August 1991

Report No. STAN-CS-91-1374

Polynomial Dual Network Simplex Algorithms

by

J. Orlin, S. Plotkin, E. Tardos

Department of Computer Science

Stanford University
Stanford, California 94305

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)

2. REPORT DATE

3. REPORT TYPE AND DATES COVERED

4. TITLE AND SUBTITLE

Polynomial Dual Network Simplex Algorithms

5. FUNDING NUMBERS

6. AUTHOR(S)

James B. Orlin, Serge A. Plotkin, Éva Tardos

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Sloan School of Management Science, MIT
Dept. of Computer Science, Stanford University
School of Operations Research, Cornell University

8. PERFORMING ORGANIZATION REPORT NUMBER

STAN-CS-91-1374

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

NSF Research Initiation Award CCR-900-8226 and
ONR Contract N00014-88-K-0166

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT

unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

We show how to use polynomial and strongly polynomial capacity scaling algorithms for the transshipment problem to design a polynomial dual network simplex pivot rule. Our best pivoting strategy leads to an $O(m^2 \log n)$ bound on the number of pivots, where n and m denotes the number of nodes and arcs in the input network. If the demands are integral and at most B, we also give an $O(m(m + n \log n) \min(\log nB, m \log n))$-time implementation of a strategy that requires somewhat more pivots.

14. SUBJECT TERMS

analysis of algorithms, mathematical theory of computation, combinatorial mathematics

15. NUMBER OF PAGES

23

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT

18. SECURITY CLASSIFICATION OF THIS PAGE

19. SECURITY CLASSIFICATION OF ABSTRACT

20. LIMITATION OF ABSTRACT

Polynomial Dual Network Simplex Algorithms

*James B. Orlin**
Sloan School of Management Science
MIT

Serge A. Plotkin†
Department of Computer Science,
Stanford University

Éva Tardos‡
School of Operations Research,
Cornell University

April 1991

Abstract

We show how to use polynomial and strongly polynomial capacity scaling algorithms for the transshipment problem to design a polynomial dual network simplex pivot rule. Our best pivoting strategy leads to an $O(m^2 \log n)$ bound on the number of pivots, where n and m denotes the number of nodes and arcs in the input network. If the demands are integral and at most B, we also give an $O(m(m + n \log n) \min(\log nB, m \log n))$-time implementation of a strategy that requires somewhat more pivots.

*Research supported by AFOSR-88-0088 through the Air Force Office of Scientific Research, by NSF grant DOM-8921835 and by grants from Prime Computer Corporation and UPS.

†Research supported by NSF Research Initiation Award CCR-900-8226 and by ONR Contract N00014-88-K-0166.

‡Research supported in part a David and Lucile Packard Fellowship and by the National Science Foundation, the Air Force Office of Scientific Research, and the Office of Naval Research, through NSF grant DMS-8920550.

1 Introduction

The transshipment problem is one of the central problems in network optimization. There exist several polynomial and strongly polynomial algorithms for solving this problem (see the surveys [7, 1]). Nevertheless, the method of choice in practice still seems to be the network simplex method.

In this paper we shall consider the dual network simplex method. We show that some excess scaling algorithms for the uncapacitated transshipment problem can be used to guide the pivot selection of the dual network simplex algorithm for both the capacitated and the uncapacitated transshipment problems. The resulting simplex algorithm can be viewed as a special implementation of the capacity scaling algorithm. This implement ation maintains the property that all arcs with non-zero flow are in a. tree, a property that seems to help the practical performance.

We give a simple pivoting strategy that, leads to an 0($n^2 \log nB$) bound on the number of pivots for the uncapa.citated transshipment problem, assuming that the demands are integral, and at most B. We also show how to modify this strategy to achieve a strongly polynomial $O(n^3 \log n)$-pivot algorithm. We describe a more complicated strategy that leads to an $O(nm \log n)$ bound on the number of pivots. The first two pivoting strategies can be implemented using Fibonacci Heaps to run in 0($n(m + n \log n) \log nB$) and $O(n^2(m + n \log n) \log n)$ time respectively. Dual network simplex algorithms for the uncapacitated transshipment problem can also be used to solve the capacitated version of the problem. Bounds for the resulting algorithms can be obtained by substituting $O(m)$ for n.

Earlier versions of this paper have appeared as technical reports and in conference proceedings. The technical report of Orlin [9] described the first polynomial and strongly polynomial dual network simplex pivoting strategies. These strategies were based on capacity scaling algorithms. He also gave a fast implementation of the resulting $O(n^3 \log n)$-pivot dual network simplex algorithm. The extended abstract of Plotkin and Tarclos [12] presented an improved $O(nm$ log n) pivoting strategy. The strongly polynomial simplex algorithm described by Orlin in [9] assumed a model of computation, in which we are allowed to use some other operations in addition to the usual arithmetic operations (addition, multiplication, and comparison). The algorithms presented in this paper use additions and comparisons only.

The dual network simplex algorithms described in this paper are based on polynomial and strongly polynomial excess scaling algorithms. The scaling algorithms work in iterations, where each iteration executes an augmentation between a pair of nodes. The simplex implementation maintains a tree 7 that contains all arcs with non-zero flow values. The tree is changed using simplex pivot steps, and all augmentations are done through the arcs which are in the current tree. The pivoting strategy that results in the lowest number of pivots is based on Orlin's [11] strongly polynomial transshipment algorithm. A direct translation of this algorithm into the simplex framework is infeasible. For some pairs of nodes that the algorithm might choose for augmentation there might be no sequence of pivot steps that make an augmentation between these two nodes possible in the tree. The version of the transshipment algorithm used here allows greater freedom in the choice of the augmentation done at each iteration, and we show that a simple pivoting strategy can be viewed as implementing one of the possible choices. In effect, we let the simplex algorithm choose among the possible augmentations.

A related important open problem is whether there exists a primal simplex pivoting strategy for the transshipment problem that leads to a polynomial bound on the number of pivots. Some special cases of the transshipment problem are known to be solvable by polynomial versions of the primal simplex method. There are polynomial and strongly polynomial primal simplex algorithms known for the single source shortest path problem and the assignment problem (see [3, 10]). Recently, Goldfarb and Hao [S] gave a pivoting strategy for the primal network simplex method that solves the maximum flow problem in $O(mn)$ pivots (see also [G]). Tarjan [13] developed the first subexponential primal simplex algorithm for minimum-cost flow problem. The pivoting strategy used in Tarjan's algorithm is guided by a polynomial cost scaling algorithm.

The paper consists of four sections. Section 2 reviews the terminology and the dual network simplex framework. Section 3 presents our simplest pivoting strategy and describes a modification of this strategy that leads to a strongly polynomial bound on the number of pivots. Section 4 presents an improved strategy that leads to a better strongly polynomial bound on the number of pivots.

2 Preliminaries

In this section we define the transshipment problem, review some fundamental facts about it, and review the dual net work simplex framework. A *network* is a directed graph $G = (V, E)$. We shall use n and m to denote the number of nodes and the number of arcs in this graph, respectively. To simplify the bounds, we assume that $m \geq n$. For notational convenience, we assume that G has no parallel or opposite arcs. If there is an arc from a node v to a node w, this arc is unique by the assumption, and we denote it by (v, w).

The input to the *transshipment problem* consists of a network $G = (V, E)$, a *capacity function* $u : E \rightarrow \mathbf{R}^+ \cup \{\infty\}$, a *demand function* $b : V \rightarrow \mathbf{R}$ such that $\sum_{v \in V} b(v) = 0$, and a *cost function* $c : E \rightarrow \mathbf{R}$. We shall use the notation $c(v, w) = -c(w, v)$ for $v, w \in V$ such that $(w, v) \in E$. In the special case of integer demands, we shall use B to denote the maximum absolute value of a. demand.

