Copyright © 1982 American Telephone and Telegraph Company
THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 61, No. 6, July–August 1982
Printed in U.S.A.

A Primal Algorithm for Finding Minimum-Cost Flows in Capacitated Networks With Applications

By C. L. MONMA and M. SEGAL

(Manuscript received December 7, 1981)

Algorithms for finding a minimum-cost, single-commodity flow in a capacitated network are based on variants of the simplex method of linear programming. We describe an implementation of a primal algorithm which is fast and can solve large problems. The major ideas incorporated are (i) the sparsity of the network is used to reduce the time and computer storage space requirements; (ii) basic solutions are stored compactly as spanning trees of the network; (iii) a candidate stack is used to allow flexible strategies in choosing an arc to enter the basis tree; (iv) the predecessor and thread data structures are used to efficiently traverse the tree and to update the solution at each iteration; (v) rules are implemented to avoid cycling or stalling caused by degeneracy; and (vi) piecewise-linear, convex arc costs are handled implicitly. The Primal Network Flow Convex (PNFC) code implements this algorithm and three examples, from communication networks, that can be solved with PNFC are discussed: (i) solving the area transfer problem; (ii) scheduling the collection of traffic data records; and (iii) planning the placement of pair-gain systems.

I. INTRODUCTION

This paper describes an implementation of the network simplex method for solving the transshipment problem. The problem consists of finding a minimum-cost, single-commodity flow in a capacitated network satisfying supply-and-demand constraints. The familiar transportation, assignment, and shortest route problems are all related to the transshipment problem. The area of network-flow theory has been well studied and has proven useful in many areas of applications. (See Refs. 1 to 7.)

A number of different methods have been proposed over the years to solve network-flow problems. These include primal, dual, primal-dual, out-of-kilter, negative-cycle, and scaling methods. A survey of computer codes is presented by Charnes et al.[8] Until recently, primal-dual approaches were believed to be superior to the others. However, much work has been done involving data structures to reduce the computation time and space requirements for primal-network flow codes. The primal approach is a specialization of the bounded-variable, primal-simplex algorithm of linear programming which takes advantage of the special structure of network problems. We describe an implementation of a primal algorithm that is fast and can solve large problems. The major ideas used are as follows:

(*i*) The sparsity of the network is used to reduce the time and space requirements for solving network-flow problems.

(*ii*) Basic solutions are stored compactly as spanning trees of the network.

(*iii*) A candidate stack is used to allow flexible strategies in choosing an arc to enter the tree.

(*iv*) The predecessor and thread data structures are used to efficiently traverse the spanning tree and to update the solution at each iteration.

(*v*) Rules by Cunningham[9,10] are implemented to avoid cycling or stalling caused by degeneracy.

(*vi*) Piecewise-linear, convex arc costs (or, equivalently, multiple-parallel arcs) are handled implicitly.

The Primal Network Flow Convex (PNFC) code is an implementation of this algorithm and is described in more detail by Seery.[11,12] An overview flowchart of the PNFC code is shown in Fig. 1. We will only discuss the INITIAL and SOLVE modules, which perform analogous functions to Phases I and II of the simplex method of linear programming.

The problem of finding a minimum-cost flow in a network with linear costs is described in detail in Section II. The primal network approach is then outlined in Section III. We describe the data structures used to implement the algorithm in Section IV. The major steps of determining an initial solution, finding an entering arc and leaving arc, and exchanging these arcs are detailed in Sections V, VI, VII, and VIII, respectively. Extensions to other than linear arc costs are described in Section IX. Finally, in Section X, we describe three problems in communication networks that can be modeled as network-flow problems:

(*i*) Solving the area transfer problem.

(*ii*) Scheduling the collection of traffic data records.

(*iii*) Planning the placements of pair-gain devices.

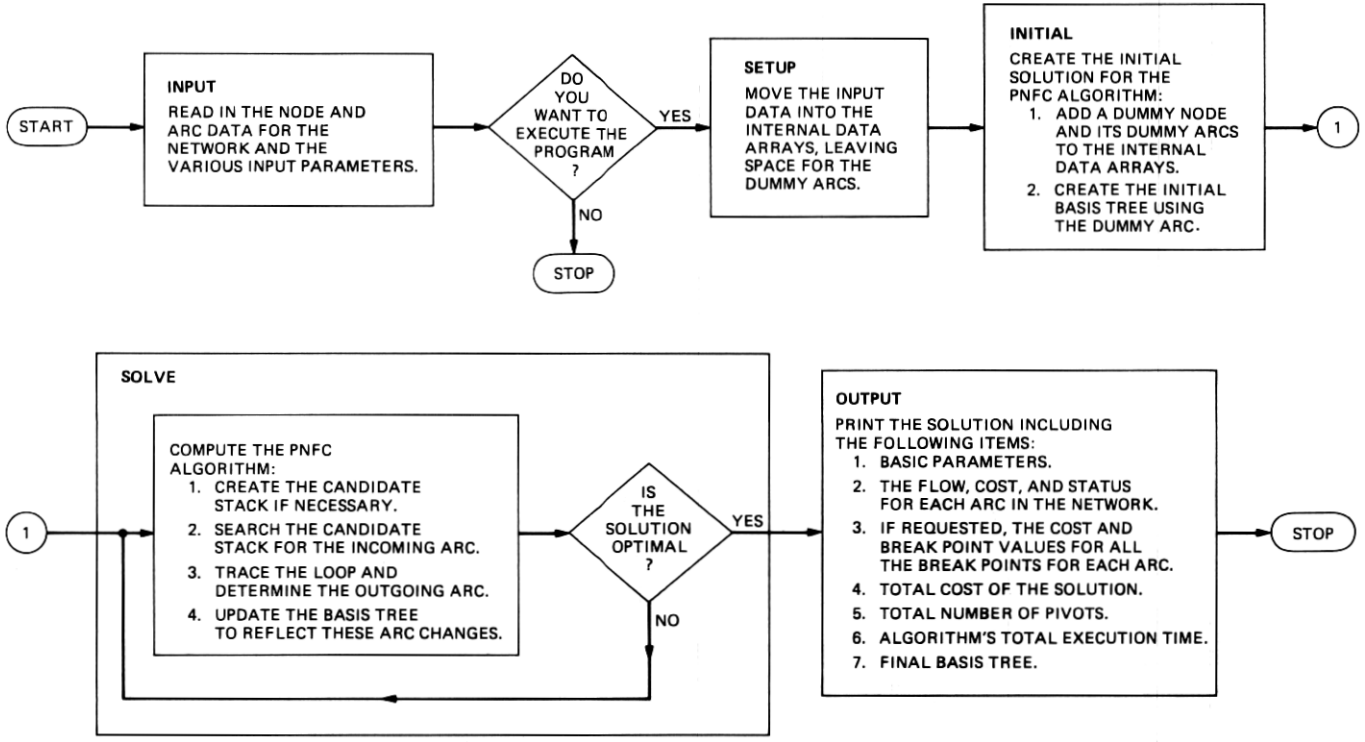

Fig. 1—Overview of the PNFC program.

