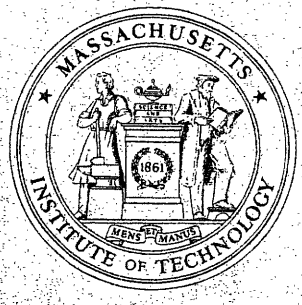
OPERATIONS RESEARCH CENTER

working paper



MASSACHUSETTS INSTITUTE OF TECHNOLOGY

A Faster Strongly Polynomial Minimum Cost Flow Algorithm

By

James B. Orlin

OR 175-88 March 1988

A FASTER STRONGLY POLYNOMIAL MINIMUM COST FLOW ALGORITHM

James B. Orlin*
Sloan School of Management
Massachusetts Institute of Technology,
Cambridge, MA. 02139, USA

ABSTRACT

We present a new strongly polynomial algorithm for the minimum cost flow problem, based on a refinement of the Edmonds-Karp scaling technique. Our algorithm solves the *uncapacitated* minimum cost flow problem as a sequence of O(n log n) shortest path problems on networks with n nodes and m arcs and runs in O(n log n (m + n log n)) steps. Using a standard transformation, this approach yields an O(m log n (m + n log n)) algorithm for the *capacitated* minimum cost flow problem. This algorithm improves the best previous strongly polynomial algorithm due to Galil and Tardos, by a factor of m/n. Our algorithm is even more efficient if the number of arcs with finite upper bounds, say m', is much less than m. In this case, the number of shortest path problems solved is O((m' + n) log n).

1. Introduction

The minimum cost flow problem is one of the most fundamental problems within network flow theory, and it has been studied extensively. Researchers have developed a number of different algorithmic approaches that have lead both to theoretical and practical improvement in the running times. The polynomial algorithms for theminimum cost flow problem can be classified into two categories: (weakly) polynomial and strongly polynomial. We call a network algorithm strongly polynomial if it satisfies the following two conditions:

SP1. The number of arithmetic operations is polynomially bounded in the number of nodes n, and the number of arcs m.

SP2. The only arithmetic operations in the algorithm are comparisons, additions and subtraction.

Here, we are using a slightly more restrictive notion of strong polynomiality than has been used by others including Meggido [1983].

The first (weakly) polynomial algorithm for the minimum cost flow problem was deveoped by Edmonds and Karp [1972]. The other (weakly) polynomial algorithms were suggested by Rock [1980], Bland and Jensen [1985], Goldberg and Tarjan [1987, 1988], and Ahuja, Goldberg, Orlin and Tarjan [1988]. Tardos [1985] developed the first strongly polynomial algorithm for the minimum cost flow problem, which was followed by many other strongly polynomial algorithms. Figure 1 summarizes these developments. In the figure, m' denotes the number of arcs with finite upper bounds, and S(n, m) denotes the time to solve a shortest path problem on a network with n nodes and m arcs. The best current bound for S(n,m) is O(m + n log n) due to Fredman and Tarjan [1984].

* This research was supported in part by Presidential Young Investigator Grant 8451517-ECS of the National Science Foundation, by Grant AFOSR-88-0088 of the Air Force Office of Scientific Research, and by Grants from Analog Devices, Apple Computer Inc. and Prime Computer.

Due to	Running Time
Tardos [1985]	$O(m^4)$
Orlin [1984]	$O((n + m')^2\ S(n, m))$
Fujishige [1986]	$O((n + m')^2\ S(n, m))$
Galil and Tardos [1986]	$O(n^2 \log n\ S(n, m))$
Goldberg and Tarjan [1987]	$O(nm^2 \log n\ \log(n^2/m))$
Goldberg and Tarjan [1988]	$O(nm^2 \log^2 n)$
Orlin (this paper)[1988]	$O((n + m') \log n\ S(n, m))$

FIGURE 1. STRONGLY POLYNOMIAL ALGORITHMS FOR THE MINIMUM COST FLOW PROBLEM

In this paper, we develop a new strongly polynomial algorithm for the minimum cost flow problem, based on a refinement of Edmonds-Karp scaling algorithm. Our algorithm improves the best previous strongly polynomial algorithm due to Galil and Tardos, by a factor of m/n. Our algorithm is even more efficient if the number of arcs with finite upper bounds is much less than m.

This paper is organized as follows. The notations and definitions are given in Section 2. A brief discussion on the optimality conditions for the minimum cost flow problem is presented in Section 3. In Section 4, we describe a modified version of Edmonds-Karp scaling technique on uncapacitated networks (i.e., there is no upper bound on arc flows, the lower bound being 0). This algorithm solves the uncapacitated network flow problem as a sequence of O(n log U) shortest path problems, where U is an upper bound on the maximum supply. In Section 5, we describe how to modify the algorithm so as to solve the uncapacitated network flow problem as a sequence of O(min (n log U, n log n)) shortest path problems. Using a standard transformation, this leads to O(min (m log U, m log n) S(n, m)) time algorithm for solving the capacitated network flow problem. This generalization is described in Section 6. In Section 7, we discuss possible implications of our results to parallel algorithms as well as to other sequential algorithms. In particular, we point out the possible significance of our strongly polynomial algorithm under the logarithmic (or bit) model of computation.

