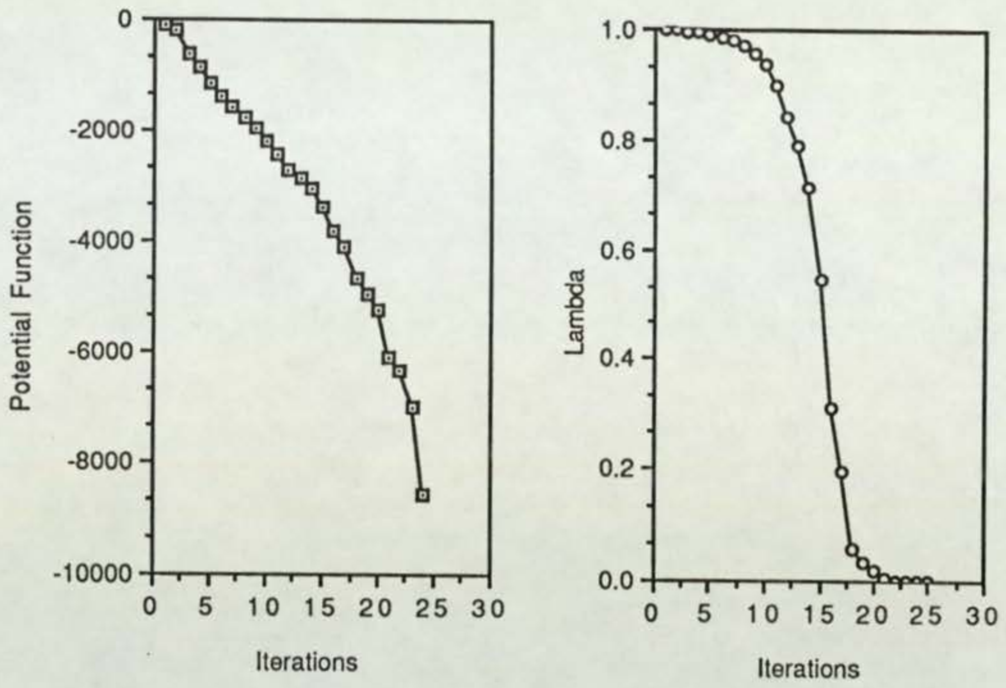


Fig 7.3 Decrease in the Potential and Objective Functions for Problem Scagr25

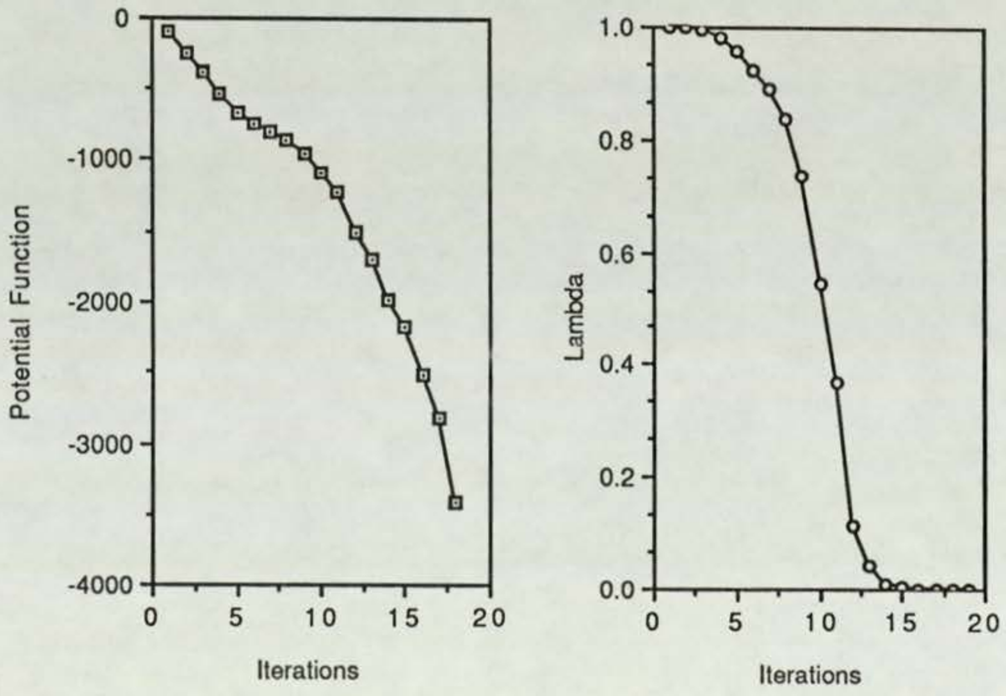


Fig 7.4 Decrease in the Potential and Objective Functions for Problem Sc205

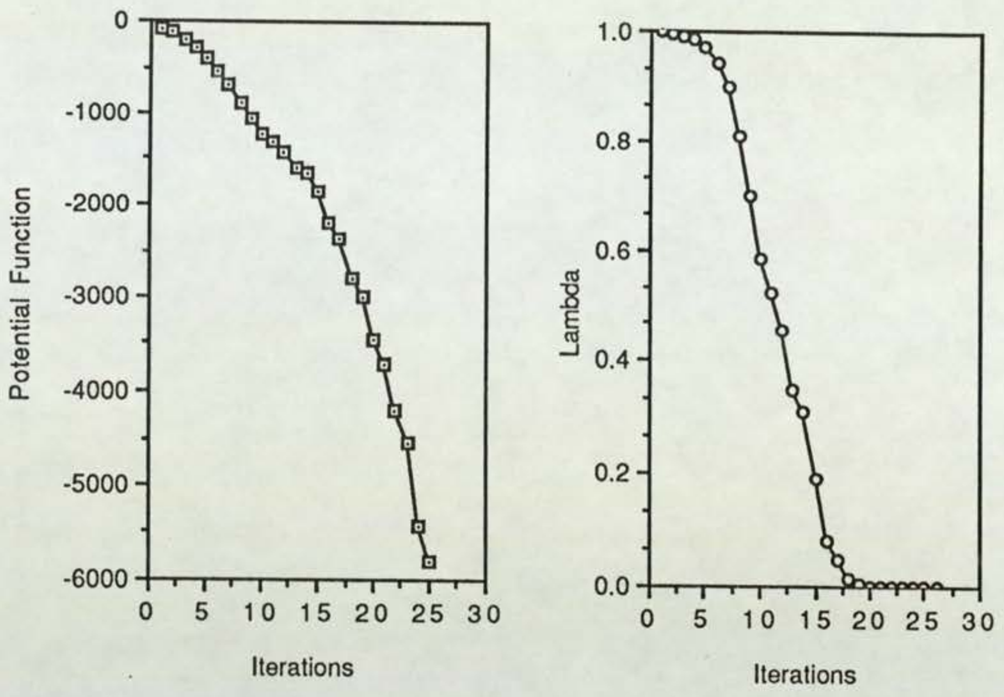


Fig 7.5 Decrease in the Potential and Objective Functions for Problem Scfxm1

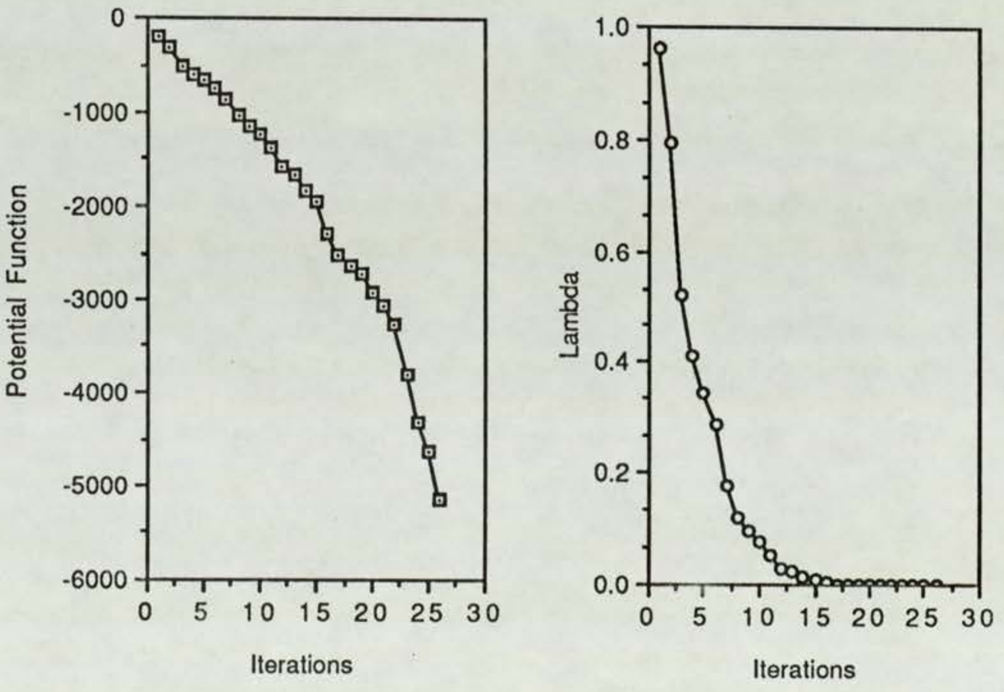


Fig 7.6 Decrease in the Potential and Objective Functions for Problem Sctap1

7.3.3.1 Hilbert-Type LP problems

A version of LPKAR1 which does not take account of sparsity was tested on a set of LP problems whose constraints matrix is based on Hilbert matrix. These problems already used in limited experiments in Chapter 4, can be described as follows. They are of the form

$$\begin{aligned} &\min \ \mathbf{c}^T\mathbf{x} \\ &\text{s.t.} \quad A\mathbf{x} \geq \mathbf{b}, \\ &\qquad\quad \mathbf{x} \geq 0, \end{aligned}$$

where $\mathbf{x} \in R^n$, $A \in R^{nxn}$, $\mathbf{c} \in R^n$ and $\mathbf{b} \in R^n$. Matrix A has entries $[a_{ij}] = [\ 1/(i+j)\ ]$, for $i = 1, \ ..., n$ and $j = 1, \ ..., n$. The RHS is given by

$$b_i = \sum_{j=1}^{n} \frac{1}{i+j}, \qquad\qquad i = 1, 2, \ldots, n.$$

The cost vector is given by

$$c_i = \frac{2}{i+1} + \sum_{j=2}^{n} \frac{1}{i+j}, \qquad\qquad i = 1, 2, \ldots, n.$$

The primal optimum solution to these problems is $\mathbf{x}^* = (1, 1, ..., 1)^T$. Problems with $n = 4, 6, 10, 15, 20, 25, 30$ and 40 were solved and the results depicted in Fig.7.7.

As one would expect, the number of iterations is approximately the same for problems with $n > 6$. Around iteration 16, the potential function levels out and shows hardly any noticeable improvement.

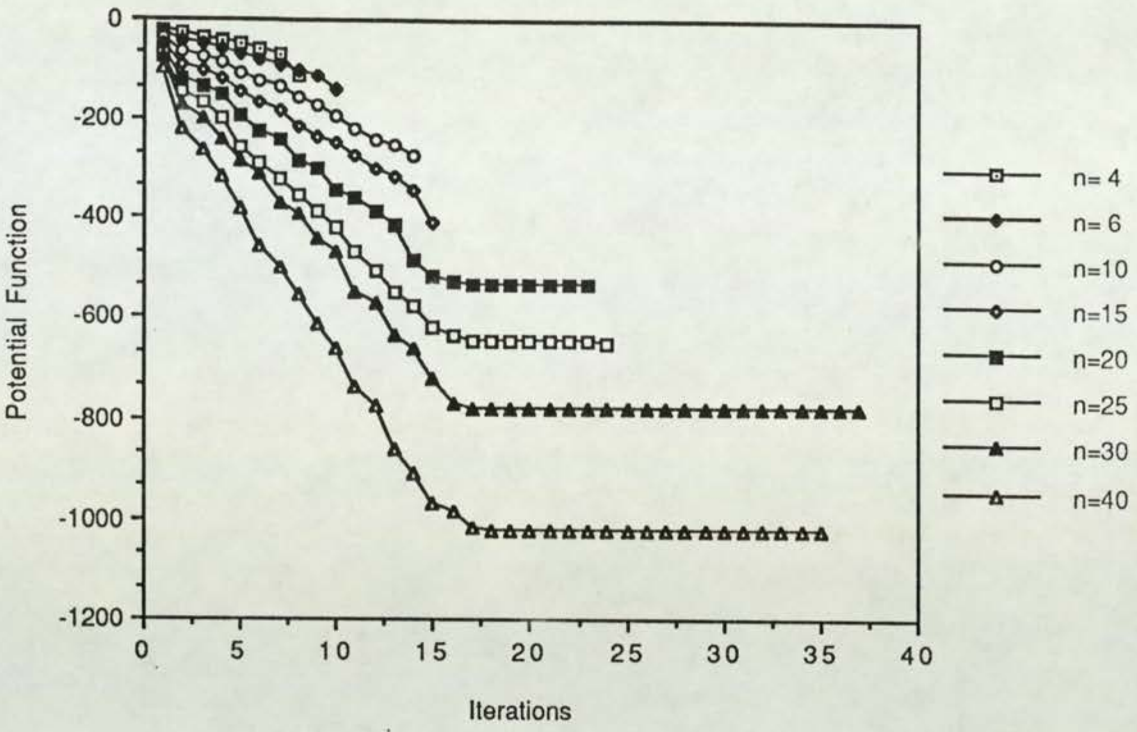


Fig 7.7 Results from LPKAR1 on Hilbert-Type Problems

7.3.3.2 Klee-Minty Problems

The class of problems originally proposed by Klee and Minty (1972) is well known as linear programming problems with n variables for which the simplex method with various pivot rules takes an exponential, in n, number of pivots to reach the optimum. The following form due to Avis & Chvátal (1978) is considered in our experiments, as well as in [Iri & Imai, 1987].

$$\max \sum_{j=1}^{n} \mu^{n-j} x_j,$$

$$\text{s.t. } 2\sum_{j=1}^{i-1} \mu^{i-j} x_j + x_i \leq 1, \quad (i = 1, \ldots, n),$$

$$x_j \geq 0, \quad (j = 1, \ldots, n),$$

where $0 < \mu < 0.5$. The optimum solution of this problem is $x_j = 0$, $(j = 1, ..., n-1)$ and $x_n = 1$. We performed experiments for the cases with $\mu = 0.4$ and n = 6, 12, 18, 24, 30 and 40. The results are shown in Fig 7.8.

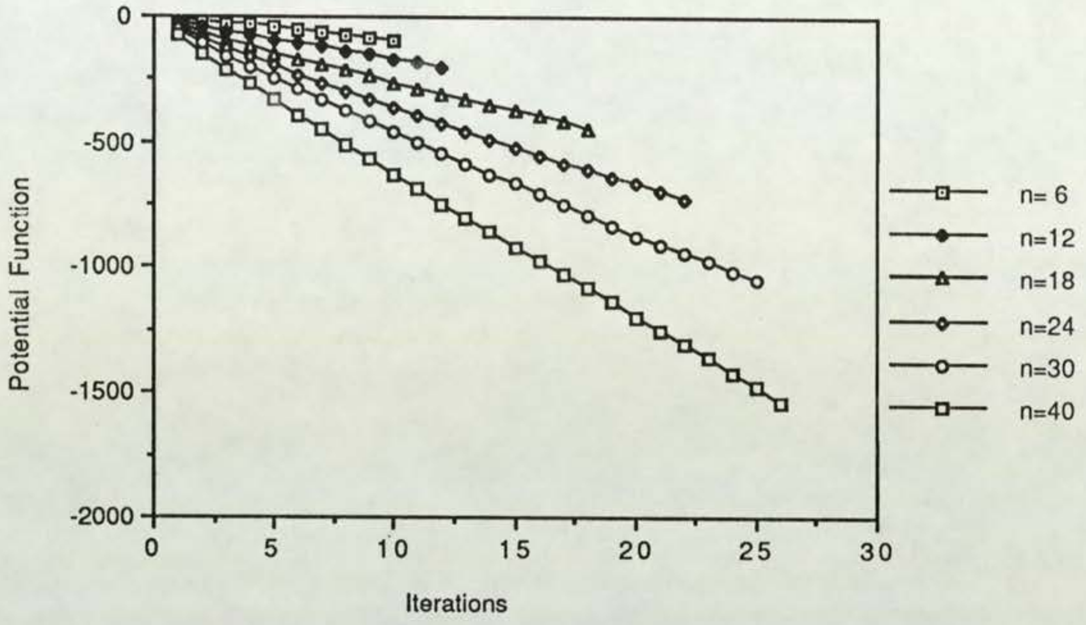


Fig 7.8 Results from LPKAR1 on Klee-Minty Problems

Although the iteration count is still low for the Klee-Minty problem, the number of iterations seems to grow slightly with the size of the problem. But it is nothing like the simplex method. For the Klee-Minty problem of order 40, for instance, the standard simplex would take approximately 10^{12} iterations, as compared to 27 iterations the Karmarkar algorithm takes to find the optimum solution. The growth with the size is logarithmic and not exponential.

7.4 Alternative Least Squares methods

LPKAR1 uses Cholesky method to deal with the least squares problem of step 4. Two other versions of it were written which respectively use Givens orthogonalization method and the iterative technique of Paige and Saunders (1982), which is a conjugate gradient method whose FORTRAN 77 code is known as subroutine LSQR. Although small problems were successfully solved with both versions, on larger problems they seemed slow and unreliable. In the case of Givens rotations, the data structures are mainly those used for Cholesky method except for a supplementary one-dimensional array to handle the nonzeros created during the zeroing process.

Subroutine LSQR was intended for large sparse linear systems and least squares problems. Its use requires the problem matrix to be stored in suitable sparse data structures and a user supplied routine that performs the product of a matrix A (or its transpose A^T) in sparse format with a vector. More precisely the routine will compute $\mathbf{x} = \mathbf{x} + A\mathbf{y}$ and $\mathbf{y} = \mathbf{y} + A^T\mathbf{x}$. Parameters such as tolerance, machine precision and iteration limit must also be set before calling the subroutine.

The data structures mainly consist of three one-dimensional arrays: RA(NZ), JA(NZ) and NA(M), where NZ is the number of nonzeros in the problem matrix and M the number of rows. The problem matrix is stored row-wise in RA, i.e. nonzero elements of row one are stored first then those of row two and so on. The corresponding column index of each nonzero is stored in JA, an INTEGER array. Another INTEGER array, NA, holds the number of nonzero elements of each row of the matrix.

The main difficulties encountered with LSQR were probably due to instability. The solutions returned by the subroutine were bad approximations. The iterative process was never stable and took a number of iterations most of the time equal to the iteration limit parameter set up at the start of the procedure. The lack of a preconditioner may be the cause for this inefficiency.

7.5 Applying Algorithm 7.1 when z* is not *a priori* known

Before dealing with the unknown optimum objective value, it is necessary to find a starting feasible point. This can be done by solving a feasibility problem, otherwise known as Phase 1 problem. This problem is similar to the one solved by LPKAR1. Alternative forms are described in Chapter 2.

The unknown optimum objective value is dealt with by updating the initial value of z in Algorithm 7.1 with $\mathbf{c}^T\mathbf{x}^{(k)}$ after iteration k. This approach is known as the cutting objective function method. Algorithm 7.1 with the cutting objective is a primal-dual algorithm. The vector **y** computed in step 4 is dual feasible. At the end of Phase 2, **y** is the true dual optimum solution if the problem is nondegenerate, i.e. $\mathbf{x}^*$ has at least m+1 positive entries, where m is the number of constraints. Otherwise the dual solution is not unique.

LPKAR2 is a FORTRAN 77 code of this algorithm. Step 4 of Algorithm 7.1 is carried out using the Nag subroutine F01BLF for computing the pseudoinverse.

