6 MINIMUM COST NETWORK FLOW ALGORITHMS

Jeffery L. Kennington[1] and Richard V. Helgason[1]

[1]Southern Methodist University
Dallas, Texas, 75275, USA
jlk@engr.smu.edu
helgason@engr.smu.edu

Abstract: The minimal cost network flow model is defined along with optimality criteria and three efficient procedures for obtaining an optimal solution. Primal and dual network simplex methods are specializations of well-known algorithms for linear programs. The primal procedure maintains primal feasibility at each iteration and seeks to simultaneously achieve dual feasibility, The dual procedure maintains dual feasibility and moves toward primal feasibility. All operations for both algorithms can be performed on a graphical structure called a tree. The scaling push-relabel method is designed exclusively for optimization problems on a network. Neither primal nor dual feasibility is achieved until the final iteration.

Keywords: Networks-graphs flow algorithms, integer programming algorithms, linear programming algorithms, linear programming simplex algorithms.

6.1 INTRODUCTION

The special structure found in the minimum cost network flow problem has been exploited in the design of highly efficient solution techniques. In the early 1950s it was known that this special structure permits radical simplifications of the simplex method (see Dantzig (1951)). Methods for solving this problem are among the most efficient known for solving large-scale optimization problems. The mathematical foundations for these highly successful procedures can be found in the following classic books that appeared in the 1960s: Ford and Fulkerson (1962), Dantzig (1963), and Charnes and Cooper (1967). Much of the history of this problem along with some thirteen distinct algorithms may be found in the award-winning manuscript Ahuja et al. (1993). Of the many algorithms found in the literature, the most computationally efficient appear to be the following: primal network simplex, dual network simplex, and scaling push-relabel. A pseudo-code for each of these algorithms is given in this presentation.

An important aspect of real-world network management is the rapid restoration of service in the event of fiber cuts and equipment failures. Our colleagues at MCI in Dallas, Texas have developed complex models for restoration that require the use of efficient solution software. Their first real-time restoration system involved solving a series of minimum cost network flow problems, each of which determined a least-cost restoration path for a prioritized list of customers. This particular application used a software implementation of the primal network simplex algorithm, as described in this presentation.

6.1.1 Notation

A *network* is a directed graph $[V,E]$ with node set $V = \{1,\ldots,\bar{v}\}$ and arc set $E = \{1,\ldots,\bar{e}\}$ of ordered pairs of nodes. We adopt the mild notational abuse of referring to an arc k of E only by its endpoints, e.g. $(i,j) \in E$. Ambiguity results when there are multiple arcs from node i to node j. Alternatively, we may refer to i as the *tail* and to j as the *head* of arc $k \in E$. An arc is said to be *incident* to its head and tail nodes and vice-versa. A arc is said to be *directed* from its tail node to its head node, corresponding to some mechanism being modeled which permits flow in that direction only. Arcs (i,j) and (j,i) are said to be *oppositely directed.* For a given $v \in V$, the sets $F^*(v) = \{(v,j) \in E\}$ and $B^*(v) = \{(i,v) \in E\}$ are known as the *forward star* and *backward star*, respectively, of node v.

A finite odd length sequence $P = \{v_1, e_1, v_2, e_2, \ldots, v_p, e_p, v_{p+1}\}$, whose odd elements are nodes of V and whose even elements are arcs of E, is called a *walk* of *length* p in $[V,E]$ if $e_k \in \{(v_k, v_{k+1}), (v_{k+1}, v_k)\}$. If the nodes of P are distinct with the exception that $v_1 = v_{p+1}$ is allowed, then P is said to be a *path* from v_1 to v_{p+1} and those nodes are said to be *linked* by P. A graph is said to be *connected* if any two of its nodes can be linked. Note that in a path of length $p > 2$ the arcs will be distinct. If P is a path of length $p \geq 2$ with $v_1 = v_{p+1}$ and the arcs of P are distinct, then P is called a *cycle.* The requirement that the arcs of a cycle be distinct is necessary to prevent an exceptional case from being a cycle when $p = 2$ (see example path $\bar{C}$ below). Given path or cycle P, the arc (v_k, v_{k+1}) in P is said to be traversed in the *normal direction* while the arc (v_{k+1}, v_k) in P is said to be traversed in the *reverse direction.* A graph from which no cycles can be formed is said to be *acyclic.* A connected acyclic graph is called a *tree.* A node in a tree having a single incident arc is known as a *leaf node.* A tree with at least one arc will have at least two leaf nodes. It is important to note that there is a unique path linking every pair of nodes in a tree. For the network illustrated in Figure 6.1, $V = \{1,2,3,4\}$, $E = \{(1,2),(1,3),(2,3),(2,4),(3,4)\}$, $F^*(2) = \{(2,3),(2,4)\}$, $B^*(2) = \{(1,2)\}$, $W = \{1,(1,3),3,(3,4),4,(3,4),3,(2,3),2\}$ is a walk of length 4 but not a path, $P = \{1,(1,3),3,(2,3),2,(2,4),4\}$ is a path from 1 to 4 with arcs $(1,3)$ and $(2,4)$ traversed in the normal direction and arc $(2,3)$ traversed in the reverse direction, $C = \{1,(1,3),3,(2,3),2,(1,2),1\}$ is a cycle, and $\bar{C} = \{1,(1,3),3,(1,3),1\}$ is a path but not a cycle.

