A COMPUTATIONAL COMPARISON OF THE NETWORK SIMPLEX METHOD WITH THE DUAL AFFINE SCALING METHOD

Andrew Armacost and Sanjay Mehrotra*

Department of Industrial Engineering and Management Sciences
Northwestern University, Evanston, IL 60208, U.S.A.

(*Received*: *May 1990, Revised*: *November 1990*)

ABSTRACT

We compare the performance of an implementation of the dual affine scaling method using complete Cholesky factors with the network simplex method on (sparse) minimum cost network flow, transportation and assignment problems. We find that the network simplex method out-performs this implementation of affine scaling method. The ratio of times required by the two methods is found to grow with the number of nodes and the density of network. However, for very sparse problems we find the two methods to be competitive.

1. Introduction

We consider the minimum cost network flow (MCNF) problem with m nodes and n arcs. The MCNF problem can be written in the form

$$\text{Minimize } c^T x,$$

$$\text{subject to} \quad Ax = b, \qquad (1)$$

$$x \geqslant 0,$$

where $A \in R^{m \times n}$ is the node-arc incidence matrix. The demand in the network is assumed to be balanced, i.e., $\sum_{i=1}^{m} b_i = 0$.

The purpose of this paper is to compare the performance of an adaptation of simplex method for MCNF problems, with that of the dual affine scaling method using complete Cholesky factorization. As a result of this study we hope to develop further insight into the performance of interior point methods (IPM) for solving network problems.

*Research was supported in part by the Office of Naval Research grant N00014-87-K-0214 and by the National Science Foundation grant CCR-8810107.

The question is of interest because implementations of IPM based on complete Cholesky factors are found to be considerably superior than implementations of simplex method for large sparse problems. It is natural to compare how well this method compares for structured problems, such as the network flow problems.

In Aronson et al. [8], these methods were compared. The work here differs from [8] in several aspects. The implementation in [8] solved problems in the primal formulation by using subroutine LSQR [13]. The codes were developed to solve the scaled least squares problems at each interation in range and null space. No preconditioners were used while implementing LSQR. This resulted in prohibitably large number of conjugate gradient iterations. In [8] results were obtained on dense assignment problems. These results were very preliminary and most of the problems were not even solved to (near) optimality because of numerical difficulties.

The network problems, we consider in this paper, are relatively sparse. We use a complete Cholesky factor to solve dual problems. More importantly we differ in our conclusion about the prospects of using an IPM for solving these problems. We find enough evidence to suggest that IPM should be reconsidered as a viable method for these problems. In particular, we think that it might be possible to take advantage of IPM in designing algorithms for solving large network problems (particularly on parallel computers). This paper is organized as follows:

In Section 2 we describe our adaptation of the computer implementation of the network simplex method given in Chvatal [4, pp. 284-319]. We describe our implementation of the affine scaling method in Section 3. In this section we also give an approach to efficiently form the matrix AD^2A^T within the framework of network data structures. In Section 4 we discuss our computational results and make some concluding remarks. In the remainder of the current section we describe some data structures which are used in the later sections.

The kth column of A has exactly two nonzeros, at locations *head*(k) and *tail*(k). The matrix A is saved using the arrays *head*() and *tail*() of size n as follows:

arc #	head	tail
k	*head*(k)	*tail*(k).

Exhibit 1

As a convention an arc k, outgoing from node *tail*(k), results in $A_{tail(k)k} = -1$. The information about the network is also stored using a

first-next list involving arrays *first*() and *next*() of size m and n, respectively. The first-next list constructed for NETSIM is based on the *tail* array. It is formed as follows:

Initialize $first(i) = 0$, $i = 1,\ldots,m$
do $k = 1,\ldots,n$
$next(k) \leftarrow first(tail(k))$
$first(tail(k)) \leftarrow k$
od.

Exhibit 2

2. Implementation of Network Simplex Method

The implementation of NETSIM involves three main computational steps:

Step 1. Calculate (update) the dual price vector.

Step 2. Determine an entering arc.

Step 3. Find the leaving arc on the cycle created by the entering arc and update the flows.

We follow the computer implementation given in [4] to perform these steps. In particular, we use the *predecessor*, *depth*, and *thread* arrays in our implementation. We now describe specific approaches used in our implementation that influence the performance of NETSIM.

