Division of Research
Graduate School of Business Administration
The University of Michigan

April 1980

NEW METHOD FOR LINEAR PROGRAMMING IS FASTER THAN SIMPLEX FOR A CERTAIN CLASS OF PROBLEMS

Working Paper No. 209

Zvi Drezner

The University of Michigan
Dearborn

FOR DISCUSSION PURPOSES ONLY

None of this material is to be quoted or
reproduced without the express permission
of the Division of Research.

New Method for Linear Programming Is Faster than Simplex for a Certain Class of Problems

Zvi Drezner
School of Management
The University of Michigan-Dearborn

Abstract

We introduce an algorithm which is a combination of Khachian's method and the relaxation method for linear programming. We tested the algorithm on problems of finding a feasible point subject to linear constraints, when we have finite lower and upper bounds for each variable. On a randomly generated set of test problems, our method ran faster than the simplex method in C.P.U. time. The run times for the new method ranged between 4.6 percent and 26 percent of the run times for the simplex method for reasonably sized problems.

Introduction

In this paper we present the relaxation method for solving a set of linear inequalities [1,5] and relate this method to Khachian's method for linear programming [4]. We show how to perform every iteration efficiently and present two improvements in the basic relaxation method: compound constraints and the nestled ball principle. Computational results are presented.

The Basic Relaxation Approach

Consider the problem of finding a feasible solution to:

$$\sum_{j=1}^{n} a_{ij}x_j \leq b_i \quad \text{for } i=1,\dots,m. \tag{1}$$

Let:

$$\theta_i(x) = \sum_{j=1}^{n} a_{ij}x_j - b_i \quad \text{for } i=1,\dots,m. \tag{2}$$

The basic relaxation method is:

Step 1: Choose an initial solution $x^{(0)}$; set k=0. Normalize each constraint by dividing a_{ij} and b_i by $(\sum_{j=1}^{n} a_{ij}^2)^{\frac{1}{2}}$.

Step 2: Find the constraint which is most violated, i.e., find

$$\theta^{(k)} = \max_i (\theta_i(x^{(k)}))$$

and let r be the constraint for which $\theta^{(k)}$ is obtained.

Step 3: If $\theta^{(k)} \leqslant 0$, stop. $x^{(k)}$ is feasible.

Step 4: Otherwise, update $x^{(k)}$:

$$x_j^{(k+1)} = x_j^{(k)} - \theta^{(k)} a_{rj} \quad \text{for } j=1,\ldots,n.$$

Note that $\theta_r(x^{(k+1)}) = 0$. Go to step 2.

It is not clear whether the relaxation method will lead to a feasible solution if there is one, or how to recognize a nonfeasible situation. The next two theorems are concerned with these questions.

Theorem 1:

If a_{ij} and b_i are integers, then the relaxation method determines in a finite number of iterations if a feasible solution exists.

Proof:

The proof is based on the following observation: According to Khachian [4], there exists a radius $R^{(0)}$ such that if a feasible solution exists, it must exist in the ball centered at $x^{(0)}$ with radius $R^{(0)}$. Let us assume that we have proven that in the k'th iteration a feasible solution must exist inside a ball centered at $x^{(k)}$ with radius $R^{(k)}$. The hyperplane passing through $x^{(k+1)}$ cuts off more than one half of the ball. In fact, it is distant by $\theta^{(k)}$ from the center of the k'th ball, which is $x^{(k)}$. Therefore, the k+1'th ball, which is centered at $x^{(k+1)}$, has a radius of $(R^{(k)^2} - \theta^{(k)^2})^{\frac{1}{2}}$. As shown by Khachian [4],

$\theta^{(k)} > 2^{-L}$ (where L is defined there) if there is no feasible solution and if $\theta^{(k)} \leqslant 2^{-L}$ a feasible solution exists. Therefore, if $\theta^{(k)} \leqslant 2^{-L}$, we know that there is a feasible solution and if $\theta^{(k)} > 2^{-L}$ for every k, then in a finite number of steps we get $R^{(k)^2} < 0$, and there is no feasible solution.

