SCHOOL OF OPERATIONS RESEARCH
AND INDUSTRIAL ENGINEERING
COLLEGE OF ENGINEERING
CORNELL UNIVERSITY
ITHACA, NEW YORK

TECHNICAL REPORT NO. 653

February 1985

COMPUTATIONAL EXPERIENCE WITH A POLYNOMIAL-TIME DUAL SIMPLEX ALGORITHM FOR THE TRANSPORTATION PROBLEM*

by

Yoshiro Ikura**
and
George L. Nemhauser

*This work was supported by National Science Foundation Grant ECS8307473 to Cornell University.

**Currently at the Consilium, Inc., 2479 E. Bayshore Road, Suite 700, Palo Alto, California 94303.

Computational Experience with a Polynomial-time Dual Simplex Algorithm for the Transportation Problem [1]

Yoshiro Ikura [2]
and
George L. Nemhauser [3]

[1] Presented at the ORSA/TIMS meeting at San Francisco, May 1984.

[2] Currently at *Consilium*, Inc., 2479 E. Bayshore Road, Suite 700, Palo Alto, California 94303

[3] School of Operations Research and Industrial Engineering, Upson Hall, Cornell University, New York 14853.
This work was supported by National Sciences Foundation Grant ECS8307473 to Cornell University.

ABSTRACT

This paper reports on the implementation and computational experience obtained with a polynomial-time dual simplex algorithm for the transportation problem. The polynomial-time version of the algorithm is shown to be empirically, as well as theoretically, more efficient than a standard version. In particular, scaling of the supplies and demands is very effective computationally. While our computational results indicate that the dual simplex algorithm is slower than a primal simplex algorithm for large problems by a factor of 30 - 60%, it is competitive for smaller problems. Therefore, it should be useful in applications where dual methods are desirable for other reasons.

Keywords: Transportation problem, Dual simplex algorithm, Computational study.

1 Introduction

This paper reports on the implementation of dual network simplex methods for solving the transportation problem. The implementation is based on an algorithm developed by Ikura and Nemhauser [10], which employs a special column selection rule to guarantee that the total number of dual simplex pivots is bounded by a polynomial in the number of sources and sinks and the sum of supplies and demands. By incorporating a scaling technique developed by Edmonds and Karp [5], the bound on the number of pivots becomes a polynomial in the input length.

This report also compares the empirical efficiency of our dual transportation simplex method with a primal network simplex method. There are numerous reports in the literature (e.g., [3], [7], [8], [14]) on efficient implementation of primal network simplex methods. It is generally believed that primal simplex methods are practically superior to dual, primal-dual and out-of-kilter methods [3], [7].

Some of the earlier results on dual methods can be found in [1]. Recently, a dual approach [16] has been shown to be competitive with the primal approach. Moreover, dual simplex algorithms [2], [10] and a parametric simplex algorithm with a dual-type pivoting scheme [15] are known to be theoretically efficient. These results motivated us to investigate if our polynomial-time dual simplex algorithm for the transportation problem is practically efficient as well.

The dual simplex algorithm uses a spanning tree representation of bases. As in the primal simplex algorithm, it is possible to use efficient data structures for representing and manipulating trees. A fundamental difference between the primal and dual algorithms is that in the primal the tree edges represent basic variables, while in the dual they represent nonbasic variables. Thus, the primal ratio test is faster than the dual ratio test, and the dual pricing is faster than the primal one. However, since partial pricing is possible, but a partial ratio test is not, a primal pivot step with partial pricing is faster than a dual pivot step. On the other hand, the primal algorithm typically requires more pivots and a much greater percentage of the pivots are degenerate. Consequently, there is a tradeoff between number of pivots and time per pivot, with the dual simplex being superior for the former criterion and the primal simplex being superior for the latter. Our computational experience supports these conclusions, but also shows that for the larger problems we tested the time per pivot dominates and that the primal simplex algorithm is 40 - 50% faster. However, for dense problems with up to 16,000 variables and sparse ones with up to 10,000 variables, the polynomial-time dual simplex algorithm was at least as fast as a state-of-the-art primal code.

