AD-763 382

EXTENSIONS OF THE AUGMENTED PRE-
DECESSOR INDEX METHOD TO GENERALIZED
NETWORK PROBLEMS

Fred Glover, et al

Texas University
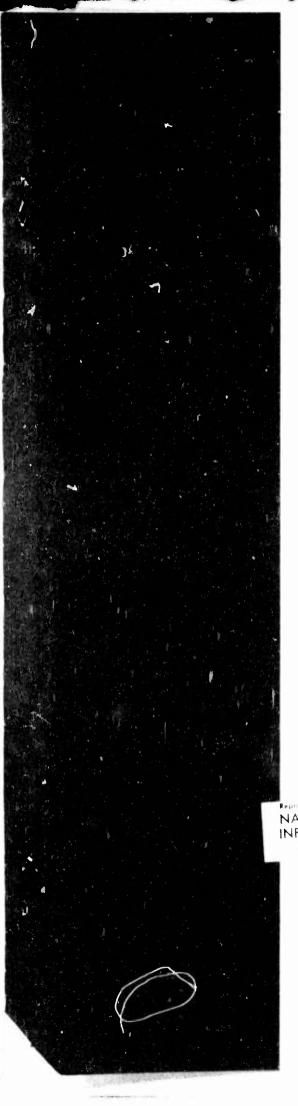
Prepared for:

Office of Naval Research

February 1973

EXTENSIONS OF THE AUGMENTED
PREDECESSOR INDEX METHOD TO
GENERALIZED NETWORK PROBLEMS

by

F. Glover*

D. Klingman**

J. Stutz***

CENTER FOR CYBERNETIC STUDIES

The University of Texas
Austin, Texas 78712

EXTENSIONS OF THE AUGMENTED
PREDECESSOR INDEX METHOD TO
GENERALIZED NETWORK PROBLEMS

by

F. Glover*

D. Klingman**

J. Stutz***

February 1973

* Professor of Management Science at the University of Colorado,
    Boulder, Colorado
** Associate Professor of Operations Research at the University of Texas
    Austin, Texas
*** Assistant Professor of Operations Research at the University of Texas
    Austin, Texas

CENTER FOR CYBERNETIC STUDIES

A. Charnes, Director
Business-Economics Building, 512
The University of Texas
Austin, Texas   78712

iii

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Networks						
Generalized Network						
Generalized Transportation						
Sequential Linked Lists						

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1 ORIGINATING ACTIVITY (Corporate author)	2a. REPORT SECURITY CLASSIFICATION
Center for Cybernetic Studies University of Texas at Austin	Unclassified
	2b. GROUP

3 REPORT TITLE

Extensions of the Augmented Predecessor Index Method to Generalized
Network Problems

4 DESCRIPTIVE NOTES *(Type of report and inclusive dates)*

5 AUTHOR(S) *(First name, middle initial, last name)*

F. Glover   J. Stutz
D. Klingman

6 REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
February 1973	~~17~~ 2C	16

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
NR-047-021	Center for Cybernetic Studies
b. PROJECT NO	Research Report CS 124
N00014-67-A-0126-0008	
c.	9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*
d. N00014-67-A-0126-0009	

10 DISTRIBUTION STATEMENT

This document has been approved for public release and sale; its
distribution is unlimited.

11 SUPPLEMENTARY NOTES	12 SPONSORING MILITARY ACTIVITY
	Office of Naval Reasearch (Clde 434) Washington, D.C.

13 ABSTRACT

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.

Abstract

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.

1.0 INTRODUCTION

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 x 100

transportation problems in 1.4 seconds and 1000 x 1000 transportation problems

in 15 seconds 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, 5, 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 of [3, 9, 14, 15, 16] which show that special purpose transportation

and network codes (utilizing the updating procedures which 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 con-

siderations relevant to computer implementation.

2.0 THE GENERALIZED NETWORK PROBLEM

The generalized network problem may be defined as

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

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

$$x_{ij} \geq 0, \; (i,j) \in A \tag{3}$$

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 which 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 which 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 [5, p. 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 non-negativity inequalities (3) are often supplemented by upper bound inequalities of the form $u_{ij} \geq x_{ij}$, 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.

3.0 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 API method.

