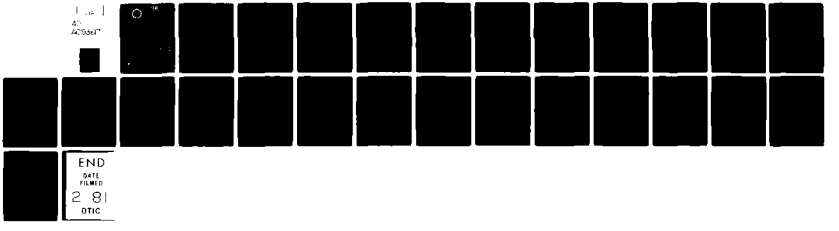
AD-A093 617 STANFORD UNIV CA SYSTEMS OPTIMIZATION LAB F/G 12/1
A NUMERICAL INVESTIGATION OF ELLIPSOID ALGORITHMS FOR LARGE-SCA--ETC(U)
OCT 80 P E GILL, W MURRAY, M A SAUNDERS DAAG29-79-C-0110
UNCLASSIFIED SOL-80-27 NL



END
DATE FILMED
2 81
DTIC

LEVEL II

Systems
Optimization
Laboratory

AD A093617

DTIC
ELECTE
JAN 0 9 1981
S D
E

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

DDC FILE COPY

Department of Operations Research
Stanford University
Stanford, CA 94305

81 1 08 070

LEVEL II

1

SYSTEMS OPTIMIZATION LABORATORY
DEPARTMENT OF OPERATIONS RESEARCH
STANFORD UNIVERSITY
STANFORD, CALIFORNIA 94305

A NUMERICAL INVESTIGATION OF ELLIPSOID ALGORITHMS FOR LARGE-SCALE LINEAR PROGRAMMING

by

Philip E. Gill, Walter Murray,
Michael A. Saunders and Margaret H. Wright

TECHNICAL REPORT SOL 80-27
October 1980

Research and reproduction of this report were supported by the Department of Energy Contract DE-AC03-76SF00326, PA No. DE-AT03-76ER72018; National Science Foundation Grants MCS-7926009 and ENG77-06761; the Office of Naval Research Contract N00014-75-C-0267; and the Army Research Office Contract DAAG29-79-C-0110.

A NUMERICAL INVESTIGATION OF ELLIPSOID ALGORITHMS FOR LARGE-SCALE LINEAR PROGRAMMING†

by

Philip E. Gill, Walter Murray,
Michael A. Saunders and Margaret H. Wright

Systems Optimisation Laboratory
Department of Operations Research
Stanford University
Stanford, CA 94305

October 1980

ABSTRACT

The ellipsoid algorithm associated with Shor, Khachiyan and others has certain theoretical properties that suggest its use as a linear programming algorithm. Some of the practical difficulties are investigated here. A variant of the ellipsoid update is first developed, to take advantage of the range constraints that often occur in linear programs (i.e., constraints of the form $l \leq a^T x \leq u$, where $u - l$ is reasonably small). Methods for storing the ellipsoid matrix are then discussed for both dense and sparse problems. In the large-scale case, a major difficulty is that the desired ellipsoid cannot be represented compactly throughout an arbitrary number of iterations. Some schemes are suggested for economizing on storage, but any guarantee of convergence is effectively lost. At this stage there remains little room for optimism that an ellipsoid-based algorithm could compete with the simplex method on problems with a large number of variables.

†This is a slightly revised version of a paper prepared for the Proceedings of the Workshop on Large-scale Linear Programming, held at IIASA, Laxenburg, Austria, June 2–6, 1980.

1. INTRODUCTION

Considerable interest has been generated by the publication (Khachiyan, 1979) of the result that a certain ellipsoid algorithm can find a feasible point for certain sets of linear inequalities in polynomial time. The algorithm defines an initial ellipsoid that encloses a finite volume of the feasible region, and proceeds by defining a sequence of shrinking ellipsoids, each of which contains this feasible region. The center of one of the ellipsoids must eventually be a feasible point, for if not, the volume of the ellipsoids will ultimately be smaller than that of the feasible region they contain (a contradiction).

It is known that a feasible-point algorithm can be adapted to solve the linear programming problem, and conversely. It is also well known that the simplex method (Dantzig, 1963), the standard technique for solving linear programs, is potentially an exponential-time algorithm, and that simple examples exist that elicit the algorithm's worst-case performance. It is therefore natural for the question to have been raised: for the solution of linear programs, could Khachiyan's algorithm prove to be superior to the simplex method?

There are several practical difficulties with the Khachiyan algorithm, and with various derivatives that have since been proposed. For large-scale problems, perhaps the most obvious difficulty is that the matrix defining the required ellipsoid is far too large to be stored. The corresponding basis matrix in the simplex method is smaller in dimension and is invariably very sparse. Also, the known practical performance of the simplex method (except on contrived examples) is exceedingly good; in fact it is safe to regard it as a *linear*-time algorithm. Therefore, for problems involving more than 100 variables (say), the hope that an ellipsoid algorithm might prove superior should never have been high.

A redeeming feature of ellipsoid algorithms is that they do not require the solution of any large systems of equations each iteration, and therefore do not require a matrix factorization for that purpose. Also, there is a degree of arbitrariness about the definition of the ellipsoid, and the general flavor is that of an iterative procedure. These properties we hope to exploit. Although the bound on the number of iterations for Khachiyan's algorithm is low compared to that of the simplex method, it is still far too large to be meaningful. In abandoning the polynomial-time features of the algorithm, little of practical value will be lost.

No attempt is made here to review the literature concerning ellipsoid algorithms (most of which is theoretical in nature). For chronological details we refer the reader to Lawler (1980), Wolfe (1980a,b), and Goffin (1980). Although Shor and others have previously developed ellipsoid algorithms in a much more general setting (e.g., Shor, 1977), we use the term "Khachiyan's algorithm" to denote the particular ellipsoid algorithm discussed by Khachiyan (1979) and later

analyzed by Gács and Lovász (1979) and Aspvall and Stone (1980).

2. SOLVING LINEAR PROGRAMS

The primal simplex method is usually implemented to solve linear programming problems in the following standard form:

LP1:	minimize	$c^T x$
	subject to	$Ax + y = b$
		$l \le x \le u$
		$l' \le y \le u'$

where A is $m \times n$. There are no restrictions on m and n, but it is usually true that $m < n$. A typical ratio is $m \approx \frac{1}{2}n$.

The slack variables y are present to allow the simplex method to be implemented using column operations only. The upper and lower bounds on x and y allow for equality and inequality constraints of all types. Many of the bounds could be infinite, but a user would usually be able to assign reasonable values to them if pressed to do so.

It is important to note that in most real-life examples a large proportion of the general constraints are equalities, so that many components of y should be zero at a solution. (Thus, for perhaps half of the constraints, the associated bounds will be $l'_i = u'_i = 0$.) It has been suggested that for each such i, some variable x_j could be eliminated. However, when so many equality constraints are present, this would duplicate much of the computational effort involved in the simplex method. Furthermore, the number of general constraints would not be reduced unless the bounds on x_j were both infinite. (For example, the simple lower bound $x_j \ge l_j$ would be transformed into a general constraint that would have to be retained, unless $l_j = -\infty$.)

