Algorithms for the Simple Equal Flow Problem

by
Ravindra K. Ahuja
James B. Orlin
Giovanni M. Sechi
Paola Zuddas

SWP# 3960 June 1997

ALGORITHMS

FOR

THE SIMPLE EQUAL FLOW PROBLEM

Ravindra K. Ahuja*
Faculty of Management &
Rutgers Center for Operations Research (RUTCOR)
Rutgers University
New Brunswick, NJ 08903, USA

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

Giovanni M. Sechi
Dept. of Hydraulics
University of Cagliari
09123 Cagliari, Sardinia
ITALY

Paola Zuddas
Dept. of Hydraulics
University of Cagliari
09123 Cagliari, Sardinia
ITALY

(Revised May 25, 1997)

* On leave from Indian Institute of Technology, Kanpur - 208 016, INDIA.

Algorithms for The Simple Equal Flow Problem

Ravindra K. Ahuja[1] , James B. Orlin[2] , Giovanni M. Sechi[3], and Paola Zuddas[4]

ABSTRACT

In this paper, we study a variant of the minimum cost flow problem where each arc in the specified set R of arcs must carry the same amount of flow. This problem, which we call the *simple equal flow problem*, arose while modeling some real-life problems. In this paper, we describe one application of the simple equal flow problem arising in water resource system management. We consider the simple equal flow problem in a directed network with n nodes, m arcs, and where all arc capacities and node supplies are integer and bounded by U. In this paper, we develop several algorithms for the simple equal flow problem - the network simplex algorithm, the parametric simplex algorithm, the combinatorial parametric algorithm, the binary search algorithm, and the capacity scaling algorithm. The binary search algorithm solves the simple equal flow problem in O(log(nU)) applications of any minimum cost flow algorithm. The capacity scaling algorithm solves it in O(m log(nU)(m + n log n)) time. These algorithms can be easily modified to obtain an integer solution of the simple equal flow problem.

[1] Faculty of Management and Rutgers Center for Operations Research (RUTCOR), Rutgers University, New Brunswick, NJ 08903. On leave from Indian Institute of Technology, Kanpur 208 016, INDIA.

[2] Sloan School of Management, Massachusetts Institute of Technology, Cambridge, MA 02139, USA.

[3] Dept. of Hydraulics, University of Cagliari, 09123 Cagliari, Sardinia, ITALY.

[4] Dept. of Hydraulics, University of Cagliari, 09123 Cagliari, Sardinia, ITALY.

1. INTRODUCTION

In this paper, we study a variant of the minimum cost flow problem, which we call the *simple equal flow problem.* This problem is defined as follows. Let G = (N, A) be a directed network defined by a set N of n nodes and a set A of m directed arcs. Each arc $(i, j) \in A$ has an associated cost c_{ij} and an integer capacity u_{ij}. Each node $i \in N$ has an associated integer number b(i) representing its supply if $b(i) > 0$ and its demand if $b(i) < 0$. Let $R \subset A$ be a specified set of arcs. Let $S = A - R$. The simple equal flow problem is a minimum cost flow problem where each arc in R is required to carry the same amount of flow. This problem can be mathematically stated as follows:

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

subject to

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

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

$$x_{ij} = x_{kl} \text{ for every pair of arcs } (i, j) \text{ and } (k, l) \text{ in } R. \tag{1d}$$

The simple equal flow problem arose while modeling several real life problems. We describe here an example of a water resource management problem. Water is a scarce commodity in Sardinia due to scanty rainfalls. Its efficient utilization is of utmost importance to the economy of the state. It is a multiperiod (or, dynamic) problem and the planning period typically consists of five years or longer. This dynamic problem may be transformed into a static problem by using a standard technique of time-expanding the underlying network. In this technique, (see, for example, in Ahuja, Magnanti and Orlin [1993], we first discretize the time horizon into a finite number of periods (typically, each period represents a month), and replicate the underlying network for each period. We then connect relevant nodes of different time periods by additional arcs. The time-expanding techniques for water resource management have been proposed by Simeone [1974], Sechi and Zuddas [1995], and Sechi and Zuddas [1987]. The model requires that the same quantity of water be available for each period for drinking purposes. This requirement gives rise to the equal flow constraints on a set of arcs in the time-expanded network. Hence the water resource management problem is a simple equal flow problem.

For a more refined analysis of this water resource management problem, it is desirable to consider a shorter time period, which leads to larger number of replications of the underlying network, thereby increasing the size of the network substantially. Since this model is used within a decision support system to answer a variety of "what if" questions, an efficient algorithm for solving the simple equal flow problem is required. Linear programming techniques are not well-suited for solving the simple equal flow problem due to their excessive time and memory requirements. The network flow based algorithms developed in this paper allow us to solve the simple equal flow problem in a highly efficient manner for different scenarios and management policies.

Ali, Kennington and Shetty [1988] have studied a similar variant of the minimum cost flow problem, where K pairs of arcs $\{(i_k, j_k), (p_k, q_k)\}$ are specified and the decision problem is to optimize (1d) subject to (1b), (1c), and the following constraints:

$$x_{i_k j_k} = x_{p_k q_k} \quad \text{for each } k = 1, 2, \ldots, K. \qquad (2)$$

They refer to this problem as the *equal flow problem.* This problem finds applications in federal matching of funds to various projects (Beck, Lasdon and

Engquist [1983]). An integer version of the equal flow problem studied by Ali et al. (where arc flows must be integer) is NP-complete and finds applications in crew scheduling (Carraresi and Gallo [1984]), estimating driver costs for transit operations (Turnquist and Malandraki [1984]), and the two-duty period scheduling problem (Shepardson and Marsten [1980]). Ali et al. [1988] present a heuristic algorithm to solve the equal flow problem using a Lagrangian relaxation technique. This technique relaxes the equal flow constraints, yielding the minimum cost flow problem, and uses subgradient optimization technique to solve the Lagrangian dual.

The problem presented here is simpler than the equal flow problem studied by Ali et al. [1988] and, therefore, we call it the *simple equal flow problem*. The simple equal flow problem can be solved more efficiently. Indeed, it is polynomially solvable. In this paper, we pursue two different algorithmic approaches to solve the simple equal flow problem. In the first approach, we model the problem as a generalization of the minimum cost flow problem where one column has a "non-network" structure. We then develop a special-purpose primal simplex algorithm for solving it. The resulting algorithm generalizes of the well known network simplex algorithm for the minimum cost flow problem. In the second approach, we model the simple equal flow problem as a parametric minimum cost flow problem, yielding several algorithms: (i) a parametric simplex algorithm, (ii) a combinatorial parametric algorithm, (iii) a binary search algorithm, and (iv) a capacity scaling algorithm. The latter two algorithms run in polynomial time. The binary search algorithm solves the simple equal flow problem as a sequence of O(log(mU)) = O(log(nU)) minimum cost flow problem. The capacity scaling algorithm solves the simple equal flow problem in O(m log U (m + n log n)) time. Integer versions of the simple equal flow problem can also be solved in the same time.

For the sake of brevity, we shall henceforth refer to the simple equal flow problem as the equal flow problem.

2. NETWORK SIMPLEX ALGORITHM

The equal flow problem is a linear programming problem and, therefore, linear programming methods can be adapted to solve it. An adaptation of the simplex algorithm for network flow problems is often referred to as a *network simplex algorithm.* In this section, we work out the details of the network simplex algorithm for the equal flow problem.

The network simplex algorithm for the equal flow problem generalizes the network simplex algorithm for the minimum cost flow problem studied extensively in the literature (see, for example, Kennington and Helgason [1980], Grigoriadis [1986], and Ahuja, Magnanti and Orlin [1993]).

