Mathematical Programming 68 (1995) 1–13

A strongly polynomial algorithm for the transportation problem

P. Kleinschmidt [a,*], H. Schannath [b]

[a] *Universität Passau, Wirtschaftswissenschaftliche Fakultät, Innstrasse 29, D-94030 Passau, Germany*
[b] *Westdeutsche Landesbank, Mathematische Beratung, D-40217 Düsseldorf, Germany*

Received 10 August 1992; revised manuscript received 31 May 1994

Abstract

For the (linear) transportation problem with m supply nodes, n demand nodes and k feasible arcs we describe an algorithm which runs in time proportional to $m \log m(k+n \log n)$ (assuming w.l.o.g. $m \geqslant n$). The algorithm uses excess scaling. The complexity bound is a slight improvement over the bound achieved by an application of a min-cost-flow algorithm of Orlin to the transportation problem.

Keywords: Transportation problem; Strongly polynomial algorithm; Excess scaling

1. Introduction

Throughout this paper let $I := \{1, \ldots, m\}$, $J := \{1, \ldots, n\}$ for some positive integers m and n. Let E be some subset of $I \times J$ and $k := |E|$. For $e = (i, j) \in E$ we set $e^- := i$ and $e^+ := j$. For $i \in I$ and $j \in J$ let a_i and b_j be given positive integers (the "supply" of i and the "demand" of j, respectively). Assume that $\sum_{i \in I} a_i = \sum_{j \in J} b_j$. Let $c: E \to \mathbb{Z}$ be a cost function. Then the classical transportation problem (TP for short) in its sparse version can be formulated as follows.

Find some integer function $x: E \to \mathbb{Z}^+$ such that

$$\sum_{\substack{e \in E \\ e^- = i}} x(e) = a_i, \quad i \in I, \tag{1.1}$$

* Corresponding author. Research supported in part by grant no. I-84-095.06/88 of the German–Israeli-Foundation for Scientific Research and Development.

SSDI 0025-5610(94)00041-Q

$$\sum_{\substack{e \in E \\ e^+ = j}} x(e) = b_j, \quad j \in J, \tag{1.2}$$

and such that $\sum_{e \in E} c(e)x(e)$ is minimal.

The dual of the LP-relaxation of TP (DTP for short) can be formulated as:

$$\max \sum_{i \in I} a_i u_i + \sum_{j \in J} b_j v_j$$

$$\text{subject to} \quad u_i + v_j \leqslant c((i,j)), \quad (i,j) \in E. \tag{1.3}$$

The function x can be viewed as a flow on a directed graph G whose arc set is E and whose node set is $I \cup J$. We call the elements of I and J the *row nodes* and *column nodes* of G, respectively. Accordingly, we call the dual variables u_i and v_j *row variables* and *column variables*, respectively. For notational convenience we sometimes write x_{ij} and c_{ij} instead of $x((i,j))$ and $c((i,j))$. For the vector consisting of the dual variables u_i and v_j we write (u, v) for short. W.l.o.g. we assume $m \geqslant n$ throughout. It is well known that a pair $(x, (u, v))$, where x is feasible for TP and (u, v) is feasible for DTP, is optimal provided $x_{ij}(c_{ij} - u_i - v_j) = 0$ for all $(i,j) \in E$. This is called complementary slackness. A basic feasible solution of DTP can be represented by a *dual feasible tree* T, i.e. T is a spanning tree of the complete bipartite graph with node set $I \cup J$ for which there is a dual feasible (u, v) with the property that $u_i + v_j = c_{ij}$ whenever (i,j) is an edge of T. A dual feasible tree corresponds to an optimal solution of TP if it carries a primal feasible flow. The basic idea of a non-polynomial version of our algorithm is to generate a sequence $T_1, \ldots, T_l$ of dual feasible trees which carry flows $x^1, \ldots, x^l$ such that property (1.1) is always valid, i.e. the full supplies are always shipped from I to J. The final x^l will also satisfy (1.2) and hence be optimal. Our algorithm can be viewed as a dual algorithm which relaxes (1.2).