Finding the Initial Basis

It is well-known that the linear programming basis for a MCNF problem is a connected tree of m nodes and m—1 arcs. The process of finding basis is called phase 1.

As in Chvatal [4], we begin Phase 1 by creating an artificial network, consisting of all the original nodes, some original arcs and some artificial arcs. We create a two-tiered tree, with each arc going from the root to another node, or vice versa. This gives m—1 nodes of degree 1 and one node of degree m—1. The node m in the natural order of problem data is designated as the permanent root. The artificial network is constructed as follows.

To satisfy the demand at the nodes, we determine whether node i is a source or a sink. If it is a source ($b_i < 0$), the arc we add to the network

is directed from node i to the root, and it is assigned an initial flow of $-b_i$. If it is a sink or a pure transshipment node ($b_i \geq 0$), then the arc is directed from the root to node i. It is assigned an initial flow of b_i. First we use as many arcs as possible from the original network to satisfy the demand at nodes in the above manner. If we can not satisfy the demand at node i using the arcs in the original network, we add an artificial arc between node i and m to satisfy this demand. Associated with this artificial network is a new set of costs, c', where

$$c_k' = \begin{cases} 1 & \text{if arc } k \text{ is artificial} \\ 0 & \text{if arc } k \text{ is in the original network.} \end{cases}$$

We apply NETSIM to the artificial problem, using the costs c', until an optimal solution to the artificial problem (phase 1) is found. The tree at the optimal solution of phase 1 serves as the initial basis for the original problem.

Finding an Entering Arc

At each iteration, the entering arc was chosen as follows. We find the arc with most negative reduced cost at node i by using the tail *first-next* list of Exhibit 2. If an arc with negative reduced cost is found, we use it as an entering arc. Otherwise, we consider node $i+1$ (If $i = m$, $i + 1 \leftarrow 1$). If an arc with negative reduced cost was found at node i in the current iteration, we use node $i + 1$ to begin our search for an arc with negative reduced cost in the next interation. If no arc with a negative reduced cost is found after searching all the nodes, we know that an optimal solution has been reached.

Doubly Linked Thread

Chvatal [4] uses a *thread* array $s()$ of size n to perform a (*preorder*) depth-first search [4, p. 314]. When updating the information at the change of bases, it is necessary to find a node i which is the predecessor in the thread to a given node j [4, p. 317, Box 19.3 (step 0)]. Since the thread is known only in one direction (by the order of successors), one must traverse a portion of the thread to find the predecessor. This causes slight inefficiency. We doubly link the thread to find the predecessor, whenever needed. The array doubly linking the thread, which we call $rs()$, is updated whenever the thread is updated so that if $j = s(i)$, we have $i = rs(j)$.

3. Implementation of Dual Affine Scaling Method

We apply the affine scaling algorithm which was first proposed by Dikin

[5], and later discovered again by Barnes [3], Vanderbei et al. [14] and others, to the dual of (1) given by

$$\begin{aligned} &\text{Maximize} && b^T y, \\ &\text{subject to} && A^T y \leqslant c. \end{aligned} \tag{1}$$

Our implementation of this algorithm follows the work of Adler et al. [1], and Monma and Morton [11]. The affine scaling algorithm for (1) is outlined in Exhibit 3.

Begin Procedure ASCALE

Let y^0 be such that $v^0 \equiv c - A^T y^0 > 0$.

For $k = 0,1,\ldots$ until a termination criterion is satisfied do:

Step 1 $D \leftarrow \text{diag}\left(\frac{1}{v_1^k}, \ldots, \frac{1}{v_n^k}\right)$

Step 2 Solve $Bdy \equiv AD^2A^T dy = b$ for dy.

Step 3 $dv \leftarrow -A^T dy$

Step 4 $\alpha = \gamma \cdot \min_i \left\{ \frac{y_i^k}{dv_i} \,\middle|\, dv_i < 0, i=1, \ldots, n \right\}$, $0 < \gamma \leqslant 1$