A *pseudoflow* is a. function $f : E \rightarrow \mathbf{R}^+$, such that $f(v, w) \leq u(v, w)$ for every $(v, w) \in E$. Given a pseudoflow f, we define the *excess* function $e_f : V \rightarrow \mathbf{R}$ by

$$e_f(v) = \sum_{w:(w,v) \in E} f(w, v) - \sum_{w:(v,w) \in E} f(v, w) - b(v),$$

the amount by which the net flow into v exceeds the demand. We say that a node v has *excess* if $e_f(v)$ is positive, and has *deficit* if it is negative. For a subset S of the nodes we shall use $e_f(S)$ to denote $\sum_{v \in S} e_f(v)$. For a node v, we clefine the *flow conservation constraint* by

$$e_f(v) = 0 \quad \text{(flow conservation constraint).} \tag{1}$$

A pseudoflow f is a *transsshipment* if it satisfies the flow conservation constraints at every node.

The *residual graph* with respect to a pseudoflow f is given by $G_f = (V, E_f)$, where $E_f = \{(v, w) | f(v, w) < u(v, w) \text{ or } f(w, v) > 0\}$.

The *cost* of a pseudoflow f is given by

$$c(f) = \sum_{(v,w) \in E} f(v, w) c(v, w).$$

The *transshipment* problem is that of finding a minimum-cost (*optimal*) transshipment in an input network $(G; u, b, c)$. In the *uncapacitated transshipment* problem all capacities are equal to ∞.

In order to simplify the presentation we restrict our attention to the uncapacitated transshipment problem and assume that the graph G is strongly connected. The adaptation of the presented results to the general transshipment problem is straightforward. Given an instance of the transshipment problem with capacities, we can construct an equivalent uncapacitated instance by introducing a new node in the middle of every arc. The dual simplex method applied to the resulting uncapacitated problem is the same as the dual network simplex algorithm for the original problem. This construction, however, increases the number of nodes to $O(m$ + ?a). The assumption that the graph G is strongly connected can be satisfied by introducing at most 272 additional expensive arcs (s, v) and (v, s) for some node s and every v. If the original problem is feasible then no optimal solution uses the additional arcs. This assumption guarantees that the transshipment problem is feasible and implies the following characterization of the existence of an optimal solution.

Theorem 2.1 There exists a minimum-cost transshipment if and only if the input network contains no negative-cost cycles.

Linear programming duality theory provides a criterion for the optimality of a transshipment. To state the criterion we need the notions of a price function and a reduced cost function. A *price function* is a node labelling $p: V \rightarrow$ R. The *reduced cost function* with respect to a price function p is defined by $c_p(v, w) = c(v, w) + p(v) - p(w)$. For the uncapacitated transshipment problem a *feasible dual solution* is a set of prices such that each arc in E has non-negative reduced cost.

Theorem 2.2 [5] A flow f is an optimal solution for the uncapacitated transshipment problem if and only if there is a price function p such that, for each arc $(v, w) \in E$,

$$\begin{array}{ll} c_p(v, w) \geq 0 & \text{(dual feasibility constraints) and} \\ c_p(v, w) > 0 \Rightarrow f(v, w) = 0 & \text{(complementarity slackness conditions).} \end{array} \tag{2}$$

procedure PIVOT($\mathcal{T}$, (v', w'));

We assume that:
7 is a dual feasible tree;
p is the price function that corresponds to 7;
(v', w') an arc such that $f^{\mathcal{T}}(v', w') < 0$;

Let $(w, v) \in E$ be the arc with minimum reduced cost δ leaving $H_{\mathcal{T},(v',w')}$;
Replace the arc (v', w') in 7 by (v, w);
Decrease p on all nodes in $H_{\mathcal{T},(v',w')}$ by δ;
end.

Figure 1: A Pivot Step of the Dual Network Simplex Method.

A basis of the linear program corresponding to the uncapacitated transshipment problem is the set of columns corresponding to the arcs in a. spanning tree 7 of G. A transshipment is a basic primal solution if and only if the arcs with non-zero flow form a forest. A set of dual feasible prices is a basic dual solution if and only if the arcs with zero reduced cost form a spanning subgraph.

Let 7 be a spanning tree of the underlying undirected graph. Consider the *cut* obtained by deleting an arc $(v, w) \in 7$ from 7. Let $H_{\mathcal{T},(v,w)}$ denote the set of nodes that are on the same side of this cut as w, the head of the arc (v, w).

For a tree 7 let $f^{\mathcal{T}}$ denote the corresponding basic flow, that is,

$$f^{\mathcal{T}}(v, w) = \begin{cases} 0 & \forall (v, w) \notin \mathcal{T} \\ \sum_{v' \in H_{\mathcal{T},(v,w)}} b(v') & \forall (v, w) \in \mathcal{T} \end{cases} \tag{3}$$

Similarly, we let $p^{\mathcal{T}}$ denote the prices defined by the spanning tree 7, *i.e.*, prices such that the reduced costs of the arcs in the tree 7 with respect to prices $p^{\mathcal{T}}$ are 0.

Let $p = p^{\mathcal{T}}$ denote the prices defined by the tree 7. The spanning tree 7 is *dual feasible* if $c_p(v, w) \geq 0$ for all arcs $(v, w) \in E$. A spanning tree 7 is called primal feasible if $f^{\mathcal{T}} \geq 0$. A spanning tree is *optimal* if it is both primal feasible and dual feasible. Theorem 2.2 implies that in this case the defined flow $f^{\mathcal{T}}$ is an optimal transshipment.

The *dual network simplex* algorithm (see Figure 1) maintains a dual feasible tree 7. An initial dual feasible tree can be found by a shortest path computation. In the case that all arcs are non-negative, one can find the initial tree $\mathcal{T}^0$ in $O(n)$ pivots [4]; in case the costs may be negative, one can find $\mathcal{T}^0$ in $O(n^2)$ pivots using the dual simplex algorithm of Balinski [2]. A pivot step of the dual network simplex algorithm can be applied to an arc (v', w') in 7 with $f^{\mathcal{T}}(v', w') < 0$. The pivot step changes 7 by deleting the arc (v', w') and replacing it by one of the arcs (v, w) of minimum reduced cost leaving the set $H_{\mathcal{T},(v',w')}$. The arc (v', w') is the *leaving* arc, and (v, w) is the *entering* arc.

Let $\mathcal{T}' = 7 - (v', w') + (v, w)$ be the tree obtained after the pivot. It is easy to show that 7' is dual feasible. Moreover, the dual objective function $\sum_{v \in V} p(v)b(v)$ has not decreased after the pivot. Let δ be the reduced cost of the entering arc (v, w). We may obtain the price vector $p^{\mathcal{T}'}$ from the previous price vector $p^{\mathcal{T}}$ by subtracting δ from the price of each node in $H_{\mathcal{T},(v',w')}$ and keeping all other prices the same.

The dual simplex algorithm iterates this basic pivot step until the current tree becomes primal feasible. Pivoting strategies give rules for choosing among the possible leaving and entering arcs.

3 A simple polynomial time dual simplex algorithm

In this section we describe how to use an excess scaling algorithm to guide the selection of dual pivots. In the first subsection we present a. basic subroutine consisting of a sequence of at most n pivot steps that makes the nest augmentation possible. Next we show how to use a version of the Edmonds-Karp capacity scaling algorithm to derive a polynomial dual network simplex algorithm that makes at most $O(n^2 \log nB)$ pivots. Finally we give a strongly polynomial version with an $O(n^3 \log n)$ bound on the number of pivots. The algorithm is a simplification of the polynomial time dual simplex algorithm presented in [9]. Also, as opposed to the algorithm presented in [9], the strongly polynomial algorithm described in this section uses only the usual arithmetic operations (in fact, only additions and comparisons).