The strongly polynomial algorithm "STP" will be described in Section 2. It uses the idea of excess scaling introduced by Orlin [7] (see also the survey [4]). The algorithm runs in $O(m \log m(k + n \log n))$ time. This is a slight improvement over the bound achieved by Orlin's algorithm when the latter is applied to TP. In Orlin's algorithm our term $n \log n$ has to be replaced by $m \log m$. So our algorithm has theoretical advantages when the numbers of row nodes and column nodes differ substantially. The improvement is achieved by exploiting the unbalanced bipartite structure for the shortest path computations occurring in such algorithms. The non strongly polynomial algorithm in [1] also takes advantage of the unbalanced bipartite structure when applied to TP. When applied to the assignment problem, both algorithms are identical to an algorithm of Hung and Rom [5]. In [6] it has been shown that the latter algorithm is equivalent to an algorithm of Balinski [2]. In this sense our strongly polynomial algorithm seems to be the natural extension of Balinski's signature method which is based on the fact that the Hirsch-conjecture is true for dual transportation polyhedra.

In Section 3 we prove the correctness and the complexity bound.

We would like to express our deep thanks to Jim Orlin who suggested a substantial

improvement of an earlier draft of this paper. Due to his suggestions we can avoid some complications which arise in a related algorithm of the second author [8].

2. The strongly polynomial algorithm STP

For a better understanding of the strongly polynomial algorithm STP it is worthwhile to briefly describe some ideas of a non-polynomial version (NTP). We first introduce some more notation. With respect to some given flow we define the excess $\bar{a}_i$ of a row node and the excess $\bar{b}_j$ of a column node as:

$$\bar{a}_i := a_i - \sum_{\substack{e \in E \\ e^- = i}} x(e)\,, \quad i = 1, \ldots, m\,,$$

$$\bar{b}_j := \sum_{\substack{e \in E \\ e^+ = j}} x(e) - b_j\,, \quad j = 1, \ldots, n\,.$$

The *reduced costs* with respect to some dual feasible (u, v) are defined by $\bar{c}_{ij} := c_{ij} - u_i - v_j$. We define $J_S := \{j \in J \mid \bar{b}_j > 0\}$ (the set of "surplus" nodes) and $J_D := \{j \in J \mid \bar{b}_j < 0\}$ (the set of deficit nodes). In NTP, for each $i \in I$ we determine some $j(i) \in J$ such that $c_{ij(i)} = \min\{c_{ij} \mid j \in J\}$; we set $x_{ij(i)} := a_i$, $u_i := c_{ij(i)}$ and $T := T \cup (i, j(i))$. All other x_{ij} and all v_j are set to zero. This way, we make sure that (1.1) is satisfied, i.e. all $\bar{a}_i$ are zero. If at this stage we have $J_S = \emptyset$ then $J_D = \emptyset$, too, and (1.2) holds as well. This means that we have already reached optimality. Observe that the way we defined the u_i and v_j we have a dual feasible (u, v). If $J_S \neq \emptyset$, we apply a procedure "Maketree" which adjoins further edges to T and turns T into a dual feasible tree. At the beginning of "Maketree" T will always consist of a collection of trees some of which may be isolated column nodes. All arcs of T initially carry some positive flow. For $j \in J$ we call T_j the component of T which contains j or, if j is isolated, T_j is j itself. Then "Maketree" works as follows:

```
procedure Maketree;
begin
choose some r∈J_S (the root); set T̄ = T_r;
while T̄ is not a spanning tree do begin
  find ε := min{c̄_ij | i∈T̄∩I, j∈J\T̄}  (= c̄_{i*j*});
  for all i, j∈T_{j*} do begin v_j := v_j + ε; u_i := u_i := u_i − ε end;
  T̄ := T̄ ∪ T_{j*} ∪ (i*, j*);
  end;
T := T̄;
end;
```

It is clear that T is a spanning tree. Dual feasibility follows from the fact that we have dual feasibility at the start of "Maketree" and from the fact that the ϵ's which occur in "Make-

tree'' are nondecreasing. The latter is an easy consequence of the fact that the sets $\bar{T} \cap I$ and $J \setminus \bar{T}$ which occur in the computation of ϵ are growing and shrinking, respectively. As the ingredients of this are essentially already contained in [5] we do not repeat the arguments. The total work for ''Maketree'' is $O(k+n \log n)$. This bound can be obtained if we use a Fibonacci-heap (F-heap for short) as a data structure for the computation of ϵ. Fibonacci-heaps introduced in [3] have meanwhile become a standard tool in this context and so we restrict ourselves only to some essential points here.

Initially, for each T_l (except T_r) as defined above we keep an item of our F-heap. The item of T_l is labelled by the arc (i^*, j^*) for which

$$\epsilon_l := \min\{\bar{c}_{ij} \mid i \in \bar{T} \cap I, j \in T_l \cap J\}$$

is achieved and ϵ_l is called the *key* of the item.