A graph $[\bar{V},\bar{E}]$ is said to be a *subgraph* of $[V,E]$ if $\bar{V} \subseteq V$ and $\bar{E} \subseteq E$ and when $\bar{V} = V$, the subgraph is said to *span* $[V,E]$ or to be a *spanning subgraph* of $[V,E]$. A *spanning tree* of $[V,E]$ is a tree that spans $[V,E]$. Some spanning trees for the example network in Figure 6.1 are illustrated in Figure 6.2.

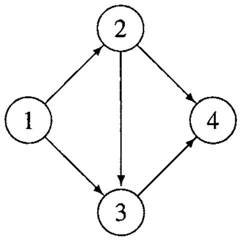

Figure 6.1 Example Network

6.1.2 *The Problem*

In network $[V,E]$, let $c(k)$ and $u(k)$ denote the *unit cost* and *arc capacity*, respectively, for arc $k \in E$ with corresponding $\bar{e}$-vectors c and u. In this presentation it is assumed that $0 \leq u(k) < \infty$. Let $r(v)$ denote the *requirement* for node $v \in V$ with corresponding $\bar{v}$-vector r. If $r(v) > 0$, then node v is said to be a *supply node* with supply $r(v)$. If $r(v) < 0$, then node v is said to be a *demand node* with demand $|r(v)|$. Note that some authors reverse this sign convention. If $r(v) = 0$, then node v is said to be a *transshipment node*.

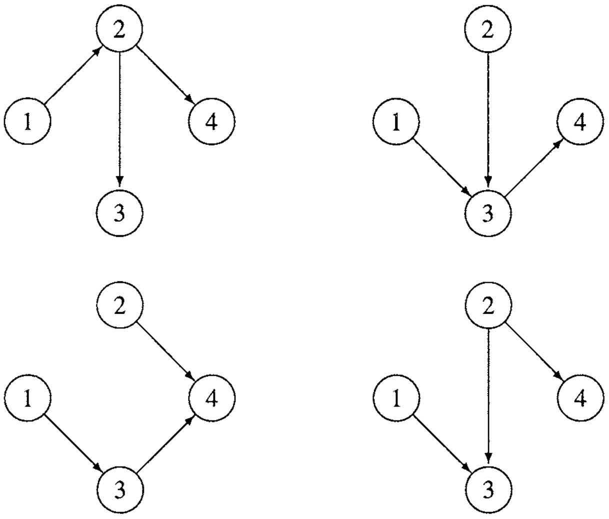

Figure 6.2 Spanning Trees

Let $x(k)$ denote the flow on arc $k \in E$ with corresponding $\bar{e}$-vector or *flow vector* x. Given problem data $D = (V,E,c,u,r)$, the set of *feasible flows* is $X = X^{fc} \cap X^{sb}$, where

$$X^{fc} = \left\{ x : \sum_{k \in F^*(v)} x(k) - \sum_{k \in B^*(v)} x(k) = r(v), \forall v \in V \right\}$$

are the *flow conservation equations* and

$$X^{sb} = \{x : 0 \le x(k) \le u(k), \forall k \in E\}$$

are the *simple flow bounds*. Note that for each node $v \in V$, there is a flow conservation equation specifying that the difference of the total flow out of v and the total flow into v to be the requirement at v. Given D, the *minimal cost network flow problem* is to find a flow vector $\bar{x}$ such that

$$c\bar{x} = \min\{cx : x \in X\}.$$

Defining the $\bar{v} \times \bar{e}$ *node-arc incidence matrix* A associated with $[V, E]$ by

$$A_{n,k} = \begin{cases} +1, & \text{if the tail of arc k is node n} \\ -1, & \text{if the head of arc k is node n} \\ 0, & \text{otherwise}, \end{cases}$$

allows us to express this problem concisely as

$$\begin{array}{ll} \text{minimize} & cx \\ \text{subject to} & Ax = b \\ & 0 \le x \le u \end{array}$$

For the network illustrated in Figure 6.1 the node-arc incidence matrix is

$$A = \begin{bmatrix} 1 & 1 & & & \\ -1 & & 1 & & \\ & -1 & & & 1 \\ & & -1 & -1 & \end{bmatrix}$$

and the flow conservation equations are

$$\begin{array}{rcrcrcrcrcl} x(1,2) & + & x(1,3) & & & & & & & = & r(1) \\ -\,x(1,2) & & & + & x(2,3) & + & x(2,4) & & & = & r(2) \\ & - & x(1,3) & - & x(2,3) & & & + & x(3,4) & = & r(3) \\ & & & & & - & x(2,4) & - & x(3,4) & = & r(4) \end{array}$$

Let $\bar{1}$ denote a $\bar{v}$-component row vector with all entries 1. Note that $\bar{1}A = 0$ since the only two nonzero entries in each column of A are $+1$ and -1. Hence, $\bar{1}A = \bar{1}r = \sum_{v \in V} r(v) = 0$, which implies that $X \neq \phi$ only if total supply equals total demand.

6.1.3 Specializations of the Network Flow Problem

The assignment problem, the semi-assignment problem, and the transportation problem are all special cases of the minimum cost network flow problem. The shortest path problem between a pair of nodes, say s and t, can be modeled by setting $r(s) = 1$, $r(t) = -1$, and $r(v) = 0, \forall v \in V \setminus \{s,t\}$. The cost $c(k)$ for $k \in E$ is the length of arc k and $u(k) = 1, \forall k \in E$. The maximal continuous flow problem between node 1 and node 4 in Figure 6.1 is modeled as illustrated in Figure 6.3, where $c(4,1) = -1$ and all other costs are zero. Physical links which permit an arbitrary direction of flow are referred to as *undirected links*. Models which use undirected links can be accommodated in the network structure presented here by replacing each undirected link with a pair of oppositely directed arcs.