The code was run on a subset of the problems listed in Table 7.1. The results of these runs are given below. For each problem 5 columns were produced, which respectively are: The iteration number, the optimum step α taken at that iteration, the primal objective value, a lower bound on it and the dual objective value. The blank entries to the last two columns correspond to Phase 1 iterations in which an interior feasible point is found.

The stopping criterion used is based on the gap between the primal objective and its lower bound. Steplength α is computed at each iteration using the blocking variable technique described in Chapter 3.

<u>Problem Name:</u> RandD

```
ITERAT.        α            PRIMAL          L.BOUND           DUAL
-------        --           ------          -------           ----
   1      6.441861428   0.6531709614
   2      4.559867337   0.820182E-01
   3      1.445073937   0.771727E-03
   4      4.621143626   -8884.832099    -16428.78711     -16175.50098
   5      3.841302410   -9320.974037    -13733.94824     -13591.17871
   6      3.682589382   -9412.218261    -11239.68359     -11188.44922
   7      3.033384722   -9436.795379    -11637.96973     -11584.12500
   8      2.566412224   -9450.617668    -.100000E+21     -.975998E+20
   9      2.358197628   -9463.677588    -9715.130859     -9708.828125
  10      1.669588275   -9466.686918    -9491.047852     -9490.426758
  11      2.635119288   -9467.923004    -9471.314453     -9471.197266
  12      1.636255498   -9468.274640    -9468.140625     -9468.111328
  13      6.255412489   -9472.807924    -9467.330078     -9467.324219
  14      2.247326661   -9477.173856    -9565.544922     -9563.179688
  15      6.222729319   -9475.909702    -9474.494141     -9474.494141
```

Problem Name : Chvtl1

ITERAT.	α	PRIMAL	L.BOUND	DUAL
1	2.865809331	0.9934241435		
2	3.966572662	0.7941611308		
3	2.050811475	0.290249E-01		
4	1.008751969	0.287112E-03		
5	9.301639104	-11483.16638	-19023.27539	-18619.72656
6	2.452507189	-13027.91157	-17058.04688	-16811.80273
7	1.972840325	-13286.31012	-17406.45898	-17182.81641
8	14.74104330	-13651.34425	-16813.26172	-16627.70703
9	2.797717475	-13771.54418	-16001.35547	-15872.47559
10	4.927777042	-13778.51607	-16125.33691	-15988.55273
11	5.805054359	-13779.43317	-16142.97949	-16004.45898
12	21.45803992	-13791.03023	-16141.19727	-16002.74707
13	10.02461707	-13878.35165	-15772.87598	-15660.45605
14	4.149567088	-14012.74683	-14595.31836	-14553.69141
15	20.14787443	-14015.04564	-14152.19531	-14144.03809
16	20.12815984	-14015.08041	-14123.12012	-14116.76367
17	25.84468262	-14015.08173	-14122.71289	-14116.38184
18	30.98095870	-14015.08194	-14122.70898	-14116.37793
19	108.0583358	-14015.07551	-14122.70898	-14116.37793
20	4.082054439	-14015.33463	-14122.70898	-14116.37793
21	7700.428266	-14015.08186	-14122.70801	-14116.37695
22	4.081874178	-14015.08029	-14122.70898	-14116.37793
23	24228.34324	-14015.05874	-14122.70703	-14116.37598
24	2371190485.	-14015.22800	-14122.57227	-14116.25000
25	70.28025742	-14021.00474	-14117.38281	-14111.38184
26	44.80198279	-14021.03772	-14021.04102	-14021.04102

Problem Name : Chvtl2

ITERAT.	α	PRIMAL	L.BOUND	DUAL
1	4.403545515	0.9944982521		
2	2.450272242	0.5077499693		
3	1.131387576	0.965917E-02		
4	1.005887171	0.944669E-04		
5	2.147604523	-228535.2657	-642805.9375	-626204.3125
6	3.016425300	-250208.7442	-317029.2813	-314349.6250
7	2.819401851	-263412.8438	-312417.2500	-310173.6875
8	3.022792842	-267849.2021	-315956.2188	-313710.1875
9	4.931871498	-269401.4924	-315477.8438	-313141.0000
10	6.209180867	-269913.1435	-314532.0625	-312117.5000
11	9.077869172	-273307.8685	-309012.2813	-306870.6875
12	29.57530165	-273377.0912	-273385.5000	-273385.3125
13	29.02232090	-273381.8568	-273382.0000	-273382.0000

Problem Name : Alfaut

ITERAT.	α	PRIMAL	L.BOUND	DUAL
1	5.020584304	0.9987008107		
2	5.016395983	0.7061022659		
3	1.224532672	0.226881E-01		
4	0.997201850	0.228803E-03		
5	7.701221428	-9256687.028	-36732648.00	-36303604.00
6	3.943619602	-10490237.48	-31063900.00	-30724012.00
7	2.934036697	-11106791.07	-16443347.00	-16347721.00
8	3.248521773	-11658852.95	-14401056.00	-14348024.00
9	7.467750399	-12046247.56	-13226854.00	-13200235.00
10	4.741095164	-12171514.61	-12525954.00	-12517692.00
11	5.997308168	-12218466.46	-12263207.00	-12262387.00
12	8.387838113	-12230475.94	-12236620.00	-12236546.00
13	13.59068778	-12233368.10	-12234132.00	-12234121.00
14	27.19466703	-12233714.86	-12233751.00	-12233751.00
15	65.21966319	-12233741.39	-12233742.00	-12233742.00

Problem Name : Scagr7

ITERAT.	α	PRIMAL	L.BOUND	DUAL
1	7.405831680	0.9981415303		
2	6.199722355	0.9863109023		
3	3.120554655	0.9243653430		
4	1.987366289	0.988973E-01		
5	.9975537117	0.108856E-02		
6	9.235479623	-2023372.180	-21206690.00	-21084080.00
7	3.001158895	-2171044.627	-10496797.00	-10442040.00
8	3.235121117	-2250286.867	-3377933.750	-3370697.250
9	4.528271088	-2293157.895	-2505680.500	-2504344.000
10	6.289297716	-2319053.392	-2383567.750	-2383144.500
11	10.90774247	-2328609.473	-2354395.750	-2354224.000
12	7.018505189	-2329835.032	-2335393.000	-2335359.000
13	9.477241468	-2330885.659	-2334042.250	-2334019.500
14	7.012442289	-2331150.209	-2333566.750	-2333549.250
15	8.106021664	-2331280.375	-2332203.250	-2332196.750
16	7.107331053	-2331328.956	-2331503.750	-2331502.750
17	18.04304854	-2331369.400	-2331467.000	-2331466.500
18	14.72354161	-2331381.338	-2331431.750	-2331431.500
19	21.64842011	-2331386.993	-2331391.500	-2331391.500
20	109.6028688	-2331389.556	-2331389.750	-2331389.750

7.6 Comparative Results between LPKAR1 (Case 3) and LINDO

LINDO (Linear INteractive Discrete Optimizer), [Schrage, 1983], is a commercial package which does Linear as well as Integer and Quadratic Programming. It is available on Aston University's VAX 11/750 computer. To have an idea about the performance of our codes, we ran a version of LPKAR1 (Case 3) and LINDO on nine of the test problems given in Table 7.1. The results are recorded in Table 7.6.

	LPKAR1 (Case 3)		LINDO	
Problem	CPU(s)	IT	CPU(s)	IT
Chvtl1	5.79	15	3.69	11
Chvtl2	5.60	14	3.48	9
Alfaut	13.30	15	5.91	43
Scsd1	428.62	12	84.23	454
Scagr7	81.18	18	26.89	213
Sc205	148.73	21	54.89	207
Sctap1	1171.95	26	89.61	412
Scfxm1	1614.57	24	191.56	654
Scagr25	1514.95	27	377.13	1284

Table 7.6 Comparative Results: LPKAR1 (Case 3) v LINDO

From the iteration count point of view, LPKAR1 is superior to LINDO except on the small problems Chvtl1 and Chvtl2. However, LINDO requires less CPU time to solve all the problems.

Note that the difference in CPU times required by LPKAR1 (Case 3) given in Table 7.4 and in the above table, is due to the computers used; the results of Table 7.4 were obtained on a VAX 8650 machine (6.5 mips), while the above results where obtained on a VAX 11/750 machine (0.7 mips).

7.7 Conclusion

Throughout these experiments, it is confirmed that Karmarkar's algorithm preserves its attractive features on various types of problems especially the real world problems listed in Table 7.1. These features, namely, are its low iteration count (logarithmic in the size of the problem) and its acceptance and use of the duality aspects of linear programming. Although the work in an iteration of the algorithm is substantially higher than that of the simplex [Tomlin, 1985], it may be effectively reduced when existent sparsity techniques such as ordering and partitioning are used. In this way, large real world LP problems can be solved in realistic times as shown in Tables 7.2 through 7.5. The dependence of the performance of the algorithm on least squares techniques [Lindfield & Salhi, 1987] is also shown in those tables. This may be held against the algorithm. However, any improvement in the solution of the least squares problem can readily be used in Karmarkar's algorithm.

Chapter 8

Conclusions and Further Development

When Karmarkar's algorithm came to public attention in 1984 [Kolata, 1984; Emmett, 1985], many criticisms were made regarding its alleged efficiency. It was thought to be inherently slow despite its polynomial complexity [Charnes *et al.*, 1984]. Its applicability was restricted to a special class of LP problems with homogeneous constraints and optimum objective $z^* = 0$. A feasible interior point is also required to start the algorithm. This is a restrictive requirement because it is known, (von Newmann, cited by Charnes *et al.*, 1984), that any method which finds a feasible point to a linear programming problem can find its optimum solution. However, probably the most serious criticism concerned the algorithm being only primal with no prospects for the important duality concepts to be used; postoptimality analysis was, thus, not possible.

At the beginning of this thesis the Karmarkar algorithm was discussed in the context of early development of polynomial time algorithms for linear programming. Theoretical as well as computational results of extensive research aimed at alleviating the difficulties of the original Karmarkar's algorithm and assessing its performance were reviewed.

For efficient implementation of the algorithm, advanced least squares techniques are required. In this respect, and for the sake of completeness, this topic was also reviewed with emphasis on sparsity exploitation. It was found, after an early implementation of the algorithm, that the size of the step taken in the search direction, greatly influences the convergence of the algorithm. Its optimum choice and, in general, the conditions under which large steps are allowed, were investigated.

The duality aspects of the algorithm were studied in conjunction with three main variants due to Todd and Burrell (1986), Gay (1987) and Ye and Kojima (1987). The variant of Ye and Kojima seems to be superior, because it works under mild assumptions, it is easy to implement and theoretically it generates better bounds on the optimum objective value. However, from their paper it was not clear how, in practice, these bounds are found. A procedure which works on most problems was, thus, developed. With dual variables being available through these variants (discussed in Chapter 4), we were encouraged to investigate their potential use for postoptimality analysis. Postoptimality analysis for the right-hand side, the cost and the rim was briefly studied as a result.

Following the underlying ideas of interior point methods, an attempt was made to design an algorithm for LP based on generating a finite sequence of Chebyshev points. The algorithm was shown to work on small problems. However, in its present form, it does not seem to be efficient. Improvements to the algorithm were suggested.

The study of decomposition and partitioning as strategies for reducing the work in an iteration of the Karmarkar algorithm, constitutes one of the main objectives of this thesis. Structured LP problems being an important class of problems frequently occurring in real applications, it was felt that extending the Karmarkar algorithm to such problems was worthwhile. As a consequence, a specialized variant of the dual Karmarkar algorithm for structured LP problems was designed and tested on randomly and non-randomly generated problems. It appears from the experiments that the partitioning variant is superior to Algorithm 4.2 on structured LP problems. A practical implementation of a variant of the Karmarkar algorithm, in which sparsity preservation and exploitation is the central issue, has been developed. The updating algorithm of Heath (1984) and the nested dissection algorithm [George & Liu, 1981] were used in the resulting code. The code was shown to work on various types of problems in realistic CPU times. Alternative least squares methods were also used in different versions of the code. The performance of the Karmarkar algorithm was discussed in Chapter 7.

Among aspects of Karmarkar's algorithm and linear programming discussed or investigated in this work, postoptimality analysis is without doubt the topic that needs to be further studied. Without it simplex would not be the powerful decision making tool it is today. It would be very interesting and useful, therefore, to know how, in real applications, postoptimality analysis can be carried out via Karmarkar's algorithm or its variants. Indeed, the future of the algorithm would be rather bleak if it is found to be unsuitable for postoptimality analysis in real applications. It is unfortunate, due mainly to time limitations, that our investigation is not conclusive in this respect.

The Chebyshev approach discussed in Chapter 5, is also worthwhile to further investigate. The method is strongly related to the simplicial algorithm of Levin and Yamnitsky (1982). To our knowledge, there is no efficient implementation of this algorithm. Our approach may help understanding how to efficiently implement it. Chebyshev problems convert into LP problems with very sparse objective vectors. Only the entry corresponding to the deviation is nonzero. Exploiting this sparsity may be beneficial.

At the end of Chapter 6, an attempt to justify decomposition (lemma 6.1) using the concept of volumes corresponding to the potential function of Karmarkar was made. The partitioning algorithm developed in that chapter, was not based on that concept. However, the idea is attractive as it brings more closely the projective and the ellipsoid algorithms. Indeed, recent developments show the merits of such an approach. Ye (1987), showed that the potential function of Karmarkar characterizes the logarithmic volume of an ellipsoid that contains all of the dual solutions. As the potential function decreases, the volume of the ellipsoid monotonically shrinks to zero. Todd (1988a), also discusses the construction of a dual ellipsoid during the course of the Karmarkar algorithm. The idea of solving smaller problems and working in lower dimensions, in order to start with an overall smaller volume, may be profitable for quick convergence in both the Khachyan and the Karmarkar algorithms.

Decomposition is also attractive from the point of view of parallelism or concurrency. We have shown that favourable structure present in large LP problems may be used to advantage and our partitioning algorithm lends itself readily to parallel processing. However, it is in large, dense and unstructured linear programming problems that parallel architecture and concurrent processing are expected to have an impact. In this respect, the general form of Algorithm 6.1, which applies the concept of decomposition to unstructured problems, is worth investigating. Note that a parallel version of Karmarkar's algorithm has been developed by Pan and Reif (1985).

During the investigation of the Karmarkar algorithm, and the review of least squares techniques, codes were written in order to find out about the practical value of the methods. The codes are independent from one another. For instance, the code for the Karmarkar algorithm in which ordering and partitioning are used, is separate from the code in which no such measures are taken. It would, therefore, be interesting to put the programs in a library equipped with a user friendly interface. Depending on the size, density and condition of a LP problem, appropriate routines can, thus, be chosen for its solution.

From our investigations it appears that Karmarkar's algorithm is a serious alternative to the simplex method. However, the questions raised here need to be answered before the algorithm is fully adopted as the standard method for linear programming.

References

Anstreicher, K.M. (1986a), "A Strengthened Acceptance Criterion for Approximate Projections in Karmarkar's Algorithm", *Operations Res. Lett.* **5**(4), pp. 211-214.

Anstreicher, K.M. (1986b), "A Monotonic Projective Algorithm for Fractional Linear Programming", *Algorithmica* **1**, pp. 483-498.

Anstreicher, K.M. (1988), "Linear Programming and the Newton Barrier Flow", *Mathematical Programming* **41**, pp. 367-373.

Apsvall, B. & R.E.Stone (1980), "Khachiyan's Linear Programming Algorithm", *Journal of Algorithms* **1**, pp. 1-13.

Avis, D. & V.Chvátal (1978), "Notes on Bland's Pivoting Rule", *Mathematical Programmimg* **8**, pp. 24-34.

Barnes, E.R. (1986), "A Variation on Karmarkar's Algorithm for Linear Programming Problems", *Mathematical Programming* **36**(2), pp. 174-182.

Benichou, M., J.M.Gauthier, G.Hentges & G.Ribière (1977), "The Efficient Solution of Large Scale Linear Programming Problems- Some Algorithmic Techniques and Computational Results", *Mathematical Programming* **13**, pp. 280-322.

Bisshopp, F. (1981), "Khachyan's Algorithm for Linear Programming, Optimisation and Implementation", *Quart. App. Math* **38**, pp. 415-426.

Bland, R.G. (1977), "New Finite Pivoting Rules for the Simplex Method", *Mathematics of Operations Research* **2**, pp. 103-107.

Bland, R.G., D.Goldfarb & M.J.Todd (1981), "The Ellipsoid Method: A Survey", *Operations Research* **29**(6), pp. 1039-1091.

Burrell, B.P. & M.J.Todd (1985), "The Ellipsoid Method Generates Dual Variables", *Mathematics of Operations Research* **10**(4), pp. 688-700.

Cavalier, T.M. & K.C.Schall (1987), "Implementing an Affine Scaling Algorithm for Linear Programming", *Comput. Opns. Res.* **14**(5), pp. 341-347.

Charnes, A., T.Song & M.Wolfe (1984), "An Explicit Solution Sequence and Convergence of Karmarkar's Algorithm", Research Report CCS 501, Center for Cybernetic Studies, College of Business Administration S.202, The University of Texas at Austin, Texas 78712-1177 (512), 471-1821.

Chvátal, V. (1983), *Linear Programming*, W.H.Freeman & Co, USA.

Cook, S. (1983), "An Overview of Computational Complexity", *Communications of the ACM* **26**(6), pp. 401-408.

Dantzig, G.B. & P.Wolfe (1960), "Decomposition Principle for Linear Programs", *Operations Research* **8**, pp. 101-111.

Dantzig, G.B. (1963), *Linear Programming and Extensions*, Princeton University Press, Princeton, NJ.

Dantzig, G.B. (1987), Private Communication.

Dennis, J.E., Jr., A.M.Morchedi & K.Turner (1986), "A Variable-Metric Variant of the Karmarkar Algorithm for Linear Programming", Technical Report 86-13, June 1986, Dept. of Math. Sci., Rice University, Houston, Texas 77251, USA

Dennis, J.E., Jr., A.M.Morshedi & K.Turner (1987), "A Variable-Metric Variant of the Karmarkar Algorithm for Linear Programming", *Mathematical Programming* **39**, pp. 1-20.

Dikin, I.I. (1967), "Iterative Solution of Problems of Linear and Quadratic Programming", *Soviet Math. Doklady*, **8**(3), pp. 674-675.

Dodani, M.H. & A.J.G.Babu (1987), "Karmarkar's Projective Method for Linear Programming: A Computational Survey", *Computers and Industrial Engineering* **13**(1-4), pp. 285-289.

Duff, I.S., A.M.Erisman & J.K.Reid (1976), "On George's Nested Dissection Method", *SIAM Journal of Numerical Analysis* **13**(5).

Duff, I.S., A.M.Erisman & J.K.Reid (1986), *Sparse Matrix Computation*, Academic Press, London.

Edmonds, J. (1965), "Paths, Trees and Flowers", *Canadian Journal of Mathematics* **17**, pp. 449-467.

Edmonds, J. (1967), "Systems of Distinct Representatives and Linear Algebra", *Journal of Research of the National Bureau of Standards*, 71B, pp. 214-245.

Emmett, A. (1985), "Karmarkar's Algorithm: A Threat to Simplex?", *IEEE Spectrum*, pp. 54-55, (December).

Evans, D.J. (1985), *Sparsity and Its Applications,* Cambridge University Press.

Ferris, M.C. & A.B.Philpott (1988), "On the Performance of Karmarkar's Algorithm", *J. Opl. Res. Soc.* **39**(3), pp. 257-270.

Fiacco, A.N. & G.D.McCormick (1968), *Non-Linear Programming: Sequential Unconstrained Minimization Techniques*, Wiley & Sons Ltd.