2. Notations and Definitions

Let $G = (N, A)$ be a directed network with a cost c_{ij} associated with each arc $(i, j) \in A$. We consider uncapacitated networks in which there is no upper bound on the flow on any arc and the lower bound on arc flow is 0. Let $n = |N|$ and $m = |A|$. We associate with each node $i \in N$ a real number $b(i)$ which indicates the supply (demand) of the node if $b(i) > 0$ $(b(i) < 0)$. Let $U = 1 + \max\{b(i) : i \in N\}$.

A *flow* x is a function $x : A \rightarrow R$ satisfying

$$\sum_{j \in N} x_{ji} - \sum_{j \in N} x_{ij} = b(i), \quad \text{for all } i \in N \qquad (1)$$

$$x_{ij} \geq 0, \quad \text{for all } (i, j) \in A. \qquad (2)$$

The *uncapacitated minimum cost flow problem* is to identify a flow x for which

$$\sum_{(i, j) \in A} c_{ij} x_{ij} \quad \text{is minimum.}$$

We consider the uncapacitated minimum cost flow problem satisfying the following assumptions:

Assumption 1. (Dual feasibility) All arc costs are nonnegative.

Assumption 2. (Strong connectedness) For each pair i,j of nodes in N, there is a directed path in G from i to j.

Assumption 3. (No arc multiplicity) There is at most one arc between any pair of nodes i and j, i.e., we can have arcs (i, j) or (j, i) in A, but not both.

The first two assumptions are made without any loss of generality. There is some loss of generality in the third assumption; but this assumption is made solely to simplify the notation needed in this paper. We can easily relax this assumption on arc multiplicity and extend the theory presented here to the case in which there are multiple arcs.

A *pseudoflow* x is a function $x : A \rightarrow R$ satisfying only the nonnegativity constraints, i.e., $x \geq 0$. The algorithms described in this paper maintain a pseudoflow at each intermediate step.

For a given pseudoflow x, for each node $i \in N$ we define the *imbalance* at node i to be

$$e(i) = b(i) + \sum_{j \in N} x_{ji} - \sum_{j \in N} x_{ij}$$

A positive e(i) is referred to as an *excess* and a negative e(i) is called a *deficit*. A node with excess is called a *source* node, and a node with deficit is referred to as a *sink* node. A node i with e(i) = 0 is called a *balanced* node. We denote by S and T, the sets of source and sink nodes, respectively.

For any pseudoflow x, we define the *residual network* G(x) as follows: We replace each arc $(i, j) \in A$ by two arcs (i, j) and (j, i). The arc (i, j) has cost c_{ij} and a *residual capacity* $r_{ij} = \infty$, and the arc (j, i) has cost $-c_{ij}$ and residual capacity $r_{ji} = x_{ij}$. The residual network consists *only* of arcs with positive residual capacity. The imbalance of node i in the residual network G(x) is e(i), that is, it is same as the imbalance of node i for the pseudoflow.

3. Optimality Conditions

A dual solution to the minimum cost flow problem is a vector π of node potentials. We assume that $\pi(1) = 0$. For a given vector π of node potential, the reduced cost $\bar{c}$ is defined as $\bar{c}_{ij} = c_{ij} - \pi(i) + \pi(j)$. A pair x, π of pseudoflow and node potential is optimum if it satisfies the following linear programming optimality conditions:

C1. (primal feasibility) x is a feasible flow.

C2. (dual feasibility) $\bar{c}_{ij} \geq 0$ for all arcs (i, j) in the residual network G(x).

It is easy to derive the above conditions from the well-known characterization (see Lawler [1976]) that x is an optimum flow for the minimum cost flow problem if and only if the residual network G(x) does not contain any negative cycle.

The algorithms described in this paper always maintain a pseudoflow satisfying dual feasibility and successively reduce the amount of primal infeasibility of the solution. The correctness of our algorithms relies on the following well-known result:

Lemma 1. Let x be a dual feasible pseudoflow and suppose x' is obtained from x by sending flow along a minimum cost path in the residual network. Then x' is also dual feasible. ■

4. Edmonds-Karp Scaling Technique

In this section, we present a description of a version of Edmonds-Karp right hand-side scaling technique which we call the *RHS-scaling algorithm*. Our version of the Edmonds-Karp algorithm is a modification of their original algorithm, but it differs in several computational aspects. Our version appears to be particularly well-suited for generalization to a strongly polynomial algorithm. In addition, our proof of the correctness of the RHS-scaling algorithm is of further use in proving the correctness of the strongly polynomial algorithm.

