Quaestiones Informaticae

Vol. 2 No. 3 September, 1983

Quaestiones Informaticae

An official publication of the Computer Society of South Africa and of the South African Institute of Computer Scientists

'n Amptelike tydskrif van die Rekenaarvereniging van Suid-Afrika en van die Suid-Afrikaanse Instituut van Rekenaarwetenskaplikes

Editor: Prof G. Wiechers
Department of Computer Science and Information Systems
University of South Africa
P.O. Box 392, Pretoria 0001

Subscriptions

Annual subscriptions are as follows:

	SA	US	UK
Individuals	R6	$7	£3,0
Institutions	R12	$14	£6,0

Circulation manager
Mr E Anderssen
Department of Computer Science
Rand Afrikaanse Universiteit
P O Box 524
Johannesburg 2000
Tel.: (011) 726-5000

Quaestiones Informaticae is prepared for publication by Thomson Publications South Africa (Pty) Ltd for the Computer Society of South Africa and the South African Institute of Computer Scientists.

Data Structures for Generalized Network Algorithms

Desmond C. Currin
National Research Institute for Mathematical Sciences of the CSIR, Pretoria

Abstract
Generalized network problems involve the optimization of a flow through a network. In contrast to normal networks, generalized networks include multipliers which alter the flow as it passes through the arcs. This enables the modelling of changes in flow caused by factors such as interest rates, as well as by processes such as energy conversion.

Generalized network problems can be solved by using the simplex algorithm. However, if the simplex bases are represented as forests of quasi-trees, connected graphs with a single cycle, a far more efficient algorithm can be obtained. Matrix multiplications required by the simplex algorithm can be interpreted as traversals of the quasi-trees, and consequently many of the arithmetic operations can be replaced by logical operations. This increases the numerical stability of the algorithm.

In this paper we outline this approach and discuss some of the data structures that can be used in its implementation.

CR Categories and Subject Descriptors : G.1.6 Numerical Analysis: Optimization — Linear Programming; G.2.2 Discrete Mathematics : Graph Theory — Network Problems; E.1 Data Structures — Graphs and Trees
General Terms: Algorithms
Additional Key Words and Phrases: generalized networks, network optimization, quasi-trees

1. Introduction

Generalized network models include the assignment, transportation and transshipment problems. The major advantage over these models is that the introduction of multipliers permits the modelling of problems where material appreciates, depreciates or changes from one form to another. The utility of generalized networks is illustrated by the many applications [1,2,3].

A *generalized network* model consists of a set of n nodes N and a set of arcs A. The problem is to find a flow x_k ($k \in A$) along each of the arcs such that:

(a) it is of minimum cost (i.e. it minimizes $\Sigma c_k x_k$); (1)

(b) it satisfies the capacity constraints placed on the arcs (i.e. $\ell_k \leq c_k \leq w_k$): and (2)

(c) it is sufficient to meet certain demands d_i ($i \in N$) at each of the nodes. When d_i is negative, this is interpreted as a supply of $(-d_i)$ units available at node i.

For generalized networks this last requirement may be formulated as

$$\underset{\text{Arcs leading to node i}}{\Sigma\, m_k x_k} - \underset{\text{Arcs leading from node i}}{\Sigma\, x_k} = d_i \qquad (3)$$

The coefficients m_k are the *multipliers,* and they indicate by which factor the flow will increase (or decrease) as it passes along the arc. The above equation states that the total flow into node i minus the total flow from the node should equal the demand, or the negative of the supply, at that node.

The algorithm used to solve these models is derived from the simplex algorithm for linear programs, but, because of the special structure of generalized networks, the solution can be obtained thirty to fifty times faster than by means of a comparably sized linear program [4].

Past experience with this and a similar problem, namely the transshipment problem, has shown that the speed of the algorithm depends on the data structures used in its implementation.

2. Linear Programs and the Simplex Algorithm

The generalized network problem, as formulated above, corresponds to the capacitated linear program

$$\begin{aligned} &\text{Min} && c\,x \\ &\text{subject to} \\ &N\,x = d && (4) \\ &\ell \leq x \leq w \end{aligned}$$

The coefficient matrix N has one column corresponding to each arc in the nework. Each column has at most two non-zero elements; if k = (i,j) is an arc leading from node i to node j, then $n_{ik} = -1$ and $n_{jk} = m_k$; and if k = (i,i) is a loop on node i, then $n_{ik} = m_k - 1$.