2.1 PROBLEM REFORMULATION

For the network simplex algorithm, it will be helpful to reformulate the equal flow problem. We will henceforth assume that the set S = N - R of arcs contains at least one spanning tree. There is no loss of generality in this assumption because we can add artificial arcs with large costs. Since each arc $(i, j) \in R$ carries equal flow, we may substitute all of these arcs by a single variable x_R. Substituting $x_{ij} = x_R$ for each arc $(i, j) \in R$ in (1) gives the following statement of the equal flow problem:

$$\text{Minimize} \quad \sum_{(i,\ j)\in S} c_{ij}\, x_{ij} + c_R\, x_R \tag{3a}$$

subject to

$$\sum_{\{j:(i,j)\in S\}} x_{ij} - \sum_{\{j:(j,i)\in S\}} x_{ji} + d(i)\, x_R = b(i) \text{ for all } i \in N, \tag{3b}$$

$$0 \le x_{ij} \le u_{ij} \text{ for all } (i, j) \in S, \tag{3c}$$

$$0 \le x_R \le u_R, \tag{3d}$$

where $c_R = \sum_{(i,j)\in R} c_{ij}$, $u_R = \min\{u_{ij}: (i, j) \in R\}$, and the vector d is the sum of the column vectors corresponding to the arcs in R. The equal flow problem can alternatively be expressed in the matrix notation in the following manner:

$$\text{Minimize} \quad cx + c_R x_R \tag{4a}$$

subject to

$$\mathcal{N}x + d\, x_R = b, \tag{4b}$$

$$0 \le x \le u, \tag{4c}$$

$$0 \le x_R \le u_R, \tag{4d}$$

where $\mathcal{N}$ is the node-arc incidence matrix of the network $G' = (N, S)$. Notice that d(i) represents the difference of the number of arcs in R emanating from node i and the number of arcs entering node i. This observation yields the following:

Property 1. *The vector d satisfies the following conditions:*

(a) *d(i) is integer for each node $i \in N$;*

(b) $-(n-1) \le d(i) \le (n-1)$ *for each node $i \in N$;*

(c) $\sum_{i\in N} d(i) = 0$;

(d) $-K \le \sum_{i\in H} d(i) \le K$ *for any subset H of nodes.*

We may point out that the equal flow problem is a special case of the minimum cost flow problem plus an additional variable. In general, the minimum cost flow problem plus an additional variable has the rank of the constraint matrix equal to n, but the equal flow problem has the rank of the constraint matrix equal to (n-1) (see Lemma in Section 2.2). This property leads to a more efficient algorithm for the equal flow problem compared to the (general) minimum cost flow problem with an additional variable.

2.2 STRUCTURE OF THE BASIS

A basis of the linear programming problem is a collection of r basic variables x_B whose columns in the constraint matrix are linearly independent, where r is the rank of the constraint matrix. A basis structure of a bounded variable simplex algorithm (that is, a linear programming problem with upper bounds on variables) consists of a set of basic variables x_B, a set of nonbasic variables x_L at their lower bounds, and a set of nonbasic variables x_U at their upper bounds. The following is a well known result in linear programming and characterizes the basis of a linear programming problem.

Property 2. *Let x_B represent a subset of variables for the linear programming problem $\mathcal{A} x = b$, and $\mathcal{B}$ denote the submatrix of $\mathcal{A}$ corresponding to the variables in x_B. Then the variables in x_B define a basis if and only if the system of equations $\mathcal{B}x_B = b$ is a unique solution.*

An alternative way to represent the basis of a linear programming problem is by the index set **B** of variables, and we shall henceforth adopt this notation. In this notation, we represent the basis structure by (**B**, **L**, **U**). For the network flow problem, the index set represents the sets of arcs because there is a one-to-one correspondence between flow variables (x_{ij}'s) and arcs ((i, j)'s). Consequently, the sets **B**, **L**, and **U**, will henceforth represent the sets of arcs. We shall, however, make an exception for the variable x_R because it does not represent a single arc but denotes the set R of arcs. In other words, except the variable x_R, we shall interchangeably refer to a variable x_{ij}, $(i, j) \in S$ by the arc (i, j) and vice-versa.

The network simplex algorithm for the minimum cost flow problem derives its efficiency from the fact that its basis is a spanning tree. We will show in Lemma 2 that the basis of the equal flow problem is a variant of a spanning tree. We define the concept of a two-tree. A *two-tree* is a set of two node-disjoint trees $\mathbf{T}'$ and $\mathbf{T}''$ that together span all nodes. (Alternatively, a two-tree is a spanning tree of G minus a tree arc.) We refer to a two-tree as a *good two-tree* if $d(\mathbf{T}') \neq 0$. Observe that Property 1 implies that $d(\mathbf{T}') = - d(\mathbf{T}'')$ and, therefore, $d(\mathbf{T}') \neq 0$ if and only if $d(\mathbf{T}'') \neq 0$. We are now ready to discuss the structure of the basis for the equal flow problem.

Lemma 1. *The constraint matrix of the equal flow problem has rank equal to (n-1).*

Proof. Observe that summing n constraints in (3b) yields a zero row (here we use Property 1(c)). Thus the rank of the equal flow problem is at most (n-1). Now consider any spanning tree **T** of S. Our assumption implies that there always exists such a tree. Since **T** contain (n-1) arcs which, due to the acyclicity of **T**, are linearly independent, it follows that the rank of the equal flow problem is at least (n-1). Combining these results with our previous result establishes the lemma.

Lemma 2. *A basis of the equal flow problem either (i) consists of a spanning tree of S, or (ii) consist of a good two-tree in S and* x_R.

Proof. A basis of the equal flow problem either contains x_R or it does not. In the latter case, the basis is a spanning tree of S. We now consider the former case. In this case, the basis contains x_R and an acyclic set of (n-2) arcs of S. Notice that an acyclic set of (n-2) arcs must be a two-tree. We will show later in Section 2.4 that the flow associated with a two-tree plus the equal flow arcs is unique if and only if the two-tree is a good two-tree. Using this result in conjuction with Property 2 implies that the two-tree in the basis must be a good two-tree. This completes the proof of the lemma. ♦

We give an example of the two possibilities of the basis in Figure 1. Figure 1(a) gives the underlying network where arcs in R are represented by dotted lines. Figure 1(b) shows a spanning tree basis not containing x_R, whereas Figure 1(c) shows a basis containing x_R. In case, a basis of the equal flow problem consists of a spanning tree, we denote it by **T** and if it consists of a good two-tree plus x_R then we denote it by $\mathbf{T}' \cup \mathbf{T}'' \cup \{x_R\}$.

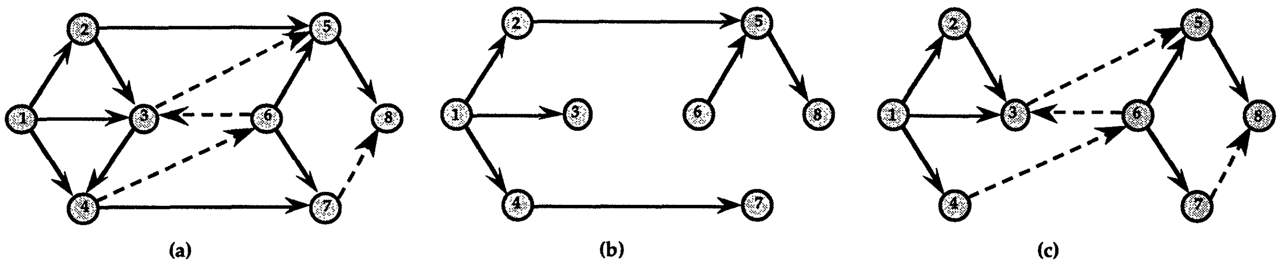

Figure 1. Illustrating the basis of the equal flow problem.

2.3 Obtaining Flow for a Basis Structure

Each basis structure of the equal flow problem has an associated flow vector x. If **B** = T, then we can obtain the flow x associated with the basis structure (**B**, **L**, **U**) using the same method as used in the network simplex algorithm for the minimum cost flow problem. In case **B** = $\mathbf{T}' \cup \mathbf{T}'' \cup \{x_R\}$, then we need a different method, which we describe next.

