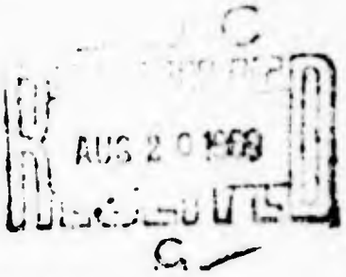
The George Washington University

LOGISTICS RESEARCH PROJECT

Contract N00014-67-A-0214

Task 0001, Project NR 047 001

OFFICE OF NAVAL RESEARCH



THIS DOCUMENT HAS BEEN APPROVED FOR PUBLIC RELEASE AND SALE; ITS DISTRIBUTION IS UNLIMITED

A DYNAMIC MULTI-COMMODITY, MULTI-MODE NETWORK FLOW MODEL

by

Donald J. Hunt
Erling F. Rosholdt

Serial T-225

30 June 1969

THE GEORGE WASHINGTON UNIVERSITY
Logistics Research Project

Contract N00014-67-A-0214
Task 0001, Project NR 047 001
Office of Naval Research

THE GEORGE WASHINGTON UNIVERSITY
Logistics Research Project

Abstract
of
Serial T-225

A DYNAMIC MULTI-COMMODITY, MULTI-MODE NETWORK FLOW MODEL

by

Donald J. Hunt
Erling F. Rosholdt

This paper describes a dynamic multi-commodity, multi-mode network flow model which permits time phasing of commodity load inputs and derives delivery schedules to the respective destinations over a time span of interest to the user. The model makes use of a time-expanded network. Methodology for time expanding a basic network is described and an algorithm for determining the commodity flow allocations is provided Implementation of the model has been made in PI-1 programming language for use with the IBM 360/50 computer. A number of innovative programming steps which make possible very efficient processing are described and computing experience with several different network problems is reported.

TABLE OF CONTENTS

THE GEORGE WASHINGTON UNIVERSITY
Logistics Research Project

A DYNAMIC MULTI-COMMODITY, MULTI-MODE NETWORK FLOW MODEL

by

Donald J Hunt
Erling F. Rosholdt

Introduction

In [1], a computationally feasible procedure for solving the multi-commodity maximum flow problem was described. The procedure described the use of a shortest path algorithm to select the column vector to enter the basis in the simplex method and thus avoided the storage problem incurred for the usually very large arc-chain incidence matrix used to formulate the problem as originally expressed by Ford and Fulkerson in 1958 [2].

The multi-commodity network flow model described in this paper has extended the procedure from [1] into a dynamic multi-commodity multi-mode model by coupling the procedure with a time-expanded network so as to permit the time-phasing of commodity load inputs and the derivation of delivery schedules to the respective destinations over a time span of interest to the user.

The methodology for expanding a basic network into a time-expanded network and the algorithm for determining the delivery schedules has been written in programming language PL-1 for use on an IBM 360/50 computer. A number of innovative programming steps which make possible very efficient processing are described and computing experience with several different network problems is reported.

1. Statement of the Problem

Consider the network $G = (N,A)$, where N is a finite set of points called nodes and A is a set of pairs of points from N called arcs. If each arc (x,y) in A is an ordered pair, the network G is called a directed network. If the arcs are unordered pairs then the network is an undirected network. The networks under specific consideration are directed but the methodology will apply also to undirected networks. Each arc has a capacity $c_{x,y} \geq 0$ and associated with each arc is a cost, namely, the traversal time $(a_{x,y})$. There is a set of sources $S \subseteq N$ which are originating points for commodities K in the basic network, and a set of sinks $T \subseteq N$ which are terminating points for commodities K in the basic network. A source-sink pairing is defined for each commodity. Each real source in the network is connected to a dummy source node and each real sink to a dummy sink node. To each of the arcs connecting dummy source to real source and dummy sink to real sink a cost of zero is assigned.