II. PROBLEM FORMULATION

We refer the reader to Harary[13] for graph-theoretic concepts necessary for defining the problem; however, we do define a few of these concepts here. Consider a directed graph $G = (N, A)$ consisting of a set N of nodes and a set A of ordered pairs of nodes called arcs. Associated with each node i is a value d_i, representing the amount of supply of a commodity available at the node. A negative "supply" represents a demand for the commodity at the node. Each arc (i, j) has a lower bound a_{ij} and upper bound b_{ij} on the number of units of the commodity that is allowed to flow along the arc from node i to node j. Also, there is a shipping cost of c_{ij} per unit of flow along each arc (i, j). An example network is shown in Fig. 2, with each node i labeled by d_i, and each arc (i, j) labeled by (arc no. a_{ij}, b_{ij}, c_{ij}). A feasible flow $x = (x_{ij})$ assigns flow values x_{ij} to arcs (i, j) which satisfy the conservation-of-flow constraints eq. (2) and the capacity requirements eq. (3). An example of feasible flow for the network in Fig. 2 is shown in Fig. 3, where each arc is labeled by its flow value. The

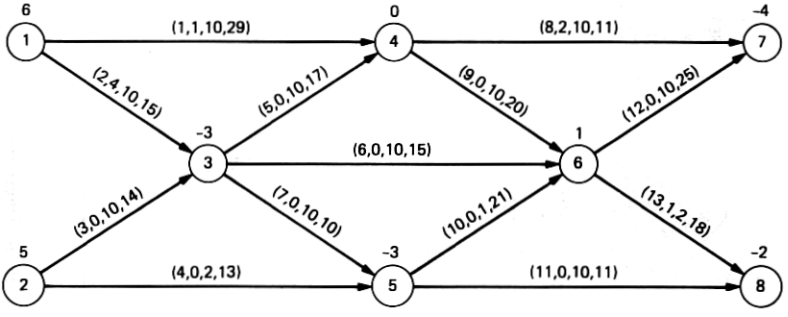

Fig. 2—Network.

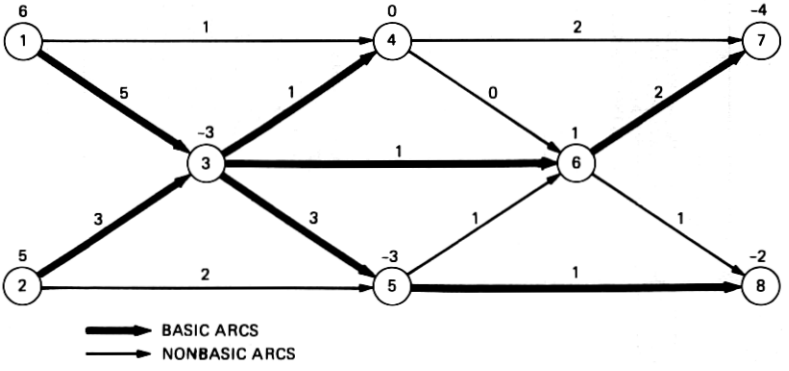

Fig. 3—Feasible flow.

objective is to find a feasible flow that minimizes the total cost of eq. (1). This problem corresponds to the following Primal Linear Program (PLP): Minimize

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

subject to

$$\sum_{j:(i,j)\in A} x_{ij} - \sum_{k:(k,i)\in A} x_{ki} = d_i \qquad i \in N \tag{2}$$

$$a_{ij} \le x_{ij} \le b_{ij} \qquad (i,j) \in A. \tag{3}$$

The summations are taken only over existing arcs. This emphasizes the fact that sparsity of the network is exploited by only storing data for arcs that are present in the network and only performing operations on these arcs.

We do, of course, allow multiple (or parallel) arcs between nodes i and j. These arcs can each be given explicitly with associated costs and bounds and would be treated as distinct arcs. In Section IX, we describe how the implementation of this algorithm handles parallel arcs implicitly.

It is well known that PLP can be reformulated with either all supply values equal to zero or all lower bounds equal to zero. However, since some problems are formulated naturally in one way and some in the other way, we allow both the supply and lower-bound values to be nonzero.

We assume that the network is connected. If not, the problem decomposes into independent subproblems on the connected components. We make this assumption merely to simplify the discussion. In fact, the algorithm described here converts the given network into a connected one by appending a dummy node and dummy arcs as part of obtaining an initial feasible solution.

III. PRIMAL NETWORK ALGORITHM

The primal-simplex algorithm of linear programming was specialized to the transshipment problem by Dantzig[3]. It leads to the "stepping-stone" approach as described by Charnes and Cooper.[1] This approach was generally believed to be computationally inferior to the existing primal-dual methods. However, recent work on primal network approaches has taken advantage of data structures to streamline the calculations involved and reduce the storage requirement. In this section, we outline the major facts related to the application of the primal-simplex approach to network problems. A more complete analysis is provided by Bradley et al.[14]

3.1 Basic solutions

The primal network algorithm is a specialization of the bounded-variable, simplex method. This allows the capacity constraints eq. (3) to be handled implicitly in the computations. The remaining constraints eq. (2) can be written in matrix notation as $Ax = d$, where A is the node-arc incidence matrix for the network and d is the vector of supply values. A solution x to eq. (2) is called basic if the columns of A associated with the variables x_{ij}, which are strictly between their bounds, are linearly independent. The value of the basic variables are uniquely determined once the nonbasic variables have been specified to be equal to their lower or upper bounds. Basic solutions correspond to the bases of the column space of A. Also, the feasible basic solutions are exactly the extreme points of the polyhedron described by eqs. (2) and (3). Hence, for any (linear) objective function eq. (1), some basic solution is optimal.

