Yugoslav Journal of Operations Research
4 (1994), Number 2, 137-148

A NEW EXTERIOR POINT ALGORITHM FOR LINEAR PROGRAMMING PROBLEMS

Konstantinos DOSIOS, Konstantinos PAPARRIZOS

Department of Applied Informatics, University of Macedonia
156 Egnatia Str, POB 1591, 54006 Thessaloniki, Greece

Abstract: We present a simplex type algorithm for the linear programming problem $\min \{ c^T x : A x = b, x \geq 0 \}$. It solves a sequence of problems of increasing dimension. The path generated by the algorithm contains basic solutions that are neither primal nor dual feasible. Results from an elementary computational study are also presented.

Keywords: Simplex method, linear programming, pivoting, exterior point algorithms.

1. INTRODUCTION

The simplex algorithm for linear programming problems (Dantzig, 1963), performs sufficiently well in practice, particularly on problems of small or medium size. Although recently developed interior point algorithm (Karmarkar, 1984), performs substantially better on large scale linear problems, see for example (Adler et.al., 1989), the computational improvement of the simplex algorithm remains of great importance, due mainly to the fact that exact optimal solutions are required for a large number of practical problems, as for example network flow problems.

Simplex algorithms move from a vertex of the polyhedral set of the feasible region to an adjacent one with a better objective value. One way to improve the simplex method is the development of algorithms capable of moving to non adjacent improving vertices. As long as we prefer simplex type algorithms the moving to non adjacent improving vertices can be accomplished by avoiding the boundary of the feasible polyhedral set and constructing basic solutions (BS) that are not feasible. A simplex type algorithm which is capable of following a simplex type path containing basic solutions that are not feasible is called an *exterior point simplex algorithm* (EPSA). Since the appearance of the simplex method researchers tried to develop algorithms that avoid the feasible region. Early examples of efforts of this kind are Dantzig's self dual algorithm (Dantzig, 1963) and the successive shortest path method for minimum cost

---

Paper presented at the 2nd Balkan Conference on OR, Thessaloniki, Greece, 18–21 October, 1993

network flow problems, see (Ford and Fulkerson, 1962) and (Tomizawa, 1972). More recently, Terlaky developed a convergent pivoting rule (Terlaky, 1985) for Zionts Criss–Cross method (Zionts, 1969).

Terlaky' s algorithm is a non improving exterior point algorithm in the sense that it can be initialized with a basic solution that is neither primal nor dual feasible. The completely combinatorial nature and the lack of any improvement criteria make someone to believe that this algorithm is not efficient in practice. A more promising approach forwards developing efficient EPSAs, seems to be algorithm for assignment problems developed by Paparrizos (Paparrizos, 1989). Computational results verified that a variant of this algorithm is one of the most efficient algorithms for assignment problems (Ahatz et.al., 1989).

In some sense Paparrizos (Paparrizos, 1993) has been able to generalize his algorithm for assignment problems to the general linear programming problem. Contrary to Terlaky' s Criss– Cross algorithm, the algorithm by Paparrizos is improving. It is initialized with a basic feasible solution (BFS) to the dual problem. An artificial variable is then introduced and a pivot operation is performed transforming the initial problem into an equivalent one. All the variables of the original problem are feasible in the transformed problem except the artificial variable which is strictly negative. The objective function improves and the artificial variable increases (strictly under non degeneracy assumptions) from iteration to iteration. The algorithm terminates either when an optimal solution is reached or when it is detected that the dual problem is unbounded (and, hence, the primal is not feasible).

The algorithm presented in this paper is a generalization of Paparrizos' EPSA. It is also initialized with a dual feasible basic solution. However, no artificial variable is introduced. A subproblem is then defined containing all the dual variables, all the primal variables that are feasible (to the primal problem) and a unique primal variable which is not feasible. We call this variable distinguished. While trying to solve the subproblem no primal variable is allowed to become infeasible. Every primal variable, which becomes feasible at some stage of the algorithms is introduced to the subproblem being solved, thus increasing its dimension. The subproblem is solved when the distinguished primal variable, becomes zero valued.

