A FASTER STRONGLY POLYNOMIAL MINIMUM COST FLOW ALGORITHM

James B. Orlin

Sloan W.P. No. 3060-89-MS August, 1989

A FASTER STRONGLY POLYNOMIAL MINIMUM COST FLOW ALGORITHM

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)) 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 algorithm due to Galil and 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 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 (e.g., Ford and Fulkerson [1962], Fulkerson [1961], Busaker and Gowen [1961], Klein [1967], Glover, Karney and Klingman [1974], Bradley, Brown and Graves [1977], Grigoriadis [1986], Ahuja, Magnanti and Orlin [1989], Goldberg, Tardos and Tarjan [1989], and the references given in Figure 1). Researchers have developed a number of different algorithmic approaches that have lead both to theoretical and practical improvements in the running time. The table given in Figure 1 summarizes the theoretical developments in solving the minimum cost flow problem. The table reports running times for networks with n nodes and m arcs, of which m' arcs are capacitated. It is assumed that the integral cost coefficients are bounded in absolute value by C, and the integral capacities, supplies and demands are bounded in absolute value by U. The term $S(\cdot)$ is the running time for the shortest path problem and the term $M(\cdot)$ represents the running time to solve a maximum flow problem.

The running times of Figure 1 are partitioned into two groups, according as the time bound does or does not depend on the *log* of the numbers involved. The rationale for this partition is as follows: For techniques in which the running times depend on the capacities or the right hand sides or the costs, one can ask how much degradation in the running time there is as the numbers have increasingly large size or have increasingly large precision. A polynomial time algorithm whose running time depends only on the dimension of the problem (i.e., the number of nodes and arcs) and not on the precision in which the data is given is called *strongly polynomial.* A more precise definition will be given in Section 2.

In comparing the polynomial and strongly polynomial algorithms, we assume that $C = O(n^{O(1)})$ and $U = O(n^{O(1)})$. This assumption is known as the *similarity assumption* (see Gabow [1985]). Under the similarity assumption, the best bounds for the shortest path and maximum flow problems are as follows:

Polynomial Bounds	**Due to**	**Year**
$S(n, m, C) = m + n\sqrt{\log C}$	Ahuja, Mehlhorn, Orlin and Tarjan	1988
$M(n, m, C) = nm \log\left(\frac{n\sqrt{\log U}}{m} + 2\right)$	Ahuja, Orlin and Tarjan	1988

Strongly Polynomial Bounds	Due to	Year
$S(n, m) = m + n \log n$	Fredman and Tarjan	1984
$M(n, m) = nm \log (n^2/m)$	Goldberg and Tarjan	1986

Polynomial Algorithms

#	Due to	Year	Running Time
1	Edmonds and Karp	1972	$O((n + m') \log U\ S(n, m, C))$
2	Rock	1980	$O((n + m') \log U\ S(n, m, C))$
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	Bertsekas and Eckstein	1987	$O(n^3 \log (nC))$
7	Goldberg and Tarjan	1988	$O(nm \log n \log (nC))$
8	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	$O((n + m') \log n\ S(n, m))$

$S(n, m)$ and $S(n, m, C)$ denote the best strongly polynomial and (weakly) polynomial times needed to solve a shortest path problem. $M(n, m)$ and $M(n, m, U)$ denote the best strongly polynomial and (weakly) polynomial times needed to solve a max flow problem. U is the largest capacity of an arc. C is the largest cost of an arc (in absolute value).

FIGURE 1. POLYNOMIAL ALGORITHMS FOR THE MINIMUM COST FLOW PROBLEM

Edmonds and Karp [1972] were the first to solve the minimum cost flow problem in polynomial time. Their algorithm, now commonly referred to as Edmonds-Karp scaling technique, was to reduce a network flow problem to a sequence of $O((n + m') \log U))$ shortest path problems. Although Edmonds and Karp did resolve the question of whether network flow problems 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 algorithm for the minimum cost flow problem. This question is motivated in part by the existence of strongly polynomial algorithms for several subclasses of network flow problems including the assignment problem, the shortest path problem, and the maximum flow problem.

Tardos [1985] resolved this question by showing how to solve the minimum cost flow problem by solving O(m) minimum cost flow problems each of which has its cost coefficients bounded by n^2. If we were to use her technique in conjunction with Goldberg and Tarjan's [1987] recent minimum cost flow algorithm, the running time would be $O(nm^2 (\log (n^2/m)) \log n)$. As originally developed, the running time was significantly worse than this bound. Tardos [1986] subsequently gave a strongly polynomial algorithm for linear programs in which the number of bits in the constraint matrix coefficients are bounded in n. Frank and Tardos [1985] generalized the result to solving simultaneous diaphontine approximations. Orlin [1986] gave a strongly polynomial dual version of Tardos' linear programming algorithm.

Orlin [1984] and Fujishige [1986] improved the running time of the minimum cost flow problem to $O((n + m)^2 \log n\, S(n, m))$ steps. Galil and Tardos [1986] improved the time bound further to $O(n^2 \log n\, S(n, m))$. Recently, Goldberg and Tarjan [1987, 1988] developed algorithms that rely on ideas from cost scaling. As opposed to the algorithms of Tardos [1985], Fujishige [1986], and Galil and Tardos [1986], the algorithms of Orlin [1984], Goldberg and Tarjan [1987, 1988], and Orlin (presented in this paper) solve one minimum cost flow problem.

In [1984], Orlin presented the first polynomial time simplex algorithm for the minimum cost flow problem. The number of pivots is $O(m^3 \log n)$, and the running time is $O(m^2 \log n)\ (m + n \log n))$. Recently, Plotkin and Tardos [1989] using ideas developed in Orlin [1984] as well as those in this paper, reduce the number of dual simplex pivots to $O(m^2 \log n)$. Unfortunately, the running time of their strongly polynomial dual simplex algorithm requires much more computational time than the algorithm presented here.

This paper is organized as follows. In Section 2, we present a discussion on strongly polynomial algorithms. The notations and definitions are given in Section 3. A brief discussion on the optimality conditions for the minimum cost flow problem is presented in Section 4. In Section 5, 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 minimum cost flow problem as a sequence of $O(n \log U)$ shortest path problems. In Section 6, we describe how to modify the algorithm so as to solve the uncapacitated minimum cost 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 minimum cost flow problem. This generalization is described in Section 7. In Section 8, 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. A preliminary revision of this paper is given in Orlin [1988]. The version presented here is somewhat simpler and uses some ideas on strongly polynomial algorithms as presented in Goldberg, Tardos and Tarjan [1989].

2. On Strongly Polynomial Algorithms

We call a network algorithm *strongly polynomial* if it satisfies the following two conditions:

SP1. The number of arithmetic operations is polynomially bounded in n and m.

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

Here, we are using a more restrictive notion of strong polynomiality than has been used by others including Megiddo [1983] and Tardos [1986]. Any algorithm satisfying the above definition of strong polynomiality will also satisfy the other definitions as well. Actually, we will permit other operations that can be easily simulated by a small number of additions, subtractions, and comparisons. For example, multiplication of a number x by an integral constant k can be simulated by O(log k) additions.