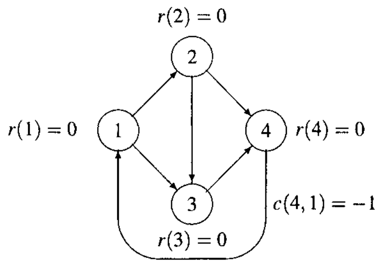

Figure 6.3 Maximum Flow From Nodes 1 to 4

6.1.4 Duality Theory and Optimality Conditions

The dual of the primal problem $\min\{cx : Ax = r, 0 \leq x \leq u\}$ is $\max\{r\pi - u\mu : \pi A - \mu \leq c, \mu \geq 0\}$. If $\bar{x} \in \{x : Ax = r, 0 \leq x \leq u\}$ and $(\bar{\pi},\bar{\mu}) \in \{(\pi,\mu) : \pi A - \mu \leq c, \mu \geq 0\}$, then $c\bar{x} \geq r\bar{\pi} - u\bar{\mu}$. If x^* is optimal for the primal and (π^*,μ^*) is optimal for the dual, then $cx^* = r\pi^* - u\mu^*$. For any $\bar{v}$-component $\bar{\pi}$ and any arc $k = (i,j)$, $\bar{c}(k) = c(k) - \bar{\pi}(i) + \bar{\pi}(j)$ is called the *reduced cost* for arc k. If $(\bar{x},\bar{\pi})$ satisfy the following conditions:

$$A\bar{x} = r \tag{6.1a}$$

$$\text{if } \bar{c}(k) = 0 \text{ , then } 0 \leq \bar{x}(k) \leq u(k) \tag{6.1b}$$

$$\text{if } \bar{c}(k) > 0 \text{ , then } \bar{x}(k) = 0 \tag{6.1c}$$

$$\text{if } \bar{c}(k) < 0 \text{ , then } \bar{x}(k) = u(k) \tag{6.1d}$$

then $\bar{x}$ is an optimum for the primal problem. Proofs of these results may be found in any standard textbook on linear programming.

6.1.5 Basic Solutions

Members of X of particular interest can be produced by partitioning E into three mutually disjoint sets (B,L,U) with $[V,B]$ a spanning tree for $[V,E]$. (Thus $B \cup L \cup U = E$

and $B \cap L = B \cap U = L \cap U = \phi$.) This induces partitionings of the flow vector, the arc capacity vector, and the flow conservation equations as $A^B x^B + A^L x^L + A^U x^U = r$. Let $\bar{x}(k) = 0, \forall k \in L$ and $\bar{x}(k) = u(k), \forall k \in U$. Since $[V,B]$ is a tree, it can easily be shown that $\bar{v} - 1$ is the rank of both A^B and the augmented matrix $\left[A^B | r - A^U \bar{x}^U\right]$, so that $A^B x^B = r - A^U \bar{x}^U$ can be solved uniquely. Let $\bar{x}^B$ be that unique solution. If $0 \leq \bar{x}^B \leq u^B$, then $0 \leq \bar{x} \leq u$ and $\bar{x}$ is called the *basic feasible solution* relative to the partitioning (B,L,U) with A^B often referred to as the *basis* (even though it is rank-deficient).

Since $[V,B]$ is a tree, the unique solution to $A^B x^B = r - A^U \bar{x}^U$ can be constructed one component at a time by the following simple method.

```
procedure $Flows\text{-}On\text{-}A\text{-}Tree(V,B,b,\bar{x})$
/* Flows On A Tree solves $Nx = b$. */
/* $[V,B]$ is a tree with node-arc incidence matrix $N$. */
/* $b$ is the right-hand side. */
/* $\bar{x}$ is the solution. */
begin
    $[\bar{V},\bar{B}] \leftarrow [V,B]$ and $\bar{b} \leftarrow b$;
    while $\bar{V} \neq \phi$ do
        $\exists$ a leaf node $n$ of $[\bar{V},\bar{B}]$;
        if $\exists (n,v) \in \bar{B}$ then
            $\bar{B} \leftarrow \bar{B} \setminus \{(n,v)\}$; $\bar{V} \leftarrow \bar{V} \setminus \{n\}$;
            $\bar{x}(n,v) \leftarrow \bar{b}(n)$; $\bar{b}(v) \leftarrow \bar{b}(v) + \bar{b}(n)$;
        else
            select $(v,n) \in \bar{B}$;
            $\bar{B} \leftarrow \bar{B} \setminus \{(v,n)\}$; $\bar{V} \leftarrow \bar{V} \setminus \{n\}$;
            $\bar{x}(v,n) \leftarrow -\bar{b}(n)$; $\bar{b}(v) \leftarrow \bar{b}(v) - \bar{b}(n)$;
        end
    end
end
```

Algorithm 6.1: Flows on a tree

For the tree illustrated in Figure 6.4, $\bar{x}(1,6) = 2$, $\bar{x}(3,6) = -3$, $\bar{x}(4,2) = -3$, $\bar{x}(4,3) = 2$, $\bar{x}(5,3) = -6$, and $\bar{x}(7,6) = 1$.

It is also relatively easy to create a set of dual variables $\bar{\pi}$ such that the reduced costs are zero for all $k \in B$, i.e. $\bar{c}(k) = c(k) - \bar{\pi}(i) + \bar{\pi}(j) = 0$ for all $k = (i,j) \in B$. Note that $-\bar{\pi}(i) + \bar{\pi}(j) = -(\bar{\pi}(i) + \alpha) + (\bar{\pi}(j) + \alpha)$, which implies that adding any α to every component of a set of dual variables $\bar{\pi}$ yielding reduced costs of zero will produce another set of dual variables yielding reduced costs of zero. Since $[V,B]$ is a tree, a set of dual variables yielding zero reduced costs can be constructed by the following simple method.