The time-expanded network of G over p time periods, $G(p)$ may be construed as one which is produced in layers or levels emanating from dummy source S_0 at successive time intervals. At each level there will be a replica of G with all paths or chains traceable from dummy source to dummy sink, whose total traversal time does not exceed the time span of interest. Holdover arcs of infinite capacity and cost = 1 are added at each of the sources in set S and holdover arcs of infinite capacity and cost = 0 are added at each of the sinks in set T.

We are asked to find a delivery schedule for a k-commodity flow in G which satisfied the capacity constraints on the arcs and the flow requirements and which minimizes the total cost.

Given:

$c^k_{(x,y)}$ = individual flow capacity of arc (x,y) for commodity K per unit time

C_m = mutual flow capacity of arc (x,y) per unit time

$a_{x,y}$ = traversal time of arc (x,y)

$F^k_{x,y;\tau}$ = amount of flow of commodity K leaving x along (x,y) at time τ and arriving at y at time $\tau + a(x,y)$

$F^k_{(x,x;\tau)}$ = holdover at x from τ to $\tau + 1$

$S_1, S_2, \ldots, S_k$ = sources for commodities 1,2,...,k

$T_1, T_2, \ldots, T_k$ = sinks for commodities 1,2,...,k

p = time periods in span of interest

$V^k_{(p)}$ = net flow out of source S_k or entering sink T_k during the p periods 0 to 1, 1 to 2, ..., p-1 to p

then the problem is:

$$\text{maximize } \sum_k V^k(p)$$

subject to the constraints

$$\sum_{T=0}^{p} \sum_{y \in N} [F^k(S_k,y;\tau) - F^k(y,s_k;\tau - a(y,s))] - V^k(p) = 0$$

$$\Sigma[F^k(x,y;\tau) - F^k(y,x;\tau - a(y,x))] = 0$$

$$x \neq s_k, T_k ; \tau = 0,1,\ldots,p$$

$$\sum_{\tau=0}^{p} \sum_{y \in N} [F^k(T,y;\tau) - F^k(y,T_k;\tau - a(y,T))] + V^k(p) = 0$$

$$0 \leq F^k(x,y;\tau) \leq C^k(x,y)$$

$$0 \leq \sum_k F^k(x,y;\tau) \leq C_m(x,y) .$$

In addition, the model permits the introduction of variable commodity loads at the commodity source at specified departure times and will attempt to satisfy desired delivery quantities at specified times at the commodity sink.

The term "requirements" in this model refers to a desired delivery quantity or upper bound rather than the usual meaning of an absolute demand at the sink.

A schematic diagram of the structure of a time-expanded network is given in Figure 1. Here,

d = shortest path from S_0 to T_0

p = time span of interest

τ = arrival/departure time at a node

u = p - d .

Not shown in Figure 1 is an internal network structure, indicated in Figure 2, which exists when for any arc there is both a mutual capacity, C_m, and an individual commodity capacity, $C^k_{x,y}$.

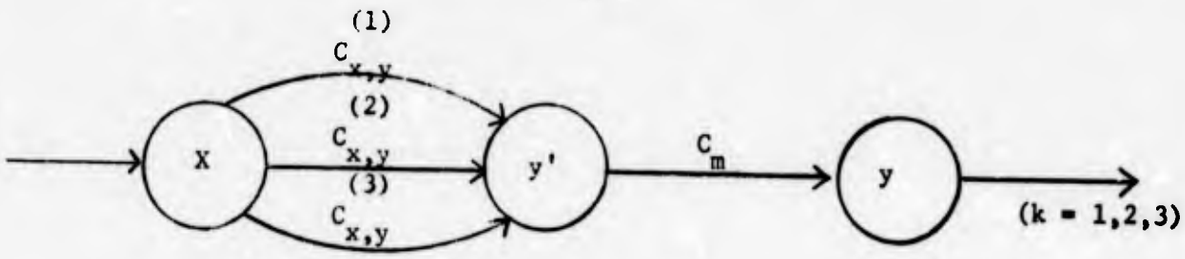


Figure 2

Individual and Mutual Capacity Arc Structure

General Program Description

1.1 Function

The program has the following functions:

a. to accept the time-expanded network as input;

b. to accept a list of commodities with their defining sources and sinks;

c. To accept a list of one or more available loads for each commodity, to be considered available at that commodity's defining source at a specified time;

