Transportation Science

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

Letter to the Editor—Extensions of the Augmented Predecessor Index Method to Generalized Network Problems

F. Glover, D. Klingman, J. Stutz,

Letter to the Editor

Extensions of the Augmented Predecessor Index Method to Generalized Network Problems

F. GLOVER

University of Colorado, Boulder, Colorado

and

D. KLINGMAN and J. STUTZ

University of Texas, Austin, Texas

The augmented predecessor indexing method is a procedure for efficiently updating the basis representation, flows, and node potentials in an adjacent extreme point (or 'simplex' type) method for network problems, utilizing ideas due to Ellis Johnson *in his proposed application of a triple-label representation to networks. The procedure is extended here to accommodate the more complex basis structures and updating processes of the generalized network problem, specifying rules for expediting the calculations.*

The augmented predecessor indexing (API) method[10] is a procedure for efficiently updating the basis representation, flows, and dual evaluators in transportation and network optimization problems. This procedure, which is based on Ellis Johnson's triple label representation,[12] has been recently incorporated into computer programs for transportation problems, with noteworthy success. Computational studies demonstrate these computer programs[9, 16] to be substantially faster than those previously available, solving 100×100 transportation problems in 1.4 sec and 1000×1000 transportation problems in 15 sec on the CDC 6600. In this paper we show how the API method can be extended to generalized network problems, making it possible to update the more complex 'quasi-tree' basis structures of these problems with the same types of computational efficiencies that result for ordinary network problems.

The potential applications for an efficient and clearly organized computer code for generalized network problems are significant, due to the wide range of problems that can be given a generalized network formulation.[4-6, 11] The computational advantages of a special purpose algorithm for these problems as opposed to a general purpose linear programming method are demonstrated by the studies in references 6, 14, and 15, which show that special purpose transportation and network codes (utilizing the updating procedures that are extended in this paper) solve transportation and network problems 150 times faster than the state-of-the-art commercial linear programming code, OPHELIE.

The following sections introduce the generalized network problem and describe the procedures of the extended API method, focusing on considerations relevant to computer implementation.

THE GENERALIZED NETWORK PROBLEM

THE GENERALIZED NETWORK problem may be defined as

$$\text{minimize} \quad \textstyle\sum_{(i,j)\epsilon A} c_{ij} x_{ij}, \tag{1}$$

$$\text{subject to} \quad \textstyle\sum_{(i,j)\epsilon A} a_{p,ij} x_{ij} = b_p, \qquad (p\epsilon N) \tag{2}$$

$$x_{ij} \geqq 0, \qquad [(i,j)\epsilon A] \tag{3}$$

where A is the set of arcs and N is the set of nodes for the network. Each arc (i, j), $i \neq j$, has a nonzero coefficient in exactly two of the node equations (2), i.e., the two equations corresponding to the arc's endpoints. Specifically, $a_{p,ij} \neq 0$ only if $p = i$ or $p = j$. In an ordinary network $a_{i,ij} = -1$ and $a_{j,ij} = 1$, but in a generalized network $a_{i,ij}$ and $a_{j,ij}$ can be any two nonzero quantities. Typically, however, $a_{i,ij}$ is assumed to be -1 and $a_{j,ij}$ is assumed to be positive, in which case $a_{j,ij}$ is called the 'multiplier' of the arc directed from node i to node j. This multiplier can be thought of as a factor that magnifies or attenuates the flow x_{ij} across the arc, according to whether $a_{j,ij}$ is greater or less than one.

A generalized network can also contain arcs that are 'self-loops,' leading from a node back to itself. That is, for some nodes i, there may exist 'arcs' (i, i) in A. In this case, $a_{p,ii} \neq 0$ only if $p = i$. Such self-loops are customarily used to introduce slack variables into the problem (to change inequalities into equations) and have been called slack loops (reference 5, pp. 413–424).

The quantities b_p of the node equations represent the supplies and demands at the nodes, where $b_p > 0$ is interpreted as a demand, $b_p < 0$ is interpreted as a supply.

The nonnegativity inequalities (3) are often supplemented by upper bound inequalities of the form $u_{ij} \geqq x_{ij}$, in which case the problem is referred to as capacitated, and the quantities u_{ij} are called the arc capacities. The inclusion of these capacities does not alter the basic structure of the problem, or the procedures we shall give for exploiting it.