The following theorem is trivial by the proof of theorem 1:

<u>Theorem 2</u>:

If for some N $\sum_{k=0}^{N} \theta^{(k)^2} > R^{(0)^2}$, then there is no feasible solution to problem (1) .

The relaxation method is probably not polynomial since the number of iterations is not polynomial. The main difficulty in applying Khachian's method [4] is the great accuracy needed in order to assure the finding of a feasible solution if one exists. In [2] a practical method for a special class of problems has been presented. That is; there are given bounds on the variables $\ell_j \leqslant x_j \leqslant \mu_j$. We can therefore, replace $R^{(0)}$ by

$$R^{(0)^2} = \sum_{j=1}^{n} (\mu_j - \ell_j)^2/4$$

with center at

$$x_j^{(0)} = (\mu_j + \ell_j)/2 \quad \text{for } j=1,\ldots,n.$$

In addition, we assume a given accuracy of $\varepsilon > 0$.

It is preferable to perform a linear transformation yielding

$$x_j' = (x_j - \ell_j)/(\mu_j - \ell_j) ,$$

$$0 \leqslant x_j' \leqslant 1 , \; R^{(0)} = (n/4)^{\frac{1}{2}} , \; x_j'^{(0)} = 0.5.$$

<u>Going Beyond the Constraints</u>

One can multiply $\theta^{(k)}$ by $1+\alpha$ for a given $-1<\alpha<1$.
There is no sense in using $\alpha<0$. $\alpha=0$ yields the greatest decrease in R^2 since

$$R^{(k+1)^2} = R^{(k)^2} - (1-\alpha^2)\theta^{(k)^2} .$$

We have checked the possibility of using $\alpha>0$ and have found that for our test problems $\alpha=0.8$ gave the best results.

A More Efficient Computational Procedure

Every iteration in the basic relaxation method is of complexity $0(mn)$. We can perform the computation of each interation in $0(m)+0(n)$ if a set=up procedure of complexity $0(m^2n)$ and a space for a matrix of size $m(m-1)/2$ are permitted.

Let:

$$\lambda_{ij} = a_i^T a_j \quad \text{for } i,j=1,\ldots,m .$$

Note that $\lambda_{ji}=\lambda_{ij}$ and $\lambda_{ii}=1$. Let $\theta_i^{(k)} = \theta_i(x^{(k)})$.

We keep the values of $\theta_i^{(k)}$ and replace the updating formula in step 4 of the algorithm to:

$$x_j^{(k+1)} = x_j^{(k)} - \theta^{(k)} a_{rj} \quad \text{for } j=1,\ldots,n \tag{4a}$$

$$\theta_i^{(k+1)} = \theta_i^{(k)} - \theta^{(k)} \lambda_{ir} \quad \text{for } i=1,\ldots,m \tag{4b}$$

$$R^{(k+1)^2} = R^{(k)^2} - \theta^{(k)^2} \tag{4c}$$

Updating the vector x is of complexity $0(n)$, updating the vector $\theta^{(k)}$ is of complexity $0(m)$, and finding the maximum violated constraint is also of complexity $0(m)$ (it can be even calculated in the same loop in which $\theta^{(k)}$ is updated). Therefore, every interation is simple and fast and is of complexity $0(m)+0(n)$. Calculating all λ_{ij} is of complexity $0(m^2n)$.

The Compound Constraint

We can replace two or more constraints by a convex combination of constraints, yielding a "better" cut. This idea was presented by Goldfarb and Todd [3] as a surrogate cut. Since an important feature of the relaxation method is the low complexity of each iteration, we would like to retain this feature, and therefore we restrict ourselves to checking only some pairs of constraints, as will be explained later.