Fieldhouse, M. & F.W.Tromans (1985), "Convergence, Scaling and Duality in Karmarkar's Projective Algorithm", Symposium on Karmarkar's and Related Algorithms for Linear Programming, organized by IMA, held on May the 7th 1985 at the Geological Society, Burlington House, Piccadilly, London.

Fletcher, R. (1986), "Recent Developments in Linear and Quadratic Programming", Report NA/94, Department of Maths Sciences, University of Dundee, Scotland.

Forsythe, G.E., M.A.Malcolm & C.B.Moler (1977), *Computer Methods for Mathematical Computations*, Prentice-Hall, Englewood Cliffs, NJ.

Fourer, R. (1982), "Solving Staircase Linear Programs by the Simplex Method, I: Inversion", *Mathematical Programming* **23**, pp. 272-313.

Freund, R.M. (1988), "An Analog of Karmarkar's Algorithm for Inequality Constrained Linear Programs, with a 'New' Class of Projective Transformations for Centering a Polytope", *Oper. Res. Lett.* **7**(1), pp. 9-13.

Frisch, K.R. (1955),"The Logarithmic Potential Method of Convex Programming", Unpublished, University Institute of Economics, Oslo.

Gács, P. & L.Lovász (1979), "Khachyan's Algorithm for Linear Programming", STAN-CS-79-750, Computer Science Department, Stanford University.

Gács, P. & L.Lovász (1981), "Khachyan's Algorithm for Linear Programming", *Mathematical Programming Study* **14**, pp. 61-68.

Gal, T. (1979), *Postoptimal Analysis, Parametric Programming and Related Topics*, McGraw-Hill, Inc., USA.

Gary, M.R. & D.S.Johnson (1979), *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H.Freeman & Company, San Francisco, CA.

Gay, D.M. (1987), "A Variant of Karmarkar's Linear Programming Algorithm for Problems in Standard Form", *Mathematical Programming* **37**(1), pp. 81-90.

Gentleman, W.M. (1973), "Least Squares Computations by Givens Transformations without Square Roots", *J. Inst. Maths Applications* **12**, pp. 329-336.

Geoffrion, A.M. (1970), "Elements of Large-Scale Mathematical Programming", *Management Science* **16**(11), pp.652-691.

George, A. & J.W.Liu (1981), *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, Inc., Englewood Cliffs, NJ 07632.

George, A. & M.T.Heath (1980), "Solution of Sparse Linear Least Squares Problems Using Givens Rotations", *Linear Algebra and Its Applications* **34**, pp. 69-83.

Ghellinck, G. de, & J.-Ph.Vial (1986), "A Polynomial Newton Method for Linear Programming", *Algorithmica* **1**, pp. 425-453.

Ghellinck, G. de, & J.-Ph.Vial (1987), "An Extension of Karmarkar's Algorithm for Solving a System of Linear Homogeneous Equations on the Simplex", *Mathematical Programming* **39**, pp. 79-92.

Gill, P.E., W.Murray & M.H.Wright (1981), *Practical Optimization*, Academic Press.

Gill, P.E., W.Murray, M.A.Saunders & M.H.Wright (1984), "Sparse Matrix Methods in Optimization", *SIAM J. Sci. Stat. Comp.* **5**(3), pp. 562-589.

Gill, P.E., W.Murray, M.A.Saunders & M.H.Wright (1988), "Numerical Issues in Interior-Point Methods", SIAM Conference, San Francisco, June 13-16.

Gill, P.E., W.Murray, M.A.Saunders, J.A.Tomlin & M.H.Wright (1985), "On Projected Newton Barrier Methods for Linear Programming and an Equivalence to Karmarkar's Projective Method", *Mathematical Programming* **36**(2), pp. 183-209.

Goldfarb, D. & J.K.Reid (1977), "A practicable Steepest-Edge Simplex Algorithm", *Mathematical Programming* **12**, pp.361-371.

Goldfarb, D. & M.J.Todd (1982), "Modification and Implementation of the Ellipsoid Algorithm for Linear Programming", *Mathematical Programming* **23**, pp. 1-19.

Goldfarb, D. & S.Mehrotra (1988a), "Relaxed Variants of Karmarkar's Algorithm for Linear Programs with Unknown Optimal Objective Value", *Mathematical Programming* **40**(2), pp. 183-195.

Goldfarb, D. & S.Mehrotra (1988b), "A Relaxed Version of Karmarkar's Method", *Mathematical Programming* **40**(3), pp. 289-315.

Golub, G. & C.Van Loan (1983), *Matrix Computations*, John Hopkins University Press, Baltimore, USA.

Golub, G. (1965), "Numerical Methods for Solving Linear Least Squares Problems", *Numerische Mathematik* **7**, pp. 206-216.

Grigoriadis, M.D. & K.Ritter (1969), "A Decomposition Method for Structured Linear and Nonlinear Programs", *J. of Computer and System Science* **3**(4), pp. 335-360.

Grötschel, M., L.Lovász & A.Schrijver (1981), "The Ellipsoid Method and Its Consequences in Combinatorial Optimization", *Combinatorica* **1**(2), pp. 169-197.

Halfin, S. (1983), "The Sphere Method and the Robustness of the Ellipsoid Algorithm", *Mathematical Programming* **26**, pp. 109-116.

Heath, M.T. (1981), "Some Extensions of An Algorithm for Sparse Linear Least Squares Problems", Computer Sciences Division at Oak Ridge National Laboratory, PO Box Y, Oak Ridge Tennessee 37830.

Heath, M.T. (1984), "Numerical Methods for Large Sparse Linear Least Squares Problems", *SIAM J. Sci. Stat. Comp*. **4**(3), pp. 497-513.

Ho, J.K. & E.Loute (1979), "A Comparative Study of Two Methods for Staircase Linear Programs", *ACM Transactions on Mathematical Software* **5**(4), pp. 17-30.

Ho, J.K. & E.Loute (1980), "A set of Staircase Linear Programming Test Problems", *Mathematical Programming* **20**, pp. 245-250.

Ho, J.K. & E.Loute (1981), "An Advanced Implementation of the Dantzig-Wolfe Decomposition Algorithm for Linear Programming", *Mathematical Programming* **20**, pp. 303-326.

Hooker, J.N. (1986), "Karmarkar's Linear Programming Algorithm", *Interfaces* **16**, pp. 75-90.

Imai, H. (1988), "On the Convexity of the Multiplicative Version of Karmarkar's Potential Function", *Mathematical Programming* **40**, pp. 29-32.

Iri, M. & H.Imai (1986), "A Multiplicative Barrier Function Method for Linear Programming", *Algorithmica* **1**, pp. 455-482.

Jones, P.C. & E.S.Marwil (1982), "Dimensional Reduction Variant of The Ellipsoid Algorithm for Linear Programming Problems", *Mathematics of Operations Research* **7**(2), pp. 245-252.

Kantorovich, L.V. (1939), "Mathematical Methods in the Organization and Planning of Production", Translated in *Management Sci.*, **6**, pp. 366-422, (1958).

Karmarkar, N. (1984a), "A New Polynomial-Time Algorithm for Linear Programming", *Proceedings of the 16th Annual ACM Symposium on Theory of Computing*, pp. 302-311, Washington D.C.

Karmarkar, N. (1984b), "A New Polynomial-Time Algorithm for Linear Programming", *Combinatorica* **4**(4), pp. 373-395.

Khachyan, L.G. (1979), "A Polynomial Algorithm in Linear Programming", *Soviet Math. Dokl.* **20**(1), pp. 191-194.

Klee, V. & G.J.Minty (1972), "How Good Is the Simplex Algorithm?", in *Inequalities III*, O. Shisha (Ed.), Academic Press, NY, pp. 159-179, Kojima, M. (1986), "Determining Basic Variables of Optimal Solutions in Karmarkar's New LP Algorithm", *Algorithmica* **1**, pp. 499-515.

Kolata, G. (1984), "A Fast Way to Solve Hard Systems", *Science* **225**(21), pp. 1379-1380.

Kortanek, K.O. & M.Shi (1987), "Convergence Results and Numerical Experiments on a Linear Programming Hybrid Algorithm", *E.J.O.R.* **32**, pp. 47-61.

König, H. & D. Pallaschke (1981), "On Khachyan' s Algorithm and Minimal Ellipsoids", *Numerische Mathematik* **38**, pp. 211-223.

Kronsjö, L. (1985), *Computational Complexity of Sequential and Parallel Algorithms*, Wiley & Sons Ltd, GB.

Kronsjö, L. (1987), *Algorithms: Their Complexity and Efficiency*, (2nd edition), Wiley & Sons Ltd, GB.

Lawson, C.L. & R.J.Hanson (1974), *Solving Least Squares Problems*, Prentice-Hall, Englewood Cliffs, NJ.

Levin, J.A. (1965), "On an Algorithm for the Minimization of Convex Functions", *Doklady Akademii Nauk SSSR* **160**, #6. Translated in Soviet Math. Doklady vol.160.

Lindfield, G.R. & A.Salhi (1987), "A Comparative Study of the Performance and Implementation of the Karmarkar Algorithm", presented at the *Martin Beale Memorial Symposium*, 6-8 July, The Royal Society, London.

Longley, W.J (1984), *Linear Least Squares Computations Using Orthogonalisation Methods*, Marcel Dekker, Inc., NY 10016, USA.

Lovász, L. (1980), "The Ellipsoid Algorithm: Better or Worse than the Simplex?", *Mathematical Intelligencer* **2**, pp. 141-146.

Lovász, L. (1984), "The Mathematical Notion of Complexity", *Proceedings of the 9th Triennial World Congress of IFAC*, Budapest, 2-6 july. (Vol. 3, pp. 1105-1110).

Lustig, I.J. (1985), "A Practical Approach to Karmarkar's Algorithm", TR SOL 85-5, Department of Operations Research, Stanford University, Stanford, CA 94305.

Mangasarian, O.L. (1985), "Iterative Solution of Linear Programs", *SIAM J. Numer. Anal.*, **18**(4), pp. 606-614.

McCall, E.H. (1982), "Performance Results of the Simplex Algorithm for a set of Real World Linear Programming Models", *Communications ACM* **25**(3), pp. 207-212.

Megiddo, N. (1986), "Introduction: New Approaches to Linear Programming", *Algorithmica* **1**, pp. 387-394.

Megiddo, N. (1987), "Linear Programming (1986)", *Ann. Rev. Comput. Sci.* **2**, pp. 119-145.

Monma, C.L. & A.J.Morton (1987), "Computational Experience With a Dual Affine Variant of Karmarkar's Method for Linear Programming", *Operations Research Letters* **6**(6), pp. 261-267.

Murty, K.G. & Y.Fathi (1984), "A Feasible Direction Method for Linear Programming", *Operations Research Letters* **3**(3), pp. 121-127.

Nemirovskiy, A.S. (1987), "An Algorithm of the Karmarkar Type", *Soviet J. Comput. Syst. Sc.* **25**(5), pp. 61-74.

Nickels, W., W.Rödder, L.Xu & H.-J.Zimmermann (1985), "Intelligent Gradient Search in Linear Programming", *E.J.O.R.* **22**, pp. 293-303.

Orden, A. (1980), "A Step Towards Probabilistic Analysis of Simplex Method Convergence", *Mathematical Programming* **19**, pp. 3-13.

Padberg, M.W. (1986), "A Different Convergence Proof of the Projective Method for Linear Programming", *Operations Research Letters* **4**(6), pp. 253-257.

Paige, C.C. & M.A.Saunders (1982), "LSQR: An Algorithm for Sparse Linear Equations and Sparse Least Squares", *ACM Transactions Math. Software* **8**(1), pp. 43-71.

Pan, V. & J.Reif (1985), "Fast and Efficient Algorithms for Linear Programming and for thr Linear Least Squares Problem", TR-11-85, Harvard University, Center for Research in Computing Technology.

Papadimitriou, C.H. & K.Steiglitz (1982), *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, Englewood Cliffs, NJ.

Rockett, A.M. & J.C.Stevenson (1987), "Karmarkar's Algorithm: A Method for Solving Large Linear Programming Problems", *BYTE* , pp. 146-160, (September).

Roos, C. (1985), "On Karmarkar's Projective Method for Linear Programming", Report 85-23, Department of Mathematics and Informatics, Delft University, Netherland.

Rosen, J.B. & J.C.Ornea (1963), "Solution of Nonlinear Programming Problems by Partitioning", *Management Science* **10**(1), pp. 160-173.

Rosen, J.B. (1964), "Primal Partition Programming for Block Diagonal Matrices", *Numerische Mathematik* **6**, pp. 250-260.

Salhi, A. & G.R.Lindfield (1988), "Postoptimality Analysis via Karmarkar's Algorithm", *3rd SIAM Conference on Applied Linear Algebra*, May 23-26, the Concourse Hotel, Madison, Wisconsin, USA.

Schönlein, A. (1986), "Der Algorithmus von Karmarkar - Idee, Realisation, Beispiel und Numerische Erfahrungen", *Angewandte Informatik* **8**, pp. 344-53.

Schrage, L.E. (1983), *User's Manual for LINDO*, University of Chicago, USA.

Schreck, H. (1986), "Experiences with an Implementation of Karmarkar's LP Algorithm", *Methods of Operations Research* **54**, pp. 535-542.

Schrijver, A. (1986), *Theory of Linear and Integer Programming*, J. Wiley & Sons Ltd.

Shamir, R. (1987), "The Efficiency of the Simplex Method: A Survey", *Management Science* **33**(3), pp. 301-334.

Shanno, D.F. & R.E.Marsten (1988), "A Reduced-Gradient Variant of Karmarkar's Algorithm and Null-Space Projections", *J. Opt. Theory and App.* **57**(3), pp. 383-397.

Shanno, D.F. (1988), " Computing Karmarkar Projections Quickly", *Mathematical Programming* **41**, pp. 61-71.

Shetty, C.M. & M.Ben Daya (1985), "On the Step Size in Karmarkar's Algorithm", PDRC Report Series 85-02, School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, Georgia 30332, USA.

Shor, N.Z. (1977), "Cut-off Method with Space Extension in Convex Programming", *Kibernetica* **13**, pp. 94-95. English Translation: *Cybernetics* **13**, pp. 94-96.

Simonnard, M. (1966), *Linear Programming*, Prentice-Hall, Englewood Cliffs, NJ.

Smale, S. (1983), "On the Average Number of Steps of The Simplex Method of Linear Programming", *Mathematical Programming* **27**, pp. 241-262.

Sonnevend, Gy. (1985), "An Analytical Centre for Polyhedrons and New Classes of Global Algorithms for Linear (Smooth, Convex) Programming", *Proceedings of the12th Conference on System Modelling*, Budapest, pp. 866-875.

Strang, G. (1985), "Karmarkar's Algorithm in a Nutshell", *SIAM News* **18**.

Strang, G. (1987), "Karmarkar's Algorithm and Its Place in Applied Mathematics", *The Mathematical Intelligencer* **9**(2), pp. 4-10.

Tellier, H.Le, (1984), "La Géomètrie Du Marchand des Quatre Saisons", *Science & Avenir* **448,** pp. 47-51, (juin).

Todd, M.J. & B.P.Burrell (1986), "An Extension of Karmarkar's Algorithm for Linear Programming Using Dual Variables", *Algorithmica* **1**, pp. 409-424.

Todd, M.J. (1982), "On Minimum Volume Ellipsoids Containing Part of a Given Ellipsoid", *Mathematics of Operations Research* **7**(2), pp. 253-261.

Todd, M.J. (1988a), "Improved Bounds and Containing Ellipsoids in Karmarkar's Linear Programming Algorithm", *Mathematics of Operations Research* **13**(4), pp. 650-659.

Todd, M.J. (1988b), "Exploiting Special Structure in Karmarkar's Linear Programming Algorithm", *Mathematical Programming* **41**, pp. 97-113.

Tomlin, J.A. (1985), "An Experimental Approach to Karmarkar's Projective Methods for Linear Programming", *Proceedings of Symposium on Karmarkar's and Related Algorithms for Linear Programming*, organized by IMA, held on May the 7th 1985 at the Geological Society, Burlington House, Piccadilly, London.

Traub, J.F. & Wozniakowki (1982), "Complexity of Linear Programming", *Operations Research Letters* **1**(2), pp. 59-62.

Turner, K. (1987), "A Variable-metric Variant of the Karmarkar Algorithm for Linear Programming", Technical Report 87-13, May 1987, Dept. of Math. Sci., Rice University, Houston, Texas 77251, USA.

Vanderbei, R.J., M.S.Meketon & B.A.Freedman (1986), "A Modification of Karmarkar's Linear Programming Algorithm", *Algorithmica* **1**, pp. 395-407.

Vial, J.-P. (1986), "Approximate Projections in a Projective Method for the Linear Feasibility Problem", CORE Discussion Paper N°8707, Center for Operations Research and Econometrics, Universite Catholique de Louvain, Belgium.

Vial, J.-P. (1987), "A Fully Polynomial-Time Projective Method", CORE Discussion Paper N°8713, Center for Operations Research and Econometrics, Universite Catholique de Louvain, Belgium.

Yamnitsky, B. & J.A.Levin (1982), "An Old Linear Programming Algorithm Runs in Polynomial Time", *23rd Symposium on Foundations of Computer Science, IEEE,* pp. 327-328.

Yamnitsky, B. (1982), "Notes on Linear Programming", Master's Thesis, Boston University.

Ye, Y. & M.Kojima (1987), "Recovering Optimal Dual Solutions in Karmarkar's Polynomial Algorithm for Linear Programming", *Mathematical Programming* **39**(3), pp. 305-317.

Ye, Y. (1987), "Karmarkar's Algorithm and the Ellipsoid Method", *Operations Research Letters* **6**(4), pp. 177-182.

Zoutendijk, G. (1960), *Methods of Feasible Directions*, Elsevier Publishing Co., Amsterdam.

Zukhovitskiy, S.I. & L.I.Avdeyeva (1966), *Linear and Convex Programming*, W.B.Saunders Company, Philadelphia and London.

Appendix A: Simplicial Algorithm of Yamnitsky and Levin (1982)

Let L be the length of the input data of a LP problem, and V_0 the volume of the smallest simplex S_0 containing the feasible region K of the problem. According to Yamnitsky (1982), given that $\forall \mathbf{v} \in K$, $\mathbf{v}$ a vertex, $|\mathbf{v}| \leq 2^L$, S_0 can be the regular simplex with edges of length 2^L. And the vertices of S_0 can be determined without much work. To outline his algorithm, some definitions are necessary .

Suppose that the regular 3-dimensional simplex (Fig A.1) is the enclosing initial polyhedron.