THE TRIPLE-LABEL REPRESENTATION

THE TRIPLE-LABEL representation is a standard way to record and manipulate trees in computer list processing. Its application to network problems was proposed by Ellis Johnson, who showed that it could be used efficiently to organize the labeling and flow augmenting operations of a maximal flow algorithm,[12] sketching some of the fundamental ideas that were later elaborated in the literature via the API method.

The triple-label scheme orients the tree so that it is in fact an 'arborescence'; that is, for some single node that is identified as the 'root,' the arcs are oriented (by labels) so that the unique path from any node to the root node of the tree is a directed path. The triple-label representation may be viewed as inducing an 'ancestry relationship' on a tree, each node carrying three labels, or node indexes, which name the father, the eldest son, and next younger brother of the given node. In particular, a node is taken to be the father of all its immediate successors, these latter constituting a set of brothers, arbitrarily sequenced from eldest to youngest. Thus, the root node is the ancestor of all nodes, and has no father (immediate predecessor). Nodes at the extremities of the tree have no sons (immediate successors) and the 'last' of a set of successors of a given node has no younger brother. The father, eldest son, and next younger brother in these three extreme cases are given a 'dummy' name that communicates their nonexistence.

By the use of these labels it is possible to find all ancestors or all descendants of a given node in an obvious manner, and this constitutes the essential convenience of the triple-label representation. (It should be noted, however, that a 'threaded list' representation[13] shares this convenience, and the API method as described in this paper can as readily be implemented with the latter, using the relations developed in the next section.)

The basis structure of a generalized problem is not a tree, as in the pure network problem, but a set of disjoint quasi trees, i.e., connected graphs that have a single loop.[2, 5, 6, 15] We stipulate that the triple-label representation be applied initially to each quasi tree in the basis so that the arcs of the loop are oriented uniformly clockwise or counterclockwise, thus making each node on the loop its own ancestor. (A self-loop, which contains only one arc, may simply be assigned the orientation it receives in the network.) The 'tributary' trees (those that arise upon suppression of all loop arcs) are oriented as arborescences, whose roots consist of the nodes that lie on the loop. Hence each loop node has an 'equal' status as an ancestor of all nodes in the quasi tree, and every node has a father. (The immediate successors of a given node are arranged as usual, from eldest brother to youngest in any fashion desired.) We shall call this the 'rooted loop' orientation. This orientation, of course, has nothing to do with the 'true orientation'—i.e., actual direction of the arcs in the network.

THE EXTENDED API METHOD

Basis Representation of an Incoming Arc

In any quasi tree possessing a rooted-loop orientation, it is clear that a sequential trace of the predecessors of a given node (from father to grandfather to great grandfather, etc.) generates a *backward path* that contains all arcs on the loop. For simplicity we shall suppose that such a path is simple, i.e., not traced beyond necessity (hence, as soon as a node on this path is encountered a second time, the trace is stopped). The path itself therefore duplicated no arcs and duplicated only a single node—the node at which the loop is entered (which may be the starting node for the trace). The procedure of an adjacent extreme point algorithm identified an *incoming arc* and an *outgoing arc,* respectively from among the nonbasic and the basic arcs. The addition of the incoming arc and the removal of the outgoing arc produces a changed set of quasi trees, and this operation constitutes a fundamental step of a standard iteration or 'basis exchange' step of linear programming methods. An important aspect of this step is to identify the set of arcs on which flows will change, or more precisely, the set of basic arcs that provide a linear representation of the nonbasic incoming arc. The following observation characterizes this set of arcs.

REMARK 1. *The basic arcs that have a nonzero coefficient in the basis representation for an incoming arc* (u, v) *are contained in the backward paths* P_u *and* P_v *from node* u *and node* v.