There are several theoretical motivations for studying algorithms that are strongly polynomial. (See, for example, Goldberg, Tardos and Tarjan [1989] and Johnson [1987]. First, these algorithms help to isolate the cause of the complexity of a problem. In some numerical problems such as computing the greatest common divisor of two numbers, the computational complexity stems from the size of the numbers rather than the dimension of the problem (which is 2). In other problems, such as finding the median of n numbers, the complexity of the problem increases with the dimension of the problem rather than the size of the numbers. For minimum cost flow problems, one may wonder whether the complexity of the minimum cost flow problem is really due in part to the size of the numbers involved.

Also, for a particular algorithm that appears to run in time polynomial in the size of the integers, one may ask whether this apparent increase in complexity is realizable. An example is Dantzig's pivot rule for the assignment problem. Orlin [1985] proved that the number of pivots increases at most linearly in log C as C grows large, but this time bound is not realizable for all sufficiently large C.

One possible motivation for strongly polynomial algorithms is in subroutines in which non-rational data is used. For example, Goldberg, Plotkin and Tardos [1988] solved the generalized max flow problem as a sequence of min cost flow problems. Each min cost flow problem had cost coefficients which were obtained as logs of integers, and thus these cost coefficients were mostly irrational. The fastest solution technique for solving these minimum cost flow problems is the algorithm presented in this paper.

One motivation that is perhaps illusory is the desire for more efficient algorithms in practice. A strongly polynomial algorithm is not guaranteed to be better than an algorithm that is "weakly polynomial." First of all, log U and log C are not very large in practice. For almost all applications, these numbers are bounded by 20, and the log C and log U terms are clearly preferred to strongly polynomial bounds such as n^2 or n. Second,

the strongly polynomial algorithms typically see a competitive advantage over the usual polynomial algorithms only when the values C and U are exponentially large. Thus for realistic data, they offer no improvement over the previous algorithms. As one can see from this paper, the strongly polynomial algorithm for the minimum cost flow problem has the same running time as Edmonds-Karp Scaling technique whenever the capacities and right hand sides are numbers that are polynomially bounded in n.

Moreover, the computation times for strongly polynomial algorithms do not reflect the real time to perform arithmetic calculations on large integers. On a computer, there is a maximum integer K^* that can be stored in a single memory location. If an integer x is much larger than K^*, then x typically has to be stored as a list of smaller numbers each of which is no larger than K^*. The time to perform an arithmetic operation on x is at least the number of elements of the list, i.e., $\log x / \log K^*$. Models that more accurately reflects this increased running time are the *logarithmic* and *semi-logarithmic models of computation* as discussed in Aho, Hopcroft and Ullman [1974], Cook and Reckhow [1973], and Gabow [1985], and used in Ahuja, Orlin and Tarjan [1988] and Ahuja, Mehlhorn, Orlin and Tarjan [1988]. (In the logarithmic model K^* is assumed to be fixed in size. In the semi-logarithmic model K^* is assumed to be proportional to n.)

Although we will not discuss the logarithmic model of computation in any detail, we note that scaling techniques are ideally suited for analysis under the logarithmic model of computation because arithmetic operations in scaling techniques usually work with only the leading bits of an integer. In other words, even if integer x is exponentially large, the arithmetic operations on x typically take O(log n) steps rather than O(log x) steps since they involve only the O(log n) leading bits of x. Thus our strongly polynomial algorithm appears even more attractive if we adopt the logarithmic model of computation.

3. Notations 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 in which there is no upper bound on the flow on any arc. Let $n = |N|$ and $m = |A|$. We associate with each node $i \in N$ a real number b(i) which indicates the supply (demand) of the node if $b(i) > 0$ ($b(i) < 0$). Let $U = \max \{b(i) : i \in N\}$. The uncapacitated minimum cost flow problem can be stated as follows.

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

subject to

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

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

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

Assumption 1. (Dual feasibility) All arc costs are non-negative.

Assumption 2. (Strong connectedness) For all node i and j 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 arc (i, j) or (j, i) in A, but not both.

The first assumption is 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 non-negative. We first compute the shortest path distances d(i) from node i to all other nodes. 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 non-negative for all the arcs. This transformation does not affect the optimum solution of the problem.

The second assumption can be met by adding an artificial node s, 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.

Finally, there is some loss of generality in the third assumption. Due to this assumption, the residual network (discussed later in this section) has at most one arc (i, j), and at most one arc (j, i) for any pair of nodes i and j. But this assumption is made solely for expository reasons. By using more complex notations we can easily relax this assumption on arc multiplicity, and extend the theory presented here to the case in which there are multiple arcs.

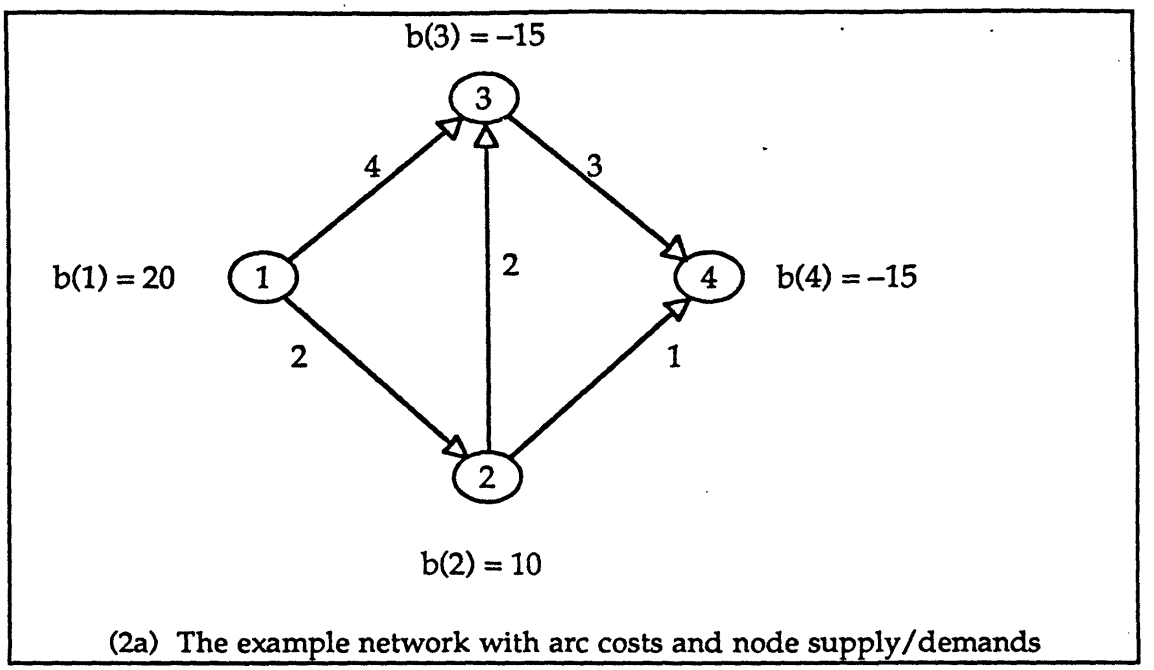
A *pseudo-flow* x is a function $x : A \rightarrow R$ satisfying only the non-negativity constraints, i.e., $x \geq 0$. The algorithm described in this paper maintains a pseudo-flow at each intermediate step.