Conversion to a feasible-point problem.

In order to convert a linear program to a feasible-point problem, various suggestions have been made concerning the dual of the linear program (e.g., Aspvall and Stone, 1980). Because of the inclusion upper *and* lower bounds above, the dual of LP1 is complex, and for the sake of brevity we will not give it here. It is sufficient to note that the combined primal-dual system involves a much larger number of constraints and (more significantly) a much larger number of variables than in the primal problem alone. Also, the volume of the feasible

region is essentially zero. At this stage of development, there seems little point in considering the primal-dual system.

Returning to LP1, we shall instead take the obvious approach of imposing a "target value" on the objective function. The aim will be to find a point x that achieves this value and also satisfies the primal constraints. With an ellipsoid algorithm there is no need to introduce slack variables. We shall therefore consider the problem

LP2:	find a point	x
	that satisfies	$c^T x \leq t$
		$b_l \leq Ax \leq b_u$
		$l \leq x \leq u$

This is exactly equivalent to LP1 if the bounds on Ax are suitably defined and if the target value t happens to be the optimal value for the objective.

Naturally we still wish to avoid treating a problem whose feasible region has zero volume. For test purposes we choose to perturb all of the bounds to obtain the following relaxed problem:

LP3:	find a point	x
	that satisfies	$c^T x \leq t + \delta t$
		$b_l - \delta b_l \leq Ax \leq b_u + \delta b_u$
		$l - \delta l \leq x \leq u + \delta u$

where the perturbations are positive vectors and will be substantial (rather than near rounding level). Some of the upper and lower bounds could still be $+\infty$ or $-\infty$ respectively, but any general equality constraints will now be range constraints.

In fact, it would be sufficient to relax just the bounds on equality constraints by a nontrivial amount, as long as the objective perturbation δt is also significant.

Clearly, if LP1 is well posed and has a known optimal objective t, then the feasible region for LP3 has nonzero volume. The larger the perturbations the larger the feasible region. Our reason for dealing with such problems is that they can be derived naturally from existing LP models with known solutions. Furthermore, if an ellipsoid algorithm is *not* able to solve such problems satisfactorily, then it is unlikely to be of any use on a problem whose solution is unknown.

3. AN ELLIPSOID FOR RANGE CONSTRAINTS

Since range constraints arise naturally, it is worthwhile developing an ellipsoid that takes advantage of them. It will then be possible to accomplish in one iteration an effect that would take many consecutive iterations with the earlier ellipsoid algorithms.

In order to define the necessary notation, we shall first derive the usual updating process in a fairly general way.

The linear transformation.
An ellipsoid may be represented in the form

$$(x - x_k)^T (R^T R)^{-1} (x - x_k) \leq \sigma^2, \tag{1}$$

where x_k is its center, σ is a scalar, and

$$B \equiv R^T R$$

is a positive-definite matrix. Given any nonzero vector a, we can transform this ellipsoid into a hypersphere as follows. Let the vector Ra be normalized to have unit Euclidean length, and then choose an orthonormal matrix Q ($Q^T Q = I$) that reduces it to the first column of the identity matrix:

$$\begin{aligned} \nu &= \|Ra\|_2, \\ q &= \frac{1}{\nu} Ra, \\ Qq &= e_1. \end{aligned} \tag{2}$$

(The matrix Q will not be needed in practice.) Now define some new variables z according to the linear transformation

$$x - x_k = \sigma R^T Q^T z.$$

It is easy to see that the original ellipsoid reduces to $z^T z \leq 1$, a sphere in n dimensions with center at the point $z_k = 0$ (i.e., the origin).

If $a^T x \geq l_i$ happens to be the i-th constraint on x, we can define a scalar ρ to be the "scaled residual",

$$\rho = (l_i - a^T x_k)/\sigma\nu,$$

and the corresponding constraint on z will become $z \geq \rho e_1$. If x_k violates the constraint, and if the constraint cuts the original ellipsoid, ρ will lie within the range $0 < \rho < 1$. The transformed ellipsoid and constraint are shown as the circle and the left-most vertical line in Figures 1, 2 and 3.

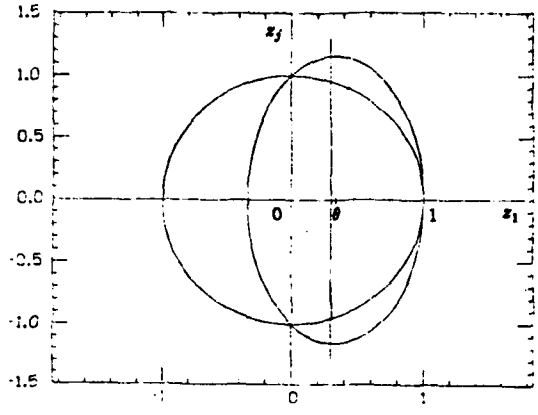


Figure 1. The original Shor/Khachiyan ellipsoid.

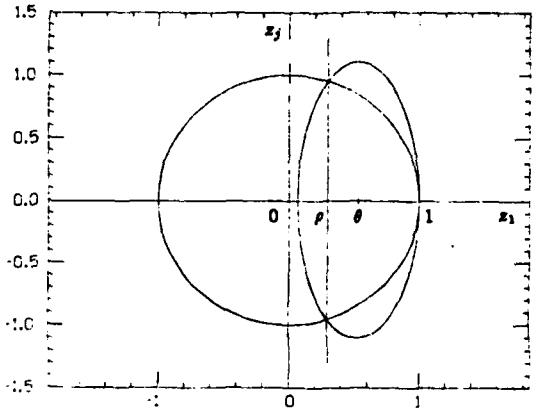


Figure 2. The deep-cut ellipsoid.

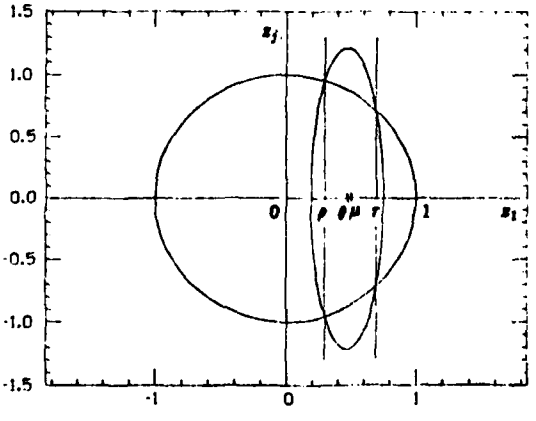


Figure 3. The range ellipsoid.