One could also keep an item for every node $j \in J \setminus \bar{T}$. This is usually done in other network flow algorithms. However, the special structure of our algorithm allows us to maintain the smaller F-heap. This may be useful for an implementation even though the theoretical bound for the size of both F-heaps is $O(n)$. The F-heap allows us to find each ϵ in ''Maketree'' in constant (amortized) time.

When some T_j is added to $\bar{T}$ we can delete the corresponding item in $O(\log n)$ time. As this can happen at most $n-1$ times we spend $O(n \log n)$ time with deletions. When we compute the labels and keys of the items we have to access every arc only once (again due to the fact that $\bar{T} \cap I$ is growing). So the total time spent on accesses is $O(k)$. Recomputing the key of some item can result in a decrease-key operation caused by an arc that was not considered before. As every decrease-key operation takes constant amortized time with F-heaps this results in $O(k)$ work. This gives the desired bound. Hence we have the following:

Lemma 1. *The procedure ''Maketree'' runs in $O(k+n \log n)$ (amortized) time.*

The fact that our F-heaps have size $O(n)$ (also in the strongly polynomial algorithms STP) is crucial for our improvement of the bound in Orlin's algorithm. The F-heap of that algorithm may have $O(n+m) = O(m)$ items.

After the construction of a dual feasible tree we apply a procedure ''Augment'' which will reduce the quantity $\sum_{j \in J} \bar{b}_j$. In ''Augment'' we pick some $t \in J_D$ (note that $J_D \neq \emptyset$ if $J_S \neq \emptyset$) and determine the unique path P from r to t in T ignoring directions. Taking the arc of P containing r as the first one, we call the arcs of P odd or even. Observe that the arcs with flow zero which were added in ''Maketree'' are by construction even and hence all odd arcs carry positive flow. We compute the positive number Δ as the minimum of the three quantities $\bar{b}_r$, $|\bar{b}_t|$ and $\min\{x_{ij} \mid (i,j) \in P \text{ odd}\}$, add it to all x_{ij} where $(i,j) \in P$ even and subtract it from the x_{ij} where $(i,j) \in P$ odd.

The main loop of NTP is as follows:

```
while J_S ≠ ∅ do begin
  Maketree;
  Augment;
end;
```

Note that "Augment" preserves dual feasibility and the fact that all $\bar{a}_i$ are zero. Also x_{ij} is nonzero only if $(i, j) \in T$. As Δ is positive, the quantity $\sum_{\bar{b}_j > 0} \bar{b}_j$ is strictly decreasing after every call of "Augment". Hence the algorithm terminates. As $\Delta \leqslant \bar{b}_r$ and $\Delta \leqslant |\bar{b}_t|$, the sets J_S and J_D are shrinking (not necessarily strictly). It is clear that the algorithm stops at optimality. Also it is easy to see that its complexity is $O(k + n \log n)$ times the number of augmentations. The latter however may be nonpolynomial in m.

We have not discussed the possibility that the arcs to be determined in "Maketree" do not exist. This can be settled by using artificial arcs with prohibitive costs as incoming arcs. The costs could be chosen to be $1 + \max_{(i,j) \in E} c_{ij} \cdot \sum_{i=1}^{m} a_i$. If such an arc has positive flow in the final solution the problem is infeasible. Only those artificial arcs that are in T have to be stored in the data structure. As there are at most $n + m - 1$ such arcs, this does not change the statements. The main change in the strongly polynomial algorithm STP is the excess-scaling which guarantees that the Δ which occurs in the procedure "Augment" of the last section is "large" enough. The sets $J_S(\Delta)$ and $J_D(\Delta)$ are parametrized analogues of the sets J_S and J_D in STP. They will be defined below. The structure of the algorithms STP is as follows:

```
algorithm STP;
Δ := min{max_{i∈I} a_i, max_{j∈J} b_j};
Initialize;
while Δ ⩾ 1 do begin
  Augmentrows;
  Contract;
  Skipphase;
  while J_S(Δ) ≠ ∅ and J_D(Δ) ≠ ∅ do begin
    Maketree;
    Augment;
    end;
  Δ := Δ/2;
  end;
```

Throughout we make the assumptions that the problem is feasible, that all considered arcs exist in E and that Δ is even at all times. These assumptions are made for expository reasons only. The problem of feasibility and nonexisting arcs can be treated as in the non-polynomial version. Problems with the parity of Δ can be settled as in [7] by multiplications of the flows, excesses and capacities by 2. At all times, it will be evident that the flows on all arcs are integer multiples of Δ. We will use this fact at various places without further mention.