We first compute the flow variable x_R. The procedure compute-x_R given in Figure 2 describes a method to compute the value of x_R. The procedure uses the fact that $d(\mathbf{T}') \neq 0$, which is true because $\mathbf{T}'$ is part of a good two-tree. We may point out that if we select the tree $\mathbf{T}''$ in place of $\mathbf{T}'$ we would obtain the same value of x_R because $b(\mathbf{T}'') = -b(\mathbf{T}')$, $u(\mathbf{T}'') = -u(\mathbf{T}')$, and $d(\mathbf{T}'') = -d(\mathbf{T}')$.

```
procedure compute-$x_R$;
begin
    $b(T') := \sum_{i \in T'} b(i)$;
    $d(T') := \sum_{i \in T'} d(i)$;
    $u(T') := \sum_{\{i \in T', j \in T'', (i,j) \in U\}} u_{ij} - \sum_{\{i \in T'', j \in T', (i,j) \in U\}} u_{ij}$;
    compute $x_R := [b(T') - u(T')]/d(T')$;
end;
```

Figure 2. Procedure to compute the flow on the equal flow arcs.

Once we know the flow on the equal flow arcs, then there is a unique flow on arcs in T' and T'' that will also satisfy the mass balance constraints. The method to compute the flow on arcs in T' (or T'') is the standard method used in the network simplex algorithm. The flow x thus obtained is the basic solution associated with the basis structure (**B**, **L**, **U**). If $0 \le x \le u$, then it is a basic feasible solution; otherwise, it is not a basic feasible solution.

We illustrate our procedure on the numerical example shown in Figure 3(a). Assume that all nonbasic arcs are at their lower bounds. Then, $b(T') = 2$ and $d(T') = 1$. Hence $x_R = b(T')/d(T') = 2$. Setting flow equal to 2 gives the updated supplies/demands shown in Figure 3(b). The flows on tree arcs that satisfy these supplies/demands are shown in Figure 3(c).

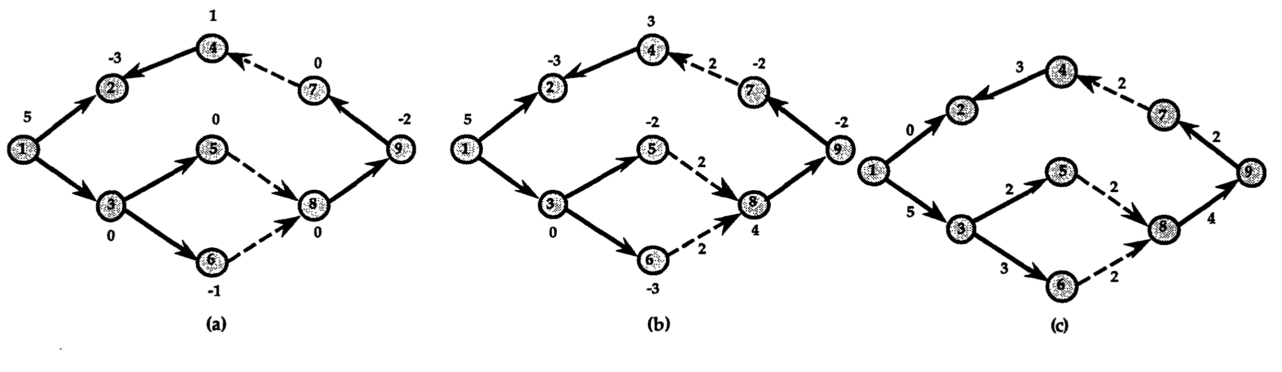

Figure 3. Illustrating the computation of flows associated with a basis structure.

In proving Lemma 2, we used the fact that the flow associated with a basis $T' \cup T'' \cup \{x_R\}$ is unique. The proof follows from our preceding discussion. The procedure compute-x_R yields a unique value of x_R satisfying the mass balance constraints across the cut. The flow on arcs in T' and T'' satisfying the remaining mass balance constraints is then uniquely determined.

The solution associated with a basis structure of the equal flow problem may be noninteger. But each arc flow will be a rational number of the form p/q, where q is no more than K (recall that $K = |R|$); we call such a number a *K-fractional number.* To see this, observe from Property 1(d) that the value of x_R computed in the procedure compute-x_R is always a K-fractional number. Since the flow on equal flow arcs is a K-fractional number, the flow on tree arcs will also be a K-fractional number.

2.4 OBTAINING NODE POTENTIALS FOR A BASIS STRUCTURE

Each basis structure (**B**, **L**, **U**) of the equal flow problem has an associated set of node potentials π, which is obtained by setting $c^{\pi}_{ij} = 0$ for each variable in the basis **B**. If **B** = **T**, then we can obtain the node potentials π using the same method as used in the network simplex algorithm for the minimum cost flow problem. However, if $\mathbf{B} = \mathbf{T}' \cup \mathbf{T}'' \cup x_R$, then this method needs to be modified as described next. Assume, as earlier, that the subtrees $\mathbf{T}'$ and $\mathbf{T}''$ have the roots r' and r".

We need to find the node potentials π so that $c^{\pi}_{ij} = 0$ for each arc $(i, j) \in \mathbf{T}' \cup \mathbf{T}''$ and $c^{\pi}_R = c_R - \sum_{i \in N} d(i)\, \pi(i) = 0$. To do so, we first determine a set of node potentials π so that $c^{\pi}_{ij} = 0$ for each arc $(i, j) \in \mathbf{T}' \cup \mathbf{T}''$. We first set $\pi(r') = 0$ and compute the node potentials of nodes in $\mathbf{T}'$ by using $c^{\pi}_{ij} = 0$ for each arc $(i, j) \in \mathbf{T}'$. Then we set $\pi(r'') = 0$ and compute the node potentials of nodes in $\mathbf{T}''$ by using $c^{\pi}_{ij} = 0$ for each arc $(i, j) \in \mathbf{T}''$. Next we compute $c^{\pi}_R = c_R - \sum_{i \in N} d(i)\, \pi(i)$. If $c^{\pi}_R = 0$, then

π is the set of node potentials associated with the basis structure (**B**, **L**, **U**). If $c^{\pi}_{R} \neq 0$, then we modify the node potentials in the following manner:

$$\pi'(i) = \begin{cases} \pi(i) \text{ for all } i \in \mathbf{T}' \\ \pi(i) + c^{\pi}_{R}/d(\mathbf{T}'') \text{ for all } i \in \mathbf{T}'' \end{cases} \tag{5}$$

Observe that since $\mathbf{T}''$ is a part of a valid two-tree, $d(\mathbf{T}'') \neq 0$, and potentials can always be modified in the manner indicated by (5). It can be easily verified that the modified node potentials π' satisfy $c^{\pi'}_{ij} = 0$ for each arc $(i, j) \in \mathbf{T}' \cup \mathbf{T}''$. We will now show that they also satisfy $c^{\pi'}_{R} = 0$. Observe that

$$c^{\pi'}_{R} = c_R - \Sigma_{i \in N}\, d(i)\, \pi'(i) = c_R - \Sigma_{i \in N}\, d(i)\, \pi(i) - \Sigma_{i \in \mathbf{T}''}\, d(i)\, c^{\pi}_{R}/d(\mathbf{T}''),$$

$$= c^{\pi}_{R} - c^{\pi}_{R} = 0.$$

We illustrate the computation of node potentials on the basis shown in Figure 4(a) (the number besides each arc gives its cost). Suppose that node 1 is the root of the tree $\mathbf{T}'$ and node 9 is the root of the tree $\mathbf{T}''$. Figure 4(b) gives the node potentials obtained by setting $\pi(1) = \pi(9) = 0$ and using $c^{\pi}_{ij} = 0$ for all tree arcs. We next compute $c^{\pi}_{R} = c_R - \Sigma_{i \in N}\, d(i)\, \pi(i) = 25$. Since $d(\mathbf{T}'') = -1$, we obtain $c^{\pi}_{R}/d(\mathbf{T}'') = -25$. Adding -25 to the potentials of nodes in $\mathbf{T}''$ gives the potentials associated with the current basis structure which in shown in Figure 4(c).