Let us consider two constraints:

$$a_1^T x \leq b_1$$

$$a_2^T x \leq b_2$$

with $\theta_1(x)=a_1^T x-b_1$ and $\theta_2(x)=a_2^T x-b_2$. For $0 \leq \mu \leq 1$ it must be true that:

$$(\mu a_1^T + (1-\mu)a_2^T)x \leq \mu b_1 + (1-\mu)b_2 \ . \qquad (5)$$

We would like to maximize the residual θ of the compound constraint (5) . Since the constraint (5) must be first normalized, we find that:

$$\theta(\mu) = (\mu\theta_1 + (1-\mu)\theta_2)/\|\mu a_1 + (1-\mu)a_2\| \ .$$

Since $a_1^T a_1=a_2^T a_2=1$ and $a_1^T a_2=a_2^T a_1=\lambda_{12}$, then:

$$\theta(\mu) = (\mu\theta_1 + (1-\mu)\theta_2)/[\mu^2 + (1-\mu)^2 + 2\lambda_{12}\mu(1-\mu)]^{\frac{1}{2}} \ . \qquad (6)$$

Solving $d\theta(\mu)/d\mu = 0$ leads to the constraint:

$$(\alpha a_1^T + \beta a_2^T)x \leq \alpha b_1 + \beta b_2 \qquad (7)$$

where:

$$\alpha = (\theta_1-\lambda_{12}\theta_2)/(1-\lambda_{12}^2)$$

$$\beta = (\theta_2-\lambda_{12}\theta_1)/(1-\lambda_{12}^2) \ .$$

This combination is valid only if $\alpha \geq 0$ and $\beta \geq 0$.

The updating formulas (4) are now:

$$x_j^{(k+1)} = x_j^{(k)} - \alpha a_{1j} - \beta a_{2j} \quad \text{for } j=1,\ldots,n \tag{8a}$$

$$\theta_i^{(k+1)} = \theta_i^{(k)} - \alpha\lambda_{1i} - \beta\lambda_{2i} \quad \text{for } i=1,\ldots,m \tag{8b}$$

$$R^{(k+1)^2} = R^{(k)^2} - (\theta_1^2 + \theta_2^2 - 2\lambda_{12}\theta_1\theta_2)/(1-\lambda_{12}^2). \tag{8c}$$

It can easily be verified that $\theta_1^{(k+1)} = \theta_2^{(k+1)} = 0$.

We apply the compound constraint formula by the following strategy. Let θ_1 be a maximum violated constraint. We check combinations of another constraint combined with the maximum violated one. Since $\theta_1 = \max_i \left\{\theta_i^{(k)}\right\}$ we have $\alpha \geq 0$ for every i. Therefore, we check only if $\theta_i - \lambda_{i1}\theta_1 \geq 0$.

Note that:

$$(\theta_1^2 + \theta_2^2 - 2\lambda_{12}\theta_1\theta_2)/(1-\lambda_{12}^2) = \theta_1^2 + (\theta_2 - \lambda_{12}\theta_1)^2/(1-\lambda_{12}^2), \tag{9}$$

so we look for:

$$\overline{\theta} = \max_{\theta_i - \lambda_{1i}\theta_1 \geq 0} \left\{ (\theta_i - \lambda_{1i}\theta_1)^2/(1-\lambda_{1i}^2) \right\} . \tag{10}$$

If $\overline{\theta}$ exists we choose constraint number 2 as the constraint that maximizes $\overline{\theta}$. Calculations performed by this scheme retain the complexity 0(m)+0(n) for each iteration, but the computational effort is increased.

The Principle of the Nestled Ball

In this section we will prove that if after some interations the ball is inside the initial ball, then there is no feasible solution.

Let $R^{(0)} = R$, $R^{(k)} = r$, and $(x^{(k)} - x^{(0)})^T(x^{(k)} - x^{(0)}) = d^2$.

Theorem 3:

If $R > r+d$ then there is no feasible solution to the problem (1) .

