Discrete Applied Mathematics 45 (1993) 93–115
North-Holland

A genuinely polynomial primal simplex algorithm for the assignment problem

Mustafa Akgül

Department of Industrial Engineering, Bilkent University, 06533 Bilkent, Ankara, Turkey

Received 21 March 1990
Revised 2 August 1991

Abstract

Akgül, M., A genuinely polynomial primal simplex algorithm for the assignment problem, Discrete Applied Mathematics 45 (1993) 93–115.

We present a primal simplex algorithm that solves the assignment problem in $\frac{1}{2}n(n+3)-4$ pivots. Starting with a problem of size 1, we sequentially solve problems of size $2,3,4,\ldots,n$. The algorithm utilizes degeneracy by working with strongly feasible trees and employs Dantzig's rule for entering edges for the subproblem. The number of nondegenerate simplex pivots is bounded by $n-1$. The number of consecutive degenerate simplex pivots is bounded by $\frac{1}{2}(n-2)(n+1)$. All three bounds are sharp. The algorithm can be implemented to run in $O(n^3)$ time for dense graphs. For sparse graphs, using state of the art data structures, it runs in $O(n^2 \log n+nm)$ time, where the bipartite graph has $2n$ nodes and m edges.

Keywords. Assignment problem, network simplex method, linear programming, polynomial algorithms, strongly feasible bases, Hirsch conjecture.

Introduction

The assignment problem is one of the most-studied, well-solved and important problems in mathematical programming. Solution procedures vary from primal-dual/successive shortest paths [12,25,26,33,35,46] (see [23] for a survey), cost parametric [43], recursive [45], relaxation [24,31] to primal methods [10,21]. It has many applications, in particular it occurs as a relaxation of the travelling salesman problem. It has been generalized to bottleneck, quadratic and algebraic cases, see [15,16] for references.

Correspondence to: Professor M. Akgül, Department of Industrial Engineering, Bilkent University, 06533 Bilkent, Ankara, Turkey.

There are several efficient primal simplex algorithms for the assignment problem, either especially designed for the assignment problem [11] or designed for the transshipment problem [28,42]. Naturally, they all work well in practice, but theoretically they are exponential algorithms. Roohey-Laleh [39] exhibits a family of problems with exponentially long nondegenerate pivot sequences.

Balinski [8] introduced a dual simplex algorithm with the number of pivots bounded by $\frac{1}{2}n(n-1)$. Balinski's algorithm starts with a dual feasible tree, and searches for a strongly (primal) feasible tree. Strictly speaking the algorithm is not a dual simplex method; since it allows replacing primal feasible tree edges with cotree edges. Nevertheless, it is a pivotal method. With proper implementation it runs in $O(n^3)$ time. Goldfarb [29], modified Balinski's signature method to a sequential method and improved its efficiency. Balinski [9] also gave a purely dual simplex algorithm with $O(n^2)$ pivot and $O(n^3)$ time complexity. He developed a theory of strongly feasible dual trees with remarkable properties. Akgül [4] gave a sequential version of Balinski's method similar to Goldfarb's algorithm. Balinski's algorithm led to signature guided dual based forest algorithms (see, e.g., [1,6]).

Hung [30] gave a polynomial primal simplex method that requires $O(n^3 \log \Delta)$ pivots, where $\Delta = \Delta_0$ and $\Delta_k = cx^k - cx^*$ is the difference in the objective function value between the current solution x^k and an optimal solution x^*, and x^0 is the initial basic solution. He notices that when a nondegenerate pivot occurs with Dantzig's rule, we have the relationship $\Delta_{k+1} \leq ((n-1)/n)\Delta_k$. Thus, the number of nondegenerate pivots by Dantzig's rule is bounded by $O(n \log \Delta)$. Cunningham [20] earlier bounded the number of degenerate pivots at an extreme point by $n(n-1)$ by utilizing strongly feasible trees and a certain pivot rule. Combining these one obtains the given bound.

Orlin [36] using a perturbation technique (which is equivalent to strongly feasible trees) reduced the bound on the number of pivots to $O(n^2 \log \Delta)$. He later reduced the bound to $O(n^2 m \log n)$ where m is the number of edges and n is the number of nodes in the graph by showing that there exist an equivalent network with cost coefficients bounded by $4^m (m!)^2$; and hence proving $\log \Delta$ is $O(m \log n)$. The above algorithms, [30,36] at least implicitly, are influenced by the ellipsoidal algorithm. Their common feature is the reduction of the objective function value by a fraction depending on n or m, independent of the rest of the problem parameters. In the ellipsoidal algorithm this ratio is $\exp(-1/m^2)$, whereas, say, in Orlin's algorithm it is $\exp(-1/n^2)$. It is worth mentioning that for totally unimodular linear programs the ellipsoidal algorithm needs $O(m^2 \log(m \|c\| \, \|b\|))$ iterations [2, Chapter 4], where as before m is the number of variables, and $\|x\|$ denotes the (Euclidean) norm of the vector x.

Cunningham and Roohy-Laleh [39] developed a genuinely polynomial primal simplex algorithm. The algorithm needs $O(n^3)$ pivots and $O(n^5)$ time in the worst case.

Here we present a primal simplex algorithm with $O(n^2)$ pivot and $O(n^3)$ time bound. We cast the problem as an instance of transshipment problem and work on a directed graph. The algorithm has three features. We consider an increasing sequence of subgraphs, the last of which is the graph of the original problem, and each one differs from the previous one by addition of some of the edges incident with one node. In matrix terms, we solve the subproblems defined by principal minors of the cost matrix. The motivation for this approach came from the author's work on the shortest path problem [5]. Moreover, we restrict the feasible basis to strongly feasible trees. Interestingly, degeneracy together with strongly feasible trees is very helpful, at least theoretically. The third component of the algorithm is the use of Dantzig's rule restricted to the current subgraph. Our algorithm is a purely primal simplex algorithm, because we carry a full basis of the original problem all the time. We do not attempt to evaluate the change in the objective function value. Instead, we study the structure of the set of nodes on which we make dual variable changes during the solution of the current subproblem. We call these sets cut-sets. It turns out that: (i) cut-sets are disjoint, (ii) edges originating from a cut-set are dual feasible once for all the subgraph under consideration, (iii) dual infeasible edges have the property that their tails have no dual variable change and (iv) each node is subject to at most one dual variable change. Thus passing from a subproblem of size $k \times k$ to a subproblem of size $(k+1)\times(k+1)$ can be done with at most $k+2$ pivots. Hence, we have the bound $\frac{1}{2}n(n+3)-4$ for the number of pivots. The total number of nondegenerate pivots is bounded by $n-1$. The total number of consecutive degenerate pivots is bounded by $\frac{1}{2}(n+2)(n-1)$. All of these bounds are sharp. A very naive implementation runs in $O(n^3)$ time for dense graphs. For sparse graphs, using the state of the art data structures [27,40], the algorithm runs in $O(n^2 \log n + nm)$ time, which is currently the best available bound [27,29].

1. Preliminaries

We view the assignment problem as an instance of the transshipment problem over a directed (bipartite) graph, $G=(U,V,E)=(N,E)$, where U is the set of source (row) nodes, V is the set of sink (column) nodes, $N=U\cup V$ and E is the set of edges. The edge $e=(i,j)\in E$ has *tail* $t(e)=i$ and *head* $h(e)=j$, is directed from its tail to its head, has cost $c_e=c_{ij}$ and flow x_e. Thus the AP can be formulated compactly as

$$\min\{cx:\ Ax=b,\ x\geq 0\} \tag{1}$$

where $x\in\mathbb{R}^E$, $b\in\mathbb{R}^N$ with $b_u=-1$, $u\in U$, $b_v=+1$, $v\in V$, and A is the node-edge incidence matrix of G.

The dual of (1) is

$$\max \sum\ (y_v b_v:\ v\in N)$$

such that

$$y_{h(e)}-y_{t(e)}\leq c_e,\quad \forall e\in E. \tag{2}$$

The network simplex method is a specialization of the primal simplex method of general linear programming to the transshipment problem. It is well known that any basis of (1) corresponds to a tree T of G. Given any T, it is well known that the flow values x_e, $e \in T$ are uniquely determined for each compatible ($\sum b_v = 0$) "supply" vector b. Moreover, the complementary dual basic solution is also uniquely determined once one of the y's is fixed at an arbitrary level. Optimality conditions for the LP are: (i) primal feasibility, (ii) dual feasibility, (iii) complementary slackness. The simplex method maintains a primal feasible basis, and by construction the complementary slackness condition is automatically satisfied. Thus, the simplex method, while maintaining primal feasibility and complementary slackness, tries to satisfy the dual feasibility conditions, namely the constraints (2).