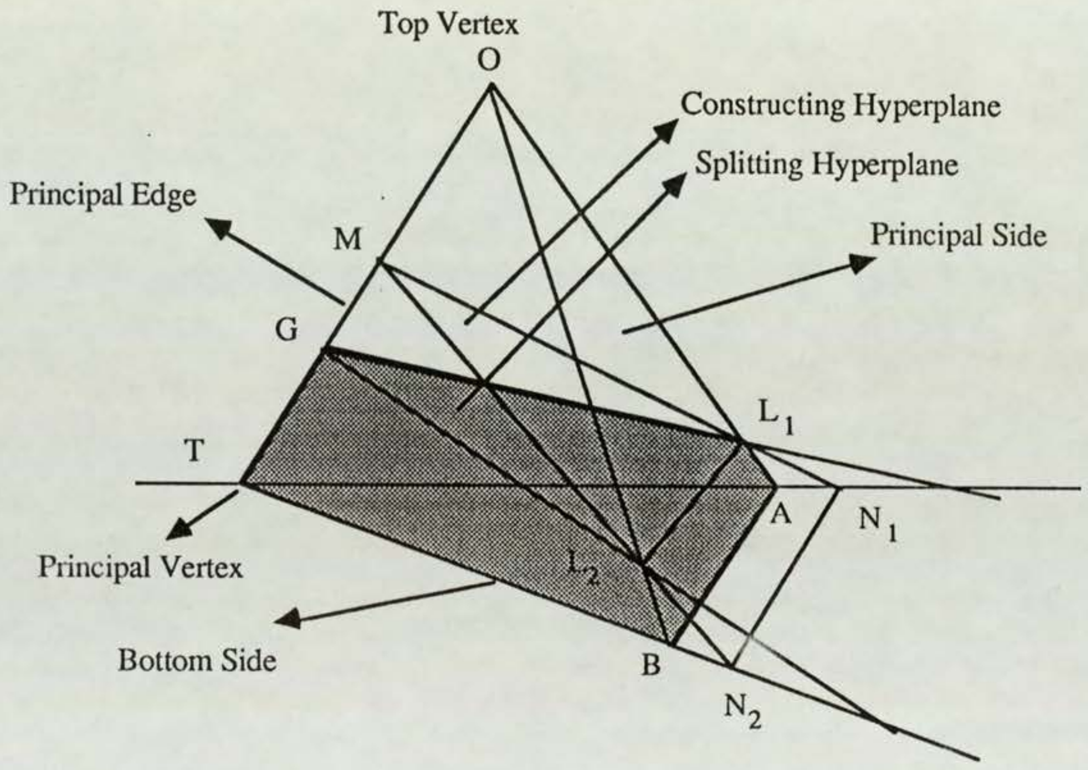


Fig A.1 Splitting and Enclosing Process of the Simplicial Algorithm

Definitions:

Top Vertex: The vertex among cut off vertices which is most distant from the cutting hyperplane. Distance here refers to the largest interval (vertex to intersection point of principal edge with cutting hyperplane.)

Set SIDE = {OTB,OTA}, sides containing the principal edge OT.

Set ET = {OB,OA}, all edges coming out of top vertex O, except the principal edge OT.

Set EB = {TB,TA}, all edges coming out of principal vertex T, except the principal edge OT.

Set L = {L_1, L_2}, all points which are the intersections of the splitting hyperplane with edges from ET.

Set N = {N_1, N_2}, all points which are the intersection of the bottom hyperplane with lines joining M to points from L.

Constructing Hyperplane: An (m-1)-dimensional hyperplane containing M and all points of L.

New Simplex: Can be constructed with m+1 points, i.e. all points of N, the principal vertex T and M.

With these definitions in mind, the algorithm may be outlined as follows.

Algorithm F:

0- Construct initial simplex S_0 containing the feasible region.

1- Construct a hyperplane passing through the center of S_0, parallel to bottom side, resulting into set points G, L_1, L_2.

2- Define the constructing hyperplane which passes through L_1, L_2 and between top vertex O and intersection of principal edge and cutting hyperplane, i.e. point G.

3- **if** the centre of the new simplex is optimal **then** stop, **else go to** 1-.

Appendix B : Cholesky Decomposition

The Cholesky method is a variant of Gauss elimination for symmetric positive semidefinite nxn-matrices. If M is such a matrix then it can be written in the factored form $M = LL^T$, as depicted below.

$$\begin{pmatrix} M_{11} & M_{12} & \dots & M_{1n} \\ M_{21} & M_{22} & \dots & \\ \cdot & & & \\ \cdot & & & \\ \cdot & & & \\ M_{n1} & M_{n2} & \dots & M_{nn} \end{pmatrix} = \begin{pmatrix} L_{11} & & & 0 \\ L_{21} & L_{22} & & \\ \cdot & & & \\ \cdot & & & \\ \cdot & & & \\ L_{n1} & L_{n2} & \dots & L_{nn} \end{pmatrix} \begin{pmatrix} L_{11} & L_{21} & \dots & L_{n1} \\ & L_{22} & \dots & L_{n2} \\ & & & \cdot \\ & & & \cdot \\ & & & \cdot \\ 0 & & & L_{nn} \end{pmatrix}.$$

L is lower triangular and sometimes called square root of M, given its similarity with the scalar case. The method, due to Cholesky and Banachiewicz is described in the following algorithm for computing L:

```
for $i = 1, \dots, n$ do

    $L_{ii} = \sqrt{M_{ii} - \sum_{j=1}^{i-1} L_{ji}^2}$        (B.1)

    $L_{ji} = \begin{cases} 0 \text{ for } j < i, \\ \frac{1}{L_{ii}}\left(M_{ji} - \sum_{k=1}^{i-1} L_{jk}L_{ik}\right), & j = i+1, \dots, n \end{cases}$        (B.2)

endfor
```

Appendix C: Sparsity Preservation and Ordering Algorithms

Sparse techniques have been used in the implementation of the simplex algorithm, for a long time, before their application in any other domaine [Gill *et al.*, 1984; Dantzig, 1963]. Commercial codes for large LP problems seem even to predate codes for sparse linear systems of equations. However, it should be noted that the simplex method reduces to the solution of a set of linear systems.

One of the main problems in solving sparse systems is that when the matrix is factored, it suffers fill-in. In other words, sparsity tends to be destroyed. In the case of the normal equations, for example, the Cholesky factor L presents more nonzeros than the lower part of A^TA. However, it has been observed that a judicious reordering of the matrix rows and columns can drastically reduce fill-in, i.e. the number of nonzeros created as a consequence of the factorization. Therefore, the computation and storage requirements are also reduced, if sparsity is exploited. Such a reordering is practically embodied in a permutation matrix, which is defined as follows.

A permutation matrix P is a square matrix whose columns are some permutation of those of the identity matrix. P is orthogonal, i.e. $P^TP = I$.

If we consider again the system of normal equations $A^TA\mathbf{x} = \mathbf{b}$, then a reordered equivalent symmetric system is the following:

$$P(A^TA)P^T(P\mathbf{x}) = P\mathbf{b}. \qquad (C.1)$$

The Cholesky method is still applicable to the above system.

The ordering problem can, thus, be defined as that of finding a permutation matrix P.

Although the main objective of ordering algorithms is to reduce the overall fill-in, there are approaches to the problem, with a different objective. That is reducing the number of operations by confining the fill-in to some part of the matrix, or locally. We can talk then about ordering algorithm with local or global strategies [Duff *et al.*, 1986]. Global strategies have the advantage of being easy to design and to implement. They are the most popular and well studied. In the following, we will present two such algorithms, namely the minimum degree algorithm of Markowitz, and the nested dissection algorithm of Alan George [George & Liu, 1981].

Before going into the presentation of the algorithms, let us see an example to illustrate the effect of reordering, on the process of factorization.

Consider the symmetric system

$$A \quad . \quad \mathbf{x} = \mathbf{b},$$

$$\begin{pmatrix} * & * & * & * & * \\ * & * & & & \\ * & & * & & \\ * & & & * & \\ * & & & & * \end{pmatrix} \begin{pmatrix} * \\ * \\ * \\ * \\ * \end{pmatrix} = \begin{pmatrix} * \\ * \\ * \\ * \\ * \end{pmatrix}.$$

The Cholesky factor of A is

$$L = \begin{pmatrix} * & & & & \\ * & * & & & \\ * & * & * & & \\ * & * & * & * & \\ * & * & * & * & * \end{pmatrix}.$$

L has nonzeros where the lower part of A has zeros. The factorization of A has caused the fill-in. However, if the system is reordered using the permutation matrix

$$P = \begin{pmatrix} & & & & 1 \\ & & & 1 & \\ & & 1 & & \\ & 1 & & & \\ 1 & & & & \end{pmatrix},$$

it becomes

$$A' \;.\; \mathbf{x}' = \mathbf{b}',$$

$$\begin{pmatrix} * & & & & * \\ & * & & & * \\ & & * & & * \\ & & & * & * \\ * & * & * & * & * \end{pmatrix} \begin{pmatrix} * \\ * \\ * \\ * \\ * \end{pmatrix} = \begin{pmatrix} * \\ * \\ * \\ * \\ * \end{pmatrix}.$$

The factorization of the A' leads to the sparse Cholesky factor

$$L' = \begin{pmatrix} * & & & & \\ & * & & & \\ & & * & & \\ & & & * & \\ * & * & * & * & * \end{pmatrix}.$$

The reduction of fill-in leads to the reduction of work from $O(n^3)$ operations for the original system to $O(n)$ operations for the reordered system.

Beside ordering algorithms, there are more direct ways of exploiting sparsity and preserving it. For some algorithms, like the Cholesky method, it is easy to see how sparsity comes to be destroyed. The computation of the cross-product A^TA is the source of the problem. Avoiding this computation is a step in the right direction, to preserve sparsity. Orthogonalization methods were discovered as a result of such strategy. Their success is, however, also limited.

C-1 Some Useful Terminology and Definitions in Graph Theory

The formal approach to the ordering problem is based on graph theory, which is appropriate for handling sparse matrices. Some terms and definitions need be given here.

A graph G consists of a set of vertices X, and a set of edges E. It is noted G = (X, E).

If graph G has n nodes then an ordering is a mapping of $\{1, 2, ..., n\}$ onto X.
Let x_i and $x_j \in X$, be two nodes of G, then $\{x_i, x_j\}$ is an edge or element of E, if the two nodes are linked.

For a symmetric nxn-matrix A, we can associate a graph $G_A = (X_A, E_A)$, where X_A is the set of diagonal elements and E_A the set of links such that $a_{ij} = a_{ji} \neq 0$ and $i \neq j$. Consider the following matrix A with diagonal elements numbered from 1 to 5.

$$A = \begin{pmatrix} (1) & * & & & * \\ * & (2) & & * & \\ & & (3) & * & \\ & * & * & (4) & * \\ * & & & * & (5) \end{pmatrix}.$$

The graph G_A of A can be represented as follows

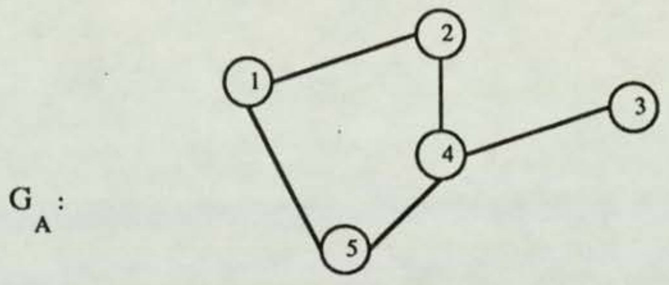


Fig C.1 Graph Representation of a Matrix

If we consider a permutation matrix $P \neq I$, the graphs of A and PAP^T are similar. However, the labelling of their nodes are different. Thus, an ordering is just a different labelling of the nodes [George & Liu, 1981].

Two nodes x_i and x_j of G are adjacent if $\{x_i , x_j\} \in E$.

The degree of a node x_i, noted $\deg(x_i)$, is the number of connections it has with other nodes. In the above graph G, $\deg(x_4) = 3$.

The adjacency structure of a graph is important in the implementation of ordering algorithms.

The *diameter* of a graph is the length of the longest path between two nodes.

Nodes being at the extremities of the longest path are termed *peripheral*, ore nodes with *highest eccentricity*.

Pseudo-Peripheral nodes are those with high eccentricity, but not with the highest eccentricity.

C-2 The Minimum Degree Algorithm

This is by far the most popular ordering algorithm. The minimum degree algorithm of Tinney is based on the Markowitz scheme for reducing fill in the solution of unsymmetric systems of linear equations by Gauss elimination [Duff *et al.*, 1986; George & Liu, 1981]. The algorithm startegy is to start reducing the columns with few entries, which corresponds to choosing nodes with least degree. This gave the name of the algorithm.

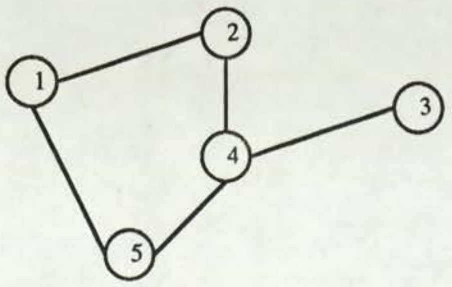
For a given symmetric graph, the algorithm may be sketched as follows.

```
1- Initilize i to 1.
2- Choose node $x_i$ with minimum degree from graph $G_{i-1} = (X_{i-1}, E_{i-1})$.
3- Eliminate node $x_i$ from $G_{i-1}$.
4- $i = i + 1$,
     **if** ( $i > Card(X)$ ) **then** stop **else** repeat from 2-.
```

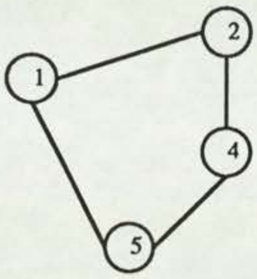
The algorithm produces a new labelling of the graph.

When choosing the node with lest degree, the situation where many such nodes are present, often arise. A random choice is then made, corresponding to a tie-breaking. However, tie-breaking strategies give different versions of the minimum degree algorithm [George & Liu, 1981].

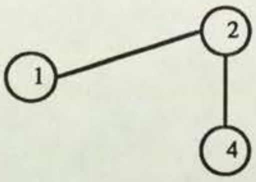
Application of the algorithm to the graph of matrix A, above, proceeds as follows.



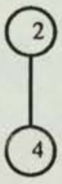
Select node $\boxed{3}$ relabel it 1.



Select node $\textcircled{5}$ relabel it 2.



Select node $\textcircled{1}$ relabel it 3.



Select node $\textcircled{2}$ relabel it 4.

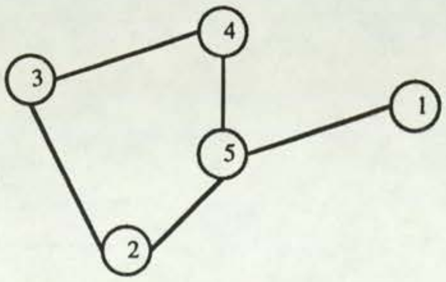
Last node $\textcircled{4}$ relabel it 5.

Fig C.2 Minimum Degree Algorithm Applied to

Graph C.1

The reordered graph becomes:

G_{PAP^T} :



The permutation matrix produced by the algorithm is

$$P = \begin{pmatrix} & & 1 & & \\ & & & 1 & \\ 1 & & & & \\ & & & & 1 \\ & 1 & & & \end{pmatrix},$$

thus

$$PAP^T = \begin{pmatrix} (3) & & & & * \\ & (5) & * & * & * \\ & * & (1) & * & \\ & * & * & (2) & * \\ * & * & & * & (4) \end{pmatrix}.$$

The underlying theory of the Markowitz strategy is beyond the scope of this work. For details see [Duff *et al.* 1986, p.128]. It must be said, however, that the algorithm is a heuristic approach and may fail to produce the "best" ordering. Proving that the produced ordering is optimal is NP-complete.

C-3 The Nested Dissection Algorithm

The nested dissection algorithm is similar to the minimum degree algorithm in that it attempts to reduce the overall fill-in, and both produce similar orderings. The nested dissection, however, according to George and Liu (1981), has the advantage of speed and modest and predictable storage requirements.

The algorithm is a devide-and-conquer method: A heuristic algorithm is used to to choose a set of nodes, (separator), such that it devides the given graph, and its matrix, into two parts of approximately equal size. The nodes of the two parts are renumbered consecutively followed by those in the separator. The process is then recursively repeated on each part and so on, until the dissection of the remaining components is no longer possible.

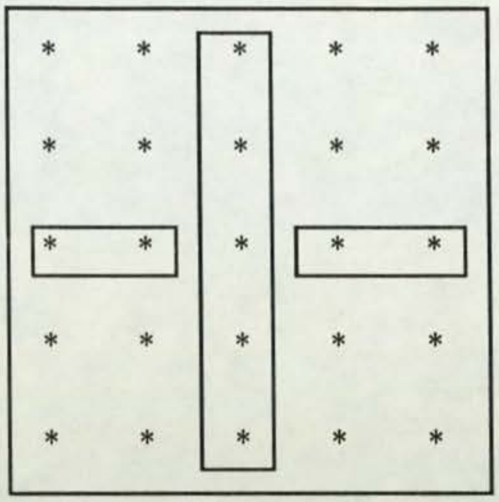
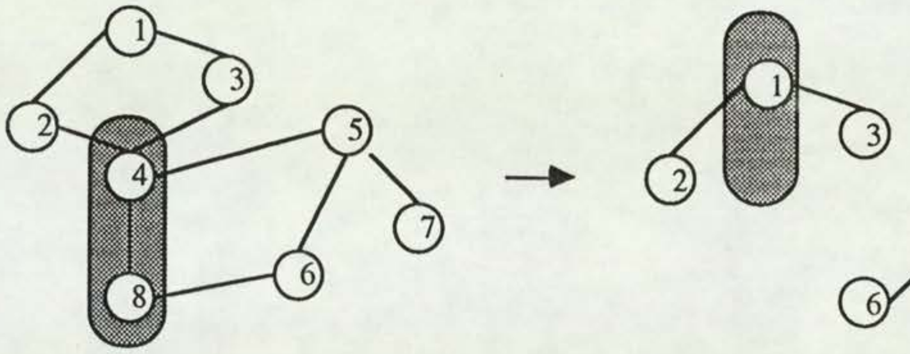


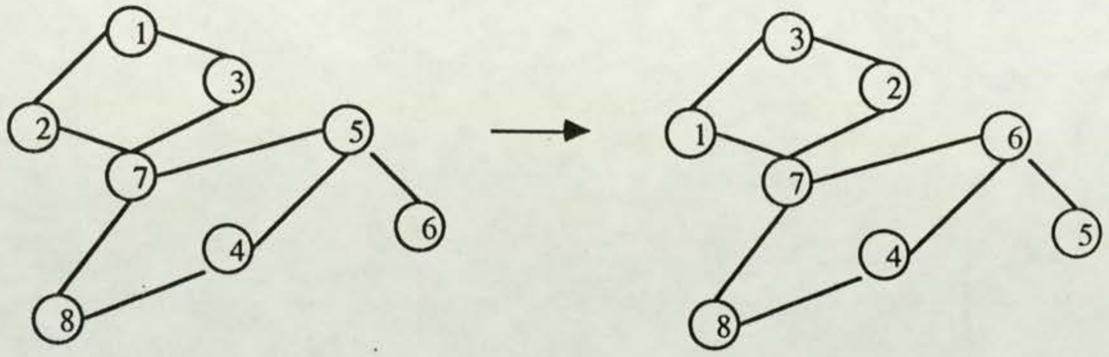
Fig C.3 Nested Dissection of a 5x5 Grid

The nested dissection algorithm may be sketched as follows.

```
0- $G_{k=1} = G_A$.
1- Disconnect the graph $G_k$ of the matrix into two subgraphs $G_i$ and $G_j$ by removing some nodes, (separator).
2- Relabel the nodes starting from those of the disconnected sub-graph, followed by those in the separator.
3- **If** card($X_i$) **and** card($X_j$) $\leq 2$ **then stop,**
   **else** repeat from 1- **for** $k = i$ and $k = j$.
   **endif**.
```

Example: Apply the Nested Dissection Ordering to the following graph





The corresponding matrix and its reordering, as in above graphs, according to the nested dissection algorithm, are:

$$
\underset{A}{\begin{pmatrix}
(1) & * & * & & & & & \\
* & (2) & & * & & & & \\
* & & (3) & * & & & & \\
 & * & * & (4) & * & & & * \\
 & & & * & (5) & * & * & \\
 & & & & * & (6) & & * \\
 & & & & * & & (7) & \\
 & & & * & & * & & (8)
\end{pmatrix}},
\qquad
\underset{PAP^T}{\begin{pmatrix}
(1) & & * & & & & * & \\
 & (2) & * & & & & * & \\
* & * & (3) & & & & & \\
 & & & (4) & & * & & * \\
 & & & & (5) & * & & \\
 & & & * & * & (6) & * & \\
* & * & & & & * & (7) & * \\
 & & & * & & & * & (8)
\end{pmatrix}}.
$$

Although the minimum degree algorithm is widely used, it can be shown that the nested dissection method is more advantageous in many cases. The major feature of the algorithm is that its performance can be analysed for some model problems, and it has been shown to produce optimum ordering for these problems (25-node problems arising in finite element). Its operation count and the fill-in introduced can be precisely identified (see Duff *et al.*, 1986). $O(n^3)$ operations are needed for nxn grids and the fill-in is $O(n^2Ln(n))$.