We shall use r to denote the special root node. We shall use 7 to denote a. tree rooted at r, and use $\mathcal{T}_v$ to denote the subtree of 7 routed at node v. We use $pred(v)$ to denote v's parent in the tree. An arc (v, w) $\in$ 7 is called *downward* if v is the parent of w in the tree (*i.e.* (v, w) points away from the root node), otherwise (v, w) is called *upward.*

The algorithm will maintain a pair (7, f) of a tree and a pseudoflow satisfying the following conditions :

P1. 7 is a dual feasible tree.

P2. $f(v, w) \geq 0$ for $(v, w) \in$ 7.

P3. $f(v, w) = 0$ for $(v, w) \notin$ 7.

P4. $e_f(v) > 0$ for each $v \neq r$.

Recall that we have defined $f^{\mathcal{T}}$ as the basic flow corresponding to tree 7.

Lemma 3.1 If $(v$, w) is a downward arc of 7, then $f^{\mathcal{T}}(v$, w) $= f(v, w) - e_f(\mathcal{T}_w)$. If (v, w) is an upward arc of 7, then $f^{\mathcal{T}}(v, w) = f(v, w) + e_f(\mathcal{T}_v)$.

```
procedure MAKE-GOOD(T, f);

    We assume that:
        T is a dual feasible tree;

    Let p be the price function that corresponds to T;
    while T has bad arcs do begin
        let S denote the set of bad nodes for T w.r.t. f;
        let (v, w) be the minimum reduced cost arc leaving S;
        let δ ← c_p(v, w);
        let (v', w') be the first bad arc on the path from r to v;
        let T ← T + (v, w) − (v', w');
        Decrease p on all nodes in H_{T,(v',w')} by δ;
    end;
end.
```

Figure 2: Procedure **MAKE-GOOD**.

Proof: One can obtain the flow $f^{\mathcal{T}}$ from the flow f by sending $e_f(v)$ units of flow from v to r for each node $v \neq r$. The increase of the flow in an upward arc (v, w) is $e_f(T_w)$. The decrease of the flow in a downward arc (v, w) is $e_f(T_w)$. ∎

Suppose that the pair $(f, \mathcal{T})$ satisfies P1-P4. An arc (v, w) in $\mathcal{T}$ is called *bad* if it is downward and its flow is 0. Otherwise, it is called good. A node $v \in \mathcal{T}$ is called *good* if every arc on the path from r to v is good. Otherwise, node v is called *bad*.

Corollary 3.2 Suppose that $(f, \mathcal{T})$ satisfies P1-P4. Then every bad arc is an eligible exiting arc.

3.1 The procedure MAKE-GOOD

We will now describe a pivoting procedure **MAKE-GOOD** (see Figure 2) for transforming a tree $\mathcal{T}$ satisfying P1-P4 that has some bad arcs, into a tree in which all arcs axe good. Given a tree $\mathcal{T}$ and flow f, MAKE-GOOD proceeds in iterations. In each iteration it considers the set S of *bad* nodes and finds a minimum reduced cost arc (v, w) leaving this set and an arc (v', w') that is the first bad arc on the path from r to v. Then it removes (v, w) from the tree and adds (v', w') instead. **MAKE-GOOD** terminates when there are no more bad nodes.

Since (v', w') is a. bad arc, Corollary 3.2 implies that it is an eligible arc for exiting. In general, adding the arc (v, w) instead of (v', w') does not maintain dual feasibility. However, MAKE-GOOD is called only under certain conditions that ensure that it executes only legal pivots.

MAKE-GOOD is called by the simplex algorithms described in the subsequent sections after a flow augmentation is done from some node to the root r, if this augmentation created a bad arc by reducing to zero the flow on one of the downward arcs. MAKE-GOOD transforms the tree into a new tree where there are no bad arcs, and hence we can do an augmentation from any node with positive excess to the root.

Let 7 be a tree and let f denote a flow satisfying P1-P4. We call a path P in 7 from some node t to the root r an *inverse-good* path if all upward arcs on P have positive flow. Notice that if we were to reverse the direction of each arc on an inverse-good path then the resulting path would consist of good arcs. Observe that if a tree had no bad arcs, an augmentation from some node to the root can introduce bad arcs only on the path used by the augmentation and hence all bad arcs will lie on an inverse-good path.

Theorem 3.3 Suppose that the pair $(\mathcal{T}, f)$ satisfies Pl-P4 and all of the bad arcs of the tree $\mathcal{T}$ lie on some inverse-good path P from some node t to the root r. Then the steps of the procedure MAKE-GOOD$(\mathcal{T}, f)$ are dual simplex pivots.

Proof: Let S denote the set of bad nodes of 7. Let (v, w) denote the arc that is pivoted in and let (v', w') denote the arc that is pivoted out. We first claim that S is the set of descendants of node w'. By definition, all descendants of w' axe bad. To see the converse, suppose that s is a bad node. By the choice of $(v'$, w'), all arcs on the path from r to v' are good. Let P' be the path from s to r and let (v'', w'') denote a bad arc of P'. By hypothesis (v'', w'') is also on path P, and thus s is a descendent of w'.

We now claim that the first pivot of procedure MAKE-GOOD is a dual simplex pivot. By Corollary 3.2 the arc $(v'$, w') is an eligible exiting arc. For the pivot to be a dual simplex pivot, the entering variable must be the least cost arc from the subtree $\mathcal{T}_{w'}$ to the rest of the graph. This is implied by the choice of (v, w), and the fact that the set of bad nodes of 7 is the set of nodes of $\mathcal{T}_{w'}$.

Let 7' be the tree obtained after a pivot from tree 7. We will show that all of the bad arcs of 7' lie on an inverse good path from t to r in 7'. The theorem then follows by induction.

Node v is a bad node of 7, and thus the path from v to r intersects the path P at some node, say node s. Since all bad arcs lie on P, the path P' from v to s is good. Let P_1 denote the subpath of P from s to w', and let P_2 denote the subpath of P from t to s. Observe that each arc of 7 \ P_1 \ P' – (v', w') has the same direction (upward or clownward) in 7' as it does in 7. Each arc of P_1 and P' has the opposite direction in 7' as it does in 7. It follows that P_1 is transformed by the pivot from an inverse-good path into a good path, and that P' is transformed from a good path to an inverse-good path. Subsequently, all bad arcs of 7' lie on P_2 or on P', and thus on the inverse good path P_2, P' from t to r in 7'. ∎

The following theorem bounds the number of pivots that can be made by the procedure **MAKE-GOOD.**

Theorem 3.4 Suppose that the pair (7, f) satisfies P1-P4 and all of the bad arcs of the tree $\mathcal{T}$ lie on some inverse-good path P from some node t to the root r. The procedure **MAKE-GOOD** terminates with a good tree $\mathcal{T}$ in at most $n - 1$ pivots.

Proof: First we show that any good node in 7 is also a good node in 7'. Suppose that t is a good node of $\mathcal{T}$. Then the path P from t to r in $\mathcal{T}$ does not contain the bad arc (v', w') and hence P is also a path in 7'. It follows that t is good in $\mathcal{T}'$.

Now, we note that w is a good node in 7, and v is bad in 7, and thus v becomes good in 7'. We conclude that the number of good nodes in $\mathcal{T}'$ is greater than the number of good nodes in 7. ∎

Next we prove that this sequence of pivots can be performed in $O(m + n \log n)$ time using Fibonacci heaps.

Theorem 3.5 Suppose that tree $\mathcal{T}$ satisfies Pl-P4 and all of the bad arcs of 7 lie on some inverse-good path P from some node t to the root r. Then the **MAKE-GOOD** procedure can be implemented in $O(m + n \log n)$ time.

Proof: The proof is based on a. closer look at the proof of Theorems 3.4 and 3.3. The algorithm will maintain the first bad node w' on the path from r to t, and for every bad node v it maintains $d(v) = \min_{(w \text{ good})} c_p(v, w)$ and the good node w_v on which the minimum is attained.