For every co-tree edge $e \in T^{\perp} = E - T$, $T \cup e$ contains a unique cycle $C(T,e)$, called the *fundamental cycle* determined by T and e. We orient $C(T,e)$ in the direction of e. This will give us a partition of $C(T,e)$ as

$$C(T,e) = C^+(T,e) \cup C^-(T,e), \quad e \in C^+(T,e) \tag{3}$$

where $C^+(T,e)$ contains all edges of $C(T,e)$ having the same orientation as e. Let $P = P(T,e)$ be the unique path in T from $t(e)$ to $h(e)$ with the natural partition of the edges in P as P^+ and P^-. Then we have the following relationship between P and $C(T,e)$

$$C^+(T,e) = P^- \cup e, \qquad C^-(T,e) = P^+. \tag{4}$$

Then, the edge f leaving T is determined by

$$\theta = x_f = \min\{x_j : j \in C^-(T,e)\}. \tag{5}$$

The new flow values will be

$$x_j = \begin{cases} x_j + \theta, & j \in C^+(T,e), \\ x_j - \theta, & j \in C^-(T,e), \\ x_j, & j \notin C(T,e). \end{cases} \tag{6}$$

In general, any f satisfying (5) can leave the basis. Clearly $f \in T$. Then $T - f$ will have exactly two components, say X and $X^c = N - X$, with $t(e) \in X$. Then the dual variable change will be

$$y_v = \begin{cases} y_v, & v \in X^c, \\ y_v + \varepsilon, & v \in X \end{cases} \tag{7}$$

where ε is determined so that the entering edge e will satisfy

$$y_{h(e)} - y_{t(e)} = c_e \tag{8}$$

with respect to new dual variables, i.e., ε is the amount of dual infeasibility of the edge e. Unconventionally, we will call f the *cut-edge*, and X the *cut-set*. Following Rockafellar [38], let us define the left-hand side of (2) as the *differential* z_e of e.

Then (2) can be rewritten as

$$z_e \le c_e, \quad \forall e \in E. \tag{2'}$$

The set of edges with one end in X and the other in X^c is called the *fundamental co-cycle* of G determined by T and f, $D(T,f)$. This can now be partitioned into

$$D(T,f) = D^+(T,f) \cup D^-(T,f), \quad e,f \in D^+(T,f), \tag{9}$$

where $D^+(T,f)$ contains all the edges in the cocycle having the same orientation as f, i.e., $\{j \in E\colon t(j) \in X,\ h(j) \in X^c\}$.

Thus, dual variable (potential) change defined by (7) will cause the following changes in differential:

$$z_j = \begin{cases} z_j - \varepsilon, & j \in D^+(T,f), \\ z_j + \varepsilon, & j \in D^-(T,f), \\ z_j, & j \notin D(T,f). \end{cases} \tag{10}$$

The relationship between the basis tree and the simplex tableau is well known [13,38]. The efficiency of the network simplex method comes from the fact that the algorithm works on trees combinatorially rather than on the tableaux algebraically.

Any selection rule specifying the choice of one dual infeasible co-tree edge e and the choice of $f \in T$ satisfying (5) will define a simplex method. For more information on network simplex method see, e.g., [18,19,32].

Cunningham [19] and Barr et al. [11], introduced the concept of strongly feasible trees. Here we follow Barr et al.'s convention. Given a specified node, say, r as a root, let $d_T(x)$ be the distance of the node x from r in the (undirected) tree T, i.e., the number of edges in the unique path from r to x. We say $e \in T$ is directed toward r or a *reverse edge*, if $d_T(t(e)) = d_T(h(e)) + 1$, otherwise directed away from r or a *forward edge*. A feasible rooted tree is *strongly feasible* (*SFT*) if $\forall f \in T$, $x_f = 0$ implies f is a reverse edge.

We need to classify co-tree edges as *forward*, *reverse* and *cross*. $e \in T^\perp$ is a for-

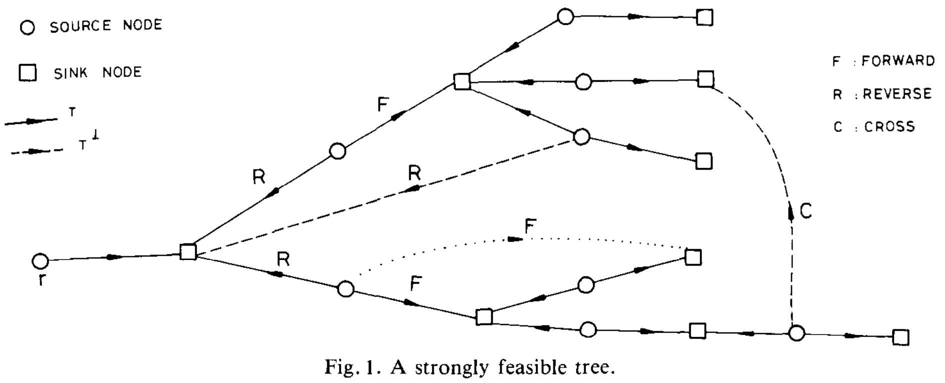

Fig. 1. A strongly feasible tree.

ward edge if $t(e)$ lies on the unique path from r to $h(e)$ and a reverse edge if $h(e)$ lies on the path from r to $t(e)$. Otherwise a co-tree edge is called a cross edge. For nodes u and v, the *nearest common ancestor* NCA(u, v) is the last node common to paths from r to u and v respectively. Then, $e=(u, v)\in E$ is forward if $u=\text{NCA}(u, v)$, reverse if $v=\text{NCA}(u, v)$, otherwise e is a cross edge.

A typical example of a strongly feasible tree for the AP is shown in Fig. 1.

Let $\vec{T}$ be the tree obtained from T by changing all reverse edges to forward edges. $\forall v\in N$, $v\neq r$, there is a unique edge $e=(u,v)\in\vec{T}$ with $h(e)=v$. Through such an edge the *parent* of v is defined as $p(v)=u$. $\vec{T}$ is called a *branching* rooted at r. T is represented as a data structure using parent and a few other pointers. For $v\neq r$, when $(p(v), v)$ is deleted from $\vec{T}$, the component containing v is called the subtree rooted at v and denoted by $\vec{T}(v)$. This subtree contains all nodes that can be reached from v by a directed path in $\vec{T}$. Clearly, $v\in\vec{T}(v)$ and $r\in\vec{T}(v)\Leftrightarrow v=r$. To store the current matching we use an array named *mate*. If $e=(u, v)$ is a matching edge then we have $mate(u)=v$ and $mate(v)=u$.

When rooted at a source node, an SFT has the following properties:

Lemma 1.1. (i) *Every forward edge has flow value* 1, *and every reverse edge has flow value* 0.

(ii) *The root has degree* 1, *every other source node has degree* 2.

(iii) *If e, f satisfy*

$$e\in T^{\perp},\ f\in C(T,e),\quad t(e)=t(f), \tag{11}$$

then the selection of f as the departing variable is valid and maintains strong feasibility.

(iv) *For $e\in T^{\perp}$, the pivot determined by e and* (11) *is nondegenerate iff $e\in T^{\perp}$ is forward iff $f\in T$ is forward.*

(v) *For e, f satisfying* (11) *the pivot is nondegenerate iff $r\in X$.*

(vi) *For e, $f=(u, w)$ satisfying* (11): *the pivot is degenerate iff $X=\vec{T}(u)$, and the pivot is nondegenerate iff $X=N\setminus\vec{T}(w)$.*

Proof. Most of these observations are elementary and were known by Cunningham [19,20], and Barr et al. [11]. (v) will be used to bound the total number of non-degenerate pivots. □

We will use the following property of Dantzig's rule frequently.

Lemma 1.2. *Let $G=(N,E)$ be the graph of a transshipment problem, e be a pivot edge selected by Dantzig's rule, f a cut-edge and X be the corresponding cut-set. Then all edges in $\delta(X,N\setminus X)$ are dual feasible after the pivot.*

Proof. Just notice that $\delta(X,N\setminus X)=D^{+}(T,f)$. Correctness of the lemma follows from (10). □

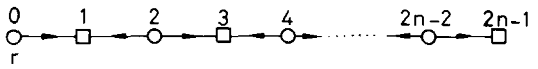

Fig. 2. Initial basis T_0.

2. The new algorithm

We number the source nodes as $0, 2, 4, \ldots, 2n-2$, and call them *even* nodes, denoting them by circles in the diagrams. Similarly, the sink nodes are numbered as $1, 3, 5, \ldots, 2n-1$, called *odd* and denoted by squares. Clearly then, in an SFT rooted at a source node r, a node v is even (odd) iff $d_T(v)$ is even (odd).

Our initial tree T_0 is the path shown in Fig. 2. Since T_0 might seem somewhat restrictive, we remark that:

(1) Given a matching $\sigma: \{1, 2, \ldots, n\} \rightarrow \{1, 2, \ldots, n\}$, one can obtain such a tree containing the given matching by adding the degenerate edges $(i, \sigma(i-1))$, for $i = 2, 3, \ldots, n$.

(2) If any other strongly feasible tree is available, then one can renumber the nodes in preorder, leading to the same analysis.