For the tree illustrated in Figure 6.4, $\bar{\pi}(1) = -10$, $\bar{\pi}(2) = -4$, $\bar{\pi}(3) = 0$, $\bar{\pi}(4) = 2$, $\bar{\pi}(5) = -1$, $\bar{\pi}(6) = -3$, and $\bar{\pi}(7) = -3$.

6.2 THE PRIMAL NETWORK SIMPLEX ALGORITHM

This algorithm is a specialization of the primal simplex method for linear programming that exploits the underlying network structure. It constructs a series of basic feasible solutions $\bar{x}$ and associated partitionings (B,L,U) such that $A\bar{x} = r, 0 \leq \bar{x} \leq u$.

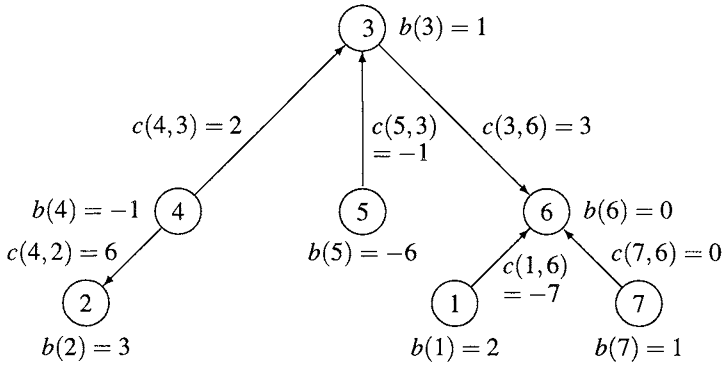

Figure 6.4 Example Tree

```
procedure Duals-On-A-Tree $(V, B, c, \bar{\pi})$
/* produces dual variables $\bar{\pi}$ */
/* $[V, B]$ is a tree */
/* $c$ is the unit cost vector */
/* $\bar{\pi}$ satisfies $\bar{c}(k) = c(k) - \bar{\pi}(i) + \bar{\pi}(j) = 0 \, \forall k = (i,j) \in B$ */
begin
    select a node $v \in V$; $\bar{V} \leftarrow V \setminus \{v\}$; $\bar{\pi}(v) \leftarrow 0$;
    /* a arbitrary node has its dual set to 0 above */
    /* all other duals are set recursively as $\bar{V}$ shrinks */
    while $\bar{V} \neq \phi$ do
        if $\exists (n, v) \in B$ with $v \in \bar{V}$ and $n \in V \setminus \bar{V}$ then
            $\bar{\pi}(v) \leftarrow \bar{\pi}(n) - c(n,v)$; $\bar{V} \leftarrow \bar{V} \setminus \{v\}$;
        end
        if $\exists (v, n) \in B$ with $v \in \bar{V}$ and $n \in V \setminus \bar{V}$ then
            $\bar{\pi}(v) \leftarrow \bar{\pi}(n) + c(n,v)$; $\bar{V} \leftarrow \bar{V} \setminus \{v\}$;
        end
    end
end
```

Algorithm 6.2: Duals on a tree

As the basic feasible solutions progress, the dual variables $\bar{\pi}$ are updated to maintain reduced costs of zero for all $k \in B$. At optimality the basic feasible solution will also satisfy $\bar{c}(k) \geq 0 \, \forall k \in L$ and $\bar{c}(k) \leq 0 \, \forall k \in U$. The corresponding specialized procedure follows.