Step 5 $y^{k+1} \leftarrow y^k + \alpha dy$

Step 6 $v^{k+1} \leftarrow v^k + \alpha dv$

roF

Exhibit 3. Dual affine scaling algorithm

Implementation Details for ASCALE

We terminated ASCALE if criterion

$$\text{rel}(z) \equiv \frac{b^T(y^{k+1} - y^k)}{\max(1, b^T y^k)} \leqslant 10^{-6}$$

was satisfied. The assumption that an initial interior point is available was satisfied by introducing an artificial variable y_e and considering the artificial problem

$$\begin{aligned} &\text{Maximize} && b^T y + M y_e, \\ &\text{subject to} && A^T y + e\pi_e \leqslant c, \\ & && y_e \leqslant 0, \end{aligned} \tag{2}$$

where e is a vector of all ones and M is a large positive number. $M = 10^6$ was used for all the test problems in our implementation. To get an interior point for (2) we set $y^0 = 0$, computed

$$\hat{y}_e^0 = \min_j c_j,$$

and used

$$y_e^0 = \begin{cases} 2 \cdot \hat{y}_e & \text{if } \hat{y}_e < 0 \\ 0 & \text{if } \hat{y}_e > 0 \\ 10^{-2} & \text{if } \hat{y}_e = 0. \end{cases}$$

Although we do not propose this way of starting the algorithm, we note that it worked quite satisfactorily for our test runs. $\gamma = .95$ was used for all iterations, except when $\gamma = 1.0$ resulted in $y_e = 0$. In this case we used $\gamma = 1.0$ and dropped the artificial variable from the problem.

The main work at each interation of ASCALE comes from solving the equations $Bdy = b$. This is performed by forming a Cholesky factor L of B, $LL^T = B$. Important to the efficiency is the permutation of the constraint matrix A. We find the permutation of rows of A by using an in-house implementation of the minimum degree ordering heuristic [6] on B. While finding this permutation, we also find the location of nonzero elements in L. Henceforth we assume that the rows in A have been permuted, i.e., the nodes of the network have been renumbered.

We compute one column of L at a time. The approach we used to form L is similar to the one described in Section 4 of [10]. But, in our implementation we used a different approach to compute a column of B at a time. This approach, which we describe next, efficiently forms columns of B using network data structure.

Formation of AD^2A^T

Let the arrays *maxnode* and *minnode* be defined as

$$maxnode(j) = \max(head(j), tail(j))$$
$$minnonde(j) = \min(head(j), tail(j)).$$

When the artificial variable is present in A, the direction of the arcs is important to find elements of the corresponding (dense) row in B. We use the array *maxnode*( ) also to indicate whether *maxnode*(j) is *head*(j) or *tail*(j). This flagging is done as follows:

$$maxnode(j) = \begin{cases} -maxnode(j), & \text{if } maxnode(j) = tail(j) \\ maxnode(j), & \text{if } maxnode(j) = head(j). \end{cases}$$

Next, we form a first-next list as in Exhibit 2 using the *minnode* array. The corresponding arrays are called *fmin—nmin*. The arrays *minnode*, *maxnode*, *fmin* and *nmin* are used in Exhibit 4 to form one column of B at a time.

In procedure FORMB the nonzeros corresponding to the artificial row are stored in array *art*( ). The arrangements of computation ensure that at the completion of computations for ith column of B, (note that only B_{ji}, $j \geq i$ are computed), B_{m+1i} is in $art(i)$, B_{ii} is in $diag(i)$ and B_{ji}, $m+1>j>i$ are in q( ).

Note that the computations in procedure FORMB use each arc of the network exactly once. Also note that B_{ji}, $m+1>j>i$ are calculated by

$$q(node) \leftarrow q(node) - D_k^2.$$

This allows proper calculation when the network contains nodes which are connected with arcs in both directions, i.e., we have an arc going from node i to j as well as an arc going from node j to i. If nodes are connected with exactly one arc, this assignment can be replaced with

$$q(node) \leftarrow -D_k^2.$$

Since $rank(A) = m - 1$ and row m is linearly dependent on the previous rows, when computing L, we do not perform computations for this row.

After the artificial variable is dropped from A, the calculations are simplified further. The calculations corresponding to the $(m+1)$th row of L are no longer needed. Consequently, any computations involving *art* are removed. Furthermore, we do not need to use array *maxnode* as a flag array.