(3) In case of a sparse graph where some of the edges (in T_0) are absent, the big M technique can be applied with the introduction of artificial edges.

Let us further define for $S, X, Y \subset N$ and $X \cap Y = \emptyset$,

$$\begin{aligned} &\gamma(S) = \{e \in E\colon\ t(e), h(e) \in S\}, \\ &[i] = \{0, 1, 2, \ldots, i\}, \quad \gamma_i = \gamma([i]), \\ &\delta(X, Y) = \{e \in E\colon\ t(e) \in X,\ h(e) \in Y\}, \\ &G[S] = (S, \gamma(S)). \end{aligned} \tag{12}$$

Let $T_2 = T_0$, $G_0 = G_1 = G_2 = (N, \gamma_2 \cup T_2) = (N, T_2)$. Clearly T_2 is an optimal basis for the assignment problem defined by the graph G_2.

We will sequentially define G_i, $2 \le i \le 2n-1$, as subgraphs of G so that (i) $G = G_{2n-1}$, (ii) AP defined by G_{i+1} is solved optimally by relatively few simplex iterations starting with optimal basis T_i^* of G_i.

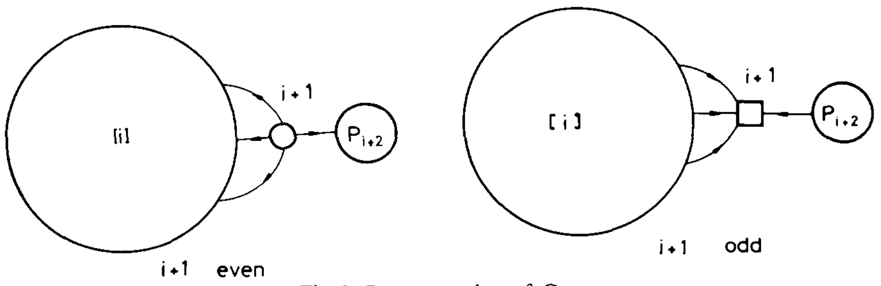

Fig. 3. Representation of G_{i+1}.

Define $G_{i+1}=(N,\gamma_{i+1}\cup T_i^*)=(N,\gamma_{i+1}\cup T_0)$. For i odd (even) the only edges added to G_i to obtain G_{i+1} are $\delta(i+1,[i])$, $(\delta([i],i+1))$. By P_i we denote the subpath of T_0 between i and $2n-1$. We can represent these concepts pictorially as in Fig. 3.

Clearly each G_i defines an assignment problem denoted by AP_i. We will refer G_i instead of AP_i, e.g., we will say G_i is optimal when AP_i is optimal. Thus, we start with G_2 which is optimal. Given G_i optimal, we then solve G_{i+1} optimally starting with optimal tree T_i^* of G_i, which is called *stage* $i+1$ or *processing* of node $i+1$. A stage is called *even* (*odd*), if the node added is even (odd). Our selection of incoming edges is restricted to dual infeasible edges in G_{i+1}. From these edges we choose a most violated edge as the pivot edge, i.e., we use the *restricted Dantzig's rule.* Selection of departing variable will be by (11).

We will call the method the sequential network simplex algorithm (SNSA). We need a few observations:

Lemma 2.1. (i) *At the end of stage i, i.e., when G_i is solved optimally, the subpath P_i is a part of T_i.*

(ii) *In P_i every forward edge has value* 1 *and every reverse edge has value zero.*

When i is odd, the passage from G_i to G_{i+1} is easy. Consider Fig. 4. Let $e\in\delta(i+1,[i])$ be a most violated edge and f be the edge $(i+1,i)$. By Lemma 2.1, $f\in T_i$, $x_f=0$. Clearly, $f\in C^-(T_i,e)$. Since $x_f=0$, the pivot is degenerate and the selection of f as the departing edge is valid. Then, increasing y_j by ε for $j>i$, where $\varepsilon=y_{h(e)}-y_{t(e)}-c_e$ will make G_{i+1} optimal. Thus we have proved that

Lemma 2.2. *For odd i, the passage from G_i to G_{i+1} can be done in at most one pivot, and that pivot is degenerate.*

For even i solving G_{i+1} from an optimal solution of G_i is nontrivial. We will show that we need at most $l=i/2+1$ (the number of source nodes in $[i]$), simplex pivots for that stage.

Let us set up some notation: let $S=[i]$, $v=i+1$, and $f_0=(i,i+1)=(u_0,v)$. We say an edge is *external* if $t(e)\in S$, $h(e)=v$. If $h(e)\in S$, $t(e)\in S$, then e is called *internal.*

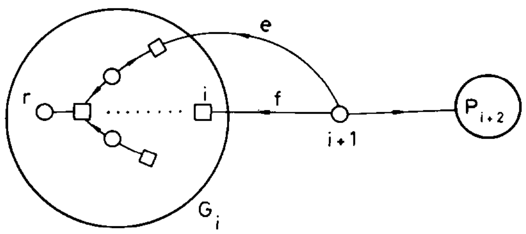

Fig. 4. An even stage.

During the current stage let $\dot{T}_k$ be the basis tree after the kth pivot, with $\dot{T}_0 = T_i^*$. Clearly $f_0 \in \dot{T}_0 = T_i^*$ by construction. At iteration k, let $e_k = (u_k, v_k)$ be the entering edge and $f_k = (u_k, w_k)$ the departing edge with $\dot{T}_k = \dot{T}_{k-1} + e_k - f_k$. We let X_k be the component of $\dot{T}_{k-1} - f_k$ containing u_k. Then the dual variable change will increase y_j by ε_k for $j \in X_k$, where ε_k is the amount of dual infeasibility of the pivot edge e_k. Let

$$Y_k = \bigcup_{j=1}^{k} X_k, \qquad S_k = S - Y_k, \qquad Y_k^+ = Y_k \cup v, \qquad Y_0 = \emptyset. \tag{13}$$

Y_k is the set of nodes which were subject to dual variable change during the first k pivots and S_k is the set of nodes in S with no dual variable change.

Notice that, at the beginning the only dual infeasible edges in G_{i+1}, if any, are among the external edges, i.e., in $\delta(S_0, Y_0^+) = \delta(S, v)$. The algorithm will maintain *the main invariant*: *the only dual infeasible edges are among* $\delta(S_k, Y_k^+)$. It turns out that $X_{k+1} \subset S_k$, hence X_{k+1} is disjoint from $X_1, X_2, \ldots, X_k$, and X_{k+1} contains at least one source node, namely u_{k+1}. Hence after at most l iterations, we have $S_k = \emptyset$, arriving at an optimal tree. Of course, a stage can be over earlier and some X_k may contain more than one source node. It should be noted that the crucial observation that $X_1, X_2, \ldots, X_k$ are disjoint is achieved by exploiting degeneracy, and three components of the algorithm; namely, SFT's, the sequentiality and Dantzig's rule are instrumental in achieving this result.

The algorithm will also maintain a *second invariant*: for $u \in S_k$ the path from u to v in $\dot{T}_k$ contains f_0 (and u_0), and for $u \in Y_k^+$ the path in $\dot{T}_k$ from u to v does not contain f_0. This invariant implies that *all fundamental cycles* $C(\dot{T}_{k-1}, e_k)$ *will contain* v *and* f_0. We will show that if f_0 is cut during the stage then the stage is over, i.e., G_{i+1} is solved optimally. We will also show that $f_{k+1} = (u_{k+1}, w_{k+1}) \in \gamma(S_k)$, if $f_{k+1} \neq f_0$. Since in any tree there is a unique path between any two nodes, it suffices to show that for $u \in Y_k$ the second invariant holds, for $u \in S_k$ it was true at the beginning of the stage and remains valid.

Let us introduce two more operations to make the presentation clearer: *shrinking* and *rerooting*. Let us shrink Y_k^+ into a single node, say, $\hat{v}$ and call the resulting tree $\hat{T}_k$. By shrinking $\dot{T}_k$ is transformed into $\hat{T}_k$. Clearly $\dot{T}_0 = \hat{T}_0$ and shrinking puts each $\dot{T}_k$ into the same form as $\dot{T}_0$. The main invariant states that only dual infeasible edges for the current stage are among $\delta(S_k, \hat{v})$. The second invariant simply states that $f_0 \in C(\hat{T}_{j-1}, e_j)$, $\forall j$. Subtrees of $\dot{T}_k$ induced by Y_k are connected to v by external pivot edges and among themselves by internal pivots. The subtree of $\dot{T}_k$ induced by S_k, which we denote by $\dot{T}_k[S_k]$, is connected to v only by f_0. Then, when $f_{k+1} = f_0$, $X_{k+1} = S_k$ and the stage is over. Similarly when $f_{k+1} \neq f_0$, $f_{k+1} \in \dot{T}_k$ implies $f_{k+1} \in \dot{T}_k[S_k] \subset \gamma(S_k)$ which in turn implies that $X_{k+1} \subset S_k$.