The basic idea behind the RHS-scaling algorithm is as follows. For some integer Δ, let $S(\Delta) = \{i \in N: e(i) \geq \Delta\}$, and $T(\Delta) = \{i \in N: e(i) \leq -\Delta\}$. We call a pseudoflow x *Δ-optimal*, if x is dual feasible and either $S(\Delta) = \emptyset$ or $T(\Delta) = \emptyset$. The RHS-scaling algorithm starts with a Δ-optimal pseudoflow with $\Delta = 2^{\lceil \log U \rceil}$. Given a Δ-optimal pseudoflow , the scaling algorithm obtains a $(\Delta/2)$-optimal pseudoflow by solving at most n shortest path problems. An iteration during which Δ remains unchanged is called a *Δ-scaling phase.* Subsequent to the Δ-scaling phase, Δ is replaced by $\Delta/2$, until $\Delta < 1$. Clearly, there are $\lceil \log U \rceil + 1$ scaling phases. A formal description of the RHS-scaling algorithm is given below.

```
algorithm RHS-SCALING;
begin
    set x := 0, π := 0, and e := b;
    set U := 1 + max {e(i) : i ∈ N};
    Δ := 2^⌈log U⌉ - 1;
    while there is an imbalanced node do begin
       {Δ-scaling phase}
        S(Δ) := {i : e(i) ≥ Δ};
        T(Δ) := {i : e(i) ≤ -Δ};
        while S(Δ) ≠ ø and T(Δ) ≠ ø do
        begin
            let k ∈ S(Δ) and l ∈ T(Δ);
            considering reduced costs as arc lengths,
               use modified Dijkstra's algorithm to
               compute shortest path distances d(i)
               from node k to all other nodes;
            π(i) := π(i) – d(i), for all i ∈ N;
            augment Δ units of flow along the
               shortest path from node k to node l;
            update x, r, e, c̄ S(Δ) and T(Δ);
        end;
          Δ := Δ/2;
    end; {Δ-scaling phase}
end;
```

In order to prove that the RHS-scaling algorithm is correct, we first observe that it satisfies the following flow invariant.

Flow Invariant 1. The residual capacity of each arc in the residual network is an integral multiple of Δ.

Lemma 2. The Flow Invariant 1 is satisfied prior to and subsequent to each augmentation in the RHS-scaling algorithm. ■

Flow Invariant 1 ensures that during an augmentation, Δ units of flow can be sent on the path P. Lemma 2 implies that the flow after the augmentation is still dual feasible. The algorithm terminates when all nodes are balanced; i.e., there is a feasible flow in the network. As the flow is always dual feasible, the resulting flow is optimum.

We now come to the complexity of the algorithm. Our proof uses additional constructs that will be useful in proving improved time bounds for the strongly polynomial algorithm. A node i is *active* in the Δ-scaling phase if $|e(i)| \geq \Delta$, and is *inactive* if $|e(i)| < \Delta$. A node i is said to be *regenerated* in the Δ-scaling phase if i was not in $S(2\Delta) \cup T(2\Delta)$ at the end of 2Δ–scaling phase, but is in $S(\Delta) \cup T(\Delta)$ at the beginning of the Δ-scaling phase. Clearly, for each regenerated node i, $\Delta \leq |e(i)| < 2\Delta$. The following lemma shows that there can be at most n augmenting paths found in any scaling phase. In fact, it proves a slightly stronger result that is useful in the proof of the strongly polynomial algorithm.

Lemma 3. The number of augmentations per scaling phase is at most the number of nodes that are regenerated at the beginning of the phase.

Proof. We first observe that at the beginning of the Δ-scaling phase either $S(2\Delta) = \emptyset$ or $T(2\Delta) = \emptyset$. Let us consider the case when $S(2\Delta) = \emptyset$. Then each node in $S(\Delta)$ is a regenerated node. Each augmentation starts at an active node in $S(\Delta)$ and makes it inactive after the augmentation. Thus, the number of augmentations are bounded by $|S(\Delta)|$ and the lemma follows. A similar proof for the lemma can be given for the case when $T(2\Delta) = \emptyset$. ■

It is now easy to observe the following result:

Theorem 1. The RHS-scaling algorithm determines an optimum solution of the uncapacitated minimum cost flow problem after O(log U) scaling phases and runs in O((n log U) S(n, m) time. ■

5. The Strongly Polynomial Algorithm

In this section, we present a strongly polynomial version of the RHS-scaling algorithm discussed in the previous section. We first introduce the idea of *contraction*, point out why an obvious extension of the usual RHS-scaling algorithm is not strongly polynomial, and then present a modification that is strongly polynomial.

5.1. Contraction

The key step to make the RHS-scaling algorithm strongly polynomial is to identify arcs whose flow are so large in the Δ-scaling phase that they are guaranteed to have positive flow in all subsequent scaling phases. In fact, we can quickly justigy why a flow of 4nΔ is sufficiently large. In the Δ–scaling phase of the modified RHS-scaling, the flow in any arc can change by at most 2nΔ units,

since there can be at most 2n augmentations. (The number of augmentations is at most n in the RHS-scaling algorithm.) If we sum the changes in flow in any arc over all scaling phases, the total change is at most $2n(\Delta + \Delta/2 + \Delta/4 + \ldots + 1) = 4n\Delta$. It thus follows that any arc whose flow exceeds $4n\Delta$ at any point during the Δ-scaling phase will have positive flow at each subsequent iteration. We will refer to any arc whose flow exceeds $4n\Delta$ during the Δ-scaling phase as *strongly basic*. A natural operation would then be to contract (or shrink) the basic arcs and continue the scaling procedure on a problem involving fewer nodes.