The *simplex algorithm,* which is used to solve the linear program, maintains a set of n *basic* variables. The non-basic variables are taken to be at either their lower limit ℓ_k or at their upper limit w_k. Consequently, the values of the n basic variables are determined by the n linear equations in [4].

At each iteration the simplex algorithm determines whether movement of one of the non-basic variables x_k from its lower or upper limit will reduce the total cost. If there is no such variable, the current solution is optimal. Otherwise, the algorithm determines how far that non-basic variable can move before it, or one of the basic variables x_r, reaches its upper or lower limit.

In the former case the non-basic variable x_k goes to its opposite limit, and in the latter case the basic variable x_r becomes non-basic while the non-basic variable x_k becomes basic. This is referred to as a *pivot.* The algorithm pivots until an optimal solution is obtained.

In brief the simplex algorithm employs three major steps:

(a) firstly it establishes an initial basis;

(b) in subsequent iterations it determines whether a non-basic variable is eligible for entry to the basis; and if so,

(c) it computes the effect of changing the value of that variable on the variables that are currently basic.

In this paper we will concentrate on step (b) and show how this can be efficiently implemented for generalized network models. The simplifications obtained for this step are comparable to simplifications that can be obtained in the other two steps.

3. The Structure of the Basis and the Effects of a Pivot

It can be shown that the arcs, associated with the basic variables in a generalized network, form a forest of graphs, each having a single *cycle* and a number of attached sub-trees. These graphs are called *quasi-trees*[5].

As an example consider the network in Figure 1. This shows the basic arcs as solid lines and the non-basic arcs as dotted lines. The basic arcs form two separate quasi-trees.

When the algorithm pivots, either of the following may happen:

(a) the non-basic arc selected may move from its current level

to its opposite level without any of the basic arcs becoming non-basic; or

(b) the non-basic arc replaces one of the basic arcs in the basis.

In the former case the structure of the basis is unaffected, but in the latter case a number of interesting occurrences may take place. When the basic arc leaves the basis, it either splits the cycle of a quasi-tree, transforming it into a tree as in Figure 2, or it disconnects a sub-tree from a quasi-tree as in Figure 3. In either case we are left with one tree and a number of quasi-trees.

The arc entering the basis changes this tree into a quasi-tree, either by forming a new cycle as in Figure 4, or by linking it onto an existing quasi-tree as in Figure 5.

4. The Predecessor Array

The basis can be represented by means of a *predecessor* array p(.). For a node on the cycle of quasi-tree this indicates the previous node encountered when circling the cycle in a clockwise direction. For a node in a tree attached to a cycle, the predecessor indicates the father of that node.

In Figures 1-5, the predecessors of each node is indicated by means of the arrows drawn on the basic arcs.

These figures also show that only those arcs on the path joining the incoming arc to the outgoing arc alter during a pivot and these changes merely involve a reversal of the direction of the arrows. Updating of the predecessor array during each pivot is thus very simple.

5. Finding a Variable Eligible for Entry to the Basis

In this section we consider the problem of finding a variable that will reduce the total cost when introduced into the basis.

Most books on linear programming contain an equation, which, apart from changes in notation, reads

$$\bar{c}_k = c_k - c_B B^{-1} N_k \qquad (5)$$

The quantity $\bar{c}_k$ is the *reduced cost* and indicates how the total cost will alter when the value of x_k is changed. If $\bar{c}_k$ is negative, then the total cost will decrease when x_k is increased. Similarly, if $\bar{c}_k$ is positive, we should decrease x_k in order to reduce the total cost. Consequently, the non-basic variables eligible for entry to the basis are those which are at their lower limit and have $\bar{c}_k$ negative, as well as those which are at their upper limit and have $\bar{c}_k$ positive.

Expression (5) involves

(a) the cost c_k associated with the variable x_k;

(b) the costs c_B associated with all basic variables;

(c) the matrix B which comprises the columns of N corresponding to the basic variables; and

(d) N_k which is the k'th column of N.

Efficient implementations of the simplex algorithm normally store B^{-1} as a product of elementary matrices $B^{-1} = E_1 E_2 \ldots E_\ell$ and then compute the quantity $c_B B^{-1} N_k$ by using matrix multiplications.