The remaining of the paper is organized as follows. In the next section the algorithm is formally introduced and illustrated with an example. Section 3 includes the proofs showing correctness and finiteness of the algorithm. Results from an elementary computational study indicating superiority of the EPSAs over Dantzig' s maximum coefficient pivoting rule (Dantzig, 1963) are presented in the last section.

2. THE ALGORITHM

We consider the linear problem in standard form

$$\begin{aligned} \min \quad & c^{T} \cdot x, \\ & A \cdot x = b, \\ & x \geq 0, \end{aligned} \qquad \text{(LP)}$$

where $c \in \mathcal{R}^n$, $A \in \mathcal{R}^{m \times n}$, $b \in \mathcal{R}^m$ and T denotes transposition. It is assumed that an initial basic feasible solution to the dual problem

$$\begin{aligned} \max \quad & b^{\mathrm{T}} \cdot \pi, \\ & A^{\mathrm{T}} \cdot \pi + s = c, \\ & s \geq 0, \end{aligned} \qquad \text{(DP)}$$

where $\pi \in \mathcal{R}^m$ and $s \in \mathcal{R}^n$, is available. Under these assumptions the primal problem is either optimal or infeasible. In order to simplify the proof of correctness we also assume that the primal problem is non degenerate.

Observe that the non degeneracy assumptions are imposed on the primal problem although the algorithm is initialized with a dual feasible basic solution.

We find convenient to describe the algorithm in tableau format. We use an $(m+1) \times n$ tableau the entries of which are denoted by a_{ij}, $i = 0, 1, \ldots, m$, $j = 1, \ldots, n$. Row zero of the tableau is preserved for the objective function which is initially written as $z - c^{\mathrm{T}} x = 0$. As dual feasibility of the initial point is assumed, the zero row entries of the initial tableau satisfy the conditions $a_{0j} = -c_j \leq 0$, $j = 1, 2, \ldots, n$. The RHS is denoted by b. As a result, b_i, $i = 1, 2, \ldots, m$, is the value of the basic variable, x_{B_i}, associated with the i-th row. The objective function value is denoted by b_0. With this notation at hand the algorithm is formally stated as follows.

Step 0. (*Initialization*). Start with a dual feasible tableau. If $b_i \geq 0$, $i = 1, 2, \ldots, m$ STOP. Otherwise choose an arbitrary index k such that $b_k < 0$.

Step 1. (*Test of optimality*). If $b_k < 0$, go to step 2. Otherwise, i.e., if $b_k = 0$, check whether the conditions

$$b_i \geq 0, \quad 1 \leq i \neq k \leq m,$$

are satisfied. If they are, STOP (the current solution is optimal). If they are not, choose a new index k such that $b_k < 0$.

Step 2. (*Pivot column determination*). Set $J = \{ j : a_{kj} < 0 \}$. If $J = \varnothing$, STOP (the dual problem (DP) is unbounded). Otherwise, set

$$\frac{a_{0l}}{a_{kl}} = \min \{ \frac{a_{0j}}{a_{kj}} : j \in J \}$$

Step 3. (*Pivot row determination*). Set

$$b_r = \min \{ \frac{b_k}{a_{kl}}, \min \{ \frac{b_i}{a_{ij}} : a_{ij} > 0, b_i \geq 0 \} \}$$

Step 4. (*Pivot operation*). Perform a pivot operation on entry a_{rl}, compute the new tableau and go to step 1.

The test of optimality used by our algorithm is precisely that of the dual simplex algorithm. However, there are striking differences between our algorithm and the dual method. The order by which the pivot row and column is determined is reversed. First

the pivot column is determined and then the pivot row. The distinguished index k is only used to determine the pivot column. Observe that dual feasibility may be destroyed in the very first iteration as long as row k is not the pivot row. We will see later that dual feasibility is restored every time the pivot operation is performed on row k.