An updated ellipsoid.
A new ellipsoid with center at the point $z_{k+1} = \theta e_1$, and with all major axes equal except for the first, must take the form

$$\alpha(z_1 - \theta)^2 + \sum_{j=2}^{n} \beta z_j^2 \leq 1.$$

(The three quantities θ, α and β have yet to be specified.) In matrix notation, this is

$$(z - z_{k+1})^T \begin{pmatrix} \alpha & & \\ & \beta I_{n-1} & \end{pmatrix} (z - z_{k+1}) \leq 1,$$

which becomes

$$(z - z_{k+1})^T R^{-1} Q^T \begin{pmatrix} \alpha/\beta & & \\ & I_{n-1} & \end{pmatrix} Q R^{-T} (z - z_{k+1}) \leq \frac{\sigma^2}{\beta}$$

in the original coordinates. We wish to write this in a form analogous to equation (1), namely

$$(z - z_{k+1})^T (\bar{R}^T \bar{R})^{-1} (z - z_{k+1}) \leq \bar{\sigma}^2, \tag{3}$$

where

$$\bar{B} \equiv \bar{R}^T \bar{R} \quad \text{and} \quad \bar{\sigma}^2 \equiv \sigma^2/\beta.$$

It follows that

$$\begin{aligned} \bar{B} &= R^T Q^T (I - (1 - \frac{\beta}{\alpha}) e_1 e_1^T) Q R \\ &= R^T (I - \delta q q^T) R \\ &= R^T (I - \xi q q^T)^2 R, \end{aligned}$$

and so

$$\bar{B} = B - \delta p p^T, \tag{4}$$

$$\bar{R} = (I - \xi q q^T) R, \tag{5}$$

$$\bar{\sigma} = \sigma \gamma, \tag{6}$$

$$x_{k+1} = x_k + \sigma \theta p, \tag{7}$$

where

$$p = R^T q, \tag{8}$$

$$\gamma^2 = 1/\beta, \tag{9}$$

$$\phi^2 = \beta/\alpha, \tag{10}$$

$$\delta = 1 - \phi^2, \tag{11}$$

$$\xi = \delta/(1 + \phi). \tag{12}$$

The range ellipsoid.

We must now choose θ, α and β in some optimal way. To do this, we first specify where the surface of the new ellipsoid should be, by imposing two simple constraints. This determines α and β in terms of θ. The volume of the new ellipsoid can then be minimized with respect to θ.

Figure 1 illustrates where Khachiyan's updated ellipsoid was chosen to lie, and Figure 2 shows the so-called deep-cut ellipsoid that was later proposed by many authors. Both cases can be obtained from the "range ellipsoid", which we shall now derive.

In general, the constraint defining the above transformations will take the form

$$l_i \leq a^T x \leq u_i.$$

The lower bound gave rise to the scaled residual ρ, and the upper bound defines a similar quantity τ as follows:

$$\begin{aligned} \rho &= (l_i - a^T x_k)/\sigma\nu, \\ \tau &= (u_i - a^T x_k)/\sigma\nu, \\ \mu &= \tfrac{1}{2}(\rho + \tau), \\ \psi &= (1 - \rho^2) + (1 - \tau^2), \end{aligned} \tag{13}$$

where μ is the mid-point of the scaled range, and ψ will be useful below. The transformed range constraint is

$$\rho e_1 \leq z \leq \tau e_1, \qquad \text{or} \qquad \rho \leq z_1 \leq \tau.$$

This is shown as the two vertical lines in Figure 3. (We are considering the case where u_i is small enough for the second line to pass through the hypersphere, i.e., the inequality $\tau \leq 1$ is satisfied. Otherwise, we simply set $\tau = 1$.)

Clearly the ellipsoid in Figure 3 will contain all of the relevant feasible region if it cuts the hypersphere at the points $z_1 = \rho$ and $z_1 = \tau$. Assuming $\rho < \tau$, this gives the two conditions

$$\alpha(\rho - \theta)^2 + \beta(1 - \rho^2) = 1, \tag{14}$$

$$\alpha(\tau - \theta)^2 + \beta(1 - \tau^2) = 1, \tag{15}$$

from which we can deduce expressions for α and β in terms of θ:

$$\begin{aligned} \frac{\beta}{\alpha} &= 1 - \frac{\theta}{\mu}, \\ \frac{1}{\alpha} &= \theta^2 - \frac{1 + \rho\tau}{\mu}\theta + 1, \\ \frac{1}{\beta} &= 1 - \mu\theta + \frac{\theta}{\mu - \theta}\frac{(\tau - \rho)^2}{4}. \end{aligned} \tag{16}$$

Now the volume of the new ellipsoid relative to the hypersphere is $1/\sqrt{\alpha\beta^{n-1}}$. Hence, the volume can be minimized by choosing θ to satisfy

$$\frac{\partial}{\partial\theta}\left(\frac{1}{\alpha\beta^{n-1}}\right) = 0.$$

This leads to the quadratic equation

$$(n+1)\theta^2 - \left(2n\mu + \frac{1+\rho\tau}{\mu}\right)\theta + (n\rho\tau + 1) = 0,$$

and since it is clear from the figure that θ must lie to the left of the mid-point μ, the required root of the quadratic is given by

$$\begin{aligned} \theta &= \mu - \Delta, \\ \Delta &= \frac{1}{4\mu(n+1)}\left(\sqrt{(n^2-1)(\tau^2-\rho^2)^2 + \psi^2} - \psi\right). \end{aligned} \tag{17}$$

This completes the derivation. The optimal range ellipsoid is defined by equations (2)–(13) and (16)–(17).

Discussion.
Setting $\tau = 1$ gives the deep-cut ellipsoid in Figure 2, and $\rho = 0$, $\tau = 1$ gives the original (Shor/Khachiyan) ellipsoid in Figure 1.

If $\rho = \tau$, i.e., if the original range constraint was actually an equality constraint, equations (14) and (15) will be dependent. In this case the updated ellipsoid reduces to a subspace, and in equations (4)–(12) we would take $\theta = \rho$, $1/\alpha = 0$, $1/\beta = 1 - \rho^2$, $\phi = 0$, and $\delta = \xi = 1$.

Postscript.
A recent translation from *Kibernetika* (Shor and Gershovich, 1979) shows that both the deep-cut ellipsoid and the range ellipsoid were known to those authors prior to July 1978. (They use the terms "segment" and "layer" for the respective ellipsoid constructions.) The expression given for the center of the layer ellipsoid matches the quadratic equation above, and therefore confirms our particular derivation.

4. STORING THE ELLIPSOID

For derivation purposes we represented the ellipsoid matrix above by $B = R^TR$. The best way to store B in practice is naturally open to question. Although we are primarily concerned with large problems, it is worthwhile considering small problems first.

The dense case.

One pleasing feature of an ellipsoid algorithm is that it can be "super stable", provided a little care is taken with its implementation. By super stable we mean that rounding errors may slightly *retard* convergence but will not *prevent* it. This statement may come as a surprise, since the original algorithm has been criticized on numerical grounds. It is important to note that the mere fact that the matrix B can become ill-conditioned does not necessarily imply that the algorithm is unstable. Indeed it is quite possible for B to be singular without incurring any ill effects, provided it is represented in an appropriate manner.

The main operation in question is the recurrence of a positive-definite matrix given a rank-one modification:

$$\bar{B} = B - \delta pp^T.$$

This problem occurs in many algorithms and a fully satisfactory solution is known. In other situations we usually wish to solve some linear equations with the updated matrix, and hence it has proved beneficial to recur the Cholesky factorization $B = LDL^T$, where L is lower triangular and D is diagonal. In the present context we only need to form products of the form Ba, so "invertibility" of B or its factors is not relevant. Nevertheless, it can be helpful to recur the Cholesky factors, because there are procedures for doing so that guarantee B will retain numerical positive definiteness (e.g., Gill, Murray and Saunders, 1975). Any rounding errors incurred simply cause the computed ellipsoid to be slightly larger than it would be analytically.