We now describe the algorithm in more detail.

procedure Initialize;
begin
for all $i \in I$ do $\bar{a}_i := a_i$; for all $j \in J$ do $\bar{b}_j := -b_j$;
$I(\Delta) := \{i \in I \mid \bar{a}_i \geqslant \Delta\}$; $I' := I(\Delta)$; $J' := J$;
for all $j \in J$ do $v_j := 0$; for all $(i,j) \in E$ do $x_{ij} := 0$;
$T := \emptyset$;
end;

Observe that it follows from the definition of Δ that $I(\Delta)$ is nonempty. The variables u_i will be initialized in the procedure "Augmentrows".

While in the non-polynomial version the set $I^* := \{i \in I \mid \bar{a}_i > 0\}$ was empty immediately after the initialization, in the polynomial version we will make $I(\Delta)$ empty in the procedure "Augmentrows", i.e. we will always maintain the property $0 \leqslant \bar{a}_i \leqslant a_i$ and $\bar{a}_i < \Delta$.

I' will be the "working set" of row nodes in T. A row node will enter T only if its excess exceeds Δ for the first time. J' is the set of column nodes initially. However, in the contraction process of the procedure "Contract", J' may change.

The main differences to the non-polynomial version are Δ-scaling and edge contraction. For the Δ-scaling, the sets J_D, J_S and I^* are parametrized by Δ. The "row augmentations" are performed as follows:

procedure Augmentrows;
begin
$I(\Delta) := \{i \in I \mid \bar{a}_i \geqslant \Delta\}$; $I' := I(\Delta) \cup I'$;
for all $i \in I'$ which are incident to an arc of T do begin
choose an arc $(i,j) \in T$; $x_{ij} := x_{ij} + \Delta$;
$\bar{a}_i := \bar{a}_i - \Delta$; $\bar{b}_j := \bar{b}_j + \Delta$;
end;
for all $i \in I'$ which are not incident to an arc of T do begin
determine $c_{ij*} := \min\{c_{ij} \mid (i,j) \in E\}$;
$u_i := c_{ij*}$; $h := \max\{\alpha \in \mathbb{Z} \mid \alpha\Delta \leqslant \bar{a}_i\}$;
$x_{ij*} := h\Delta$; $\bar{a}_i := \bar{a}_i - x_{ij*}$; $\bar{b}_{j*} := \bar{b}_{j*} + x_{ij*}$;
$T := T \cup (i, j^*)$;
end;
end;

At this point $I(\Delta)$ is empty. We call the flow changes which occur in "Augmentrows" "row augmentations" in contrast to the augmentations which will be performed in "Augment". Note that the way "Augmentrows" is described it appears as if I' can only grow. This is true if no contractions occur. A contraction may cause a node to be deleted from I'.

```
procedure Contract;
begin
for all (i, j) ∈ T with x_ij > 4(n+m)Δ do begin
   contract the edge (i, j) into a new column node v;
   update I', J' and T accordingly;
   (remove i and j from I' or J';
   add v to J'; in edges incident to i or j
   replace i or j by v);
   if i ∈ I' then b_v := b_j − a_i and b̄_v := ā_i + b̄_j;
   if i ∈ J' then b_v := b_i + b_j and b̄_v := b̄_i + b̄_j;
   end;
end;
```

The contraction is an ordinary edge contraction on T. We keep the directions and the flows of the arcs in T. The bipartite structure of column and row nodes will be destroyed because after a contraction there may be arcs between column nodes (we keep calling I' and J' row and column nodes, respectively and we still use a_v ($\bar{a}_v$) or b_v ($\bar{b}_v$) for the capacities (excesses) of row and column nodes). Observe that b_v may be negative for a contracted node $v \in J'$. For expository reasons we chose to preserve the "uncontracted" data of T in a separate data structure because this will allow us to simplify the explanation of dual updates and augmentations.