We illustrate the algorithm on the linear programming problem

$$\begin{aligned} \min \quad & x_1 + 2x_2 + 3x_3 \\ \text{s.t.} \quad & x_1 - x_2 + x_3 \le 2 \\ & -x_1 - 2x_3 \le 4 \\ & -x_1 + x_2 \le -1 \\ & x_1, \quad x_2, \quad x_3 \ge 0 \end{aligned}$$

Iteration 1

Step 0. Introduction of the slack variables x_4, x_5 and x_6 results in the initial tableau.

	x_1	x_2	x_3	x_4	x_5	x_6	RHS	
z	-1	-2	-3	0	0	0	0	$k=2$
x_4	(1)	-1	1	1	0	0	2	
x_5	-1	0	-2	0	1	0	-4	
x_6	-1	1	0	0	0	1	-1	

As it is not $b_i \ge 0$, $i = 1, 2, 3$, we choose (arbitrarily) $k = 2$. The subproblem we are trying to solve is associated with the distinguished primal variable $x_{B_2} = x_5$. It contains all the primal variables except x_6.

Step 1. As $b_k = b_2 = -4 < 0$ we go to step 2.

Step 2. $J = \{1, 3\} \neq \emptyset$. As

$$\min \{ \frac{a_{0i}}{a_{2i}} : i = 1, 3 \} = \min \{ \frac{-1}{-1}, \frac{-3}{-2} \} = 1 = \frac{a_{01}}{a_{21}}$$

we have $l = 1$.

Step 3. As

$$\min \{ \frac{b_2}{a_{21}}, \min \{ \frac{b_i}{a_{i1}} : b_i \ge 0, a_{i1} > 0 \} \} = \min \{ \frac{-4}{-1}, \frac{2}{1} \} = 2 = \frac{b_1}{a_{11}}$$

we have $r = 1$.

Step 4. Pivoting on element $a_{11} = 1$, which is included into parenthesis in the tableau, results in the following tableau

	x_1	x_2	x_3	x_4	x_5	x_6	RHS
z	0	− 3	− 2	1	0	0	2
x_1	1	− 1	1	1	0	0	2
x_5	0	− 1	− 1	1	1	0	− 2
x_6	0	0	(1)	1	0	1	1

Observe that the current tableau is not dual feasible. Observe also that the objective value has been improved (increased) and the value of the distinguished basic variable $x_{B_k} = x_5$ has been increased from − 4 to − 2. As the primal basic variable x_6 is now feasible (≥ 0), it will be introduced into the current subproblem.

Iteration 2

Step 1. As $b_2 = -2 < 0$(the current subproblem is not solved), we go to step 2.

Step 2. $J = \{2, 3\} \neq \varnothing$. As $k = 2$, we have

$$\min\{\frac{a_{02}}{a_{22}}, \frac{a_{03}}{a_{23}}\} = \min\{\frac{-3}{-1}, \frac{-2}{-1}\} = 2 = \frac{a_{03}}{a_{23}}$$

and, hence, $l = 3$.

Step 3. As

$$\min\{\frac{b_2}{a_{23}}, \min\{\frac{b_1}{a_{13}}, \frac{b_3}{a_{33}}\}\} = \min\{\frac{-2}{-1}, \min\{\frac{2}{1}, \frac{1}{1}\}\} = 1 = \frac{b_3}{a_{33}}$$

we have $r = 3$.

Step 4. Pivoting on element a_{33} results in the tableau

	x_1	x_2	x_3	x_4	x_5	x_6	RHS
z	0	− 3	0	3	0	2	4
x_1	1	− 1	0	0	0	− 1	1
x_5	0	(− 1)	0	2	1	1	− 1
x_6	0	0	1	1	0	1	1

Iteration 3

Step 1. As $b_k = b_2 = -1 < 0$ we go to step 2.

Step 2. As $J = \{2\} \neq \varnothing$, column 2 ($l = 2$) is the pivot column.

Step 3. Row 2 is the unique eligible pivoting row.