If equality constraints are admitted, B will become singular and we can again update the Cholesky factors in a way that guarantees positive semidefiniteness. (Some of the diagonals of D will be exactly zero.) This would be preferable to updating B itself, but it is not safe to assume that subsequent iterates will satisfy the equality constraints to within working accuracy. To avoid unreasonable loss of feasibility with respect to the equality constraints it would be necessary to continue checking them. If one of them is not satisfied to within the required tolerance, it would be necessary to repeat the rank-reduction process. The net effect is therefore almost identical to treating equalities as range constraints.

An alternative to $B = LDL^T$ is the factorization $B = R^TR$ with R held as a dense, square matrix. On numerical grounds there is little to choose between the two approaches, even if singularity arises owing to the presence of equality constraints. (For example, Wolfe (1980b) has implemented the deep-cut ellipsoid using $\sigma^2 B = J^TJ$ with square J, and mentions performing 30,000 updates without numerical difficulty.) However, the Cholesky factorization requires less work per iteration and only half the storage, i.e., essentially the same requirements as if B itself were maintained as a symmetric matrix.

The sparse case.
For large n it is clearly not practical to represent B using dense matrices. The only obvious approach is to start with $B = I$ or some diagonal matrix (requiring minimal storage) and then update B or its factors in some kind of product form. It is known how to update Cholesky factors in the form

$$L = L_k L_{k-1} \ldots L_0,$$

where each factor L_j is triangular and can be stored compactly using two n-vectors. However, there are more efficient alternatives. From equation (4) above we see that the factorization $B = R^T R$ would give the product form

$$R = (I - \xi_k q_k q_k^T) \ldots (I - \xi_1 q_1 q_1^T) R_0, \tag{18}$$

requiring storage of a scalar ξ_j and one n-vector q_j per iteration. Even more simply, we have

$$B = B_0 - \delta_1 p_1 p_1^T - \cdots - \delta_k p_k p_k^T, \tag{19}$$

a direct summation that requires the same storage as the product form of R, but about half the work per iteration.

One possible advantage of using (18) rather than (19) is that the vectors $\{q_j\}$ are almost certainly more sparse than the vectors $\{p_j\}$, at least initially. However, since both sets of vectors rapidly become dense (particularly if the violated constraint vector a is chosen as an aggregation of several violated constraints), this advantage is of little significance. The fact that only one pass is needed through the vectors $\{p_j\}$ each iteration, weighs heavily in favor of (19).

In the dense case we argued against working with B itself. This was because the Cholesky factorization offered superior numerical reliability at no cost in terms of storage or work. In fact, it would be safe to work with B as long as there are no equality constraints or very narrow ranges, and as long as the scaled residual ρ is prevented from being very close to one.

A fundamental difficulty with both the product form (18) and the summation form (19) is that the storage required grows steadily with each iteration. This is analogous to sparse implementations of the simplex method, in which the factorization of the basis matrix tends to occupy more and more storage each time it is updated. A vital difference is that the basis matrix can be refactorized periodically in a compact form that seldom requires storage for more than $5m$ nonzeros. In other words, the simplex method's workspace can be condensed when necessary *without any loss of ground*; the next iteration will not be materially different from what it would have been had the condensation not taken place. Unfortunately this does not appear to be true for the ellipsoid algorithm.

The need to compactify storage is discussed further below under the heading of "resetting" and "cycling" strategies.

5. IMPLEMENTATION ASPECTS

An ellipsoid algorithm for solving the feasible-point problem (such as problem LP3) would ideally take the following form:

1. (Initialize.) Set $k = 0$ and choose an initial ellipsoid defined by x_0, B and σ, such that B is positive definite and at least part of the feasible region satisfies $(x - x_0)^T B^{-1}(x - x_0) \leq \sigma^2$.
2. (Terminate?) If x_k satisfies the constraints to within a required tolerance, accept it as a feasible solution and terminate.
3. (Choose a constraint.) Select, or construct, a constraint of the form $l_i \leq a^T x \leq u_i$ that is not satisfied to the desired accuracy.
4. (Update.) Obtain new quantities x_{k+1}, $\bar{B}$ and $\bar{\sigma}$ by a process such as the one described in section 3. Set $k = k + 1$ and return to step 2, using the new quantities in place of the old.

We need to consider how each step of such an algorithm might be implemented on a machine with finite precision and finite storage. It will be necessary to introduce certain changes to the algorithm, and some of these will unfortunately invalidate the proof of convergence.

Choice of initial ellipsoid.
Since the method proceeds by shrinking the volume of an ellipsoid enclosing a solution, the efficiency of the method will be doubly enhanced if the initial ellipsoid has a small volume. (Over-estimating the initial size of the ellipsoid permits x_k to move far away from x_0. It also slows the initial rate of reduction in volume.)

In some cases the user may be able provide an estimate of the distance from x_0 to the feasible region. Alternatively, it would not be unreasonable to ask the user to place sensible lower *and* upper bounds on all variables. An initial ellipsoid could then be defined from a diagonal matrix with assurance that it contained some feasible points. However, it would probably be a gross over-estimate of the dimensions that would in reality suffice.

In practice, some general procedure for choosing an initial ellipsoid is required, whether sensible bounds on the variables are available or not. Barring use of the simplex method, it seems that any procedure that is *guaranteed* to enclose part of the feasible region is likely to give an initial ellipsoid that is much too large for the subsequent algorithm to be efficient. An estimate without such guarantees may be adequate, provided we have some means of increasing the size of the ellipsoid should the estimate prove to have been too small.

The method we have used is to set the initial $\sigma^2 B$ equal to the hypersphere $\sigma^2 I$, where the radius σ is chosen so that the initial scaled residual ρ takes a

specified value, such as 0.5 or 0.1. The smaller the value of ρ the more likely the ellipsoid will be sufficiently large. However, the initial rate of convergence will be correspondingly slower.

Expanding and shrinking the ellipsoid.
It is worth noting that it is not essential for there to be a feasible point within the initial ellipsoid. Subsequent ellipsoids will always contain regions that were not contained in their predecessors. Also, at every iteration we will consider altering the current radius σ to ensure that the scaled residual satisfies

$$0 < \rho_{min} \leq \rho \leq \rho_{max} < 1.$$

(Typical values are $\rho_{min} = 0.01$, $\rho_{max} = 0.9$.) These are heuristic values to prevent the current ellipsoid from being "too big" or "too small", respectively. In particular, if $\rho > \rho_{max}$, we assume that the ellipsoid is too small and increase σ accordingly.

Unfortunately, a value of $\rho > 1$ will ultimately arise in the case where no feasible point exists. The strategy of expanding the ellipsoid therefore eliminates any hope of confirming the non-existence of a feasible point. (On the other hand, such confirmation was never a practical reality with the original algorithm either.) More seriously, if ρ_{max} becomes active and forces an increase in σ, the proof of convergence is invalidated because the ellipsoids are no longer continually shrinking.

The interpretation of a small value of ρ is that all the violated constraints pass close to the center of the ellipsoid. Under such circumstances it seems reasonable to shrink the size of the ellipsoid by reducing σ. This should not interfere too seriously with the proof of convergence. In any event, not to do so would result in a long succession of small ρ values, and the corresponding volume reduction would be negligible.