Each contracted arc has positive flow, and its reduced cost is $\bar{c}_{ij} = 0$. By contraction of arc (i, j), we mean replacing nodes i and j with a single contracted node, say v, and replacing each arc (k, i) or (k, j) (respectively, (i, k) or (j, k)) with an arc (k, v) (respectively, (v, k)) whose reduced cost is the same as the arc it replaces. In addition, we let $b(v) = b(i) + b(j)$, and the resulting imbalance is $e(v) = e(i) + e(j)$. (This contraction may lead to having arcs (i, j) and (j, i), as well as to multiple arcs. We ignore these possibilities as before, but solely for notational convenience.) This contraction is the implicit foundation of a number of strongly polynomial algorithms including those of Tardos [1985], Tardos and Galil [1986], Fujishige [1986], and Orlin [1984]. The algorithm continues to contract arcs until ultimately an optimum flow is determined in the contracted network. At this point, all of the contracted arcs are expanded, and an optimal flow for the original network is recovered.

The following lemma shows that contraction of an arc incident to node i will take place by the time that Δ is sufficiently small relative to $|b(i)|$.

Lemma 4. Suppose at the termination of the Δ-scaling phase, $\Delta < |b(i)|/8n^2$ for some node i in N. Then there is an arc incident to node i whose flow exceeds $4n\Delta$.

Proof. We first claim that $|e(i)| < 2n\Delta$. To see this, recall that either all of $S(\Delta)$ or all of $T(\Delta)$ is regenerated at the beginning of the Δ-scaling phase, and thus either the total excess or the total deficit is strictly bounded by $2n\Delta$. We now prove the lemma in the case when $b(i) > 0$. The case when $b(i) < 0$ is proved analogously. There are at most n-1 arcs directed out from i, and at least one of these arcs has a flow at least $(b(i) - e(i))/(n-1)$, which exceeds $4n\Delta$. ■

5.2. A Pathological Example for RHS-scaling Algorithm

Lemma 4 comes very close to yielding a strongly polynomial algorithm for the following reason: At the beginning of the algorithm, all of the nodes have $e(i) = b(i)$, and thus each node can be regenerated O(log n) times before $\Delta < |b(i)|/8n^2$ and i is incident to a strongly basic arc. At this point two nodes get contracted into a single node. This almost leads to a O(n log n S(n, m)) time bound. There is, however, a difficulty which is illustrated in Figure 3. In fact, without additional modification, the algorithm is not strongly polynomial. The difficulty lies with the nodes that are created via a contraction. One can create a contracted node v for which b(v) is nearly 0, but e(v) is relatively large. It is possible for v to be regenerated a large number of times before an arc incident to v is contracted.

In Figure 2(a), we give the initial supply, demands and flows. Assume that all the costs are 0. There is a unique feasible solution that the algorithm must determine. In the 8M-scaling phase, 8M units of flow is sent from node 3 to node 2. Subsequently, in each of the 4M, 2M, and M-scaling phases, flow is sent from node 1 to node 4. In Figure 2(b), we give the flows at the end of M-scaling phase. At this point, we may contract arcs (1, 2) and (3, 4), but not the arc (3, 1). The resulting contracted graph is given in Figure 2(c). Observe that $b(A) = -2$ and $b(B) = 2$. At this point, it will take O(log M) scaling iterations before the flow in arc (B, A) is sufficiently small (relative to the unique feasible flow of 2 units from B to A) so that this arc gets contracted and the algorithm terminates. Thus the algorithm is not strongly polynomial.

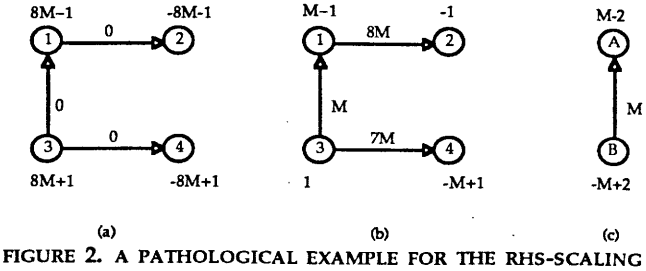


FIGURE 2. A PATHOLOGICAL EXAMPLE FOR THE RHS-SCALING ALGORITHM

To overcome the difficulty we observe that the bad situation can occur only in unusual circumstances; in particular, we first make the following observation. Since all flows at the 2Δ-scaling iteration are multiples of 2Δ,

$$e(i) \equiv b(i) \mod 2\Delta.$$

If node v is regenerated at the beginning of the Δ scaling iteration and if $|b(v)| = \varepsilon$ for some ε that is very small relative to Δ, then one of the following two situations occurs. Either the excess $e(v) = 2\Delta - \varepsilon$ for some very small positive ε, and $b(v) = -\varepsilon$, or else $e(v) = -2\Delta + \varepsilon$ for some very small positive ε, and $b(v) = \varepsilon$. In these bad cases one could regenerate v a large number of times prior to an arc incident to v being contracted. However, one can overcome the difficulty with one additional "special augmentation." First observe that e(v) and b(v) are opposite in sign. In these cases, if $e(v) = 2\Delta - \varepsilon$, we can try to send 2Δ units of flow from v at the 2Δ-scaling iteration. If such a "special augmentation" were to take place, then subsequent to the augmentation, $e(v) = b(v) = -\varepsilon$. (If e(v) were negative, then we would send 2Δ units of flow to v.) This type of "special augmentation" is the last ingredient needed for the strongly polynomial algorithm.

5.3. The Algorithm

Our strongly polynomial algorithm proceeds as the RHS-scaling algorithm with the following differences.

