A Faster Strongly Polynomial Minimum Cost Flow Algorithm
Author(s): James B. Orlin
Source: *Operations Research*, Vol. 41, No. 2 (Mar. - Apr., 1993), pp. 338-350
Published by: INFORMS
Stable URL: http://www.jstor.org/stable/171782
Accessed: 30-04-2018 09:20 UTC

A FASTER STRONGLY POLYNOMIAL MINIMUM COST FLOW ALGORITHM

JAMES B. ORLIN

Massachusetts Institute of Technology, Cambridge, Massachusetts

(Received August 1989; revisions received April, November 1991; accepted December 1991)

In this paper, we present a new strongly polynomial time 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 $O(n \log n (m + n \log n))$ time. 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 time algorithm, due to Z. Galil and E. Tardos, by a factor of n^2/m. Our algorithm for the capacitated minimum cost flow problem is even more efficient if the number of arcs with finite upper bounds, say m', is much less than m. In this case, the running time of the algorithm is $O((m' + n)\log n(m + n \log n))$.

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 led both to theoretical and practical improvements in the running time. We refer the reader to the text of Ahuja, Magnanti and Orlin (1993) for a survey of many of these developments. In Figure 1, we summarize the developments in solving the minimum cost flow problem by polynomial time algorithms.

Figure 1 reports the running times for networks with n nodes and m arcs, of which m' arcs are capacitated. Those algorithms whose running times contain the term $\log C$ assume integral cost coefficients whose absolute values are bounded by C. Those algorithms whose running times contain the term $\log U$ assume integral capacities and supply/demands whose absolute values are bounded by U. In the figure, the term $S(\cdot)$ denotes the running time to solve a shortest path problem with nonnegative arc costs and $M(\cdot)$ denotes the running time to solve a maximum flow problem. The best time bounds for the shortest path and maximum flow problems are also given in Figure 1.

Edmonds and Karp (1972) were the first to solve the minimum cost flow problem in polynomial time. Their algorithm, now commonly known as the *Edmonds-Karp scaling technique*, was to reduce minimum cost flow problem to a sequence of $O((n + m')\log U)$ shortest path problems. Although Edmonds and Karp did resolve the question whether the minimum cost flow problem can be solved in polynomial time, an interesting related question was unresolved. As stated in their paper:

> A challenging open problem is to give a method for the minimum cost flow problem having a bound of computation which is polynomial in the number of nodes and arcs, and is independent of both costs and capacities.

In other words, the open problem was to determine a strongly polynomial time algorithm for the minimum cost flow problem. In a strongly polynomial time algorithm, the number of operations performed by the algorithm is polynomially bounded in n and m where the number of operations allowed are additions, subtractions, and comparisons. (We point out that our definition of strongly polynomial time algorithms is more restrictive than the one typically used in the literature as it allows fewer operations.) There are several reasons for studying strongly polynomial time algorithms. A primary theoretical motivation for developing strongly polynomial time algorithms is that they are useful in the subroutines where irrational data are used (e.g., the generalized maximum flow algorithm of Goldberg, Plotkin and Tardos 1988). Another motivation for developing these algorithms is that they help to identify whether the size of the numbers involved increases the inherent complexity of solving the problem.

The first strongly polynomial time algorithm for the minimum cost flow problem was developed by Tardos

Subject classification: Networks/graphs, flow algorithms: a faster strongly polynomial minimum cost flow algorithm.
Area of review: OPTIMIZATION.

0030-364X/93/4102-0338 $01.25

Polynomial Algorithms

#	Due to	Year	Running Time
1	Edmonds and Karp	1972	O((n + m') log U S(n, m, nC))
2	Rock	1980	O((n + m') log U S(n, m, nC))
3	Rock	1980	O(n log C M(n, m, U))
4	Bland and Jensen	1985	O(m log C M(n, m, U))
5	Goldberg and Tarjan	1987	O(nm log (n^2/m) log (nC))
6	Goldberg and Tarjan	1988	O(nm log n log (nC))
7	Ahuja, Goldberg, Orlin and Tarjan	1988	O(nm log log U log (nC))

Strongly Polynomial Algorithms

#	Due to	Year	Running Time
1	Tardos	1985	O(m^4)
2	Orlin	1984	O($(n + m')^2$ log n S(n, m))
3	Fujishige	1986	O($(n + m')^2$ log n S(n, m))
4	Galil and Tardos	1986	O(n^2 log n S(n, m))
5	Goldberg and Tarjan	1987	O(nm^2 log n log(n^2/m))
6	Goldberg and Tarjan	1988	O(nm^2 log^2 n)
7	Orlin (this paper)	1988	O((n + m') log n S(n, m))

S(n, m)	= O(m + n log n)	Fredman and Tarjan [1984]
S(n, m, C)	= O(Min (m + $n\sqrt{\log C}$), (m log log C))	Ahuja, Mehlhorn, Orlin and Tarjan [1990] Van Emde Boas, Kaas and Zijlstra[1977]
M(n, m)	= O(min (nm + $n^{2+\varepsilon}$, nm log n) where ε is any fixed constant.	King, Rao, and Tarjan [1991]
M(n, m, U)	= O(nm log ($\frac{n}{m}\sqrt{\log U}$ + 2))	Ahuja, Orlin and Tarjan [1989]

Figure 1. Polynomial algorithms for the minimum cost flow problem. (Year indicates the publication date of the paper, except for Bland and Jensen (1985) and Orlin (1984), where it indicates the year the technical report was written. The papers are listed in the order they were developed.)

(1985). This algorithm motivated the development of several other strongly polynomial time algorithms by Orlin (1984), Fujishige (1986), Galil and Tardos (1986), Goldberg and Tarjan (1987, 1988), Plotkin and Tardos (1990), and Orlin, Plotkin and Tardos (1991). The fastest strongly polynomial time algorithm prior to the algorithm presented in this paper is due to Galil and Tardos and runs in $O(n^2 \log n\, S(n, m))$ time.

In this paper, we describe a new strongly polynomial time algorithm that solves the minimum cost flow problem in $O(m \log n\, S(n, m))$ time. Our algorithm is a variation of the Edmonds and Karp scaling algorithm and solves the minimum cost flow problem as a sequence of $O(m \log n)$ shortest path problems. Hence, our algorithm improves the running time of Galil and Tardos's algorithm by a factor of n^2/m. Besides being (currently) the fastest strongly polynomial time algorithm for the minimum cost flow problem, our algorithm has the following nice features:

1. The algorithm is fairly simple, intuitively appealing, and easy to program.
2. The algorithm yields an attractive running time for solving the minimum cost flow problem in parallel. There exists a parallel shortest path algorithm that uses n^3 parallel processors and runs in $O(\log^2 n)$ time. Using this shortest path algorithm as a subroutine, our minimum cost flow algorithm can be implemented in $O(m \log^3 n)$ time using $O(n^3)$ processors. Further improvements in processor utilizations are possible through the use of fast matrix multiplication techniques.