Each column of A corresponds to an arc in the network. Therefore, since the network is connected, the basic solutions correspond to subsets of arcs that form spanning trees in the network; the remaining arcs have flow values that are restricted to their lower or upper bound. The darker arcs in Fig. 3 form a basis tree.

3.2 Optimality criteria

Assign dual variables (or node potentials) u_i to each constraint in eq. (2), and variables v_{ij} and w_{ij} to each lower- and upper-bound constraint, respectively, in eq. (3). The Dual Linear Program (DLP) for PLP is defined as follows: Maximize

$$\sum_{i \in N} d_i u_i + \sum_{(i,j) \in A} a_{ij} v_{ij} - \sum_{(i,j) \in A} b_{ij} w_{ij} \tag{4}$$

subject to

$$u_i - u_j + v_{ij} - w_{ij} \leq c_{ij} \qquad (i, j) \in A \tag{5}$$

$$v_{ij}, w_{ij} \geq 0 \qquad (i, j) \in A. \tag{6}$$

Using linear programming duality theory, the complementary slackness conditions that optimal solutions to PLP and DLP must satisfy are given by

$$x_{ij}(c_{ij} - u_i + u_j - v_{ij} + w_{ij}) = 0, \tag{7}$$

and

$$v_{ij}(x_{ij} - a_{ij}) = 0, \tag{8}$$

$$w_{ij}(b_{ij} - x_{ij}) = 0. \tag{9}$$

By defining the reduced cost of an arc (i, j) to be

$$\bar{c}_{ij} = c_{ij} - u_i + u_j, \tag{10}$$

the optimality conditions for a pair of primal and dual solutions become

$$\text{if} \quad a_{ij} < x_{ij} < b_{ij} \quad \text{then} \quad \bar{c}_{ij} = 0, \tag{11}$$

$$\text{if} \quad x_{ij} = a_{ij} \quad \text{then} \quad \bar{c}_{ij} \geq 0, \tag{12}$$

and

$$\text{if} \quad x_{ij} = b_{ij} \quad \text{then} \quad \bar{c}_{ij} \leq 0. \tag{13}$$

3.3 Basis exchange

Consider a basic feasible solution x^* for PLP. The *node potentials* u^* are determined using eq. (11) for the basic arcs; these values are uniquely determined once any one node potential value is fixed. If eqs. (12) and (13) are satisfied, then the optimality of the solutions x^* and u^* follows from their feasibility and complementary slackness. Otherwise, an entering (nonbasic) arc and a leaving (basic) arc are identified and exchanged to obtain a new basic solution. This process is repeated until optimality is reached.

A nonbasic arc that violates eq. (12) or (13) is selected to enter the basis; that is, arc (i, j) is chosen if

$$x_{ij} = a_{ij} \quad \text{and} \quad \bar{c}_{ij} < 0, \quad \text{or}$$

$$x_{ij} = b_{ij} \quad \text{and} \quad \bar{c}_{ij} > 0.$$

The entering arc forms a unique cycle (or loop) when added to the basis tree. By interpreting $u_i - u_j$ to be the cost of sending one unit of flow from node j to node i, using arcs only in the basis tree, $\bar{c}_{ij} < 0$ implies that increasing the flow along arc (i, j) and sending this flow around the loop from j back to i will strictly decrease the cost. [This is true also when decreasing the flow on arc (i, j) when $\bar{c}_{ij} > 0$.] Since this conserves the flow at each node, the amount of flow change on the loop is limited only by the capacity constraints on the arcs of the loop. One of the arcs whose flow reaches its bound first is selected to leave the basis. For example, increasing the flow on the nonbasic arc (4, 7) in Fig. 3 increases the flow on arc (3, 4) and decreases the flow on arcs (3, 6) and (6, 7). Arc (3, 6) reaches its bound first. Exchanging arcs (4, 7) and (3, 6) forms the new basis tree shown in Fig. 4.

IV. DATA STRUCTURES

The contemporary work on network algorithms has focused on using data structures to improve the effectiveness of primal network codes. Since network problems are typically sparse, with many node pairs having no arcs between them, we only store the data and perform

Table I—Data for the network of Fig. 2

Arc	1	2	3	4	5	6	7	8	9	10	11	12	13
Lower Bound	1	4	0	0	0	0	0	2	0	0	0	0	1
Upper Bound	10	10	10	2	10	10	10	10	10	1	10	10	2
Cost	29	15	14	13	17	15	10	11	20	21	11	25	18

calculations for the arcs actually present in the network. The arcs are stored in a list, with all of the arcs pointing out of a node grouped together. The costs and bounds are kept in similar arc-length lists. This is illustrated in Table I for the network in Fig. 2.

The representation of a basis as a tree results in a compact format for storing the relevant data. This format is shown in Table II for the basis tree in Fig. 4. A basis tree is specified by identifying one node to be the root node and indicating a predecessor NPRED(i) for each node i, which is the next node after i on the unique path from i to the root node in the tree. The physical direction of the arc between i and NPRED(i) is distinguished by setting NDIRECT(i) equal to 1 if the arc points from i to NPRED(i) and to -1 otherwise. The variable DUALV(i) is the dual variable associated with each node i. The amount of flow on the arc between i and NPRED(i) is given by FLOW(i).