On the other hand there are no known formulae for the order of fill-in or operation count for the minimum degree algorithm. The difficulty of analysing the performance of the algorithm stems from the fact that tie-breaking is critical. There are many tie-breaking strategies and they greately influence the performance of the algorithm. (One such strategy is, for example, to choose the node of minimum degree which is first in the original order.) Whatever strategy used, the nested dissection algorithm seems to be superior for most problems, both in speed and storage requirements.

Appendix D : Updating Algorithm for Least Squares

Updating methods are an important feature of LSQ problems. In the real world problem data, most of the time, are incomplete. Often new observations are made after the problem has been already solved. It is crucial, therefore, to be able to incorporate the effects of these observations into the solution without having to solve it *de novo*. In our case, however, the usefulness of such techniques is mainly concerned with efficient exploitation of sparsity in the problem data. When the problem matrix is sparse except for few rows, it is attractive to discard the nonsparse rows, which will certainly cause severe fill-in in the Cholesky factor. The resulting incomplete problem is then solved and its solution updated, taking account of the removed rows. In the following we shall present an updating algorithm, due to Heath (1981), a modification of which we use in implementing some variants of Karmarkar's algorithm.

The problem to solve is

$$\min_{\mathbf{x}} \|\mathbf{b} - A\mathbf{x}\|_2. \tag{D.1}$$

Consider the partitioning of A and **b** into

$$A = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix}, \text{ and } \mathbf{b} = \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{bmatrix}.$$

Then (D.1) is written as

$$\min_{\mathbf{x}} \left\| \begin{pmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{pmatrix} - \begin{pmatrix} A_1 \\ A_2 \end{pmatrix} \mathbf{x} \right\|_2 \tag{D.1'}$$

Let $\mathbf{r}_1(\mathbf{x}) = \mathbf{b}_1 - A_1\mathbf{x}$ and $\mathbf{r}_2(\mathbf{x}) = \mathbf{b}_2 - A_2\mathbf{x}$ and solve the incomplete problem

$$\min_{\mathbf{y}} \|\mathbf{b}_1 - A\mathbf{y}_1\|_2 \tag{D.2}$$

using orthogonal factorization

$$A_1 = Q^T\begin{bmatrix} R \\ 0 \end{bmatrix} \quad \text{and} \quad b_1 = \begin{bmatrix} c \\ d \end{bmatrix}.$$

If $\mathbf{z}$ is the effect of the removed rows on the solution $\mathbf{x}$ to (D.1) then $\mathbf{x} = \mathbf{y} + \mathbf{z}$. Thus

$$\mathbf{r}_1(\mathbf{x}) = \mathbf{b}_1 - A_1(\mathbf{y} + \mathbf{z}) = \mathbf{b}_1 - A_1\mathbf{y} - A_1\mathbf{z} = \mathbf{r}_1(\mathbf{y}) - A_1\mathbf{z}. \qquad \text{(D.3)}$$

Similarly $\mathbf{r}_2(\mathbf{x}) = \mathbf{r}_2(\mathbf{y}) - A_2\mathbf{z}$.

Because $\mathbf{r}_1(\mathbf{y})$ is othogonal to the column space of A_1^T then

$$A_1^T\mathbf{r}_1(\mathbf{y}) = A_1^T(\mathbf{b}_1 - A_1\mathbf{y}) = 0.$$

From (D.3) we can write

$$A_1^T\mathbf{r}_1(\mathbf{x}) = A_1^T\mathbf{r}_1(\mathbf{y}) - A_1^TA_1\mathbf{z} = 0 - A_1^TA_1\mathbf{z}.$$

Replacing A_1^T by its QR decomposition, we can write

$$(R^T \quad 0)Q\mathbf{r}_1(\mathbf{x}) = -\,(R^T \quad 0)QA_1\mathbf{z}.$$

Thus $Q\mathbf{r}_1(\mathbf{x}) = -\,QA_1\mathbf{z}$. Since length is invariant under an orthogonal transformation, we have

$$\|\, Q\mathbf{r}_1(\mathbf{x}) \,\|_2^2 = \|\, \mathbf{r}_1(\mathbf{x}) \,\|_2^2$$

and

$$\|\, QA_1\mathbf{z} \,\|_2^2 = \|\, A_1\mathbf{z} \,\|_2^2.$$

Thus, minimizing $\|\, \mathbf{r}_1(\mathbf{x}) \,\|_2^2$ is equivalent to minimizing $\|\, A_1\mathbf{z} \,\|_2^2$. (D.2) can, therefore, be written as

$$\min_{\mathbf{z}} \left\| \begin{matrix} A_1 \mathbf{z} \\ \mathbf{r}_2(\mathbf{y}) - A_2 \mathbf{z} \end{matrix} \right\|_2 = \min_{\mathbf{z}} \left\| \begin{matrix} Q^T R\mathbf{z} \\ \mathbf{r}_2(\mathbf{y}) - A_2 R^{-1} R\mathbf{z} \end{matrix} \right\|_2 . \tag{D.4}$$

Again invariance of length under orthogonal transformations permits to write

$$\| Q^T R\mathbf{z} \|_2^2 = \| R\mathbf{z} \|_2^2.$$

(D.4) is then equivalent to

$$\min_{\mathbf{z}} \left\| \begin{matrix} R\mathbf{z} \\ \mathbf{r}_2(\mathbf{y}) - A_2 R^{-1} R\mathbf{z} \end{matrix} \right\|_2 . \tag{D.5}$$

Let $\mathbf{u} = R\mathbf{z}$ and $\mathbf{v} = \mathbf{r}_2(\mathbf{y}) - A_2 R^{-1}\mathbf{z}$ and write (D.5) as

$$\min_{\mathbf{u},\, \mathbf{v}} \left\| (A_2 \;\; I) \begin{pmatrix} \mathbf{u} \\ \mathbf{v} \end{pmatrix} - \mathbf{r}_2(\mathbf{y}) \right\|_2 . \tag{D.6}$$

More explicitly we solve the problem

$$\begin{bmatrix} A_2 R^{-1} & I \end{bmatrix} \begin{pmatrix} \mathbf{u} \\ \mathbf{v} \end{pmatrix} = \mathbf{r}_2(\mathbf{y}). \tag{D.7}$$

Using an orthogonal matrix U, (D.7) may be cast into

$$\begin{bmatrix} A_2 R^{-1} & I \end{bmatrix} U^T U \begin{pmatrix} \mathbf{u} \\ \mathbf{v} \end{pmatrix} = \mathbf{r}_2(\mathbf{y})$$

where

$$U \begin{bmatrix} R^{-T} A_2^T \\ I \end{bmatrix} = \begin{bmatrix} L^T \\ 0 \end{bmatrix}. \tag{D.8}$$

Thus

$$[L \;\; 0]\, U \begin{pmatrix} \mathbf{u} \\ \mathbf{v} \end{pmatrix} = \mathbf{r}_2(\mathbf{y})$$

or

$$[L \;\; 0] \begin{pmatrix} \mathbf{s} \\ \mathbf{t} \end{pmatrix} = \mathbf{r}_2(\mathbf{y}), \quad \text{where} \begin{pmatrix} \mathbf{s} \\ \mathbf{t} \end{pmatrix} = U \begin{pmatrix} \mathbf{u} \\ \mathbf{v} \end{pmatrix}.$$

Now, we have the triangular system $L\mathbf{s} = \mathbf{r}_2(\mathbf{y})$, which delivers $\mathbf{s}$. The vector $\mathbf{u}$ is obtained from

$$\begin{pmatrix}\mathbf{u}\\ \mathbf{v}\end{pmatrix} = U^T\begin{pmatrix}\mathbf{s}\\ 0\end{pmatrix},$$

with $\mathbf{t}$ chosen as zero to minimize the norm, and the updating vector is $\mathbf{z} = R^{-1}\mathbf{u}$.

Algorithm D

1- Solve incomplete problem (D.2) using Cholesky method or orthogonal factorization to obtain $\mathbf{y} = R^{-1}\mathbf{c}$.

2- Compute orthogonal factorization (D.8).

3- Compute $\mathbf{r}_2(\mathbf{y}) = \mathbf{b}_2 - A_2\mathbf{y}$.

4- Compute $\mathbf{s} = L^{-1}\mathbf{r}_2(\mathbf{y})$.

5- Compute

$$\begin{bmatrix}\mathbf{u}\\ \mathbf{v}\end{bmatrix} = U^T\begin{bmatrix}\mathbf{s}\\ 0\end{bmatrix}.$$

6- Compute $\mathbf{z} = R^{-1}\mathbf{u}$.

7- $\mathbf{x} = \mathbf{y} + \mathbf{z}$.

Note: Algorithm D needs storing orthogonal matrix U whose dimension is that of x. A more efficient version of this algorithm is given in Chapter 3, (Algorithm 3.1).

Appendix E: MATLAB code of Agorithm 6.3

```
% Implementation of the Partitioning Karmarkar Algorithm,
% Problem has form: min cTx s.t. Ax=b, x≥0
%
clear;
% Read problem data given in file prob
prob;
%
% Set up problem into canonical form
%
 A(:,nn+1) = b-A*ones(nn,1);
 A(:,nn+2) = -b;
 c=[zeros(1,nn),1]; phase=1; nn=nn+2;
 x=ones(1,nn); zp=10000; z0=0; z=10000; y=zeros(mm,1);
 y1=zeros(mm,1); k=0;
%
% initialize t0 to clock
 t0=clock;
%
% Main loop
%
 while abs(z0-zp)/(1+abs(zp))>0.0001
     D=diag(x(1:nn));
     Adash=A(:,1:nn)*D;
%
% Set up partitioning of Adash
%
     n0=nn-sum(n(2:nb+1));
     if phase==1,z=0;end;
     cr(1:nn)=[c(1:nn-1),-z];
     Dcr=D*cr(1:nn)';
     Dcr0=Dcr(sum(n(1:nb+1)):nn);
     resid1=Dcr0;
     resid2=[Dcr0(1:n0-1);0];
     u0=eye(n0);
     for i=1: nb
          i1=sum(m(1:i)); i2=sum(m(1:i+1))-1;
          j1=sum(n(1:i)); j2=sum(n(1:i+1))-1;
```

```
           j01=sum(n(1:nb+1));
           Bi=Adash(i1:i2, j1:j2);
           Ai=Adash(i1:i2, j01:nn);
           Dcri=Dcr(j1:j2);
%
% Apply Algorithm 6.2 to find dual variables, i.e.
% first solve each subproblem using QR method
%
           [q,r]=qr(Bi'); di=q'*Dcri;
           Rinvi=pinv(r(1:m(i+1),1:m(i+1)));
           Rinv=[Rinv,[Rinvi;zeros(max(m)-m(i+1),m(i+1))]];
           yi=Rinvi*di(1:m(i+1));
           y0=[y0;yi];
           Fi=Ai'*Rinvi; F=[F, Fi];
           u0=u0+Fi*Fi';
           resid1=resid1-Ai'*yi;
        if phase==2
           y2=[y2;yi];
           resid2=resid2-Ai'*yi;
        end;
     end;
%
% Update the solution of the incomplete least squares
% problem
%
  pinvu=pinv(u0);
  u1=pinvu*resid1;
  for i=1: nb
     i1=sum(m(1:i)); i2=sum(m(1:i+1))-1;
     y(i1:i2) = y0(i1:i2)
              + Rinv(1:m(i+1),i1:i2)*(F(1:n0,i1:i2))'*u1;
  end;
%
  if phase==2
     u2=pinvu*[zeros(1,n0-1),-1]';
     for i=1: nb
        i1=sum(m(1:i)); i2=sum(m(1:i+1))-1;
        y1(i1:i2)=Rinv(1:m(i+1),i1:i2)*(F(1:n0,i1:i2))'*u2;
     end;
%
```

```
    u3 = pinvu*resid2;
    for i=1: nb
     i1=sum(m(1:i));
     i2=sum(m(1:i+1))-1;
     y2(i1:i2) = y0(i1:i2)
               + Rinv(1:m(i+1),i1:i2)*(F(1:n0,i1:i2))'*u3;
    end;
%
     a1=[zeros(1,nn-1),-1];
     b1=[Dcr(1:nn-1)',0];
    for i=1:nb
       i1=sum(m(1:i)); i2=sum(m(1:i+1))-1;
       j1=sum(n(1:i)); j2=sum(n(1:i+1))-1;
       j01=sum(n(1:nb+1));
       a1(j1:j2)=a1(j1:j2)-y1(i1:i2)'*Adash(i1:i2,j1:j2);
       a1(j01:nn) = a1(j01:nn)
                  - y1(i1:i2)'*Adash(i1:i2,j01:nn);
       b1(j1:j2) = b1(j1:j2)-y2(i1:i2)'*Adash(i1:i2,j1:j2);
       b1(j01:nn)=b1(j01:nn)
                 - y2(i1:i2)'*Adash(i1:i2,j01:nn);
    end;
%
% Call Function supin1 or supin2 to find
% Max {z | b1 + a1*z ≥ 0}
%
     z0=supin1(a1,b1,nn,ty);
z0
     y0=y;
     yt(1:mm)=y2+z0*y1;
     if z<yt*b & (z0~=-1e20),z=yt*b;else, y=y0; z=zp;end;
     disp(yt);
     disp('Giving dual objective function'),disp(yt*b);
    end;
%
% Calculate search direction
%
   cp(1:nn)=Dcr(1:nn);
   for i=1:nb
     i1=sum(m(1:i)); i2=sum(m(1:i+1))-1;
     j1=sum(n(1:i)); j2=sum(n(1:i+1))-1;
     j01=sum(n(1:nb+1));
     cp(j1:j2)=cp(j1:j2)'-Adash(i1:i2,j1:j2)'*y(i1:i2);
     cp(j01:nn)=cp(j01:nn)'-Adash(i1:i2,j01:nn)'*y(i1:i2);
   end;
   cp(1:nn) = cp(1:nn)'
           - (c(1:nn-1)*x(1:nn-1)'z)/nn*ones(1,nn)';
   cpn(1:nn)=cp(1:nn)/norm(cp(1:nn),2);
%
% Segment to optimise step size
%
   tc=0;
   for i=1:nn
    step(i)=1/(cpn(i));
    if step(i)<0, step(i)=1e20; tc=tc+1;end;
   end;
   if tc<nn
     [beta,minb]=min(step(1:nn));
     if abs(x(minb)*x(minb)/cpn(minb))<1e-9,f=1;
     else,f=.99;end;
     s=f*beta;
   else
     s=.95;
   end;
%
     bdash=ones(1,nn)-s*cpn(1:nn);
     bbar(1:nn)=D*bdash'/(ones(1,nn)*D*bdash');
     bf=bbar(1:nn)/bbar(nn);
     k=k+1;
     x=bf;
     disp('Iteration:');disp(k);
     disp('Primal objective function value');
     if phase==2
          disp(c1(1:nn-1)*x(1:nn-1)');
     else
          disp(x(nn-1));
     end;
     if (abs(x(nn-1))<.005) & (phase==1)
          disp('End of Phase 1');
          nn=nn-1;phase=2;
          A(:,nn)=-b;
          zp=c1(1:nn-1)*x(1:nn-1)';c(1:nn-1)=c1;
          x(nn)=1;D=diag(x(1:nn));
     elseif phase == 2
          zp=c1(1:nn-1)*x(1:nn-1)';
          else
          zp=x(nn-1);
     end;
% Remove content of F, Rinv, y0 and y2
F=[]; Rinv=[]; y0=[]; y2=[];
time = etime(clock,t0)
end
disp('Primal solution'); disp(x(1:nn));
disp('Least squares solution giving dual variable values');
disp(yt);

%
% Function Supin1 is called from the program above. It is
% basically a ratio test to find z'= sup{z| b1+a1z≥0}.
% Among the values -b1(i)/a1(i) may be found z' which is
% a lower bound on the optimum objective value z* of the
% problem, i.e. z'≤ z*.
% a1 and b1 are n-vectors defined in Chapter 4
%
 function zdash = supin1 (a1, b1, n, ty)
 ratio=-b1./a1;
 disp('ratio='); disp(ratio);
 ratiomin=ones(1,n)*1e10;
 ratiomax=-ratiomin;
 if a1>zeros(1,n)
     allpos=1;
  elseif a1<zeros(1,n)
     allpos=-1;
   else allpos=0;
 end;
 if allpos==0
  for i=1:n
   if abs(a1(i))>.0005
      if a1(i)>0,ratiomax(i)=-b1(i)/a1(i);
          else,ratiomin(i)=-b1(i)/a1(i);end;
      end;
  end;
     z1=max(ratiomax);
     z2=min(ratiomin);
     zsign=-1;
     if ty=='min', zsign=1;end;
     if zsign*z2>z1, zt=z2; else zt=z1;end;
          elseif allpos==1
          zt=max(ratio);
     else
          zt=min(ratio);
end;
zdash=zt;


% The following function Supin2 is an alternative to Supin1
%
 function zdash=Supin2(a1,b1,n,ty)
 ratio=-b1(1:n)./a1(1:n);
 zgtn=-1e20;
 zgtp=-1e20;
 zltn=1e20;
 zltp=1e20;
 for i=1:n
  if abs(a1(i))>.0005
   if a1(i)>0 & b1(i)>0 & zgtn<ratio(i), zgtn=ratio(i);
   elseif a1(i)>0 & b1(i)<0 & zgtp<ratio(i), zgtp=ratio(i);
    elseif a1(i)<0 & b1(i)>0 & zltp>ratio(i),
        zltp=ratio(i);
     elseif a1(i)<0 & b1(i)<0 & zltn>ratio(i),
        zltn=ratio(i);
   end;
  end;
 end;
 if ty=='max',z0=zltn; else, z0=zltp;end;
 if zltn<1e20 & zgtp>-1e20, z0=-1e20;
  elseif zgtn>zltn, z0=-1e20;
    elseif zgtp>zltp, z0=-1e20;
 end;
zdash=z0;
```

Sample Input File

```
% Data file Little4
% Problem Little4, is a 4-block problem
% B0i, i = 1, ..., 4 corresponds to the ith block
B01=[4 2;2 5];
B02=[2 4;-1 3;4 6];
B03=[1 -2 20;1 0 2;4 -6 47;1 -2 11];
B04=[2 4 3;3 1 4;1 2 2];
% A0i, i = 1, ..., 4 corresponds to the linking variables
A01=[1 2;1 -1];
A02=[-10 1;5 0;0 -1];
A03=[-1 2;-1 -2;2 -1;3 -1];
A04=[10 0.1;0.5 1;1 -1];
% Set up canonical standard form by adding slack variables
% where necessary
B01=[B01,(-1)*eye(2),zeros(2,18),A01];
B02=[zeros(3,4),B02,(-1)*eye(3),zeros(3,13),A02];
B03=[zeros(4,9),B03,(-1)*eye(4),zeros(4,6),A03];
B04=[zeros(3,16),B04,(-1)*eye(3),A04];
% A is the problem matrix
A=[B01;B02;B03;B04];
% b the right-hand side
b=[3;4;3;2;5;3;1;3;1;5;4;3];
% c1 the cost vector
c1=[8 12 0 0 12 18 0 0 0 6 -8 48 0 0 0 0 5 11 8 0
    0 0 34 -11];
% number of variables
nn=24;
% number of constraints
mm=12;
% m(i+1) of array m contains the number of constraints in
% block i
m=[1 2 3 4 3];
% n(i+1) of array n contains the number of variables in
% block i
n=[1 4 5 7 6];
% nb = number of blocks; ty = min or max
nb=4;ty='min';
```