This paper is organized as follows. In Section 1, we present the notation and definitions. A brief discussion on the optimality conditions and some basic results for the minimum cost flow problem are presented in Section 2. In Section 3, we describe a modified version of the Edmonds-Karp scaling technique for

uncapacitated networks (i.e., where all arc capacities are ∞ and all lower bounds are zero). This algorithm solves the uncapacitated minimum cost flow problem as a sequence of $O(n \log U)$ shortest path problems. In Section 4, we describe how to modify the algorithm, to solve the uncapacitated minimum cost flow problem as a sequence of $O(n \log n)$ shortest path problems. The running time of this algorithm is $O(n \log n\, S(n, m))$. Using a standard transformation, this leads to an $O(m \log n\, S(n, m))$ time algorithm for solving the capacitated minimum cost flow problem; we describe this generalization in Section 5.

1. NOTATION AND DEFINITIONS

Let $G = (N, A)$ be a directed network with a cost c_{ij} associated with every arc $(i, j) \in A$. We consider uncapacitated networks, i.e., networks in which there is no upper bound on the flow on any arc. Let $n = |N|$ and $m = |A|$. We associate with each node $i \in N$ an integer number $b(i)$, which indicates the supply (or demand) of the node if $b(i) > 0$ (or $b(i) < 0$). Let $U = \max\{|b(i)|: i \in N\}$. The uncapacitated minimum cost flow problem can be stated as follows.

$$\text{Minimize} \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{1a}$$

subject to

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

$$x_{ij} \geq 0 \quad \text{for all } (i, j) \in A. \tag{1c}$$

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

Assumption 1. For all nodes i and j in N, there is a directed path in G from i to j.

Assumption 2. All arc costs are nonnegative.

The first assumption can be met by adding an artificial node s with $b(s) = 0$, and adding artificial arcs (s, i) and (i, s) for all nodes i with a very large cost M. If the value of M is sufficiently high, then none of these artificial arcs will appear with positive flow in any minimum cost solution, provided that the minimum cost flow problem is feasible. This assumption allows us to ignore the infeasibility of the minimum cost flow problem.

The second assumption is also made without any loss of generality. If the network contains some negative cost arcs, then using the following standard transformation arc costs can be made nonnegative. We first compute the shortest path distances $d(i)$ from node s to all other nodes, where the length of arc (i, j) is c_{ij}. If the shortest path algorithm identifies a negative cycle in the network, then either the minimum cost flow problem is infeasible or its optimum solution is unbounded. Otherwise, we replace c_{ij} by $c_{ij} + d(i) - d(j)$, which is nonnegative for all the arcs. We show later in Lemma 2 that this transformation does not affect the value of the optimal minimum cost flow.

The algorithm described in this paper maintains a "pseudoflow" at each intermediate step. A *pseudoflow* x is a function $x: A \to R$ that satisfies only the nonnegativity constraints (1c) on arc flows, i.e., we allow a pseudoflow to violate the mass balance constraints (1b) at nodes. For a given pseudoflow x, we define the *imbalance* at node i to be

$$e(i) = b(i) + \sum_{\{j:(j,i)\in A\}} x_{ji} - \sum_{\{j:(i,j)\in A\}} x_{ij}.$$

A positive $e(i)$ is referred to as an *excess*, and a negative $e(i)$ is called a *deficit*. A node i with $e(i) > 0$ is called a *source* node, and a node i with $e(i) < 0$ is referred to as a *sink* node. A node i with $e(i) = 0$ is called a *balanced* node, and *imbalanced* otherwise. 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 *residual capacity* $r_{ij} = \infty$, and the arc (j, i) has cost $-c_{ij}$ and residual capacity $r_{ji} = x_{ij}$. A 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 the same as the imbalance of node i for the pseudoflow. We illustrate the construction of the residual network in Figure 2. Figure 2a shows the network G with arc costs and node supplies, Figure 2b shows a pseudoflow x, and 2c gives the corresponding residual network $G(x)$.

We point out that in the residual network $G(x)$ there may be multiple arcs, i.e., more than one arc with the same tail and head nodes, but possibly with different arc costs. The notation we use to address an arc, say (i, j), is appropriate only when there is one arc with the same tail and head nodes. In the case of multiple arcs, we should use more complex notation. However, for the sake of simplicity, we will use the same notation. We observe that even if the original minimum cost flow problem does not have multiple arcs, our algorithm can produce multiple arcs through the contraction of certain nodes. For this reason, it is important to observe that all of the analysis in this

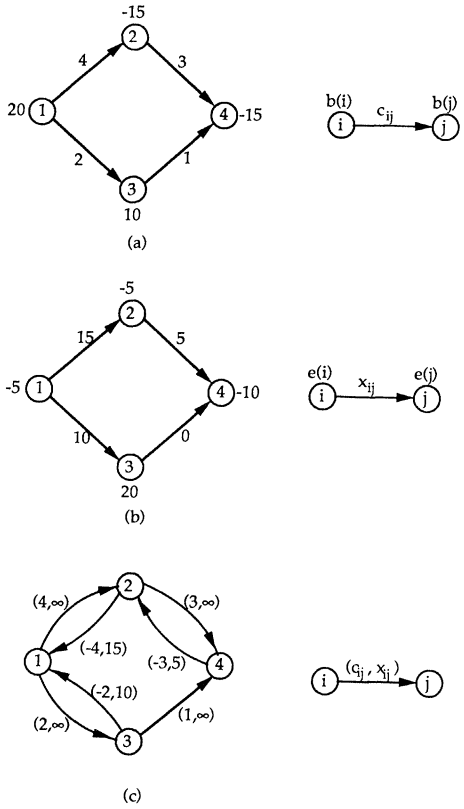

Figure 2. The construction of the residual network.

paper extends readily to the case in which multiple arcs are present. Moreover, most of the popular data structures available to implement network flow algorithms, such as the forward star representation and the adjacency list representation, are capable of handling multiple arcs.

2. OPTIMALITY CONDITIONS

In this section, we discuss the optimality conditions for the uncapacitated minimum cost flow problem and prove some basic results that our algorithms use. We first state the dual of the minimum cost flow problem. We associate a dual variable $\pi(i)$ with the mass balance constraint of node i in (1b). In terms of these variables, the dual of the minimum cost flow problem is as follows.

$$\text{Maximize} \sum_{i \in N} b(i)\pi(i) \tag{2a}$$

subject to

$$\pi(i) - \pi(j) \leqslant c_{ij} \quad \text{for all } (i, j) \in A. \tag{2b}$$

In the subsequent discussion, we refer to $\pi(i)$ as the *potential* of node i. Let $c_{ij}^{\pi} = c_{ij} - \pi(i) + \pi(j)$. We refer to c_{ij}^{π} as the *reduced cost* of arc (i, j). The complementary slackness conditions for the uncapacitated minimum cost flow problem are:

$$c_{ij}^{\pi} > 0 \Rightarrow x_{ij} = 0; \tag{3a}$$

$$x_{ij} > 0 \Rightarrow c_{ij}^{\pi} = 0. \tag{3b}$$

In terms of the residual network $G(x)$, the complementary slackness conditions can be simplified as:

$$c_{ij}^{\pi} \geqslant 0 \quad \text{for every arc } (i, j) \text{ in } G(x). \tag{4}$$

It is easy to show that (3) and (4) are equivalent (see, for example, Ahuja, Magnanti and Orlin). We refer to (3a)–(3b) or (4) as the optimality conditions for the minimum cost flow problem. We call a feasible flow x of the minimum cost flow problem an *optimum flow* if there exists a set of node potentials π so that the pair (x, π) satisfies the optimality conditions. Similarly, we call a set of node potentials π an *optimum potential* if there exists a feasible flow x so that the pair (x, π) satisfies the optimality condition.