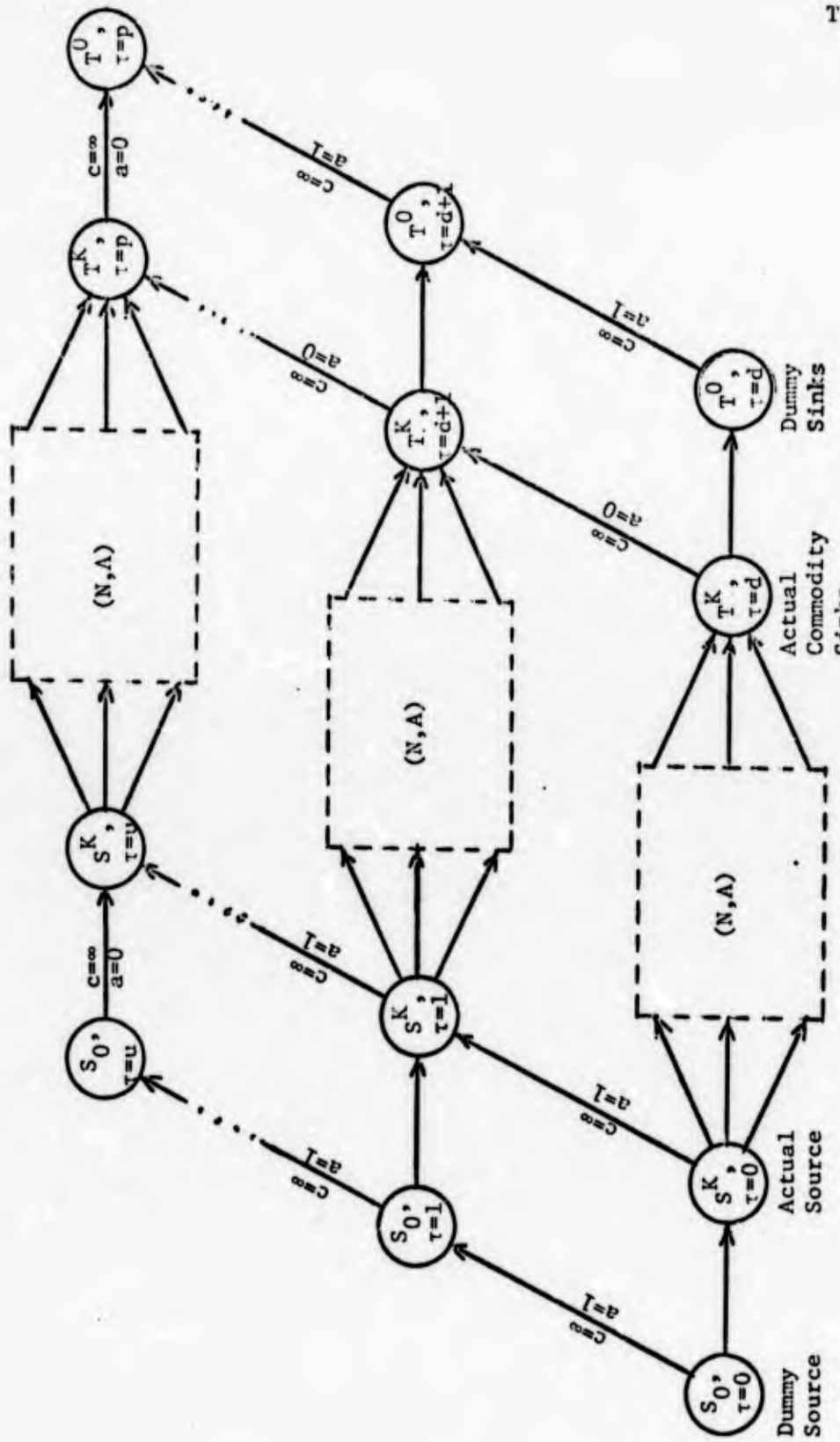


Figure 1

Time-Expanded Network Structure

d. to accept a list of one or more commodity requirements to be delivered at that commodity's defining sink at a specified time; and finally,