Let us denote by $\mathscr{T}_k$ the branching obtained from $\dot{T}_k$ making v as the root, i.e., rerooting $\dot{T}_k$ at v. Since $f_0 = (u_0, v)$, the second invariant simply states that S_k is the node set of one branch of $\dot{T}_k$, the branch which contains f_0. Clearly nodes in Y_k lie in different branches, each one corresponding to an external pivot edge. (We can

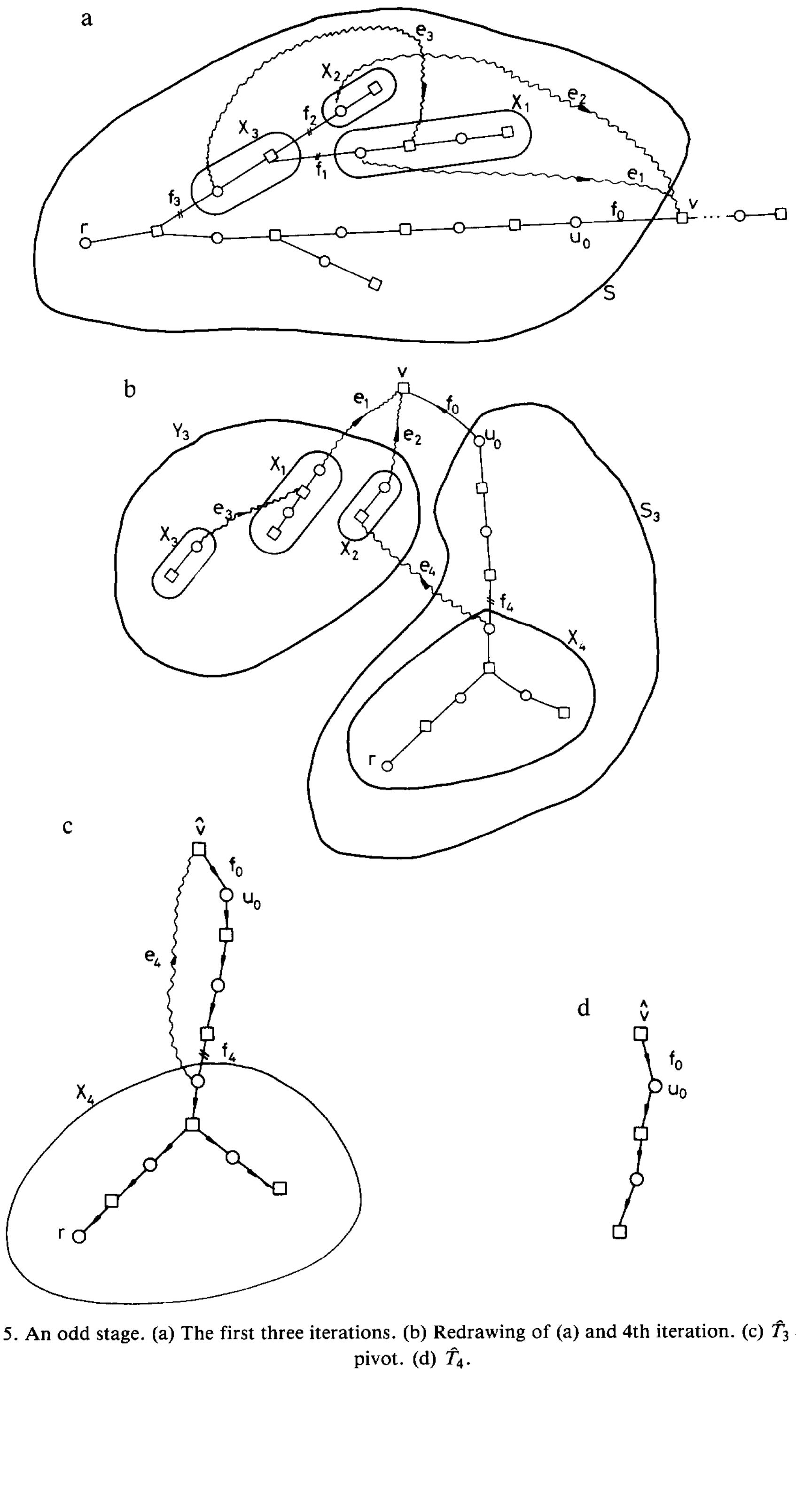

Fig. 5. An odd stage. (a) The first three iterations. (b) Redrawing of (a) and 4th iteration. (c) $\hat{T}_3$ and 4th pivot. (d) $\hat{T}_4$.

safely ignore nodes on P_{i+2} for this discussion.) Thus each pivot will move some set of nodes, namely X_k from the branch containing f_0 to another branch which is connected to v by an external edge. Moreover, the cut-set X_k will be the node-set of $\mathcal{T}_k(u_k)$ independent of whether f_k was forward or reverse.

Let us make some remarks concerning Fig. 5. Pivots e_1 and e_2 are external, they increase the degree of v in $\dot{T}$. The first three pivots are degenerate and their cut-sets are subtrees of current $\vec{T}$. Pivot e_4 is nondegenerate, and $r \in X_4$. X_1 and X_4 contain two and four source nodes respectively.

Our main technical lemma is the follwoing:

Lemma 2.3. *After the kth pivot we have*:

(i) $\varepsilon_1 \geq \varepsilon_2 \geq \cdots \geq \varepsilon_k$.

(ii) *The* X_j, $j \leq k$ *are disjoint.*

(iii) *If* $f_k = f_0$ *then the stage is over. Moreover* $f_0 \in C(\dot{T}_{j-1}, e_j)$, $\forall j \leq k$.

(iv) *The only dual infeasible edges, if any, are among*

$$\delta(S_k, Y_k^+).$$

(v) $\forall j \leq k$, e_j *connects* X_j *to either* $X_{\bar{j}}$ *with* $\bar{j} < j$ *or to* v.

Proof. By induction on k, the number of pivots at the current stage. Clearly, for $k=0$, only (iv) makes sense and it holds by definition of the stage. For $k=1$, clearly e_1 is an external edge, since all the internal edges are dual feasible at the beginning of the stage. X_1 is transferred from $S_0 = S$ to Y_1. Notice that all the edges in $\delta(X_1, v) \cup \delta(X_1, S_1)$ are dual feasible by Lemma 1.2. Clearly edges in $\gamma(X_1) \cup \gamma(S_1)$ are dual feasible, since they were dual feasible before the pivot and did not have any differential change. We may, however, create some internal dual infeasible edges entering into X_1, i.e., in $\delta(S_1, X_1)$. Hence we have the main invariant. Since e_1 is external, $u_1 \neq u_0$, and $f_1 \neq f_0$. Since nodes in X_1 are connected among themselves in $\dot{T}_1$ and connected to v via e_1 and f_0 and e_1 both connected to v we have the second invariant, for in any tree there is a unique path between any two nodes. Thus the lemma holds for $k=1$.

Edge e_2 can be external or internal; in any case $u_2 \in S_1$. Let us show (i). If e_2 is external then $v_2 = v$. Then the differential of e_2 did not change by the first pivot and it was dual infeasible before the first pivot. Then the fact that we have not chosen e_2 as the first edge implies $\varepsilon_2 \leq \varepsilon_1$. If e_2 is internal then $v_2 \in X_1$. Since it was dual feasible before the first pivot, the dual infeasibility introduced by the first pivot $\leq \varepsilon_2$. This shows $\varepsilon_2 \leq \varepsilon_1$ in both cases. If $f_2 = f_0$ then $X_2 = S_1$, and the stage is over, for all the edges in $\delta(S_1, Y_1^+)$ are dual feasible by Lemma 1.2 and since $S_2 = \emptyset$, the second pivot will not create any dual infeasible edges. If $f_2 \neq f_0$ then $u_2 \neq u_0$ and $w_2 = h(f_2)$ is a neighbour of u_2 (w_2 is either $p(u_2)$ or $mate(u_2)$). $u_2 \in S_1$ and w_2 is a neighbour of u_2 implies $w_2 \in S_1$ hence $f_2 \in \gamma(S_1)$. Thus $X_2 \subset S_1$ and disjoint from X_1. This proves (ii). We have already shown (v) in proving (i). The second part of (iii) and the second invariant follow from the observation that Y_2 and S_2 are connected to v by external pivot edges and f_0 respectively. Let us show (iv). Edges in $\gamma(S_2) \cup \gamma(X_1) \cup \gamma(X_2)$ are dual feasible simply because they were dual feasible at the beginning of the stage and their differential did not change. Edges in $\delta(X_1, Y_1^+ \cup S_2)$ are dual feasible by Lemma 1.2. To show that edges in $\delta(X_1, X_2)$ are dual feasible we use (i). Recall that they were dual feasible at the beginning of the stage and the change in differential is $\varepsilon_2 - \varepsilon_1 \leq 0$, so they remain dual feasible. This finishes the proof of (iii). Thus the lemma holds for $k=2$.