Appendix F: Version 4 of LPKAR1

```
C FORTRAN77 CODE OF THE KARMARKAR ALGORITHM, WHICH TAKES ACCOUNT OF
C SPARSITY. THE NESTED DISSECTION ORDERING METHOD OF GEORGE IS
C INCLUDED AS WELL AS THE PARTITIONING (UPDATING) METHOD OF HEATH.
C THESE SPARSITY PRESERVATION AND EXPLOITATION TECHNIQUES ARE DEPLOYED
C WHEN SOLVING THE LEAST SQUARES PROBLEM ARISING IN THE COMPUTATION OF
C THE SEARCH DIRECTION
C
      SUBROUTINE KRMRKR (M, N, NZ, NA, JA, RA, IT, ICOL, ALIST,
     1                          C0, C, X, RHS, D, CP)
C
      DOUBLE PRECISION D(*), CP(*), RHS(*),
     1 VAR1, VAR, BETA, FI, PF, C0(*), C(*), ALIST(*),
     1 CTX, X(*), RA(*), ALPHA
      INTEGER N, M, ITER, NZ
      INTEGER ICOL(*), IT(*)
      INTEGER NA(*), JA(*)
      INTEGER RCHLNK(1000), MRGLNK(1000), MASK(1000), PERM(1000),
     1 XADJ(1000), ADJNCY(250000), XNZSUB(1000), TEMP(1000),
     2 NZSUB(15000), XLNZ(1000), FIRST(1000), LINK(1000), INVP(1000),
     3 INVP0(1000), LS(2000), XLS(2000)
C F IS A 2-ROW ARRAY OF REALS THAT WILL CONTAIN THE DENSE THE DENSE
C  ROWS OF BT.
      DOUBLE PRECISION LNZ(50000), DIAG(1000), ROW(1000),
     1 YBAR(1000), F1(1000), F(2, 1000), STEP(2000)
C  INITIALIZATION
C  ITERATION LIMITE IS SET TO 35
      ITLIM = 35
      ITER = 1
      DO 100  I = 1, N
         X(I) =  1.0D0
 100  CONTINUE
         C(N) = 1.0D0
C  INITILIZE CPU TIME VARIABLE
      IRESLT = LIB$INIT_TIMER()
      IF ( .NOT. IRESLT ) CALL LIB$STOP(%VAL(IRESLT))
 150  CONTINUE
C START MAIN ITERATION
```

```
C
C RHS IS IDENTICALLY ZERO EXCEPT FOR ITS LAST ELEMENT RHS(N), DUE TO
C THE FACT THAT C=(0, 0, ..., 0, 1)
C
      DO 300 J = 1, N
         D(J) = X(J)
         RHS(J) = D(J) * C(J)
         CP(J) = RHS(J)
 300  CONTINUE
C
C CALCULATE CP: WE SOLVE A LINEAR LEAST SQUARES PROBLEM
C FOR THAT WE NEED TO PASS B AND RHS
C
      CALL HNDATA (M, N, ITER, NZ, RCHLNK, MRGLNK, MASK, LS, XLS,
     1 PERM, INVP, INVP0, XADJ, ADJNCY, XNZSUB, NZSUB, XLNZ, FIRST,
     1 LINK, TEMP, NA, JA, RA, IT, ICOL, ALIST, D, RHS, LNZ, DIAG,
     1 ROW, YBAR, F1, F)
C
C SOLUTION RETURNED IN RHS: COMPUTE CP = DC - B(TRANSPOSE)Y
C CP(J) HAS BEEN INITIALIZED PREVIOUSLY
C
         Z=0.0D0
         VAR1=0.0D0
         DO J = 1, N-2
            VAR1 = VAR1 + C(J)*X(J)
         ENDDO
            VAR1=VAR1+C(N)*X(N)
         DO 600 J = 1, N
            KSTRT = NA(J)
            KSTOP = NA(J+1) - 1
            DO 450 K = KSTRT, KSTOP
               I = JA(K)
               CP(J) = CP(J)-RA(K)*D(J)*RHS(I)
 450        CONTINUE
           CP(J)=CP(J)-(VAR1-Z)/DFLOAT(N)
 600     CONTINUE
C NORMALIZATION OF CP
        VAR = 0.0D0
        DO 700 J=1, N
           VAR = VAR + CP(J) * CP(J)
 700    CONTINUE
        DO J=1, N
           CP(J)=CP(J)/DSQRT(VAR)
        ENDDO
C OPTIMIZATION OF STEPSIZE
        ITC = 0
        DO I=1, N
C THE FOLLOWING TEST IS NECESSARY TO AVOID DIVISION BY ZERO
         IF (DABS(CP(I)) .EQ. 0.0D0) THEN
            STEP(I) = 1.0D20
            GO TO 301
         ENDIF
           STEP(I) = 1.0D0/CP(I)
301        IF(STEP(I) .LT. 0.0D0) THEN
              STEP(I) = 1.0D20
              ITC = ITC + 1
           ENDIF
        ENDDO
C
        IF (ITC .LT. N) THEN
           INDEX = 1
           BETA = STEP(1)
           DO I=1, N
             IF( STEP(I) .LT. BETA) THEN
               BETA=STEP(I)
               INDEX=I
             ENDIF
           ENDDO
           IF(DABS(X(INDEX)*X(INDEX)/CP(INDEX)) .LT. 1.0D-10) THEN
             FI = 1.0D0
           ELSE
             FI = .99D0
           ENDIF
          ALPHA = FI * BETA
        ELSE
          ALPHA = 0.95D0
        ENDIF
 C INVERSE TRANSFORMATION
        VAR1 = 0.0D0
        DO 800 J = 1, N
```

```
            X(J) = 1.0D0 - ALPHA * CP(J)
            X(J) = D(J) * X(J)
            VAR1 = VAR1 +  X(J)
 800    CONTINUE
        DO J=1, N
            X(J) = X(J)/VAR1
        ENDDO
        X(N) = X(N)/X(N-1)
        DO J = 1, N-1
            X(J) = X(J)/X(N-1)
        ENDDO
C TEST FOR OPTIMALITY
        CTX = 0.0D0
        DO J = 1, N-2
            CTX = CTX + C0(J) * X(J)
        ENDDO
        IF(DABS(X(N)) .GT. 1.0D-6 .AND. ITER .LT. ITLIM) THEN
        PF = 0.0D0
        DO J = 1, N
         PF = PF + DLOG(X(N)/X(J))
        ENDDO
        PRINT *, ITER, X(N)
        WRITE(11, 9994)ITER, ALPHA, X(N), CTX, PF
        ITER =ITER + 1
9994    FORMAT(I4, 4(2X, G16.10))
        GO TO 150
        ENDIF
C USED CPU TIME
        IRESLT = LIB$SHOW_TIMER()
        IF(.NOT. IRESLT) CALL LIB$STOP(%VAL(IRESLT))
        CTX=0.0D0
        DO J=1, N-2
            CTX = CTX + C0(J) * X(J)
        ENDDO
        WRITE(11, 9993) ITER, ALPHA, X(N), CTX
        WRITE(11, 9996)ITER, (J, X(J), J = 1, N )
 9993   FORMAT(I4, 3(2X, G16.10))
 9996   FORMAT(/, ' END OF SOLUTION AT ITERATION ', I6, //,
     1 ' THE SOLUTION TO PROBLEM IS X = ', //, 4(I6, G14.6))
        RETURN
```

```
      END
C
C******* GENND : GENERAL NESTED DISSECTION (GEORGE & LIU, 1981)
C
      SUBROUTINE GENND(NEQNS, XADJ, ADJNCY, MASK, PERM, XLS, LS)
      INTEGER ADJNCY(*), XADJ(*)
      INTEGER MASK(*), LS(*), PERM(*), XLS(*)
      INTEGER I, NEQNS, NSEP, NUM, ROOT
      DO 100 I=1, NEQNS
         MASK(I)=1
 100  CONTINUE
      NUM = 0
      DO 300 I= 1, NEQNS
C       FOR EACH MASKED COMPONENTS
 200    IF(MASK(I).EQ.0) GO TO 300
        ROOT=I
C       FIND A SEPARATOR AND NUMBER THE NODES NEXT.
        CALL FNDSEP(NEQNS, ROOT, XADJ, ADJNCY, MASK, NSEP,
                                PERM(NUM+1), XLS, LS)
             NUM=NUM+NSEP
             IF ( NUM.GE.NEQNS ) GO TO 400
             GO TO 200
 300  CONTINUE
C     SINCE SEPARATORS FOUND FIRST SHOULD BE ORDERED LAST, ROUTINE
C     REVRSE IS CALLED TO ADJUST THE ORDERING VECTOR.
 400  CALL REVRSE(NEQNS, PERM)
      RETURN
      END
C
C******** INVRSE: GETS THE ORIGINAL ORDERING FROM PERM
C
      SUBROUTINE INVRSE(M, PERM, INVP)
      INTEGER PERM(*), INVP(*), N, I, K
        DO 100 I=1, M
           K = PERM(I)
           INVP(K) = I
 100    CONTINUE
      RETURN
      END
```

```
C
C******** PERMRV: GETS THE ORIGINAL ORDERING OF THE VECTOR SOLUTION X
C
      SUBROUTINE PERMRV(N, RHS, PERM)
      INTEGER PERM(*), N, I, NUM, TEMP, INDEX
      DOUBLE PRECISION RHS(*), VAR
  2   INDEX = 0
      NUM = N-1
      DO 1 I = 1, NUM
         IF ( PERM(I).LT.PERM(I+1) ) GO TO 1
            TEMP = PERM (I)
            VAR = RHS (I)
            PERM(I) = PERM(I+1)
            RHS(I) = RHS(I+1)
            PERM(I+1) = TEMP
            RHS(I+1) = VAR
            INDEX = 1
  1   CONTINUE
      NUM = NUM - 1
      IF ( INDEX .NE. 0 ) GO TO 2
      RETURN
      END
C
C******** REVRSE: CHANGES ORDER OF ELEMENTS OF PERM
C
      SUBROUTINE REVRSE(NEQNS, PERM)
C
       INTEGER PERM(*), NEQNS, SAUV
       DO 10 I = 1, INT(NEQNS/2)
          SAUV = PERM(I)
          PERM(I) = PERM(NEQNS-I+1)
          PERM(NEQNS-I+1) = SAUV
 10    CONTINUE
       RETURN
       END
C
C******** FNDSEP: FIND SEPARATOR
C
      SUBROUTINE FNDSEP(NEQNS, ROOT, XADJ, ADJNCY, MASK, NSEP, SEP,
                                                   XLS, LS)
```

```
C
      INTEGER LS(*), MASK(*), SEP(NEQNS*5), XLS(*), ADJNCY(*)
      INTEGER XADJ(*), I, J, JSTOP, JSTRT, MIDBEG, MIDEND, MIDLVL,
     1        MP1END, NBR, NLVL, NODE, NSEP, ROOT, MP1BEG
C
      CALL FNROOT(NEQNS, ROOT, XADJ, ADJNCY, MASK, NLVL, XLS, LS )
C
C     IF THE NUMBER OF LEVELS IS LESS THAN 3, RETURN THE WHOLE
C     COMPONENT AS THE SEPARATOR.
C
      IF ( NLVL.GE.3 ) GO TO 200
         NSEP = XLS(NLVL+1)-1
         DO 100 I=1, NSEP
            NODE = LS(I)
            SEP(I)=NODE
            MASK(NODE)=0
 100  CONTINUE
      RETURN
C     FIND THE MIDDLE LEVEL OF THE ROOTED LEVEL STRUCTURE.
 200  MIDLVL = (NLVL+2)/2
      MIDBEG = XLS(MIDLVL)
      MP1BEG = XLS(MIDLVL + 1 )
      MIDEND = MP1BEG - 1
      MP1END = XLS(MIDLVL+2) - 1
C
C     THE SEPARATOR IS OBTAINED BY INCLUDING ONLY THOSE MIDDLE-LEVEL
C     NODES WITH NEIGHBORS IN THE MIDDLE+1 LEVEL. XADJ IS USED TEMPO-
C     RARILY TO MARK THOSE NODES IN THE MIDDLE+1 LEVEL.
C
      DO 300 I = MP1BEG, MP1END
         NODE=LS(I)
         XADJ(NODE)=-XADJ(NODE)
 300  CONTINUE
      NSEP = 0
      DO 500 I = MIDBEG, MIDEND
         NODE = LS(I)
         JSTRT= XADJ(NODE)
         JSTOP= IABS(XADJ(NODE+1)) - 1
         DO 400 J = JSTRT, JSTOP
            NBR = ADJNCY(J)
            IF ( XADJ(NBR) .GT. 0 ) GO TO 400
            NSEP=NSEP+1
            SEP(NSEP)=NODE
            MASK(NODE) = 0
            GO TO 500
  400   CONTINUE
  500   CONTINUE
C       RESET XADJ TO ITS CORRECT SIGN.
        DO 600 I = MP1BEG, MP1END
           NODE = LS(I)
           XADJ(NODE) = -XADJ(NODE)
  600   CONTINUE
        RETURN
        END
C
C*********** ROOTLS: ROOTED LEVEL STEUCTURE
C
        SUBROUTINE ROOTLS(NEQNS, ROOT, XADJ, ADJNCY, MASK, NLVL, XLS,
      1                                   LS)
C
        INTEGER ADJNCY(*), LS(*), MASK(*)
        INTEGER XLS(*), XADJ(*), I, J, JSTRT, JSTOP, LBEGIN
        INTEGER CCSIZE, LVLEND, LVSIZE,NBR,NLVL,NODE,ROOT
C
C       INITIALIZATION
C
        MASK(ROOT) = 0
        LS(1) = ROOT
        NLVL = 0
        LVLEND = 0
        CCSIZE = 1
C
C       LBEGIN IS THE POINTER TO THE BEGINNING OF THE CURRENT LEVEL, AND
C       LVLEND POINTS TO THE END OF THIS LEVEL.
C
  200   LBEGIN = LVLEND + 1
        LVLEND = CCSIZE
        NLVL = NLVL + 1
        XLS(NLVL) = LBEGIN
C
```

```
C      GENERATE THE NEXT LEVEL BY FINDING ALL THE MASKED NEIGHBORS OF
C      NODES IN THE CURRENT LEVEL.
C
       DO 400 I = LBEGIN, LVLEND
          NODE = LS(I)
          JSTRT = XADJ(NODE)
          JSTOP = XADJ(NODE + 1) - 1
          IF ( JSTOP.LT.JSTRT ) GO TO 400
             DO 300 J = JSTRT, JSTOP
                NBR = ADJNCY(J)
                IF (MASK(NBR) .EQ. 0) GO TO 300
                   CCSIZE = CCSIZE + 1
                   LS(CCSIZE) = NBR
                   MASK(NBR) = 0
 300   CONTINUE
 400   CONTINUE
C
C      COMPUTE THE CURRENT LEVEL WIDTH.
C      IF IT IS NONZERO, GENERATE THE NEXT LEVEL.
C
       LVSIZE = CCSIZE - LVLEND
       IF ( LVSIZE .GT. 0 ) GO TO 200
C
C      RESET MASK TO ONE FOR THE NODES IN THE LEVEL STRUCTURE.
C
       XLS ( NLVL + 1 ) = LVLEND + 1
       DO 500 I = 1,CCSIZE
          NODE = LS(I)
          MASK(NODE) = 1
 500   CONTINUE
       RETURN
       END
C
C********** FNROOT : FIND PSEUDO-PERIPHERAL NODE
C
       SUBROUTINE FNROOT(NEQNS,ROOT,XADJ,ADJNCY,MASK,NLVL,XLS,LS)
C
       INTEGER LS(*),MASK(*),XLS(*),
     1         ADJNCY(*)
       INTEGER XADJ(*),CCSIZE,J,JSTRT,K,KSTOP,KSTRT,MINDEG,
     1        NABOR,NDEG,NLVL,NODE,NUNLVL,ROOT,NEQNS
C
C     DTERMINE THE LEVEL TRUCTURE ROOTED AT ROOT.
C
      CALL ROOTLS(NEQNS,ROOT,XADJ,ADJNCY,MASK,NLVL,XLS,LS)
      CCSIZE = XLS(NLVL+1) - 1
      IF ( NLVL .EQ. 1 .OR. NLVL .EQ. CCSIZE ) RETURN
C
C     PICK A NODE WITH MINIMUM DEGREE FROM THE LAST LEVEL.
C
 100   JSTRT = XLS ( NLVL)
       MINDEG = CCSIZE
       ROOT = LS ( JSTRT )
       IF ( CCSIZE .EQ. JSTRT ) GO TO 400
          DO 300 J = JSTRT, CCSIZE
             NODE = LS(J)
             NDEG = 0
             KSTRT = XADJ(NODE)
             KSTOP = XADJ(NODE+1) - 1
             DO 200 K = KSTRT,KSTOP
                NABOR = ADJNCY(K)
                IF ( MASK(NABOR) .GT. 0 )  NDEG = NDEG + 1
 200         CONTINUE
             IF ( NDEG .GE. MINDEG ) GO TO 300
                ROOT = NODE
                MINDEG = NDEG
 300      CONTINUE
C
C     AND GENERATE ITS ROOTED LEVEL STRUCTURE.
C
 400   CALL ROOTLS(NEQNS,ROOT,XADJ,ADJNCY,MASK,NUNLVL,XLS,LS)
       IF ( NUNLVL .LE. NLVL ) RETURN
          NLVL = NUNLVL
          IF ( NLVL .LT. CCSIZE ) GO TO 100
          RETURN
       END
C
C************ SMBFCT : SYMBOLIC FACTORISZATION
C
       SUBROUTINE SMBFCT ( NEQNS,XADJ,ADJNCY,PERM,INVP,XLNZ,MAXLNZ,
     1                XNZSUB,NZSUB,MAXSUB,RCHLNK,MRGLNK,MARKER,FLAG)
C
        INTEGER INVP(*),MRGLNK(*),NZSUB(*),
     1          RCHLNK(*),MARKER(*),PERM(*),
     1          ADJNCY(*)
        INTEGER XADJ(*),XLNZ(*),XNZSUB(*),
     1          FLAG,I,INZ,J,JSTOP,JSTRT,K,KNZ,
     1          KXSUB,MRGK,LMAX,M,MAXLNZ,MAXSUB,
     1          NABOR,NEQNS,NODE,NP1,NZBEG,NZEND,
     1          RCHM, MRKFLG
C
C      INITIALIZATION
C
        NZBEG = 1
        NZEND = 0
        XLNZ(1) = 1
        DO 100 K = 1, NEQNS
           MRGLNK(K) = 0
           MARKER(K) = 0
 100    CONTINUE
C
C      FOR EACH COLUMN.... KNZ COUNTS THE NUMBER OF NONZEROS IN
C      COLUMN K ACCUMULATED IN RCHLNK.
C
        NP1 = NEQNS + 1
        DO 1500 K = 1, NEQNS
           KNZ = 0
           MRGK = MRGLNK(K)
           MRKFLG = 0
           MARKER (K) = K
           IF ( MRGK .NE. 0 ) MARKER(K) = MARKER(MRGK)
           XNZSUB(K) = NZEND
           NODE = PERM(K)
           JSTRT = XADJ(NODE)
           JSTOP = XADJ(NODE + 1) - 1
           IF ( JSTRT.GT.JSTOP ) GO TO 1500
C
C       USE RCHLNK TO LINK THROUGH THE STRUCTURE OF A(*,K) BELOW
C       DIAGONAL.
C
```

```
            RCHLNK(K) = NP1
            DO 300 J = JSTRT, JSTOP
               NABOR = ADJNCY(J)
               NABOR = INVP(NABOR)
               IF ( NABOR .LE. K ) GO TO 300
                  RCHM = K
 200              M = RCHM
                  RCHM = RCHLNK(M)
                  IF ( RCHM .LE. NABOR ) GO TO 200
                     KNZ = KNZ + 1
                     RCHLNK(M) = NABOR
                     RCHLNK(NABOR) = RCHM
                     IF ( MARKER(NABOR) .NE. MARKER(K) ) MRKFLG = 1