Procedure FORMB

Initialize $q(i) \leftarrow 0$, $i=1,\ldots,m$

$art(i) \leftarrow 0, i=1,\ldots,m+1$

$diag(i) \leftarrow 0, i=1,\ldots,m$

for $i=1,\ldots,m-1$

$k \leftarrow fmin(i)$

while $k \neq 0$ do

$node \leftarrow maxnode(k)$

if $node < 0$ then

$node = -node$

$art(node) \leftarrow art(node) - D_k^2$

$art(i) \leftarrow art(i) + D_k^2$

else

$art(node) \leftarrow art(node) + D_k^2$

$art(i) \leftarrow art(i) - D_k^2$

fi

$art(m+1) \leftarrow art(m+1) + D_k^2$

$diag(node) \leftarrow diag(node) + D_k^2$

$q(node) \leftarrow q(node) - D_k^2$

$diag(i) \leftarrow diag(i) + D_k^2$

$k \leftarrow nmin(k)$

elihw

$q(i) \leftarrow diag(i)$

$q(m+1) \leftarrow art(i)$

{Compute the ith column of the Cholesky factor using the information stored in $q(i), \ldots, q(m)$}

rof

$q(m+1) \leftarrow art(m+1) + D_{n+1}^2$

{Compute the $(m+1, m+1)$th element of L.}

Exhibit 4. Pseudo-code to compute columns of B.

4. Test Problems, Computational Comparison and Conclusions

Test Problems

The Network Generator NETGEN [9] was used to generate the test

set. The random seed 13502460 was used to generate all the problems. Three classes of problems were generated: assignment problems, transportation problems, and pure minimum cost network flow problems. The number of nodes in the network were either 100,200, or 400, and the number of arcs varied as a percentage of the number of nodes. This allows us to observe the behaviour of NETSIM and ASCALE on the basis of problem class, number of nodes and number of arcs. The actual size of these problem is shown in Tables 1 to 3, respectively.

The number of nodes, number of arcs, number of source nodes, number of sink nodes, total supply and the objective value of the generated problems are given in Columns 2, 3, 4, 5, 6 and 7, respectively. All problems were generated by allowing the cost to vary between —10 and 100.

Computational Comparison

The NETSIM and ASCALE were implemented in Fortran 77. All floating point operations were performed in double precision. All runs of NETSIM and ASCALE were made on a VAX 8650. Source codes were compiled with highest optimization level.

TABLE 1—SIZE OF THE ASSIGNMENT PROBLEMS

Problem	*Nodes*	*Arcs*	*Sources*	*Sinks*	*Supply*	*Objective*
1	100	150	50	50	50	1652
2	100	200	50	50	50	1392
3	100	300	50	50	50	1120
4	100	400	50	50	50	727
5	100	600	50	50	50	372
6	100	800	50	50	50	214
7	200	300	100	100	100	3373
8	200	400	100	100	100	2830
9	200	600	100	100	100	2157
10	200	800	100	100	100	1578
11	200	1000	100	100	100	1097
12	200	1200	100	100	100	805
13	400	600	200	200	200	7572
14	400	800	200	200	200	6118
15	400	1200	200	200	200	3989
16	400	1600	200	200	200	2467

TABLE 2—SIZE OF THE TRANSPORATATION PROBLEMS

Problem	*Nodes*	*Arcs*	*Sources*	*Sinks*	*Supply*	*Objective*
17	100	159	50	50	10000	453992
18	100	205	50	50	10000	390535
19	100	303	50	50	10000	222581
20	100	400	50	50	10000	257366
21	100	600	50	50	10000	105715
22	100	800	50	50	10000	117339
23	200	314	100	100	20000	855350
24	200	415	100	100	20000	870133
25	200	610	100	100	20000	621836
26	200	810	100	100	20000	690686
27	200	1010	100	100	20000	433388
28	200	1207	100	100	20000	340194
29	400	627	200	200	40000	2096668
30	400	827	200	200	40000	1649510
31	400	1230	200	200	40000	1273089
32	400	1623	200	200	40000	891325

TABLE 3—SIZE OF THE PURE NETWORK PROBLEMS

Problem	*Nodes*	*Arcs*	*Sources*	*Sinks*	*Supply*	*Objective*
33	100	157	15	50	10000	1572384
34	100	200	15	50	10000	1192889
35	100	300	15	50	10000	65536
36	100	400	15	50	10000	469192
37	100	600	15	50	10000	331997
38	100	800	15	50	10000	144349
39	200	318	30	100	20000	2798560
40	200	411	30	100	20000	2756577
41	200	600	30	100	20000	1659636
42	200	800	30	100	20000	760791
43	200	1000	30	100	20000	942029
44	200	1200	30	100	20000	654223
45	400	637	60	200	40000	5526523
46	400	806	60	200	40000	4571204
47	400	1200	60	200	40000	2972503
48	400	1600	60	200	40000	1929708