We point out that our strongly polynomial time algorithm essentially solves the dual minimum cost flow problem. It first obtains optimum node potentials and then uses these to obtain an optimum flow. Our minimum cost flow algorithms use the following well known results.

Lemma 1. *If x is an optimal flow and if π is an optimal set of potentials, then the pair (x, π) satisfies the optimality conditions.*

Lemma 2. *Let π be any set of node potentials. If x is an optimum flow of the minimum cost flow problem with the cost of each arc $(i, j) \in A$ as c_{ij}, then it is also an optimum flow of the minimum cost flow problem with the cost of each arc $(i, j) \in A$ as $c_{ij}^{\pi} = c_{ij} - \pi(i) + \pi(j)$.*

Our minimum cost flow algorithms will maintain a pseudoflow x at every step that satisfies the optimality conditions. The algorithms proceed by augmenting flows along shortest paths where the length of an arc is defined to be its reduced cost. We show in the following lemmas that the pseudoflow obtained by these augmentations also satisfies the optimality condition.

Lemma 3. *Suppose that a pseudoflow (or a flow) x satisfies the optimality condition (4) with respect to some node potentials π. Let the vector d represent the shortest path distances from some node s to all other nodes in the residual network $G(x)$ with c_{ij}^{π} as the length of an arc (i, j). Then the following properties*

are valid:

a. *The pseudoflow x satisfies the optimality condition with respect to the node potentials $\pi' = \pi - d$.*
b. *The reduced costs $c_{ij}^{\pi'}$ are zero for all arcs (i, j) in a shortest path from node s to some other node.*

Proof. Since x satisfies the optimality condition (4) with respect to π, we have $c_{ij}^{\pi} \geq 0$ for every arc (i, j) in $G(x)$. Furthermore, since the vector d represents shortest path distances with c_{ij}^{π} as arc lengths, it satisfies the shortest path optimality condition (see, e.g., Ahuja, Magnanti and Orlin), that is

$$d(j) \leq d(i) + c_{ij}^{\pi} \quad \text{for all } (i, j) \text{ in } G(x). \tag{5}$$

Substituting $c_{ij}^{\pi} = c_{ij} - \pi(i) + \pi(j)$ in (5), we obtain $d(j) \leq d(i) + c_{ij} - \pi(i) + \pi(j)$. Alternatively, $c_{ij} - (\pi(i) - d(i)) + (\pi(j) - d(j)) \geq 0$, or $c_{ij}^{\pi'} \geq 0$. This conclusion establishes part a of the lemma.

Consider next a shortest path from node s to some node k. For each arc (i, j) in this path, $d(j) = d(i) + c_{ij}^{\pi}$. Substituting $c_{ij}^{\pi} = c_{ij} - \pi(i) + \pi(j)$ in this equation, we obtain $c_{ij}^{\pi'} = 0$. This conclusion establishes part b.

The following result is an immediate consequence of the preceding lemma.

Lemma 4. *Suppose that a pseudoflow (or a flow) x satisfies the optimality condition* (4) *and x' is obtained from x by sending flow along a shortest path from node s to some other node k, then x' also satisfies the optimality condition.*

Proof. Define the potentials π and π', as in Lemma 3. The proof of Lemma 3 implies that for every arc (i, j) in the shortest path P from node s to the node k, $c_{ij}^{\pi'} = 0$. Augmenting flow on any such arc might add its reversal (j, i) to the residual network. But since $c_{ij}^{\pi'} = 0$ for each arc $(i, j) \in P$, $c_{ji}^{\pi'} = 0$ and the arc (j, i) also satisfies the optimality condition. The lemma follows.

3. THE EDMONDS-KARP SCALING TECHNIQUE

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

The basic ideas behind the RHS-scaling algorithm are as follows. Let x be a pseudoflow and let π be a vector of node potentials. For some integer Δ, let $S(\Delta) = \{i \in N: e(i) \geq \Delta\}$ and let $T(\Delta) = \{i \in N: e(i) \leq -\Delta\}$. We call the pair (x, π) Δ-*optimal*, if (x, π) satisfies the optimality conditions and either $S(\Delta) = \emptyset$ or $T(\Delta) = \emptyset$. Clearly, if $x = 0$ and $\pi = 0$, then (x, π) is Δ-optimal for every $\Delta > U$. If x is integral and if (x, π) is Δ-optimal for some $\Delta < 1$, then x is an optimum flow. The RHS-scaling algorithm starts with a Δ-optimal pseudoflow with $\Delta = 2^{\lceil \log U \rceil}$, at each iteration it replaces Δ by $\Delta/2$ and obtains a Δ-optimal pseudoflow, and terminates when $\Delta < 1$. Given a 2Δ-optimal pseudoflow, the scaling algorithm obtains a Δ-optimal pseudoflow by solving at most n shortest path problems. A maximal period during which Δ remains unchanged is called a Δ-*scaling phase*. We refer to Δ as the *scale factor*. Clearly, there are $\lceil \log U \rceil + 1$ scaling phases. A formal description of the RHS-scaling algorithm is given in Figure 3.

We next prove the correctness and complexity of the RHS-scaling algorithm.

Lemma 5. *At every step of the RHS-scaling algorithm, the flow and the residual capacity of every arc in the residual network is an integral multiple of the scale factor Δ.*

Proof. We show this result by performing induction on the number of augmentations and adjustments in the scale factor. This result is clearly satisfied at the

```
algorithm RHS-SCALING;
begin
    set x := 0, π := 0; and e := b;
    set U := max { | b(i) | : i ∈ N};
    Δ := 2^⌈log U⌉;
    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
            choose some k ∈ S(Δ) and v∈ T(Δ) ;
            considering reduced costs as arc lengths, 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 v;
            update x, r, e, S(Δ) and T(Δ);
        end;
    end; {Δ-scaling phase}
    Δ = Δ/2;
end;
```

Figure 3. The RHS-scaling algorithm.

beginning of the RHS-scaling algorithm because the initial flow is 0 and all arc capacities are ∞. Each augmentation modifies the residual capacities of arcs by 0 or Δ units and, consequently, preserves the induction hypothesis. Reducing the scale factor of Δ to $\Delta/2$ also maintains the invariant.