Proof. The validity of the remark follows immediately from the fact that the network $P_u \cup P_v$ consists of one or two 'stripped' quasi trees that represent a linearly independent subsystem of the full basis, containing as many variables (arcs) as equations (nodes). To provide a more useful justification, we will show how to generate constructively a representation of (u, v) from arcs of P_u and P_v. First note that to satisfy a node requirement r_i for any node i that has exactly one incident arc (i, j), it is necessary to assign a flow w_{ij} to (i, j) precisely equal to $r_i/a_{i,ij}$. This in turn transmits a node requirement to node j of $-a_{j,ij}w_{ij}$. [If the arc incident to node i is (j, i), the double subscript ij should be replaced in the preceding by ji.] The constructed representation of (u, v) begins by setting node requirements at nodes u and v of $a_{u,uv}$ and $a_{v,uv}$ respectively. These requirements and their transmitted requirements are met by assigning appropriate flows to successive arcs of the nonintersecting portions of P_u and P_v. (This 'domino' procedure of assigning flows and transmitting requirements down a path is standard. We need to show that the procedure can be satisfactorily brought to conclusion on the paths P_u and P_v, leaving no transmitted node requirements unsatisfied.) If these paths do not intersect or if the paths encounter a loop before meeting, then they transmit requirements to the node(s) where they meet the loop(s), and the flow for the loop arcs that accommodate these requirements are generated by the procedures of references 8 and 15. If these paths intersect before encountering a loop, then they transmit a requirement to their intersecting node (the sum of the two requirements generated by the nonintersecting portions of the paths), and flows are then determined exactly as before in the intersecting portions of their paths until reaching the loop, whereupon the procedures of references 8 and 15 may be used. (A zero imbalance may be transmitted to the first node of the intersection, in which case all subsequent arcs receive 0 flow and no further determination is necessary. This is the situation that occurs in ordinary network problems.) Thus, all alternatives are resolved, leaving no 'dangling' node requirements, and the basis representation of (u, v) is completed by the specified assignment of flows to the arcs of P_u and P_v. (From the fact that a basis representation is unique, it follows that the one determined is the one sought.)

The preceding construction of the basis representation is particularly useful to determine the outgoing arc in the primal simplex method. For explicitness in the present context, the outgoing arc is identified quite simply by considering the flow w_{ij} thus generated for each arc (i, j), and computing the upper bound value α that satisfies $x_{ij} - \alpha w_{ij} \geqq 0$ (i.e., $\alpha \leqq x_{ij}/w_{ij}$ for w_{ij} positive). The arc that provides the most restrictive upper bound α^* for α becomes the outgoing arc, and the new flow value of each of the arcs (i, j) in the basis representation is

$$x_{ij} - \alpha^* w_{ij},$$

with the flow value x_{uv} of the incoming arc itself equal to α^*. For the capacitated problem α represents the value to be assigned either to x_{uv} or $u_{uv} - x_{uv}$, depending on whether currently $x_{uv} = 0$ or $x_{uv} = u_{uv}$. In the latter case, the representation sought is of the negative of the arc (u, v), giving requirements of $-a_{u,uv}$ and $-a_{v,uv}$ at nodes u and v. For this case, α is also limited by the inequalities $u_{ij} \geqq x_{ij} - \alpha w_{ij}$ for all arcs in the basis representation $\alpha \leqq u_{uv}$. The new flow values are determined exactly as before for the variables x_{ij}, but if $\alpha^* = u_{uv}$ then no arc enters or leaves the basis, and x_{uv} is simply set equal to $\alpha^* = u_{uv}$ or $u_{uv} - \alpha^* = 0$, as appropriate.

Updating the Basis

A crucial concern of the extended API method is to impart the rooted-loop orientation to the updated basis by appropriate reindexing, thus making it possible to take advantage of this orientation in the manner described in the preceding and following sections. The basis exchange step modifies the composition of the quasi trees in the basis in any of a variety of ways, depending on the relation of the incoming and outgoing arcs to the current basis and to each other. For example, the endpoints of the incoming arc can lie in two separate quasi trees, or in the same quasi tree, and may at the same time lie either on a loop or a 'tributary' of a quasi tree, creating different possible configurations in each case. The outgoing arc may disconnect two quasi trees that have been joined by the incoming arc, destroy a previously existing loop in one of these quasi trees, disconnect a newly created loop from an old loop, break one or the other of a new and old loop presently connected, or break overlapping loops on any of three possible chains. In spite of this proliferation of cases (there are something like 17 of them, depending on which configurations are identified as 'equivalent'), it is possible to accommodate every alternative by a single, easily stated rule. This rule automatically generates the appropriate indexing scheme without reference to the manner in which loops are created or broken, or to their relation to each other. If it were not a matter of convenience to do so (for reasons to be discussed later) there would be no need to identify which arcs lie in which loops, or even to identify the presence or absence of loops, in order to determine the updated indexing that permits the basis exchange paths and other updating calculations to be carried out. To describe this rule, let P denote one of the backward paths from the incoming arc (u, v) that contains the outgoing arc. (Either or both of P_u and P_v will contain this arc, but only one of these paths is selected.) Define the 'upper path' P^* corresponding to P to be the portion of P that lies between the incoming and outgoing arcs. (P^* contains at least one node, but may contain no arcs if the incoming and outgoing arcs are adjacent.) Then the updating rule that achieves an appropriate reorientation of the basic arcs for all of the cases indicated may be stated as follows.