Perhaps our most interesting empirical finding is that scaling, which is easily incorporated in the dual simplex algorithm, is computationally very effective. Our results indicate that scaling helps in the selection of a good variable to enter the basis. This conclusion is also supported by recent results (see [4],[9]) on network flow problems. These results refute the claim that scaling is merely a theoretical device for achieving polynomiality (e.g. [13] p. 158).

We state the transportation problem in an unconventional format so that its dual will be in the format we need. Let $G = (V, E)$ be a bipartite graph, where $V = V_1 \cup V_2, V_1$ is the

set of supply vertices and V_2 is the set of demand verticies. For $i \in V_1$ and $j \in V_2$, $(i,j) = e \in E$ if shipments can be made directly from i to j. For all $e \in E$, the unit shipping cost is assumed to be a non-negative integer b_e. For all $i \in V$, let w_i be the non-negative integral supply or demand, which we call the *weight*, associated with vertex i.

A transportation problem can be formulated as

(TP) : $\quad min \sum_{e \in E} b_e y_e$

$$\sum_{\{e:e=(i,j) \in E\}} y_e + z_i = w_i \qquad i \in V \tag{1}$$

$$y_e \geq 0 \qquad e \in E \tag{2}$$

$$z_i \leq 0 \qquad i \in V. \tag{3}$$

For the standard transportation problem, we have $\sum_{i \in V_1} w_i = \sum_{i \in V_2} w_i = M$ and $z_i = 0$ for all $i \in V$. (TP) is equivalent to the standard problem by changing the shipping cost b_e to $b'_e = b_e + K$ for each e, where K is an integer such that $K > M \cdot max\{b_e : e \in E\}$. The dual of (TP) is

(DTP) : $\quad max \sum_{i \in V} w_i x_i$

$$x_i + x_j + s_e = b_e \qquad (i,j) = e \in E \tag{4}$$

$$x_i \geq 0 \qquad i \in V \tag{5}$$

$$s_e \geq 0 \qquad e \in E. \tag{6}$$

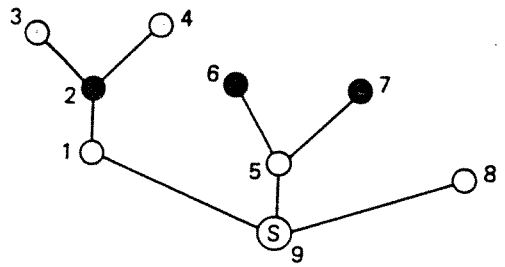
Our algorithm is a primal simplex method that solves the dual of the transportation problem (DTP). Thus, in the remainder of the paper, we assume that (DTP) is the primal problem; (x, s) are the primal variables and (y, z) are the dual variables.

2 Graphical Description of Solutions and Statement of the Algorithm

We describe a complementary pair of primal and dual solutions graphically by considering a *rooted spanning forest* of G, denoted by (F, R). The forest F is a union of disjoint trees $T_r = (V_r, E_r)$, where V_r contains a designated vertex r called the *root* of T_r. The subset R of V is a collection of roots of F. An edge is a *tree edge* if it is contained in one of the trees of F. The set of tree edges is denoted by $E(F)$. We color the vertices of T_r either black (B) or white (W) in such a way that the root is white and the colors alternate so that each edge of E_r connects a black vertex and a white vertex. We denote the set of black vertices by V^B and the set of white vertices by V^W. Let $V_r^B = V^B \cap V_r$ and $V_r^W = V^W \cap V_r$, $r \in R$.

For $e \in E$, the *branch* $B_e = (V_e, E_e)$ of T_r is the component of $T_r/\{e\}$ that does not contain r. Let $V_e^B = V^B \cap V_e$ and $V_e^W = V^W \cap V_e$. The *root* of B_e, where $e = (t, u)$, is the vertex $t \in V_e$. The branch is called black or white according to the color of its root. The *leaves* of T_r are the non-root vertices of degree one.

To represent a rooted spanning forest (F, R), we add a super vertex s and define an augmented spanning tree $T(F, R)$ which is the union of the trees T_r for $r \in R$, the super vertex s and the artificial edges between s and each root $r \in R$. An example is shown in Figure 2.1.