The following sets have to be updated at the beginning of each while-block:

$$J_S(\Delta) := \{j \in J' \mid \bar{b}_j \geqslant \Delta\} \cup \{j \in J' \mid \bar{b}_j \geqslant \tfrac{1}{2}\Delta,\ b_j > 0\},$$

$$J_D(\Delta) := \{j \in J' \mid \bar{b}_j \leqslant -\Delta\} \cup \{j \in J' \mid \bar{b}_j \leqslant -\tfrac{1}{2}\Delta,\ b_j < 0\}.$$

The following procedure guarantees that no "unnecessary" Δ-phases are considered.

```
procedure Skipphase;
begin
if x_ij = 0 for all (i, j) ∈ T and J_S(Δ) ∪ J_D(Δ) = ∅ then begin
   set Δ := max{(ā_i, i ∈ I uncontracted), (|b̄_j|, j ∈ J')};
   start a new Δ-phase; (i.e. start the outer while-loop of STP with the new Δ)
   end;
end;
```

If no contractions are performed, the procedures "Maketree" and "Augment" are identical to the corresponding procedures in the non-polynomial version. Otherwise there are a few differences.

In "Maketree" we choose some $r \in J_S(\Delta)$ which may now be a contracted new node. The process of building up the tree is then performed on the "blown up" tree exactly as in the non-polynomial version. The first starting $\bar{T}$ is the blown up component of T which contains r and "I" has to be replaced by "I"'. The incoming arcs computed in "Maketree" are added both to the contracted tree T and its uncontracted counterpart. After the construc-

tion of T in "Maketree" let P be a path (ignoring directions) starting from r to any other column node in J'. Observe that the incoming arcs which lie on P are by construction directed "away from r".

```
procedure Augment;
begin
choose some t∈J_D(Δ);
compute the "augmenting" path P in T from r to t ignoring directions;
for all arcs e∈P do
  if e is directed away from r then x(e) := x(e) + Δ
    else x(e) := x(e) − Δ; (such arcs have by construction a positive flow)
b̄_r := b̄_r − Δ; b̄_t := b̄_t + Δ;
delete all arcs with x(e) = 0 from T;
update J_D(Δ) and J_S(Δ);
end;
```

Observe that from the way the excesses are updated we always have

$$\bar{b}_j = \sum_{(i,j)\in T} x_{ij} - \sum_{(j,i)\in T} x_{ji} - b_j\,,$$

$$\bar{a}_i = a_i - \sum_{(i,j)\in T} x_{ij} \quad \text{and} \quad \sum_{\substack{i\in I\\ i \text{ uncontracted}}} \bar{a}_i + \sum_{j\in J'} \bar{b}_j = 0\,.$$

This concludes the description of the algorithm. By an *augmentation* we mean a call of "Augment".

3. Complexity and correctness

The algorithm of Section 3 is quite similar to Orlin's algorithm [7] for the transshipment problem. The main difference is that we maintain the different roles of the row nodes and column nodes throughout. This allows us to compute the incoming arcs with Fibonacci-heaps whose size is bounded by n rather than $n+m$. Another additional feature of our algorithm are the row augmentations which also imply some changes in the complexity proof. In the sequel we will restrict ourselves to those aspects of the complexity proof which contain essential differences to Orlin's arguments. Also we will sometimes omit the discussion of all possible case distinctions as they can be treated very similarly.

Theorem. *The algorithm STP solves TP correctly in time* $O(m \log m(k+n \log n))$.

As "Maketree" runs in time $O(k+n \log n)$ for the same reasons as in the non-polynomial algorithm, the essential facts that have to be shown for the complexity bound are an

$O(m \log m)$-bound both for the number of augmentations and for the number of Δ-phases. The other operations trivially are subsumed under those bounds.

A column node $j \in J'$ (an uncontracted row node $i \in I$) is called *activated* at the beginning of a Δ-phase if $j \notin J_S(\Delta') \cup J_D(\Delta')$ $(i \notin I(\Delta))$ at the end of the preceding Δ'-phase and $j \in J_S(\Delta) \cup J_D(\Delta)$ $(i \in I(\Delta))$ at the beginning of the Δ-phase.