Proof:

If a feasible solution exists, it must be in the ball centered at $x^{(k)}$ with radius r. Therefore, there must be a feasible solution in the ball centered at $x^{(0)}$ with radius r+d. Since R>r+d, we could have started the iterations with a ball with radius of r+d as $R^{(0)}$. Since the value of $R^{(0)}$ does not affect the values of θ_i, x, etc., we would have passed through exactly the same points and would have reached $x^{(k)}$ in k iterations. Since R^2-r^2 remains unchanged, we would have ended with a smaller r (let it be r', where $r'^2 = (r+d)^2 - (R^2-r^2)$) .
This smaller r yields yet a smaller $R^{(0)}$, namely $R^{(0)} = r'+d$, and so on. We get a sequence of r's; let the sequence be r_0, r_1,... where:

$$r_0 = r \qquad (11)$$

$$r_{k+1}^2 = (r_k+d)^2 - (R^2-r^2) .$$

There are two possibilities. either $r_k^2<0$ for some k, or $r_k^2 \geq 0$ for every k. If $r_k^2<0$ for some k, then there is no feasible solution, since we have proven that a feasible solution must lie in a nonexisting ball. If $r_k^2 \geq 0$ for every k, then a limit to the sequence r_k exists, since r_k is monotonically decreasing and bounded by zero. Let this limit be z. For the limit point we must have by equation (11):

$$z^2 = (z+d)^2 - (R^2-r^2)$$

$$z = (R^2-r^2-d^2)/2d .$$

Since R>r+d:

$$z > ((r+d)^2-r^2-d^2)/2d = r .$$

But z>r is impossible, since $r_0=r$ and the sequence is monotonically decreasing. The theorem is proven.

Theorem 3 provides us with a better stopping criterion in case of infeasible solution. The geometric interpretation of Theorem 3 is quite interesting. There is a growing ball centered at $x^{(0)}$ such that if $x^{(k)}$ enters this ball there is no feasible solution. The radius of this ball is R-r, and $R-r = (R^2-r^2)/(R+r) = \Sigma\theta^{(k)^2}/(R+r)$.

It can be shown that there exists a ball centered at $x^{(0)}$ inside which there is no feasible solution. The condition of Theorem 3 holds if the radius of that ball is greater than $R^{(0)}$. We have not yet found a "good use" for this result, even though it gives us more information about the set of possible locations of feasible points. More research is yet to be done.

Further Suggestions

Equality Constraints

If there are equality constraints in the problem, we can, of course, change each of them to a pair of inequalities. However, I believe that the following is a better approach. Suppose we have an equality constraint:

$$\sum_{j=1}^{n} a_{1j}x_j = b_1 \quad .$$

There must exist some $a_{1j} \neq 0$; or else, if $b_1=0$, the constraint can be ignored, and if $b_1 \neq 0$, then the system is inconsistent. Choose any $a_{1j} \neq 0$ (or choose the maximal in absolute value); let it be a_{11} . We have:

$$x_1 = (b_1 - \sum_{j=2}^{n} a_{1j}x_j)/a_{11} \quad .$$

We can substitute x_1 in all other inequalities (and equalities), thus reducing the number of variables by one. We must add two constraints for the bounds of x_1 , namely,

$$\ell_1 \leq (b_1 - \sum_{j=2}^{n} a_{1j}x_j)/a_{11} \leq \mu_1 \ .$$

In substituting x_1 for all equalities, we replace each equality in turn by two inequalities but reduce the number of variables.

We can also employ the following strategy. Let every constraint be defined as:

$$b_i - e_i \leqslant \sum_{j=1}^{n} a_{ij} x_j \leqslant b_i \qquad (12)$$

where e_i is a big number if the left constraint is not applicable. Every two inequalities of type (12) are, for practical purposes, one constraint only. The solution procedure is almost unchanged. We still have to calculate only $\theta_i^{(k)}$, but we must check for $\max\left\{\theta_i^{(k)}, -e_i - \theta_i^{(k)}\right\}$ instead of $\max\left\{\theta_i^{(k)}\right\}$, which requires almost no additional computational effort. We believe that this approach is superior to that of handling an equality as a pair of inequalities, since the presence of an equality gives a feasible region of zero volume but transforming into a lower dimension space probably yields a nonzero volume for the feasible region.