The triple-label scheme orients the tree so that it is in fact an "arborescence"; that is, for some single node which 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 predeseccor). 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 which 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 relationships developed in Section 4.)

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 which 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 counter-clockwise, 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 trees (that are identified by suppressing 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.

4.0 THE EXTENDED API METHOD

4.1 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 which 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 identifies 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 which 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 which represent a linearly independent subsystem of the full basis, containing as many variables (arcs) as equations (nodes). To provide a more useful justifica-

tion, we will show how to constructively generate 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 conslusion 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 accomodate these requirements are generated by the procedures of [8, 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 [8, 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 which 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 F_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 are (i,j), and computing the upper bound value α that satisfies $x_{ij} - \alpha w_{ij} \geq 0$ (i.e., $\alpha \leq x_{ij}/w_{ij}$ for w_{ij} positive). The arc which provides the most restrictive upper bound $\alpha *$ 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 $\alpha *$. 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 $u_{ij} \geq x_{ij} - \alpha w_{ij}$ for all arcs in the basis representation $\alpha \leq u_{uv}$. 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.

4.2 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 Sections 4.1 and 4.3. 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 which 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 which 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 orien-
tation, 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 which 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 sub-
network 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
which 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 which 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 which 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 which, 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.

4.3 Updating Node Potentials and Integrating Computations

The determination of the node potentials (dual evaluators) π_i, $i \epsilon N$, 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 [8,15]. 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 outgoing 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 which is instantly determined by reference to P_u and P_v - then the new node potentials are first computed for this loop (following [8,15]) 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 [6,8]), 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 created (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 oy 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 [8]) to be implemented effectively.

References

1. Balas, Egon. "The Dual Method for the Generalized Transportation Problem," *Management Science* 12, 7 (1966), 555-568.

2. Balas, Egon and P.L. Ivanescu (Hammer). "On the Generalized Transportation Problem," *Management Science* 11, 1 (1964), 188-202.

3. Barr, R.S., F. Glover, and D. Klingman. "An Improved Version of the Out-of-Kilter Method and a Comparative Study of Computer Codes," C.S. #102, Center For Cybernetic Studies, *University* of Texas at Austin, 1972.

4. Charnes, A. and W.W. Cooper. *Management Models and Industrial Applications of Linear Programming*, Vol. I-II, New York: John Wiley and Sons, Inc., 1961.

5. Dantzig, G.B. *Linear Programming and Extensions* Princeton, N.J.: Princeton University Press, 1963.

6. Eisemann, Kurt. "The Generalized Stepping Method for the Machine Loading Model," *Management Science* 11,1 (1964), 154-177.

7. Glover, Fred and D. Klingman. 'On the Equivalence of Some Generalized Network Problems to Pure Network Problems," To appear in *Mathematical Programming*.

8. Glover, Fred and D. Klingman. "A Note On Computational Simplications In Solving *Generalized Transportation Problems*." To appear in *Transportation Science*.

9. Glover, Fred, D. Karney, D. Klingman, and A. Napier. "A Computation Study on Start Procedures, Basis Change Criteria, and Solution Algorithms for Transportation Problems," To appear in *Management Science*.

10. Glover, Fred, D. Karney, and D. Klingman. "The Augmented Predecessor Index Method for Locating Stepping Stone Paths and Assigning Dual Prices in Distribution Problems," *Transportation Science*, 6, 1(1972), 171-180.

11. Jewell, W.S. "Optimal Flow Through Network with Gains," *Operations Research* 10,4 (1962), 476-499.

12. Johnson, Ellis. "Networks and Basic Solutions," *Operations Research*, 14,4 (1966), 89-95.

13. Knuth, D.E. *The Art of Computer Programming*: Vol. 1, Fundamental Algorithms, Reading, Mass.: Addison-Wesley, 1968.

14. Lourie, Janice. "Topology and Computation of the Generalized Transportation Problem," _Management Science_ 11,1 (1964), 177–187.

15. Maurras, J.F. "Optimization of the Flow Through Networks With Gains," _Mathematical Programming_ 3,2 (1972), 135–145.

16. Srinivasan, V. and G.L. Thompson, "Benefit Cost Analysis of Coding Techniques for the Primal Transportation Algorithm," To appear in _ACM_.