Recall that the set of bad nodes is the set of nodes in $\mathcal{T}_{w'}$. The entering edge is (v, w_v) for a bad node v with d(v) minimum. The leaving edge is $(pred(w'), w')$. Nest we have to show how to update the above information. The new w' is the head of the first bad arc along the path from v to t. To find the new w' after the update takes time proportional to the number of nodes from v to the new w'. Notice that these are the nodes that become good during this pivot.

To update $d(v)$ for the remaining bad nodes we have to consider all edges leaving the nodes that became good during this pivot. We shall maintain $d(v)$ in a Fibonacci heap. Overall, there are at most m updates (one for every arc). The total time spent over all iterations of **MAKE-GOOD** on searching for nodes that become good and for the new w' is bounded by $O(n)$. The claim follows since updating and finding the minimum can be clone in $O(m + n \log n)$ time. I

3.2 A polynomial dual simplex algorithm.

We are now prepared to describe the polynomial time dual simplex algorithm. The algorithm finds an optimal solution for the minimum cost flow problem starting with the shortest path tree $\mathcal{T}^0$ directed from node 1'. In the case that all arcs are non-negative, one can find $\mathcal{T}^0$ in $O(n)$ pivots [4]; in case the costs may be negative, one can find $\mathcal{T}^0$ in $O(n^2)$ pivots using the dual simplex algorithm of Balinski [2]. We assume that $\mathcal{T}^0$ is obtained using an appropriate subroutine.

In our algorithm, we will keep track of a scaling parameter Δ that is nondecreasing from iteration to iteration. We will refer to the Δ-*scaling phase* as all iterations in which the param-

```
procedure SCALING-SIMPLEX(T);

    We assume that:
            7 is a dual feasible tree and a.11 arcs are good subject to the flow f = 0;

    Δ ← 2^⌈log(B+1)⌉
    Define f by sending A units of flow from r to every node v ≠ r in 7.
    while A > 1/(2n) do begin
        while S_f (A) ≠ ∅ do begin
            let v be a node in S_f (A);
            send A units of flow from v to r in 7.
            if there is a. bad arc in T
              then begin
                call MAKE-GOOD(T, f)
              end;
        end;
        A ← Δ/2
    end;
end.
```

Figure **3: SCALING-SIMPLEX** algorithm.

eter A has a fixed value. Throughout the algorithm, the tree 7 and the flow f will satisfy P5 in addition to P1-P4.

P5. All flows are multiples of A.

Let $S_f(\Delta)$ = $\{v : e_f(v) >$ A$\}$. The termination of the algorithm will be guaranteed by the following lemma.

Lemma 3.6 Suppose that the supplies and demands are integral and $\mathcal{T}$ is dual feasible. Suppose that f is a nonnegative flow such that

- $f(v, w) = 0$ for every $(v, w) \notin$ 7,
- $\square \leq e_f(v) \leq 1/n$ for every v $\neq r$.

Then 7 is primal feasible.

Proof: By Lemma 3.1, $f^{\mathcal{T}}(v, w) > f(v, w) - 1$. The flow $f^{\mathcal{T}}$ is integral, and f is non-negative. It follows that $f^{\mathcal{T}}$ is non-negative. ∎

Procedure **SCALING-SIMPLEX** is described in Figure 3. It starts with an initial dual-feasible tree $\mathcal{T}^0$ where all arcs are pointing away from the root, and a parameter A = $2^{\lceil \log(B+1) \rceil}$. The initial flow is constructed by sending A units of flow from the root to each one of the nodes through the edges of the tree. The procedure proceeds in phases. Each phase considers the set $S_f($ A j of nodes whose excess is above A, and iteratively augments a flow from one of the nodes in this set to the root by A. If such an augmentation creates a bad arc, the tree is updated by calling the procedure MAKE-GOOD described above. A phase ends when the set S,(A) becomes empty. Then A is halved and a new phase is started. **SCALING-SIMPLEX** terminates when A falls below $1/(2n)$.

Theorem 3.7 The **SCALING-SIMPLEX** algorithm, if started from a shortest path tree rooted at r, performs dual simplex pivots, and finds the optimal spanning tree for the uncapacitated transshipment problem after $O(n^2 \log nB)$ pivots, and in $O(n(m + n \log n) \log nB)$ time.

Proof: We claim that the algorithm maintains properties P1-P5. First note that all flows are multiples of A throughout the algorithm, and so P5 is satisfied. We now claim that all excesses are strictly positive. It is true initially, since initially A > B by definition. During the algorithm we send A units of flow only from nodes with excess more than A, and so all excesses remain positive after an augmentation. Therefore P4 is satisfied.

Since all arc flows are multiples of Δ, one can send A units of flow on any good path. For this reason, P2 and P3 are satisfied throughout. Initially $\mathcal{T}$ is a dual feasible tree with all arcs being good. After sending A units of flow on a good path P from some node v to r the path P becomes an inverse-good path. If P has any bad arcs, then the procedure MAKE-GOOD takes $O(n)$ pivots and 0($m + n$ log nj time by Theorems 3.4 and 3.5. Property P1 is satisfied throughout the execution of the procedure by Theorem 3.3.

SCALING-SIMPLEX terminates with A $\leq 1/(2n)$. At the end of the last scaling phase we obtain a tree $\mathcal{T}$ and a flow f. At this point, $e_f(v) \leq 1/n$ for each node $v \neq r$. Lemma, 3.6 implies that $\mathcal{T}$ is primal feasible, and hence 7 is an optimal basis.

During a scaling phase each node in v $\neq r$ starts with an excess of less than 2A and ends with a positive excess. Each augmentation reduces the excess of a node $v \neq r$ by A. Thus the number of augmentations per scaling phase is at most n. The algorithm terminates when A < $1/(2n)$, therefore the total number of augmentations is $O(n \log nB)$.

The time between successive augmentations is $O(m + n \log n)$ by Theorem 3.5, and the number of pivots is $O(n)$ by Theorem 3.4. We can conclude that the total number of pivots is $O(n^2 \log nB)$ and the total running time is $O(n(m + n \log n) \log nB)$. ∎

3.3 A strongly polynomial dual simplex algorithm.

We will modify the dual simplex algorithm described in the previous section to make it strongly polynomial. We will divide scaling phases into two types, according to whether a pivot was performed during the phase, and will bound the number of scaling phases that involve pivots by 0 $(n \log n)$. The second type of scaling phase does not involve pivots and hence the spanning tree at the beginning of the scaling phase is the same as the spanning tree at the end of the scaling phase. The number of such scaling phases for the **SCALING-SIMPLEX** algorithm is at least log nB. We will show how to modify the algorithm appropriately so that the number of these scaling phases is 0(n).

We first show that the number of scaling phases in which some pivot takes place is 0(*n* log n j. To prove this bound, we first observe that all flow changes in the algorithm are due to sending flow on a path to the root. Since the root node is the only node with a negative excess throughout the algorithm, the total flow change on any arc is bounded at $-e_f($ I'). Thus, we have the following lemma,.

Lemma 3.8 Suppose that $e_f(r)$ < 0 throughout the algorithm, and that each flow change is the result of sending flow from some node $v \neq r$ to node r. If $f(v, w)$ > $-e_f(r)$ for some flow f obtained in the algorithm, then $f'(v, w)$ > $-e'_f(r)$ for all subsequent flows f' obtained by the algorithm, including the optimum flow.

We will refer to an arc (v, w) as *strongly feasible* if $f(v,w)$ > $-e_f(r)$. The above lemma implies that once an arc becomes strongly feasible it will stay strongly feasible throughout the rest of the algorithm, and it will have a strictly positive flow in the optimum solution obtained by the algorithm.