Choice of constraint.
The proof of convergence does not depend on which violated constraint is chosen to construct the next ellipsoid, but it seems reasonable to suppose that the *rate* of convergence may depend critically on the choice made. The reduction in volume of succeeding ellipsoids is greater the larger the value of ρ. Therefore it may appear that ρ should be maximized. This could be done if the quantities $a_i^T B a_i$ were recurred for each constraint vector a_i. However, additional work would be required, and in reality it would be a poor strategy. The reason is that the ellipsoid may be very oblate. (This would be certain if the problem has narrow range constraints.) The scaled residual $\rho_i = (i\text{-th violation})/(\sigma(a_i^T B a_i)^{\frac{1}{2}})$ may be very large just because $a_i^T B a_i$ is very small. It is not the volume of the

ellipsoid that is of overriding concern, since the volume can be arbitrarily small if the ellipsoid tends towards a subspace, and this can happen at any stage.

Instead of the above, we adopted the obvious strategy of choosing constraints with the largest violations. For consistency the general constraints were always scaled so that $\sum_{j=1}^{n}|a_{ij}| = 1$.

Intuitively, choosing just a single violated constraint would seem to be a myopic strategy, and indeed it was found to give a poor rate of convergence in the early iterations. It is worth noting that when B is represented in summation form, the product

$$Ba = (B_0 - \delta_1 p_1 p_1^T - \cdots - \delta_k p_k p_k^T)a$$

is computed by setting $w \leftarrow B_0 a$ and then performing the operations

$$\pi \leftarrow a^T p_j, \qquad w \leftarrow w - \pi\delta_j p_j$$

for j from 1 to k. The p_j's are stored as dense vectors, but if a is just one row a_i from the original constraint matrix it will be very sparse, and so the scalars π can be computed at negligible cost. Hence in the sparse case, an iteration can be performed almost twice as fast if violated constraints are not aggregated.

In spite of the previous comment, overall performance is usually much improved if a violated constraint is constructed from the set of all violated constraints, according to

$$a = \sum \omega_i a_i$$

for appropriate indices i and weights ω_i. The weights we have experimented with are $\omega_i = r_i$, $\sqrt{r_i}$, and 1, where r_i is the violation for the i-th constraint.

A disadvantage of this aggregated constraint (apart from giving a dense a) is that unless all of the constraints involved have reasonable upper and lower bounds, the aggregated range is unlikely to be small. Most of the advantage of the range ellipsoid will therefore be lost. One way of avoiding this drawback would be to form three separate aggregated constraints, one from any narrow range constraints (e.g., perturbed equalities), one from normal range constraints, and one from the remainder. At each iteration, one of these aggregated constraints would be chosen. However, this variation was not tried.

Resetting strategies.

Even with B represented in product or summation form, the most serious implementation difficulty is still the amount of storage required. After k updates to an initial ellipsoid we need to store k dense vectors p_j, each of length n. For large n it is clearly not practical to allow k to exceed 100 (say), and the work per iteration for such a large k would far exceed that involved in a typical iteration of the simplex method.

Ideally we would like to define a new ellipsoid at some stage, with the same center x_k and the following three properties:

1. it should have a sparse representation;
2. it should be similar in volume;
3. it should enclose the original ellipsoid.

For example, the current $\sigma^2 B$ could conceivably be replaced by $\sigma^2 \lambda I$ where λ is the largest eigenvalue of B. This would obviously satisfy properties 1 and 3. However, the new volume is likely to be considerably larger than before. We have been unable to define an ellipsoid that has all three properties, and it is probable that no such ellipsoid exists. Instead, since we have some means of increasing the size of the ellipsoid should it prove to be too small, it may be sufficient to satisfy properties 1 and 2.

The resetting strategy we have used is as follows. At some specified frequency (every K iterations where $K = 20$, say), the current ellipsoid

$$(x - x_k)^T B^{-1} (x - x_k) \leq \sigma^2$$

is replaced by

$$(x - x_k)^T D^{-1} (x - x_k) \leq (\varphi\sigma)^2,$$

where $D = \text{diag}(B)$ and $\varphi = 0.9$, provided the choice of 0.9 leads to a satisfactory value of ρ on the next iteration. It can be shown that if $\varphi = 1$, the new ellipsoid would enclose the old one along its smallest axis, but not along its largest axis. From Figure 3 we see that the volume at the fringe of the ellipsoid along the largest axis may not even be within the initial ellipsoid, so its omission may not prove to be crucial. The "reduction factor" φ is an attempt to compensate for the over-estimate that the diagonal ellipsoid makes along the shortest axis. Little can be said about how the new ellipsoid compares to the old along intermediate axes, but we would expect the shorter ones to be enclosed and the longer ones not to be.

Cycling strategies.

"Resetting" amounts to discarding all modification vectors p_j every K iterations. An alternative is to retain K modifications throughout. At each iteration a new update is added but the one from K iterations earlier is discarded. (We call this "cycling" because the new update simply overwrites the old one in storage; the point at which the replacement occurs cycles around a workspace of fixed size.)

It can be shown that at each iteration, this cyclic update gives an ellipsoid that encloses both the ellipsoid from the previous iteration *and* the ellipsoid that would be present had no discards ever been made.

Note that although updates are discarded from B, their effect on x_k and σ is not. This latter point is of some importance, since the volume of the current ellipsoid would otherwise reflect only the last K updates. Also, it is vital in this variation of the algorithm that σ decrease every iteration. This will occur only if ρ is larger than $1/n$. (Hence the introduction of ρ_{min} earlier.)

6. RESULTS AND OBSERVATIONS

Most of the ideas discussed here have been implemented in a Fortran program on an IBM 370/168. Some existing LP models were used as test problems. These were input in standard MPS format and stored in single precision. Since access is required to the rows of the constraint matrix, a row list of its nonzero elements was formed from the usual column list. All computation was performed in double precision (approximately 15 decimal digits).

The dimensions of some of the LP models are as follows:

Name	*Rows*	*Columns*	*Equalities*
WEAPON	12	100	0
SHARE2B	99	79	13
ISRAEL	175	142	0
BANDM	306	472	305
STAIR	357	467	209

Many test runs were made on these and other problems. Figures 4–6 illustrate a typical set of results. The inescapable conclusion is that even the best variant of the ellipsoid algorithm performs exceedingly poorly. The hope that a point would be reached where a "good basis" could be identified was rarely realized, even when a small sum of infeasibilities was attained. Some variants could be said to perform better than others, but the difficulties of comparison were complicated by the fact that convergence does not occur in any conventional sense. There is no quantity (such as the sum of infeasibilities) that decreases monotonically, and only in exceptional circumstances was a feasible point ever found.

The following are some tentative conclusions, observations, and (where possible) explanations.

1. There was usually a rapid initial reduction in the sum of infeasibilities. This was followed by slow but discernible convergence. Eventually the sum of infeasibilities oscillated around a steady-state value. If this value was small enough, a feasible point would occasionally be obtained by chance.
2. Choosing a single constraint is usually much worse than aggregating constraints. (Clearly, reducing one infeasibility without regard to the others will have, in the short term, an unpredictable effect on the total sum of infeasibilities.)
3. Weighting aggregated constraints by r_i can produce oscillations about a (perturbed) equality constraint; i.e., the iterates x_k are reflected back and forth across the constraint and converge only slowly towards it. This was a symptom of the single-constraint strategy when the range ellipsoid was not used. It arises when the aggregated range is too large for the features of the range ellipsoid to take effect.