One simple strategy for obtaining the initial basic feasible solution is to append one additional node w and $\bar{v}$ additional arcs to the network. Before enlarging the network all original arcs are placed in L with zero flows and U is empty. Then as additional arcs are appended they are placed in B. For each $v \in V$ with $r(v) \geq 0$, append the arc $z = (v, w)$ with $c(z) = 0$, $u(z) = r(v)$, and $\bar{x}(z) = r(v)$. For each $v \in V$ with $r(v) < 0$, append the arc $z = (w, v)$ with $c(z) = \infty$ and $u(z) = \bar{x}(z) = -r(v)$. This strategy is illustrated in Figure 6.5. The procedure *Duals On A Tree* can be used to determine $\bar{\pi}$ at each iteration (see /* Dual Calculation */ in procedure *Primal Network Simplex*), although it is possible to update only a portion of $\bar{\pi}$ whenever B changes.

```
procedure Primal-Network-Simplex($V, E, c, u, r$)
/* returns $\bar{x} \in X$ such that $\bar{x} \in \arg\min\{cx : x \in X\}$ */
/* $[V, E]$ is a network */
/* $c$ is the unit cost vector */
/* $u$ is the arc capacity vector */
/* $r$ is the requirements vector */
/* assumption: $X = \{x : Ax = r, 0 \leq x \leq u\} \neq \phi$ */
/* assumption: $u(k) < \infty \, \forall k \in E$ */
begin
    Obtain $(B, L, U)$ with $\bar{x}$ the corresponding basic feasible solution;
    repeat
        /* Dual Calculation */
        Obtain a set of duals $\bar{\pi}$ for which $\bar{c}(k) = 0 \, \forall k \in B$;
        /* Pricing */
        $N^- \leftarrow \{k \in L : \bar{c}(k) < 0\}$ ; $N^+ \leftarrow \{k \in U : \bar{c}(k) > 0\}$;
        if $N^- \cup N^+ = \phi$ then
            return $\bar{x}$
        else
            select $k = (t, h) \in N^- \cup N^+$;
        end
        if $k \in L$ then
            $(o, d) \leftarrow (h, t)$;
        else
            $(o, d) \leftarrow (t, h)$;
        end
        /* Column Update */
        $P \leftarrow$ the path in $[V, B]$ from $o$ to $d$; $Q \leftarrow \{k \in B : k \in P\}$;
        $Q^+ \leftarrow \{k \in Q : k$ is traversed in the normal direction in $P\}$;
        $Q^- \leftarrow Q \setminus Q^+$;
        /* Ratio Test */
        $\Delta^+ \leftarrow \min\{u(a) - \bar{x}(a) : a \in Q^+\}$; $\Delta^- \leftarrow \min\{\bar{x}(a) : a \in Q^-\}$;
        $\Delta \leftarrow \min\{u(k), \Delta^+, \Delta^-\}$;
        /* Flow Update */
        $\bar{x}(a) \leftarrow \bar{x}(a) + \Delta \forall a \in Q^+$; $\bar{x}(a) \leftarrow \bar{x}(a) - \Delta \forall a \in Q^-$;
        if $k \in L$ then
            $\bar{x}(k) \leftarrow \Delta$;
        else
            $\bar{x}(k) \leftarrow u(k) - \Delta$;
        end
        /* Basis Exchange */
        $L \leftarrow L \setminus \{k\}$; $U \leftarrow U \setminus \{k\}$;
        if $\Delta = u(k)$ then
            if $k \in L$ then
                $U \leftarrow U \cup \{k\}$;
            else
                $L \leftarrow L \cup \{k\}$;
            end
        else
            $R \leftarrow \{a \in Q^- : \bar{x}(a) = 0\} \cup \{a \in Q^+ : \bar{x}(a) = u(a)\}$;
            select $r \in R$ ; $B \leftarrow (B \setminus \{r\}) \cup \{k\}$;
            if $r \in Q^-$ then
                $L \leftarrow L \cup \{r\}$;
            else
                $U \leftarrow U \cup \{r\}$;
            end
        end
    until exited;
end
```

Algorithm 6.3: Primal network simplex method

Efficient software implementations of the primal simplex algorithm use special heuristics for the pricing operation and special data structures for maintaining and traversing the tree $[V,B]$. The data associated with the arcs is often indexed using a structure associated with the forward or backward stars for the nodes. Details may be found in Glover et al. (1974a), Glover et al. (1974b), Glover et al. (1974c), Bradley et al. (1977), Barr et al. (1979), Kennington and Helgason (1980), and Grigoriadis (1986).

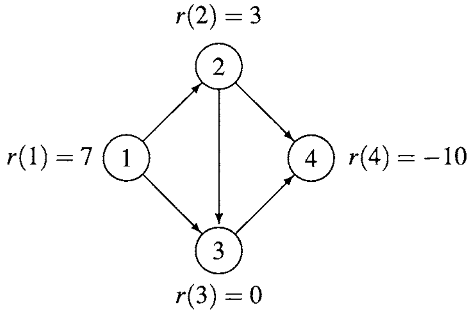

Figure 6.5a Example With 2 Source Nodes And 1 Demand Node

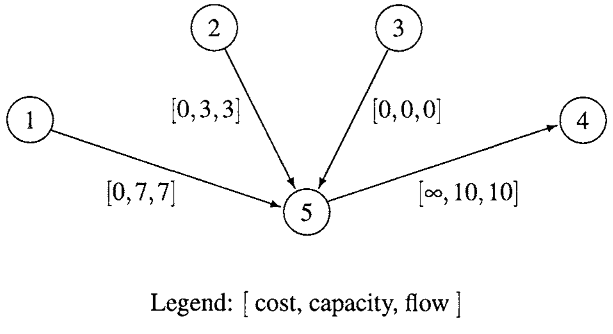

Figure 6.5b Initial Basic Feasible Solution

Figure 6.5 The Starting Solution

6.3 THE DUAL NETWORK SIMPLEX ALGORITHM

This algorithm is a specialization of the dual simplex method for linear programming that exploits the underlying network structure. It constructs a series of basic solutions $\bar{x}$ and associated partitionings (B,L,U) such that $A\bar{x}=r$ and $\bar{c}(k)=0\,\forall k\in B, \bar{c}(k)>0\,\forall k\in L$, and $\bar{c}(k)<0\,\forall k\in U$. Any basic solution that satisfies these conditions is said to be *dual feasible.* At optimality $0 \leq \bar{x}(k) \leq u(k) \, \forall k \in B$. That is, primal feasibility is only attained at the final step. The corresponding specialized procedure follows.

Under the assumption that all arc capacities are finite an initial dual feasible solution can easily be constructed. If this assumption does not hold, it is still possible to obtain such a solution using the two-phase approach described in Kennington and Mohamed (1997). The starting procedure (under finite arc capacities) follows.

6.4 THE SCALING PUSH-RELABEL ALGORITHM

The scaling push-relabel method extends the ideas of cost-scaling developed by Röck (1980) and Bland and Jensen (1992). The strategy is based on the concept of ε-optimality which can be found in Tardos (1985) and Bertsekas (1991). The presentation here follows that of Goldberg (1997) and Ahuja et al. (1993).