e. based on the above inputs, to derive a detailed shipping schedule that shows how and when each commodity's loads were depleted and delivered to that commodity's destination satisfying a requirement.

Definition and Explanation of Terms

All nodes of the time-expanded network consist of two identifying parts, a node name n and a time τ. Since the set S of sources and set T of sinks belong to the set of expanded nodes, they also have the identifying part, τ. For a source, τ is to be interpreted as a departure time; for a sink, τ is to be interpreted as an arrival time. If a load is specified as originating at a source, to completely identify that source the time of departure, τ, must be specified; if a requirement is specified for a load terminating at a sink, to completely identify that sink the time of arrival, τ, must be specified.

A chain of commodity K is a finite sequence of arcs in the expanded network from S_k/τ to t_k/τ arranged in order so that

$$M_j^{(k)} = [S_k/\tau , n_1/\tau) , (n_1/\tau , n_2/\tau) , \ldots , (n_n/\tau , t_k/\tau)] .$$

The n_i/τ in the chain are members of the set of expanded nodes. If there are m expanded arcs, k commodities, a available loads, and r required loads, a chain with its source load and sink requirement will be represented as an $(m + a + r) \times 1$ partitioned column vector, called a chain vector,

$$\hat{P}_j^{(k)} \begin{bmatrix} M_j^{(k)} \\ \hline A_j^{(k)} \\ \hline R_j^{(k)} \end{bmatrix}$$

where

(1) $M_j^{(k)} = \{m_{ij}\}$ is a column sub-vector in which

$$m_{ij} = \begin{cases} 1 & \text{if arc } i \text{ is in chain } M_j^{(k)} \\ 0 & \text{if arc } i \text{ is not in chain } M_j^{(k)} \end{cases},$$

(2) $A_j^{(k)} = \{a_{ij}\}$ is a column sub-vector in which

$$a_{ij} = \begin{cases} 1 & \text{if load } i \text{ is available at the source for chain } M_j^{(k)} \\ 0 & \text{if load } i \text{ is not available at the source for chain } M_j^{(k)} \end{cases}$$

(3) $R_j^{(k)} = \{r_{ij}\}$ is a column sub-vector in which

$$r_{ij} = \begin{cases} 1 & \text{if load } i \text{ is required at the sink for chain } M_j^{(k)} \\ 0 & \text{if load } i \text{ is not required at the sink for chain } M_j^{(k)} \end{cases}$$

S_k/τ , t_k/τ represents any source sink pair; a commodity is defined by a source-sink pair, i.e., $S_1/0$, $t_1/40$ defines commodity 1 as departing S at time zero with required arrival at $t = 40$; a load is a quantity of a commodity available at that commodity's defining source; a requirement is a quantity of a commodity to be delivered to that commodity's defining sink; the basis matrix[1] $\underline{B}$ and its inverse $\underline{B}^{-1}$ are initially identity matrices of order $m + a + r$, where

$\underline{m}$ = the number of arcs in the time-expanded network;

$\underline{a}$ = the number of loads available at the various commodity sources;

$\underline{r}$ = the number of requirements for delivery at the various commodity sinks;

---

[1] For comparable definitions of the linear program elements for the multi-commodity solution of the basic network G , see [1], Sections I and II.