Lemma 2. *$|\bar{b}_j| \leqslant (n+m)\Delta$ for all $j \in J'$ at the end of a Δ-phase.*

Proof. The Δ-phase may end because $J_S(\Delta) = \emptyset$ (the other cases are treated similarly). Then $\bar{b}_j < \Delta$ for all $j \in J'$. So the claim follows for all j with $\bar{b}_j \geqslant 0$. From $\sum_{i \in I} \bar{a}_i + \sum_{j \in J'} \bar{b}_j = 0$ it follows that

$$\sum_{\bar{b}_j \leqslant 0} |\bar{b}_j| = \sum_{\substack{i \in I \\ i \text{ uncontracted}}} \bar{a}_i + \sum_{\bar{b}_j > 0} \bar{b}_j < m\Delta + n\Delta$$

and the claim follows for negative $\bar{b}_j$ as well. The last inequality follows from the fact that $I(\Delta)$ is always empty after the row augmentations have taken place and it will remain empty throughout the iteration. □

Lemma 3. *If some $j \in J'$ becomes activated in the Δ-phase then $|b_j| \geqslant \Delta$.*

Proof. Assuming that the preceding scaling parameter was 2Δ (the case of a skipped phase is trivial) four cases for the activation of j have to be considered:
(1) $\frac{1}{2}\Delta \leqslant \bar{b}_j < \Delta$, $b_j > 0$,
(2) $-2\Delta < \bar{b}_j \leqslant -\Delta$, $b_j > 0$,
(3) $\Delta \leqslant \bar{b}_j < 2\Delta$, $b_j < 0$,
(4) $-\Delta < \bar{b}_j \leqslant -\frac{1}{2}\Delta$, $b_j < 0$.
The cases (1) and (3) mean that $j \notin J_S(2\Delta)$ at the end of the 2Δ-phase but $j \in J_S(\Delta)$ at the beginning of the Δ-phase. The cases (2) and (4) are the corresponding cases for $J_D(2\Delta)$ and $J_D(\Delta)$. From the fact that $b_j + \bar{b}_j = \sum_{(i,j) \in T} x_{ij} - \sum_{(j,i) \in T} x_{ji}$ is an integer multiple of 2Δ the claim follows by inspection of the cases. □

Lemma 4. *If at the end of a Δ-phase $|b_j| \geqslant 4(n+m)^2\Delta$ for some $j \in J'$ then there is an arc incident to j in T such that the arc will be contracted in the next phase. The corresponding statement holds for some $i \in I'$ provided $a_i \geqslant 4(n+m)^2\Delta$.*

Proof. We will prove only the first part of the Lemma and only for $b_j \geqslant 0$. The case $b_j < 0$ and the second part are proved similarly.

From the definition of $\bar{b}_j$, from $|b_j| \geqslant 4(n+m)^2\Delta$, from $|\bar{b}_j| \leqslant (n+m)\Delta$ (Lemma 1) and from the nonnegativity of $\sum_{(j,i) \in T} x_{ji}$ it follows that:

$$\sum_{(i,j) \in T} x_{ij} = b_j + \bar{b}_j + \sum_{(j,i) \in T} x_{ji} \geqslant 4(n+m)^2\Delta - (n+m)\Delta . \tag{3.1}$$

Let $x := \max_{(i,j) \in T} x_{ij}$. Then it follows from (3.1) and the fact that T has at most $n+m-1$ arcs that

$$x \geqslant (4(n+m)^2\Delta - (n+m)\Delta)/n+m-1 \geqslant 4(n+m)\Delta .$$

Then the arc which carries the flow x will be contracted in the next phase. For (3.1) we used the fact that $|\bar{b}_j| \leqslant (n+m)\,\Delta$ at the end of a Δ-phase. For the corresponding argument in the second part of the Lemma we need the fact that $I(\Delta) = \emptyset$, i.e. $\bar{a}_i < \Delta$ at the end of the Δ-phase. □

Lemma 5. *A node is activated $O(\log m)$ times.*

Proof. Let $i \in I$ be activated for the first time. Then $a_i = \bar{a}_i \geqslant \Delta$. From Lemma 4 it follows that after $\lceil \log(4(n+m)^2) \rceil = O(\log m)$ Δ-phases we have for the current scaling parameter Δ':

$$\Delta' \leqslant \frac{\Delta}{4(n+m)^2} \leqslant \frac{a_i}{4(n+m)^2} .$$

Hence Lemma 4 applies for i and Δ', i.e. i will be contracted in the next Δ-phase and can no longer be activated. For $j \in J$ we use the same argument except that we need Lemma 3 for the fact that $|b_j| \geqslant \Delta$ after an activation. □

Lemma 6. *The number of augmentations in a Δ-phase is bounded by $A + 2C$ where A is the number of nodes activated at the beginning of the Δ-phase and C is the number of nodes contracted in this phase.*

Proof. The proof of this Lemma contains the key facts of the complexity results. It is relatively complicated and has many parallels to the arguments in [7]. However, it is also quite different in parts, e.g. because some types of augmentations which are charged to contractions in [7] have to be charged to row augmentations in our case. To avoid too lengthy arguments we chose to restrict a detailed proof to the case that the last phase ended because $J_S(2\Delta)$ became empty. This case includes all the essential ideas. The remaining cases will be discussed more briefly after the first one. However, the first case being the most complicated one, the reader should be able to fill in the missing details. A complete proof can also be found in the second author's thesis [8].