For a given pseudo-flow x, we define the *imbalance* at node i to be

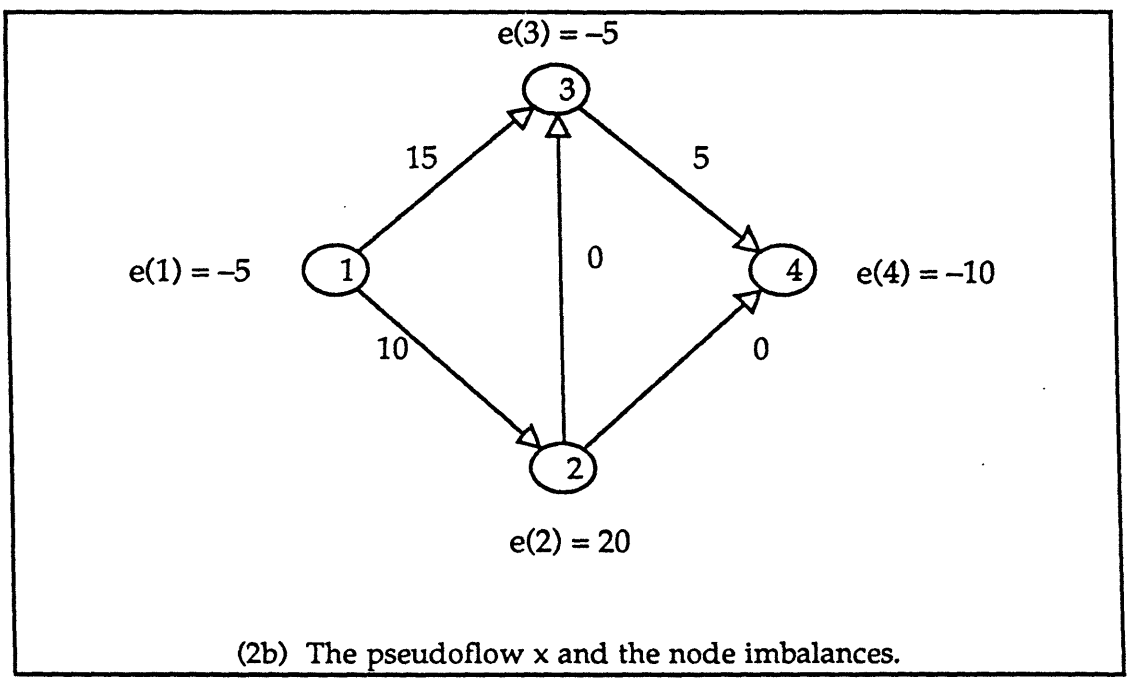
$$e(i) = b(i) + \sum_{\{j:(j,i) \in A\}} x_{ji} - \sum_{\{j:(i,j) \in A\}} x_{ij}, \quad \text{for each } i \in N.$$

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.

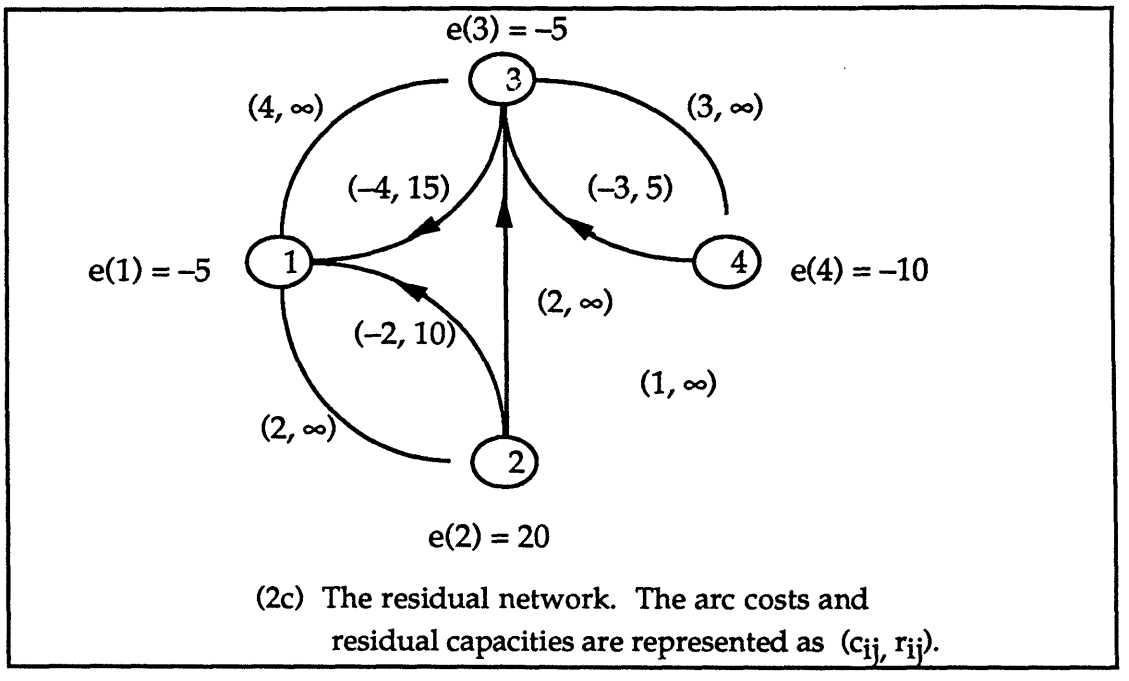
For any pseudo-flow x, we define the *residual network* G(x) as follows: We replace each arc $(i, j) \in A$ by two arcs (i, j) and (j, i). The arc (i, j) has cost c_{ij} and a *residual capacity* $r_{ij} = \infty$, and the arc (j, i) has cost $-c_{ij}$ and residual capacity $r_{ij} = c_{ij}$. The residual 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 pseudo-flow. For an illustration of a pseudo-flow and the corresponding residual network, see Figure 2.



(2a) The example network with arc costs and node supply/demands



(2b) The pseudoflow x and the node imbalances.



(2c) The residual network. The arc costs and residual capacities are represented as (c_{ij}, r_{ij}).

FIGURE 2. ILLUSTRATIONS OF PSEUDO-FLOW AND RESIDUAL NETWORK.

4. Optimality Conditions

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

C1. x is a feasible flow.

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

We say that the pair (x, π) is *dual feasible* if C2 is satisfied. The algorithms described in this paper always maintain a pseudo-flow and a vector of node potentials that are dual feasible, and they successively reduce the primal infeasibility of the solution. Our algorithms rely on the following results.

Lemma 1. If x and $\bar{c}$ satisfy C2, then for each arc $(i, j) \in A$ with $x_{ij} > 0$ we must have $\bar{c}_{ij} = 0$.

Lemma 1. If x and $\bar{c}$ satisfy C2, then for each arc (i , j) ∈ A with $x_{ij} > 0$ we must have $\bar{c}_{ij} = 0$.