Large Problems

The basic relaxation method is well suited for working with direct-access secondary storage. We keep in core the vectors $x_i^{(0)}$, $x_i^{(k)}$, $\theta_i^{(k)}$, μ_i, ℓ_i, and e_i. If we use the transformation (3), we no longer need $x^{(0)}$, μ_i, and ℓ_i in the memory core, so the in-core storage is only of size 2m+n. a_{ij} is stored on a disk by rows, and so is λ_{ij}. For every iteration we need only one row of a_{ij} and one of λ_{ij}. Therefore, only two vectors, one of length n and the other of length m, must be read into the core for every iteration.

Sparse Matrices

If every row of a_{ij} is given by a list of all nonzero elements as (j, a_{ij}), then updating $x^{(k)}$ is trivial and very fast, and calculating λ_{ij} is also very convenient. It might be even more economical to calculate a row of λ_{ij} whenever it is needed rather than calculating all λ's and storing them. Every iteration will require one pass of the entire matrix a_{ij} .

If all nonzero a_{ij} are ones, a bit presentation may be very efficient. We calculate a vector holding the number of ones in each row vector of a_{ij} instead of normalization. Calculating λ_{ij} simply involves taking a "logical and" operation between words and counting the number of ones in the resulting words. If the number of ones in a word is small, we can efficiently count the number of ones by the following observation. If we arithmetically subtract "one" in the rightmost position of the word from a nonzero number and perform a "logical and" between the result and the word, then the rightmost one is wiped out while the rest of the bits in the word remain the same. By this we do the following:

Step 1: Set count to 0.

Step 2: If word = 0, go to step 7.

Step 3: Set count=count+1.

Step 4: Calculate word'=word-1.

Step 5: Perform a "logical and" between word and word' and put result in word.

Step 6: Go to Step 2.

Step 7: Exit with count as the number of "ones" in word.

Computational Comparison

We have generated two types of problems, feasible problems and infeasible problems. The coefficients a_{ij} for the feasible problems were generated uniformly on the segment (-1,1). We set $b_i = \sum_{j=1}^{n} a_{ij}/4$.

so that x=0.25 is a feasible point. The feasible simplex is not necessarily big. In fact, if m>>n , the point x=0.25 is practically the only feasible point. In addition, we assumed $0 \leq x_i \leq 1$.

The constraints for the infeasible problems were identically generated, but the last constraint was replaced by:

$$a_{mj} = -\sum_{i=1}^{m-1} a_{ij} \quad \text{for } j=1,\ldots,n$$

$$b_m = -\left(\sum_{i=1}^{m-1} b_i + 0.1n((m-1)/3)^{\frac{1}{2}}\right) .$$

The expected value of the term added to b_m after normalization of the constraint is equal to 0.1 . We use single precision variables on AMDAHL 470/v7 at the University of Michigan, Ann Arbor, Michigan. We use $\varepsilon=10^{-4}$ (i.e., $\theta^{(k)} \leq \varepsilon$ means feasible solution). All run times exclude input and are expressed in terms of seconds of C.P.U.

In Tables 1 and 2, the basic approach is compared with the basic approach with compound constraints. We have used $\alpha=0.8$. The number of iterations decreases when the compound constraints are used, but run times on feasible problems remain almost the same. We have decided not to include the compound constraints in the new method for further comparisons.

Table 1: Run Times for Compound Constraints: Feasible Problems

m	n	Single Constraints		Compound Constraints	
		Iterations	Time (sec)	Iterations	Time (sec)
10	10	5	0.004	4	0.004
10	20	5	0.004	3	0.004
10	50	6	0.006	4	0.006
50	10	114	0.032	71	0.040
50	50	47	0.080	34	0.090
50	100	36	0.148	20	0.160
50	200	30	0.290	16	0.314
100	10	86	0.083	55	0.096
100	50	299	0.370	905	0.848
100	100	100	0.582	70	0.619
100	200	60	1.115	31	1.179
200	10	75	0.274	48	0.313
200	20	138	0.497	79	0.565
200	100	10000*	9.061	10000*	16.095
200	200	167	4.415	117	4.693

* Run terminated due to iteration limit of 10000.