Now let us assume that the assertions of the lemma hold after the iteration k and we want to prove for the iteration $k+1$. Let $e_{k+1} = (u_{k+1}, v_{k+1})$ be the pivot edge. Let y, y' be the vectors of dual variables before pivot 1 and after pivot k and z, z' be the corresponding differentials respectively. By the induction hypothesis $u_{k+1} = t(e_{k+1}) \in S_k$, thus $y'_{u_{k+1}} = y_{u_{k+1}}$. For (i) we differentiate several cases. If $v_{k+1} = v$, i.e., e_{k+1} is external, then $y'_{v_{k+1}} = y_{v_{k+1}}$, hence dual infeasibility of e_{k+1} did not change during the first k iterations; and the fact that it has not been selected as the pivot edge at iteration k forces $\varepsilon_{k+1} \leq \varepsilon_k$ by Dantzig's rule. Otherwise e_{k+1} is internal. If $v_{k+1} = h(e_{k+1}) \in X_k$, then it was dual feasible before the pivot k, hence dual infeasibility introduced at pivot k is bounded by ε_k, i.e., $\varepsilon_{k+1} \leq \varepsilon_k$. If $v_{k+1} \in Y_j$, $j < k$, then it was dual infeasible before pivot k, and the fact that it was not chosen as the pivot edge at the iteration k implies that the dual infeasibility of $e_{k+1} \leq \varepsilon_k$. Thus we have $\varepsilon_{k+1} \leq \varepsilon_k$.

If $f_{k+1} = f_0$ (this could happen only if $u_{k+1} = u_0$), then $X_{k+1} = S_k$, and all the edges in $\delta(X_{k+1}, Y_k^+)$ are dual feasible by Lemma 1.2. In this case since $S_{k+1} = \emptyset$ the stage will be over once we prove (iv). Notice that $C(\dot{T}_k, e_{k+1})$ is the union of paths from u_{k+1} to v, v_{k+1} to v and e_{k+1}. The second part of (iii) follows from the induction hypothesis. To prove the second invariant it suffices to notice that nodes in X_{k+1} are connected by e_{k+1} to Y_k^+ hence to v, and paths from these nodes to v do not involve f_0.

To prove (ii) it suffices to show that $f_{k+1} = (u_{k+1}, w_{k+1}) \in \gamma(S_k)$. Since $u_{k+1} \in S_k$ and w_{k+1} is a neighbour of $u_{k+1} \neq u_0$, it follows that $w_{k+1} \in S_k$ implying $f_{k+1} \in \gamma(S_k)$.

(iv) Recall that each pivot will change only the differential of edges in the fundamental co-cycle: $D^+(\dot{T}_k, f_{k+1}) = \delta(X_{k+1}, S_{k+1} \cup Y_k^+)$ and $D^-(\dot{T}_k, f_{k+1}) = \delta(Y_k^+, X_{k+1}) \cup \delta(S_{k+1}, X_{k+1})$. Dual infeasibility of the edges in $\gamma(Y_k^+)$, $\gamma(S_{k+1})$ and $\gamma(X_{k+1})$ will not change, since either both or neither of the ends of these edges are subject to the same amount of dual variable change. For edges in $\delta(Y_{k+1}, S_{k+1})$ the dual variable change will not create any dual infeasibility because they were dual feasible with respect to y, and the differentials of such edges will decrease, since the dual variable change occurs at $t(e)$ for such an edge e. Edges in $\delta(X_{k+1}, S_{k+1} \cup Y_k^+)$, are dual feasible by Lemma 1.2. Since the symmetric difference of $\delta(S_k, Y_k^+)$ and $\delta(S_{k+1}, Y_{k+1}^+)$ is $\delta(X_{k+1}, Y_k^+) \cup \delta(S_{k+1}, X_{k+1})$, to prove (iv) it suffices to show that the edges in $\delta(Y_k, X_{k+1})$ are dual feasible. For this we use (i). Let $e = (u, w)$ be such an edge with $u \in X_j$, $w \in X_{k+1}$, $j \leq k$. Since e is internal, it was dual feasible with respect to y. But the change in the differential of e is

$\varepsilon_{k+1}-\varepsilon_j\leq 0$, implying that e is dual feasible after the pivot $k+1$. This proves (iv). We have already shown (v) in proving (i). □

Since each X_j contains at least one source (even) node, namely $t(e_j)$, and they are disjoint it follows that

Lemma 2.4. *For even i, passage from G_i to G_{i+1} can be done in l, the number of even nodes in $[i]$, pivots.*

Combining Lemmas 2.2 and 2.4 we have:

Theorem 2.5. *SNSA solves the assignment problem in $\frac{1}{2}n(n+3)-4$ simplex pivots.*

Proof. Processing of even nodes requires $n-2$ pivots. Processing of odd nodes requires $1+\sum_{j=3}^{n} j$ pivots. Note that when node 3 is added to the current graph, we can have at most one pivot. Summing up these numbers will give the above result. □

The following, which is interesting by itself, can be seen as a corollary of Lemma 2.3.

Lemma 2.6. *During a stage*
(i) *The pivot edges $e_1, e_2, \ldots, e_k$ will never leave the basis.*
(ii) *The cut-edges $f_1, f_2, \ldots, f_k$ will never re-enter the basis.*

Now, we will determine the maximum number of nondegenerate pivots during the course of the algorithm.

Lemma 2.7. *In an odd stage there is at most one nondegenerate pivot.*

Proof. Recall that by (v) of Lemma 1.1 a pivot e_k is nondegenerate iff $r\in X_k$, and by (ii) of Lemma 2.3 all X_k are disjoint. Thus root r can only be in one X_k, hence there is at most one nondegenerate pivot. □

Theorem 2.8. *SNSA can have at most $n-1$ nondegenerate simplex pivots.*

Proof. This follows from Lemmas 2.2 and 2.7. Note also that G_2 is already optimal. □

Remark. Theorem 2.8 can be seen as a proof of a variant of Hirsch conjecture [22] for the assignment problem.

Next we would like to determine the largest number of consecutive degenerate pivots in a stage, and then in the whole algorithm. Clearly in an even stage we can have only one degenerate pivot by Lemma 2.2.

Consider an odd stage. Let P be the unique path from r to v in the current tree at the beginning of stage v. Suppose we have l even nodes in P, say $\bar{u}_1, \bar{u}_2, \ldots, \bar{u}_l$, numbered from r towards v. Initially any pivot of the form $(\bar{u}_k, v)$ is a nondegenerate pivot. More generally, we have:

Lemma 2.9. (i) *The first pivot with entering edge $(\bar{u}_p, w)$, $p \le l$, $w \in Y_j^+$, for some j will be nondegenerate. Suppose it occurs as qth pivot, if it occurs.*

(ii) *Pivot edges e_k, $k < q$, if any, will be cross edges.*

(iii) *Pivot edges e_k, $k > q$, if any, will be reverse or cross.*

Proof. (i) Suppose first such pivot is $e_q = (\bar{u}_p, w)$. For such a pivot, the path from $\bar{u}_p$ to w will contain $\{\bar{u}_k \colon k \ge p\}$ and in particular edge f_0 (see Lemma 2.3) and node v, and the cut-edge f_q will be $(\bar{u}_p, mate(\bar{u}_p))$. Consequently, the cut-set X will contain the part of P from r to $\bar{u}_p$, hence $r \in X$ implying that the pivot is nondegenerate.

(ii) Consider the pivots before $e_k = (u, w)$, $k < q$. Clearly the $u \ne \bar{u}_k$, $k = 1, \ldots, l$. Then $u \in S \setminus P$, and $w \in Y_k^+$. Hence $\mathrm{NCA}(u, w) \notin \{u, w\}$ proving that e_k is a cross edge.

(iii) After the nondegenerate pivot r will be in Y^+. Pivots with head on the path between r and v will be reverse and others will be cross. □

As a corollary of Lemma 2.9 we have:

Lemma 2.10. *The maximum number of consecutive degenerate pivots in an odd stage v is bounded by $\frac{1}{2}(v+1) - l$.*

Proof. By Lemma 2.9, we can have only pivots with tails on $U \cap (S \setminus P)$. Clearly the number of such nodes is as given above. □

Since every path P from r to v, $v \ge 3$ odd, contains at least two even nodes, we have:

Theorem 2.11. *The maximum number of consecutive degenerate pivots is bounded by*

$$n - 2 + \sum_{j=3}^{n} (j-2) = \tfrac{1}{2}(n-2)(n+1).$$

Note that for the transshipment problem SFT's prevent cycling [19]. In fact, that was the main motivation for the development of strongly feasible trees. However, it is possible to have an exponential sequence of degenerate pivots, without cycling. This is known as stalling. Cunningham [20] introduced several pivot strategies, the best of which together with SFT's bounds the length of consecutive degenerate pivots, for the AP, by $n(n-1)$. Hung [30] used this particular pivot strategy to obtain his polynomial primal simplex algorithm. (Recall that Roohy-Laleh [39] exhibits a class of problems having an exponentially long sequence of nondegenerate pivots.) Our Theorem 2.11 reduces this bound by half. Notice that the difference between the bounds in Theorems 2.5 and 2.11 is $\leq 2n$.