Step 4. Pivoting on a_{22} results in the following tableau

	x_1	x_2	x_3	x_4	x_5	x_6	RHS
z	0	0	0	−3	−3	−1	7
x_1	1	0	0	−2	−1	−2	2
x_2	0	1	0	−2	−1	−1	1
x_3	0	0	1	1	0	1	1

Iteration 4

Step 1. As the current tableau is primal feasible, the algorithm stops.

Observe that dual feasibility has been restored in the last tableau and that the algorithm has only solved one subproblem.

3. CORRECTNESS AND FINITENESS

The proof of the correctness will be done by modifying properly the results in (Paparrizos, 1993).

THEOREM 1. If the algorithm stops at step 2, the primal problem (LP) is not feasible.

PROOF. Suppose that the algorithm stops at step 2.

Let Q be the set of indices j such that x_j is strictly negative at the current tableau and B_k be the current distinguished basic variable. Let also $\hat{x}$ be the subvector of x containing all the components of x, the values of which are non negative at the current tableau. Denote by $\bar{A}$ the submatrix of the matrix A associated with the initial problem LP, which is produced by deleting from A all columns j such that $j \in Q$ and all rows i such that $B_i \in Q$. Finally, denote by $\bar{b}$ the subvector of b associated also with the initial problem LP, which is produced by deleting from b the components b_i such that $B_i \in Q$. Consider now the following linear subproblem of LP,

$$\begin{aligned} \max \quad & x_{B_k}, \\ \text{s.t.} \quad & \bar{A} \cdot \bar{x} = \bar{b}, \\ & \bar{x} \geq 0, \end{aligned} \qquad \text{(SP)}$$

From the stopping conditions of step 2 we easily conclude that the current tableau is optimal for subproblem SP. As the optimal solution of SP is strictly negative, the linear system $Ax = b, x \geq 0$ has no feasible solution.

The proof of the correctness will be completed by showing the following.

THEOREM 2. If the algorithm stops at step 1 the current solution is optimal. The proof of Theorem 2 will be done by induction on the number of iterations. To this end we will show first the following lemma.

LEMMA 3. Every tableau generated by the algorithm satisfies the following condition

$$f = \min\{\frac{a_{0j}}{a_{kj}} : a_{kj} < 0\} \geq \max\{\frac{a_{0j}}{a_{kj}} : a_{kj} \geq 0\} = g \qquad (1)$$

In the notation used in relation (1) it is meant that if there exists an index j such that $a_{kj} = 0$ and $a_{0j} > 0$ $(a_{0j} < 0)$, then $g = \infty$ $(g = -\infty)$. Also, if $a_{0j} = a_{kj} = 0$, then $a_{0j}/a_{kj} = 0$. It is easily concluded that $g \geq 0$. Indeed, let x_s be the basic variable associated with row k. Then, $a_{0s} = 0$, $a_{ks} = 1$ and, hence, $g \geq 0$.

Before proving the lemma we will show how it can be used to show Theorem 2. Let $((a_{ij}), b)$ and $((\hat{a}_{ij}), \hat{b})$ be two consecutive tableaux generated by the algorithm. In order to simplify the proof we assume that no ties occur in the minimum ratio tests.

PROOF OF THEOREM 2. Suppose that the algorithm has stopped at step 1 and that the current tableau is $((\hat{a}_{ij}), \hat{b})$. From the rules of the algorithm we can easily see that the pivot row in the previous tableau $((a_{ij}), b)$ was row k.

The current tableau is primal feasible. Hence, it suffices to show that it is dual feasible as well. Assume it is not. Then there exists an index s such that $\hat{a}_{0s} < 0$. It is well known that

$$\hat{a}_{0s} = a_{0s} - \frac{a_{01}}{a_{k1}} \times a_{ks} \qquad (2)$$

where l is the pivot column.

As $a_{kl} < 0$, there is no index j such that $a_{0j} < 0$ and $a_{kj} = 0$. Otherwise, it is $f < \infty = g$ and Lemma 3 does not hold for the previous tableau, a contradiction to the induction hypothesis. Hence, there are two cases to be considered.