$V = \{1, ..., 8\}$

$R = \{1, 5, 8\}$

$F = \{T_1, T_5, T_8\}$

super vertex: $s = 9$

Figure 2.1

For a given rooted tree $T(F, R)$, we define the following primal and dual solutions. The primal values are determined recursively from each root toward the leaves in each tree of F:

$$x_i = \begin{cases} 0 & for\ i \in R \\ b_e - x_j & for\ e = (i, j) \in E(F), i \in V_e \end{cases} \tag{7}$$

$$s_e = b_e - x_i - x_j \quad for\ e = (i, j) \in E. \tag{8}$$

The dual values are determined recursively from the leaves toward each root:

$$y_e = w_i - \sum_{\{f : f = (k,i) \in E_e\}} y_f \ for\ e = (i, j) \in E(F) \tag{9}$$

$$z_i = y_e \ given\ in\ (9)\ for\ e = (i, s), i \in R. \tag{10}$$

In (9), we set $y_e = 0$ for $e \notin E(F)$, and in (10), $z_i = 0$ for $i \notin R$. Note that in (9), $y_e = w_i$ if B_e is a leaf i. The solution to (7) - (10) satisfies the complementary slackness conditions: $z_i x_i = 0$ for each $i \in V$, and $y_e s_e = 0$ for each $e \in E$.

The algorithm starts with an initial tree $T(F_o, R_o)$ in which each $i \in V$ is a root. In each pivot-operation, we first choose an edge to be eliminated from the spanning tree, which corresponds to choosing a variable to enter the basis, and then choose an entering edge, which corresponds to selecting a variable to leave the basis. Specifically, one pivot operation of the simplex algorithm for (DTP) changes the tree $T(F_t, R_t)$ to a new tree $T(F_{t+1}, R_{t+1})$

by cutting one branch of $T(F_t, R_t)$ and attaching it to another branch of $T(F_t, R_t)$. We keep the primal feasibility conditions (4) - (6) and the dual feasibility condition (1). The algorithm stops with an optimal tree $T(F_*, R_*)$ when $y_e \geq 0$ for $e \in E$ and $z_i \leq 0$ for $i \in V$.

To describe the pivot operations, we use the convention that each tree $T_r, r \in R_t$, is a black branch B_e of $T(F_t, R_t)$ where $e = (r, s)$. There are two types of pivot operations depending on whether a new root is created. Let $B_e = (V_e, E_e)$ be the branch to be cut in the tree $T(F_t, R_t)$. If B_e is black, define $q*$ by

$$q* = min\{min\{x_i : i \in V_e^B\}, min\{s_f : f = (i,j), i \in V_e^W, j \notin V_e\}\} \qquad (11a)$$

and if B_e is white,

$$q* = min\{min\{x_i : i \in V_e^W\}, min\{s_f : f = (i,j), i \in V_e^B, j \notin V_e\}\} \qquad (11b)$$

Equation (11) is just the standard ratio test of the simplex algorithm. Operation 1 takes place if $q* = x_i$ for some i. This operation corresponds to changing the root in the tree T_r from vertex r to vertex i, or making the branch B_e an independent tree T_i. Thus, $T(F_{t+1}, R_{t+1})$ is obtained from $T(F_t, R_t)$ by replacing edge e by the edge (i, s) where s is the super vertex. Operation 2 occurs if $q* = s_f$ for some $f = (i, j)$. In this case, the tree T_r is attached to a vertex j by the edge (i, j), or the branch B_e is cut and attached to the vertex j by the edge (i, j). Thus, $T(F_{t+1}, R_{t+1})$ is obtained from $T(F_t, R_t)$ by adding the edge (i, j) and deleting edge e.

We now give a formal statement of the general algorithm, which we call DTRAN. This statement omits the rule for choosing the variable to become basic except for requiring it to have the proper sign.

DTRAN ALGORITHM

STEP 0: [Initialization]

Let $t = 0, (x, s) = (0, b), (y, z) = (0, w)$ and $R_o = V$. Thus, initially, each rooted tree is a single vertex and $T(F_o, R_o)$ is the spanning tree obtained by joining each $i \in V$ to the super vertex s.