3. Worst-case examples

Now, we would like to show that the bounds in Theorems 2.5, 2.8 and 2.11 are sharp. We first look at Theorems 2.5 and 2.8. We will construct a cost matrix C_n, one for each $n \geq 3$ with the following properties:

(i) C_n is integral.

(ii) $c_e = 0$, for $e \in T_0$.

(iii) For T_n^*, the optimal tree for C_n, we have $\bar{c}_e \neq 0$, $\forall e \in T_n^{*\perp}$. In other words, the reduced cost of the co-tree edges would be nonzero. This will imply that the optimal tree is unique among strongly feasible trees rooted at r.

(iv) C_{n+1} will be obtained from θC_n by appending one row and one column subject to (ii), where θ is a positive integer to be specified later. Furthermore, starting with T_0 the first $2n-1$ stages of the algorithm for C_{n+1} will yield identical pivots with C_n (except ε's will be multiplied by θ) and T_n^* augmented with a path of length 2 will be the tree obtained at the end of stage $2n-1$.

(v) Starting with T_n^*, C_{n+1} will require exactly $n+2$ pivots.

Let $w = 2n-1$, $u = 2n$, $v = 2n+1$, $T = T_n^*$, with u, v, w being nodes. We assume as the induction hypothesis that T is of the form in Fig. 6. Thus, the source nodes are $r, u_0, u_1, \ldots, u_k$, and sink nodes are $v_0, v_1, \ldots, u_k, w$ where $k = n-2$, and there is a path P containing the nodes r, w, u_0, v_0, u_k, v_k in that order. The rest of the tree consists of paths (w, u_i, v_i) for $i = 1, 2, \ldots, k-1$. Note that the numbering of u_i, v_i has no relation to initial numbering and is arbitrary except for the ordering induced by the path, P. By adding edges (w, u) and (u, v) we obtain a strongly feasible tree T'' for C_{n+1}. Then $T' = T'' + (u, v_0) - (u, w)$ will be the resulting tree after pivoting in the edge (u, v_0) in the even stage $2n$. Let T^- be the tree obtained by shrinking the path P', from r to v, i.e., the nodes r, w, u_0, v_0, u, v, into a single (odd) node, say r'. The tree T^- consists of paths (r', u_i, v_i), $i = 1, 2, \ldots, k$.

Now we will describe the last row and column of C_{n+1}. Let $\theta = 5$. Consider the optimal dual solution for θC_n. Instead of giving the last row of C_{n+1}, we will describe the reduced costs ($\bar{c}_e = y_{h(e)} - y_{t(e)} - c_e$) of the edges (u, v_i), for $i = 0, 1, \ldots, k$. Recall that $c(u, v) = c(u, w) = 0$. Let the reduced costs of the remaining edges be

$$\begin{aligned} &\bar{c}(u, v_0) = 1, \\ &\bar{c}(u, v_i) = -5, \quad i = 1, 2, \ldots, k. \end{aligned} \tag{14}$$

Consequently, the edge (u, v_0) will be the pivot edge and T' will be the new tree; the dual variables on the nodes u, v will be changed. Furthermore, we have $\bar{c}(u, w) = -1$. Let the reduced cost coefficients of the edges entering into v with respect to dual variables of T' be

$$\begin{aligned} &\bar{c}(r, v) = 4, \\ &\bar{c}(u_i, v) = 3, \quad 1 \le i \le k, \\ &\bar{c}(u_0, v) = 2. \end{aligned} \tag{15}$$

It is easy to see that in the odd stage $2n+1$ the first pivot will be the edge (r, v),

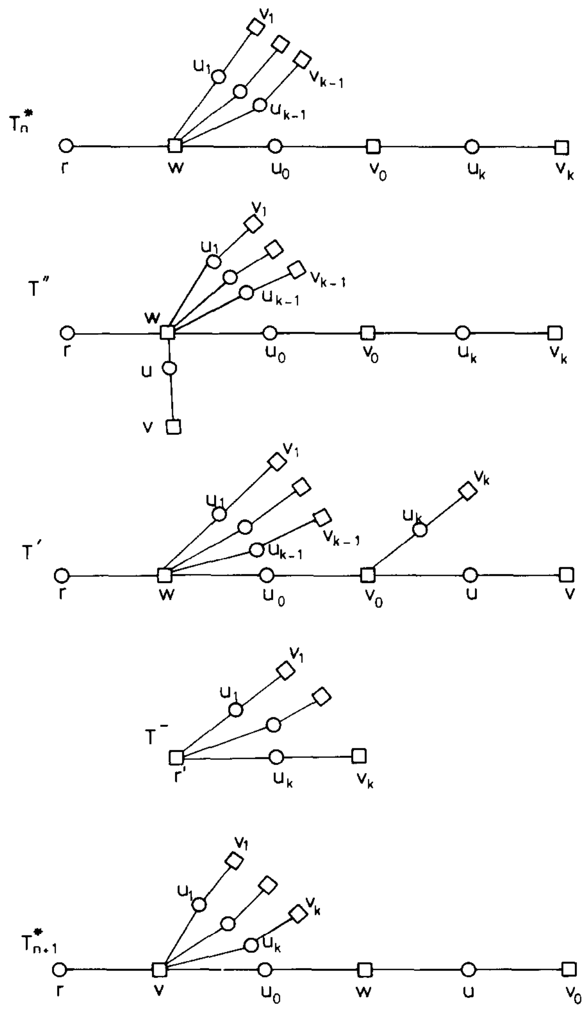

Fig. 6.

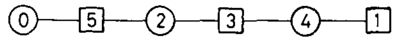

Fig. 7.

which will result in a nondegenerate pivot. The algorithm, then, will select edges (u_i, v), $1 \le i \le k$, in any order as pivot edges. Since nodes u_i, $1 \le i \le k$ are on different branches of T^-, the order in which these nodes are selected as the tail of pivot edges is immaterial. Up to now, we have a total of n pivots including the pivot (u, v_0). Clearly the next pivot will be (u_0, v) which will result in the edge (u, w) being dual infeasible with $\bar{c}(u, w) = 1$. Thus the edge (u, w) will be the last pivot.

We need to show that the above sequence of pivots is valid. By the induction hypothesis, $\bar{c}_e \le -1$, $\forall$ co-tree edge $e \in T_n^*$ for C_n. Multiplying C_n by θ guarantees that the reduced cost of co-tree edges is at most $-\theta$ for θC_n. Since the dual variable changes will be strictly less than θ, none of the co-tree edges of T_n^* will become dual infeasible. Moreover, the reduced cost of these edges will be ≤ -1. The last pivot edge (u, w) will have $\varepsilon = 1$, resulting in $\bar{c}(u, v) = -1$. This completes the proof of the inductive step.

To complete the proof it suffices to give a C_3 with the desired properties. One such matrix is

$$\begin{bmatrix} 0 & -1 & -4 \\ 0 & 0 & -2 \\ -1 & 0 & 0 \end{bmatrix}. \tag{16}$$

The resulting optimal tree is given in Fig. 7. The optimal matching is (0, 5), (2, 3), (4, 1) and it requires five pivots.

Note that, if desired, one can remove the nondeterminism of pivots. Let K be the ordered set $r, u_1, u_2, \ldots, u_k, u_0$. Starting with u_0, let $\bar{c}(u_0, v) = 2$, and increase the reduced cost of each edge $\bar{c}(s, v)$ by one for $s \in K$ in the reverse order, and let $\theta = \bar{c}(r, v) + 1$. The result will be a unique sequence of pivots induced by the order in K.

We have thus proved

Theorem 3.1. *The bounds in Theorems* 2.5 *and* 2.8 *are sharp.*

Next we would like to show that the bound in Theorem 2.11 is attainable. The construction is very similar to one given above. We start the induction with the following C_3:

$$\begin{bmatrix} 0 & 2 & 3 \\ 0 & 0 & 1 \\ -2 & 0 & 0 \end{bmatrix}. \tag{17}$$

for which T_3^* is given in Fig. 8.

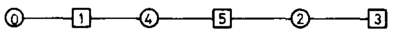

Fig. 8.