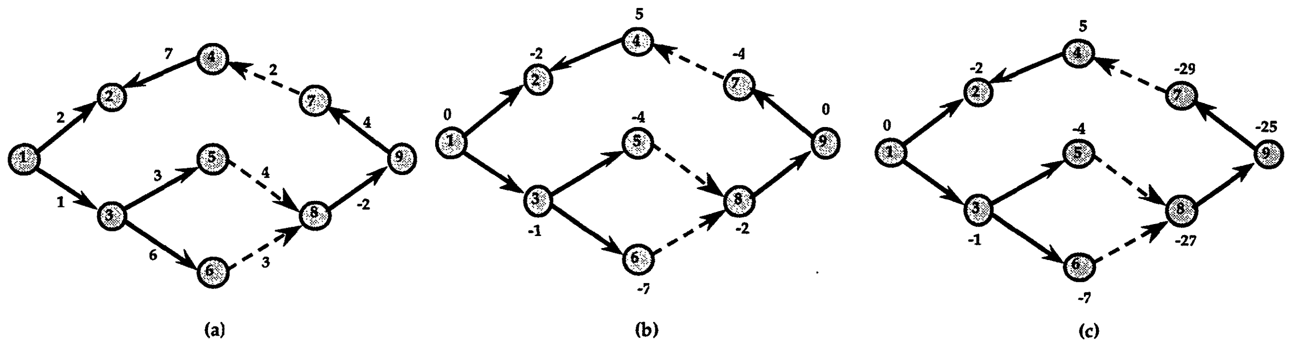

Figure 4. Illustrating the procedure compute-potentials.

2.5 Algorithmic Description

We are now in a position to describe the details of the network simplex algorithm for the equal flow problem. We give in Figure 5 an algorithmic description of the network simplex algorithm. In the subsequent subsections, we will give a detailed description of the various steps in the network simplex algorithm.

```
algorithm network-simplex;
begin
    determine an initial feasible basis structure (B, L, U);
    let x be the flow and $\pi$ be the node potentials associated
      with this basis structure;
    while some nonbasic variable violates its optimality condition do
    begin
      select an entering variable violating its optimality condition;
      add entering variable to the basis and determine the leaving variable;
      perform a pivot operation, update the basis structure, flow x, and
          the node potentials;
    end;
end;
```

Figure 5. Network simplex algorithm for the equal flow problem.

2.6 Obtaining Initial Basis Structure

In the initial basis structure, we may have a full artificial basis. To do so, we introduce an additional node s, an arc (s, j) for each node $j \in N$ with $b(j) < 0$, and arc (j, s) for each node $j \in N$ with $b(j) \geq 0$. We set the cost and capacity of these additional arcs equal to M, where M is a sufficiently large number. The initial basis **B** is a spanning tree containing all arcs leaving or entering node s. All other arcs are nonbasic arcs at their lower bounds. The variable x_R is also a nonbasic variable. To simplify the notation, we assume that the initial network G contains the artificial arcs.

2.7 Optimality Testing and Selecting Entering variables

Let (**B**, **L**, **U**) be a feasible basis structure of the equal flow problem. Suppose that $\mathbf{B} = \mathbf{T}' \cup \mathbf{T}'' \cup x_R$. In this case, we check whether the basis structure satisfies the following optimality conditions:

$c^{\pi}_{ij} \geq 0$ for each arc $(i, j) \in \mathbf{L}$, (6a)

$c^{\pi}_{ij} \leq 0$ for each arc $(i, j) \in \mathbf{U}$. (6b)

In case, $\mathbf{B} = \mathbf{T}$, then x_R is a nonbasic variable and we must test out its optimality conditions given in (7) in addition to the ones given in (6).

$c^{\pi}_{R} \geq 0$ if x_R is at its lower bound, (7a)

$c^{\pi}_{R} \leq 0$ if x_R is at its upper bound. (7b)

If the given basis structure satisfies the optimality condition, it is optimal and the algorithm terminates. Otherwise, the algorithm selects a nonbasic arc in $\mathbf{L} \cup \mathbf{U}$ violating the condition in (6) or x_R violating the condition in (7). The selected variable is added to the basis and a pivot operation is performed. Different rules for selecting entering arcs, called *pivot rules*, yield algorithms with different empirical behavior. We can use any of the pivot rules used for the network simplex algorithm for the minimum cost flow problem.

2.8 Selecting the Leaving Variable

Let (**B**, **L**, **U**) be a basis structure of the equal flow problem with the associated flow x. Suppose we have selected an entering variable. We will now describe a method to determine the leaving variable. There are four cases which need to be considered separately.

Case 1. $\mathbf{B} = \mathbf{T}$, and the entering variable is not x_R (see Figure 6(a)).

Case 2. $B = T$, and x_R is the entering variable (see Figure 6(b) for an example of this case).

Case 3. $B = T' \cup T'' \cup x_R$, and the arc corresponding to the entering variable has both the endpoints in the same tree T' or T'' (see Figure 6(c) for an example of this case).

Case 4. $B = T' \cup T'' \cup R$, and the arc corresponding to the entering variable has its endpoints in different trees (see Figure 6(d) for an example of this case).

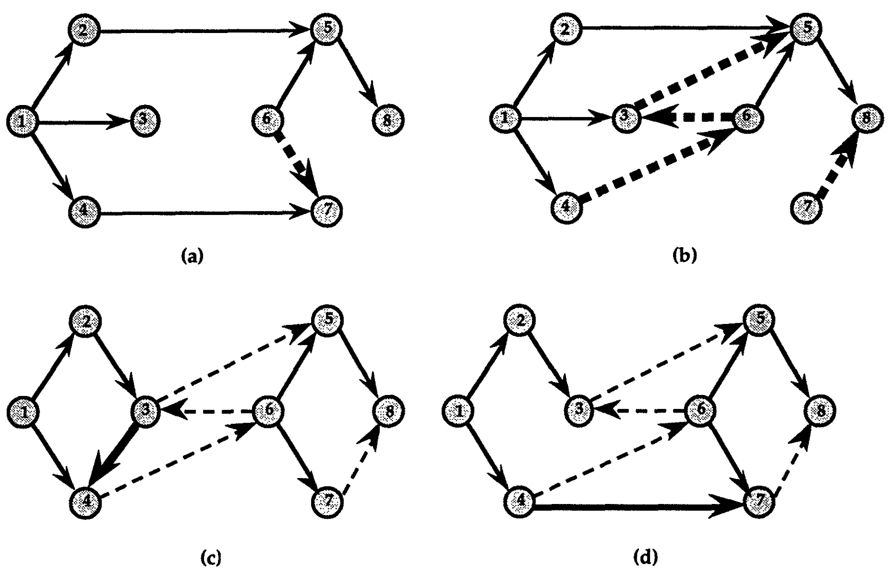

Figure 6. Different possibilities for the entering variable.

It is easy to observe that in Cases 1 and 3 methods to perform the flow changes and to determine the leaving variable are the same as in the network simplex algorithm for the minimum cost flow problem; we therefore omit discussion of these two cases. We next consider the determination of leaving arcs in the Cases 2 and 4.

We first consider Case 2, where x_R is the entering variable. If x_R enters at its lower (upper) bound, then we increase (or decrease) flow on arcs in R by one unit, and using standard techniques in network flow theory determine the change s_{ij} in the flow on any arc (i, j) $\in$ T. We next determine the maximum flow that can be increased (or, decreased) on arcs in R using the following inequalities:

$$0 \le x_{ij} + \ s_{ij} \le u_{ij} \text{ for each arc } (i, j) \in \mathbf{T} \text{ and } \ \le u_R. \qquad (8)$$

At this value of , flow on some arc in **T** or on x_R equals its lower or upper bound, which becomes a nonbasic variable. Replacing the leaving variable by the entering variable gives us a new basis structure.