$\underline{b}$ = the constraint vector and is an $(m + a + r) \times 1$ column vector composed of the m capacities of the arcs in the expanded network, the a available loads, and the r requirements;

$\underline{x_b}$ = the solution vector and is an $(m + a + r) \times 1$ column vector initially equal to the constraint vector b ;

$\underline{C_B}$ = the cost vector and is a $1 \times (m + a + r)$ row vector which is initially all zeros;

$\underline{C_B B^{-1}}$ = the linear program "pseudo-cost" vector and is a $1 \times (m + a + r)$ row vector of the form,

$$\overline{C} = (\alpha_1, \ldots, \alpha_m, \pi_1, \ldots, \pi_a, \sigma_1, \ldots, \sigma_r) ,$$

where the α's are associated with m arcs, the π's are associated with a available loads, and the σ's are associated with r requirements;

$\underline{VC}$ = the cost computed for the current vector to enter the basis;

$\underline{(x,y)}$ = any arc of the time-expanded network where x is the originating node of the arc and y is the terminating node of the arc, and associated with the arc are the values:

$\underline{Q(x,y)}$ = the arc length, determined by the pseudo-cost α_i of $\overline{C}$;

$\underline{c(x,y)}$ = the capacity of the arc;

$\underline{\bar{c}(x,y)}$ = the residual capacity of the arc;

$\underline{a(x,y)}$ = the traversal time of the arc;

$\underline{LCTR}$ = a label counter used to indicate whether or not a node y is labeled for the current shortest chain labeling pass;

$\underline{W_x}$ = the label count associated with node x ;

$\underline{W_y}$ = the label count associated with node y ;

$\underline{H_x}$ = the amount of flow available to leave node x ;

$\underline{H_y}$ = the greatest amount of flow able to reach node y ;

$\underline{K}$ = the number assigned to a commodity's defining source-sink pair;

A_k = the available load assigned to the defining source of commodity K ;

$\bar{A}_k$ = the residual load available to the source of commodity K ;

R_k = the requirement assigned to the sink of commodity K ;

$\bar{R}_k$ = the residual requirement assigned to the sink of commodity K ;

QA_k = the pseudo-cost associated with available load A_k and determined by the corresponding π_i of $\bar{C}$;

QR_k = the pseudo-cost associated with available load R_k and determined by the corresponding σ_i of $\bar{C}$;

V_x = the least cost for using node x in a chain;

V_y = the current least cost for using node y in a chain;

V'_y = the least cost over arc (x,y) for using node y ;

H'_y = the greatest amount of flow able to reach node y over arc (x,y) ;

CCL = the maximum V_y allowed for labeling or relabeling the sink node;

CCC = maximum H_y flow that can reach the sink over any path which gives the minimum V_y at the sink.

The term scanned is used in reference to a node to indicate that all arcs radiating from the node have been examined for the purpose of labeling the terminal nodes of the arc; p is the span of time over which the network has been expanded; a candidate chain is the chain currently to improve most the objective function of the linear program; $\hat{P}_j^{(k)}$ is the final candidate chain vector over all source-sink pairs which enters the basis;

P_i = a slack variable and is an (m + a + r) x 1 column vector of all zero elements except that the ith element = 1 .

Algorithm[1]

Step I: Initial solution.

Let $B = I_{(m+a+r)x(m+a+r)}$ and $x_B = b$ be a basic feasible solution. $B^{-1} = I$ and $C_B = (00...0)$.

[1]This subsection is based on [1], Section III.

Step II: Selecting P_s (the column vector to enter the basis matrix B).

1. Compute the linear program pseudo-costs

$$\bar{C} = C_B B^{-1} .$$

   a. If any element i of $\bar{C}$ is negative, then the first such element i determines that a slack variable P_i will be the vector P_s . Set VC equal to zero. Go to Step III.

   b. If all elements of $\bar{C}$ are greater than or equal to zero, then associate the ∞'s to arcs, the π's to available loads, and the σ's to requirements.

   c. Let the first source-sink pair be s_1/τ , t_1/τ , and for the first iteration only, initialize CCL to the cost of the shortest chain in the basic network. Set CCC = 0 .