Case 1. $a_{ks} < 0$. From (2) and $a_{0s} < 0$ we have

$$\frac{a_{0s}}{a_{ks}} > \frac{a_{0l}}{a_{kl}}$$

Hence, we have $g > f$, a contradiction to Lemma 3.

Case 2. $a_{ks} < 0$. From (2) and $a_{0s} < 0$ we have

$$\frac{a_{0s}}{a_{ks}} < \frac{a_{0l}}{a_{kl}}$$

As both indices k and s belong to the set J, the last inequality shows that the rules of the algorithm for the pivot column choice have been violated, a contradiction.

We prove now Lemma 3. As in the proof of Theorem 2, the entries of two consecutive tableaux are denoted by $((a_{ij}), b)$ and $((\hat{a}_{ij}), \hat{b})$ respectively.

PROOF OF LEMMA 3. We show first that condition (1) is satisfied by every dual feasible tableau. Indeed, if the current tableau is dual feasible, then $a_{0j} \leq 0$, $j = 1, 2, \ldots, n$. Hence, $f \geq 0$ and $g = 0$. Therefore, condition (1) is satisfied by the initial tableau.

Assume now that the current tableau satisfies condition (1). There are two cases to be considered.

Case 1. $r = k$. In this case, the proof of Theorem 2 can be used to show that the next tableau is dual feasible. Hence, condition (1) is satisfied by the next tableau.

Case 2. $r \neq k$. From the rules of the algorithm we can easily see that the pivot element a_{rl} is strictly positive and a_{kl} is strictly negative. Let x_s be the basic variable associated with row k. From $a_{0s} = 0$ and $a_{ks} = 1$ we have $g \geq 0$. As the induction hypothesis (1) is satisfied by the current tableau, we conclude that $f \geq 0$, and, hence, $a_{0l} \leq 0$.

Let $\hat{g} = \hat{a}_{0p}/\hat{a}_{kp}$ and $\hat{f} = \hat{a}_{0q}/\hat{a}_{kq}$. Assume that

$$\hat{g} > \hat{f} \tag{3}$$

We consider two subcases.

Subcase 1. $\hat{a}_{kp} = 0$. From the relations

$$\hat{a}_{kp} = a_{kp} - \frac{a_{rp}}{a_{rl}} \times a_{kl} = 0$$

we have $\dfrac{a_{kp}}{a_{kl}} = \dfrac{a_{rp}}{a_{rl}}$.

Hence, $\hat{a}_{0p} = a_{0p} - \dfrac{a_{rp}}{a_{rl}} \times a_{0l}$

$$= a_{0p} - \frac{a_{kp}}{a_{kl}} \times a_{0l} \tag{4}$$

From relation (4) and the fact that $g \geq 0$ we have $\hat{a}_{0p} < 0$ and, hence,

$$a_{0p} < \frac{a_{kp}}{a_{kl}} \times a_{0l} \tag{5}$$

If $a_{kp} = 0$ then $a_{0p} < 0$ and hence $g = \infty < f$.

If $a_{kp} < 0$ we have from relation (5) that

$$g \geq \frac{a_{0p}}{a_{kp}} = \frac{a_{0l}}{a_{kl}} = f.$$

In both cases the induction hypothesis does not hold, a contradiction.

Finally, if $a_{kp} < 0$ we have from (5) that

$$\frac{a_{0p}}{a_{kp}} < \frac{a_{0l}}{a_{kl}}.$$

The last inequality contradicts the pivot column selection rule.