4. Using $\sqrt{r_i}$ as weights tended to reduce the oscillations, because the smaller weights would soon allow some other equality constraint to dominate in the aggregation. It would sometimes result in slightly poorer performance on problems for which oscillation was not a difficulty, but it seemed to be the best compromise.

5. Choosing equal weights was often a poor strategy, since the most violated constraint would sometimes remain the same for thousands of iterations. This is the opposite extreme to oscillation and justifies the previous comment.

6. The resetting frequency K was not critical. Resetting more frequently often resulted in a more rapid initial convergence, particularly if the initial ellipsoid was very large. This was because of the reduction factor φ that was applied each reset. However, the subsequent convergence would usually be slower. The net result was a degree of invariance with respect to the frequency (assuming that even the largest K was small compared to n). The range of values tried was 5, 10, 20, 30, 40, and 50. The value $K = 20$ seems to be a reasonable value in practice. The work and storage per iteration are then roughly equivalent to refactorizing the basis in the simplex method every 50 iterations.

7. Problem SHARE2B was small enough to allow use of $K = 200$ ($= 2.2n$ — normally an unthinkable ratio). This was the one case where termination at a feasible point could reasonably be expected. However, the total work and iterations far exceeded that required by the simplex method to solve the unperturbed problem exactly.

8. In spite of its promising properties, the cycling algorithm did not perform more favorably than the resetting algorithm. In fact, since there was no artificial shrinking of σ at the end of each K iterations, both the initial rate of convergence and the overall performance tended to be worse.

9. Forcing a reduction in σ on resetting eventually results in violated constraints lying outside the current ellipsoid. *Not* forcing a reduction meant poorer initial convergence. Possibly the size of the reduction should be related to the estimated progress made during the last K iterations. Progress is ultimately so slow that this would suppress any artificial reduction each reset. This in turn would *ensure* that progress was slow or non-existent, once ρ_{max} starts forcing an increase in σ during the iterations.

10. In most cases the target objective value was reached at a point where the sum of infeasibilities was reasonably low. (Of course it could also be reached in an early iteration at the expense of gross infeasibility elsewhere.)

11. The algorithm is highly sensitive to scaling — much more so than the simplex method. Scaling the constraints by rows is essential. Scaling them by columns (i.e., scaling the variables) would doubtless help in general, but was not tried. The problem ISRAEL was one with poor column scaling, and although the algorithm was able to reduce the sum of infeasibilities by several orders of magnitude, it was unable to obtain an objective value anywhere close to the target.

12. If the original problem was really one of finding a feasible point and if the volume of the feasible region was large, the algorithm tended to perform much the same as described, but with a somewhat greater chance of terminating. A dramatic improvement could be made in this situation by shifting the constraints *inwards*, i.e., by changing $Ax \geq b$ to $Ax \geq b + \delta b$, where δb is positive. In effect, a feasible point to the original problem was then found in the rapid convergence stage of the algorithm.

13. One of the more difficult matters is to propose a stopping criterion that recognizes the time when further progress is unlikely. Of all the usual quantities that could be monitored, the most stable would probably be the size of the maximum constraint violation, r_i.

Some of these observations are illustrated in the following figures. For comparison, the simplex method was applied to unperturbed linear programs, cast in the form LP2 with t set to the optimal objective value (see section 2). Hence, all iterates except the last were infeasible. The beginning and end of the simplex iterations are marked by a cross.

The ellipsoid algorithms were applied to perturbed problems of the form LP3, with equality-constraint bounds perturbed each way by 5%. (If the i-th components of b_l and b_u were both β, the quantity $\delta\beta = 0.05(|\beta| + 1)$ was computed and the bounds on the i-th row of Ax were taken to be $\beta - \delta\beta$ and $\beta + \delta\beta$.) The perturbation to the objective value was $\delta t = 0.01t$. During the test for feasibility, a constraint was considered violated only if the residual r_i exceeded 0.05.

The curves drawn for the ellipsoid algorithms have been smoothed to show the general trend. The quantity plotted is the *minimum* sum of infeasibilities achieved during the previous 10, 20 or 50 iterations. The actual sum varies erratically with iteration number and would lie above the curves shown.

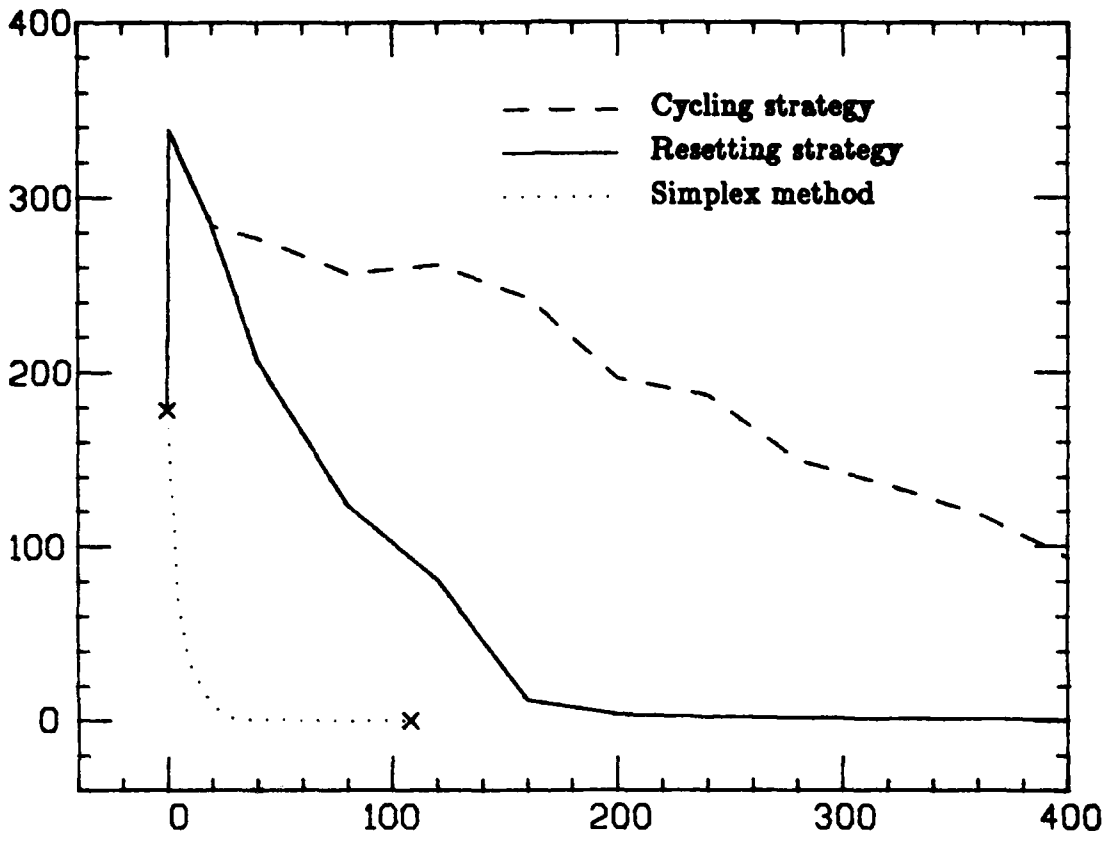


Figure 4. Problem SHARE2B. Sum of infeasibilities vs. iteration number.