Lemma 5 implies that during an augmentation, Δ units of flow can be sent on any path P with positive residual capacity. Lemma 4 implies that the pseudoflow after the augmentation still satisfies the optimality condition. The algorithm terminates when all nodes are balanced; i.e., the current pseudoflow is a flow. Clearly, this flow must be optimum.

We now address the complexity of the algorithm. We need some additional notations. 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 the algorithm can find at most n augmenting paths in any scaling phase. In fact, it proves a slightly stronger result that is useful in the proof of the strongly polynomial time algorithm.

Lemma 6. *The number of augmentations in a 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. Furthermore, since the augmentation terminates at a sink node, it does not create any new active node. Thus, the number of augmentations is 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 within $O(\log U)$ scaling phases and runs in $O((n \log U)\, S(n, m, C))$ time.*

In the next section, we describe a modification of the RHS-scaling algorithm that runs in $O(n \log n\, S(n, m))$ time. This modified algorithm improves upon the previous bound when $\log U \geq \log n$.

4. THE STRONGLY POLYNOMIAL TIME ALGORITHM

In this section, we present a strongly polynomial time version of the RHS-scaling algorithm discussed in the previous section. We first introduce the idea of *contraction* in the RHS-scaling algorithm. The contraction operation is also a key idea in a number of other strongly polynomial time algorithms including those of Orlin (1984), Tardos (1985), Fujishige (1986), and Galil and Tardos (1986). We next point out that the RHS-scaling algorithm using contractions is not a polynomial time algorithm, and then we show how to modify the RHS-scaling algorithm with contractions so that it becomes a strongly polynomial time algorithm. We again emphasize that contrary to the RHS-scaling algorithm, the strongly polynomial time algorithm solves the dual minimum cost flow problem; it first obtains optimum node potentials and then uses these to obtain an optimum flow.

4.1. Contraction

The key idea that allows us to make the RHS-scaling algorithm strongly polynomial time is to identify arcs whose flow is so large in the Δ-scaling phase that they are guaranteed to have positive flow in all subsequent scaling phases. In the Δ-scaling phase, the flow in any arc can change by at most $n\Delta$ units, because there are at most n augmentations. If we sum the changes in flow in any arc over all subsequent scaling phases, the total change is at most $n(\Delta + \Delta/2 + \Delta/4 + \ldots + 1) = 2n\Delta$. It thus follows that any arc whose flow exceeds $2n\Delta$ 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 $2n\Delta$ during the Δ-scaling phase as a *strongly feasible arc.*

Our strongly polynomial time algorithm is based on the fundamental idea that any strongly feasible arc, say (k, l), can be *contracted* into a single node p. The contraction operation consists of: letting $b(p) = b(k) + b(l)$ and $e(p) = e(k) + e(l)$; replacing each arc (i, k) or (i, l) by the arc (i, p); replacing each arc (k, i) or (l, i) by the arc (p, i); and letting the cost of an arc in the contracted network equal that of the arc it replaces. We point out that the contraction operation may lead to the creation of multiple arcs, i.e., several arcs with the same tail and head nodes. We now give a theoretical justification for the contraction operation.

Lemma 7. *Suppose it is known that $x_{kl} > 0$ in an optimum solution of the minimum cost flow problem. Then with respect to every set of optimum node potentials, the reduced cost of arc (k, l) is zero.*

Proof. Let x satisfy the optimality condition (3) with respect to the node potential π. It follows from (3b) that $c^{\pi}_{kl} = 0$. We next use Lemma 2, which implies that if x satisfies the optimality condition (3) with respect to some node potential, then it satisfies this condition with respect to every optimum node potential. Hence, the reduced cost of arc (k, l) is zero with respect to every set of optimum node potentials.

Let **P** denote the minimum cost flow problem stated in (1). Suppose that we are solving the minimum cost flow problem **P** with arc costs c_{ij} by the RHS-scaling algorithm and at some stage we realize that arc (k, l) is a strongly feasible arc. The optimality condition (3b) implies that

$$c_{kl} - \pi(k) + \pi(l) = 0. \tag{6}$$

Now consider the same minimum cost flow problem, but with the cost of each arc (i, j) equal to $c'_{ij} = c^{\pi}_{ij} = c_{ij} - \pi(i) + \pi(j)$. Let **P′** denote the modified minimum cost flow problem. It follows from (6) that

$$c'_{kl} = 0. \tag{7}$$

We next observe that the problems **P** and **P′** have the same optimum solutions (see Lemma 2). Since arc (k, l) is a strongly feasible arc, it follows from the optimality condition (3b) that in **P′** the reduced cost of arc (k, l) will be zero. If π' denotes an optimum set of node potentials for **P′**, then

$$c'_{kl} - \pi'(k) + \pi'(l) = 0. \tag{8}$$

Substituting (7) in (8) implies that $\pi'(k) = \pi'(l)$.

We next observe that if we solve the dual minimum cost flow problem (2) with the additional constraint that the potentials of nodes k and l must be the same, then this constraint will not affect the optimality of the dual. But how can we solve the dual minimum cost flow problem where two node potentials must be the same? One solution is that in the formulation (2), we replace both $\pi(k)$ and $\pi(l)$ by $\pi(p)$. This substitution gives us a linear programming problem with one less dual variable (or node potential). The reader can easily verify that the resulting problem is a dual minimum cost flow problem on the network where nodes k and l have been contracted into a single node p. The purpose of the contraction operation should also be clear to the reader; it reduces the size of the network by one node which implies that there can be at most n contraction operations.

We next show that there will be a strongly feasible arc after a "sufficiently small" number of scaling phases.

Lemma 8. *Suppose that at the termination of the Δ-scaling phase, $|b(i)| > 5n^2\Delta$ for some node i in N. Then there is an arc (i, j) or (j, i) incident to node i such that the flow on the arc is at least $3n\Delta$.*

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 $b(i) < 0$ can be proved analogously.) Clearly, the net flow going out of node i is $(b(i) - e(i))$. As fewer than n arcs emanate from i, at least one of these arcs has a flow strictly more than $(b(i) - e(i))/n \geq (5n^2\Delta - 2n\Delta)/n \geq 3n\Delta$. The lemma follows.

4.2. A Pathological Example for the RHS-Scaling Algorithm

Lemma 8 comes very close to yielding a strongly polynomial time algorithm for the following reason. At the beginning of the algorithm, all the nodes have $e(i) = b(i)$. Within $\log(5n^2) = O(\log n)$ scaling phases, Δ has been decreased by a factor of at least $5n^2$ and so $|b(i)| > 5n^2\Delta$. Then node i is incident to a strongly feasible arc, at which point two nodes get contracted into a single node. The analysis presented later will show that we will charge each augmentation to a node, and one can show that $O(\log n)$ augmentations are charged to each original node i. This *almost* leads to an $O(n \log n\ S(n, m))$ time algorithm. There is, however, a difficulty which is illustrated in Figure 4. The problem lies with the nodes that are created via a contraction.