The reason that we wanted arc flows to be multiples of A was so that we could be assured that we could send A units of flow on any good path. The above lemma implies that it is sufficient to weaken property *P5* and require that the flow will be multiple of A on arcs that are not strongly feasible. We will also need to strengthen property P4.

P'4. A/2 < $e_f(v) \leq$ 3A for each $v \neq r$.

P'5. For each (v, w) for which $f(v, w) \leq -e_f(r)$, $f(v, w)$ is an integral multiple of A.

For convenience, we will write P'1-P'5 to mean P1, P2, P3, P'4, P 5.

Lemma 3.9 If (7, f, A) satisfies P'1-P'5, and if v is a good node then one can send A units of flow from v to r, and the resulting flow f' is non-negative.

The strengthening of property P4 is needed for the nest lemma..

Lemma 3.10 Suppose that the triple (7, f, A) satisfies P'1-P'5. Suppose further that (v, w) is a bad arc for spanning tree T. Then within an additional $3 + 2 \lceil \log n \rceil$ scaling phases the number of strongly feasible arcs will increase.

Proof: Since arc (v, w) is bad, it is a downward arc and $f($v, $w)$ = 0. Therefore, f is zero on all arcs leaving T_w. By property P'4, each node in T_w has excess at least A/2. Now let A' be the scaling factor $3 + 2\lceil\log n\rceil$ scaling phases later, and let f' denote a flow after the Δ' scaling phase. Then $\Delta' \leq \Delta/(8n^2)$. By property P'4, $e_f(T_w) \geq |T_w|\Delta/2 \geq 4n^2|T_w|\Delta'$. Moreover, $e_{f'}(T_w) \leq 3|T_w|\Delta'$. Thus, at least $3n^2|T_w|\Delta'$ units of flow have been sent from T_w to $V \setminus T_w$ while transforming flow f into flow f'. At most $n-1$ arcs have positive flow in f'. This implies that there must be some arc leaving T_w with flow greater than $3n|T_w|\Delta' \geq 3n\Delta' > -e_{f'}(r)$, where the last inequality follows from $P'4$. By definition, such an arc is strongly feasible. Since there are no arcs leaving T_w with positive flow subject to f, it follows that the number of strongly feasible arcs has increased. ∎

We have shown that if some arc becomes bad during a scaling phase, then within $3+2\lceil\log n\rceil$ additional scaling phases there is a new strongly feasible arc. Since there are at most n arcs that can become strongly feasible, it follows that there are $O(n \log n)$ iterations in which an arc becomes bad, causing the algorithm to execute at least one pivot.

Corollary 3.11 The number of scaling phases in which some pivot takes place is $O(n\log n)$.

The above corollary implies a strongly polynomial bound on the number of pivots. In order to show a strongly polynomial bound on the running time we have to limit the number of scaling phases that do not execute pivots. Let $\mathcal{T}$ be the current spanning tree and let f denote the current flow at the Δ scaling phase. Suppose that $(\mathcal{T}, f, \Delta)$ satisfies P'1 - P'5. We will determine a new scaling factor $\Delta' \leq \Delta$ and a flow f', such that $(\mathcal{T}, f', \Delta')$ also satisfies P'1 - P'5, and such that there is no flow f'' for which $(\mathcal{T}, f'', \Delta/8)$ satisfies P'1-P'5. Consequently, if we continue the scaling algorithm starting with flow f', then within 3 scaling phases the spanning tree $\mathcal{T}$ cannot be feasible, and there has to be a pivot.

Suppose that $(\mathcal{T}, f, \Delta)$ satisfies P'1 - P'5. Let $\Delta' = \max(-2f^{\mathcal{T}}(v, w)/|\mathcal{T}(w)| : (v, w) \in \mathcal{T})$. Observe that $\Delta' \leq 0$ if and only if $\mathcal{T}$ is an optimal tree. We will show subsequently that there is a flow f' such that $(\mathcal{T}, f', \Delta')$ satisfies P'1-P'5. First we show that there can be no flow f'' such that $(\mathcal{T}, f'', \Delta'/8)$ satisfies P'1-P'5.

Lemma 3.12 If $\Delta' > 0$ then there is no flow f'' such that $(\mathcal{T}, f'', \Delta'/8)$ satisfies P'1-P'5.

Proof: Let f'' be such a flow. Consider the flow on arc (v, w) where $\Delta' = -2f^{\mathcal{T}}(v, w)/|\mathcal{T}(w)|$. By Lemma, 3.1 and Property P'4, $f''(v, w) = f^{\mathcal{T}}(v, w) + e_{f''}(T_w) = -\Delta'|\mathcal{T}(w)|/2 + e_{f''}(\mathcal{T}_w) \leq -\Delta'|\mathcal{T}(w)|/2 + |\mathcal{T}(w)|(3\Delta'/8) < 0$. ∎

Nest we show how to find a feasible flow f' such that $(\mathcal{T}, f', \Delta')$ satisfies P'1-P'5, assuming that $\Delta' \leq \Delta/3$. The procedure **MAKE-FLOW** is given in Figure 4. We start by creating an excess of $3\Delta'/2$ at each node in $v \neq r$. Let g denote the resulting flow. We then examine nodes in the reverse of a breadth first search ordering (*i.e.*, we start at the leaves and work towards the root), and for each node v examined we modify g to make sure that the flow on $(pred(v), v)$ satisfies $P5$.

```
procedure MAKE-FLOW(T);
    A' ← max_{(v,w)∈T}(−2f^T(v,w)/|T(w)|).
    Define g by sending b(v) + 3Δ'/2 units of flow from r to every node v ≠ r in T.
    for v ∈ V in a the reverse BFS ordering do begin
        Let w = pred(v).
        if (w, v) is a downward arc and (w, v) is not strongly feasible
          then begin
            send f(w, v) mod A' units of flow from v to node r in T.
          end;
        if (v, w) is an upward arc and (v, w) is not strongly feasible
          then begin
            send −f(w, v) mod A' units of flow from v to node r in 7.
          end;
    end;
end.
```

Figure 4: **MAKE-FLOW** algorithm.

Lemma 3.13 Suppose that $\Delta' \leq \Delta/3$. Then the flow g computed by MAKE-FLOW satisfies $e_g(v) \leq e_f(v)$ for every $v \neq r$.

Proof: By property $P'4$, we have that $e_f(v) \geq \Delta/2$, $\Delta' \leq \Delta/3$ and $e_g(v) \leq 3\Delta'/2$. This implies the lemma. ∎

Lemma 3.14 Suppose that $(\mathcal{T}, f,$ A) satisfies P'1 - P'5, and $\Delta' \leq \Delta/3$. Then $(\mathcal{T}, g, \Delta')$ computed by MAKE-FLOW satisfies P'1-P'5 and all arcs of $\mathcal{T}$ are good.

Proof: P1 and P3 are satisfied by definition. By construction, g satisfies P'4 and P'5. It remains to show that the resulting flow $g(v, w)$ is non-negative on all upward arcs in $\mathcal{T}$ and positive on all downward arcs. This will imply imply P2, and the fact that all arcs are good.

Since f is nonnegative and, by Lemma 3.13, we have $e_g(v) \leq e_f(v)$ for v $\neq r$, the flow g on upward arcs is nonnegative. Suppose now that (v, w) is a downward arc. Then $g(v, w)$ = $f^{\mathcal{T}}(v,w)$ t $e_g(\mathcal{T}_w) \geq -|\mathcal{T}(w)|\Delta'/2 + e_g(\mathcal{T}_w) > 0$, since each node in $\mathcal{T}_w$ has at least A'/2 excess. ∎

Notice that Lemma 3.13 implies that the procedure **MAKE-FLOW** can be also thought of as modifying f by sending flow to the root r from some other nodes.

Lemma 3.15 Suppose that $(\mathcal{T}, f,$ A) satisfies P'1 - P'5, and A' $\leq \Delta/3$. Then every arc that is strongly feasible subject to f is also strongly feasible subject to g.