Assume that at the beginning of the Δ-phase $J_S(2\Delta) = \emptyset$. Thus the following facts follow from the definition of activated nodes:

(1) $\bar{b}_j < 2\Delta, j \in J'$;
(2) if $b_j > 0$ then $\bar{b}_j < \Delta$;
(3) Some $j \in J'$ with $\bar{b}_j \geqslant 0$ is activated if and only if
  (a) $\Delta \leqslant \bar{b}_j < 2\Delta$, $b_j < 0$ or
  (b) $\frac{1}{2}\Delta \leqslant \bar{b}_j < \Delta$, $b_j > 0$;

(4) Let $F := \sum_{\bar{b}_j \geq 0} \lfloor \bar{b}_j / \Delta \rfloor$. Then at the beginning of the Δ-phase F is equal to $F_1 :=$ the number of activated column nodes of type (3a); (this uses (2));

(5) Let F_2 be the number of row nodes activated at the beginning of the Δ-phase (not counting those activated row nodes which send flow to some j with $\bar{b}_j < 0$ in "Augmentrows"). Then after "Augmentrows" $F \leq F_1 + F_2$;

(6) Let F_3 be the number of contractions in the Δ-phase. Then after all contractions have happened we have $F \leq F_1 + F_2 + F_3$ (a contraction contributes at most 1 to F);

So $F \leq F_1 + F_2 + F_3$ before the augmentations in the while-block take place. If an augmentation starts with some root r we either have $\bar{b}_r \geq \Delta$ ("type 1") or $\frac{1}{2}\Delta \leq \bar{b}_r < \Delta$, $b_r > 0$ ("type 2"). From (6) it follows that

(7) The number of augmentations of type 1 is bounded above by $F_1 + F_2 + F_3$.

Then Lemma 6 will follow from (7) and

(8) The number of augmentations of type 2 is bounded from above by $F_3 + F_4 + F_5$ where $F_4 :=$ number of activated column nodes of type (3b), $F_5 :=$ number of activated row nodes not counted in F_2 (i.e. row nodes which send flow to some j with $\bar{b}_j < 0$).

Clearly the activated nodes counted in F_1, F_2, F_4 and F_5 are all different which implies the correctness of Lemma 6.

For the proof of (8) we consider how a node $j \in J'$ could become the root r of an augmentation, i.e. how the situation

$$\tfrac{1}{2}\Delta \leq \bar{b}_j < \Delta \quad \text{and} \quad b_j > 0 \qquad (*)$$

can arise.

There are three exclusive cases to be considered:

(i) j was created by a contraction; we can charge this to F_3;

(ii) $\bar{b}_j < 0$ at the beginning of the Δ-phase and $(*)$ was generated by "Augmentrows"; we can charge this to F_5;

(iii) $(*)$ was already true at the beginning of the Δ-phase; this can be charged to F_4.

From (2) it follows that no other cases are possible.

All the remaining cases for the end of a phase are easier, because the number of augmentations is even smaller than $A + 2C$:

– The last phase ended because $J_D(2\Delta)$ became empty:

The proof of this case is very symmetrical to the first one. The facts about surplus nodes have to be replaced by the corresponding facts about deficit nodes, e.g. property (1) becomes: $\bar{b}_j > -2\Delta$, $j \in J'$. As the number of augmentations is controlled by deficit nodes, the potential F has to be set to $\sum_{\bar{b}_j < 0} \lfloor |\bar{b}_j| / \Delta \rfloor$, instead.

– The last phase ended, because the new Δ was determined in "Skipphase":

By the definition of Δ in "Skipphase" we have $|\bar{b}_j| \leq \Delta$ for all $j \in J'$. This is much stronger than the known bounds for $\bar{b}_j$ in the first case. In this Δ-phase no contractions occur, because initially all flows are zero and every arc can get a flow of at most Δ in "Augmentrows". This implies that the proof is almost the same as in the first case except that all complications arising from contractions do not occur. This implies that the bound $A + 2C$ can even be replaced by A.