(1) We contract an arc whenever it becomes strongly basic. The reason that our generalization of the RHS-scaling algorithm is strongly polynomial is that we can locate an additional strongly basic arc after O(log n) scaling phases, and that there are at most $n - 1$ contractions. At termination, the contracted arcs are expanded and exact arc flows are computed.

(2) We allow an anomalous special augmentation to take place in some cases for which the outflow of a supply node i exceeds its supply b(i) or the inflow of a demand node i exceeds its demand –b(i). These special augmentations are needed to ensure that arcs will become strongly basic at a sufficiently fast rate. Without these special augmentations the algorithm is not strongly polynomial.

(3) We require that all solutions be spanning tree solutions, i.e., arcs with positive flows do not contain cycles. This requirement can be enforced by using a "modified Dijkstra" algorithm in which such arcs are given preference. (We omit the details of modified Dijkstra in this paper, but it is included in a more complete report available from the author.) The requirement of spanning tree solutions greatly simplifies the expansion of contracted arcs at the end of the algorithm.

(4) We no longer require Δ to be a power of 2, and we no longer require Δ to be divided by 2 at each iteration. These modifications are technically required by the algorithm to be strongly polynomial. (The algorithm given below does divide Δ by 2. These divisions can be replaced by multiplications of x, r, and e by 2. The resulting algorithm would still correctly calculate the optimum spanning tree, from which the optimum solution for the original data could be recovered.) Our strongly polynomial algorithm is given below. In the algorithm description, we denote the contracted network by G' = (N', A').

```
algorithm STRONGLY POLYNOMIAL;
begin
    set x := 0, π := 0, and e := b;
    set Δ := max { e(i) : i ∈ N};
    while there is an imbalanced node do begin
        if x_ij = 0 for all (i, j) in A' and |e(i)| < Δ for
           all i ∈ N' then Δ := max {e(i) : i ∈ N' };
        while there is an arc (i, j) ∈ A' with x_ij ≥ 4nΔ
           do contract arc (i,j);
        S(Δ) := {i ∈ N': e(i) ≥ Δ};
        T(Δ) := {i ∈ N': e(i) ≤ –Δ};
        S*(Δ): = S(Δ) ∪ {i ∈ N': e(i) ≥ Δ/2 and
           b(i) < 0};
        T*(Δ): = T(Δ) ∪ {i ∈ N': e(i) ≤ -Δ/2 and
           b(i) > 0};
        while S*(Δ) ≠ ø and T*(Δ) ≠ ø do
        begin
            let k ∈ S*(Δ) and l ∈ T*(Δ);
            considering reduced costs as arc lengths,
               use modified Dijkstra's algorithm to
               compute shortest path distances d(i)
               from node k to all other nodes,
            π(i) := π(i) – d(i), for all i ∈ N';
            augment Δ units of flow along the
               shortest path from node k to node l;
            update x, r, e, c̄, S*(Δ) and T*(Δ);
        end;
        Δ := Δ/2;
    end;
end;
```

Let Δ' and Δ be the scale factors in two consecutive scaling phases. We define a node k to be *regenerated* in the Δ-scaling phase if $k \notin S^*(\Delta') \cup T^*(\Delta')$ at the end of the Δ'-scaling phase and $k \in S^*(\Delta) \cup T^*(\Delta)$ at the beginning of the Δ-scaling phase. Note that by this definition the new nodes formed by contraction of an arc (i,j) in the Δ-scaling phase are *not* called regenerated nodes; however, i and j may be called regenerated nodes.

Observe that an ordinary augmentation originates at a node in $S(\Delta)$ and terminates at a node in $T(\Delta)$. A special augmentation either originates at a node in $S^*(\Delta) - S(\Delta)$ or terminates at a node in $T^*(\Delta) - T(\Delta)$. The strongly polynomial algorithm is essentially the same as the RHS-scaling algorithm except that it performs special augmentations and sometimes adjusts the scale factor by a factor larger than 2.

5.4 Accuracy and Complexity of the Algorithm

The accuracy of the strongly polynomial algorithm is easily established. We show that the algorithm always satisfies Flow Invariant 1 and, consequently, each augmentation can carry Δ units of flow. The algorithm always maintains dual feasibility of the solution and terminates when primal feasibility conditions are also satisfied. Thus the algorithm terminates with an optimum flow in the contracted network. Expanding the contracted arcs of this solution yields an optimum flow in the original network.

The complexity proof of the algorithm is rather involved. However, the essence of the argument can be summarized as follows. The number of augmentations is bounded by the number of regenerations plus the number of contractions, and this number is O(n log n) since each node is regenerated O(log n) times. In addition, since we are maintaining spanning tree solutions, there is an additional O(n) time spent per scaling iteration even if no augmentations occur and the number of scaling iterations is O(n log n). Finally, there is a total time of O(mn) spent in contractions. The resulting running time is O((n log n)(m + n log n)) which is O(n log nS(n,m)).

Lemma 5. At each stage of the algorithm, Flow Invariant 1 is satisfied.

Proof. This result is easily shown by performing induction on the number of augmentations, contractions, and adjustments in the scale factor. ■

Lemma 6. In the Δ-scaling phase, if $x_{ij} \geq 4n\Delta$ for some arc (i, j) in A', then $x_{ij} \geq 0$ in all subsequent scaling phases.