Proof: The lemma is implied by Lemmas 3.13 and 3.8. ∎

```
procedure STRONG-SCALING-SIMPLEX(T);

    We assume that:
        T is a dual feasible tree and all arcs are good subject to the flow f = 0;

    A ← ∞.
    while T is not optimum do begin
        let A' = max(−2f^T(v,w)/|T(w)| : (v,w) ∈ 7)
        if Δ' = 0
            then begin
                quit and return the optima.1 tree 7;
            end;
        if A'> Δ/6
            then A ← A/2
            else begin
                call MAKE-FLOW(T), f ← g and A ← A'.
            end;
        while S_f (3Δ/2) ≠ ∅ do begin
            let v be a node in S_f(3Δ/2); send A units of flow from v to r.
            if there is a bad arc in 7
                then begin
                    call MAKE-GOOD(T, f)
                end;
        end;
    end;
end.
```

Figure 5: **STRONG-SCALING-SIMPLEX** algorithm.

Theorem 3.16 The **STRONG-SCALING-SIMPLEX** algorithm, if started from a shortest path tree rooted at r, performs dual simplex pivots, and finds the optimal spanning tree for the uncapacitated transshipment problem after $O(n^3 \log n)$ pivots, and $O(n^2(m + n\log n) \log n)$ time.

Proof: The tree 7 is good before each augmentation, and all bad arcs are on an inverse good path before each call to **MAKE-GOOD.** The fact that the algorithm maintains P'1-P'5 can be proved analogously to the proof of Theorem 3.7. The procedure **MAKE-FLOW** takes $O(n)$ steps. Subsequently, within 0(1) scaling phases there will be a bad arc, and by Lemma, 3.10 in $O(\log n)$ scaling phases a new asc will become strongly feasible. There can be at most n strongly feasible arcs at a. time. Therefore, Lemma 3.15 implies that there will be at most $O(n \log n)$ scaling phases. Now Theorems 3.4 and 3.5 imply the bounds on the number of pivots and the running time. ∎

Using the reduction from the minimum-cost flow problem to the uncapacitated transshipment problem we obtain the following corollary.

Corollary 3.17 The (capacitated) transshipment problem can be solved by the dual network simplex method in $O(m^3 \log n)$ pivots, and the algorithm can be implemented to run in $O(m^2(m +$

$n \log n) \log n)$ time.

4 An Efficient Pivoting Strategy

In this section we show how to use a. variation on Orlin's [11] fast minimum-cost transshipment algorithm to design a pivoting strategy that decreases the number of pivots for the dual network simplex algorithm to $O(mn \log n)$.

The simplex algorithm in the previous section can be viewed as an implementation of the corresponding scaling algorithm. During the scaling algorithm all nodes, except the root 'r', have positive excess. The simplex implementation maintains a dual feasible tree $\mathcal{T}$ such that an augmentation from every node v to the root r is possible in the tree. Whenever the scaling algorithm augments the flow from some node v to r, this augmentation can be done in the tree.

Such a direct simplex implementation of the faster scaling algorithm is not possible. For some pairs of nodes that the scaling algorithm might choose for an augmentation there might be no sequence of pivot steps that makes an augmentation between these two nodes possible in the tree. The version of the transshipment algorithm used here allows greater freedom in choosing the augmentation. We show that a simple pivoting strategy can be viewed as implementing one of the possible choices. In effect, we let the simplex algorithm choose among the possible augmentations.

In Section 4.1 we give the modified version of Orlin's scaling algorithm, and review the proof of its running time. In Section 4.2 we use this algorithm to design a dual network simplex algorithm that takes at most $O(mn \log n)$ pivots.

4.1 Modified version of Orlin's excess scaling algorithm

The main idea of all excess scaling algorithms is to maintain a feasible pseudoflow and a price function, such that the reduced cost of every arc is non-negative, and the reduced cost of arcs with positive flow is zero. The initial flow is zero, initial prices can be computed by a shortest path computation. We repeatedly augment the flow along shortest paths from a node with positive excess to a node with negative excess, gradually reducing all the excesses to zero. Two observations justify this method: (1) moving the flow along a minimum-cost path preserves the invariant that the current pseudoflow has the minimum-cost among all the pseudoflows with the same excess; (2) a shortest path computation suffices both to find a path along which to augment the flow and to find appropriate price changes that preserve the nonnegativity of the reduced costs. Let AUGMENT(S, T, Δ) denote the subroutine that augments the flow by Δ along a shortest path from some node in S to some node in T.

The scaling algorithm that was the basis of the simplex algorithm in the previous section maintains that all nodes except the root r has positive excess. In Orlin's scaling algorithm we need to augment the flow by more then the excess at the end on the path, thereby changing the sign of the excess during the algorithm. Let us define the set of nodes with large deficit $T_f(\Delta) = \{v \in V : e_f(v) < -\Delta\}$ analogously to $S_f(\Delta)$.

The algorithm maintains a *scaling parameter* Δ, and it consists of a number of scaling phases. Each phase consists of a sequence of augmentations. The main difference between this algorithm and the one used in the previous section is that at the Δ-scaling phase we either augment the flow by Δ from some node in $S = S_f(\frac{n-1}{n}\Delta)$ to a node in $T = T_f(\frac{1}{n}\Delta)$, or, reversing the roles of S and T, from some node in $S = S_f(\frac{1}{n}\Delta)$ to a node in $T = T_f(\frac{n-1}{n}\Delta)$ until both $S_f(\frac{n-1}{n}\Delta)$ and $T_f(\frac{n-1}{n}\Delta)$ become empty. At this point we divide Δ by two and a new scaling phase starts.

The idea of making this algorithm strongly polynomial is similar to the previous one. Notice that the sets $S_f(\frac{n-1}{n}\Delta)$ and $T_f(\frac{n-1}{n}\Delta)$ are monotone decreasing during the Δ scaling phase. Therefore, there can be at most n augmentations per scaling phase. The amount of flow moved during the Δ scaling phase is no more than $n\Delta$, and if an arc has more than $2n\Delta$ flow then it will have positive flow in all subsequent phases. Such arcs will be declared strongly feasible.

More precisely, the algorithm FAST-EXCESS-SCALING maintains a set of *strongly feasible arcs* E_s and a pseudoflow f that satisfies the following properties.

S1. $f(v,w) \geq 0$ on every $(v,w) \notin E_s$.

S2. $f(v,w) > 0$ on every $(v,w) \in E_s$.

S3. $c_p(v,w) \geq 0$ for all arcs in the residual graph.

S4. At most one node has non-zero excess in every connected component of $G_s = (V, E_s)$.

S5. $f(v,w)$ is a multiple of Δ on every arc $(v,w) \notin E_s$.

An arc becomes strongly feasible if it carries at least $5n\Delta$ flow. The algorithm FAST-EXCESS-SCALING is shown in Figure 6. Initially, $\Delta = \max_{v\in V} |b(v)|$. Each time we declare an arc strongly feasible we do a special augmentation to collect the excess of each connected component of E_s to a single node. After a scaling phase Δ is divided by 2 as long as the current pseudoflow is non-zero on some arc not in E_s. When every arc with non-zero flow is in E_s the scaling is restarted by setting Δ to be the maximum absolute value of a current excess.

Lemma 4.1 *Algorithm* FAST-EXCESS-SCALING *maintains properties S1-S5.*

Proof: Notice that if $S_f(\frac{n-1}{n}\Delta)$ is not empty then $T_f(\frac{1}{n}\Delta)$ must not be empty since excesses sum to zero. Similarly if $T_f(\frac{n-1}{n}\Delta)$ is not empty then $S_f(\frac{1}{n}\Delta)$ must not be empty. This proves that the augmentations are possible.