In Figure 4a, we give the initial node excesses and arc flows where M represents a large number. Since the initial flow is zero, $b(i) = e(i)$ for every node i. Assume that all the arc costs are 0. In the $8M$-scaling phase, $8M$ units of flow are sent from node 3 to node 2. Subsequently, in each of the $4M$, $2M$ and M-scaling

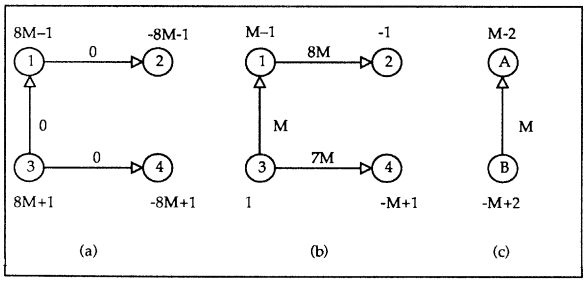

Figure 4. A pathological example for the RHS-scaling algorithm.

phases, flow is sent from node 1 to node 4. In Figure 4b, we give the flows at the end of M-scaling phase. At this point, we can contract the arcs (1, 2) and (3, 4), but not the arc (3, 1). The resulting contracted graph is given in Figure 4c. Observe that $b(A) = -2$ and $b(B) = 2$, although $e(A) = M - 2$ and $e(B) = -M + 2$. 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 two units from B to A) so that this arc gets contracted and the algorithm terminates. Thus, the algorithm is not strongly polynomial time.

We summarize the difficulty. It is possible to create contracted nodes in which the excess/deficit is comparable to Δ and this is much larger than the supply/demand at the node. Therefore, a node can be regenerated a large number of times. To overcome this difficulty we observe that the bad situation occurs infrequently; in particular, when the excess $e(k) = \Delta - \epsilon$ for some very small positive ϵ, and $b(k) = -\epsilon$, or else when $e(k) = -\Delta + \epsilon$ for some very small positive ϵ, and $b(k) = \epsilon$.

To overcome this difficulty, we modify the augmentation rules slightly. Rather than requiring the excess of the initial node of the augmenting path to be at least Δ, we require that it be at least $\alpha\Delta$ for some constant α with $\frac{1}{2} < \alpha < 1$. Similarly, we require that the terminal node of the augmenting path has a deficit of at least $\alpha\Delta$.

4.3. The Algorithm

Our strongly polynomial algorithm is the RHS-scaling algorithm with the following modifications:

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

2. We allow augmentations from a node i with $e(i) \geq \alpha\Delta$ to a node j with $e(j) \leq -\alpha\Delta$. If $\frac{1}{2} < \alpha < 1$, the algorithm is strongly polynomial. If $\alpha = 1$, then the algorithm is not strongly polynomial.

3. We no longer require Δ to be a power of two, and we no longer require Δ to be divided by two at each iteration.

4. Whereas the RHS-scaling algorithm solves the primal minimum cost flow problem (1) and obtains an optimum flow, the strongly polynomial time algorithm solves the dual minimum cost flow problem (2). This algorithm first obtains an optimum set of node potentials for (2) and then uses it to determine an optimum flow.

Our strongly polynomial algorithm is given in Figure 5. In the algorithmic description, we denote the contracted network by $\hat{G} = (\hat{N}, \hat{A})$.

We point out that in consecutive scaling phases, the strongly polynomial time algorithm either decreases Δ by a factor of two or by a factor larger than two. Whenever the algorithm decreases Δ by a factor larger than two, then we call this step a *complete regeneration*. We also point out that whenever the algorithm contracts two nodes into a single node, it redefines the arc costs equal to the current reduced costs. We have seen earlier in Lemma 2 that redefining the arc costs does not affect the optimal solution of the minimum cost flow problem.

4.4. Correctness and Complexity of the Algorithm

We now prove the correctness of the strongly polynomial time algorithm.

Lemma 9. *At every step of the strongly polynomial time algorithm, the flow and the residual capacity of each arc (i, j) in the residual network is an integral multiple of the scale factor Δ.*

```
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 Â and e(i) < Δ for all i ∈ N̂ then
            Δ := max {e(i) : i ∈ N̂};
        {Δ-scaling phase begins here}
        while there is an arc (i, j) ∈ Â with x_ij ≥ 3nΔ do contract nodes i and j and
            define arc costs equal to their reduced costs;
        S(Δ) := {i ∈ N̂ : e(i) ≥ αΔ};
        T(Δ) := {i ∈ N̂ : e(i) ≤ -αΔ};
        while S(Δ) ≠ ø and T(Δ) ≠ ø do
        begin
            choose some k ∈ S(Δ) and v ∈ T(Δ);
            considering reduced costs as arc lengths, compute shortest
                path distances d(·) in G(x) from node k to all other nodes;
            π(i) := π(i) – d(i), for all i ∈ N̂;
            augment Δ units of flow along the shortest
                path from node k to node v;
            update x, r, e, S(Δ) and T(Δ);
        end;
        {Δ-scaling phase ends here}
        Δ = Δ/2;
    end;
    uncontract the network and compute optimum arc flows;
end;
```

Figure 5. The strongly polynomial minimum cost flow algorithm.

Proof. We show this result by performing induction on the number of augmentations, contractions, and adjustments in the scale factor. The initial flow is 0, all arc capacities are ∞, and hence the result is true. Each augmentation carries Δ units of flow and preserves the induction hypothesis. Although the contraction operation may delete some arcs, it does not change flows on arcs in the contracted network. At the end of a scaling phase, either Δ is replaced by $\Delta/2$ or by $\max\{e(i):i \in \hat{N}\}$. The former case clearly preserves the induction hypothesis, and the latter case occurs when all arc flows are zero and the hypothesis is again satisfied.

The preceding lemma implies that in each augmentation we can send Δ units of flow. The algorithm always maintains a pseudoflow that satisfies the optimality condition and terminates when the pseudoflow is a flow. Thus, the algorithm terminates with an optimum flow in the contracted network. We will describe later in subsection 4.5 how to expand the contracted network and obtain an optimum flow in the uncontracted network.

We next analyze the complexity of the strongly polynomial time algorithm. The complexity proof of the algorithm will proceed in the following manner. Let Δ' and Δ be the scale factors in two consecutive scaling phases and recall that $\Delta' \geq 2\Delta$. A node k is called *regenerated* if at the beginning of the Δ-scaling phase, $\alpha\Delta \leq |e(k)| < \alpha\Delta'$. We point out that the new nodes that are formed during contractions in the Δ-scaling phase are not called regenerated nodes in that phase. We first show that the total number of augmentations in the algorithm is at most n plus the number of regenerated nodes. We next show that the first time node k is regenerated, it satisfies $|e(k)| \leq 2\alpha|b(k)|/(2 - 2\alpha)$. This result, in conjunction with Lemma 8, implies that for a fixed α, any node is regenerated $O(\log n)$ times before it is contracted. Hence, we obtain a bound of $O(n \log n)$ on the number of augmentations. Finally, we show that the number of scaling phases is also $O(n \log n)$ to conclude the proof of the algorithm.