For generalized networks an approach without matrix multiplications is used. Firstly, we write $c_B B^{-1} N_k$ as uN_k by letting $u = c_B B^{-1}$. The values u are the *node potentials*. We then observe that for an arc k = (i,j) leading from node i to j, N has at most two non-zero values, $n_{ik} = -1$ and $n_{jk} = m_k$. Consequently, the formula for the reduced cost simplifies to $\bar{c}_k = c_k + u_i - m_k u_j$. For a loop on node i, k = (i,i), the formula reduces to $\bar{c}_k = c_k + (1 - m_k)u_i$.

It is thus simple to compute the reduced costs — provided that we know the values of the node potentials.

6. Computing the Node Potentials

Since it is known that the reduced costs of all basic arcs are zero, we can, in principle, determine the node potentials by setting the reduced costs of the n basic arcs to zero and solving the resulting n linear equations.

The problem can be simplified by observing that we can com-

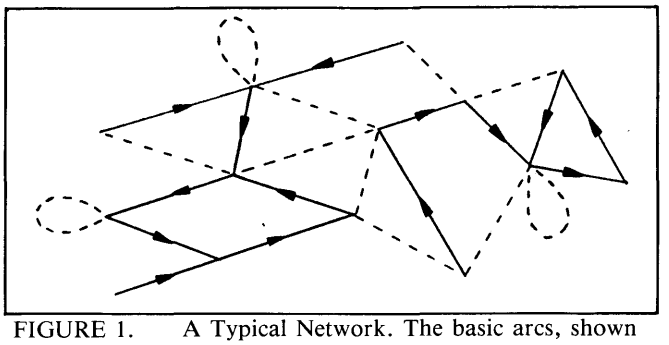

FIGURE 1. A Typical Network. The basic arcs, shown as solid lines, form two quasi-trees.

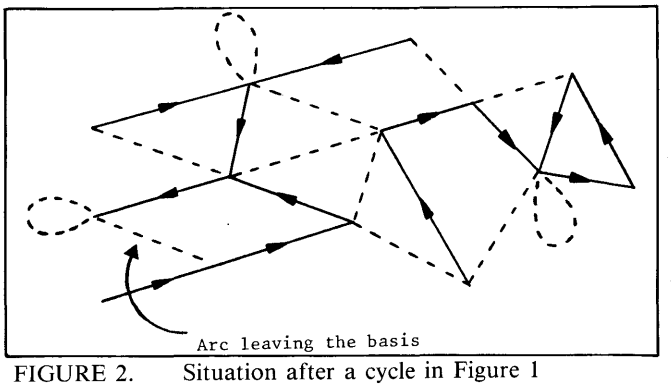

FIGURE 2. Situation after a cycle in Figure 1 has been split.

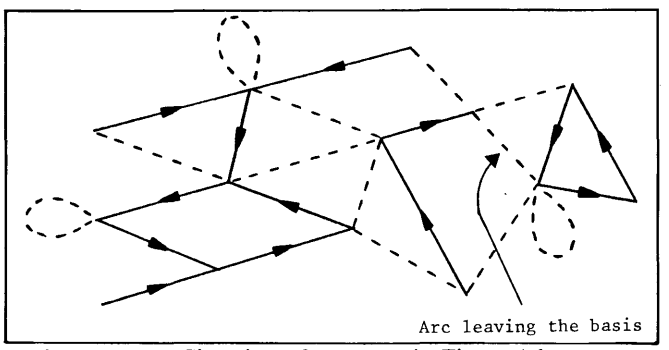

FIGURE 3. Situation after a tree in Figure 1 has been disconnected from a quasi-tree.

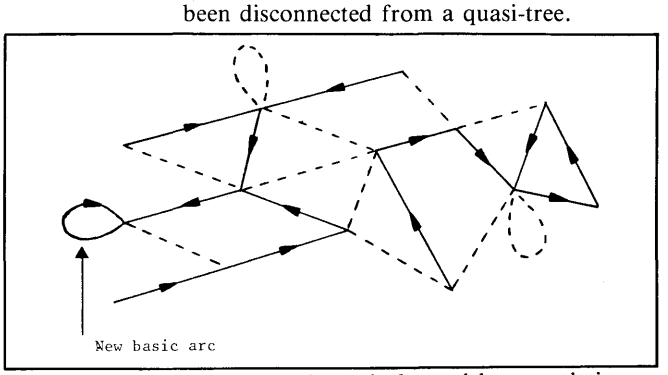

FIGURE 4. A new quasi-tree is formed by completing a cycle in Figure 2.