REMARK 2. *If the basis for a generalized network problem has a rooted-loop orientation, then the new basis after the basis exchange step will also have a rooted-loop orientation by the following steps:*

1. *Reverse the orientation of all arcs in P^*.*
2. *Orient the incoming arc so that it 'begins' this redirected path; i.e., the endpoint of the incoming arc that is not on P^* becomes the predecessor of the endpoint which is.*

Proof. Consider the network that consists of the current basis after deleting the outgoing arc but before adding the incoming arc. The connected subnetwork consisting of all arcs that can be reached from nodes of P^* is a tree, since the removal of both the incoming and outgoing arcs implies that this subnetwork can contain no loops. (This may readily be verified by an examination of cases.) Moreover, this tree is an arborescence due to the fact that every subtree of a rooted-loop structure is an arborescence. The root of this arborescence is easily identified as the node of the upper path that is an endpoint of the outgoing arc since the removal of this arc leaves the indicated endpoint without a predecessor, a condition that only the root node satisfies. A characteristic of an arborescence is that reversing the direction of every arc lying on some path from the root node creates a new arborescence, whose root is the opposite endpoint of the reversed path. Thus, the reversal of P^* creates an arborescence rooted at the node that is an endpoint of the incoming arc. Now there are two possibilities: either the opposite endpoint of the incoming arc lies in this same arborescence or it does not. If it does, the addition of the incoming arc creates a loop, and an arc that joins any node of an arborescence to the root node, directed toward the latter, creates a rooted-loop quasi tree, as desired. In the other case, the incoming arc grafts the arborescence to a disjoint network that, by the known structure of the basis and the previously established orientation of the arcs, must be a rooted-loop quasi tree. But since the arborescence is connected by its root node, using an arc directed from the quasi tree to the arborescence, the rooted-loop structure is maintained.

The simplicity of the operation described in Remark 2 makes it highly attractive from a computational standpoint. It should be noted that the process of generating the new basis structure can be viewed in terms of deleting and adding arcs. An arc reversal consists of deleting the arc in one direction and then adding it back in the reverse direction. Denoting an arc with an induced orientation from i to j by $[i, j]$—which may correspond either to the arc (i, j) or (j, i)—the rules for deleting and adding arcs may be stated as follows.

To delete $[i, j]$: *Case 1*: j is the eldest son of i: Name j's next younger brother as the (new) eldest son of i. *Case 2:* j is not the eldest son of i: Identify j's next older and next younger brother, and name the latter to be the (new) next younger brother of the former.

To add arc $[i, j]$: Name i as the predecessor of j, name the eldest son of i to be j's next younger brother and name j to be the new eldest son of i.

In carrying out the foregoing operations, the use of a standard dummy index (e.g., 0) to name a nonexistent eldest son or next younger brother makes it unnecessary to check for exceptional cases. It is also unnecessary to modify (or 'clear') the index labels that name j's predecessor and next younger brother when deleting $[i, j]$ since j will always receive appropriately reassigned labels in the process of updating the basis network.

Updating Node Potentials and Integrating Computations

The determination of the node potentials (dual evaluators) π_i, $i \in N$, so that the marginal cost $c_{ij} - \pi_i a_{i,ij} - \pi_j a_{j,ij}$ for each basic arc (i, j) in the network equals 0, can be performed efficiently using the guidelines of references 8 and 15. The principal observations to be made in the present setting are two. First, updated calculations need only be made for descendents of the first endpoint of the incoming arc in the updated basis. (Note that this consists of the nodes of P^* and their descendants after the updating step.) Thus, if the incoming arc does not create a new loop (i.e., connects two previously disjoint quasi trees) or if a newly created loop is broken by the out-going arc, then the new node potentials are straightforwardly computed—using the existing potential of the first node of the incoming arc—by fanning out through the arborescence that is identified in the proof of Remark 2. In such a fanning out process, as soon as the potential for a node's predecessor has been updated, the potential for the node itself is immediately updated; e.g., if π_j is known for arc (i, j) oriented as $[j, i]$, then

$$\pi_i = (-\pi_j a_{j,ij} + c_{ij})/a_{i,ij} .$$

None of the other node potentials in the network need to be examined. On the other hand, if the incoming arc creates a new loop that survives the deletion of the outgoing arc—a loop that is instantly determined by reference to P_u and P_v—then the new node potentials are first computed for this loop (following references 8 and 15) after which the remaining potentials to be updated are determined by fanning out to the successors of these nodes.