The loop formed by an entering nonbasic arc (i, j) and the basis tree can be traced, in one pass, by following the predecessor paths from i and j back towards the root until these paths meet. The depth NDEPTH(i) of a node i is the number of nodes on the path from i to the root. The predecessor and depth arrays are used to proceed up

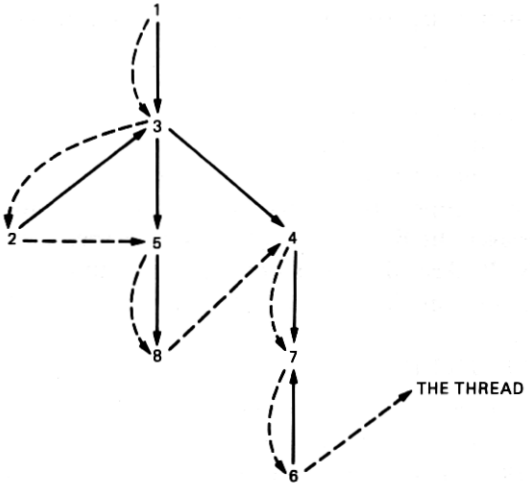

Fig. 4—Basis tree.

Table IIa—Data structures for the basis tree of Fig. 4—node data

Node	1	2	3	4	5	6	7	8
NPRED	0	3	1	3	3	7	4	5
NDIRECT	0	1	−1	−1	−1	1	−1	−1
DUALV	0	1	15	32	25	18	43	36
FLOW	0	3	5	2	3	1	3	1
NDEPTH	1	3	2	3	3	5	4	4
LTHRED	3	5	2	7	8	0	6	4

Table IIb—Data structure for the basis tree of Fig. 4—arc data

Arc	1	2	3	4	5	6	7	8	9	10	11	12	13
KASE	0	−1	−1	1	−1	0	−1	−1	0	1	−1	−1	0

these paths, staying at the same depth on each path so that the search ends at the node where the paths meet, thus forming a loop.

After the entering and leaving arcs have been determined, it is necessary to update the basis tree. This involves traversing all of the successors of a node to update their dual variable and depth values. A useful tool for doing this is the "thread." The thread LTHRED(i) of a node i labels the nodes of the tree in "preorder" according to the following recursive rule: First label the root. Then label the nodes in each of the subtrees belonging to the immediate successors of the root in preorder. (The order in which these subtrees are visited is immaterial.) The thread is illustrated by dotted arcs in Fig. 4.

A final data structure keeps track of the status of each arc. The array KASE(LN) is set to 0 if arc LN is currently nonbasic at its lower bound, 1 if it is nonbasic at its upper bound, and −1 if it is basic.

These data structures are illustrated in Tables IIa and IIb for the basic solution in Fig. 4, with node 1 as the root.

V. INITIAL SOLUTION

The primal network-flow algorithm proceeds from one basic solution to another until an optimal solution is found. It is necessary to provide an initial feasible solution to begin this iterative process. Several methods are available for obtaining initial feasible solutions.[14] We use the "Big-M" method of artificially creating a feasible flow by adding a dummy node and dummy arcs. This method is easy to implement and understand, and works well.[15]

The method begins by initialing the flow in all arcs to be at their lower bounds, that is, all arcs are nonbasic at their lower bound and the KASE array is set to zero. This induces a net supply at each node i with value

$$\text{SPLYNT}(i) = d_i - \sum_{j:(i,j)\in A} a_{ij} + \sum_{k:(k,i)\in A} a_{ki}.$$

For the network in Fig. 2, the values are shown in Table III. A feasible flow is formed by adding a dummy node as the root of the tree and dummy arcs between the dummy node and all the other nodes. These dummy arcs form the basis tree. The direction of the dummy arc in the tree is towards the root if the net supply of the node is nonnegative, and away from the root otherwise. The amount of flow is equal to the absolute value of the net supply of the node. The cost of the dummy arcs is set to be very large if the dummy arc points away from the root and is set to zero otherwise. The dummy arcs have a lower bound of zero and an upper bound of "infinity." The dual values are calculated by setting the dual variable for the root to zero and solving for the others using eq. (11). The initial basis tree for the example network in Fig. 2 is shown in Fig. 5 with basis array data in Table IV. Later stages of the algorithm serve to remove the dummy arcs from the basis. If the algorithm terminates with positive flow in a dummy arc, then there is no feasible solution to the original problem.

VI. ENTERING ARC

Suitable candidates for arcs to enter the basis are nonbasic arcs (i, j), which are either at their lower bound with negative reduced cost, or at their upper bound with positive reduced cost, i.e., eq. (12) or eq. (13) is violated. Many strategies have been proposed and tested for

Table III—Values of the net supply for the network in Fig. 2

Node	1	2	3	4	5	6	7	8
SPLYNT	1	5	1	−1	−3	0	−2	−1

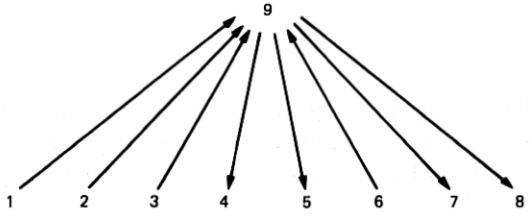

Fig. 5—Initial basis tree.

Table IV—Initial basis tree

Node	1	2	3	4	5	6	7	8	9
NPRED	9	9	9	9	9	9	9	9	0
NDIRECT	1	1	1	−1	−1	1	−1	−1	0
DUALV	0	0	0	∞	∞	0	∞	∞	0
FLOW	1	5	1	1	3	0	2	1	0
NDEPTH	2	2	2	2	2	2	2	2	1
LTHRED	2	3	4	5	6	7	8	0	1

selecting an arc to enter the basis; these range from choosing the first violating arc to choosing the most violating arc. We use a candidate stack approach[14,16] which allows flexible strategies in controlling the choice of an incoming arc.

The usage of the candidate stack is illustrated in Fig. 6. The candidate stack is replenished during a major iteration whenever it is empty or when an upper bound MXITER on the number of consecutive minor iterations is reached. The stack is filled from the arc list which has all of the arcs out of a node grouped together. The reduced costs are computed for the arcs out of a node i, and the nonbasic arc (i, j), with the largest violation (if any) among these, is placed on the stack. This is repeated for nodes $i + 1$, $i + 2$, etc., until either all nodes have been processed or the stack has reached its maximum allowable size of MXSTAK. The next major iteration begins with the node where the last major iteration left off. This periodic examination of all arcs helps prevent "stalling," which is a long sequence of pivot exchanges that does not improve the cost of the solution.[10]

Once the stack has been formed in a major iteration, minor iterations are performed to choose the nonbasic arcs from the stack with the largest violation to enter the basis. At each minor iteration, the arcs on the stack are checked for violations. If an arc no longer has a violation, it is removed from the stack. Once the stack becomes empty, or MXITER consecutive minor iterations are performed, the stack is rebuilt in a major iteration.