Let $(u_i, v_i) = (2i, 2i+1)$, $i = 0, 1, \dots, n-1$ be the edges of the matching in T_0, where $u_0 = r$. As the induction hypothesis we assume that T_n^* consists of a path $(u_0, v_0, u_{n-1}, v_{n-1})$ together with paths (v_{n-1}, u_i, v_i), for $1 \le i < n-1$ attached to v_{n-1} as shown in Fig. 9. Let $T = T_n^*$, $T'' = T + (u, w) + (u, v)$, $T' = T'' + (u, v_0) - (u, w)$ where u, v, w are defined as before. Let $\theta = 4$ and let us choose the reduced cost of edges (u, v_i) with respect to dual variables of T'' after multiplying C_n by θ as

$$\bar{c}(u, v_0) = 3, \qquad\qquad (18)$$
$$\bar{c}(u, v_i) = -4, \quad 1 \le i < n-1.$$

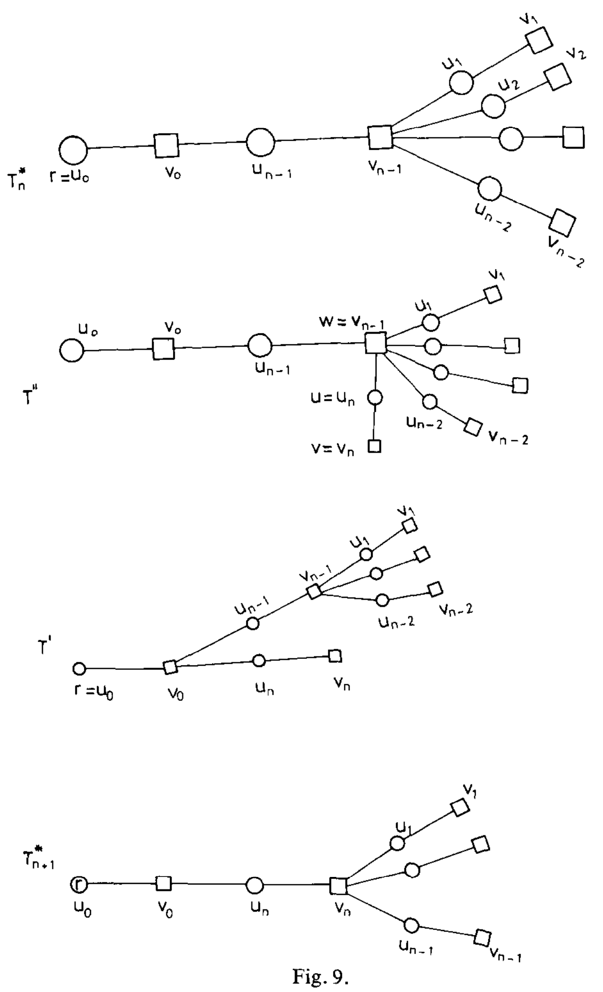

Fig. 9.

The reduced cost of the edges (u_i, v) with respect to dual variables of T' are chosen as

$$\begin{aligned} &\bar{c}(r, v) = -4, \\ &\bar{c}(u_i, v) = 2, \quad 1 \le i < n-1, \qquad (19) \\ &\bar{c}(u_{n-1}, v) = 1. \end{aligned}$$

Recall that $c(u, v) = 0$, and $\bar{c}(u, v) = 0$ since $(u, v) \in T'$. Then by Lemmas 2.1 and 2.10, it is easy to see that in passing from θC_n to C_{n+1} one will have exactly $1 + (n+1) - 2$ consecutive degenerate pivots, and T^*_{n+1} will be the unique optimal tree for C_{n+1} provided that T^*_n is the unique optimal tree for C_n (among SFT's rooted at r). Thus we have proved

Corollary 3.2. *The bound in Theorem* 2.11 *is attainable.*

4. Implementation and time complexity

It is well known that the primal simplex method performs well in practice and is at least competitive with other (polynomial) methods. We now show that properly implemented, our algorithm attains the same time complexity as with the state of the art algorithms.

First let us look at dense graphs; in particular when $m = n^2$, the complete bipartite graph. It is well known that an update of the basis and dual variables can be performed in $O(n)$ time using any of the algorithms in [7,14,41]. The difficulty arises in the selection of incoming edges. If we just look at any simplex pivot, the selection of incoming edges by Dantzig's rule, in fact any rule, will require $O(m)$ time in the worst case. The key to the analysis lies in looking at all the pivots in a stage simultaneously, instead of analyzing each pivot by itself. Clearly, we can restrict our attention to an odd stage, say the last stage. For the pivot k we need to determine

$$\max\{\bar{c}_e \colon e \in \delta(S_{k-1}, Y^+_{k-1})\}. \qquad (20)$$

The technique that we will employ is very similar to the technique used in the primal-dual algorithm. Following, say, [37], for each $x \in Q = U \cap S_{k-1}$ we define

$$s(x) = \max\{\bar{c}(x, y) \colon y \in Y^+_{k-1}\}, \qquad (21)$$

$$nb(x) = (x, s), \quad \text{if } \bar{c}(x, s) = s(x). \qquad (22)$$

$s(x)$ measures the largest reduced cost of the edges in the current graph that start at x, and $nb(x)$ keeps one such edge. Thus the maximum in (20) can be computed by finding $\max\{s(x) \colon x \in Q\}$, and if this maximum is ≤ 0, the stage is over. If this maximum is positive, then $nb(x)$ for a maximizing x is a pivot edge. After the kth pivot, we first set $Q \leftarrow Q \setminus (X_k \cap U)$, then update the $s(x)$ by computing the reduced cost of edges in $\delta(S_k, X_k)$. We initially start with $s(x) = \bar{c}(x, v)$. Since the X_k are disjoint, we need to compute the reduced cost of each edge only once. Thus, during an odd stage, updating of reduced costs will take $O(m)$ time. Since $s(x)$, $nb(x)$ facilitate the selection of incoming edge in $O(n)$ time, the total complexity of a stage is $O(n^2 + m)$, resulting in the total complexity of the algorithm being $O(n^3)$.

For sparse graphs we can do better. Our initial T_0 may not be feasible, or finding an initial tree may be nontrivial. In that case, we appeal to the big M method: that is we start with T_0, by adding artificial edges if necessary.

We assume an adjacency list representation of graphs: for every $x \in U$ we carry a list of edges $N^+(x)$ leaving the node x, and for $x \in V$ we carry a list of edges $N^-(x)$ entering the node x. We need a sublist of $N^+(x)$ and $N^-(x)$ at each stage. In addition to pointers necessary to manipulate the tree we carry an array $y(v)$ of dual variables, and $mate(v)$ of flow values (matching) for $v \in N$, i.e., if $\sigma : U \to V$ represents the current matching, then for $v \in U$, $mate(v) = \sigma(v)$ and for $v \in V$, $mate(v) = \sigma^{-1}(v)$.

In order to improve the time bound we need to utilize two new data structures: the *dynamic trees* of Sleator and Tarjan [40,44], and the *Fibonacci heaps* of Fredman and Tarjan [27]. We need the first to maintain the basis tree, and the second for selecting incoming edges.

Now we will describe the working of the dynamic tree data structure and its use in maintaining a tree basis. A basis tree is represented via $\vec{T}$. For $v \in N$, $p(v)$ is the parent of v, and the edge $(p(v), v)$ is directed from $p(v)$ to v in $\vec{T}$. The dynamic trees data structure allows the execution of the following primitives in $O(\log n)$ time: cut, link, NCA. To determine the leaving or cut-edge we use the NCA primitive. Let $e = (u, v)$ be the pivot edge. Then by Lemma 1.1, e is a forward edge (i.e., the pivot is nondegenerate) iff $u = \mathrm{NCA}(u, v)$. When $e = (u, v)$ is a forward edge, $f = (u, mate(u))$ is the cut-edge. Otherwise, the cut-edge is $f = (p(u), u)$. Once the cut-edge $(p(x), x)$ is determined, then the primitive, $cut(x)$ will result in (T_1, r) and (T_2, x), and then the primitive, $link(T_1, T_2, v)$ will give the required new tree. Thus each tree update can be done in $O(\log n)$ time. Update of the array *mate* can be done in $O(n)$ time. Since there is at most one nondegenerate pivot in a stage, all tree updates in a stage can be achieved in $O(n \log n)$ time.

We will update dual variables while updating $s(x)$, $x \in Q = U \cap S_k$. The Fibonacci heap of Fredman and Tarjan [27] is used to select the incoming edges via (20) and (21). Items in the heap will be $(s(x), nb(x))$, $x \in Q$. The key of an item is its $s(x)$ component. We will describe the heap in terms of the minimum of keys rather than the maximum. A Fibonacci heap supports the following operations in $O(1)$ amortized time: (i) make-heap, (ii) update key of any item, (iii) add an item into the heap with a known key, (iv) find the minimum key of items in the heap, and delete any item, including the min-key item, in $O(\log n)$ amortized time. Notice that a pointer to the position of the item in the heap is maintained to achieve these bounds. At the beginning of each odd stage we apply make-heap, compute $s(x) = -\bar{c}_e = c_e - y_v + y_x$, add

$s(x)$ to the heap for each external edge $e=(x,v)$. At the kth pivot we need to perform the following operations: find the item x with minimum key, i.e., $s(x)$ whose $nb(x)$ gives the pivot edge, determine cut-edge f_k and cut-set X_k. We then traverse set X_k, make dual variable changes, update sets Y_k, S_k and Q, apply delete-item for source nodes in X_k, and put sink nodes in X_k in a list. We then process nodes in the list and update the $s(x)$ by examining edges in $N^-(v)$ for v in the list. Thus during an odd stage we apply one make-heap, at most m update or add-item and n find-min and n delete-item operations. Clearly all heap operations during an odd stage will require $O(n \log n + m)$ time, and dual variable updates will require $O(n)$ time. Since each tree update can be done in $O(\log n)$ time, the total work required in a stage will be bounded by $O(n \log n + m)$. Thus the overall bound is $O(n^2 \log n + nm)$. This is the same as for the best known algorithms, the successive path algorithm of Fredman and Tarjan and the dual simplex algorithm of Balinski and Goldfarb [27,29].