 300        CONTINUE
C
C         TEST FOR MASS SYMBOLIC ELIMINATION
C
            LMAX = 0
            IF ( MRKFLG .NE. 0 .OR. MRGK .EQ.0 ) GO TO 350
            IF ( MRGLNK (MRGK) .NE. 0 ) GO TO 350
            XNZSUB(K) = XNZSUB(MRGK) + 1
            KNZ = XLNZ(MRGK + 1) - (XLNZ(MRGK) + 1)
            GO TO 1400
C
C         LINK THROUGH EACH COLUMN I THAT AFFECTS L(*,K).
C
 350        I = K
 400        I = MRGLNK(I)
            IF (I.EQ.0) GO TO 800
               INZ = XLNZ(I+1) - (XLNZ(I)+1)
               JSTRT = XNZSUB(I) + 1
               JSTOP = XNZSUB(I) + INZ
               IF (INZ.LE.LMAX) GO TO  500
                  LMAX = INZ
                  XNZSUB(K) = JSTRT
C
C          MERGE STRUCTURE OF L(*,I) IN NZSUB INTO RCHLNK.
C
 500              RCHM = K
                  DO 700 J = JSTRT, JSTOP
```

```
                       NABOR = NZSUB(J)
 600                   M = RCHM
                       RCHM =RCHLNK(M)
                       IF (RCHM.LT.NABOR) GO TO 600
                       IF (RCHM.EQ.NABOR) GO TO 700
                          KNZ = KNZ + 1
                          RCHLNK(M) = NABOR
                          RCHLNK(NABOR) = RCHM
                          RCHM = NABOR
 700                 CONTINUE
                     GO TO 400
C
C          CHECK IF SUBSCRIPTS DUPLICATE THOSE OF ANOTHER COLUMN.
C
 800            IF (KNZ.EQ.LMAX) GO TO 1400
C
C               OR IF TAIL OF K-1ST COLUMN MATCHES HEAD OF KTH.
C
                IF (NZBEG.GT.NZEND) GO TO 1200
                   I = RCHLNK(K)
                   DO 900 JSTRT=NZBEG,NZEND
                      IF (NZSUB(JSTRT) - I) 900,1000,1200
 900               CONTINUE
                   GO TO 1200
 1000              XNZSUB(K) = JSTRT
                   DO 1100 J=JSTRT,NZEND
                      IF (NZSUB(J).NE.I) GO TO 1200
                      I = RCHLNK(I)
                      IF (I.GT.NEQNS) GO TO 1400
 1100              CONTINUE
                   NZEND = JSTRT - 1
C
C              COPY THE STRUCTURE OF L(*,K) FROM RCHLNK TO THE DATA
C              STRUCTURE  (XNZSUB, NZSUB).
C
 1200             NZBEG = NZEND + 1
                  NZEND = NZEND + KNZ
                  IF (NZEND.GT.MAXSUB) GO TO 1600
                  I = K
                  DO 1300 J=NZBEG,NZEND
```

```
                 I = RCHLNK(I)
                 NZSUB(J) = I
                 MARKER(I) = K
 1300         CONTINUE
              XNZSUB(K) = NZBEG
              MARKER(K) = K
C
C          UPDATE THE VECTOR MRGLNK. NOTE COLUMN L(*,K) JUST FOUND
C          IS REQUIRED TO DETERMINE COLUMN L(*,J), WHERE L(J,K) IS
C          THE FIRST NONZERO IN L(*,K) BELOW DIAGONAL.
C
 1400         IF (KNZ.LE.1) GO TO 1500
                 KXSUB = XNZSUB(K)
                 I = NZSUB(KXSUB)
                 MRGLNK(K) = MRGLNK(I)
                 MRGLNK(I) = K
 1500         XLNZ(K+1) = XLNZ(K) + KNZ
           MAXLNZ = XLNZ(NEQNS) - 1
           MAXSUB = XNZSUB(NEQNS)
           XNZSUB(NEQNS+1) = XNZSUB(NEQNS)
           FLAG = 0
           RETURN
C          ERROR - INSUFFICIENT STORAGE FOR NONZERO SUBSCRIPTS.
 1600      FLAG = 1
           RETURN
           END
C
C********** : GENERAL SPARSE SYMMETRIC FACTORIZATION
C
      SUBROUTINE GSFCT(NEQNS,XLNZ,LNZ,XNZSUB,NZSUB,DIAG,LINK,
     1                 FIRST,TEMP,IFLAG)
C
C
        DOUBLE PRECISION DIAG(*), LNZ(*),TEMP(*), DIAGJ, LJK
        INTEGER LINK(*), NZSUB(*)
        INTEGER FIRST(*),XLNZ(*),XNZSUB(*),I,IFLAG,II,
     1          ISTOP,ISTRT,ISUB,J,K,KFIRST,NEQNS,NEWK
C
C     INITIALIZE WORKING VECTORS
C
```

```
        DO 100 I=1,NEQNS
           LINK(I) = 0
           TEMP(I) = 0.0D0
 100    CONTINUE
C      COMPUTE COLUMN L(*,J) FOR J=1, ..., NEQNS.
        DO 600 J = 1, NEQNS
C      FOR EACH COLUMN L(*, K) THAT AFFECTS L(*, J).
        DIAGJ = 0.0D0
        NEWK = LINK(J)
 200    K = NEWK
        IF ( K .EQ. 0 ) GO TO 400
           NEWK = LINK(K)
C          OUTER PRODUCT MODIFICATION OF L(*, J) BY L(*, K)
C          STARTING AT FIRST(K) OF L(*, K).
           KFIRST = FIRST(K)
           LJK    = LNZ(KFIRST)
           DIAGJ  = DIAGJ + LJK*LJK
           ISTRT = KFIRST + 1
           ISTOP = XLNZ(K+1) - 1
           IF ( ISTOP .LT. ISTRT ) GO TO 200
C             BEFORE MODIFICATION, UPDATE VECTORS FIRST AND LINK
C             FOR FUTURE MODIFICATION STEPS.
              FIRST(K) = ISTRT
              I = XNZSUB(K) + (KFIRST - XLNZ(K)) + 1
              ISUB = NZSUB(I)
              LINK (K) = LINK(ISUB)
              LINK(ISUB) = K
C             THE ACTUAL MOD IS SAVED IN VECTOR TEMP.
              DO 300 II = ISTRT, ISTOP
                 ISUB = NZSUB(I)
                 TEMP(ISUB) = TEMP(ISUB) + LNZ(II)*LJK
                 I = I + 1
 300          CONTINUE
          GO TO 200
C
C       APPLY THE MODIFICATION ACCUMULATED IN TEMP TO COLUMN L(*,J).
C
 400    DIAGJ = DIAG(J) - DIAGJ
        IF ( DIAGJ .LE. 0.0D0 ) GO TO 700
        DIAGJ = DSQRT(DIAGJ)
```

```
          DIAG(J) = DIAGJ
          ISTRT = XLNZ(J)
          ISTOP = XLNZ(J+1) - 1
          IF ( ISTOP .LT. ISTRT ) GO TO 600
             FIRST(J) = ISTRT
             I = XNZSUB(J)
             ISUB = NZSUB(I)
             LINK(J) = LINK(ISUB)
             LINK(ISUB) = J
             DO 500 II = ISTRT, ISTOP
                ISUB = NZSUB(I)
                LNZ(II) = (LNZ(II)-TEMP(ISUB))/DIAGJ
                TEMP(ISUB) = 0.0D0
                I = I + 1
 500         CONTINUE
 600      CONTINUE
          RETURN
C         ERROR - ZERO OR NEGATIVE SQUARE ROOT IN FACTORIZATION.
 700      IFLAG = 1
          RETURN
          END
C
C******** HNDATA : STORES DATA OF LEAST SQUARES PROBLEM IN APPROPRIATE
C                  DATA STRUCTURES, PERFORMS THE UPDATING OF THE
C                  SOLUTION OF THE INCOMPLETE PROBLEM AFTER REMOVING
C                  THE DENSE COLUMNS
C
        SUBROUTINE HNDATA (M,N,ITER,NZ,RCHLNK,MRGLNK,MASK,LS,XLS,PERM,
     1            INVP,INVP0,XADJ,ADJNCY,XNZSUB,NZSUB,XLNZ,FIRST,LINK,
     1                 TEMP,NA,JA,RA,IT,ICOL,ALIST,D,RHS,LNZ,DIAG,ROW
     1                  ,YBAR,F1,F )
C
        INTEGER RCHLNK(*), MRGLNK(*), MASK(*), PERM(*), INVP(*),
     1          INVP0(*), XADJ(*), ADJNCY(*), XNZSUB(*), TEMP(*),
     2          LS(*), XLS(*), NZSUB(*),XLNZ(*),FIRST(*),LINK(*),
     3          NA(*),JA(*),IT(*),ICOL(*)
C
        INTEGER I, J, K, M, N, ITER, NZ, FLAG, MAXSUB, MAXLNZ
C
        DOUBLE PRECISION  LNZ(*), DIAG(*), RHS(*), ROW(*),
     1   RA(*), ALIST(*), D(*), XNON0, VAR1, F(2, *), F0(2),
     1   F1(*), FFT(2, 2), V(2), INVFFT(2, 2), YBAR(*)
       COMMON/E1/FFT,INVFFT
C DISCARD THE TWO DENSE COLUMNS TO SET UP THE INCOMPLETE PROBLEM
        N = N - 2
C SET THE DATA STRUCTURES AND FIND ORDERING ONCE FOR ALL
        IF(ITER.LT.2) THEN
        CALL STRCTR ( M,N,NZ,IT,ICOL,XADJ,ADJNCY)
        CALL GENND(M,XADJ,ADJNCY,MASK,PERM,XLS,LS)
        FLAG = 0
C SET NUMBER OF NONZEROS TO A MAXIMUM
        MAXSUB = 110000
        CALL INVRSE(M,PERM,INVP)
        CALL SMBFCT(M,XADJ,ADJNCY,PERM,INVP,XLNZ,MAXLNZ,
     1          XNZSUB,NZSUB,MAXSUB,RCHLNK,MRGLNK,MASK,FLAG)
        PRINT *, 'NONZEROS IN R =',MAXLNZ
        WRITE(11,'(15HNONZEROS IN R =,I10)')MAXLNZ
        IF ( FLAG .EQ. 1 ) WRITE(11,'(6HFLAG =,I2)')
        WRITE(11, 9994)
9994    FORMAT('ITERAT.',3X,'ALPHA',12X,'LAMBDA',14X,'CTX',15X,'PF')
        WRITE(11, 9995)
9995    FORMAT('-------',3X,'-----',12X,'------',14X,'---',15X,'--',/)
        ENDIF
C
C COMPUTE ELEMENTS OF DIAG
C
          DO 600 I= 1, M
           KSTRT = IT(I)
           KSTOP = IT(I+1)-1
           XNON0 = 0.0D0
              DO 650 KK= KSTRT, KSTOP
                 J = ICOL(KK)
                 IF( J.GT.N ) GO TO 650
                 XNON0 = XNON0 + ALIST(KK)*ALIST(KK)*D(J)*D(J)
 650          CONTINUE
              DIAG(INVP(I)) = XNON0
 600      CONTINUE
C
C COMPUTE OFF DIAGONAL ELEMENTS OF BBT AND INSERT THEM IN LNZ
C
```

```
      DO 700 I=2,M
         KSTRT = IT(I)
         KSTOP = IT(I+1) - 1
         DO 550 JL =1, I-1
            XNON0 = 0.0D0
            DO 460 KK = KSTRT, KSTOP
               J = ICOL(KK)
               IF(J.GT.N) GO TO 460
                ISTRT = IT(JL)
                ISTOP = IT(JL+1)-1
               DO 350 IH= ISTRT, ISTOP
                   II = ICOL(IH)
                   IF(J.NE.II) GO TO 350
                   XNON0 = XNON0 + ALIST(KK)*ALIST(IH) * D(J) * D(J)
                   GO TO 460
 350           CONTINUE
 460        CONTINUE
         II = INVP(I)
         JJ = INVP(JL)
         IF(II.GT.JJ) GO TO 89
            INDEX=II
            II = JJ
            JJ = INDEX
89          LSTRT = XLNZ(JJ)
            LSTOP = XLNZ(JJ+1)-1
            IF( LSTOP.LT.LSTRT ) GO TO 550
              KSUB = XNZSUB(JJ)
            DO 100 K = LSTRT, LSTOP
               IF( NZSUB(KSUB) .EQ. II) GO TO 200
                 KSUB = KSUB + 1
 100         CONTINUE
             GO TO 550
 200         LNZ(K) = XNON0
 550        CONTINUE
 700     CONTINUE
C
C THE RIGHT-HAND SIDE OF THE INCOMPLETE SYSTEM TO BE SOLVED
C IS ALL SPARSE, THUS IT NEED NOT BE COMPUTED
      IFLAG = 0
      CALL GSFCT(M,XLNZ,LNZ,XNZSUB,NZSUB,DIAG,LINK,FIRST,
     1                 TEMP,IFLAG)
C
C RESTORE DIMENSION AND UPDATE THE SOLUTION (ALGORITHM 3.1)
        N = N + 2
C
C  COMPUTE  F = R(INVERS) B2(TRANSPOSE)
       L = N - 1
       DO  K=1, 2
          KSTRT = NA(L)
           KSTOP = NA(L+1) - 1
           DO   KK = KSTRT, KSTOP
              J = JA(KK)
              F1(INVP(J)) = RA(KK) * D(L)
           ENDDO
        L = L + 1
        CALL FORSUB ( M, XLNZ, LNZ, XNZSUB, NZSUB, DIAG,F1)
           DO  J = 1, M
             F(K,J) = F1(J)
             F1(J) = 0.0D0
           ENDDO
        ENDDO
C
C  COMPUTE f=D2c2 - B2(TRANSPOSE) y^
C  D2c2 IS THE SECOND PART OF RHS AS PER DECOMPOSITION
C
        ITEMP = N-1
        DO  I= 1, 2
           F0(I) = RHS (ITEMP)
           KSTRT = NA(ITEMP)
           KSTOP = NA(ITEMP + 1) - 1
           DO  K = KSTRT, KSTOP
              J = JA (K)
              F0(I) = F0(I) - RA(K) * D(ITEMP) * RHS(J)
           ENDDO
           ITEMP = ITEMP + 1
        ENDDO
 C COMPUTE I+FFT IN FFT
        DO  I = 1, 2
           DO  J = 1, 2
              FFT (I,J) = 0.0D0
              DO  K = 1, M
                 FFT (I, J) = FFT(I, J) + F(I, K) * F(J, K)
              ENDDO
           ENDDO
        ENDDO
C
C FFT = I + FFT
        FFT(1, 1) = FFT(1, 1) + 1.0D0
        FFT(2, 2) = FFT(2, 2) + 1.0D0
C INVERT FFT
        KOD = 0
        CALL INVERS(KOD, 2)
        IF ( KOD .NE. 1) GO TO 70003
        WRITE ( 11, 70004)
70004   FORMAT(/, 2X, ' UNSUCCESSFUL INVERSION OF FFT ', /)
        STOP
70003   CONTINUE
C
C THE INVERS OF I + FFT IS RETURNED IN INVFFT
C COMPUTE V = INVFFT * F0
C
        DO  I = 1, 2
           V(I) = 0.0D0
           DO  J = 1, 2
              V(I) = V(I) + INVFFT(I, J) * F0(J)
           ENDDO
        ENDDO
C
C COMPUTE F(TRANSPOSE)*V AND STORE IT IN YBAR
        DO  I = 1, M
           YBAR (I) = 0.0D0
           DO  J = 1, 2
              YBAR(I) = YBAR(I) + F(J, I) * V(J)
           ENDDO
        ENDDO
C
C COMPUTE R^(-1) * F(TRANSPOSE)*V, THAT IS R^(-1) * YBAR
        CALL BACSUB(M, XLNZ, LNZ, XNZSUB, NZSUB, DIAG, YBAR)
C
C COMPUTE SOLUTION OF COMPLETE PROBLEM IN RHS
```

```
      DO I=1, M
       INVP0(I) = PERM(I)
      ENDDO
      CALL PERMRV(M, YBAR, INVP0)
      DO  I = 1, M
C           RHS  (I) = RHS(I) + YBAR(I)
            RHS  (I) = YBAR(I)
      ENDDO
C     WRITE(11, 1000)(I, RHS(I), I=1, M)
      RETURN
      END
C
C******** STRCTR : FINDS THE STRUCTURE NOT THE NUMERICAL VALUES OF BBT
C
      SUBROUTINE STRCTR ( M, N, NZ, IROW, ICOL, XADJ, ADJNCY)
      INTEGER ADJNCY(*), XADJ(*), KSTRT, KSTOP,
     1        I, J, K, II, KK, L, M, N, NZ, IROW(*), ICOL(*)
C
      K=1
      DO 700 I=1, M
        XADJ(I) = K
        DO 600 L=1, I
           KSTRT = XADJ(L)
           KSTOP = XADJ(L+1)-1
           DO 10 II= KSTRT, KSTOP
              IF(ADJNCY(II) .NE. I) GO TO 10
                 ADJNCY (K) = L
                 K = K +1
 10        CONTINUE
 600     CONTINUE
         KSTRT = IROW (I)
         KSTOP = IROW (I+1) - 1
         DO 500 JL = I+1, M
             DO 400  KK = KSTRT, KSTOP
               J = ICOL (KK)
               IF(J.GT.N) GO TO 400
               ISTRT = IROW (JL)
               ISTOP = IROW (JL+1) - 1
               DO 300 IH = ISTRT, ISTOP
                  II = ICOL (IH)
                  IF (II.NE.J ) GO TO 300
                   ADJNCY(K) = JL
                   K = K + 1
                  GO TO 500
 300            CONTINUE
 400         CONTINUE
 500      CONTINUE
 700   CONTINUE
       XADJ(M+1)=K
      RETURN
      END
C
C MAIN PROGRAM : READS THE DATA OF THE LPP UNDER MPS FORMAT
C                CREATES A LIST CONTAINING THE CANONICAL FORM
C                REQUIRED BY THE KARMARKAR ALGORITHM.
C
       CHARACTER*72 CARD
       CHARACTER*12 STRING, JCOL
       CHARACTER*12 CONSTR(1000)
       CHARACTER*12 COL, RHS, OBFUNC
       CHARACTER*8  ROW1,RHS1,ROW2,RHS2
       CHARACTER*37 FMT,FMPROB,FSC205,FSCFXM,FSCAGR
       CHARACTER*22 FMTHLF,FSCORP,FSCSD,FSCTAP,FSCRS8
       CHARACTER*5  PROB
       DOUBLE PRECISION VAL1,VAL2,ZOPT, ALIST(20000)
     1 ,RA(20000),VRHS(1000),RC(2000),B(1000),CTX
       INTEGER JA(20000),IA(10000),NA(10000)
       INTEGER NC(2000),NRHS(1000)