Lemma 10. *The number of augmentations during the Δ-scaling phase is bounded by the number of regenerated nodes plus the number of contractions in that phase.*

Proof. Let Δ' be the scaling factor in the previous scaling phase. Then $\Delta \leq \Delta'/2$. At the end of the previous scaling phase, either $S(\Delta') = \emptyset$ or $T(\Delta') = \emptyset$. We consider the case when $S(\Delta') = \emptyset$. (A similar proof can be given when $T(\Delta') = \emptyset$.) We consider the potential function $F = \sum_{i \in S} \lfloor e(i)/\alpha\Delta \rfloor$. Each augmentation sends Δ units of flow from a node in S, and hence, F decreases by at least one unit. Thus, the total number of augmentations is bounded by the initial value of F minus the final value of F plus the total increase in F.

It is easy to see that at the beginning of the Δ-scaling phase, F is no more than the number of regenerated nodes. Each node $i \notin S$ contributes 0 to F. If $\Delta = \Delta'/2$, then $e(i) < 2\alpha\Delta$ and so $\lfloor e(i)/(\alpha\Delta) \rfloor < 2$. If $\Delta < \Delta'/2$, then $e(i) \leq \Delta$ and $\lfloor e(i)/(\alpha\Delta) \rfloor < \lfloor 1/\alpha \rfloor \leq 1$. At the end of the last scaling phase, $F \leq 0$. Furthermore, notice that a contraction can increase F by at most one, because for all real numbers $e(i)$ and $e(j)$, $\lfloor e(i) + e(j) \rfloor \leq \lfloor e(i) \rfloor + \lfloor e(j) \rfloor + 1$. The lemma now follows.

Lemma 11. *In the Δ-scaling phase, if $x_{ij} \geq 3n\Delta$ for some arc (i, j) in $\hat{A}$, then $x_{ij} > 0$ in all subsequent scaling phases.*

Proof. In the absence of contractions, the flow changes on any arc due to augmentations in all subsequent scaling phases is at most $n(\Delta + \Delta/2 + \Delta/4 + \ldots + 1) = 2n\Delta$. Each contraction causes at most one additional augmentation (see Lemma 10) and there are at most $n - 1$ contractions. Thus, the total flow change on any arc is less than $3n\Delta$ and the lemma follows.

Lemma 12. *At each stage of the algorithm, $e(k) \equiv b(k) \bmod \Delta$ for every node $k \in \hat{N}$.*

Proof. The value $e(k)$ is $b(k)$ minus the flow across the cut $(k, \hat{N} - k)$. By Lemma 9, each arc flow is a multiple of Δ. Hence, $e(k) \equiv b(k) \bmod \Delta$.

Lemma 13. *The first time that a node k is regenerated, it satisfies $|e(k)| \leq 2\alpha|b(k)|/(2 - 2\alpha)$.*

Proof. Suppose that node k is regenerated for the first time at the beginning of the Δ-scaling phase. If there has been a complete regeneration before the beginning of the phase, then each arc flow is zero and $e(k) = b(k)$; hence, the lemma is true for this case. In case there has not been a complete regeneration, then all arc flows are integral multiples of 2Δ. Consider first the case when $e(k) > 0$. Clearly, $\alpha\Delta \leq e(k) < 2\alpha\Delta$. Since $e(k) \equiv b(j) \bmod(2\Delta)$, it follows that $e(k) = b(k) + (2\Delta)w$ for some integral multiple w. If $w = 0$, then $e(k) = b(k)$ and the lemma is true. If $w \geq 1$, then $e(k) \geq b(k) + 2\Delta$. Alternatively, $b(k) \leq e(k) - 2\Delta$. Since node k is regenerated at the Δ-scaling phase, it follows that $e(k) < 2\alpha\Delta$, substituting which in the preceding inequality yields $b(k) < (2\alpha - 2)\Delta$. Since

$\alpha < 1$, $b(k) < 0$, and hence multiplying the previous inequality by -1, we get $|b(k)| > (2 - 2\alpha)\Delta$. We now use the fact that $\Delta > e(k)/2\alpha$ to obtain $(2\alpha/(2 - 2\alpha))|b(k)| > e(k)$ and the lemma follows. This results when $e(k) < 0$ can be proved analogously.

Lemma 14. *Each node is regenerated $O(\log n)$ times.*

Proof. Suppose that a node k is regenerated for the first time at the Δ^*-scaling phase. Let $\alpha^* = 2\alpha/(2 - 2\alpha)$. Then $\alpha\Delta^* \leq |e(k)| \leq \alpha^*|b(k)|$, where the second inequality follows from Lemma 13. After $\lceil \log(5\alpha^* n^2/\alpha) \rceil = O(\log n)$ scaling phases, the scale factor is at most $\Delta^*/5\alpha^* n^2 \leq |b(k)|/5n^2$, and by Lemma 8, there exists a strongly feasible arc emanating from node k. The node k then contracts into a new node and is (vacuously) not regenerated again.

Hence, the following theorem is immediate.

Theorem 2. *The total number of augmentations over all scaling phases is $O(n \log n)$.*

We next bound the number of scaling phases performed by the algorithm.

Theorem 3. *The algorithm performs $O(n \log n)$ scaling phases.*

Proof. 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. Suppose that there is a node k for which $|e(k)| > \Delta/5n^2$. We assume that $e(k) > 0$; the case when $e(k) < 0$ can be proved similarly. Then within $O(\log n)$ scaling phases, node k is regenerated and within a further $O(\log n)$ scaling phase, there is a contraction. Thus, overall, this case can occur $O(n \log n)$ times.

We now consider the case when $|e(k)| \leq \Delta/5n^2$ for each node k and all arcs in the contracted graph have zero flow. Then we set Δ to $\max\{e(i): i \in \hat{N}\}$ 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 $|e(k)| < \Delta/5n^2$ for each node k and there is some arc, say (k, l), with positive flow. By Lemma 9, $x_{kl} \geq \Delta$. In the next $\log 8n^2 = O(\log n)$ scaling phases, the flow on x_{kl} is unchanged, but Δ is replaced by $\Delta' \leq \Delta/8n^2$. At this point, the arc (k, l) is strongly feasible, and a contraction would take place. Again, the number of occurrences of this case is $O(n \log n)$.

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

Proof. We have earlier discussed the correctness of the algorithm. To complete the proof of the theorem, we only need to discuss the computational time of the algorithm. Consider the time spent in a scaling phase. Reducing the scale factor by a factor other than two requires $O(m)$ time. The contractible arcs can also be identified in $O(m)$ time. The time needed to identify the sets $S(\Delta)$ and $T(\Delta)$ is $O(n)$ even if these sets may be empty. Since there are $O(n \log n)$ scaling phases, these operations require $O((n \log n)m)$ total time. The number of augmentations in the algorithm is $O(n \log n)$. Each augmentation involves solving a shortest path problem and augmenting flow along this path. The time needed to perform these operations is clearly $O((n \log n)S(n, m))$. The proof of the theorem is complete.