We next consider Case 4 where the entering variable corresponding to the arc (k, l) has its two endpoints in two different subtrees. We assume that arc $(k, l) \in \mathbf{L}$, node k lies in $\mathbf{T}'$, and node l lies in $\mathbf{T}''$. Other cases can be handled similarly. Suppose we augment one unit of flow on arc (k, l) which takes it from node k in $\mathbf{T}'$ to node l in $\mathbf{T}''$. To satisfy the mass balance constraints, this flow must come to nodes in $\mathbf{T}'$ using the arcs in R. It is easy to see that the value of x_R will increase by $1/d(\mathbf{T}'')$ units. Next we must determine the flow change on arcs in $\mathbf{T}'$ and $\mathbf{T}''$ so that the mass balance constraints are satisfied at all the nodes; we can determine this using standard techniques in network flow theory. Let s_{ij} denote the change in the flow on an arc $(i, j) \in \mathbf{T}' \cup \mathbf{T}''$ if the flow on arc (k, l) is increased by one unit. The maximum change , that can be sent on arc (k, l), can then be determined using (8).

To summarize, we observed that the steps of the primal simplex algorithm for the equal flow problem have a close resemblance with the steps of the primal simplex algorithm for the minimum cost flow problem. The computational requirements for the primal simplex algorithm for the equal flow problem are slightly higher than for the minimum cost flow problem; but we believe that they will be higher by not more than a factor of two. Since the primal simplex algorithm for the minimum cost flow problem is known to be extremely efficient in practice, we believe that the primal simplex algorithm for the equal flow problem will be almost equally efficient.

3. PARAMETRIC ALGORITHMS

In this section, we describe the parametric algorithms for the equal flow problem. These algorithms treat x_R as a parameter instead of a variable. Recall from Section 2 that the equal flow problem is a minimum cost flow problem plus the additional variable x_R. If we treat x_R as a parameter, then the equal flow problem becomes a minimum cost flow problem. Setting x_R equal to λ gives the following minimum cost flow problem, which we refer to as $P(\lambda)$:

$$\text{Minimize } z(\lambda) = cx + c_R \lambda \tag{9a}$$

subject to

$$\mathcal{N}x = b - d\lambda, \tag{9b}$$

$$0 \leq x \leq u. \tag{9c}$$

The problem $P(\lambda)$ can be solved by any minimum cost flow algorithm. We assume without any loss of generality that $P(\lambda)$ possesses a feasible flow for every value of λ in the range $0 \leq \lambda \leq u_R$. Let $x^*(\lambda)$ denote the optimal solution $P(\lambda)$ with the objective function value as $z^*(\lambda)$. It is well known from linear programming that $z^*(\lambda)$ as a function of λ is a piecewise linear convex function. The following property is an immediate consequence of this result.

Property 3. *Let $z^*(\lambda^*) = \min\{z^*(\lambda): 0 \leq \lambda \leq u_R\}$ denote the minimum point on the curve $z^*(\lambda)$. Let $x^*_R = \lambda^*$. Then the pair $(x^*_R, x^*(\lambda^*))$ is an optimal solution of the equal flow problem.*

As an example, Figure 7 describes three possibilities for the function $z^*(\lambda)$. In Figures 7(a), (b) and (c), the function $z^*(\lambda)$ achieves the minimum at the points $\lambda = 0$, $\lambda = \lambda_m$, and $\lambda = u_\lambda$, respectively.

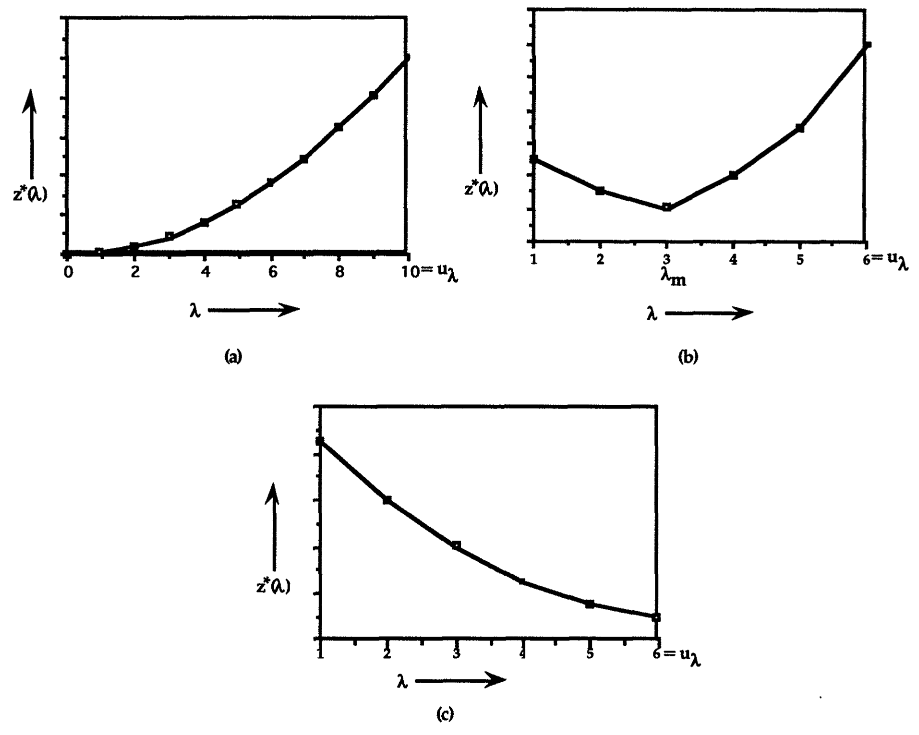

Figure 7. Three possibilities for the function z*(λ).

Property 3 implies the following general scheme to solve the equal flow problem. Identify the minimum point λ^* of the curve $z^*(\lambda)$. Set $x^*_R = \lambda^*$. Next solve the minimum cost flow problem (9) with $\lambda = \lambda^*$ and determine the optimal flow $x^*(\lambda^*)$. The pair $(x^*_R, x^*(\lambda^*))$ is an optimal solution of the equal flow problem. In the following discussion, we describe four specific implementations of this general scheme. They use different methods to identify the minimum point λ^* of the curve $z^*(\lambda)$. Whereas the parametric simplex algorithm described in Section 3.1 uses an adaptation of the parametric linear programming method to enumerate the cost curve $z^*(\lambda)$, the combinatorial parametric algorithm described in Section 3.2 solves a sequence of shortest path problems to enumerate the curve $z^*(\lambda)$. Whereas the binary search algorithm described in Section 3.3 uses binary search to locate the minimum point of the curve $z^*(\lambda)$, the capacity scaling algorithm uses the scaling of arc capacities to locate the minimum point. We next describe these algorithms in greater detail.

3.1 Parametric Simplex Algorithm

The parametric simplex algorithm, whose detailed description can be found in Srinivasan and Thompson [1972] and Ahuja, Batra and Gupta [1983], maintains an optimal basic feasible solution of the minimum cost flow problem P(λ). A basic feasible solution of the minimum cost flow problem is denoted by a basis structure (**B**, **L**, **U**), where **B** constitutes a spanning tree, **L** and **U** respectively denote the sets of nonbasic arcs at their lower and upper bounds. Let x_B, x_L, and x_U respectively denote the partition of x with respect to the sets **B**, **L**, and **U**; and $\mathcal{B}$, $\mathcal{L}$, and $\mathcal{U}$ denote the corresponding submatrices of the constraint matrix $\mathcal{N}$. The basic solution associated with (**B**, **L**, **U**) for the problem P(λ) is obtained by setting $x_L = 0$, $x_U = u_U$ and solving

$$\mathcal{B}x_B = b - d\lambda - \mathcal{U}x_U. \tag{10}$$

The parametric simplex algorithm determines an optimal basis structure of P(λ) at λ = 0 by solving a minimum cost flow problem. It next determines the largest interval $(\underline{\lambda}, \bar{\lambda})$ for the values of λ for which this basis structure continues to remain optimal; this interval is called the *characteristic interval* and the points $\underline{\lambda}$ and $\bar{\lambda}$ are called the *breakpoints*. Let (**B**, **L**, **U**) be an optimal basis structure of P(λ) for all $\lambda \in [\underline{\lambda}, \bar{\lambda}]$. Let x_B be the solution satisfying $\mathcal{B}x_B = b - \mathcal{U}u_U$ and y_B be the solution satisfying $\mathcal{B}y_B = -d$. Observe that $x^*(\lambda) = x_B + \lambda\, y_B$ is the unique basic feasible solution associated with the basis structure (**B**, **L**, **U**) for every $\lambda \in [\underline{\lambda}, \bar{\lambda}]$. The characteristic interval consists of all values of λ satisfying the following inequalities:

$$0 \le x_B + \lambda\, y_B \le u_B\,. \tag{11}$$

At $\lambda = \bar{\lambda}$, one of the basic arcs, say arc (p, q), will have flow equal to its lower bound or its upper bound. If flow equals arc's upper bound then $x_{ij} + \lambda\, y_{ij} = u_{ij}$ or λ

$\bar{} = (u_{ij} - x_{ij})/y_{ij}$, and if flow equals arc's lower bound then $\bar{\lambda} = - x_{ij}/y_{ij}$. We next perform a dual pivot operation which drops (p, q) from the basis, enters a nonbasic arc into the basis, and obtains a new basis structure.