Tables 4 to 6 show run-time information for NETSIM and ASCALE. For NETSIM, the CPU time in seconds, the number of iterations completed in Phase 1, the total number of iterations in both Phases 1 and 2 are given in Columns 2, 3 and 4, respectively. For ASCALE the CPU time in seconds, the number of nonzeros in the Cholesky factor, L, and total number of iterations are shown in Columns 5, 6 and 7, respectively. Finally, the ratio of the two CPU times (ASCALE : NETSIM) is given in Column 8 to provide a comparison of the two codes for each of the problems.

At termination the dual objective value for all the problems was accurate to 7 significant digits. A feasible solution for all the problems was found after one iteration. Also of concern to us is the ability of the affine scaling code to recover the primal solution. At the termination of ASCALE estimates of primal solution $\tilde{x}^*$, $\tilde{s}^*$ are obtained as follows:

$$\tilde{x}^* = D^2 A^T dy,$$

TABLE 4—PERFORMANCE OF NETSIM AND ASCALE ON ASSIGNMENT PROBLEMS

Problem	*NETSIM*			*ASCALE*			*CPU Ratio ASCALE: NETSIM*
	CPU (Sec.)	*Ph1 Iter.*	*Total Iter.*	*CPU (Sec.)*	*Nonz (L)*	*Total Iter.*	
1	0.13	121	178	0.27	392	15	2.08
2	0.16	112	195	0.48	657	16	3.00
3	0.25	114	296	0.91	1064	16	3.64
4	0.23	109	311	1.83	1365	17	7.96
5	0.30	94	355	2.31	1883	18	7.70
6	0.31	89	370	3.10	2380	17	10.00
7	0.40	252	374	0.93	1054	16	2.30
8	0.55	252	518	2.18	2080	17	3.96
9	0.78	231	649	4.46	3360	18	5.72
10	0.78	257	739	6.40	4316	17	8.21
11	0.90	224	783	9.81	5343	18	10.90
12	1.04	223	878	13.66	6770	17	13.13
13	1.37	526	866	3.26	3060	17	2.38
14	2.20	550	1317	10.96	6398	17	4.98
15	3.00	540	1870	32.53	12292	18	10.84
16	3.20	497	1743	45.70	15205	19	14.28

TABLE 5—PERFORMANCE OF NETSIM AND ASCALE ON TRANSPORTATION PROBLEMS

Problem	*NETSIM*			*ASCALE*			*CPU Ratio ASCALE: NETSIM*
	CPU (Sec.)	*Ph1 Iter.*	*Total Iter.*	*CPU (Sec.)*	*Nonz (L)*	*Total Iter.*	
17	0.13	134	189	0.36	426	17	2.76
18	0.17	153	222	0.63	704	19	3.71
19	0.26	153	291	1.30	1171	20	5.00
20	0.30	140	345	2.45	1481	19	8.17
21	0.31	139	370	2.96	2034	20	9.55
22	0.40	140	394	4.13	2486	20	10.33
23	0.46	264	413	1.16	1125	20	2.52
24	0.58	330	525	2.57	2031	20	4.43
25	1.04	295	811	7.08	3909	21	6.81
26	0.98	295	802	15.65	5708	24	15.97
27	1.02	299	765	17.24	6221	23	16.90
28	1.16	305	412	23.33	7078	26	20.11
29	1.54	547	881	3.81	3170	18	2.47
30	2.26	650	1219	14.97	6304	22	6.62
31	3.84	759	1786	48.12	12383	24	12.53
32	3.72	645	1773	71.72	17035	23	19.28

and

$$\hat{x}^* = \begin{cases} (D^2A^Tdy)_t, & \text{if } (D^2A^Tdy)_t > 0 \\ 0, & \text{otherwise.} \end{cases}$$

Here D and dy are the corresponding vectors used at the last iteration of ASCALE

Second column of Tables 7, 8 and 9 provide the normalized non-negativity violation $\min \{\tilde{x}_t^*/\|\tilde{x}^*\|\}$. The third column of these tables give maximum normalized violation in the complementary slackness $\max \{ | \tilde{x}_t^*(c-A^T\pi^k)_t | / \|\tilde{x}^*\|\|c-A^T\pi^k\| \}$. The fourth column of these tables give the normalized error in the feasibility of primal solution max