4.5. Expansion of Contracted Nodes

We will now explain how to expand the contracted network, and in that process we will prove that the algorithm determines an optimum solution of the minimum cost flow problem. The algorithm first determines an optimum set of node potentials of the problem (i.e., an optimum solution of the dual problem), and then by solving a maximum flow problem determines an optimum flow. The algorithm obtains an optimum set of node potentials for the original problem by the repeated use of the following result.

Lemma 15. *Let* **P** *be a problem with arc costs c_{ij} and* **P'** *be the same problem with arc costs as $c_{ij} - \pi(i) + \pi(j)$. If π' is an optimum set of node potentials for problem* **P'**, *then $\pi + \pi'$ is an optimum set of node potentials for* **P**.

Proof. This property easily follows from the observation that if a solution x satisfies the optimality condition (4) with respect to the arc costs $c_{ij} - \pi(i) + \pi(j)$ and node potentials π', then the same solution satisfies the same conditions with arc costs c_{ij} and node potentials $\pi + \pi'$.

We expand (or uncontract) the nodes in the reverse order in which they were contracted in the strongly polynomial time algorithm and obtain optimum node potentials of the successive problems. In the earlier stages, between two successive problems we performed

two transformations in the following order: we replaced arc cost c_{ij} by its reduced cost $c_{ij} - \pi(i) + \pi(j)$; and we contracted two nodes k and l into a single new node p. We undo these transformations in the reverse order. To undo the latter one, we set the potentials of nodes k and l equal to that of node p, and to undo the former one, we add π to the existing node potentials. When all contracted nodes have been expanded, the resulting node potentials are an optimum set of node potentials for the minimum cost flow problem.

We next use the optimal node potentials to obtain an optimum flow by solving a maximum flow problem. There is a well known technique to accomplish it (see, e.g., Ahuja, Magnanti and Orlin) and we include it for the sake of completeness. Let π^* denote the optimum node potentials of the minimum cost flow problem. We define the reduced cost c_{ij}^{π} of an arc $(i, j) \in A$ as $c_{ij} - \pi^*(i) + \pi^*(j)$. The optimality condition (4) implies that $c_{ij}^{\pi} \geq 0$ for each arc $(i, j) \in A$. Furthermore, at optimality, no arc $(i, j) \in A$ with $c_{ij}^{\pi} > 0$ can have positive flow; then arc (j, i) will be present in the residual network with $c_{ji}^{\pi} = -c_{ij}^{\pi} < 0$, which violates (4). Hence, arcs with zero reduced cost only are allowed to have positive flow in the optimum solution. So the problem is reduced to finding a nonnegative flow in the subnetwork $G^\circ = (N, A^\circ)$, where $A^\circ = \{(i, j) \in A: c_{ij}^{\pi} = 0\}$, that meets the supply/demand constraints of nodes. We solve this problem as follows. We introduce a *super source node* s^* and a *super sink node* t^*. For each node i with $b(i) > 0$, we add an arc (s^*, i) with capacity $b(i)$, and for each node i with $b(i) < 0$, we add an arc (i^*, t) with capacity $-b(i)$. Then we solve a maximum flow problem from s^* to t^*. The solution thus obtained is feasible and satisfies the complementary slackness condition (3); therefore, it is an optimum solution of the minimum cost flow problem.

5. CAPACITATED MINIMUM COST FLOW PROBLEM

The algorithm described in Section 4 solves the uncapacitated minimum cost flow problem as a sequence of $O(n \log n)$ shortest path problems. In this section, we consider the capacitated minimum cost flow problem. We define the capacity of an arc $(i, j) \in A$ as u_{ij} and let

$$U = \max[\max\{|b(i)|: i \in N\}, \max\{u_{ij}: (i, j) \in A\}].$$

We show how the capacitated minimum cost flow problem, with m' capacitated arcs, can be solved as a sequence of $O((n + m')\log n))$ shortest path problems, where each shortest path problem takes $S(n, m)$ time.

There is a well known transformation available to convert a capacitated minimum cost flow problem to an uncapacitated one. This transformation 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 6. 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. If each arc in A is capacitated, then the transformed network is bipartite. Each node in N_2 is a demand node. When the algorithm described in subsection 4.3 is applied to the transformed network, it solves $O((n + m')\log n)$ shortest path problems and each shortest path problem takes $S((n + m'), (m + m'))$ time. If $m' = O(m)$, then the time to solve a shortest path problem is $O(m \log n)$. To reduce the time for shortest path computation to $O(m + n \log n)$, we have to solve shortest path problems more carefully. We achieve this by solving shortest path problems on a smaller network $G' = (N', A')$.

The basic idea behind the improved shortest path algorithm is as follows. Suppose that we want to solve a shortest path problem in a residual network where node k is adjacent to only two nodes i and j. Then we can eliminate node k from the network as follows. If both (i, k) and (k, j) are in the residual network, then we add the arc (i, j) with length $c_{ik} + c_{kj}$. Similarly, if both (j, k) and (k, i) are in the network, then we add the arc (j, i) with the length $c_{jk} + c_{ki}$. Then we delete the node k and the arcs incident on this node. After

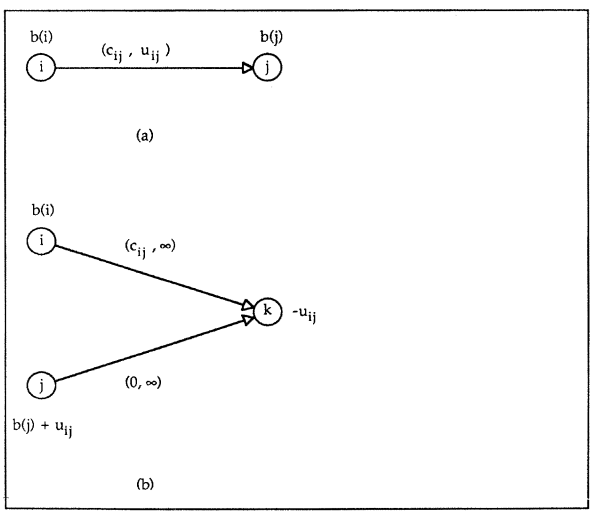

Figure 6. Transforming a capacitated arc into uncapacitated arcs.

solving the shortest path problem in the reduced network, we can obtain $d(k)$ using

$$d(k) = \min\{d(i) + c_{ik}, d(j) + c_{jk}\}.$$

Let $G(x)$ denote the residual network corresponding to the current pseudoflow x. The nodes in the residual network are either original nodes or contracted nodes. It is easy to see that each contracted node contains at least one node in N. We form the network $G' = (N', A')$ by eliminating the original nodes in N_2. We consider each original node $k \in N_2$ and replace node k and the two arcs incident on it, say (i, k) and (j, k), by at most two arcs defined as follows. If $x_{ik} > 0$, then add the arc (j, i) with length $l_{ji} = c^{\pi}_{jk} - c^{\pi}_{ik} = c^{\pi}_{jk} \geq 0$ (because by Lemma 3b, $c^{\pi}_{ik} = 0$). Furthermore, if $x_{jk} > 0$, then add the arc (i, j) with length $l_{ij} = c^{\pi}_{ik} - c^{\pi}_{jk} = c^{\pi}_{ik} \geq 0$. For each other arc (i, j) in $G(x)$ that is not replaced, we define $l_{ij} = c^{\pi}_{ij}$. Clearly, the network G' has at most n nodes, $m + 2m' = O(m)$ arcs, and in the network all arc lengths are nonnegative.