Proof. If $x_{ij} > 0$ for some arc (i, j) then both (i, j) and (j, i) are present in G(x). The condition C2 implies that $\bar{c}_{ij} > 0$ and $\bar{c}_{ji} \geq 0$. Since $\bar{c}_{ij} = -\bar{c}_{ji}$, these two inequalities can be satisfied only if $\bar{c}_{ij} = \bar{c}_{ji} = 0$. ■

Corollary 1. Suppose x^* is an optimum flow and $x^*_{ij} > 0$. Then for any dual optimum solution π , $\bar{c}_{ij} = 0$.

We remark that Corollary 1 is the usual complementary slackness conditions of linear programming.

Lemma 2. Suppose a pseudoflow x satisfies the dual feasibility condition C2 with respect to the node potentials π. Furthermore, suppose that x' is obtained from x by sending flow along a shortest path from a node k to a node l in G(x). Then x' also satisfies the dual feasibility conditions with respect to some node potentials.

Proof. Since x satisfies the dual feasibility conditions with respect to the node potentials π, we have $\bar{c}_{ij} \geq 0$ for all (i, j) in G(x). Let d(v) denote the shortest path distances from node k to any node v in G(x) with respect to the arc lengths $\bar{c}_{ij}$. We claim that x also satisfies the dual feasibility conditions with respect to the potentials $\pi' = \pi - d$. The shortest path optimality conditions (i.e., see Lawler [1976]) imply that

$$d(j) \leq d(i) + \bar{c}_{ij}, \text{ for all (i, j) in G(x).}$$

Substituting $\bar{c}_{ij} = \bar{c}_{ij} - \pi(i) + \pi(j)$ in these conditions and using $\pi'(i) = \pi'(j) - d(i)$ yields

$$\bar{c}_{ij} = c_{ij} - \pi'(i) + \pi'(j) \geq 0, \text{ for all (i, j) in G(x).}$$

Hence, x satisfies C2 with respect to the node potentials π'. Next note that $\bar{c}_{ij} = 0$ for every arc (i, j) on the shortest path P from node k to node *l*, since $d(j) = d(i) + \bar{c}_{ij}$ for every arc (i, j) ∈ P and $\bar{c}_{ij} + c_{ij} - \pi(i) + \pi(j)$.

We are now in a position to prove the lemma. Augmenting flow along any arc in P maintains the dual feasibility condition C2 for this arc. Augmenting flow on an arc (i, j) may add its reversal (j, i) to the residual network. But since $\bar{c}_{ij} = 0$ for each arc $(i, j) \in P$, $\bar{c}_{ji} = 0$, and so arc (j, i) also satisfies C2. ■

The node potentials play a very important role in this algorithm. Besides using them to prove the correctness of the algorithm, we use them to ensure that the arc lengths are nonnegative, thus enabling us to solve the shortest path subproblems more efficiently.

Let P be the minimum cost flow problem stated in (1) and P' be a minimum cost flow problem with the same supply/demand constraints but arc costs defined as $c'_{ij} = c_{ij} - \pi(i) + \pi(j)$ for some node potentials π. We call problem P and P' as *primal equivalent problems*. We need the following property of primal equivalent problems.

Lemma 3. *Primal equivalent problems have the same primal optimum solutions.*

Proof. Observe that

$$\sum_{(i,j)\in A} c'_{ij} x_{ij} = \sum_{(i,j)\in A} c_{ij} x_{ij} - \sum_{(i,j)\in A} \pi(i) x_{ij} + \sum_{(i,j)\in A} \pi(j) x_{ij},$$

$$= \sum_{(i,j)\in A} c_{ij} x_{ij} - \sum_{i\in N} \pi(i) \left[\sum_{\{j:(i,j)\in A\}} x_{ij} \sum_{\{j:(j,i)\in A\}} x_{ji} \right],$$

$$= \sum_{(i,j)\in A} c_{ij} x_{ij} + \sum_{i\in N} \pi(i)\, b(i).$$

For a given set of node potentials π, the expression $\sum_{i \in N} \pi(i)\, b(i)$ is constant.

Consequently, any solution x that minimizes $\sum_{(i,j) \in A} c_{ij} x_{ij}$ also minimizes $\sum_{(i,j) \in A} c'_{ij} x_{ij}$ and vice-versa. ■

It is worth noting that primal equivalent problems have closely related dual optimum solutions. If π' is an optimum dual solution of P', then $\pi + \pi'$ is an optimum dual solution of P.

5. 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's 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 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 algorithm.

The basic ideas behind the RHS-scaling algorithm are as follows. Let x be a pseudo-flow and let π be a vector of simplex multipliers. 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 (x, π) *Δ–optimal*, if (π, x) is dual feasible and either $S(\Delta) = \emptyset$ or $T(\Delta) = \emptyset$. If $x = 0$ and if $\pi = 0$ 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 pseudo-flow with $\Delta = 2^{\lceil \log U \rceil}$ and at each iteration it replaces Δ by $\Delta/2$ and obtains a Δ–optimal pseudo flow, until $\Delta < 1$. Given a Δ–optimal pseudo-flow, the scaling algorithm obtains a $(\Delta/2)$–optimal pseudo-flow by solving at most n shortest path problems. An iteration during which Δ remains unchanged is called a Δ-scaling phase. Clearly, there are $\lceil \log U \rceil + 1$ scaling phases. A formal description of the RHS-scaling algorithm is given in Figure 3.

```
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
            let k ∈ S(Δ) and l ∈ 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 l;
            update x, r, e, S(Δ) and T(Δ);
        end;
        Δ = Δ/2
    end; {Δ-scaling phase}
end;
```

FIGURE 3. THE RHS – SCALING ALGORITHM.

In order to prove that the RHS-scaling algorithm is correct, we will first prove 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 4. *Flow Invariant 1 is satisfied prior to and subsequent to each augmentation in the RHS-scaling algorithm.*

Proof. We show this result by performing induction on the number of augmentations, and adjustments in the scale factor. Flow Invariant 1 is satisfied at the beginning since 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 keeps the invariant satisfied. ■

Flow Invariant 1 implies that during an augmentation, Δ units of flow can be sent on any path P with positive residual capacity. Lemma 1 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 and prices are always dual feasible, the resulting flow is optimum.

We now come to 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 $S(2\Delta) \cup T2(\Delta)$ 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 algorithm.

Lemma 5. 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. Further, since the augmentation terminates at a sink node, it does not create any new active node. 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, C)) time. ■

6. The Strongly Polynomial Algorithm

In this section, we present a strongly polynomial version of the RHS-scaling algorithm discussed in the previous section. We introduce the idea of *contraction*, point out a fundamental difficulty with the usual RHS-scaling algorithm, and then present its modification which makes it strongly polynomial.