       INTEGER IS(2000), IT(20000), IROW(20000), ICOL(20000)
       INTEGER MS(50), NS(50), M, N, ISUB
       DOUBLE PRECISION  PRHS(2000), CP(2000), D(2000), X(2000),
     1                   C(2000), CDENSE(2000)
C
C INITILIZE FORMAT
C
       DATA FMTHLF/'(                    )'/
       DATA FMPROB/'(A12,2X,A8,2X,F12.6,3X,A8,2X,F12.6)'/
       DATA FSC205/'(A12,2X,A8,2X,F12.5,3X,A8,2X,F12.5)'/
       DATA FSCFXM/'(A12,2X,A8,2X,F12.5,3X,A8,2X,F12.5)'/
       DATA FSCAGR/'(A12,2X,A8,2X,F12.6,3X,A8,2X,F12.6)'/
      DATA FSCORP/'(A12,2X,A8,2X,F12.6)'/
      DATA FSCSD /'(A12,2X,A8,2X,F12.8)'/
      DATA FSCTAP/'(A12,2X,A8,2X,F12.6)'/
      DATA FSCRS8/'(A12,2X,A8,2X,F12.8)'/
      DATA FMSCRS/'(A12,2X,A8,2X,F12.5)'/
C
      OPEN(UNIT=12, FILE='INPUT.DAT;1',STATUS='OLD')
      OPEN(UNIT=11,FILE='OUTPUT.;1',STATUS='OLD')
       WRITE(11,'(33HCASE 4: ORDERING AND PARTITIONING)')
       M = 0
       N = 0
       READ(12,'(14X,A5)')PROB
       WRITE(11,'(16HPROBLEM NAME : ,A5)')PROB
C
       IF ( PROB.EQ. 'SC205' ) FMT = FSC205
       IF ( PROB.EQ. 'SCAGR' ) FMT = FSCAGR
       IF ( PROB.EQ. 'SCORP' ) FMT = FSCORP
       IF (PROB(1:4).EQ.'SCSD')FMT = FSCSD
       IF ( PROB.EQ. 'SCFXM' ) FMT = FSCFXM
       IF ( PROB.EQ. 'SCTAP' ) FMT = FSCTAP
       IF ( PROB.EQ. 'SCRS8' ) FMT = FSCRS8
       IF (PROB(1:4).EQ.'PROB')FMT = FMPROB
C
       READ(12,'(A72)')CARD
       WRITE(11,'(A72)')CARD
C
       CALL SIZE(MS,M)
       CALL SIZE(NS,N)
       WRITE(11,20)M,N
 20    FORMAT('** M =',I6,'    ** N =',I6)
C
      READ(12,'(A72)') CARD
      READ(12,11)OBFUNC
      I = 1
 200  READ(12,11)STRING
 11   FORMAT(12A)
      IF(STRING(1:3) .NE. 'COL') THEN
        CONSTR(I) = STRING
        I = I+1
        GO TO 200
      ENDIF
C  START PROCESSING COL
      J = 0
      JCOL = '            '
      IC = 1
      I = 1
C BECAUSE THE FORMAT IS CHANGING FROM PROBLEM TO ANOTHER
C ONE IS BOUND TO READ IN A BUFFER CARD AND THEN READ
C FROM THE BUFFER TO THE VARIABLES WITH APPROPRIATE FORMAT
C
 300    READ(12,'(72A)')CARD
C
        VAL1 = 0.0D0
        VAL2 = 0.0D0
C
C IN THE CASE OF PROBLEM SCRS8, WHERE TWO FORMATS F12.5 AND
C F12.8 ARE USED, A TEST IS NEEDED
C
        IF (PROB.EQ.'SCRS8') THEN
         IF(CARD(15:22) .EQ. 'COST    ') THEN
            READ(CARD,'(A12,2X,A8,2X,F12.8)')COL,ROW1,VAL1
         ELSE
            READ(CARD,'(A12,2X,A8,2X,F12.5)')COL,ROW1,VAL1
         ENDIF
            GO TO 400
        ENDIF
C
        IF(CARD(40:48) .EQ. '         ') THEN
           FMTHLF(2:19) = FMT(2:19)
           READ(CARD,FMTHLF) COL,ROW1,VAL1
         ELSE
           READ(CARD,FMT) COL,ROW1,VAL1,ROW2,VAL2
        ENDIF
 400   CONTINUE
       IF(COL.EQ.'RHS         ') THEN
         PRINT *, 'NC(IC-1)=',NC(IC-1),J
         IF(NC(IC-1).NE.J) GO TO 500
          RA(I) = RC(IC-1)
          JA(I) = M+1
          IA(I) = NC(IC-1)
          NA(J) = NA(J) + 1
          B(M+1) = B(M+1) + RC(IC-1)
          I = I + 1
          GO TO 500
       ENDIF
       IF(COL.NE.JCOL) THEN
         J = J + 1
         JCOL = COL
        IF(J.GT.1)THEN
        IF(NC(IC-1).EQ.(J-1))THEN
C INSERT COST COEFFICIENT IN THE LAST ROW
          RA(I) = RC(IC-1)
          JA(I) = M+1
          IA(I) = J-1
          NA(J-1) = NA(J-1) + 1
          B(M+1) = B(M+1) + RC(IC-1)
          I = I + 1
         ENDIF
        ENDIF
       ENDIF
         IF(ROW1.EQ.OBFUNC(5:12)) THEN
         RC(IC) = VAL1
         NC(IC) = J
         IC = IC + 1
C HERE WE STORE THE COST VECTOR AS A DENSE VECTOR, THUS:
         CDENSE(J)=VAL1
         GO TO 450
         ENDIF
          CALL SRCHI(CONSTR,ROW1,M,ISUB)
          RA(I) = VAL1
          JA(I) = ISUB
          IA(I) = J
          NA(J) = NA(J) + 1
          B(ISUB) = B(ISUB) + VAL1
          I = I + 1
 450      IF(VAL2.NE.0.0D0) THEN
             CALL SRCHI(CONSTR,ROW2,M,ISUB)
                   RA(I) = VAL2
                   JA(I) = ISUB
                   IA(I) = J
```

```
                  NA(J) = NA(J) + 1
                  B(ISUB) = B(ISUB) + VAL2
                  I = I + 1
         ENDIF
         GO TO 300
 500     CONTINUE
C PROCESS COLUMN
      NZ = I - 1
      J = 1
 550  READ(12,'(A72)')CARD
      VAL1 = 0.0D0
      VAL2 = 0.0D0
       IF(CARD(40:48) .EQ. '         ') THEN
          FMTHLF(2:19) = FMT(2:19)
          READ(CARD,FMTHLF) RHS,RHS1,VAL1
C          WRITE(11,FMTHLF) RHS,RHS1,VAL1
       ELSE
          READ(CARD,FMT) RHS,RHS1,VAL1,RHS2,VAL2
C          WRITE(11,FMT) RHS,RHS1,VAL1,RHS2,VAL2
       ENDIF
      IF(RHS.EQ.'ENDATA      ') GO TO 600
C PROCESS RHS
      CALL SRCHI(CONSTR,RHS1,M,ISUB)
      VRHS(J) = VAL1
      NRHS(J) = ISUB
       J = J + 1
      IF(VAL2 .NE.0.0D0 ) THEN
         CALL SRCHI(CONSTR,RHS2,M,ISUB)
         VRHS(J) = VAL2
         NRHS(J) = ISUB
       J = J + 1
      ENDIF
      GO TO 550
600   CONTINUE
C WE NEED THE OPTIMUM OBJECTIVE VALUE OF THE PROBLEM
      PRINT *,'**********.**** ENTER ZOPT'
      READ(*,15)ZOPT
  15  FORMAT(F15.4)
      VRHS(J) = ZOPT
      NRHS(J) = M+1
```

```
C INTRODUCE SLACK VARIABLES FOR '=<' AND '>=' CONSTRAINTS.
C NO LOGICAL COLUMN IS ADDED TO ' = ' CONSTRAINTS.
C
      NN = N
C M + 1 COMES FROM THE FACT THAT COST VECTOR IS JUST ANOTHER ROW
      DO 700 I = 1, M
      STRING = CONSTR(I)
      IF ( STRING(1:3) .EQ. ' N ' .OR.
     1     STRING(1:3) .EQ. ' E ' ) GO TO 700
         NZ = NZ + 1
         NN = NN + 1
         JA(NZ) = I
         IA(NZ) = NN
         NA(NN) = NA(NN) + 1
      IF ( STRING(1:3) .EQ. ' G ' ) THEN
         RA(NZ) = -1.0D0
C
C AS B CONTAINS THE SUM OF ELEMENTS OF EVERY ROW THEN IT HAS TO BE
C UPDATED WHEN SLACKS ARE ADDED
C
         B(I) = B(I) - 1.0D0
      ELSEIF ( STRING(1:3) .EQ. ' L ' ) THEN
         RA(NZ) = 1.0D0
         B(I) = B(I) + 1.0D0
      ENDIF
700   CONTINUE
      I = 1
      NN = NN + 1
900   IF ( VRHS(I).EQ.0.0D0 ) GO TO 750
         NZ = NZ + 1
         JA(NZ) = NRHS(I)
         IA(NZ) = NN
         RA(NZ) = -VRHS(I)
         NA(NN) = NA(NN) + 1
C
C WE PREPARE THE VALUE OF THE LAST COLUMN OF THE PROBLEM MATRIX
C THAT IS b - Ae. NOTE THAT  B = Ae
C
         B (NRHS(I)) = VRHS(I) - B (NRHS(I))
C
```

```
C WE SET B(NRHS(I)) TO -B(NRHS(I)) AND PUT IT BACK TO ITS NORMAL
C SIGN LATER IN ORDER NOT TO MISS ELEMENTS OF B CORRESPONDING TO ZERO
C IN VRHS
C
           B(NRHS(I)) = -B(NRHS(I))
            I = I + 1
         GO TO 900
750      CONTINUE
         NN = NN + 1
         DO 850 II= 1, M+1
            NZ = NZ + 1
            JA(NZ) = II
            IA(NZ) = NN
            NA(NN) = NA(NN) + 1
            RA(NZ) = - B(II)
850      CONTINUE
C
         WRITE(11,34)NN, M+1, NZ
 34      FORMAT('Prob. under Canonical Form :',  /,
     1 'N = ', I6, 3X,', M = ', I6, 3X,', NZ = ', I6/)
C
           M = M + 1
C
C  TRANSPOSE LIST RA WHICH CONTAINS A COLUMN-WISE, INTO ALIST
C  WHICH WILL CONTAIN A ROW-WISE
C  TWO VECTORS IT AND IS ARE NEEDED TO PERFORM A FAST SPARSE
C  MATRIX TRANSPOSE
            DO 901 I = 1, M
 901           IS(I) = 0
            DO 902 I = 1, NZ
 902           IS(JA(I)) = IS(JA(I)) + 1
               IT(1) = 1
            DO 903 I = 2, M
 903           IT(I) = IT(I-1) + IS(I-1)
            DO 904 I = 1, NZ
               J = IT(JA(I))
               IROW(J) = JA(I)
               ICOL(J) = IA(I)
               ALIST(J) = RA(I)
               IT(JA(I)) = J + 1
 904      CONTINUE
C IT NOW WILL CONTAIN THE ADDRESS OF THE BEGINNING OF EACH ROW
          IS(1) = 1
          DO 905 I = 2, M+1
             IS (I) = IT(I-1)
 905      CONTINUE
          IA(1) = 1
          DO 804 I = 2, NN+1
             IA(I) = IA(I-1) + NA(I-1)
 804      CONTINUE
C  PASS DATA IN A LIST TO SUBROUTINE KRMRKR
C
       CALL KRMRKR ( M, NN, NZ, IA, JA, RA, IS, ICOL,
     1              ALIST, CDENSE, C, X, PRHS, D, CP )
C
       CLOSE(UNIT=11)
       CLOSE(UNIT=12)
       STOP
       END
C
C ********* SIZE: FINDS DIMENSIONS OF PROBLEM
C
        SUBROUTINE SIZE(MS,M)
        INTEGER MS(50),M
   2    I = 1
        K = 1
        READ(12,100) ( MS(J), J=1,12)
        WRITE(11,100) ( MS(J), J=1,12)
100     FORMAT(12I6)
   1    IF(K.GT.12) GO TO 2
        PRINT *, 'MS(I)=',MS(I)
         IF(MS(I).EQ. 0) RETURN
          M = M + MS(I)
          I = I + 1
          K = K + 1
        GO TO 1
        END
C
C ********SRCHI: FINDS CONSTRAINT INDEX
C
```

```
      SUBROUTINE SRCHI(CONSTR, CHAIN, M, ISUB)
      CHARACTER*12 CONSTR(1000)
      CHARACTER*12 STRING
      CHARACTER*8  CHAIN
      INTEGER ISUB,M,II,I
      ISUB = 0
      STRING = '            '
      DO 10 II = 1, M
         STRING = CONSTR(II)
         IF(STRING(5:12).EQ.CHAIN) THEN
           ISUB = II
           RETURN
         ENDIF
 10   CONTINUE
      PRINT *,'CHAIN NOT FOUND'
      RETURN
      END
C
C************ FORSUB: GENERAL SPARSE FORWARD SUBSTITUTION TO SOLVE
C                     TRIANGULAR SYSTEMS
C
     SUBROUTINE FORSUB (NEQNS, XLNZ, LNZ, XNZSUB, NZSUB, DIAG, RHS)
C
       DOUBLE PRECISION DIAG(*), LNZ(*), RHS(*), RHSJ
       INTEGER NZSUB(*), ISUB, J, JJ, NEQNS
       INTEGER XLNZ(*), XNZSUB(*), I, II, ISTOP, ISTRT
C
            DO 200 J = 1, NEQNS
               RHSJ = RHS(J) / DIAG(J)
               RHS(J) = RHSJ
               ISTRT = XLNZ(J)
               ISTOP = XLNZ(J+1) -1
               IF ( ISTOP .LT. ISTRT ) GO TO 200
                  I = XNZSUB(J)
                  DO 100 II = ISTRT, ISTOP
                     ISUB = NZSUB(I)
                     RHS(ISUB) = RHS(ISUB) - LNZ(II)*RHSJ
                     I = I + 1
 100              CONTINUE
 200        CONTINUE
```

```
C
      RETURN
      END
C
C********** BACSUB : SPARSE BACKWARD SUBSTITUTION  TO SOLVE
C                    UPPER TRIANGULAR SYSTEMS
C
      SUBROUTINE BACSUB (NEQNS,XLNZ,LNZ,XNZSUB,NZSUB,DIAG,RHS)
C
      DOUBLE PRECISION DIAG(*), LNZ(*), RHS(*), RHSJ, S
      INTEGER NZSUB(*), ISUB, J, JJ, NEQNS, I, II
      INTEGER XLNZ(*), XNZSUB(*), ISTOP, ISTRT
C
            J = NEQNS
            DO 500 JJ = 1, NEQNS
               S = RHS(J)
               ISTRT = XLNZ(J)
               ISTOP = XLNZ(J+1) - 1
               IF ( ISTOP .LT. ISTRT ) GO TO 400
                  I = XNZSUB(J)
                  DO 300 II = ISTRT, ISTOP
                     ISUB = NZSUB(I)
                     S = S - LNZ(II)*RHS(ISUB)
                     I=I+1
 300              CONTINUE
 400           RHS(J) = S / DIAG(J)
               J = J - 1
 500        CONTINUE
            RETURN
        END
C
C***** INVERS: FINDS INVERSE OF A SQUARE MATRIX USING
C             GAUSS ALGORITHM. IT IS USED TO INVERT 2X2 MATRICES,
C             RESULTING FROM THE 2 FULL COLUMNS ADDED TO THE PROBLEM
C
      SUBROUTINE INVERS(KOD, N)
       DIMENSION A(20), BB(20), FFT(2,2 ), INVFFT(2, 2 )
       DIMENSION B(20),C(20),IL(20),IC(20)
       DOUBLE PRECISION DET, PIVO, X, EPS,A,  BB, B, C, INVFFT, FFT
       INTEGER KOD, M, N, IL, IC, ID
      COMMON/E1/FFT, INVFFT
      EPS = 1.0D-13
      NN=N*N
      K=1
      DO 22 I=1,N
      DO 22 J=1,N
         A(K)=FFT(J,I)
         K=K+1
 22   CONTINUE
      DO 1 I=1,NN
 1       BB(I)=A(I)
C FIND MAX PIVOT
      DET=1.0D0
      NM=-N
      DO 11 M=1,N
         NM=NM+N
         IL(M)=M
         IC(M)=M
         ID=NM+M
         PIVO=BB(ID)
         DO 2 J=M,N
            JJ=J*N-N
            DO 2 I=M,N
               II=JJ+I
               IF(DABS(PIVO).GE.DABS(BB(II))) GO TO 2
                  PIVO=BB(II)
                  IL(M)=I
                  IC(M)=J
  2      CONTINUE
C PERMUTATION OF LINES AND COLUMNS
         I=IL(M)
         IF(I.LE.M) GO TO 4
         IM=M-N
         DO 3 J=1,N
            IM=IM+N
            JJ=IM-M+I
            X=-BB(IM)
            BB(IM)=BB(JJ)
  3      BB(JJ)=X
  4      J=IC(M)
```

```
          IF(J.LE.M) GOTO 6
          NJ=N*J-N
          DO 5 I=1,N
             JM=NM+I
             JJ=NJ+I
             X=-BB(JM)
             BB(JM)=BB(JJ)
  5       BB(JJ)=X
C MODIFICATION OF COLUMN
  6       IF(DABS(PIVO).GT.EPS) GOTO 7
          DET=0.0D0
          KOD=1
          RETURN
  7       DO 8 L=1,N
             IF(L.EQ.M) GOTO 8
               LL=NM+L
               BB(LL)=-BB(LL)/PIVO
  8    CONTINUE
C ALGORITHM OF GAUSS
           DO 9 I=1,N
              IM=NM+I
              II=I-N
              DO 9 J=1,N
                 II=II+N
                 IF(I.EQ.M) GO TO 9
                 IF(J.EQ.M) GO TO 9
                   JJ=II-I+M
                   BB(II)=BB(II)+BB(IM)*BB(JJ)
   9       CONTINUE
C MODIFICATION OF ROW
           MJ=M-N
           DO 10 J=1,N
              MJ=MJ+N
              IF(J.EQ.M) GOTO 10
                BB(MJ)=BB(MJ)/PIVO
  10       CONTINUE
       DET=DET*PIVO
C PIVOTING
       BB(ID)=1.0D0/PIVO
 11    CONTINUE
```

```
C PERMUTATION ON THE RESULTING MATRIX
       M=N
 12    M=M-1
       IF(M.LE.0) GOTO 16
         I=IL(M)
         IF(I.LE.M) GOTO 14
           J1=N*(M-1)
           J2=N*(I-1)
       DO 13 J=1,N
          JX=J1+J
          JY=J2+J
          X=BB(JX)
          BB(JX)=-BB(JY)
 13       BB(JY)=X
 14       J=IC(M)
          IF(J.LE.M) GOTO 12
            J1=M-N
            DO 15 I=1,N
               J1=J1+N
               J2=J1-M+J
               X=BB(J1)
               BB(J1)=-BB(J2)
 15         BB(J2)=X
         GOTO 12
 16      CONTINUE
         K=0
         DO 23 I=1,N
            DO 21 J=1,N
               JJ=J+K
               INVFFT(J,I)=BB(JJ)
 21         CONTINUE
            K=K+N
 23      CONTINUE
       RETURN
       END
C
C
```