The second observation to be made is that the updating of the node potentials can be coordinated with the updating of the flow values over the portion of the network where both of these introduce change. Similarly, the determination of the 'loop factor' for a new loop can be carried out simultaneously with the reverse trace of P_u and P_v to identify the basis representation of the incoming arc. In this connection, the use of stored loop factors to accelerate calculations involving pre existing loops (as discussed in references 6 and 8), makes it desirable to keep track of loops currently in the basis. Because of the automatic manner in which the trace of the backward paths identifies a current loop (or loops) of interest, and also pinpoints the identity of a newly created loop, a particularly simple and efficient scheme for making use of loop factors is possible. When a loop is treated (initially or in a basis exchange step), the loop factor is recorded (indexed) by the nodes in the loop. That is, this number is attached to each node in the loop (using a node length array). As soon as a backward path identifies a loop, the loop factor may be retrieved from any node in the loop. When a loop is destroyed, there is no need to locate or erase the loop factor in the node list since the process of loop detection by backward paths restricts attention to nodes whose loop factors are current.

Thus, to conclude, while the underlying structures and processes of the generalized network problem are somewhat more complex than those of the ordinary network problem, the extended API method conveniently accommodates this additional complexity by means of simple rules that enable the auxiliary computational processes (e.g., those of reference 8) to be implemented effectively.

REFERENCES

1. EGON BALAS, "The Dual Method for the Generalized Transportation Problem," *Management Sci.* **12,** 555–568 (1966).
2. ——— AND P. L. IVANESCU (HAMMER), "On the Generalized Transportation Problem," *Management Sci.* **11,** 188–202 (1964).
3. R. S. BARR, F. GLOVER, AND D. KLINGMAN. "An Improved Version of the Out-of-Kilter Method and a Comparative Study of Computer Codes," to appear in *Mathematical Programming.*
4. A. CHARNES AND W. W. COOPER, *Management Models and Industrial Applications of Linear Programming,* Vols. I–II, Wiley, New York, 1961.
5. G. B. DANTZIG, *Linear Programming and Extensions,* Princeton University Press, Princeton, N. J., 1963.
6. KURT EISEMANN, "The Generalized Stepping Method for the Machine Loading Model," *Management Sci.* **11,** 154–177 (1964).
7. FRED GLOVER AND D. KLINGMAN, "On the Equivalence of Some Generalized Network Problems to Pure Network Problems," to appear in *Mathematical Programming,* **4** (1973).
8. ——— AND ———, "A Note on Computational Simplifications in Solving Generalized Transportation Problems," *Trans. Sci.* **7,** 351–361 (1973).
9. ———, D. KARNEY, D. KLINGMAN, AND A. NAPIER, "A Computational Study on Start Procedures, Basis Change Criteria, and Solution Algorithms for Transportation Problems," to appear in *Management Sci.*
10. ———, D. KARNEY, AND D. KLINGMAN. "The Augmented Predecessor Index Method for Locating Stepping Stone Paths and Assigning Dual Prices in Distribution Problems," *Trans. Sci.,* **6,** 171–180 (1972).
11. W. S. JEWELL, "Optimal Flow Through Network with Gains," *Opns. Res.* **10,** 476–499 (1962).
12. ELLIS JOHNSON, "Networks and Basic Solutions," *Opns. Res.* **14,** 89–95 (1966).
13. D. E. KNUTH, *The Art of Computer Programming: Fundamental Algorithms,* Vol. 1, Addison-Wesley, Reading, Mass., 1968.
14. JANICE LOURIE, "Topology and Computation of the Generalized Transportation Problem," *Management Sci.* **11,** 177–187 (1964).
15. J. F. MAURRAS, "Optimization of the Flow Through Networks with Gains," *Mathematical Programming* **3,** 135–145 (1972).
16. V. SRINIVASAN AND G. L. THOMPSON, "Benefit Cost Analysis of Coding Techniques for the Primal Transportation Algorithm," to appear in *JACM.*

(Received, February 1973)