VII. LEAVING ARC

A nonbasic arc to enter the basis tree is chosen because it violates eq. (12) or (13). This implies that moving its flow value away from its bound and updating the flow values of the arcs on the newly formed

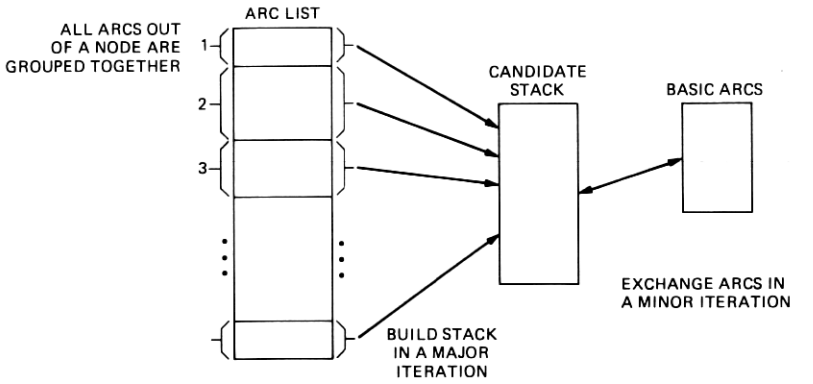

Fig. 6—Candidate stack.

loop strictly decreases the cost of the current solution. The flow is changed as much as possible until one of the arcs on the loop reaches its bound and is chosen to leave the basic solution. This is accomplished in one pass by tracing the loop using the predecessor and depth arrays.

To illustrate this, consider the basis tree in Fig. 4 with arc (6, 8) chosen as the entering arc. Arc (6, 8) can have its flow increased by one unit before reaching its upper bound. This induces a supply at node 8 and a demand at node 6. The path on the side of the loop with the induced supply is called the plus path, and the other side is called the negative path. The predecessor array is used to trace the paths from 6 and 8 towards the root, until the loop is formed where the paths join at node 3. Along the way, the amount of change in the flow value before an arc reaches its bounds is computed for each arc on the loop. The depth array is used to alternatively move up each of the two paths checking for a common node. The arcs in Fig. 4 are examined in the order (6, 7), (5, 8), (4, 7), (3, 5), and (3, 4); their maximum amount of change in flow before reaching a bound is 1, 1, 7, 7, and 8, respectively. Hence, arcs (6, 7), (5, 8), and the entering arc (6, 8) all reach their bound after a flow change of one unit. Any of these arcs could be chosen to leave the basis.

In our algorithm, the leaving arc is chosen by using Cunningham's rule[9] which is designed to avoid "cycling" caused by degeneracy. This approach hinges upon maintaining basis trees that are "strongly" feasible; that is, the arcs in the basis tree directed towards the root are required to have flow values strictly less than their upper bounds, and arcs directed away from the root have flow values strictly greater than their lower bounds. This assures us that a positive amount of flow can always be pushed up the tree. We note that the initial solution procedure always constructs a strongly feasible tree and that the tree in Fig. 3 is strongly feasible.

In the case of a tie among several arcs as candidates to leave the basis, the choice is made using the following priority: First, choose the candidate arc which is closest to the root on the plus path (if any). Next, choose the entering arc, if it is a candidate. Finally, choose the arc on the negative path furthest from the root. In the example cited above, arc (6, 7) would be chosen. This rule provides the only choice which will result in the new basis tree remaining strongly feasible.

VIII. BASIS EXCHANGE

Exchanging the entering and leaving arcs requires updating the basic solution. For example, choosing arc (1, 4) to enter the basis tree in Fig. 4 results in the choice of arc (1, 3) to leave the basis. This forms a new basis tree as illustrated in Fig. 7. This changes the data arrays associated with the basic solution. Specifically, the nodes on the loop

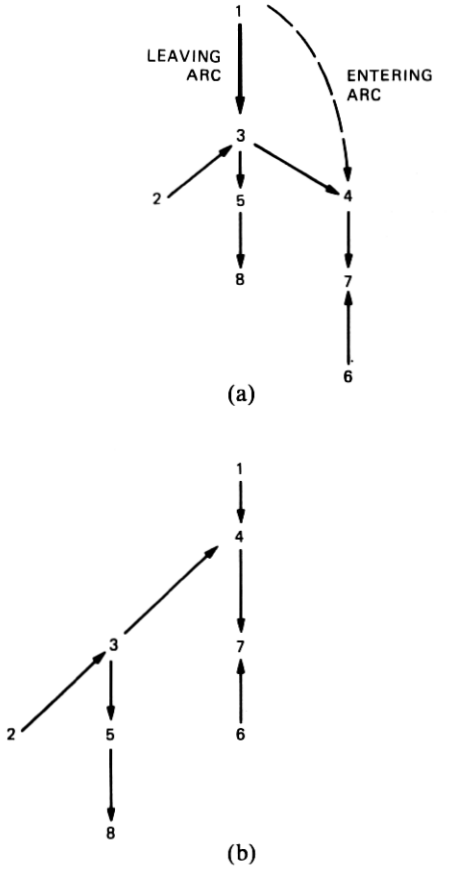

Fig. 7—Basis exchange. (a) Old basis tree. (b) New basis tree.

must be traced (as described before) to update the predecessor, direction, flow, and thread array (NPRED, NDIRECT, FLOW, and LTHRED) values. In the example, nodes 1, 3, and 4 are on the loop. In addition, the dual variable and depth array (DUALV and NDEPTH) values must be updated for the nodes in the subtree which is moved because of the basis exchange. In our example, the subtree moved is rooted at node 3 and consists of nodes 2, 3, 4, 5, 6, 7, and 8. The thread array LTHRED is used to identify the nodes in the subtree rooted at a node i as follows:

(*i*) Label the root node i to indicate that it is in the subtree. Let $j = \text{LTHRED}(i)$.

(*ii*) If $\text{PRED}(j)$ is labeled, then label j, and set $j = \text{LTHRED}(j)$, or stop with all nodes in the subtree labeled as such.