– The last case is the one of the very first Δ-phase:
In this case a column node j (a row node i) is *activated* if $j \in J_S(\Delta) \cup J_D(\Delta)$ ($i \in I(\Delta)$) at the beginning of the Δ-phase. This holds, because the condition for the status of a node in the preceding phase is void. This case contains the two subcases $\Delta = \max_{i \in I} a_i$ and $\Delta = \max_{j \in J} b_j$.

As $\bar{b}_j := -b_j < 0$ initially, the potential $F := \sum_{\bar{b}_j \geqslant 0} \lfloor \bar{b}_j / \Delta \rfloor$ is zero before the start of the first phase in the first subcase. From $\Delta = \max_{i \in I} a_i = \max_{i \in I} \bar{a}_i$ it follows that there will be no contractions in the first phase. Hence all the potential for augmentations must come from the row augmentations and can be charged to the activated row nodes with excess Δ.

In the second subcase we have initially $-\Delta \leqslant -b_j = \bar{b}_j < 0$ for all $j \in J$.

Using this fact and the potential $F := \sum_{\bar{b}_j < 0} \lfloor |\bar{b}_j| / \Delta \rfloor$ we can treat this phase almost like the case where the 2Δ-phase ended because $J_D(2\Delta) = \emptyset$.

This concludes the proof of Lemma 6. □

As there are at most $(n+m-1)$ contractions over all phases, the following Lemma follows directly from Lemmas 5 and 6.

Lemma 7. *There are $O(m \log m)$ augmentations and row augmentations. Consequently there are $O(m \log m)$ Δ-phases in which an augmentation or a row augmentation takes place.*

Lemma 8. *There are $O(m \log m)$ Δ-phases.*

Proof. From Lemma 7 it follows that we only have to show that there are $O(m \log m)$ Δ-phases in which no flow changes and no contractions take place. This can be done as in [7]. □

As all other complexity issues are straightforward or very similar to arguments in [7] this completes our proof of the complexity bound in the Theorem.

It remains to show the correctness of the algorithm. At the end of the algorithm all excesses are zero in the contracted graph. As in [7] one can successively blow up the contracted arcs one at a time in the reverse order in which they were contracted and one can show that at each step the flow of the blown up arc can be set to a nonnegative value such that the excesses of the nodes in the blown up graph are zero, too. The nonnegativity of these values follows from the fact that any x_{ij} which is present in the Δ-phase will change its value by at most $4(n+m)\Delta$ over all subsequent phases. This can easily be deduced from the proof of Lemma 6. As all excesses are zero in the final blown up tree we have a primal feasible solution on a tree which by construction is dual feasible. This completes the proof of the Theorem. □

Remark. Jim Orlin pointed out to us that our algorithm applies without changes (also of the complexity bounds) to a generalization of the transportation problem in which we have

"row nodes" with positive supplies and "column nodes" with positive or negative supplies. We may allow arcs between column nodes but row nodes are only contained in arcs which are directed to column nodes.

References

[1] R. Ahuja, J. Orlin, C. Stein and R. Tarjan, "Improved algorithms for bipartite network flow," *SIAM Journal on Computing*, to appear.

[2] M.L. Balinski, "Signature methods for the assignment problem," *Operations Research* 33 (1985) 527–536.

[3] M.L. Fredman and R.E. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms," *Journal of the Association for Computing Machinery* 34 (1987) 596–615.

[4] A.V. Goldberg, E. Tardos and R.E. Tarjan, Network flow algorithms, in: B. Korte, L. Lovász, H.J. Prömel and A. Schrijver, eds., *Paths, Flows, and VLSI-Layout* (Springer-Verlag, Berlin-Heidelberg-New York, 1990) 101–164.

[5] M.S. Hung and W.O. Rom, "Solving the assignment problem by relaxation," *Operations Research* 28 (1980) 969–982.

[6] P. Kleinschmidt, C.W. Lee and H. Schannath, "Transportation problems which can be solved by the use of Hirsch-paths for the dual problems," *Mathematical Programming* 37 (1987) 153–168.

[7] J.B. Orlin, "A faster strongly polynomial minimum cost flow algorithm," in: *Proceedings 20th ACM Symposium on the Theory of Computation* (1988) 377–387.

[8] H. Schannath, "Polynomiale und streng polynomiale Algorithmen für Netzwerkfluss-Probleme," Ph.D. Thesis, Universität Passau (1991).