Throughout the algorithm we shall maintain a pseudoflow f and a price function p. It is easy to see that S1, and S3-S5 is satisfied throughout. Strongly feasible arcs have reduced cost 0 by properties S2 and S3, therefore augmentation through a connected component of G_s can be

```
procedure FAST-ESCESS-SCALING;
    Δ ← max_{v∈V} |b(v)|;
    while Δ ≠ 0 do begin
        while S_f((n-1)/n Δ) ∪ T_f((n-1)/n Δ) ≠ ∅ do begin
            if S_f((n-1)/n Δ) ≠ ∅
              then begin
                S ← S_f((n-1)/n Δ);
                T ← T_f(1/n Δ);
                Call AUGMENT(S, T, Δ);
                Add (v, w) to E_s for every (v, w) such that f(v, w) > 5nΔ;
                Use E_s arcs to collect the excess of the new connected component of G_s to one node
              end;
              else begin
                S ← S_f(1/n Δ);
                T ← T_f((n-1)/n Δ);
                Call AUGMENT(S, T, Δ);
                Add (v, w) to E_s for every (v, w) such that f(v, w) > 5nΔ;
                Use E_s arcs to collect the excess of the new connected component of G_s to one node;
              end;
        end;
    end;
    if f is zero on all arcs not in E_s
      then Δ ← max_{v∈V} |b(v)|;
      else Δ = Δ/2;
end.
```

Figure 6: Algorithm **FAST-EXCESS-SCALING.**

done through the strongly feasible arcs. Note, that this version of the algorithm can move more than $n\Delta$ flow during the Δ-scaling phase. Each time an arc is declared strongly feasible, we do a special augmentation to collect the excess of a connected component of G_s. This might yield another $n\Delta$ amount of flow to be moved through strongly feasible arcs. Collecting the excess in a connected component of G_s can yield an increased excess at some nodes, and therefore extra augmentations. The number of augmentations during a phase is at most n plus the number of contractions. Therefore the overall amount of flow moved during and after the 4 scaling-phase, is at most, $5n\Delta$ and arcs that carry more than 5124 flow will never become empty, and hence S2 is satisfied. ∎

It will help the analysis to consider the arcs in E_s as *contracted*, and hence the connected components of (V, E_s) are *pseudo-nodes* of the contracted graph. We will use $\hat{V}$ to denote node set of the contracted graph.

The main idea in the analysis is to show after a node v (either in V or a pseudo-node) participated as an encl-point of a. shortest path computation, after at most $O(\log n)$ additional scaling phases, an arc adjacent to v will become strongly feasible. For a node v in V this follows from the fact that v $\notin S_f(\Delta) \cup T_f(\Delta)$ unless $|b(v)| \geq 4$, and therefore $O(\log n)$ scaling phases after the node first served as a starting node, an arc incident to v will carry enough flow to be contracted. However, $|b(v)| \geq |e_f(v)|$ might not hold for pseudo-nodes.

We say that a node v is *active* during a scaling phase if $v \in S_f(\frac{n-1}{n}\Delta) \cup T_f(\frac{n-1}{n}\Delta)$ at some time during the phase.

Lemma 4.2 The number of shortest path computations during a phase is bounded by the number of active nodes during the phase.

Theorem 4.3 A node v can be active in at most $O(\log n)$ phases before it is contracted.

Proof: A pseudo-node can become active once due to contraction. However, when a node v is active for the second time, it must already exist at the end of the previous scaling phase. Let 4 denote the scaling parameter in the phase when v is active for the second time. If 4 for this phase was defined by $\max_{w \in \hat{V}} e_f(w)$ then $e_f(v)$ = b(w). Otherwise, the scaling parameter in the previous phase is 24. At the end of the 24-scaling phase both $S_f(\frac{n-1}{n}2\Delta)$ and $T_f(\frac{n-1}{n}2\Delta)$ are empty. Therefore, at the beginning of this phase, we have $\frac{n-1}{n}\Delta < |e_f(v)| \leq \frac{2(n-1)}{n}\Delta$. But $b(v) - e_f(v)$ is an integer multiple of 24. This implies that $|b(v)| > \frac{1}{n}\Delta > \frac{1}{2n}e_f(v)$. In either case, after at most $O(\log$ n) more scaling phases the scaling parameter 4 will be less than $|b(v)|/(5n^2)$. At the end of that scaling phase the flow f will satisfy the following inequality.

$$|\sum_{w \in \hat{V}} f(v, w)| = |b(v) - e_f(v)| \geq |b(v)| - 4 \geq (5n^2 - 1)\Delta. \tag{4}$$

Consequently, at least one arc incident to v carries more than $5n\Delta$ flow, and hence v will be contracted. ▮

Lemma 4.2 and Theorem **4.3** bound the number of shortest path computations during the algorithm. All other work takes linear time per scaling phase. At least one arc is contracted in each group of $O(\log n)$ scaling phases, and therefore, there are at most $O(n \log n)$ scaling phases.

Theorem 4.4 Algorithm **FAST-EXCESS-SCALING** solves the transshipment problem in $O(n \log \min\{n, B\})$ computations of single-source shortest paths in networks with non-negative lengths.

4.2 Decreasing the number of pivot steps

The sequence of pivot steps in the dual network simplex algorithm in the previous section is guided by a capacity scaling algorithm. The main needed change in the scaling algorithm to guide the pivot selection is that all augmentations are carried out in the tree. In this section we give a similar simplex implementation of algorithm **FAST-EXCESS-SCALING.** If the set T is not reachable from S over residual arcs in the tree, then the algorithm initiates a sequence of dual simplex pivot steps, changing the tree $\mathcal{T}$ into one in which an augmentation in the tree

procedure TREE-AUGMENT$^+$(S, T, A, $\mathcal{T}$, f);

We assume that:
$\mathcal{T}$ is a dual feasible tree;
p is the price function which corresponds to $\mathcal{T}$;
f is a pseudoflow that is zero outside of the tree $\mathcal{T}$.

Choose node $s \in S$;
$R \leftarrow$ Nodes reachable from s over residual arcs in $\mathcal{T}$ subject to the flow f;
while R(7) $\cap T$ **=** $\emptyset$ and there is an arc (v, w) entering: R(7) such that $f^{\mathcal{T}}(v, w) < 0$ **do begin**
$(v, w) \leftarrow$ an arc entering R(7) such that $f^{\mathcal{T}}(v, w) < 0$;
CallPIVOT($\mathcal{T}$, (v, w));
Update X(7);
end;
$t \leftarrow$ A node in R(7) $\cap T$;
$P \leftarrow$ A path from s to t in through residual arcs in $\mathcal{T}$;
Move A units of flow from s to t along P;
end.

Figure 7: Pivot Steps that Make an Augmentation from s Possible.

is possible. The first call to **AUGMENT** is replaced **by** TREE-AUGMENT$^+$, and the second call **by TREE-AUGMENT-.** The TREE-AUGMENT$^+$ **procedure first chooses a node s** $\in$ S, **and then executes** a **sequence of** pivot steps until there is *some* $t \in T$ reachable from s in the tree. The **TREE-AUGMENT-** procedure first chooses a node $t \in T$, and then execute pivot steps until there is *some* $s \in$ S such that t is reachable from it through the residual arcs of the tree. Below we show that the properties **of** the **FAST-EXCESS-SCALING** algorithm allow us to prove that the number of pivots needed to be done between any two augmentations is small.

The procedure TREE-AUGMENT$^+$ (see Figure 7) starts by choosing a node $s \in$ S. Let R(7) denote the set of nodes reachable from s over residual arcs in the tree $\mathcal{T}$. If the set R(7) $\cap$ T is not empty, then we augment the flow along the path in the tree from s to some node t in the intersection. If $R(\mathcal{T}) \cap T$ is empty, then we iteratively choose an arc (v, w) entering R(7) such that $f^{\mathcal{T}}(v, w) < 0$, and pivot **on** this arc, until the intersection R(7) II T is not empty. Then we augment the flow from s to some node t in the intersection. Since **TREE-AUGMENT-** is analogous, we omit its description.