This procedure works because the nodes in the subtree appear

consecutively on the thread immediately after the root, and because the predecessor of a node always appears on the thread before the node itself does. The method used to update the basic solution values is an enhancement of a procedure described in detail by Jacobsen.[17] The change in status of the entering and leaving arcs is reflected by updating their values in the KASE array. The new array values for the basic solution of Fig. 7 are shown in Table V. The optimality of this solution may be checked by noting that the flow and dual variables are feasible to PLP and DLP, respectively, and the complementary slackness conditions eqs. (11), (12), and (13) hold.

IX. EXTENSION TO OTHER COSTS

We have described an algorithm for finding a minimum cost flow in a capacitated network with linear costs. The current implementation actually handles piecewise-linear, convex costs or, equivalently, multiple parallel arcs between two nodes. We describe how this is accomplished implicitly without any additional work required.

Consider the piecewise-linear, convex arc cost shown in Fig. 8. It consists of three line segments with cost slopes equal to c_1, c_2, and c_3, respectively. The lower bound on the allowable flow is b_0, with three *break points* of b_1, b_2, and b_3. This is equivalent to having three parallel

Table V—Data structure for the new basis tree of Fig. 7

Node	1	2	3	4	5	6	7	8
NPRED	0	3	4	1	3	7	4	5
NDIRECT	0	1	1	−1	−1	1	−1	−1
DUALV	0	−2	12	29	22	15	40	33
FLOW	0	3	1	2	3	1	3	1
NDEPTH	1	4	3	2	4	4	3	5
LTHRED	4	5	2	3	8	0	6	7

Arc Data

Arc	1	2	3	4	5	6	7	8	9	10	11	12	13
KASE	−1	0	−1	1	−1	0	−1	−1	0	1	−1	−1	0

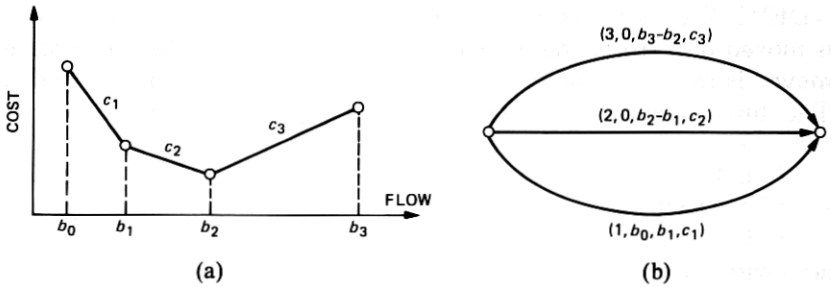

Fig. 8—Piecewise-linear, convex arc cost and equivalent parallel arc formulation.

arcs (one for each line segment) as shown in Fig. 8, with arcs labeled by arc no., lower bound, upper bound, and cost.

This situation can be handled implicitly by expanding the definition of the status of an arc LN as follows:

$$\text{KASE}(LN) = \begin{cases} k & \text{if arc } LN \text{ is nonbasic with flow value } b_k \\ -k & \text{if arc } LN \text{ is basic with flow value between } b_{k-1} \text{ and } b_k \end{cases}.$$

For this case, convexity and separability guarantees that at most one arc of each parallel arc group will be basic; therefore, we can now proceed with the algorithm as if only one arc exists between the nodes. The appropriate cost and bound values to use for this arc depend only on the status of the arc.

This feature is also useful for problems with convex arc costs. We can always approximate a convex cost function by a piecewise-linear, convex cost function as shown in Fig. 9. This can be used to obtain approximate solutions. Meyer's alternative method[18] approximates the convex cost by a piecewise-linear, convex cost, with only two line segments whose breakpoints vary from one iteration to the next. Meyer's approach is guaranteed to converge to the optimal solution.

Concave arc costs model economies of scale and are discussed by Erickson et al.,[19] where a dynamic programming solution procedure is proposed. An alternative method of solution is to use a branch-and-bound approach, solving the subproblems using a linear approximation to the concave cost. Such an approach has been applied successfully for fixed-charge transportation problems by Barr et al.[20]

X. APPLICATIONS

Numerous applications of network-flow models can be found in the references provided. We briefly describe three applications in communication systems.

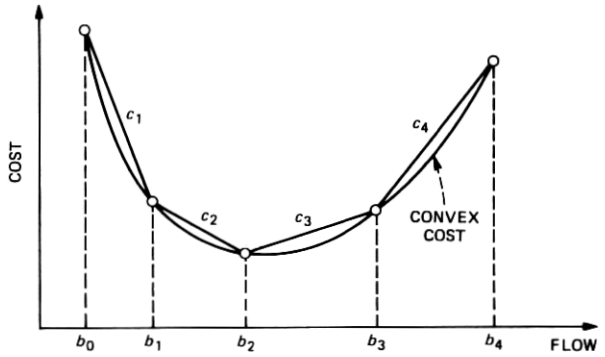

Fig. 9—Piecewise-linear approximation of a convex cost function.

Our first example illustrates a situation where piecewise-linear, convex arc costs arise in a natural way. Hammer and Segal[21] formulated a model for handling "area transfers" in a local area. The area is subdivided into districts j which are served by wire centers i. Currently, w_{ij} working lines (or wire pairs) are in use between i and j, with additional s_{ij} spare lines unused. We let

$$d_j = \sum_i w_{ij}$$

represent the current demand at district j. Future demand for lines at district j can be satisfied from wire center i by using the spare lines at a cost of λ_{ij} per line or adding additional lines beyond the available spare at a larger cost of $\lambda_{ij} + \delta_{ij}$. Alternatively, an area transfer may occur if a wire center i's capacity b_i on the number of lines it can serve is exceeded. This involves disconnecting a line from wire center i to district j at a cost of μ_{ij} per line and reconnecting it to another wire center. Given new demands $d_j + \Delta_j$ for lines at each district j, the problem of assigning customers to wire centers (with possible area transfers) at minimum total cost is a transportation problem with a piecewise-linear, cost structure for each i, j pair as shown in Fig. 10. The slope of the three line segments and the break points are indicated on the figure. Denote by x_{ij} the change in the line assignments between wire centers i and districts j, and by $c_{ij}(x_{ij})$ the separable, piecewise-linear cost functions of Fig. 10. Then, the problem is to minimize

$$\sum_i \sum_j c_{ij}(x_{ij}) \tag{14}$$

subject to

$$\left.\begin{aligned} \sum_i (w_{ij} + x_{ij}) &= d_j + \Delta_j \quad \forall_j \\ \sum_j (w_{ij} + x_{ij}) &\le b_i \quad \forall_i \end{aligned}\right\} \tag{15}$$