TABLE 6—PERFORMANCE OF NETSIM AND ASCALE ON NETWORK FLOW PROBLEMS

Problem	*NETSIM*			*ASCALE*			*CPU Ratio ASCALE: NETSIM*
	CPU (Sec.)	*Ph-1 Iter.*	*Total Iter.*	*CPU (Sec.)*	*Nonz (L)*	*Total Iter.*	
33	0.12	124	147	0.38	325	22	3.17
34	0.18	126	207	0.54	481	22	3.00
35	0.24	121	258	0.83	824	19	3.46
36	0.25	120	267	1.73	1071	22	6.92
37	0.31	118	391	2.46	1535	23	7.94
38	0.39	119	403	2.39	1391	25	6.13
39	0.38	233	325	1.03	894	21	2.71
40	0.51	372	468	2.08	1428	25	4.08
41	0.76	290	631	4.67	2551	26	6.14
42	0.89	278	719	6.12	3389	21	6.87
43	0.92	274	809	8.73	4254	22	9.49
44	1.11	264	855	10.61	4455	23	9.56
45	1.29	490	738	3.95	2624	22	3.06
46	2.06	563	1064	10.58	4327	28	5.14
47	2.72	579	1340	29.07	8570	27	10.69
48	3.74	540	1896	39.91	11507	23	10.67

$\{(b-A\hat{x}^*)_i/\| b \|\}$. And finally the last column of these tables give the primal value $c^T\hat{x}^*$ recorded at the termination. The results in these table confirm the observation made in [1] that typically primal solutions can be obtained at the termination of dual affine scaling algorithm.

The following conclusions are drawn from the results reported in Tables 4 to 9.

- The results indicate that the number of iterations required to satisfy termination criterion grow very moderately with the complexity of the problems. The termination criterion in ASCALE was satisfied after 15 to 19 iterations for assignment problems, 17 to 26 iterations for transportation problems and 19 to 28 iterations for the minimum

TABLE 7—QUALITY OF PRIMAL SOLUTIONS FOR ASSIGNMENT PROBLEMS

Problem	$Min\left\{\frac{\tilde{x}_i^*}{\|\tilde{x}^*\|}\right\}$	$Max\left\{\frac{\tilde{x}_i^*(c-A^T\pi^k)_i}{\|\tilde{x}^*\|\|c-A^T\pi^k\|}\right\}$	$Max\left\{\frac{(b-A\tilde{x}^*)_i}{\|b\|}\right\}$	$c^T\tilde{x}^*$
1	-3.83e-14	3.03e-09	3.83e-14	1652.0
2	-3.06e-15	3.97e-10	3.19e-15	1392.0
3	-6.77e-15	4.56e-10	1.48e-14	1120·0
4	-2.91e-13	2.77e-10	2.92e-13	772.0
5	-4.05e-15	7.55e-11	5.00e-15	372.0
6	-2.51e-15	8.57e-11	4.99e-15	214.0
7	7.45e-15	5.58e-10	8.06e-15	3373.0
8	-1.16e-15	1.89e-10	1.50e-15	2830.0
9	-2.21e-14	2.04e-10	3.62e-14	2157.0
10	-2.33e-14	3.00e-10	2.65e-14	1578.0
11	-7.18e-15	1.63e-10	1.28e-14	1097.0
12	-4.55e-14	1.11e-10	5.71e-14	805.0
13	-1.82e-12	6.69e-10	1.82e-12	7572.0
14	-5.04e-14	3.72e-10	6.50e-14	6118.0
15	-1.56e-12	1.48e-10	1.57e-12	3989.0
16	-2.35e-12	4.38e-11	2.35e-12	2467.0

cost network flow problems. Also the increase in the number of iterations required by ASCALE was moderate with the size of problems within each problem class.

- The performance of ASCALE is directly linked with the number of nonzeros in L. For a fixed number of nodes, an increase in the number of arcs results in an increase in the number of nonzeros in B, and consequently in L. Loosely speaking, the increase in average number of nonzeros in B increases the total work to compute L between linearly and quadratically.