Proof. In the absence of special augmentations and contractions, the flow changes on any arc due to ordinary augmentations is at most $n(\Delta + \Delta/2 + \Delta/4 + \cdots + 1) = 2n\Delta$. There are at most n special augmentations (see Lemma 7 below) and each such augmentation carries at most Δ units of flow. Each contraction causes at most one additional ordinary augmentation (see Lemma 8) and there are at most n

contractions. Thus, the total flow change on any arc is at most $4n\Delta$ and the lemma follows. ■

Lemma 7. The number of special augmentations is at most n over all scaling phases.

Proof. Suppose that a special augmentation originates at a node $k \in S^*(\Delta) - S(\Delta)$. A proof along similar lines can be given when the special augmentation terminates at a node $k \in T^*(\Delta) - T(\Delta)$. Clearly, $\Delta > e(k) \geq \Delta/2$ and $b(k) < 0$. Observe that node k is formed during a contraction operation since for an uncontracted node e(k) and b(k) have the same sign. During the special augmentation, Δ units of flow are sent out of node k. Hence $e(k) < 0$ after the augmentation, and there will not be any further special augmentation starting or terminating at node k. Thus every special augmentation can be charged to a contraction and there are at most n contractions. ■

Lemma 8. The number of ordinary augmentations during the Δ-scaling phase is bounded by the number of regenerated nodes plus the number of contractions in that phase.

Proof. At the end of 2Δ-scaling phase, either $S(2\Delta) = \emptyset$ or $T(2\Delta) = \emptyset$. We consider the case when $S(2\Delta) = \emptyset$. A similar proof can be given when $T(2\Delta) = \emptyset$.

Consider the potential function $F = \sum_{i \in S} \lfloor e(i)/\Delta \rfloor$.

Clearly, at the beginning of the Δ-scaling phase, F is no more than the number of regenerated nodes. Consider the effect of contraction of arc (i,j). It can be easily verified that if $e(i) \leq 0$ or $e(j) \leq 0$ then the contraction does not increase F. In the case when $e(i) > 0$ and $e(j) > 0$, there are four possibilities to consider:

(1) $0 \leq e(i) < \Delta$ and $0 \leq e(j) < \Delta$
(2) $0 \leq e(i) < \Delta$ and $\Delta \leq e(j) < 2\Delta$
(3) $\Delta \leq e(i) < 2\Delta$ and $0 \leq e(j) < \Delta$
(4) $\Delta \leq e(i) < 2\Delta$ and $\Delta \leq e(j) < 2\Delta$

In any of the above cases, the contraction increases F by at most one unit. Finally, we observe that each ordinary augmentation decreases F by one unit and the lemma follows. ■

Lemma 9. At each stage of the algorithm, $e(k) \equiv b(k) \bmod \Delta$ for every node $k \in N'$.

Proof. The value e(k) is b(k) minus the flow across the cutset $(k, N' - k)$. By Flow Invariant 1, $x_{ij} \equiv b(k) \bmod \Delta$. ■

Lemma 10. The first time that a node k is regenerated, it satisfies $|e(k)| \leq |b(k)|$.

Proof. Suppose that node k is regenerated for the first time at the beginning of Δ-scaling phase. There are several cases to consider. First suppose that $e(k) > 0$ and $b(k) > 0$. At the time that k is regenerated, all arc flows are integral multiples of 2Δ and $e(k) < 2\Delta$. Since $e(k) \equiv b(k) \bmod 2\Delta$, it follows that $b(k) = e(k) + w(2\Delta)$ for some integral multiple w. Further, since $0 < e(k) < 2\Delta$ and $b(k) > 0$, it is immediate that $w \geq 0$ and $e(k) \leq b(k)$.

We next consider the case when $b(k) > 0$ and $e(k) < 0$. Since k is regenerated, it was not in $T^*(2\Delta)$ in the previous scaling phase. Thus $0 > e(k) \geq -\Delta$. Since $b(k) = e(k) + w(2\Delta)$ for some integer w and $b(k) > 0$, it follows that $w \geq 1$. Consequently, $b(k) \geq e(k) + 2\Delta \geq |e(k)|$, and the claim holds.

We have thus proved the lemma for the case when $b(k) > 0$. The case when $b(k) < 0$ can be proved in an analogous manner. ■

Lemma 11. Any node is regenerated O(log n) times.

Proof. Suppose a node k is regenerated for the first time at the Δ^*-scaling phase. Then $\Delta^* \leq |e(k)| \leq |b(k)|$, where the second inequality follows from Lemma 10. After $\lceil \log (8n^2) \rceil = O(\log n)$ scaling phases, the scale factor $\Delta \leq \Delta^*/8n^2 \leq |b(k)|/8n^2$, and by Lemma 4 there exists a strongly basic arc in the cutset $(k, N' - \{k\})$. The node k contracts into a new node and is (vacuously) not regenerated. ■

Theorem 2. The total number of augmentations over all scaling phases is O(min (n log U, n log n)).

Proof. In the case that $U \leq n$, we can choose the initial scale factor to be $2^{\lceil \log U \rceil - 1}$ and the algorithm will terminate after $\lceil \log U \rceil$ scaling phases. Since there will be no contractions, the algorithm reduces to the RHS-scaling algorithm given in Section 4. In this case, the time is the same as given in Theorem 1.