6.1. Contraction

The key idea that allows us to make the RHS-scaling algorithm strongly polynomial is in identifying arcs whose flow is so large in the Δ-scaling phase, that they are guaranteed to have positive flow in all subsequent scaling phases. A quick justification is the following. In the Δ-scaling phase, the flow in any arc can change by at most nΔ units, since there are at most n augmentations. If we sum the changes in flow in any arc over all 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 2nΔ 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 2nΔ during the Δ-scaling phase as *strongly feasible.*

Our strongly polynomial algorithm is based on the fundamental idea that any strongly feasible arc can be contracted. The contraction is also an implicit foundation of a number of strongly polynomial algorithms including those of Tardos [1985], Tardos and Galil [1986], Fujishige [1988] and Orlin [1984]. By contraction of an arc (p, q), we mean replacing nodes p and q with a single contracted node, say v, and replacing each arc (k, p) or (k, q) for any k by the arc (k, v), and replacing each arc (p, k) or (q, k) by the arc (v, k). The reduced cost of an arc is the same as that of the arc it replaces. In addition, we let b(v) = b(p) + b(q) and e(v) = e(p) + e(q). (This contraction may lead to the creation of arcs (u, v) and (v, u) for some v or two arcs labeled (u, v). We ignore these possibilities as before, but solely for notational convenience.)

We now give a theoretical justification of the contraction operation. Suppose that at some stage of the RHS-scaling algorithm we find than an arc (p, q) is strongly feasible.

Let π be the current potentials and let P' be a primal equivalent problem of the original problem (1) with arc costs redefined as $c'_{ij} = c_{ij} - \pi(i) + \pi(j)$. By Lemma 3, problems P and P' possess the same set of optimum solutions. We now show that solving the contracted problem yields an optimum dual solution to the original problem.

Lemma 6. Suppose that (i, j) is strongly feasible and $c_{ij} = 0$. Let π^* be an optimum dual solution to the problem obtained by contracting nodes i and j. Let $\pi'(k) = \pi^*(k)$ for $k \neq i$ or j and let $\pi'(i) = \pi'(j) = \pi^*(v)$. Then π' is optimum for the original problem.

Proof. Suppose that (i, j) is strongly feasible. The strong feasibility implies that there is an optimum flow x^* with $x^*_{ij} > 0$. By Corollary 1, $\pi(i) = \pi(j)$ for any optimum dual solution. Thus we may add the constraint $\pi(i) = \pi(j)$ to the dual problem.

Suppose now that we let $\pi(v) = \pi(i)$, and $\pi(v) = \pi(j)$ and we substitute $\pi(v)$ for each occurrence of $\pi(i)$ and $\pi(j)$ in the dual. This substitution does not alter the optimum solution, however, this substitution corresponds to contracting nodes i and j. ■

Lemma 6 shows that solving the contracted problem creates an optimum dual solution to the original problem. Our algorithm successively augments along shortest paths and contracts arcs until it ultimately determines an optimum dual solution. Once an optimum dual solution is determined, we can find an optimum flow by solving a maximum flow problem.

Lemma 7. 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 (i, j) or (j, i) incident to node i such that the flow on the arc is at least $4n\Delta$.

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 $b(i) < 0$ can be proved analogously. There are at most n arcs directed out from i, and at least one of these arcs has a flow at least $(b(i) - e(i))/n \geq (8n^2\Delta - 2n\Delta)/n \geq 4n\Delta$. The lemma follows. ■

6.2. A Pathological Example for RHS-Scaling Algorithm

Lemma 5 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 it is incident to a strongly basic arc, at which 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. The problem lies with the nodes that are created via a contraction.

In (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 are 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 4(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 4(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.

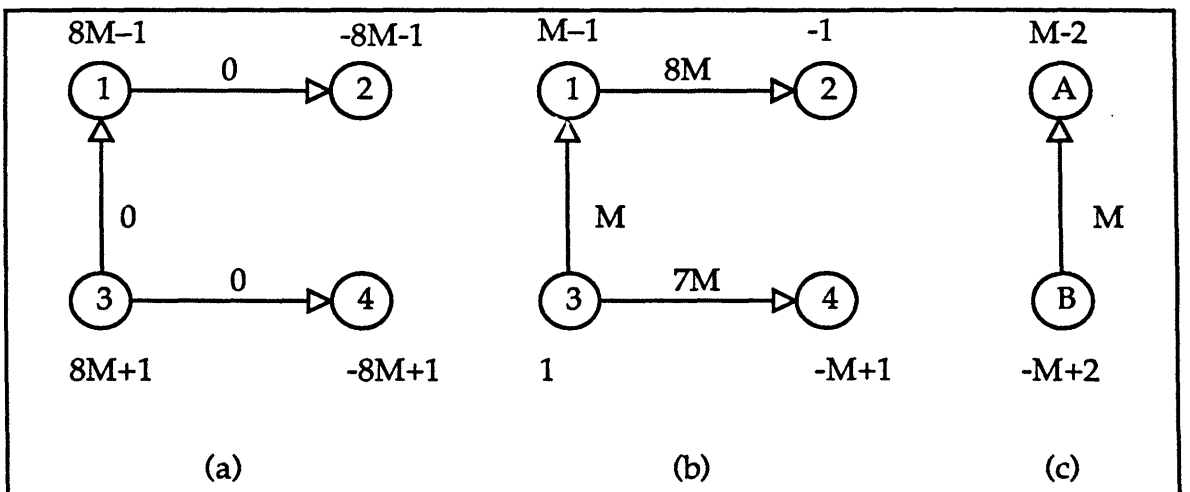


FIGURE 4. A PATHOLOGICAL EXAMPLE FOR THE RHS-SCALING ALGORITHM

We summarize the difficulty. It is possible to create contracted nodes in which the excess/deficit is comparable to Δ and this is very much greater 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 can occur only in unusual circumstances; in particular, the excess $e(i) = \Delta - \varepsilon$ for some very small positive ε, and $b(k) = -\varepsilon$, or else $e(k) = -\Delta + \varepsilon$ for some very small positive ε, and $b(k) = \varepsilon$.

In order 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 $\geq \Delta$, we require that it be $\geq \alpha\Delta$, for some constant α with $1/2 < \alpha < 1$. Similarly, we require that the terminal node of the augmenting path has deficit $\leq \alpha\Delta$.

6.3. The Algorithm

Our strongly polynomial algorithm proceeds as the RHS-scaling algorithm with the following differences. These modifications are technically required for the algorithm to be strongly polynomial.

(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 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 $1/2 < \alpha < 1$, the algorithm gives the time bound given here. If $\alpha = 1$, the algorithm is not strongly polynomial.

(3) 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.

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

```
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
        # Comment: Ĝ=(N̂, Â). is the contracted network#
        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};
        # comment : Δ-scaling phase begins here#
        while there is an arc (i, j) ∈ Â with x_ij ≥ 3nΔ do contract nodes i and j;
        S(Δ) := {i ∈ N̂ : e(i) ≥ αΔ};
        T(Δ) := {i ∈ N̂ : e(i) ≤ -αΔ};
        while S(Δ) ≠ ø and T(Δ) ≠ ø do
        begin
            let k ∈ S(Δ) and l ∈ 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 ∈ N̂;
            augment Δ units of flow along the shortest
                path from node k to node l;
            update x, r, e, S(Δ) and T(Δ);
        end;
        Δ = Δ/2;
    end;
    uncontract the network and compute optimum arc flows;
end;
```

FIGURE 5. THE STRONGLY POLYNOMIAL MINIMUM COST FLOW ALGORITHM

The strongly polynomial algorithm is the same as the RHS-scaling algorithm except that it defines S(Δ) and T(Δ) slightly differently, it contracts arcs that are strongly

feasible, and when all arc flows are 0 in the contracted network, adjusts the scale factor by a factor larger than 2. Whenever the algorithm decreases the scale factor by a factor larger than 2, then we call this step a *complete regeneration.*

We observe that if α is chosen appropriately, then each of the multiplications and divisions of the algorithm may be simulated by a constant number of additions and subtractions as per requirements of a strongly polynomial algorithm. For example, suppose that $\alpha = 2/3$. Then checking if $e(i) \geq 2\Delta/3$ is equivalent to checking if $3e(i) \geq 2\Delta$. Moreover, $3e(i)$ is obtained from $e(i)$ in 2 additions and 2Δ is obtained as the operation $\Delta + \Delta$.