Let $\varepsilon \geq 0$ be a given scalar. If $(\bar{x}, \bar{\pi})$ satisfy the conditions:

$$A\bar{x} = r \tag{6.2a}$$
$$\text{if } |\bar{c}(k)| \leq \varepsilon, \text{ then } 0 \leq \bar{x}(k) \leq u(k) \tag{6.2b}$$
$$\text{if } \bar{c}(k) > \varepsilon, \text{ then } \bar{x}(k) = 0 \tag{6.2c}$$
$$\text{if } \bar{c}(k) < -\varepsilon, \text{ then } \bar{x}(k) = u(k) \tag{6.2d}$$

then $\bar{x}$ is said to be an *ε–optimum* for $\min\{cx : Ax = r, 0 \leq x \leq u\}$. For $\varepsilon = 0$, these conditions are the optimality conditions (6.1a)–(6.1d). If $\varepsilon < 1/\bar{v}$ and the cost vector is all integer, then an ε–optimum is also optimal for $\min\{cx : Ax = r, 0 \leq x \leq u\}$. Proofs are available in both Bertsekas (1991) and Ahuja et al. (1993). The method begins with a fairly large value for ε and a solution $(\bar{x}, \bar{c})$ that satisfies (6.2b)–(6.2d). After a series of flow and reduced cost adjustments, (6.2a) is also satisfied. Then ε is reduced and the process is repeated until $\varepsilon < 1/\bar{v}$.

For a given flow vector $\bar{x}$, the excess flow $e(v)$ at node v is given by

$$e(v) = r(v) - \sum_{a \in F^*(v)} \bar{x}(a) + \sum_{a \in B^*(v)} \bar{x}(a).$$

Hence, when $e(v) > 0$ the node v has excess supply that must be distributed. A *push* operation attempts to push flow away from v. This can be accomplished by increasing flow in some $a \in F^*(v)$ or decreasing flow in some $a \in B^*(v)$. If a push maintaining (6.2b)–(6.2d) is not possible, $\bar{\pi}(v)$ is modified so that a push will be permissible. The procedure implementing this approach follows.

A number of heuristic strategies for improving the computational performance of scaling push-relabel algorithms can be found in Goldberg (1997).

6.5 SOFTWARE IMPLEMENTATIONS AND EMPIRICAL EVALUATIONS

The first software implementations which clearly demonstrated the power of the primal network simplex algorithm were developed by Srinivasan and Thompson (1973)

```
procedure $Dual\text{-}Network\text{-}Simplex(V,E,c,u,r)$
/* returns $\bar{x} \in X$ such that $\bar{x} \in \arg\min\{cx : x \in X\}$ */
/* $[V,E]$ is a network */
/* $c$ is the unit cost vector */
/* $u$ is the arc capacity vector */
/* $r$ is the requirements vector */
/* assumption: $X = \{x : Ax = r, 0 \le x \le u\} \ne \phi$ */
/* assumption: $u(k) < \infty \forall k \in E$ */
begin
    Obtain $(B,L,U)$ with $\bar{x}$ the corresponding dual feasible solution;
    repeat
        /* Check Primal Feasibility */
        $I \leftarrow \{a : \bar{x}(a) < 0$ or $\bar{x}(a) > u(a)\}$;
        if $I = \phi$ then
            return $\bar{x}$;
        end
        /* Dual Calculation */
        Obtain a set of duals $\bar{\pi}$ for which $\bar{c}(k) = 0 \forall k \in B$;
        /* Select Leaving Variable */
        select $q \in I$;
        /* Select Entering Variable */
        $d(q) \leftarrow 1; d(a) \leftarrow 0 \forall a \in B \setminus \{q\}$;
        Obtain $\gamma(v) \forall v \in V$ so that $d(k) - \gamma(i) + \gamma(j) = 0 \forall k = (i,j) \in B$;
        /* the above values follow the dual calculation scheme */
        if $\bar{x}(q) > 0$ then
            $w(i,j) \leftarrow (\gamma(j) - \gamma(i)) \forall (i,j) \in L \cup U$;
        else
            $w(i,j) \leftarrow (\gamma(i) - \gamma(j)) \forall (i,j) \in L \cup U$;
        end
        $\Gamma \leftarrow \{a \in L : w(a) < 0\} \cup \{a \in U : w(a) > 0\}$;
        $\delta \leftarrow \max\left\{\frac{\bar{c}(a)}{w(a)} : a \in \Gamma\right\}, \Lambda \leftarrow \left\{a \in \Gamma : \frac{\bar{c}(a)}{w(a)} = \delta\right\}$;
        select $k = (t,h) \in \Lambda$;
        if $k \in L$ then
            $(o,d) \leftarrow (h,t)$
        else
            $(o,d) \leftarrow (t,h)$;
        end
        /* Column Update */
        $P \leftarrow$ the path in $[V,B]$ from $o$ to $d$; $Q \leftarrow \{k \in B : k \in P\}$;
        $Q^+ \leftarrow \{k \in Q : k$ is traversed in the normal direction in $P\}$;
        $Q^- \leftarrow Q \setminus Q^+$;
        /* Flow Update */
        if $\bar{x}(q) > u(q)$ then
            $\Delta \leftarrow \bar{x}(q) - u(q)$;
        else
            $\Delta \leftarrow -\bar{x}(q)$;
        end
        $\bar{x}(a) \leftarrow \bar{x}(a) + \Delta \forall a \in Q^+; \bar{x}(a) \leftarrow \bar{x}(a) - \Delta \forall a \in Q^-$;
        if $k \in L$ then
            $\bar{x}(k) \leftarrow \Delta$;
        else
            $\bar{x}(k) \leftarrow u(k) - \Delta$;
        end
        /* Basis Exchange */
        $B \leftarrow (B \setminus \{q\}) \cup \{k\}$;
        if $\bar{x}(q) = u(q)$ then
            $U \leftarrow U \cup \{q\}$;
        else
            $L \leftarrow L \cup \{q\}$;
        end
        $L \leftarrow L \setminus \{k\}; U \leftarrow L \setminus \{k\}$;
    until exited;
end
```