STEP 1: [Pricing]

If $y_e \geq 0$, for all $e \in E$, and $z_j \leq 0$ for all $j \in R_t$.

then stop.

else find $e \in E$ such that $y_e < 0$,

or $r \in R_t$ such that $z_r > 0$, and let $e = (r, s)$.

STEP 2: [Ratio Test]

For the edge e chosen in Step 2, determine $q*$ by (11).

If $q* = x_i$ for some $i \in V$,

then do operation 1

else do operation 2.

STEP 3: [Update]

Update (x, s) and (y, z) so that relations (7) - (10) are satisfied in the new tree structure $T(F_{t+1}, R_{t+1})$. Let $t = t + 1$, and go to Step 1. □

In Step 0, various parameters are initialized. The pricing procedure in Step 1 checks the dual feasibility conditions (2) - (3). If they are all satisfied, we stop with an optimal solution. Otherwise, we pick either a y_e or a z_r that violates (2) or (3), which determines a leaving edge. The ratio test in Step 2 uses the primal values (x, s) to choose an entering edge. The spanning tree $T(F_t, R_t)$ is then restructured and the primal and dual values are updated with respect to the new spanning tree $T(F_{t+1}, R_{t+1})$.

In Step 1, there are various ways to choose a leaving edge. In [10], we gave a specific choice based on the structure of $T(F_t, R_t)$ that bounds the number of pivots by a polynomial in $\mid V \mid$ and $\sum_{i \in V} w_i$. Our criterion can be summarized in the following way. First in a tree with at least one $y_e < 0$ or $z_r > 0$, we find an edge with a negative dual value whose branch does not contain any edges with a negative value. We cut the branch temporarily (we call it a *negative branch*), and adjust the dual values in the remainder of the tree. We recursively cut the branches and adjust the dual values in this way; at the end, a tree is decomposed into negative branches. Then we choose an edge whose negative branch is closest to the root in the sense that it does not have any other negative branches above it with a differently colored root. Thus, in the example of Figure 2.2, where the large circles represent negative branches, the candidates for leaving edges are e_1, e_2 and e_6.

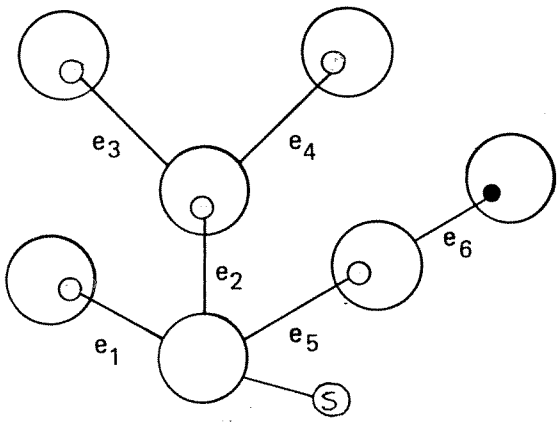


Figure 2.2

3 Scaling and Polynomiality

In the polynomial-time version of DTRAN, a scaling technique is applied to the integer weights w_i (i.e. supplies and demands in (TP)) of (DTP). In the jth scaled problem of (DTP), instead of the original weights, we use a partial binary representation w_i^j of w_i for each $i \in V$ given by $w_i^j = \lfloor w_i / 2^{J+1-j} \rfloor$ for $j = 1, ..., J, J+1$, where J is an integer such that

$J+1 > log(max\{w_i : i \epsilon V\}) \geq J$. For the first scaled problem, we start with the basis $B_o = I$, i.e., $(x, s) = (0, b)$. For the $(j+1)st$ problem, we use the optimal basis which we obtained for the jth problem, $j = 1, ..., J$. Since scaling does not affect the constraint set (4) - (6), primal feasibility is maintained in going from one set of weights to another.

To show the improvement of successive solutions given by the algorithm, in [10] we introduced a measure called *deficiency*, which is a function of the dual solution (y, z) that is related to, but not equal to, the sum of the dual infeasibilities. The deficiency function starts at $2M$, and it reaches zero if and only if (y, z) is feasible for (TP). In the polynomial version of DTRAN, the deficiency function is non-increasing and decreases to zero in a polynomial number of iterations.