We can circumvent the division of Δ by 2 by keeping Δ constant and multiplying x, r and e by 2. This alternate approach ensures that $S(\Delta)$ and $T(\Delta)$ are defined in the same way, and hence the sequence of augmentations is the same. Moreover, the algorithm terminates the scaling phases with the same optimal dual solution π, and then π may be used to create the optimal flows, as discussed in Section 6.5.

6.4 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 carries Δ units of flow. The algorithm always maintains the 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. We show later that this solution can be easily used to obtain an optimum flow in the original network.

The complexity proof of the algorithm is rather long. Let Δ' and Δ be the scale factors in two consecutive scaling phases and $\Delta' \geq 2\Delta$. A node i is called *regenerated* if at the beginning of the Δ-scaling phase, $\alpha\Delta \leq |e(i)| < \alpha\Delta'$ 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 whenever a node k is regenerated for the first time it satisfies $|e(k)| \leq 2\alpha\,|b(k)|/(2-2\alpha)$. This result in conjunction with Lemma 5 implies that for 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 are also O(n log n) to conclude the proof of the algorithm.

Lemma 8. At each stage of the algorithm, Flow Invariant 1 is satisfied.

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 Flow Invariant 1 is satisfied. Each augmentation, carries Δ units of flow and preserves the flow invariant. The contraction operation, though, may delete some arcs, but does not change arc flows in $\hat{A}$. After 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 flow invariant is again satisfied. ■

Lemma 9. In the Δ-scaling, 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 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 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 is decreased by at least 1. Thus the number of augmentations is bounded by the initial value of F plus the number of increases in F.

It can be easily seen that at the beginning of the Δ-scaling phase, F is no more than the number of regenerated nodes. This follows from the observations that (i) if $\Delta = \Delta'/2$, then $\alpha\Delta \le e(i) < 2\alpha\Delta$ for each $i \in S$, and (ii) if $\Delta < \Delta'/2$, then $e(i) \le \Delta$ for each $i \in S$ and hence $e(i) \le 2\alpha\Delta$ because $\alpha > 1/2$. Further, notice that a contraction can increase F by at most one, because for all real numbers e(i) and e(j), $\lfloor e(i) + e(j) \rfloor \le \lfloor e(i) \rfloor + \lfloor e(j) \rfloor + 1$. The lemma now follows. ■

Lemma 11. At each stage of the algorithm, $e(k) \equiv b(k) \bmod \Delta$ for every node $k \in \widehat{N}$.

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

Lemma 12. The first time that a node k is regenerated, it satisfies $|e(k)| \le 2\alpha|b(k)|/(2-2\alpha)$.

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 arcs flows are integral multiples of 2Δ. Consider first the case when e(k) > 0. Clearly, $\Delta \le e(k) < 2\alpha\Delta$. Since $e(k) \equiv b(k) \bmod (2\Delta)$, it follows that $e(k) = b(k) + w(2\Delta)$ for some integral multiple w. If w = 0, then e(k) = b(k) and the lemma is true. If $w \ge 1$, then $b(k) \le e(k) - 2\Delta < (2\alpha - 2)\Delta$ (because $e(k) < 2\alpha\Delta$). Observe that 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 result when e(k) < 0 can be proved analogously. ■

Lemma 13. Any node is regenerated O(log n) times.

Proof. Suppose a node k is regenerated for the first time at the Δ^*-scaling phase. Let $\alpha^* = 2\alpha/(2-2\alpha)$. Then $\Delta^* \le |e(k)| \le \alpha^*|b(k)|$, where the second inequality follows from Lemma 12. After $\lceil \log (8\alpha^* n^2) \rceil = O(\log n)$ scaling phases, the scale factor $\Delta^*/8\alpha^* n^2, \le |b(k)|/8n^2$, and by Lemma 4 there exists a strongly basic arc in the cut $(k, \widehat{N} - \{k\})$ The node k then contracts into a new node and is (vacuously) not regenerated again. ■

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 $2^{\lfloor \log U \rfloor}$ and the algorithm will terminate after $1 + \lfloor \log U \rfloor$ scaling phases. Since there will be no contractions, the algorithm reduces to the RHS-scaling algorithm given in Section 5. In this case, the number of augmentations is O(n log U).

We now consider the case when $U > n$. By the previous lemma, any node is regenerated O(log n) times. As there are n original nodes and at most n–1 new nodes can be formed due to contractions, the total number of regenerations is O(n log n). Lemma 10 yields that the number of augmentations is also O(n log n). ■

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 two 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. Suppose there is 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, node k is regenerated and within a further O(log n) scaling phases, there is a contraction. Thus, overall this case can occur O(n log n) times.

We now consider the case when $|e(k)| \leq \Delta/8n^2$ 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 $|e(k)| < \Delta/8n^2$ for each node k and there is some arc, say (k, *l*), with positive flow. By Flow Invariant 1, $x_{kl} \geq \Delta$. In the next $2 + \log n$ scaling iterations the flow on x_{kl} is unchanged, but Δ is replaced by $\Delta' = \Delta/4n$. 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(min (n log U, n log n) S(n, m)) time.

Proof. The algorithm terminates when all of the node imbalances are 0 and, hence, the solution is primal feasible. 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(m) time. The contractible arcs can also be identified in O(m) time. The time needed to identify the sets S(Δ) and T(Δ) is O(n) even if these sets may be empty. Since there are O(min (log U, n log n)) scaling phases, these operations require O(min (log U, n log n)m) 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)). The theorem is now immediate. ■

6.5 Expansion of Contracted Nodes

In this section, we refer to each node of $\hat{G}$ as a pseudo-node whether or not it was obtained by contracting nodes of G. Thus a pseudo-node of $\hat{G}$ may be an original node of G. We now describe how an optimum solution of the original problem P can be obtained from an optimum solution of the contracted problem. Using the optimum dual solution of the contracted problem, we first construct an optimum dual solution of a problem P' that is primal equivalent to P. Then using the optimum dual solution of problem P', we obtain a primal optimum solution of problem P' which, by Lemma 3, is also a primal optimum solution of problem P.

We uncontract the pseudo-nodes in the reverse order in which they were contracted in the strongly polynomial algorithm, and whenever a pseudo-node v is uncontracted into the two nodes (or pseudo-nodes) p and q, we assign the potentials of

nodes p and q equal to that of node v. The justification of this assignment follows from Section 6.1 where we showed that whenever nodes p and q are contracted into a single node v we replace both $\pi'(p)$ and $\pi'(q)$ by $\pi'(v)$. The potentials of nodes that were not contracted in the algorithm, do not change. When all contracted nodes have been uncontracted, we obtain a dual optimum solution of some problem P' that is primal equivalent to P. Let π^* be the current node potentials.