In fact, we can obtain the same bound without using dynamic trees. Notice that using the classical triple labels *parent*, *first child*, *next sibling* one can cut and link any tree in constant time. The main difficulty in these data structures is the determination of cut-edge. In dynamic trees we used the NCA primitive to determine whether a pivot edge is degenerate or not. But we know that, during an odd stage i, we have at most one nondegenerate pivot edge and by Lemma 2.9 the first and only nondegenerate edge must originate from a source node in the path P from r to i. Let $Z=P \cap U$. Then if the tail of $nb(x)$ lies in Z then the pivot edge $e=nb(x)$ is nondegenerate hence the cut-edge is $f=(x, mate(x))$. Otherwise the cut-edge is $f=(p(x),x)$. We also know that after the first nondegenerate pivot, the remaining pivots in that stage are all degenerate. Thus the determination of cut-edges takes $O(n)$ time during a stage. In order to perform cut and link operations in constant time it suffices to maintain (next) sibling pointers in a doubly linked circular list. For this, we maintain *parent*, *first*, *left* (sibling) and *right* (sibling) pointers. But, then, we do not need the array *mate*. For $u \in U$, $mate(u)=first(u)$ and for $v \in V$, $mate(v)=parent(v)$. Thus all basis update and dual variable changes in a stage can be done in $O(n)$ time.

Acknowledgement

This paper is based on [3] and went through considerable changes. I would like to thank the referees for their very thorough reports on earlier versions of this paper and their thoughtful suggestions for improving it.

References

[1] H. Achatz, P. Kleinschmidt and K. Paparrizos, A dual forest algorithm for the assignment problem, Tech. Rept., Universität Passau (1989).

[2] M. Akgül, Topics in Relaxation and Ellipsoidal Methods, Research Notes in Mathematics 97 (Pitman, London, 1984).

[3] M. Akgül, A genuinely polynomial primal simplex algorithm for the assignment problem, Tech. Rept., North Carolina State University, Raleigh, NC (1985); also: Working Paper IEOR 87-07, Bilkent University, Ankara (1987).

[4] M. Akgül, A sequential dual simplex algorithm for the linear assignment problem, Oper. Res. Lett. 7 (1988) 155–158.

[5] M. Akgül, Shortest paths and the simplex method, Working Paper IEOR 88-04, Bilkent University, Ankara (1988).

[6] M. Akgül and O. Ekin, A dual feasible forest algorithm for the linear assignment problem, Res. Opér. 25 (1991) 403–411.

[7] A.I. Ali, R.V. Helgason, J.L. Kennington and H.S. Lall, Primal simplex network codes: state-of-the-art implementation technology, Networks 8 (1978) 315–339.

[8] M.L. Balinski, Signature method for the assignment problem, Oper. Res. 33 (1985) 527–536; also: Presented at Mathematical Programming Symposium, Bonn (1982).

[9] M.L. Balinski, A competitive (dual) simplex method for the assignment problem, Math. Programming 34 (1986) 125–141.

[10] M.L. Balinski and R.E. Gomory, A primal method for the assignment and transportation problems, Management Sci. 10 (1964) 578–598.

[11] R.S. Barr, F. Glover and D. Klingman, The alternating basis algorithm for assignment problems, Math. Programming 13 (1977) 1–13.

[12] D. Bertsekas, A new algorithm for the assignment problem, Math. Programming 21 (1981) 152–157.

[13] R.B. Bixby, Matroids and operations research, in: H.J. Greenberg et al., eds., Advanced Techniques in the Practice of Operations Research (North-Holland, Amsterdam, 1982) 333–459.

[14] G.H. Bradley, G.G. Brown and G.W. Graves, Design and implementation of large scale primal transshipment algorithms, Management Sci. 24 (1977) 1–34.

[15] R.E. Burkard, Travelling salesman and assignment problems: a survey, in: Annals of Discrete Mathematics 4 (North-Holland, Amsterdam, 1979) 193–215.

[16] R.E. Burkard, W. Hahn and W. Zimmerman, An algebraic approach to assignment problems, Math. Programming 12 (1977) 318–327.

[17] G. Carpaneto and P. Toth, Primal-dual algorithms for the assignment problem, Discrete Appl. Math. 18 (1987) 137–153.

[18] V. Chvátal, Linear Programming (Freeman, San Francisco, CA, 1983).

[19] W.H. Cunningham, A network simplex method, Math. Programming 11 (1976) 105–116.

[20] W.H. Cunningham, Theoretical properties of the network simplex method, Math. Oper. Res. 4 (1979) 196–208.

[21] W.H. Cunningham and A.B. Marsh, A primal algorithm for optimum matching, Math. Programming Stud. 8 (1978) 50–72.

[22] G.B. Dantzig, Linear Programming and Extensions (Princeton University Press, Princeton, NJ, 1963).

[23] U. Derigs, The shortest augmenting path method for solving assignment problems—motivation and computational experience, Ann. Oper. Res. 4 (1985/6) 57–102.

[24] E.A. Dinic and M.A. Kronrad, An algorithm for the solution of the assignment problem, Soviet Math. Dokl. 10 (1969) 1324–1326.

[25] J. Edmonds and R.M. Karp, Theoretical improvements in algorithmic efficiency for network flow problems, J. ACM 19 (1972) 248–264.

[26] L.R. Ford and D.R. Fulkerson, Flows in Networks (Princeton University Press, Princeton, NJ, 1962).

[27] M.L. Fredman and R.E. Tarjan, Fibonacci heaps and their uses in improved network optimization algorithms, J. ACM 34 (1987) 596–615; also in: Proceedings 25th FOCS (1984) 338–346.

[28] F. Glover and D. Karney, Implementation and computational comparisons of primal, dual, primal-dual codes for minimum cost network flow problems, Networks 4 (1977) 191–212.

[29] D. Goldfarb, Efficient dual simplex algorithms for the assignment problem, Math. Programming 37 (1985) 187–203.

[30] M.S. Hung, A polynomial simplex method for the assignment problem, Oper. Res. 31 (1983) 595–600.

[31] M.S. Hung and W.O. Rom, Solving the assignment problem by relaxation, Oper. Res. 27 (1980) 969–982.

[32] E.L. Johnson, Flows in networks, in: J.J. Moder and S.E. Elmaghraby, eds., Handbook of Operations Research (Van Nostrand Reinhold, Princeton, NJ, 1978).

[33] R. Jonker and A. Volgenant, A shortest path algorithm for dense and sparse linear assignment problems, Computing 38 (1987) 325–340.

[34] H.W. Kuhn, The hungarian method for the assignment problem, Naval Res. Logist. Quart. 2 (1955) 83–97.

[35] J. Munkres, Algorithms for the assignment and transporation problems, SIAM J. Comput. 5 (1957) 32–38.

[36] J.B. Orlin, On the simplex algorithm for networks and generalized networks, Math. Programming Stud. 24 (1985) 166–178.

[37] C. Papadimitriou and K. Steiglitz, Combinatorial Optimization: Algorithms & Complexity (Prentice-Hall, Englewood Cliffts, NJ, 1982).

[38] R.T. Rockafellar, Network Flows and Monotropic Optimization (Wiley, New York, 1984).

[39] Roohy-Laleh, Improvements to the theoretical efficiency of the simplex method, PhD thesis, University of Charleton, Ottawa, Ont. (1981); also: Dissertation Abstract International 43 (1982) 448B.

[40] D.D. Sleator and R.E. Tarjan, A data structure for dynamic trees, J. Comput. System Sci. 26 (1983) 362–391.

[41] V. Srinivasan and G.L. Thompson, Accelerated algorithms for labelling and relabelling of trees, with applications to distribution problems, J. ACM 19 (1972) 712–726.

[42] V. Srinivasan and G.L. Thompson, Benefit-cost analysis for coding techniques for the primal transportation problem, J. ACM 20 (1973) 184–213.

[43] V. Srinivasan and G.L. Thompson, Cost operator algorithms for the transportation problem, Math. Programming 12 (1977) 372–391.

[44] R.E. Tarjan, Data Structures and Network Algorithms (SIAM, Philadelpha, PA, 1983).

[45] G.L. Thompson, A recursive method for the assignment problems, in: Annals of Discrete Mathematics 11 (North-Holland, Amsterdam, 1981) 319–343.

[46] N. Tomizawa, On some techniques useful for solution of transportation network problems, Networks 1 (1972) 173–194.