Subcase 2. $\hat{a}_{kp} < 0$. We show first that $\hat{g} = f$. Let x_s be the basic variable associated with row r. Then

$$\hat{a}_{ks} = -\frac{a_{kl}}{a_{rl}} > 0 \quad \text{and}$$

$$\frac{\hat{a}_{0s}}{\hat{a}_{ks}} = \frac{-\frac{a_{0l}}{a_{rl}}}{-\frac{a_{kl}}{a_{rl}}} = \frac{a_{0l}}{a_{kl}} = f.$$

Consider now the indices j such that $\hat{a}_{kj} < 0$. Suppose that there exists an index j such that

$$\hat{a}_{kj} < 0 \quad \text{and} \quad \hat{a}_{0j}/\hat{a}_{kj} < f = a_{0l}/a_{kl}.$$

We have from the last relation that

$$\frac{a_{0j} - a_{0l}\frac{a_{rj}}{a_{re}}}{a_{kj} - a_{kl}\frac{a_{rj}}{a_{re}}} > \frac{a_{0l}}{a_{kl}}$$

which is equivalently written as

$$a_{0j} > a_{kj}\frac{a_{0l}}{a_{kl}} \tag{6}$$

Relation (6) is similar to relation (5).

Considering the cases $a_{kj} = 0$, $a_{kj} > 0$ and $a_{kj} < 0$ we reach again the contradiction of Subcase 1. Hence, we showed that $\hat{g} = f$.

From (3) and $\hat{g} = f$ we have that

$$\frac{a_{0l}}{a_{kl}} > \frac{\hat{a}_{0q}}{\hat{a}_{kq}} \tag{7}$$

From the relations $a_{kl} < 0$, $\hat{a}_{kq} < 0$ and (7) we have

$$a_{0l} \times \hat{a}_{kq} < a_{kl} \times \hat{a}_{0q}$$

and hence

$$a_{0l}\left(a_{kq} - \frac{a_{rq}}{a_{rl}} \times a_{kl}\right) > a_{kl}\left(a_{0q} - \frac{a_{rq}}{a_{rl}} \times a_{0l}\right).$$

The last relation is equivalently written as

$$a_{0q} > a_{kq}\frac{a_{0l}}{a_{kl}},$$

which comes from (6) by replacing j with q. We have already shown that (6) does not hold. This completes the proof of the lemma.

THEOREM 4. If LP is non degenerate, the algorithm is finite.

PROOF. It is easily verified that the algorithm does not allow a primal feasible variable to become strictly negative. Hence, at most m subproblems are solved.

If the non degeneracy assumption holds, it is easily verified that the distinguished variables increase strictly from iteration to iteration.

4. COMPUTATIONAL RESULTS

In order to get an impression if we can have any benefit from the exterior point feature of the algorithm we have run several randomly generated test problems. And it seems to be the case. However, our computational study is very elementary and to verify this fact further more rigorous computational comparison is needed.

Both the primal simplex method and the algorithm were coded in standard FORTRAN 77 in a relatively simple way without using any sophisticated updating (factorization) technique. In particular, the tableau consisting of the non basic variables is updated at each iteration. The computations are performed with double precision arithmetic. The maximum coefficient pivoting rule was employed by the primal simplex algorithm. We have used a MICROWAY–AT (processor 80386/20 MHz) without a coprocessor under the operating system MS–DOS 3.3.

The problems were generated in the following form:

$$\min \{ c^T x : A x \le b, x \ge 0 \}.$$

where A is an $m \times n$ matrix. The range of the coefficients of A and b is $(-500, 500)$. The range of the coefficients of c is $(-1000, 0)$. Ten random problems of full density were generated for each of the four classes $m = n = 50, 100, 150, 200$. The CPU time is reported in the form hours : minutes : seconds. The mean CPU time is the fraction (total CPU time) / (number of problems solved). Similarly, (mean No. of iterations) = (total number of iterations) / (number of problems solved) and (mean CPU time per iteration) = (total CPU time) / (number of iterations of all the problems solved).

Table 1 reports the results of the computational study. Here EPSA (Exterior Point Simplex Algorithm) is our algorithm and the speed up factor is calculated as (dual simplex) / (EPSA). It seems to be evident, that for small problems there is no significant speed up, but as the problem size increases, the speed up is growing. The speed up seems to be more significant, if we extract those problems, that have an optimal solution. These results are presented in Table 2. For example, six of the problems in the class $m = n = 50$ were optimal. The results for these six problems are reported in Table 2.

Table 1. Computational results for random problems.