Given an optimum dual solution of a minimum cost flow problem, its primal optimum solution can be obtained by a solving maximum flow problem. This is a well-known technique (see, e.g., Ahuja, Magnanti and Orlin [1989]) and we include it for the sake of completeness. We define the reduced cost $\bar{c}_{ij}$ of an arc $(i, j) \in A$ as $c_{ij} - \pi^*(i) + \pi^*(j)$. The optimality condition C2 implies that $\bar{c}_{ij} \geq 0$ for each arc $(i, j) \in A$. Further, at optimality, no arc $(i, j) \in A$ with $\bar{c}_{ij} > 0$ can have positive flow; for then arc (j, i) will be present in the residual network with $\bar{c}_{ji} = -\bar{c}_{ij} < 0$, violating conditions C2. 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 non-negative flow in the subnetwork $G^\circ = (N, A^\circ)$, where $A^\circ = \{(i, j) \in A : \bar{c}_{ij} = 0\}$, 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 primal optimum to P' and also to problem P.

7. Capacitated Minimum Cost Flow Problem

The algorithm described in Section 6 solves the uncapacitated minimum cost flow problem as a sequence of O(min (n log U, 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(min ((n + m') log U, (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, j) 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 Section 6 is applied to the transformed network, it solves O(min ((N + m') log U, (n + m') log n)) shortest path problems and each shortest path problem takes S((n + m'), (m + m')) time. In order to reduce the shortest path computation to O(S(n, m)), we have to solve shortest path problems more carefully. We achieve this by solving shortest path problems on a smaller network G' = (N', A').

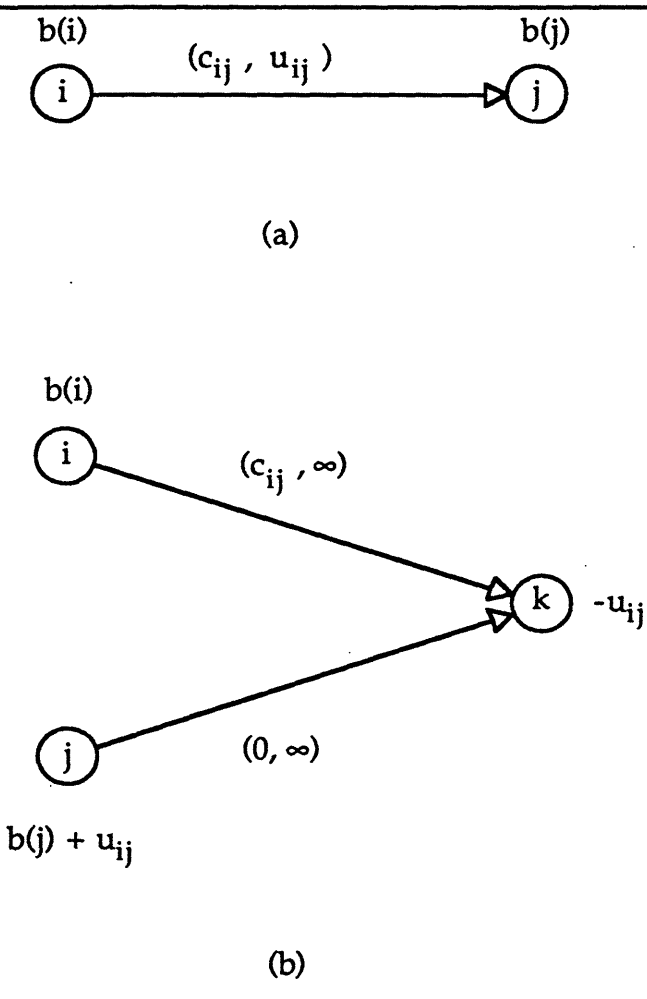


FIGURE 6. TRANSFORMING A CAPACITATED ARC INTO UNCAPACITATED ARCS.

Let G(x) denote the residual network corresponding to the current pseudo-flow x. The nodes in the residual network are either original nodes or contracted nodes. It is easily seen 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 cost $c'_{ji} = \bar{c}_{jk} - \bar{c}_{ik} = \bar{c}_{jk} \geq 0$ (because by Lemma 1, $\bar{c}_{ik} = 0$). Further, if $x_{jk} > 0$, then add the arc (i, j) with cost $c'_{ij} = \bar{c}_{ik} - \bar{c}_{jk} = \bar{c}_{ik} \geq 0$. For each other arc (i, j) in G(x) that is not replaced, we define $c'_{ij} = \bar{c}_{ij}$. Clearly, the network G' has at most n nodes, m + 2m' = O(m) and 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 [1984] implementation of Dijkstra's algorithm. Let d(i) 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 $\{d(i) + \bar{c}_{ik}\, d(j) + \bar{c}_{jk}\}$ 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) units of time.

We have thus shown 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.* ■

8. Future Directions

In this paper, we have presented a new strongly polynomial algorithm for the minimum cost flow problem. Our algorithm is superior to 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. It can be shown that the resulting computation time for the uncapacitated problem is $O((n \log n) S(n, m, C) + m \log n \log U)$. 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 nearly linearly in the size of data as U grows exponentially large. For very large values of U and moderate values of C this surpasses all other known algorithms for the minimum cost flow problem as analyzed under the logarithmic model of computation. This is also true for the capacitated network flow problem.

Parallel Computation

The most obvious way to speed up the algorithm with parallel computation is to apply parallel algorithms in 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. Faster times may be achieved using fast matrix multiplication. 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. In fact, these ideas motivated the development of the recent minimum cost flow algorithm due to Ahuja, Goldberg, Orlin, and Tarjan [1988] which also relies on the transformation of the minimum cost flow problem into the transportation 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 some node is contracted within O(log n) iterations. This argument may also be applicable to other algorithms that rely on a successive reduction in primal feasibility.

Acknowledgements

I wish to thank Eva Tardos, Leslie Hall, Serge Plotkin and Bob Tarjan for valuable comments made about this research. However, I am especially indebted to Ravi Ahuja who provided help and insight throughout this research.

This research was supported in part by the 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

Aho, A. V., J. E. Hopcraft, and J. D. Ullman. 1974. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, MA.

Ahuja, R. K., T. L. Magnanti, and J. B. Orlin. 1989. Network Flows. *Handbooks in Operations Research and Management Science*, Vol 1: *Optimization*.

Ahuja, R. K., J. B. Orlin, and R. E. Tarjan. 1987. Improved time bounds for the maximum flow problem. Sloan Working Paper 1966-87, Sloan School of Management, M.I.T., Cambridge. (To appear in *SIAM J. Comput.*)

Ahuja, R. K., K. Mehlhorn, J. B. Orlin, and R. E. Tarjan. 1988. Faster algorithms for the shortest path problem. Technical Report No. 193, Operations Research Center, M.I.T., Cambridge. (To appear in *J. ACM.*)

Ahuja, R. K., A. V. Goldberg, J. B. Orlin, and R. E. Tarjan. 1987. Finding minimum cost flows by double scaling. Working Paper No. 2047-88, Sloan School of Management, M.I.T., Cambridge.