Nest we need to show that each call to TREE-AUGMENT$^+$ (and, analogously, to TREE-**AUGMENT-),** results in a small number of pivots. More precisely, we have to prove two claims. First, we have to prove that the intersection $R(\mathcal{T})$ **n** T is empty, then there exists an arc (v, w) entering R(7) such that $f^{\mathcal{T}}(v, w) < 0$. Then we have to show that there will be only a small number of pivots (actually, at most m) needed to produce a tree such that R(7) $\cap$ T is not empty.

Note, that replacing the calls to **AUGMENT** in the description of the **FAST-ESCESS-SCALING** algorithm (see Figure **6) by calls to** TREE-AUGMENT$^+$ and **TREE-AUGMENT-,** causes TREE-**AUGMENT+** to be called with parameters S = $S_f(\frac{n-1}{n}\Delta)$, T = $T_f(\frac{1}{n}\Delta)$.

Lemma 4.5 Let $\mathcal{T}$ be a dual feasible tree, f be a pseudoflow that is zero on all arcs not in the tree, s be a node in $S_f(\frac{n-1}{n}\Delta)$ and let R(7) denote the set of nodes reachable from s over residual arcs in the tree 7. If R(7) $\cap\, T_f(\frac{1}{n}\Delta) = \emptyset$ then at least one of the tree arcs (v, w) entering R(7) has $f^{\mathcal{T}}(v, w) < 0$. An analogous statement holds for $t \in T_f(\frac{n-1}{n}\Delta)$ instead of s.

Proof: We shall only prove the first statement. By the definition of R(7) the tree arcs leaving R(7) are not in the residual graph. This implies that no arc in 7 leaves R(7) and also no flow leaves or enters R(7). Consequently,

$$\sum_{v \in R(\mathcal{T})} b(v) = -\sum_{v \in R(\mathcal{T})} e_f(v).$$

R(7) $\cap\, T_f(\frac{1}{n}\Delta) = \emptyset$ and $e_f(s) > \frac{n-1}{n}\Delta$ imply that $\sum_{v \in R(\mathcal{T})} e_f(v) > 0$. In particular, it follows that R(7) $\neq V$. Now consider the arcs $(v_1, w_1), \ldots, (v_k, w_k)$ of the tree 7 entering R(7). The sets $V \setminus H_{\mathcal{T},(v_i,w_i)}$ for $i = 1, \ldots, k$ together with *R(7)* partition V. Therefore

$$\sum_{i=1,\ldots,k} \sum_{v \notin H_{\mathcal{T},(v_i,w_i)}} b(v) = -\sum_{v \in R(\mathcal{T})} b(v).$$

The above two equations and inequality $\sum_{v \in R(\mathcal{T})} e_f(v) > 0$ imply that $\sum_{v \in V \setminus H_{\mathcal{T},(v_i,w_i)}} b(v) > 0$ for at least one index i. Hence $f^{\mathcal{T}}(v_i, w_i) < 0$. ∎

Lemma 4.6 During the execution of the procedure TREE-AUGMENT$^+$, no arc deleted from T will reenter the tree. An analogous statement holds for procedure **TREE-AUGMENT-**.

Proof: We shall only prove the first statement. The procedure **TREE-AUGMENT-** can be treated similarly. Consider an execution of the procedure TREE-AUGMENT$^+$. Let R(7) denote the set of nodes reachable from s over residual arcs in the tree 7. The pivot step deletes an arc (v, w) for some $w \in$ *R(7)* and adds to the tree an arc (v', w') for some $w' \notin$ R(7). This implies that the set R(7) is non-decreasing during the execution of the procedure, and also implies that an arc which was deleted from the tree can not be added back. ∎

Lemmas 4.5 and 4.6, and Theorem 4.4 imply the following Theorem.

Theorem 4.7 The uncapacitated transshipment problem can be solved by the dual network simplex algorithm in a sequence of $O(nm \log(\min\{n, B\}))$ simplex pivot steps.

Proof: Lemma. 4.5 implies that the simplex algorithm presented above will continue until it finds an optimal transshipment. Lemma. 4.6 implies that an augmentation is done (in the accompanying **FAST-ESCESS-SCALING** algorithm) after a sequence of at most m pivot steps. This and Theorem 4.4 imply that the number of pivot steps throughout the algorithm is $O(mn \log(\min\{B, n\}))$. ∎

A straightforward implementation of the algorithm takes $O(m)$ time per pivot step (this is how long it takes to choose the minimum reduced cost arc leaving the set $H_{T,(v,w)}$). We get an $O(nm^2 \log(\min\{B, n\}))$ overall running time. Notice, that this running time is worse then the one proved in Theorem 3.16 unless m is close to n.

Using the equivalence of the minimum-cost flow and the uncapacitated transshipment problems mentioned in Section 2 we get the following corollary.

Corollary 4.8 The (capacitated) transshipment problem can be solved by the dual network simplex algorithm in $O(m^2 \log \min\{B, n\})$ pivot steps.

References

[1] R. I<. Ahuja, T. L. Magnanti, and J. B. Orlin. Network Flows. In G. L. Nemhauser, A. H. G. Rinnooy Kan, and M. J. Todd, editors, *Handbook of Operations Research and Management Science*, volume 1: Optimization, 211-369, 1989.

[2] M. L. Balinski. Signature Methods for the Assignment Problem. *Oper.* Res., X3527-536, 1985.

[3] W. H. Cunningham. Theoretical Properties of the Network Simplex Method. *Math. of Oper. Res.,* 4:196–208, 1979.

[4] R.. Dial, F. Clover, D. Karney, , and D. Klingman. A Computational Analysis of Alternative Algorithms and Labeling Techniques for Finding Shortest Path Trees. *Networks,* 9:215–248, 1979.

[5] L. R. Ford, Jr. and D. R. Fulkerson. Flows *in Networks.* Princeton Univ. Press, Princeton, NJ., 1962.

[G] A. V. Goldb erg, M. D. Grigoriadis, and R. E. Tarjan. Efficiency of the Network Simplex Algorithm for the Maximum Flow Problem. *Math. Progr. A,* 50(3):277-290, *1990.*

[7] A. V. Goldberg, É. Tardos, and R. E. Tarjan. Network Flow Algorithms. in *Paths, Flows and VLSI-Design,* eds. B. Korte, L. Lovász, H. J. Proemel, and A. Schrijver, Springer Verlag, 101-164, 1990.

[8] D . Goldfarb and J. Hao. A Primal Simplex Algorithm that Solves the Maximum Flow Problem in at Most nm Pivots and $O(n^2m)$ Time. Technical report, Department of IE and OR, Columbia University, 1988.

[9] J. B. Orlin. Genuinely Polynomial Simplex and Non-Simplex Algorithms for the Minimum Cost Flow Problem. Technical Report No. 1615-84, Sloan School of Management, MIT, 1984.

[10] J. B. Orlin. On the Simplex Algorithm for Networks and Generalized Networks. *Math. Prog. Studies,* 24:166–178, 1985.

[11] J. B. Orlin. A Faster Strongly Polynomial Minimum Cost Flow Algorithm. Technical Report 3060-89-MS, Sloan School of Management, MIT, 1989. (Preliminary version appeared in the Proc. 20th ACM Symp. on Theory of Computing).

[12] S. Plotkin and É. Tardos. Improved Dual Network Simplex. In *Proceedings of the 1st ACM-SIAM Symposium on Discrete Algorithms,* pages 367–376, 1990.

[13] R. E. Tarjan. Efficiency of the Primal Network Simplex Algorihm for the Minimum-Cost Circulation Problem. *Math.* of *Oper. Res.*, 16(2):272-291, 1990.