We now consider the case when $U > n$. By the previous lemma, any node is regenerated

O(log n) times. As these n original nodes and at most n new nodes can be formed due to contractions, the total number of regenerations is O(n log n). The Lemma 8 yields that the number of ordinary augmentations is also O(n log n). The number of special augmentations is already shown to be at most n in Lemma 7. The theorem is now evident. ■

Theorem 3. The number of scaling phases is O(min (log U, n log n)).

Proof. The bound of O(log U) on the number of scaling phases follows from the fact that in the consecutive scaling phases, the scale factor is at least halved. By Theorem 2, the number of scaling phases in which an augmentation occurs is O(n log n). We now derive a bound of O(n log n) on the number of scaling phases in which no augmentation occurs.

Consider a Δ-scaling phase in which no augmentation occurs. Let there exist a node k for which $|e(k)| > \Delta/8n^2$. We assume that $e(k) > 0$; the case when $e(k) < 0$ can be proved similarly. Then within O(log n) scaling phases, the node k is regenerated and within further O(log n) scaling phases, there is a contraction. Thus, this case can occur O(log n) times.

We now consider the case when $|e(k)| \leq \Delta/8n^2$ for each node k. If all arcs in the contracted graph have zero flow, then we set Δ to max(e(i) : i ∈ N'), and in the same scaling phase the node with maximum excess is regenerated. Since within the next O(log n) scaling phases there will be a contraction, this case will occur O(n log n) times.

Finally, we consider the case when $|e(i)| < \Delta/8n^2$ for each node i and there is some arc, say (k, *l*), with positive flow. By Flow Invariant 1, $x_{kl} \geq 4n\Delta'$ with respect to the current scale factor Δ∋ and a contraction would take place. This last case can occur O(n) times. ■

Theorem 4. The strongly polynomial algorithm determines the minimum cost flow in the contracted network in O(min (n log U, n log n) S(n, m)) time.

Proof. The algorithm terminates when all of the node imbalances are 0. Since dual feasibility is maintained at each step, the algorithm terminates with an optimum solution. To complete the proof of the theorem, we need to discuss the computation time of the algorithm.

Consider the time spent in a scaling phase. Reducing the scale factor by a factor other than 2 requires O(n) time as the number of arcs with positive flow is at most n–1. The contractable arcs can also be identified in O(n) time. The time needed to identify the sets S*(Δ) and T*(Δ) is O(n) even if these sets may be empty. Since these are O(min (log U, n log n)) scaling phases, these operations require O(min (log U, n log n) n) total time.

The number of augmentations in the algorithm is O(min (n log U, n log n)). Each augmentation involves solving a shortest path problem and augmenting flow along such a path. The time needed to perform these operations is clearly O(min (n log U, n log n) S(n, m)). ■

5.5 Expansion of Contracted Arcs.

The optimum solution in the contracted network does not contain a cycle consisting of positive flow arcs; hence, we can easily obtain a spanning tree solution. Let T' be the corresponding tree. Now inductively expand the contracted arcs, one at a time. Since the strongly feasible arcs form a forest, the ultimate solution is again a spanning tree in the original network. Let T denote this spanning tree. As all the strongly basic arcs have zero reduced cost at the time of contraction, the potential of an uncontracted node is the same as the potential of the node it got contracted into. From this observation it is easy to prove that the expanded spanning tree solution satisfies dual feasibility.

We now need to show that the spanning tree T permits a primal feasible solution. We accomplish this by showing (details are omitted) that for each pseudoflow in the contracted network, there exists a corresponding unique pseudoflow in the original network, and that each augmentation of Δ units in the contracted network changes the flow on any arc in the original network by at most Δ units. It thus follows that any strongly basic arc has nonnegative flow at the end of all iterations and the ultimate flow is primal feasible.

6. Capacitated Minimum Cost Flow Problem

There is a well-known transformation available to convert a capacitated minimum cost flow problem to an uncapacitated one. This consists of replacing each capacitated arc (i, j) by an

additional node k and two arcs (i, k) and (j, k) as shown in Figure 3. This gives us a network with node set $N_1 \cup N_2$ where $N_1 = N$ and each node in N_2 corresponds to a capacitated arc in the original network. Each node in N_2 is a demand node (If each arc in A is capacitated, then the transformed network is bipartite.) When the algorithm described in Section 5 is applied to the transformed network, it solves O(min ((n + m') log U, (n + m') log n)) shortest path problem and each shortest path problem takes O(m + (n + m') log n). In order to reduce the shortest path computation time to O(m + n log n), we have to solve the shortest path problems more carefully. This can be accomplished by replacing each node in N_2 by at most two arcs, and solving a shortest path problem on the reduced network. We state without proof that the shortest path problem on the transformed network can indeed by solved in O(S(n, m)) time. We therefore obtain the following result:

Theorem 5. The strongly polynomial algorithm solves the capacitated minimum cost flow problem in O(min (m log U, m log n) S(n, m)) time. ■

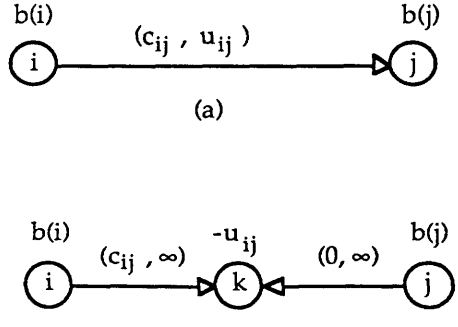


FIGURE 3. TRANSFORMING A CAPACITATED ARC INTO TWO UNCAPACITATED ARCS.