Observe that if **b** and **u** are integer, then x_B and y_B, as described above, are also integer. Also observe that $|y_{ij}| \leq K$ for each arc $(i, j) \in$ **B**. It follows from these observations that $\bar{\lambda}$ is a K-fractional number.

We now describe how to perform the dual pivot. Dropping the arc (p, q) from **B** forms two subtrees and arcs in **A** with their endpoints in two different subtrees constitute a cut. We define the orientation of the cut along arc (p, q) if arc (p, q) is at its upper bound, and opposite to arc (p, q) otherwise. Let $\bar{Q}$ and $\underline{Q}$, respectively, denote the sets of forward and backward arcs in the cut. For each arc (i, j) in the cut, we define a number α_{ij} in the following manner: (i) $\alpha_{ij} = c^{\pi}_{ij}$ if $(i, j) \in \bar{Q} \cap L$; (ii) $\alpha_{ij} = -c^{\pi}_{ij}$ if $(i, j) \in \underline{Q} \cap U$; and (iii) $\alpha_{ij} = \infty$ otherwise. Let $\alpha_{kl} = \max\{\alpha_{ij} : (i, j) \in \bar{Q} \cap \underline{Q}\}$. We select arc (k, *l*) as the entering pivot and perform a dual pivot operation. We update the potentials π. Let (**B**', **L**', **U**') denote the updated basis structure. We call an iteration of the parametric simplex algorithm a *nondegenerate* iteration if $\underline{\lambda} < \bar{\lambda}$, and a *degenerate* iteration if $\underline{\lambda} = \bar{\lambda}$.. We next obtain the characteristic interval of (**B**', **L**', **U**'). We repeat this process until we obtain an optimal solution of the equal flow problem.

We have observed earlier that each interval point of P(λ) is a K-fractional number. It is easy to observe that two distinct K-fractional number will differ by at least $1/K^2$. This observation yields a bound of K^2u_λ on the number of interval points, or the number of non-degenerate iterations performed by the parametric simplex algorithm. However, due to the degenerate pivots, the total number of pivots performed by the algorithm may be substantially more than K^2u_λ. One can, however, use linear programming cycle prevention techniques to ensure finite convergence of the parametric simplex algorithm.

3.2 DETERMINING SLOPE OF P(λ)

Using the fact that the interval points of P(λ) are K-fractional numbers, one can develop faster algorithms for the equal flow problem. Some of these algorithms require determining the *slope* of the curve P(λ) at specified values of λ (that is, the rate of change in P(λ) as λ changes). We denote the slope of P(λ) at point λ by $z^+(\lambda)$ for increasing values of λ, and denote the slope of P(λ) at point λ by $z^-(\lambda)$ for decreasing values of λ. We describe now how we can determine $z^+(\lambda)$ (or, $z^-(\lambda)$) by solving K shortest path problems. Recall that we denote by $x^*(\lambda)$ an optimal of P(λ) and by $z^*(\lambda)$ its objective function value. Let y denotes the rate of change in optimal arc flows as λ increases by an infinitesimal amount ε. Observe that y is a solution of the following linear programming problem:

$$\text{Minimize } cy \tag{12a}$$

subject to

$$\mathcal{N}y = -d, \tag{12b}$$

$$0 \le x^*(\lambda) + y\,\varepsilon \le u. \tag{12c}$$

Now notice that since ε is an infinitesimally small quantity, any strict inequality in (12c), will always be satisfied for finite values of y. In view of this observation, solving (12) is equivalent to the following minimum cost flow problem. With respect to the flow $x^*(\lambda)$, define the network $G^*(\lambda)$ as follows: (i) if $x^*_{ij}(\lambda) = 0$, then $G^*(\lambda)$ contains the arc (i, j); (ii) if $x^*_{ij}(\lambda) = u_{ij}$, then $G^*(\lambda)$ contains the arc (j, i), and (iii) if $0 < x^*_{ij}(\lambda) < u_{ij}$, then $G^*(\lambda)$ contains both the arcs (i, j) and (j, i). Each arc in the network $G^*(\lambda)$ has infinite capacity. We set the supply/demand of each node i equal to -d(i). The optimal flow y in the network $G^*(\lambda)$ gives the optimal rate of change in arc flows as λ increases and cy gives the slope $z^+(\lambda)$ of the curve $z^*(\lambda)$. In case we need to determine the slope of $G^*(\lambda)$ as λ decreases, that is, $z^-(\lambda)$, we solve (12) with $Ny = d$.

The minimum cost flow problem in $G^*(\lambda)$ has integer supplies/demands, and the sum of the node supplies is at most K (from Property 1(d)). If we use the successive shortest path algorithm (see Ahuja, Magnanti and Orlin [1993]) to solve this minimum cost flow problem, then it will solve it in at most K applications of Dijkstra's algorithm, because each augmentation will carry integer flow from a supply node to a demand node. Using Fredman and Tarjan's [1984] implementation of Dijkstra's algorithm, the successive shortest path algorithm will take $O(K(m + n \log n))$ time. Consequently, we can determine the slope $z^+(\lambda)$ (or, $z^-(\lambda)$) by solving at most K shortest path problems in $O(K(m + n \log n))$ time. We may point out that in the optimal flow y to (12), y_{ij} is integer and its value is at most K.

A byproduct of the above discussion is that we can determine by solving O(K) shortest path problems whether a given value of λ, say λ_0, is an optimal solution of the equal flow problem. To do so, we determine $z^+(\lambda_0)$ and $z^-(\lambda_0)$. If both of these quantities are negative, then λ_0 is the optimal value of the parameter.

3.3 Combinatorial Parametric Algorithm

The parametric simplex algorithm solves the minimum cost flow problem $z^*(\lambda)$ for increasing values of λ, but due to the degeneracy it is not possible to obtain a worst-case time bound on the running time of the algorithm. We now describe an alternate algorithm that solves the equal flow problem in a pseudopolynomial time.

Consider the optimal solution $x^*(\underline{\lambda})$ of $P(\underline{\lambda})$ for some value of $\underline{\lambda}$. Let $y(\underline{\lambda})$ denote the rate of change of flow vector as λ increases and $z'(\underline{\lambda})$ denote the cost of flow of this change. Then $x^*(\lambda_-) + (\lambda - \lambda_-)y(\lambda_-)$ will be an optimal solution of $P(\lambda)$ with the objective function value $z^*(\lambda_-) + (\lambda - \lambda_-) z'(\lambda_-)$ for all values of λ for which the following inequalities remain satisfied:

$$0 \le x^*(\lambda_-) + (\lambda - \lambda_-) y(\lambda_-) \le u.$$

(13)

Let $[\lambda_-, \lambda^-]$ denote the interval for λ for which all the inequalities in (13) are satisfied. Observe that $\lambda^- > \lambda_-$, because the manner in which $y(\lambda_-)$ has been computed allows λ to be strictly increased. We next set $\lambda_- = \lambda^-$, and again determine the rate of flow change $y(\lambda_-)$. We repeat this process until we obtain an optimal solution of the equal flow problem.