Algorithm 6.4: Dual simplex network method

```
procedure Dual-Feasible-Solution$(V, E, c, u, r)$
/* returns $(B, L, U)$ with $\bar{x}$ dual feasible */
/* $[V, E]$ is a network */
/* $c$ is the unit cost vector */
/* $u$ is the arc capacity vector */
/* $r$ is the requirements vector */
/* assumption: $X = \{x : Ax = r, 0 \leq x \leq u\} \neq \phi$ */
/* assumption: $u(k) < \infty \, \forall k \in E$ */
begin
    Obtain $B$ such that $[V, B]$ is a spanning tree for $[V, E]$;
    /* apply procedure Duals On A Tree */
    Obtain a set of duals $\bar{\pi}$ for which $\bar{c}(k) = 0 \, \forall k \in B$;
    $L \leftarrow \{a \in E : \bar{c}(a) > 0\}, \bar{x}(a) \leftarrow 0 \forall a \in L$;
    $U \leftarrow \{a \in E : \bar{c}(a) < 0\}, \bar{x}(a) \leftarrow u(a) \forall a \in U$;
    $b(v) \leftarrow r(v) - \sum_{(v,j) \in F^*(v)} u(v,j) + \sum_{(i,v) \in B^*(v)} u(i,v) \forall v \in V$;
    Apply procedure Flows-On-A-Tree to obtain $\bar{x}(a) \forall a \in B$;
end
```

Algorithm 6.5: Dual Feasible Solution

and by Glover and Klingman and their colleagues at the University of Texas at Austin (see Glover et al. (1974a), Glover et al. (1974b), and Glover et al. (1974c)). Other important contributions were made by Bradley et al. (1977) and Barr et al. (1979). Variations of the three methods presented in this exposition have been implemented in software and evaluated in a number of studies.

Goldberg (1997) reports on an experiment involving the five codes listed in Table 6.1. Kennington and Whitler (1998) report on an experiment with the six codes listed in Table 6.2. The scaling push-relabel code of Goldberg was found to be both fast and robust over a variety of problem structures. All the techniques and codes have their champions, but it is generally accepted that specialized network codes are at least two orders of magnitude faster than general linear programming software that can be used for these problems. Several solvers are available on-line at `http://www-neos.mcs.anl.gov/neos/` or `ftp://dimacs.rutgers.edu/pub/netflow/`.

```
procedure Scaling Push-Relabel($V, E, c, u, r, \alpha$)
/* Returns $\bar{x} \in X$ such that $\bar{x} \in \arg\min\{cx : x \in X\}$ */
/* $[V,E]$ is a network */
/* $c$ is the unit cost vector */
/* $u$ is the arc capacity vector */
/* $r$ is the requirements vector */
/* $\alpha$ is a reduction constant greater than 1 */
/* assumption: $X = \{x : Ax = r, 0 \leq x \leq u\} \neq \phi\}$ */
/* assumption: $u(k) < \infty \, \forall k \in E$ */
/* assumption: $c(k)$ is an integer $\forall k \in E$ */
begin
    $\varepsilon \leftarrow \max\{| c(a) | : a \in E\}$;
    $\bar{x}(a) \leftarrow 0 \forall a \in E, \bar{c}(a) \leftarrow c(a) \forall a \in E$;
    repeat
        if $\varepsilon < 1/\bar{v}$ then
            return $\bar{x}$;
        end
        /* Scale */
        $\varepsilon \leftarrow \varepsilon/\alpha$;
        forall $a \in E$ do
            if $\bar{c}(a) > 0$ then
                $\bar{x}(a) \leftarrow 0$;
            end
            if $\bar{c}(a) < 0$ then
                $\bar{x}(a) \leftarrow u(a)$;
            end
        end
        forall $v \in V$ do
            $e(v) = r(v) - \sum_{a \in F^*(v)} \bar{x}(a) + \sum_{a \in B^*(v)} \bar{x}(a)$;
            while $\exists v \in V$ such that $e(v) > 0$ do
                /* Push */
                if $\exists a = (v, j) \in E$ such that $\bar{c}(a) < 0$ and $\bar{x}(a) < u(a)$ then
                    $\delta \leftarrow \min\{e(v), u(a) - \bar{x}(a)\}$;
                    $\bar{x}(a) \leftarrow \bar{x}(a) + \delta$;
                    $e(v) \leftarrow e(v) - \delta$;
                    $e(j) \leftarrow e(j) + \delta$;
                else
                    if $\exists a = (i, v) \in E$ such that $\bar{c}(a) > 0$ and $\bar{x}(a) > 0$ then
                        $\delta \leftarrow \min\{e(v), \bar{x}(a)\}$;
                        $\bar{x}(a) \leftarrow \bar{x}(a) - \delta$;
                        $e(i) \leftarrow e(i) + \delta$;
                        $e(v) \leftarrow e(v) - \delta$;
                    else
                        /* Relabel */
                        $\Gamma_1 \leftarrow \min\{\bar{c}(a) : a \in F^*(v)$ and $\bar{x}(a) < u(a)\}$;
                        $\Gamma_2 \leftarrow \min\{-\bar{c}(a) : a \in B^*(v)$ and $\bar{x}(a) > 0\}$;
                        $\Gamma \leftarrow \varepsilon + \min\{\Gamma_1, \Gamma_2\}$;
                        $\bar{c}(a) \leftarrow \bar{c}(a) - \Gamma \forall a \in F^*(v)$;
                        $\bar{c}(a) \leftarrow \bar{c}(a) + \Gamma \forall a \in B^*(v)$;
                    end
                end
            end
        end
    until exited;
end
```