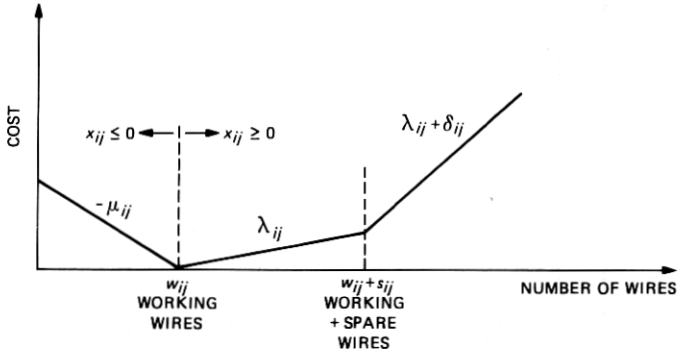

Fig. 10—Piecewise-linear, convex cost structure for the area transfer model.

$$-w_{ij} \leq x_{ij} \quad \forall_{i,j}. \tag{16}$$

In general, this formulation fits transportation models in which convex costs are incurred whenever one perturbs (increases or decreases) the established pattern of the distribution (w_{ij}). A project scheduling problem is formulated by Lawler[5] as a network-flow problem with two piecewise-linear, convex cost segments.

A second example involves scheduling the collection of call records in private communication networks and is discussed in detail by Monma and Segal.[22] A private communication network consists of tie trunks interconnecting various locations. Calls between two locations overflow to the toll network when all connecting tie trunks are busy. Each location is served by a Private Branch Exchange (PBX) which generates a call record for each call originating at its location.

A central PBX is equipped with several ports, each capable of simultaneously collecting records from different locations. According to a prescribed schedule, the central PBX initiates calls and begins transferring call records. This continues until all records stored at the start of the polling have been collected.

We describe a network-flow model used to find a good polling schedule by illustrating it in Fig. 11 for two offices (PBXs) and 48 polling periods in the day. Each period J is represented by a node J on the right side of the figure. Each office I is represented by a collection of 48 nodes arranged in a particular configuration as shown on the left side of the figure. Each node J in the configuration for office I represents the period J at the office. The arc directed horizontally into this node J indicates that a known number of records are generated during period J at office I. The arc directed from a period J to a following period $J + 1$ node signifies records in storage at the end of period J which are carried over to the start of period $J + 1$. We note that the records left in storage at the end of the last period in the day carry over to the first period of the following day.

The arc directed from the period J node of office I to the period J node on the right side of the figure indicates the polling of records from office I at the start of period J. The arc directed from the period J node to the sink node represents all of the records polled in period J among all offices. The arrrow out of the sink node means that all records generated must be polled at some time during the day.

Capacities on the arcs depend on the amount of call-record storage space available at each office and the polling capacity of the ports at the central PBX. The cost per record for polling office I during period J depends on the probability that a call from the central PBX to office I will overflow to the toll network and the cost of such a call. This network-flow model is used in an interactive way to produce polling schedules (which are determined from the flows on the arcs from

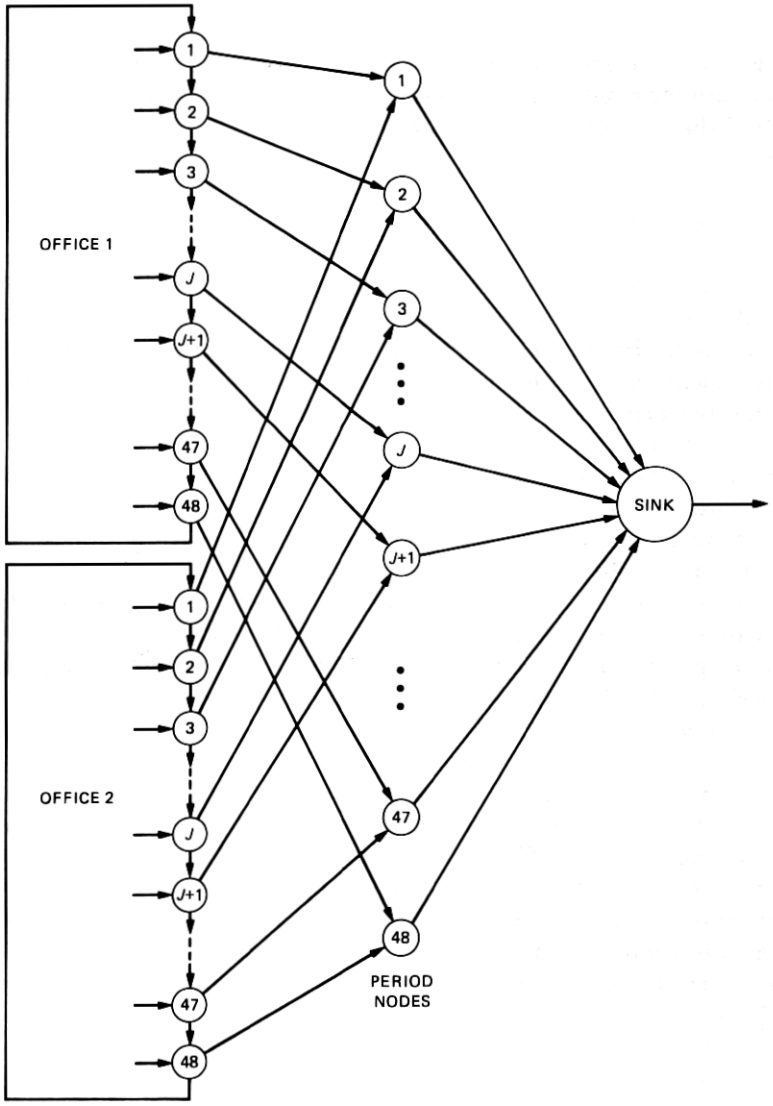

Fig. 11—Network-flow model for polling call records with two offices and 48 periods.

period J office I nodes to period J nodes) of minimum cost which avoid losing call records. (Call records may be lost if a PBX generates them faster than they are polled, thus, overflowing the storage capacity of the PBX.)