Bertsekas, D. P. and J. Eckstein. 1988. Dual coordinate step method for linear network flow problems. *Math. Prog.* 42, 203-243.

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.

Bradley, G., G. Brown, and G. Graves. 1977. Design and implementation of large scale primal transshipment algorithms. *Man. Sci.* 21, 1-38.

Busacker, R. G., and P. J. Gowen. 1961. A procedure for determining a family of minimal-cost network flow patterns. ORO Technical Report No. 15, Operations Research Office, John Hopkins University.

Cook, S.A., and R. A. Reckhow. 1973. Time bounded random access machines. *J. of Comput. System Sci.* 7, 354-375.

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.

Ford, L. R., and D. R. Fulkerson 1962. *Flows in Networks*. Princeton University Press, Princeton.

Frank, A., and E. Tardos. 1985. An application of the simultaneous approximation in combinatorial optimization. Report No. 85374, Institut für Ökonometrie und Operations Research, Bonn.

Fredman, M. L., and R. E. Tarjan. 1987. Fibonacci heaps and their uses in network optimization algorithms. *25th IEEE Symp. on Found. of Comp. Sci.* (1984), 338-346. Also in *J. of 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.

Fulkerson, D. R. 1961. An out-of-kilter method for minimal cost flow problems. *J. of SIAM* 9, 18-27.

Gabow, H. N. 1985. Scaling algorithms for network problems. *J. of Comput. System Sci.* 31, 148-168.

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

Glover, F., D. Karney, and D. Klingman. 1974. Implementation and computational comparisons of primal, dual and primal-dual computer codes for minimum cost network flow problems. *Networks* 4, 191-212.

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.

Goldberg, A. V., and R. E. Tarjan. 1986. A new approach to the maximum flow problem. *Proc. 18th ACM Symp. on the Theory of Comp.*, 136-146.

Goldberg, A. V., E. Tardos, and R. E. Tarjan. 1989. Network flow algorithms. Report No. STAN-CS-89-1252. Dept. of Computer Science, Stanford University, Stanford, CA.

Goldberg, A. V., and R. E. Tarjan. 1988. Finding minimum-cost circulations by canceling negative cycles. *Proc. 20th ACM Symp. on the Theory of Comp.*, 388-397.

Grigoriadis, M. 1986. An efficient implementation of the network simplex method. *Math. Prog. Study* 26, 83-111.

Johnson, D. S. 1987. The NP-completness column: An ongoing guide. *J. of Algorithms* 8, 285-303.

Klein, M. 1967. A primal method for minimal cost flows with applications to the assignment and transportation problems. *Man. Sci.* 14, 205-220.

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

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

Orlin, J. B. 1984. 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.

Orlin, J. B. 1985. On the simplex algorithm for networks and generalized networks. *Math. Prog. Studies* 24, 166-178.

Orlin, J. B. 1986. A dual version of Tardos's algorithm for linear programming. *Oper. Res. Letters* 5, 221-226.

Orlin, J. B. 1988. A faster strongly polynomial minimum cost flow algorithm. In *Proc. 20th ACM Symp. on Theory of Computing*, 377-387.

Papadimitrou, C. H., and K. Steiglitz. 1982. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Englewood Cliffs, N.J.

Plotkin, S. A. and E. Tardos. 1989. Improved dual network simplex. Manuscript.

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

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

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

M.I.T. Sloan School of Management
Working Paper Series

Papers by James B. Orlin
Professor of Operations Research

Paper #	*Date*	*Title/Author(s)*
3112	1/90	"Faster Parametric Shortest Path and Minimum Balance Algorithms," Young, N., Trajan, R., and Orlin, J.
3060	8/89	"A Faster Strongly Polynomial Minimum Cost Flow Algorithm," Orlin, J.
2059	12/88	"Network Flows," Ahuja, R., Magnanti, T., and Orlin, J.
2090	11/88	"Improved Primal Simplex Algorithms for Shortest Path, Assignment and Minimum Cost Flow Problems," Ahuja, R., and Orlin, J.
2047	8/88	"Finding Minimum-Cost Flows by Double Scaling," Ahuja, R., Goldberg, A., Orlin, J., and Tarjan, R.
1966	6/88	"Improved Time Bounds for the Maximum Flow Problem," Ahuja, R., Orlin, J., and Tarjan, R.
2019	5/88	"New Scaling Algorithms for the Assignment and Minimum Cycle Mean Problems," Orlin, J., and Ahuja, R.
2043	4/88	"Faster Algorithms for the Shortest Path Problems," Ahuja, R., Mehlhorn, K., Orlin, J., and Tarjan, R.
2042	3/88	"A Faster Strongly Polynomial Minimum Cost Flow Algorithm," Orlin, J.
1905	3/88	"A Fast and Simple Algorithm for the Mazimum Flow Problem," Ahuja, R., and Orlin, J.
1908	7/87	"New Distance-Directed Algorithms for Maximum Flow and Parametric Maximum Flow Problems," Orlin, J., and Ahuja, R.
1730	10/85	"Parametric Linear Programming and Anti-Cycling Pivoting Rules," Magnanti, T., and Orlin, J.
1700	8/85	"Analysis and Solution Algorithms of Sealift Routing and Scheduling Problems: Final Report," Psaraftis, H., Orlin, J., Bienstock, D., and Thompson, P.
1679	7/85	"The Company of Dynamic/Periodic Languages and Optimization Problems," Orlin, J.
1686	6/86	"A Dual Version of Tardo's Algorithm for Linear Programming," Orlin, J.

M.I.T. Sloan School of Management
Working Paper Series

Papers by James B. Orlin

Paper #	*Date*	*Title/Author(s)*
1615	12/84	"Genuinely Polynomial Simplex and Non-Simlex Algorithms for the Minimum Cost Flow Problem," Orlin, J.
1534	2/84	"A Finitely Converging Cutting Plane Technique," Orlin, J.
1527	1/84	"On the Complexity of Four Polyhedral Set Containment Problems," Freund, R., and Orlin, J.
1484	9/83	"A Polynomical-Time Parametric Simplex Algorithm for the Minimum Cost Network Flow Problem," Orlin, J.
1467		"On the Simplex Algorithm for Networks and Generalized Networks," Orlin, J.
1449		"Some Very Easy Knapsack/Partition Problems," Orlin, J.
1446		"On a 'Primal' Matriod Intersection Algorithm," Orlin, J., and VandeVate, J.
1399	1/83	"Some Problems on Dynamic/Periodic Graphs," Orlin, J.
1383	11/82	"Consecutive Optimizors for a Partitioning Problem with Applications to Optimal Inventory Groupings for Joint Replenishment," Chakravarty, A.K., Orlin, J.B., and Rothblum, U.G.
1332	7/82	"Dynamic Matchings and Quasi-Dynamic Fractional Matchings, Part II," Orlin, J.B.
1331	7/82	"Dynamic Matchings and Quasi-Dynamic FractionalMatchings, Part I," Orlin, J.B.

Please Use Attached Form When Ordering Papers