Table 2: Run Times for Compound Constraints: Infeasible Problems

m	n	Single Constraints		Compound Constraints	
		Iterations	Time (sec)	Iterations	Time (sec)
5	10	11	0.005	4	0.005
10	10	7	0.005	2	0.004
10	100	25	0.018	9	0.015
20	20	2	0.009	2	0.010
20	50	7	0.016	6	0.018
20	100	27	0.035	15	0.036
50	50	37	0.079	15	0.084
50	100	74	0.166	33	0.173
100	100	88	0.585	40	0.613

In Tables 3 and 4 compare three methods for linear programming: the simplex method, Khachian's method using deep cuts [2], and our new relaxation method. Since there are so many codes for the simplex method, we must define our method exactly. We first tried the MPS code. Run times were surprisingly high (excluding input-output). This is probably because MPS works with lists of coefficients, a method which is not suitable to our dense matrix problems. Double precision is probably used with other sophisticated techniques which are time=consuming; however, in order to give the simplex a "fair chance," I have coded the "good old" simplex method in single precision with $\varepsilon=10^{-4}$ (i.e., if the coefficients of the objective function are less than ε, I assumed optimality). The run time of this unsophisticated simplex was only a fraction of the run time on MPS for all the problems that were tested (not all problems in Tables 3 and 4 were tested on MPS). For example, the feasible problem of 50 by 50 was run by MPS in 7.0 seconds of C.P.U.. the initialization phase (input, adding slacks and artificials, etc.) took 3.6 seconds which leaves 3.4 C.P.U. seconds for the 87 iterations needed. In my simplex program the same problem was solved in 37 iterations and only 0.343 seconds which is 10 percent of the time of MPS!

The results presented in Tables 3 and 4 speak for themselves and need no additional commentary.

Table 3: Feasible Problems

m	n	Simplex		Khachian		Relaxation	
		Iter.	Time	Iter.	Time	Iter.	Time
5	5	5	0.006	8	0.003	6	0.002
10	10	7	0.010	4	0.003	5	0.004
10	20	7	0.014	6	0.009	5	0.004
20	10	21	0.035	61	0.022	21	0.006
20	20	19	0.046	17	0.021	14	0.008
20	30	15	0.049	10	0.026	8	0.010
20	100	9	0.141	36	0.830	14	0.030
30	50	25	0.171	43	0.261	25	0.033
30	80	22	0.280	32	0.496	21	0.047
40	20	38	0.144	109	0.129	28	0.025
40	60	32	0.314	58	0.518	37	0.065
40	80	28	0.411	50	0.788	27	0.082
50	50	37	0.343	119	0.765	47	0.080
50	100	48	1.059	106	2.511	36	0.148

Table 4: Infeasible Problems

m	n	Simplex		Khachian		Relaxation	
		Iter.	Time	Iter.	Time	Iter.	Time
5	10	4	0.006	2	0.003	11	0.005
10	10	5	0.007	5	0.004	7	0.005
10	100	10	0.144	12	0.254	25	0.018
20	20	14	0.035	18	0.021	2	0.009
20	50	16	0.095	27	0.153	7	0.016
20	100	21	0.328	33	0.723	27	0.035
50	50	66	0.610	116	0.728	37	0.079
50	100	160	3.591	107	2.454	74	0.166
100	100	216	7.451	385	9.530	88	0.585

References

1. Agmon S. "The Relaxation Method for Linear Inequalities." Canadian Journal of Mathematics 6 (1954), pp. 382-392.

2. Drezner Zvi "Improved Formulas for the Russian Method for Linear Programming." Submitted for publication.

3. Goldfarb, D. and Todd, M. "Combining Inequalities and Improving Stability in the Russian Method for Linear Programming" Polynomial Time Algorithm Workshop, New York, February 1980.

4. Khachian L. G. "A Polynomial Algorithm in Linear Programming" (English Translation). Soviet Mathematics Doklady 20 (1979), pp. 191-194.

5. Motzkin, T., and Schoenberg, I. J. "The Relaxation Method for Linear Inequalities." Canadian Journal of Mathematics 6 (1954) pp. 393-404.