4 Efficient Data Structures

Efficient data structures for representing networks have been the key to the successful implementation of the network simplex method. Basic simplex operations such as pricing and ratio tests are done graphically by the simplified network representation. Network data structures for representing the spanning tree have been called the predecessor and transversal method [3], or the augmented predecessor index method [7]. These data structures are summarized and compared in Kennington and Helgason [11]. We adopt one of the standard triple index methods.

In the basic DTRAN implementation, we need six vertex-length arrays. Three indices, the *predecessor* (the immediate ancestor in the tree), the *thread* (the preorder transversal) and the *depth* (the number of edges on the path from the super vertex), are used to represent the spanning tree $T(F, R)$. The other three arrays are used for storing primal and dual values and as a pointer to the edge-length arrays.

Fundamental operations of the DTRAN algorithm can be done by using the basic three indices. For example, to evaluate (11), we traverse all the vertices in the branch B_e for some $e \epsilon E(F)$. Such an operation can be done efficiently by using the thread and the depth. For the example in Figure 2.1, we give the three basic indices in Table 4.1.

vertex number $[i : i \epsilon V]$	1	2	3	4	5	6	7	8	9
predecessor	9	1	2	2	9	5	5	9	0
thread	2	3	4	5	6	7	8	9	1
depth	1	2	3	3	1	2	2	1	0

Table 4.1

To reduce computation time further, additional vertex-length arrays may be used to store the color of each vertex or a flag to indicate whether each vertex is in the branch B_e for a given leaving edge e. We found that these additional indices reduced computation

time by about 10 percent, even though extra time was spent to update them. Note that these arrays are not necessarily required, since one could get them from the basic three indices. In the non-polynomial, non-scaled version of DTRAN, we used eight vertex-length arrays. For a non-polynomial, but scaled version of DTRAN, an additional vertex-length array was used to store the temporarily adjusted dual values. For the polynomial version, another vertex-length array was used for the reverse order thread to find the leaving edge in Step 1.

Four edge-length arrays are used to store the original input data of (TP). They are from-vertex, to-vertex, the cost and a pointer to indicate the edges with the same to- (or from-) vertex. Contents of these arrays do not change during the execution of the algorithm.

5 Computational Experience with Scaling

First, we show the effects of scaling on DTRAN's computational times. We compared two algorithms: one with the original weights and the other with scaled weights. In both algorithms, we used the identical pricing and ratio test schemes. The pricing criterion used in Step 1 is based on the branch selection rule described in Section 2, which guarantees that the number of iterations is bounded by a polynomial in the number of sources, sinks and the sum of supplies and demands. In particular, when scaling is incorporated, it terminates in a polynomial number of steps. The ratio test in Step 2 is done by (11) with ties broken arbitrarily.

Test problems are randomly generated by the NETGEN program developed in [12]. Characteristics of the generated problems in the first group are given in Table 5.1. For each type (A-1 to A-3), we generated ten distinct problems.

Problem Type	Number of Sources	Number of Sinks	Number of Edges	Total Supply (M)	Cost Range (b_e)
A-1	100	100	2,000	5,000	1 - 100
A-2	200	200	4,000	10,000	1 - 100
A-3	400	400	6,000	10,000	1 - 100

Table 5.1

Results for the two algorithms are summarized in Table 5.2, where the computational times are obtained by averaging the ten results. The price time is the average elapsed time for Step 1, and the ratio time is the average time for Step 2. For the total time, the range (minimum and maximum) and the mean are given. The CPU times are in seconds on

the National Advanced Semiconductor 9050, which has the capability of 9 MIPS (roughly equivalent to the IBM 3081).