7. Future Directions

In this paper, we have presented a new strongly polynomial algorithm for the minimum cost flow problem. Our algorithm improves all previous strongly polynomial minimum cost flow algorithms. Our algorithm also appears attractive in logarithmic and parallel models of computation, as described below.

The Logarithmic Model of Computation.

Although the theoretical significance of the above algorithm is primarily for the uniform model of computation, where each arithmetic operation takes O(1) steps, a minor variation of the algorithm is very fast under the logarithmic model of computation (where we count the number of bit operations) as per Cook and Reckhow [1973]. (The modification is that Δ is always chosen to be a power of 2.) The resulting computation time for uncapacitated problem is O((n log n) S(n, m, C) + n log U). Here the shortest path problem is a function of the largest cost C as well as n and m. In other words, the running time grows linearly in log U as U grows exponentially large. For very large values of U and moderate values of C this bound surpasses all other known algorithms for the minimum cost flow problem as analyzed under the logarithmic model of computation, even when applied to the capacitated network flow problem.

Parallel Computation

The most obvious way to speed up the algorithm with parallel computation is to apply parallel algorithms to the shortest path subroutine. This leads to a running time for the fully capacitated minimum cost network flow problem of $O(m \log^3 n)$ time using n^3 processors. Here, of course, the processor utilization is not very good. Moreover, we conjecture that we can reduce the time to n times a polylog function of n by allowing randomized algorithms.

Other Algorithms

There are two approaches here that merit further investigation. First of all, if we translate the scaling method of the transformed problem back into the capacitated network flow problem we derive a new type of scaling algorithm for the minimum cost flow problem. In particular, excesses and deficits are stored on arcs as well as nodes. This new type of scaling algorithm may be of use for the maximum flow problem or the minimum cost flow problem.

Second, the essence of the strongly polynomial algorithm is that each node could be expected to be contracted within O(log n) augmentations. It was important that we could treat nodes individually, rather than argue that just one node is contracted with O(log n) iterations. Within

this context, the role of "special augmentations" played a pivotal role. This argument may well be applicable to other algorithms that rely on a successive reduction in primal feasibility.

ACKNOWLEDGMENTS

I greatfully acknowledge the help of Ravi Ahuja both with the exposition of the paper and the details of the proofs. His help was very valuable and greatly appreciated.

REFERENCES

AHUJA, R.K., GOLDBERG, A.V., ORLIN, J.B., AND TARJAN, R.E. Solving minimum cost flow problem by double scaling. To appear.

BLAND, R.G., AND JENSEN, D.L. On the computational behavior of a polynomial-time network flow algorithm. Technical Report 661, School of Operations Research and Industrial Engineering, Cornell University, Ithaca, 1985.

COOK, S.A., AND RECKHOW, R.A. Time bounded random access machines. *J. of Comput. System Sci. 7* (1973), 354-375.

DIJKSTRA, E. A note of two problems in connexion with graphs. *Numeriche Mathematics 1* (1959), 269-271.

EDMONDS, J., AND KARP, R.M. Theoretical improvements in algorithmic efficiency for network flow problems. *J. ACM 19*(1972), 248-264.

FREDMAN, M.L., AND TARJAN, R.E., Fibonacci heaps and their uses in network optimization algorithms. *25th IEEE Symp. on Found. of Comp. Sci.* (1984), 338-346.

FUJISHIGE, S. An $O(m^3 \log n)$ capacity-rounding algorithm for the minimum cost circulation problem: A dual framework of Tardos' algorithm. *Math. Prog. 35* (1986), 298-309.

GALIL, Z., AND TARDOS, E. An $O(n^2 (m + n \log n) \log n)$ min-cost flow algorithm. *Proc. 27th Annual Symp. of Found. of Comp. Sci.* (1986), 136-146.

GOLDBERG, A.V., AND TARJAN, R.E. Solving minimum cost flow problem by successive approximation. *Proc. 19th ACM Symp . on the Theory of Comp.* (1987), 7-18 .

GOLDBERG, A.V., AND TARJAN, R.E. Finding minimum-cost circulations by canceling negative cycles. *To appear in Proc. 20th ACM Symp. on the Theory of Comp.* (1988).

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

MEGIDDO, N. Towards a strongly polynomial algorithm for linear programming. *SIAM J. of Computing 12*(1983) 347-353.

ORLIN, J.B. Genuinely polynomial simplex and non-simplex algorithms for the minimum cost flow problem. Technical Report No. 1615-84, Sloan School of Management, M.I.T., Cambridge, 1984.

ROCK, H. Scaling techniques for minimal cost network flows. In. V. Page, editor, *Discrete Structures and Algorithms* , Carl Hansen, Munich, 1980. pp 101-191.

TARDOS, E. A strongly polynomial minimum cost circulation algorithm. *Combinatorica 5* (1985), 247-255.

TARDOS, E. A strongly polynomial algorithm to solve combinatorial linear programs. *Oper. Res. 34* (1986), 250-256.