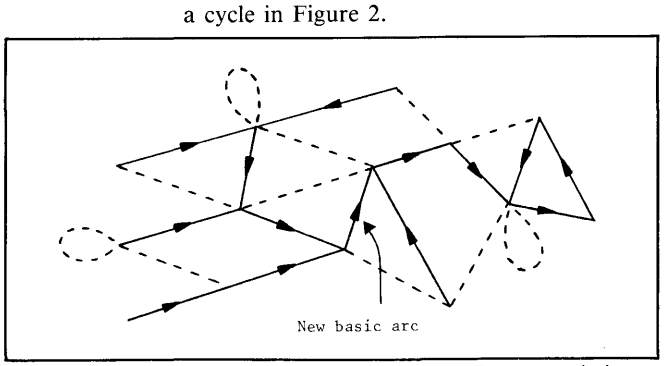

FIGURE 5. The tree has been connected to an existing quasi-tree in Figure 2.

pute the potential of a node in one of the sub-trees, if we know the potential of its predecessor (since the reduced cost of the intervening arc is zero). The potentials of the nodes in the sub-trees can thus be established by traversing the sub-trees in *pre-order* — we visit and compute the potential for a node after we have visited its predecessor.

The problem of determining the potentials of nodes lying on a cycle is slightly more complicated. If the potential of some node i is u_i we can circle the loop and compute the potentials of the other nodes j on that cycle *in terms of* u_i. These potentials will all be a linear function $a_j + b_j u_i$ of the initial potential u_i. During a first pass around the cycle we establish the values for a_j and b_j.

The last node ℓ encountered before returning to node i will relate $u_\ell = a_\ell + b_\ell u_i$ to u_i. This enables us to solve for u_i.

On a second pass around the cycle we compute the potentials for the other nodes j.

Not all the node potentials change at each pivot. When a tree is linked onto an existing quasi-tree, only the nodes on that tree need to be recomputed, and when a new cycle is formed all the potentials on the new quasi-tree need to be evaluated.

7. The Double-Linked Tree and the Thread Successor Array

In order to compute the node potentials we must be able to
(a) circle the loop of a quasi-tree; and
(b) traverse the sub-trees in pre-order.

The former can be done by using the predecessor array. While it is possible to implement the latter using only predecessor information, it is rather inefficient and for this reason a new data structure is normally introduced.

One method [6] that can be employed is storing for each node i the set S_i of its immediate neighbours in the quasi-tree. For each basic arc (i,j) we have $i \in S_j$ and $j \in S_i$. Consequently, this is referred to as a *double-linked tree.*

For a node in a sub-tree its sons are the nodes $S_i - \{p(i)\}$. By using this information and a stack-based procedure, we can traverse the sub-tree in pre-order.

During each pivot one arc becomes basic while another arc becomes non-basic. Consequently, two elements are removed from the sets while two are introduced. Minimal updating is thus required.

One disadvantage of this method is that 4n words of storage are required for its implementation. An alternative [7] is to store the *thread successor* array, which uses only n words of storage. The thread successor of a node is the next node to be visited when traversing the tree in pre-order.

The thread successor array is more difficult to implement, since the elements can change drastically from one iteration to the next. Details for updating this structure can be found in Currin [6]. Computational experience shows that, although the thread successor method is slower than the double-linked tree method, the difference is not large and the thread successor method can be employed profitably when storage is at a premium.

8. Conclusions

Solving a generalized network model by using a specialized computer code is far more efficient than using a standard linear programming code.

Because of the special structure, the basis can be stored as a graph (a forest of quasi-trees). In contrast, a linear programming code has to store the inverse basis B^{-1}. Similarly, calculations that are normally done using matrix multiplications can be replaced by simple scalar operations performed while traversing the quasi-trees.

The net result is that many arithmetic computations are omitted. Although there is an increase in the number of logical operations performed, these can be executed faster and more precisely than arithmetic operations. This results in the computer code being more efficient and numerically stable.

References

[1.] Mathematical Programming Study, 1981, vol 15.