DTRAN without scaling							
Problem Type	price time	ratio time	total time			total pivots	time per pivot
			min.	max.	mean		
A-1	0.11	0.25	0.31	0.46	0.38	334	0.0011
A-2	0.40	0.97	1.24	1.89	1.48	666	0.0022
A-3	1.47	2.72	4.04	5.14	4.68	1334	0.0035

DTRAN with scaling							
Problem Type	price time	ratio time	total time			total pivots	time per pivot
			min.	max.	mean		
A-1	0.09	0.12	0.18	0.30	0.23	342	0.0007
A-2	0.31	0.52	0.70	1.27	0.95	729	0.0013
A-3	1.05	1.47	2.00	3.93	2.81	1351	0.0021

Table 5.2

The reduction in total computation time obtained by scaling is significant and it ranges from thirty to fifty percent in the examples. The reduction is achieved by decreasing the time per pivot, since the number of pivots is slightly larger when scaling is used. In particular, the time spent in Step 2 (ratio time) is much less in the scaled version. As can be seen in (11), the ratio time is roughly proportional to the size of the branch B_e selected in Step 1. Thus, if we select small branches and if we maintain a forest that is a collection of many small trees, we should be able to reduce the computation time in Step 2. The polynomial pricing scheme is designed to choose small branches. The scaling technique, for reasons that we are not able to explain theoretically, yields small shallow trees. Typical behavior for an A-2 problem is shown in Figure 5.3. On the left, the average depth of the vertices versus the pivot number is plotted, and the graph on the right shows the number of trees versus the pivot number. In the scaled version, the average depth grows more slowly, and the number of trees decreases more slowly than it does in the non-scaled version.

We also implemented two versions of DTRAN that use heuristic pricing schemes. The

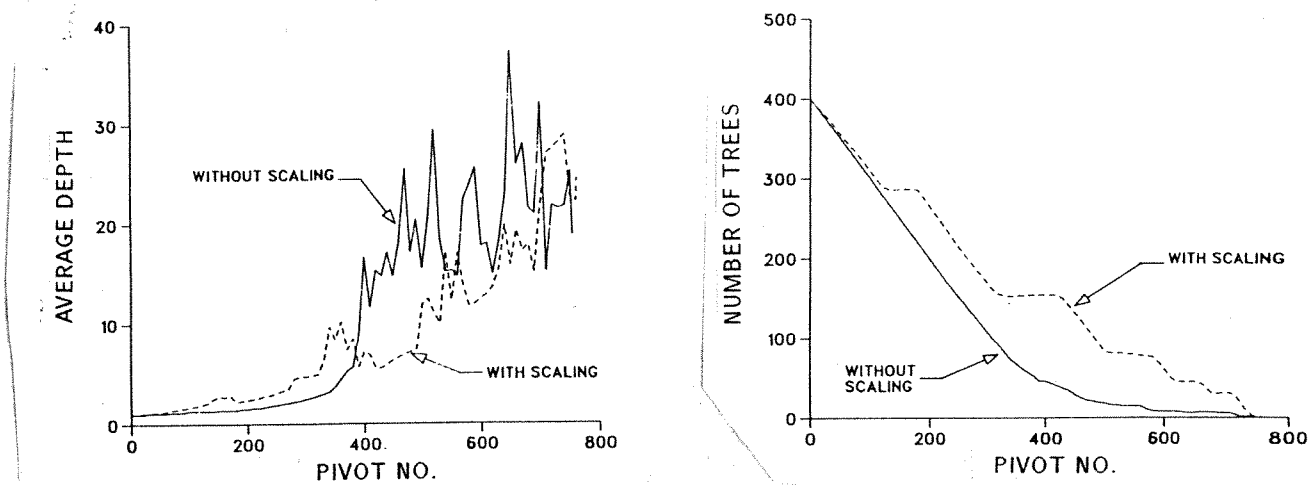


Figure 5.3

first employs a variation of the common most negative price rule. In particular, the pricing scheme in Step 1 is to find the most negative y_e (or positive z_r) in a tree T_r. The selection of T_r is arbitrary, although no tree is taken twice consecutively unless there is no choice. The results are summarized in Table 5.4.