Thus the above algorithm, which we call the combinatorial parametric algorithm, can enumerate the cost curve $z^*(\lambda)$ by determining the slopes of the curve $z^*(\lambda)$ at finitely many values of λ. It is easy to observe that each point λ_- where $z'(\lambda_-)$ is enumerated is a K-fractional number, because both $x^*(\lambda_-)$ and $y(\lambda_-)$ are integer and each component of $y(\lambda_-)$ is at most K. Since there are at most $O(K^2 u_\lambda)$ K-fractional solutions in the interval $[0, u_\lambda]$, the combinatorial parametric algorithm will perform at most $O(K^2 u_\lambda) = O(K^2 U)$ iterations. Since each iteration involves solving K shortest path problems and a shortest path computation requires $O(m + n \log n)$ time, the algorithm will run in $O(K^3 U (m + n \log n))$ time.

3.4 BINARY SEARCH ALGORITHM

We can use the binary search technique to obtain a polynomial-time algorithm for the equal flow problem. Figure 8 gives a description of the binary search algorithm which determines the value of λ for which the minimum cost flow problem attains the minimum value. The algorithm uses a minimum cost flow algorithm as a subroutine. The algorithm computes the optimal value of λ, say λ^*, for which $z^*(\lambda)$ is minimum by performing binary search over the initial interval $[0, u_\lambda]$, which is halved in every iteration by solving a single minimum cost flow problem. When the length of the interval is smaller than $1/K^2$, then it contains a unique K-fractional value of λ and this is the desired value. We can then solve a minimum cost flow problem to determine the optimal solution of the equal flow problem. It is easy to see that this algorithm would solve $O(K^2\ u_\lambda) = O(\log(nU))$ minimum cost flow problems.

```
algorithm binary-search;
begin
    solve $P(0)$ and compute $z^+(0)$;
    if $z^+(0) \geq 0$ then set $\lambda^* := 0$ and STOP;
    solve $P(u_\lambda)$ and compute $z^-(u_\lambda)$;
    if $z^-(u_\lambda) \geq 0$ then set $\lambda^* := 0$ and STOP;
    $\lambda_l := 0$ and $\lambda_u := u_\lambda$;
    while $(\lambda_u - \lambda_l) \geq 1/K^2$ do
    begin
        $\lambda_m := (\lambda_u + \lambda_l)/2$;
        solve $P(\lambda_m)$ and compute $z^+(\lambda_m)$ and $z^-(\lambda_m)$;
        if $z^+(\lambda_m) < 0$ then $\lambda_l := \lambda_m$
        else   if $z^-(\lambda_m) < 0$ then $\lambda_u := \lambda_m$
               else set $\lambda^* := \lambda_m$ and STOP;
    end;
    determine the unique K-fractional number
        in the range $[\lambda_u - \lambda_l]$ and set $\lambda^*$ to this number;
end;
```

Figure 8. The binary search algorithm for the equal flow problem.

We summarize the discussion in this section as the following theorem.

Theorem 1. *The binary search algorithm can solve the equal flow problem in O(m log U) applications of any minimum cost flow algorithm.*

3.5. Capacity Scaling Algorithm

We have shown in the last section that the equal flow problem can be solved in O(log(nU)) applications of the minimum cost flow algorithm. This gives the best time complexity to solve the equal flow problem for most classes of network densities. We will next show that using a scaling technique, the equal flow problem can be solved in O(m log(nU)) applications of any shortest path algorithm for nonnegative arc lengths. Since this technique uses scaling of arc capacities, we call it the *capacity scaling algorithm.* For some classes of network densities, the capacity scaling algorithm obtains the best time complexity to solve the equal flow problem.

For convenience in exposition, we shall henceforth assume that in the formulation (9), b = 0. This condition can be satisfied by converting the minimum cost flow problem with nonzero supply/demand vector into a circulation problem (which by definition has a zero supply/demand vector). This is accomplished using the following well known transformation: we (i) introduce a new node s with b(s) = 0; (ii) add an arc (s, i) for each node $i \in N$ satisfying $b(i) > 0$ with $u_{si} = b(i)$ and $c_{si} = 0$; and (iii) add an arc (i, s) for each $i \in N$ having $b(i) < 0$ with $u_{is} = -b(i)$ and $c_{is} = 0$.

The equal flow problem is to determine the value of λ for which the associated minimum cost flow problem stated in (9) attains the minimum objective function value. The capacity scaling algorithm determines this value of the parameter by solving a sequence of approximate problems for different values of the parameter , called *-scaled problems.* The -scaled problem solves the minimum cost flow problem stated in (9) subject to the following two additional constraints: (i) each arc capacity u_{ij} is replaced by $u_{ij}(\ )$ which is the greatest multiple of less than or equal to u_{ij} (that is, $u_{ij}(\ ) = \lfloor u_{ij}/\ \rfloor$; and (ii) the flow on each arc (including equal flow arcs) is also an integral multiple of . Let z (λ) denote the optimal objective function value of the minimum cost flow problem for a specific value of λ which we require to be an integral multiple of . In other words, z (λ) = Min cx, subject to

$Ax = \lambda d$, $0 \le x_{ij} \le u_{ij}($ $)$ and $x_{ij} = k$ for some nonnegative integer k. Figure 9 gives examples of the curves z for two values of the parameter   = 4 and 2. The -scaled problem is to determine the value of λ, say L , for which z (λ) attains its lowest value.

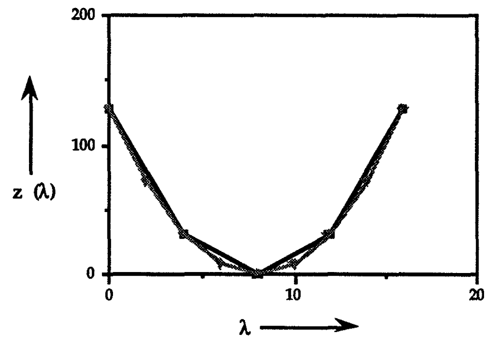

Figure 9. Illustrating the scaling technique.

The capacity scaling algorithm solves a sequence of -scaled problems. In the first scaling phase,   = $2^{\lceil \log U \rceil}$; at this value of   each arc capacity is zero and, therefore, zero flow is an optimal flow with L = 0. In the next scaling phase, the value of   is decreased by a factor of 2, and the previous flow is reoptimized to obtain an optimal solution of the modified problem. This process is repeated until   is less than $1/2K^2$.

Observe that when   is less than or equal to one, then the capacity scaling algorithm works with original arc capacities; and the approximation is only in the choice of the parameter λ which is restricted to take values which are integral multiples of . At this point, the convexity of the curve z implies that the optimal value of the parameter λ lies in the interval [L - , L +]. Consequently, each subsequent scaling phase obtains a tighter bound on the feasible values of the value λ. Finally, when   drops below $1/2K^2$, the interval [L - , L +] contains a unique K-fractional number and this is the desired value of the parameter.

To show that how we can reoptimize the solution of a scaling phase to obtain the solution of the next scaling phase, we need the following result.