[2.] Glover, F. & Klingman D. 1975. Real world applications of network related problems and breakthroughs in solving them efficiently. ACM Trans. Math. Software, vol. 1, no. 1.

[3.] Glover, F & Klingman, D. 1977. Network applications in industry and government. AIIE Trans., pp 363-376.

[4.] Glover, F., Hultz, J., Klingman, D. & Stutz, J. 1978. Generalized networds: A fundamental computer based planning tool. Man. Sci., vol. 24, no. 12.

[5.] Maurras, J. 1972. Optimization of the flow through networks with gains. Math. Prog., vol 15, pp 291-314.

[6.] Currin, D. 1983. A comparative evaluation of algorithms for generalized network problems. TWISK 289, CSIR, 141p.

[7.] Kennington, J. & Helgason, R. 1980. Algorithms for Network Programming, Wiley, 291p.

Notes for Contributors

The purpose of this Journal will be to publish original papers in any field of computing. Papers submitted may be research articles, review articles, exploratory articles of general interest to readers of the Journal. The preferred languages of the Journal will be the congress languages of IFIP although papers in other languages will not be precluded.

Manuscripts should be submitted in triplicate to: Prof. G. Wiechers at:

Department of Computer Science
University of South Africa
P.O. Box 392
Pretoria 0001
South Africa

Form of manuscript

Manuscripts should be in double-space typing on one side only of sheets of A4 size with wide margins. The original ribbon copy of the typed manuscript should be submitted. Authors should write concisely.

The first page should include the article title (which should be brief), the author's name, and the affiliation and address. Each paper must be accompanied by a summary of less than 200 words which will be printed immediately below the title at the beginning of the paper, together with an appropriate key word list and a list of relevant Computing Review categories.

Tables and figures

Illustrations and tables should not be included in the text, although the author should indicate the desired location of each in the printed text. Tables should be typed on separate sheets and should be numbered consecutively and titled.

Illustrations should also be supplied on separate sheets, and each should be clearly identified on the back in pencil with the Author's name and figure number. Original line drawings (not photoprints) should be submitted and should include all relevant details. Drawings, etc., should be submitted and should include all relevant details. Drawings, etc., should be about twice the final size required and lettering must be clear and "open" and sufficiently large to permit the necessary reduction of size in block-making.

Where photographs are submitted, glossy bromide prints are required. If words or numbers are to appear on a photograph, two prints should be sent, the lettering being clearly indicated on one print only. Computer programs or output should be given on clear original printouts and preferably not on lined paper so that they can be reproduced photographically.

Figure legends should be typed on a separate sheet and placed at the end of the manuscript.

Symbols

Mathematical and other symbols may be either handwritten or typewritten. Greek letters and unusual symbols should be identified in the margin. Distinction should be made between capital and lower case letters between the letter O and zero; between the letter l, the number one and prime; between K and kappa.

References

References should be listed at the end of the manuscript in alphabetical order of author's name, and cited in the text by number in square brackets. Journal references should be arranged thus:

1. ASHCROFT, E. and MANNA, Z. (1972). The Translation of 'GOTO' Programs to 'WHILE' Programs, in *Proceedings of IFIP Congress 71*, North-Holland, Amsterdam, 250-255.
2. BÖHM, C. and JACOPINI, G. (1966). Flow Diagrams, Turing Machines and Languages with only Two Formation Rules, *Comm. ACM*, **9**, 366-371.
3. GINSBURG, S. (1966). *Mathematical Theory of context-free Languages*, McGraw Hill, New York.

Proofs and reprints

Galley proofs will be sent to the author to ensure that the papers have been correctly set up in type and not for the addition of new material or amendment of texts. Excessive alterations may have to be disallowed or the cost charged against the author. Corrected galley proofs, together with the original typescript, must be returned to the editor within three days to minimize the risk of the author's contribution having to be held over to a later issue.

Fifty reprints of each article will be supplied free of charge. Additional copies may be purchased on a reprint order form which will accompany the proofs.

Only original papers will be accepted, and copyright in published papers will be vested in the publisher.

Letters

A section of "Letters to the Editor" (each limited to about 500 words) will provide a forum for discussion of recent problems.

Hierdie notas is ook in Afrikaans verkrygbaar.

Quaestiones Informaticae

Contents/Inhoud

*Presented at the second South African Computer Symposium held on 28th and 29th October, 1981.

MILITHO