DTRAN with scaling and heuristic pricing							
Problem Type	price time	ratio time	total time			total pivots	time per pivot
			min.	max.	mean		
A-1	0.05	0.18	0.23	0.34	0.28	370	0.0007
A-2	0.26	0.97	1.11	1.61	1.34	850	0.0016
A-3	0.97	3.54	3.53	7.50	5.33	1728	0.0030

Table 5.4

The performance of this heuristic pricing scheme is much poorer than the polynomial algorithm's scheme. Although the price time is less, the ratio time is much larger and the total number of pivots increases substantially.

The second heuristic pricing scheme we tested employs a variation of the branch selection

rule described in Section 2. It is also designed to yield small branches, but to select them much more quickly than the polynomial-time version. Although its performance is much better than the first heuristic, it is not quite as good as the polynomial time scheme. For example, the mean of the total times for A-3 problems is about 10% greater than the mean obtained with the polynomial time version.

Perhaps, a more elaborate variation of this heuristic would compete with or be superior to the polynomial algorithm's pricing scheme. Nevertheless, we must conclude that in DTRAN it is worthwhile to use a sophisticated pricing routine in order to select a proper edge to leave the spanning tree.

6 Comparison with a Primal Network Simplex Method

In this section, we present a brief computational comparison of the polynomial DTRAN algorithm with a primal network simplex method. We selected the NETFLO program [11] because it is a readily available state-of-the-art code for the primal network simplex algorithm. Two groups of test problems, one dense and the other sparse, were used to compare the performances of DTRAN and NETFLO. Density is the ratio of the actual number of edges in the graph to the total number of possible edges.

Problem ID	Number of Sources	Number of Sinks	Number of Edges	Total supplies	Cost Range (b_i)
D-1	50	200	8,000	5,000	1 - 1000
D-2	100	200	16,000	10,000	1 - 1000
D-3	150	200	24,000	20,000	1 - 1000

Table 6.1

The characteristics of the dense problems (80% density) are summarized in Table 6.1. For each problem type, we generated ten random problems. Table 6.2 presents the results of the DTRAN and NETFLO programs, including the standard deviations (s.d.) of the ten computation times. For the small problems, the polynomial DTRAN is faster than NETFLO; however, for the large problems of D-3, NETFLO is about thirty percent faster. Note that the DTRAN results have wider variations in the total computation times. In general, NETFLO requires many more pivots than DTRAN does; however, the time per pivot is much less in NETFLO. The percentage of degenerate pivots is greater for NETFLO but the difference decreases as the problems get larger.

The characteristics of the sparse problems (1% density) are summarized in Table 6.3. Again, we generated ten different problems for each group. The results for the sparse

Polynomial DTRAN							
Problem Type	total time				total pivots	degen. pivots	time per pivot
	min.	max.	mean	s.d.			
D-1	0.46	0.79	0.61	0.12	356	22	0.0017
D-2	1.76	3.30	2.07	0.45	525	48	0.0039
D-3	3.51	5.65	4.64	0.71	688	71	0.0065

NETFLO							
Problem Type	total time				total pivots	degen. pivots	time per pivot
	min.	max.	mean	s.d.			
D-1	0.72	0.93	0.83	0.07	1082	344	0.0008
D-2	1.92	2.26	2.08	0.14	1785	415	0.0011
D-3	3.13	3.53	3.32	0.15	2163	208	0.0015

Table 6.2

problems are shown in Table 6.4. DTRAN performs relatively worse for sparse problems. Although DTRAN is competititve with NETFLO for the small problems, it is slower for the larger problems by forty to sixty percent. In both DTRAN and NETFLO, sparse problems require considerably more pivots than dense problems. In both programs, when the number of edges is fixed, the increase in the number of pivots from a dense to a sparse problem is nearly proportional to the increase in the number of vertices. In NETFLO, the percentage of degenerate pivots becomes much higher for the sparse problems, whereas in DTRAN that percentage becomes slightly less. However, the time per pivot is the significant factor in the total computation times of NETFLO and DTRAN. While the time per pivot remains the same in NETFLO for dense and sparse problems with the same number of edges (e.g., D-2 versus S-3), it increases substantially in DTRAN for the sparse problems.