The shortest paths from some node s to all other nodes in G' can be determined in $O(m + n \log n)$ time by Fredman and Tarjan's (1987) implementation of Dijkstra's (1959) algorithm. Let $d(\cdot)$ denote the shortest path distances in G'. These distances are used to determine shortest path distances for the rest of the nodes in $G(x)$ in the following manner. Consider any original node k in N_2 on which two arcs (i, k) and (j, k) are incident. The shortest path distance to node k from node s is $\min\{d(i) + c^{\pi}_{ik}, d(j) + c^{\pi}_{jk}\}$ and the path corresponding to the smaller quantity is the shortest path. Thus, we can calculate shortest paths from node s to all other nodes in $G(x)$ in an additional $O(m)$ unit of time.

We have thus shown the following result.

Theorem 5. *The strongly polynomial algorithm solves the capacitated minimum cost flow problem in* $O(m \log n\,(m + n \log n))$ *time.*

To conclude, we have developed a strongly polynomial time algorithm for solving the capacitated minimum cost flow problem as a sequence of $O(m \log n)$ shortest path problems. This is the best running time for the minimum cost flow problem if the data are exponentially large, assuming that all operations take $O(1)$ steps. Moreover, for the case of the uncapacitated minimum cost flow problem, this algorithm is the fastest algorithm even in the case that the data satisfy the similarity assumption. The running time of our algorithm is $O((n \log n)(m + n \log n))$, which compares favorably to the running time of $O(nm \log(n^2/m)\log(nC))$ for an algorithm developed by Goldberg and Tarjan (1987) and to $O(nm \log\log U \log(nC))$ for an algorithm developed by Ahuja et al. (1988). In addition, the algorithm presented here leads to a good running time for solving the minimum cost flow problem in parallel.

ACKNOWLEDGMENT

I wish to thank Eva Tardos, Leslie Hall, Serge Plotkin, Kavitha Rajan and Bob Tarjan for valuable comments made about this research and about the presentation of this research. However, I am especially indebted to Ravi Ahuja who provided help and insight throughout this research. I also express my thanks to the associate editor and the referees for their detailed comments that lead to an improved presentation of the contents. The referees were particularly conscientious and very helpful in pointing out ways in which the exposition could be improved.

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

REFERENCES

AHUJA, R. K., A. V. GOLDBERG, J. B. ORLIN AND R. E. TARJAN. 1988. Finding Minimum Cost Flows by Double Scaling. *Math. Prog.* **53,** 243–266.

AHUJA, R. K., J. B. ORLIN AND R. E. TARJAN. 1989. Improved Time Bounds for the Maximum Flow Problem. *SIAM J. Comput.* **18,** 939–954.

AHUJA, R. K., K. MEHLHORN, J. B. ORLIN AND R. E. TARJAN. 1990. Faster Algorithms for the Shortest Path Problem. *J. ACM* **37,** 213–223.

AHUJA, R. K., T. L. MAGNANTI AND J. B. ORLIN. 1993. *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall, Englewood Cliffs, N.J.

BLAND, R. G., AND D. L. JENSEN. 1985. On the Computational Behavior of a Polynomial-Time Network Flow Algorithm. Technical Report 661, School of Operations Research and Industrial Engineering, Cornell University, Ithaca, New York.

DIJKSTRA, E. 1959. A Note of Two Problems in Connexion With Graphs. *Numeriche Mathematics* **1,** 269–271.

EDMONDS, J., AND R. M. KARP. 1972. Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems. *J. ACM* **19,** 248–264.

FREDMAN, M. L., AND R. E. TARJAN. 1987. Fibonacci Heaps and Their Uses in Network Optimization

Algorithms. In *25th IEEE Symp. on Found. of Comp. Sci.* 1984, 338–346. Also in *J. ACM* **34,** 596–615.

FUJISHIGE, S. 1986. An $O(m^3 \log n)$ Capacity-Rounding Algorithm for the Minimum Cost Circulation Problem: A Dual Framework of Tardos' Algorithm. *Math. Prog.* **35,** 298–309.

GALIL, Z., AND E. TARDOS. 1986. An $O(n^2 (m + n \log n) \log n)$ Min-Cost Flow Algorithm. In *Proceedings 27th Annual Symposium of Foundations of Computer Science* 136–146.

GOLDBERG, A. V., AND R. E. TARJAN. 1987. Solving Minimum Cost Flow Problems by Successive Approximation. In *Proceedings 19th ACM Symposium Theory of Computing*, 7–18. Full paper: 1990. *Math. Opns. Res.* **15,** 430–466.

GOLDBERG, A. V., AND R. E. TARJAN. 1988. Finding Minimum-Cost Circulations by Canceling Negative Cycles. In *Proceedings 20th ACM Symposium on the Theory of Computing*, 388–397. Full paper: 1989. *J. ACM* **36,** 873–886.

GOLDBERG, A. V., S. A. PLOTKIN AND E. TARDOS. 1988. Combinatorial Algorithms for the Generalized Circulation Problem. Laboratory for Computer Science Technical Report TM-358, MIT, Cambridge, Mass.

KING, V., S. RAO AND R. E. TARJAN. 1991. A Faster Deterministic Max-Flow Algorithm. WOBCATS.

ORLIN, J. B. 1984. Genuinely Polynomial Simplex and Nonsimplex Algorithms for the Minimum Cost Flow Problem. Technical Report No. 1615-84, Sloan School of Management, MIT, Cambridge, Mass.

ORLIN, J. B. 1988. A Faster Strongly Polynomial Minimum Cost Flow Algorithm. In *Proceedings 20th ACM Symposium on Theory of Computing*, 377–387.

ORLIN, J. B., S. A. PLOTKIN AND E. TARDOS. 1991. Polynomial Dual Network Simplex Algorithms. *Math. Prog.* (to appear).

PLOTKIN, S. A., AND E. TARDOS. 1990. Improved Dual Network Simplex. In *Proceedings of the 1st ACM-SIAM Symposium on Discrete Algorithms*, 367–376.

ROCK, H. 1980. Scaling Techniques for Minimal Cost Network Flows. In *Discrete Structures and Algorithms*, V. Page (ed.). Carl Hansen, Munich, 101–191.

TARDOS, E. 1985. A Strongly Polynomial Minimum Cost Circulation Algorithm. *Combinatorica* **5,** 247–255.

VAN EMDE BOAS, P., R. KAAS AND E. ZIJLSTRA. 1977. Design and Implementation of an Efficient Prioriti Queue. *Math. Syst. Theory* **10,** 99–127.