Property 4. *$L \in \{L^2 - , L^2 , L^2 + \}$.*

Proof. This property implies that there exists an optimal solution of the -scaled problem which is either $L^2 -$ or L^2 or $L^2 +$. This property is easy to observe using the convexity of the functions z and z^2 ; and (ii) that the two functions have common points for all $\lambda = k(2\ )$ for all nonnegative integer k. ♦

This property implies that when the value of is halved, we need to consider only three values of the parameter λ to identify the lowest point of the curve $z\ (\lambda)$. The curves $z\ (\lambda)$ and $z^2\ (\lambda)$ coincide at the point $\lambda = L^2$; thus we need not evaluate the function value of $z\ (\lambda)$ at this point since we already know it. We will now explain how to evaluate $z\ (\lambda)$ for $\lambda = L^2 +$. To do this, we use the ideas contained in the capacity scaling algorithm for the minimum cost flow problem whose description can be found in Ahuja, Magnanti and Orlin [1983]. Let x^0 denote the optimal flow corresponding to $z\ (\lambda)$ for $\lambda = L^2$. When we increase λ by units, and try to resolve (14), then the solution x^0 continues to satisfy the optimality conditions, but might violate the mass balance constraints because the new right-hand side vector changes from $L^2\ d$ to $L^2\ d +\ d$. With respect to the modified right-hand side vector, the flow x^0 will have excesses or deficits at nodes. Property 1 implies that the node excesses/deficits will be multiples of and the total excess (or, total deficit) is bounded by m . The mass balance constraints can be satisfied again by sending flow from excess nodes to deficit nodes along shortest paths. Each such augmentation carries flow which is a multiple of , and after at most m augmentations, we restore mass balance constraints and the resulting flow is an optimal solution of (9) with $\lambda = L^2 +$. The method for evaluating $z\ (\lambda)$ for $\lambda = L^2 -$ is similar; the difference is that the new right-hand side vector changes from $L^2\ d$ to $L^2\ d -\ d$. Finally, we take the minimum of $z\ (\lambda)$ for the three values

$\lambda = L^2$ - , L^2 , and L^2 + , and L is set to the value for which the minimum is attained.

We can now determine the worst-case complexity of the capacity scaling algorithm for the equal flow problem. The previous discussion shows that we can perform a scaling phase by solving at most m shortest path augmentations. Each such augmentation can be done in O(m + n log n) time using Fredman and Tarjan's [1984] implementation of Dijkstra's algorithm. In the first scaling phase, = $2^{\lceil \log U \rceil}$, and after O(log KU) = O(m log U) scaling phases becomes less than $1/2K^2$ and the algorithm terminates. The overall running time of the algorithm is O(m(m + n log n) log U). We state this result as a theorem.

Theorem 2. *The capacity scaling obtains an optimal solution of the equal flow problem in O(m(m + n log n) log U) time.*

4. INTEGER FLOW PROBLEMS

So far in this paper we have allowed the optimal solution of the equal flow problem to be non-integral. In some situations, however, we may like to obtain an integer optimal solution of the equal flow problem. We will refer to this problem as the *integer equal flow problem.* In this section, we will describe methods to solve the integer equal flow problem.

In the integer equal flow problem, we want to determine an integer value of λ for which $P(\lambda)$ defined by (9) attains the lowest value. Suppose that λ^* denotes the optimal value of the parameter λ for which the (real-valued) equal flow problem attains the lowest value. It follows from the convexity of the curve $P(\lambda)$ that if λ^* is non-integral, then either $\lfloor \lambda^* \rfloor$ or $\lceil \lambda^* \rceil$ is an optimal value of the parameter for the equal flow problem, depending upon whether $z^*(\lfloor \lambda^* \rfloor) \leq z^*(\lceil \lambda^* \rceil)$ or vice-versa. Consequently, if we know the optimal solution of the equal flow problem, then we can determine the optimal solution of the integer equal flow problem by solving two minimum cost flow problems.

An alternate way to solve the equal flow problem would be to slightly modify the algorithms for the equal flow problem described in Sections 3 and 4, so that we directly get a solution of the integer equal flow problem without any need to solve two minimum cost flow problems. The parametric simplex algorithm and the combinatorial simplex algorithm enumerate the entire cost curve $z(\lambda)$ for increasing values of λ until the slope of the curve $z(\lambda)$ goes from non-positive to non-negative. Suppose that it happens at $\lambda = \lambda^*$. To solve the integer equal flow problem, we go a little further upto $\lceil \lambda^* \rceil$. Comparing $z^*(\lfloor \lambda^* \rfloor)$ with $z^*(\lceil \lambda^* \rceil)$ will give us the optimal solution of the integer equal flow problem.

Next consider the modifications in the binary search algorithm. While applying the binary search algorithm, we restrict attention to integer interval points only (which we can accomplish by setting $\lambda_m := \lfloor (\lambda_u + \lambda_l)/2 \rfloor$ and terminating the algorithm when the length of the search interval becomes less than one. To adapt the capacity scaling algorithm for the equal flow problem, we apply it until $\leq 1/2$, at which point the interval $[L - , L +]$ contains a unique integer which must the optimal value of the parameter λ. Finally, let us consider the modifications needed for the primal simplex algorithm for the equal flow problem. We first solve the real-valued equal flow problem by the primal simplex algorithm. Let λ^* denotes the optimal flow on the equal flow arcs. We can then perform the sensitivity analysis on λ to decrease it first to $\lfloor \lambda^* \rfloor$ and then increase it to $\lceil \lambda^* \rceil$, and choose the solution with the smaller objective function value.

5. PROPORTIONATE FLOW PROBLEMS

We have so far assumed that the flow on every arc in the set S must be the same. In some situations, however, we may allow the flow on arcs not to be exactly equal but proportionate to one-another. For example, we may specify that $x_{ij} = \alpha_{ij}\lambda$ for each arc $(i, j) \in S$, where α_{ij} is a prespecified constant for every arc and λ is a decision variable. We refer to this problem as the *proportionate flow problem.* All of our algorithms for the equal flow problem can be easily modified to solve the proportionate flow problem if α_{ij} are constants. Let $\beta = \Sigma_{(i, j)\in R}\ \alpha_{ij}$. We state without proof that there exists an optimal solution of the proportionate flow problem which is β-fractional. Using this result, it can be shown that the binary search algorithm can solve the proportionate flow problem by solving $O(\log(\beta U))$ minimum cost flow problems, and the capacity scaling algorithm can solve the proportionate flow problem by solving $O(m \log(\beta U))$ shortest path problems.

ACKNOWLEDGEMENTS

This research was partially supported by the Office of Naval Research under Contract No. N00014-96-1-0051, as well as a grant from the United Parcel Service.

REFERENCES

Ahuja, R. K., J. L. Batra, and S. K. Gupta. 1983. The parametric network feasibility problem. *Cahiers du Centre d'Etudes de Recherche Operationnelle* **25,** 13-22.

Ahuja, R. K., T. L. Magnanti, and J. B. Orlin, 1993. *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall, Inc., NJ.

Ali, A. I., J. L. Kennington, and B. Shetti. 1988. The equal flow problem. *European Journal of Operational Research* **36,** 107-115.

Beck, P., L. Lasdon, and M. Engquist. 1983. A reduced gradient algorithm for nonlinear network problems. *ACM Transactions on Mathematical Software* **9,** 57-70.

Carraresi, P., and G. Gallo. 1984. Network models for vehicle and crew scheduling. *European Journal of Operational Research* **16,** 139-151.

Fredman, M. L., and R. E. Tarjan. 1984. Fibonacci heaps and their uses in improved network optimization algorithms, *Proceedings of the 25th Annual Symposium on Foundations of Computer Science,* pp. 338-346. Full paper in *Journal of ACM* **34**(1987) 596-615.

Grigoriadis, M. D. 1986. An efficient implementation of the network simplex method. *Mathematical Programming Study* **26,** 83-111.

Kennington, J. L., and R. V. Helgason. 1980. *Algorithms for Network Programming.* Wiley-Interscience, New York.

Sechi, G. M., and P. Zuddas. 1987. Data management for extended multi-period analysis of water resource systems. Presented at *11th International Federation of Operations Research Societies Conference (IFORS'87),* Buenos Aires, Argentina.

Sechi, G. M., and P. Zuddas. 1995. A large scale water resources network optimization algorithm. *Proceedings of the International Conference on Optimization ICOTA'95,* Chengdu, China.

Shepardson, F., and R. Marsten. 1980. A Lagrangian relaxation algorithm for the two-duty period scheduling problem. *Management Science* **26,** 2274-2281.

Simeone, B. 1974. A network flow model for water resources management. *Annual AIRO Meeting,* Rome, Italy.

Srinivason, V., and G. L. Thompson. 1972. An operator theory of parametric programming for the transportation problem. *Naval Research Logistics Quarterly* **19,** 205-252.

Turnquist, M., and C. Malandraki. 1984. Estimating driver cost for transit operations planning. *Joint National Meeting of ORSA/TIMS,* Dallas.