Comparison of resetting and cycling strategies with large initial radius.
$K = 20, \quad \rho_0 = 0.01, \quad \sigma_0 = 6500, \quad \rho_{min} = 0.01, \quad \omega_i = \sqrt{r_i}.$

1. The sum of infeasibilities increases substantially in the early iterations. This is typical when the initial radius σ_0 is large.
2. The bound $\rho \geq \rho_{min}$ was often active, particularly after each reset. This leads to a more rapid reduction in σ for the resetting strategy, and hence to greater initial progress.
3. Approximate cpu times:
   Ellipsoid algorithm, cycling: 5.6 seconds for 400 iterations.
   Ellipsoid algorithm, resetting: 4.8 seconds for 400 iterations.
   Simplex method: 2.4 seconds for 108 iterations (solution found).

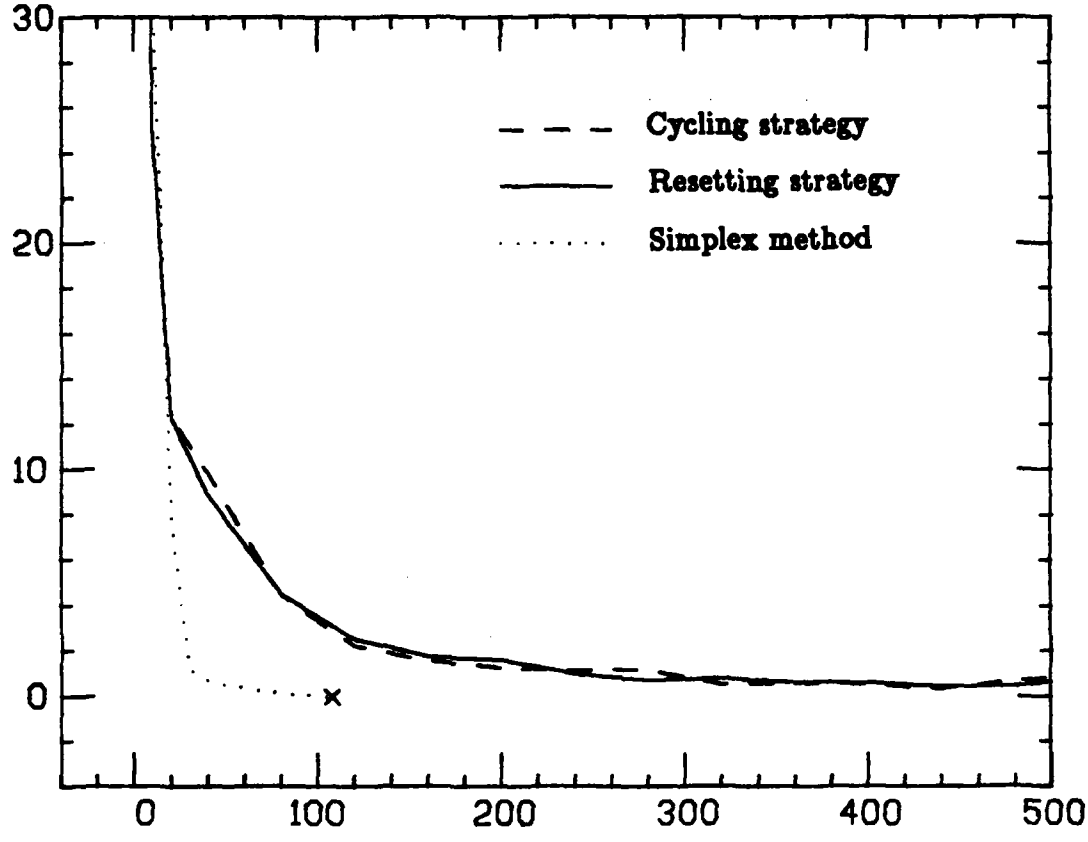


Figure 5. Problem SHARE2B. Sum of infeasibilities vs. iteration number.

Comparison of resetting and cycling strategies with smaller initial radius.
$K = 20, \quad \rho_0 = 0.1, \quad \sigma_0 = 650, \quad \rho_{min} = 0.0127 = 1/\sqrt{n}, \quad \omega_i = \sqrt{r_i}.$

1. **The bound ρ_{min} was active for both ellipsoid algorithms during early iterations, reducing σ and allowing rapid initial progress.**
2. **The finer scale used for the vertical axis illustrates the slow terminal convergence that can be expected in general.**

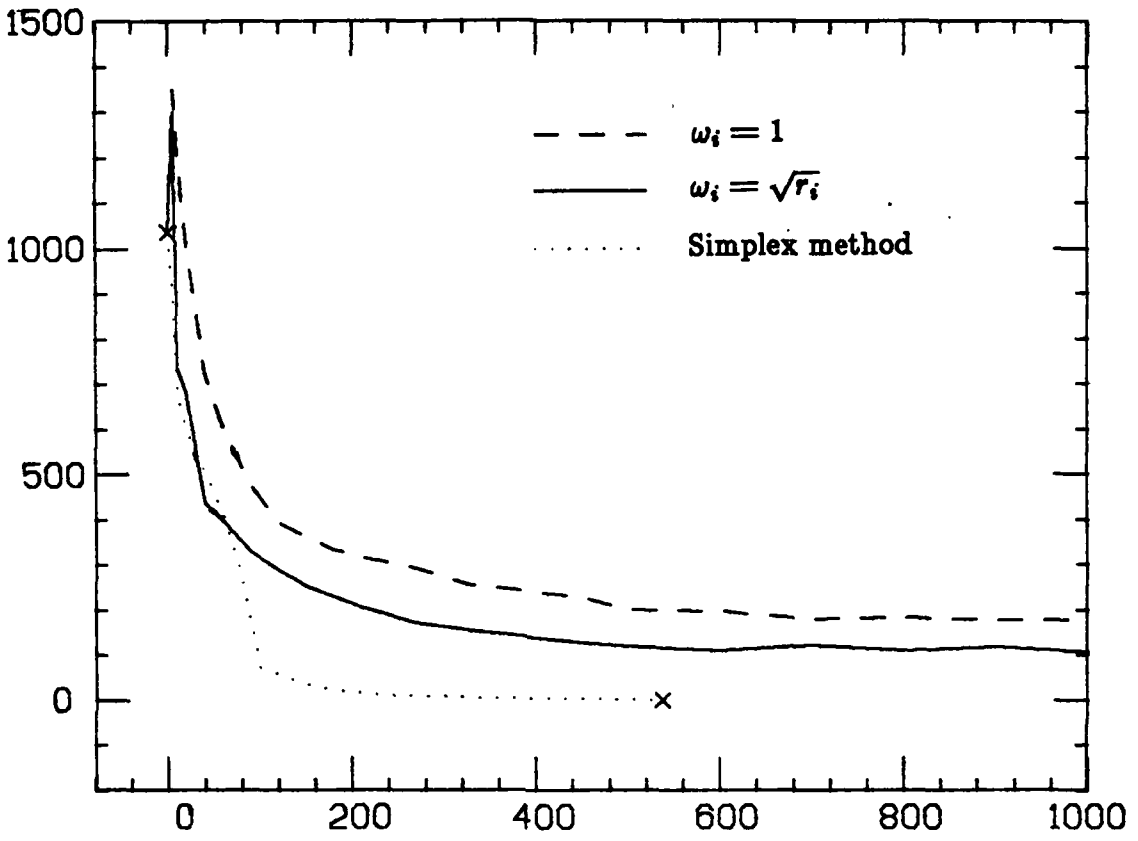


Figure 6. Problem STAIR. Sum of infeasibilities vs. iteration number.