Problem dimension		Dual Simplex	EPSA	Speed up factor
$m = n = 50$	mean CPU time	0:00:46.12	0:00:45.67	1.0098
Problems	mean No. of iter.	74.500	70.800	1.05
solved: 10	mean CPU time/iter	0:00:00.62	0:00:00.65	
$m = n = 100$	mean CPU time	0:10:34.64	0:08:16.39	1.287
Problems	mean No. of iter.	272.500	209.000	1.3038
solved: 10	mean CPU time/iter	0:00:02.33	0:00:02.37	
$m = n = 150$	mean CPU time	0:42:48.76	0:33:30.03	1.2780
Problems	mean No. of iter.	500.600	386.800	1.2942
solved: 10	mean CPU time/iter	0:00:05.13	0:00:05.20	
$m = n = 200$	mean CPU time	2:09:11.46	1:29:11.47	1.4480
Problems	mean No. of iter.	858.200	587.000	1.462
solved: 10	mean CPU time/iter	0:00:09.03	0:00:09.12	

Table 2. Computational results for problems with optimum.

Problem dimension		Dual Simplex	EPSA	Speed up factor
$m = n = 50$	mean CPU time	0:00:49.89	0:00:43.10	1.5750
Problems	mean No. of iter.	80.500	66.667	1.207
solved: 6	mean CPU time/iter	0:00:00.62	0:00:00.65	
$m = n = 100$	mean CPU time	0:12:45.53	0:07:50.60	1.6267
Problems	mean No. of iter.	328.500	198.000	1.659
solved: 4	mean CPU time/iter	0:00:02.33	0:00:02.38	
$m = n = 150$	mean CPU time	0:48:11.91	0:33:07.81	1.4548
Problems	mean No. of iter.	563.400	382.400	1.473
solved: 5	mean CPU time/iter	0:00:05.13	0:00:05.19	
$m = n = 200$	mean CPU time	2:25:10.90	1:26:11.49	1.6840
Problems	mean No. of iter.	964.200	567.200	1.6999
solved: 5	mean CPU time/iter	0:00:09.03	0:00:09.12	

These computational results suggest that the extra effort (searching both a row and a column) for the proper pivot element required by our algorithm is less costly than the benefit obtained from this. Finally we point out that the times reported are very long for the sizes of the problems solved. However, this was expected as full dense problems were generated, no co-processor was used, the computations were performed with double precision arithmetic and all non basic columns were updated at each iteration.

REFERENCES

Adler,I., Resende,M.G.C., Veiga,G. and Karmarkar,N., (1989), "An Implementation of Karmarkar's Algorithm for Linear Programming", *Mathematical Programming* 44, 297- 335.

Ahatz,H., Kleinschmidt,P. and Paparrizos,K. (1989), "A Dual Forest Algorithm Problems", *DIMACS* 4 (Applied Geometry and Discrete Mathematics : The Victor Klee Festscrift), 1- 10.

Dantizg,G.B., (1963), *Linear Programming and Extensions*, Princeton University Press, Princeton N.J.

Ford,L.R. and Fulkerson,D.R., (1962), *Flows in Networks*, Princeton University Press, Princeton N.J.

Karmarkar,N., (1984), "A New Polynomial Time Algorithm for Linear Programming", *Combinatorica* 4, 373- 395.

Paparrizos,K., (1989), "An Infeasible (Exterior Point) Simplex Algorithm for Assignment Problems", *Mathematical Programming* 51, 45- 54.

Paparrizos,K., (1993), "An Exterior Point Simplex Algorithm for General Linear Programming Problems", *Annals of Operations Research*, to appear.

Terlaky,T., (1985), "A Convergent Criss-Cross Method", *Math. Oper. und Stat. ser. Optimization* 15/5, 683- 690.

Tomizowa,N., (1972), "On Some Techniques Useful for Solution of Transportation Network Flow Problems", *Network* 1, 173- 194.

Zionts,S., (1969), "The Criss-Cross Method for Solving Linear Programming Problems", *Management Sci.* 15/7, 426- 445.