Our final example involves a capacity-expansion problem in the local network, which connects subscribers to the local switching office. Elken[23] discusses a mathematical programming formulation of this

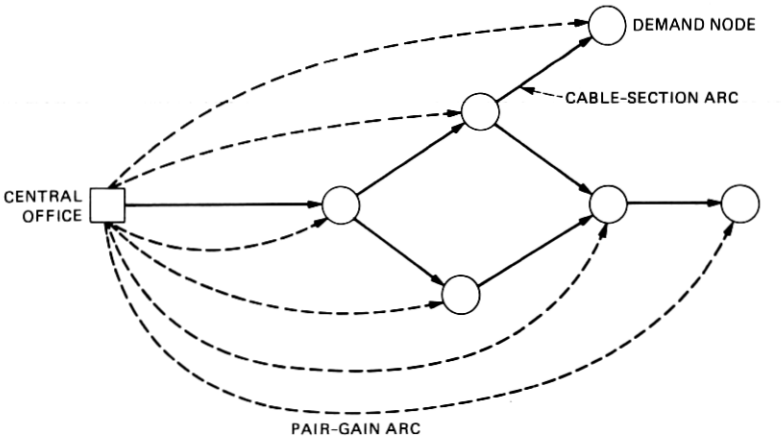

Fig. 12—Network-flow models for placement of pair-gain devices.

problem. We will describe a network-flow model by way of a stylized and simplified example. The network is composed of cable sections which connect pairs of nodes (see Fig. 12). The nodes represent demand points for groups of subscribers. The solid arcs represent cable sections. Each cable section may consist of several parallel existing or proposed cables of various sizes (and costs). The dashed arcs represent the possible placement of pair-gain systems (in which a small number of wire pairs are used to serve a larger number of customers by means of multiplexing or concentration). The piecewise-linear, convex costs arise from the multiple parallel cables in each cable section. A solution to the problem represents a least-cost expansion of cables and placement of pair-gain systems.

Finally, we mention some computational experience with the PNFC computer code. Elken[24] solved several examples of the previously described local network problem with about 50 nodes, 150 arcs, and 2 piecewise-linear sections. The average CPU time was 1.5 seconds. The scheduling of the call-records problem was solved hundreds of times for problems with about 500 nodes and 1000 arcs and required an average of 3.5 seconds. The largest example tested was a 400-node assignment problem with 40,000 arcs; this problem was solved in about 80 seconds of CPU time on the IBM Amdahl computer.

XI. ACKNOWLEDGMENT

We would like to thank A. T. Seery for an excellent job of programming and testing the PNFC.

REFERENCES

1. A. Charnes and W. W. Cooper, *Management Models and Industrial Applications of Linear Programming, I and II*, New York: John Wiley, 1961.

2. L. R. Ford and D. R. Fulkerson, *Flows in Networks*, Princeton, New Jersey: Princeton University Press, 1962.
3. G. B. Dantzig, *Linear Programming and Extensions*, Princeton, New Jersey: Princeton University Press, 1965.
4. G. H. Bradley, "Survey of Deterministic Networks," AIEE Trans., *7*, No. 3 (September 1975), pp. 222–34.
5. E. L. Lawler, *Combinatorial Optimization: Networks and Matroids*, New York: Holt, Rinehart, and Winston, 1976.
6. B. L. Golden and T. L. Magnanti, "Deterministic Network Optimization: A Bibliography," Networks, *7*, No. 2 (Summer 1977), pp. 149–83.
7. P. A. Jensen and J. W. Barnes, *Network Flow Programming*, New York: John Wiley, 1980.
8. A. Charnes et al., "Past, Present, and Future of Large-Scale Transshipment Computer Codes and Applications," Computers and Operations Res., *2*, No. 2 (September 1975), pp. 71–81.
9. W. H. Cunningham, "A Network Simplex Method," Math. Prog., *11*, No. 2 (October 1976), pp. 105–16.
10. W. H. Cunningham, "Theoretical Properties of the Network Simplex Method," Math. of Operations Res., *4*, No. 2 (May 1979), pp. 196–208.
11. A. T. Seery, unpublished work.
12. Ibid.
13. F. Harary, *Graph Theory*, Reading, Massachusetts: Addison-Wesley, 1969.
14. G. H. Bradley, G. G. Brown, and G. W. Graves, "Design and Implementation of Large-Scale Primal Transshipment Algorithms," Man. Sci., *24*, No. 1 (September 1977), pp. 1–34.
15. J. M. Mulvey, "Testing of a Large-Scale Network Optimization Program, "Math. Prog. *15*, No. 3 (November 1978), pp. 291–314.
16. J. M. Mulvey, "Pivot Strategies for Primal-Simplex Network Codes," J. Assn. Computing Mach., *25*, No. 2 (April 1978), pp. 266–70.
17. S. K. Jacobsen, "On the Use of Tree-Indexing in Transportation Algorithms," Eur. J. Operational Res., *2*, No. 1 (January 1978), pp. 54–65.
18. R. R. Meyer, "Two-Segment Separable Programming," Man. Sci., *25*, No. 4 (April 1979), pp. 385–95.
19. R. E. Erickson, C. L. Monma, and A. F. Veinott, Jr., unpublished work.
20. R. S. Barr, F. Glover, and D. Klingman, "A New Optimization Method for Large-Scale Fixed Charge Transportation Problems," Operations Res., *29*, No. 3 (May–June 1981), pp. 448–63.
21. P. L. Hammer and M. Segal, unpublished work.
22. C. L. Monma and M. Segal, unpublished work.
23. T. R. Elken, "The Application of Mathematical Programming to Loop Feeder Allocation," B.S.T.J., *59*, No. 4 (April 1980), pp. 479–500.
24. T. R. Elken, personal communication.