Comparison of weights $\omega_i = 1$ and $\omega_i = \sqrt{r_i}$ in aggregated constraints.
Resetting strategy with $K = 30$.
$\rho_0 = 0.1, \quad \sigma_0 = 2500.$

1. The ellipsoid algorithms could not be expected to converge within a tolerable time.
2. This is a difficult example even for the simplex method (in which the basis factorizations are unusually dense). The workspace required by the simplex method is slightly more than that needed by the ellipsoid algorithms (storing 30 updates).
3. Approximate cpu times:
   Ellipsoid algorithms: 53 seconds for 1000 iterations.
   Simplex method: 50 seconds for 537 iterations (solution found).

Final comments.

The main hope was that a feasible point to a perturbed linear program could be found that would suggest a good set of basic variables in the LP sense. This hope was not realized. There are several reasons for this, particularly with large problems. In practice, there are frequently many Lagrange multipliers (dual variables) that are zero or close to zero. This means that a feasible point to the perturbed problem need not be close to any vertex. Also, the size of the perturbations used was sufficiently large that for a many-variable problem, an accumulation of small changes in some variables could allow other variables to take on values quite unlike their optimal values.

Given a system of linear equations $Bx = b$, the effect on x of a perturbation to b has been studied at length (e.g., Wilkinson, 1965). For example, if the components of b were perturbed by 1% and if the condition number of B were greater than 100, then the perturbed solution may be different from x in all figures. In most linear programs, we would expect the condition number of the basis matrix to be 100 at least. Hence, even if a feasible solution could be found to a perturbed problem, we could not expect to recognize the solution if the perturbations were as much as 1%. On the other hand, for problems containing any equality constraints, the hope for obtaining a solution when the perturbations are as *small* as 1% would seem to be remote.

Regarding convergence, the difficulty remains that if the problem is too large to allow the ellipsoid matrix B to be stored and updated continuously, then the assurance of convergence is lost. In spite of certain optimism during the early stages of this research, we can only conclude that for large-scale linear programs, the prospects for developing an efficient ellipsoid algorithm are indeed quite bleak.

Acknowledgement

We wish to thank Professor Jean-Louis Goffin for bringing the paper by Shor and Gershovich (1979) to our attention.

REFERENCES

Aspvall, B. and Stone, R. E. (1980). Khachiyan's linear programming algorithm, *Journal of Algorithms* **1**, 1–13.

Dantzig, G. B. (1963). *Linear Programming and Extensions*, Princeton University Press, New Jersey.

Gács, P. and Lovász (1979). Khachiyan's algorithm for linear programming, Report STAN-CS-79-750, Computer Science Department, Stanford University, California.

Gill, P. E., Murray, W. and Saunders, M. A. (1975). Methods for computing and modifying the *LDV* factors of a matrix, *Math. Comp.* **29**, 132, 1051–1077.

Goffin, J. L. (1980). Convergence results in a class of variable metric subgradient methods, Working Paper 80-08, Faculty of Management, McGill University, Montreal, Quebec, Canada. To appear in the Proceedings of the Nonlinear Programming Symposium 4, held in Madison, Wisconsin, July 14–16, 1980.

Khachiyan, L. G. (1979). A polynomial algorithm in linear programming, *Doklady Akademiia Nauk SSSR Novaia Seriia* **244**, 5, 1093–1096. [English translation in *Soviet Mathematics Doklady* **20**, 1 (1979), 191–194.]

Lawler, E. L. (1980). The great mathematical sputnik of 1979, University of California, Berkeley, California (February 1980).

Shor, N. Z. (1977). The cut-off method with space dilation for solving convex programming problems, *Kibernetika* **13**, 1, 94–95. [English translation in *Cybernetics* **13** (1978), 94–96.]

Shor, N. Z. and Gershovich, V. I. (1979). A family of algorithms for solving convex programming problems, *Kibernetika* **15**, 4, 62–67. [English translation in *Cybernetics* **15** (1980), 502–508.]

Wilkinson, J. H. (1965). *The Algebraic Eigenvalue Problem*, The Clarendon Press, Oxford.

Wolfe, P. (1980a). A bibliography for the ellipsoid algorithm, IBM Research Center Report No. 8237 (April 1980).

Wolfe, P. (1980b). The ellipsoid algorithm, in *Optima*, **1** (Newsletter of the Mathematical Programming Society, June 1980).

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: SOL-80-27 (14)	2. GOVT ACCESSION NO.: AD-A093 627
4. TITLE (and Subtitle): A NUMERICAL INVESTIGATION OF ELLIPSOID ALGORITHMS FOR LARGE-SCALE LINEAR PROGRAMMING,	5. TYPE OF REPORT & PERIOD COVERED: (9) Technical Report,
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Philip E. Gill, Walter Murray, Michael A. Saunders, and Margaret H. Wright (10)	8. CONTRACT OR GRANT NUMBER(s): (15) DAAG29-79-C-0110, N00014-75-C-0267
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Department of Operations Research - SOL, Stanford University, Stanford, CA 94305 (12) 47	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: NR-047-143
11. CONTROLLING OFFICE NAME AND ADDRESS: U.S. Army Research Office, P.O. Box 12211, Research Triangle Park, NC 27709 (11)	12. REPORT DATE: October 1980
	13. NUMBER OF PAGES: 22
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Office of Naval Research - Dept. of the Navy, 800 N. Quincy Street, Arlington, VA 22217	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

This document has been approved for public release and sale; its distribution is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

ELLIPSOID ALGORITHM

LINEAR PROGRAMMING

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

SEE ATTACHED

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

408765 slt

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

SOL 80-27: A Numerical Investigation of Ellipsoid Algorithms for Large-Scale Linear Programming

The ellipsoid algorithm associated with Shor, Khachiyan and others has certain theoretical properties that suggest its use as a linear programming algorithm. Some of the practical difficulties are investigated here. A variant of the ellipsoid update is first developed, to take advantage of the range constraints that often occur in linear programs (i.e., constraints of the form $l \leq a^T x \leq u$, where $u - l$ is reasonably small). Methods for storing the ellipsoid matrix are then discussed for both dense and sparse problems. In the large-scale case, a major difficulty is that the desired ellipsoid cannot be represented compactly throughout an arbitrary number of iterations. Some schemes are suggested for economizing on storage, but any guarantee of convergence is effectively lost. At this stage there remains little room for optimism that an ellipsoid-based algorithm could compete with the simplex method on problems with a large number of variables.