Algorithm 6.6: Scaling Push Relabel Algorithm

Table 6.1 Computer Codes in the Goldberg (1997) Study

Code Name	*Algorithm Type*	*Reference*
NETFLO	Primal Simplex	Kennington and Helgason (1980)
RNET Version 3.61	Primal Simplex	Grigoriadis (1986)
RELAX Version III	Scaling Push-Relabel	Bertsekas and Tseng (1990)
SPUR	Scaling Push-Relabel	Goldberg and Kharitonov (1993)
CS	Scaling Push-Relabel	Goldberg (1997)

Table 6.2 Computer Codes in the Kennington and Whitler (1998) Study

Code Name	*Algorithm Type*	*Reference*
NETFLO	Primal Simplex	Kennington and Helgason (1980)
CPLEX Version 4.0	Primal Simplex	CPLEX Callable Library
RELAX Version IV	Scaling Push-Relabel	Bertsekas and Tseng (1994)
CS2	Scaling Push-Relabel	Goldberg (1992)
NETFLO2 (Primal)	Primal Simplex	Kennington and Whitler (1998)
NETFLO2 (Dual)	Primal Simplex	Kennington and Whitler (1998)

Bibliography

R. Ahuja, T. Magnanti, and J. Orlin. *Network Flows: Theory, Algorithms, and Applications.* Prentice-Hall, Englewood Cliffs, NJ 07632, 1993.

R. Barr, F. Glover, and D. Klingman. Enhancement of spanning tree labeling procedures for network optimization. *INFOR*, 17:16–34, 1979.

D. Bertsekas. *Linear Network Optimization: Algorithms and Codes.* The MIT Press, Cambridge, MA, 1991.

D. Bertsekas and P. Tseng. RELAXT-III: A new and improved version of the RELAX code, lab. for information and decision systems report p-1990. Technical report, MIT, Cambridge, MA, 1990.

D. Bertsekas and P. Tseng. RELAX-IV: A faster version of the RELAX code for solving minimum cost flow problems. Technical report, Department of Electrical Engineering and Computer Science, MIT, Cambridge, MA, 1994.

R. Bland and D. Jensen. On the computational behavior of a polynomial-time network flow algorithm. *Mathematical Programming*, 54:1–41, 1992.

G. Bradley, G. Brown, and G. Graves. Design and implementation of large-scale primal transshipment algorithms. *Management Science*, 21:1–38, 1977.

A. Charnes and W. Cooper. *Management Models and Industrial Applications of Linear Programming: Volume I.* John Wiley and Sons, Inc., New York, NY, 1967.

G. Dantzig. Application of the simplex method to a transportation problem. In T. Koopmans, editor, *Activity Analysis of Production and Allocation*, pages 359–373. John Wiley and Sons, Inc., New York, NY, 1951.

G. Dantzig. *Linear Programming and Extensions.* Princeton University Press, Princeton, NJ, 1963.

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

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

F. Glover, D. Karney, D. Klingman, and A. Napier. A computational study on start procedures, basis change criteria, and solution algorithms for transportation problems. *Management Science*, 20:793–813, 1974b.

F. Glover, D. Klingman, and J. Stutz. Augmented threaded index method for network optimization. *INFOR*, 12:293–298, 1974c.

A. Goldberg. An efficient implementation of a scaling minimum cost flow algorithm,. Technical Report STAT-CS-92-1439, Computer Science Department, Stanford University, Stanford, CA, 1992.

A. Goldberg. An efficient implementation of a scaling minimum-cost flow algorithm. *Journal of Algorithms*, 22:1–29, 1997.

A. Goldberg and M. Kharitonov. On implementing scaling push-relabel algorithms for the minimum-cost flow problem. In D. Johnson and C. McGeoch, editors, *Network Flows and Matching: First DIMACS Implementation Challenge*, pages 157–198. AMS, Providence, RI, 1993.

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

J. Kennington and R. Helgason. *Algorithms for Network Programming*. John Wiley and Sons, Inc., New York, NY, 1980.

J. Kennington and R. Mohamed. An efficient dual simplex optimizer for generalized networks. In R. Barr, R. Helgason, and J. Kennington, editors, *Interfaces in Computer Science and Operations Research*, pages 153–182. Kluwer Academic Publishers, Norwell, MA 02061, 1997.

J. Kennington and J. Whitler. Simplex versus cost scaling algorithms for pure networks: An empirical analysis. Technical Report 96-CSE-8, Department of Computer Science and Engineering, Southern Methodist University, Dallas, TX, 1998.

H. Röck. Scaling techniques for minimal cost network flows. In U. Pape, editor, *Discrete Structures and Algorithms*, pages 181–191. Carl Hanser, Munich, 1980.

V. Srinivasan and G. Thompson. Benefit-cost analysis of coding techniques for the primal transportation algorithm. *Journal of the Association for Computing Machinery*, 20:194–213, 1973.

É. Tardos. A strongly polynomial minimum cost circulation algorithm. *Combinatorica*, 5:247–255, 1985.