This computation study supports the belief that primal simplex codes are superior to dual ones for large transportation problems. Nevertheless, within the size range of problems tested, DTRAN is at least as good for the smaller problems and not more than 60% slower on the large problems. As such, DTRAN should be useful in applications where dual methods are desirable for other reasons. These include sensitivity analysis and using transportation algorithms as subroutines in the solution of more complex problems with

Problem ID	Number of Sources	Number of Sinks	Number of Edges	Total supplies	Cost Range (b_e)
S-1	500	1000	5,000	10,000	1 - 1000
S-2	1000	1000	10,000	20,000	1 - 1000
S-3	1500	1000	15,000	30,000	1 - 1000

Table 6.3

embedded network structure.

Polynomial DTRAN							
Problem Type	total time				total pivots	degen. pivots	time per pivot
	min.	max.	mean	s.d.			
S-1	4.55	5.62	4.89	0.31	2104	72	0.0023
S-2	13.22	16.44	14.61	0.94	3425	167	0.0043
S-3	23.31	28.78	26.72	1.73	4100	349	0.0065

NETFLO							
Problem Type	total time				total pivots	degen. pivots	time per pivot
	min.	max.	mean	s.d.			
S-1	4.27	4.89	4.52	0.18	7847	5365	0.0006
S-2	9.78	10.72	10.32	0.28	12408	7645	0.0008
S-3	15.87	17.83	16.69	0.53	15613	8866	0.0011

Table 6.4

REFERENCES

1. R. D. Armstrong, D. Klingman and D. Whitman, "Implementation and Analysis of a Variant of the Dual Method for the Capacitated Transshipment Problem," *European Journal of Operational Research*, 4, 403 - 420, 1980.

2. M. L. Balinski, "Signature Methods for the Assignment Problem," Report No. D253, Laboratoire d'Econometrie de l'Ecole Polytechnique, Paris, November 1982.

3. G. H. Bradley, G. G. Brown and G. W. Graves, "Design and Implementation of Large Scale Primal Transshipment Algorithms," *Management Science*, 24, 1 - 34, 1977.

4. R. Bland and D. Jensen, Private communications, 1984.

5. J. Edmonds and R. M. Karp, "Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems," *Journal of the ACM*, 19, 248 - 264, 1972.

6. F. Glover, D. Karney, D. Klingman and A. Napier, "A Computational Study on Start Procedures, Basis Change Criteria, and Solution Algorithms for Transportation

Problems," *Management Science*, 20, 793-813, January 1974.

7. F. Glover, D. Karney and D. Klingman, "Implementation and Computational Comparisons of Primal, Dual and Primal-dual Computer Codes for Minimum Cost Network Flow Problems," *Networks*, 4, 191-212, 1974.

8. M. D. Grigoriadis, "Network Optimization – Techniques and Applications," Tutorial presented at the ORSA/TIMS meeting in San Diego, November 1982.

9. M. S. Hung and C. Murphy, "Implementation of Edmonds and Karp's "Scaling" Algorithm," presented at the ORSA/TIMS meeting in San Francisco, May 1984.

10. Y. Ikura and G. L. Nemhauser, "A Polynomial-time Dual Simplex Algorithm for the Transportation Problem," SOR&IE Technical Report No. 602, Cornell University, September 1983.

11. J. L. Kennington and R. V. Helgason, *Algorithms for Network Programming,* John Wiley and Sons, New York, 1980.

12. D. Klingman, A. Napier and J. Stutz, "NETGEN: A Program for Generating Large Scale Capacitated Assignment, Transportation, and Minimum Cost Flow Network Problems," *Management Science*, 20, 814-821, 1974

13. E. L. Lawer, *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart and Winston, New York, 1976.

14. C. L. Monma and M. Segal, "A Primal Algorithm for Finding Minimum-cost Flows in Capacitated Networks with Applications," *The Bell Technical Journal*. 61, 949-968, 1982.

15. J. Orlin, "A Polynomial Parametric Simplex Algorithm for the Minimum Cost Network Flow Problem," Sloan Working Paper No. 1484-83, MIT, September 1983.

16. S. R. Schmidt, P. A. Jensen and J. W. Barnes, "An Advanced Dual Incremental Network Algorithm," *Networks*, 12, 475 - 492, 1982.