  On the other hand, the work required by NETSIM seems to grow only linearly with the increase in the number of arcs (due to partial pricing). Therefore, for all the network problems, one expects to see an increasing trend in the ratio of CPU times required by the two algorithms,

TABLE 8—QUALITY OF PRIMAL SOLUTIONS FOR TRANSPORTATION PROBLEMS

Problem	$Min \left\{ \frac{\tilde{x}_i^*}{\|\tilde{x}^*\|} \right\}$	$Max \left\{ \frac{\tilde{x}_i^*(c-A^T\pi^k)_i}{\|\tilde{x}^*\|\|c-A^T\pi^k\|} \right\}$	$Max \left\{ \frac{(b-A\tilde{x}^*)_i}{\|b\|} \right\}$	$c^T\tilde{x}^*$
17	-1.04e-12	4.21e-10	1.13e-12	453992.0
18	-1.63e-10	7.46e-10	1.63e-10	390545.0
19	-6.16e-11	1.86e-10	6.17e-11	222581.0
20	-2.43e-13	1.60e-10	2.90e-12	257366.0
21	-7.45e-12	1.62e-10	8.88e-12	105715.0
22	-5.32e-12	4.21e-11	5.34e-12	117339.0
23	-1.43e-13	1.36e-10	1.44e-14	855350.0
24	-1.63e-11	2.78e-10	1.70e-11	870133.0
25	-4.15e-10	5.44e-10	4.15e-10	621835.9
26	-6.05e-14	2.94e-11	6.20e-14	390686.0
27	-2.48e-12	6.84e-11	2.51e-12	433387.9
28	-3.99e-12	6.06e-11	7.85e-12	340194.0
29	-3.67e-10	6.11e-10	3.67e-10	2096668.0
30	-2.56e-11	8.05e-11	2.56e-11	1649510.0
31	-1.09e-11	8.21e-11	2.24e-11	1273089.0
32	-3.28e-11	4.80e-11	3.32e-11	891325.0

as the network gets richer in arcs. This is clearly visible from the results in Tables 4,5 and 6.

- It is interesting to observe that the ratio of times required by NETSIM and ASCALE also grow as the number of nodes (m) is increased, while keeping n/m fixed. This is because with an increase in m the average work required to update the information at each iteration of NETSIM does not grow as fast as the number of nonzeros in L.

- An important observation from the results is that for very sparse network problems the ratio of CPU times required by ASCALE and NETSIM is small. This is despite the fact that all the computations in NETSIM heavily exploit the integrality (± 1) of data in A and additional network properties. This property is lost in ASCALE

TABLE 9—QUALITY OF PRIMAL SOLUTION FOR NETWORK FLOW PROBLEMS

Problem	*Min* $\left\{ \frac{\tilde{x}_i^*}{\|\tilde{x}^*\|} \right\}$	*Max* $\left\{ \frac{\tilde{x}_i^*(c-A^T\pi^k)_i}{\|\tilde{x}^*\|\|c-A^T\pi^k\|} \right\}$	*Max* $\left\{ \frac{(b-A\hat{x}^*)_i}{\|b\|} \right\}$	$c^T\hat{x}^*$
33	-4.92e-14	9.54e-10	5.04e-14	1572384.0
34	-3.31e-11	9.78e-10	5.41e-11	1192889.0
35	-9.12e-11	7.88e-10	9.70e-11	655636.0
36	-4.81e-12	3.07e-10	4.85e-12	469192.0
37	-1.02e-12	8.45e-11	1.09e-12	331997.0
38	-7.86e-12	9.25e-11	7.86e-12	144349.0
39	-1.47e-11	5.86e-10	1.48e-11	2798560.0
40	-2.08e-09	4.17e-10	2.08e-09	2756577.0
41	-1.02e-10	5.14e-10	1.02e-10	1659635.0
42	-6.54e-12	1.30e-10	6.64e-12	760791.0
43	-3.65e-09	1.01e-09	3.65e-09	942029.1
44	-4.72e-11	1.41e-10	4.94e-11	654223.0
45	-2.41e-11	4.88e-10	2.41e-11	5526525.0
46	-1.12e-10	1.09e-10	1.21e-10	4571204.0
47	-6.19e-10	1.35e-10	6.20e-10	2972503.0
48	-2.39e-10	3.27e-10	2.41e-19	1929707.0

when performing computations involving L. Nonzeros in L are real numbers.

- The network problems for which results are reported in this paper are small in size. We solved some larger problems as well. On a pure network problem with 1000 nodes and 5000 arcs ASCALE was slower than NETSIM by a factor of 80. The number of nonzeros in L for this problem were about 120,000, an average of 120 per column of L. However, for similar problems with 1000 nodes and 1500 arcs and 5000 nodes and 7500 arcs, ASCALE was slower than NETSIM by a factor less than 3. This supports earlier conclusions.

Concluding Remarks

We find that the performance of affine scaling method computing complete Cholesky factor could deteriorate as the networks get richer in the

number of arcs, and as the network gets bigger while keeping arc/note ratio fixed. The growth in the nonzeros in L with an increase in the number of arcs (with fixed m) can possibly be controlled by using a preconditioned conjugate gradient method to approximately solve $Bdy = b$. We believe that the work in Mehrotra [10] would be useful in this direction. Karmarkar and Ramakrishnan [7] have reported some results using this approach on randomly generated network flow problems *on square grid graph* with 10,001 nodes and 20,000 arcs and 40,001 nodes and 80,000 arcs. It is expected that the work involved in computation of a preconditioner $\tilde{L}$ and in the implementation of preconditioned conjugate gradient method is more parallalizable than the work in updates of simplex method. Also the finding that the performance of affine scaling method is within a small factor of the performance of network simplex method is very encouraging. This indicates the possibility of designing hybrid methods in the future.

References

[1] Adler, I, Karmarkar, N., Resende, M. and Viega, G. (1989), An implementation of Karmarkar's algorithm for linear programming, *Mathematical Programming*, 44(3), 297-336.

[2] Adler, I., Karmarkar, N., Resende, M. and Viega, G. (1989), Data structure and programming techniques for the implementation of Karmarkar's algorithm, *ORSA Journal on Computing*, 1(2), 84-106.

[3] Barnes, E.R. (1986), A variation of Karmarkar's algorithm for solving linear programming problems, *Mathematical Programming*, 36, 174-182.

[4] Chvatal, V. (1983), *Linear Programming*, W.H. Freeman and Co., New York.

[5] Dikin, I.I. (1967), Iterative solution of problems of linear and quadratic programming, *Soviet Math. Dokl.* 8, 674-675.

[6] George, A. and Liu, J. (1981), *Computer Solutions for Large Positive Definite Systems*, Harcourt Press,. Princeton, NJ.

[7] Karmarkar, N. and Ramakirsiinan, K.G., Implementation and Computational Results of the Karmarkar Algorithm for Linear Programming Using an Iterative Method for Computing Projections, Extended Abstract circulated during 13th International Symposium on Mathematical Programming, Tokyo, Japan, 1988.

[8] Aronson, J., Barr, R., Helgason, R., Kennington, J., Loh, A. and Zaki, H. The Projective Transformation Algorithm by Karmarkar: A Computational Experiment with Assignment Problems, Technical Report 85-OR-3, Department of Operations Research, Southern Methodist University, 1985.

[9] Klingman, D., Napier, A. and Stutz, J. (1974), NETGEN: A program for generating large scale capacitated assignment, transportation and minimum cost flow network problems, *Management Science*, 20(5), 814-821.

[10] MEHROTRA S., Implementations of Affine Scaling Methods : Approximate Solutions of Systems of Linear Equations Using Preconditioned Conjugate Gradient Method, TR 89-04, Department of IE/MS, Northwestern University, Evanston, IL 1989.

[11] MONMA, C.L. AND MORTON, A.J. (1987), Computational experience with a dual affine variant of Karmarkar's method for linear programming, 6(6), *OR Letters*, 261-267, 1987.

[12] MURTAGH, B.A. AND SAUNDERS, M.A., MINOS 5.0 user's guide, Technical report SOL 83-20, Department of Operations Research, Stanford University, Stanford, California, 1983.

[13] PAIGE, C.C. AND SAUNDERS, M.A. (1982), Algorithm 583 LSQR: Sparse linear equations and least squares problems, *ACM Transactions on Mathematical Software*, 8(2), 195-209.

[14] VANDERBEI, R.J., MEKETON, M.S. AND FREEDMAN, B.A. (1986), A modification of Karmarkar's linear programming algorithm, *Algorithmica*, 1, 395-407.