2. Consider all nodes scanned and LCTR = LCTR + 1 (initially LCTR = 0). Then, label source S_k/τ as follows:

   a. indicate node labeled from itself,

   b. set W_x = LCTR ,

   c. set $H_x = \min(\bar{A}_k, \bar{R}_k)$,

   d. set $V_x = \tau + QA_k + QR_k$, and

   e. indicate source is unscanned.

3. Locate a node x that is unscanned (Initially, the source is the only such node.)

   A. For each arc (x,y) radiating from node x make the following computations:

      1) $V'_y = V_x + Q(x,y) + a(x,y)$ and

         $H'_y = \min[H_x , \bar{C}(x,y)]$,

      2) If $V'_y < CCL$ or if $V'_y = CCL$ and $H'_y > CCC$, then examine node y of arc (x,y) as follows:

         (a) If $W_y \neq$ LCTR , then label node y as follows:

            (1) indicate node y is labeled from node x ,

(2) set $W_y = LCTR$,

(3) set $H_y = H'_y$,

(4) set $V_y = V'_y$,

(5) indicate node y is unscanned,

(6) If node y is the current sink, set $CCL = V_y$ and $CCC = H_y$ and indicate sink is labeled, and

(7) process next arc (x,y) by returning to A.

(b) If $W_y = LCTR$, then if $V'_y < V_y$ or if $(V'_y = V_y)$ and $(H'_y > H_y)$, relabel node y as in (a)(1) through (a)(7); otherwise, do not relabel node y but process next arc (x,y) by returning to A.

(c) If arc (x,y) is not admissible to label across, process next arc (x,y) by returning to A.

B. After all arcs (x,y) radiating from node x have been examined, indicate node x has been scanned and return to 3.

4. When no unscanned node x can be found, if the current sink is labeled, go to 5; otherwise, go to 6.

5. The candidate chain will be the sequence of arcs from the current source to the current sink as indicated by the labels on the nodes. This is the chain of least cost and greatest capacity for the current source-sink pair. Save the information necessary to regenerate this chain should it become P_s , because the next series of labeling passes will destroy any current node labels.

6. If all source-sink pairs have been tried, go to 7; otherwise, locate the next source-sink pair and return to 2.

7. If a candidate chain is found, it becomes the vector P_s to enter the basis. Set VC equal to $p + 1 - [\sum_{(x,y)\in M_j^{(k)}} a(x,y)]$ and build the chain vector. Go to Step III.

If no candidate has been found yet, set CCL = CCL + 1 . If CCL now equals p + 1 , go to Step V; otherwise, start with the first source-sink pair again, set CCC = 0 , and return to 2.

Step III: Selecting P_r , the column vector to leave the basis B .

Compute

$$B^{-1}P_s = \begin{bmatrix} \bar{P}_{1s} \\ \bar{P}_{2s} \\ \vdots \\ \bar{P}_{(m+a+r)s} \end{bmatrix}$$

where P_s was selected in Step II

Compute

$$B^{-1}b = \begin{bmatrix} \bar{b}_1 \\ \bar{b}_2 \\ \vdots \\ \bar{b}_{(m+a+r)} \end{bmatrix} = x_B .$$

Form the quotient $\frac{\bar{b}_i}{\bar{P}_{is}}$ for each $\bar{P}_{is} > 0$, and select

$$\frac{\bar{b}_r}{\bar{P}_{rs}} = \min_{\bar{P}_{is} > 0} \frac{\bar{b}_i}{\bar{P}_{is}}$$

Now P_r is the vector to be removed from the basis.

Step IV: Computing $(B^*)^{-1}$.

Remove vector P_r from B and replace it with P_s , changing B to B* . Change cost vector C_B by assigning VC (cost of vector P_s) to the rth element of C_B .

$$(B^*)^{-1} = B^{-1} + KW_r$$

where W_r is the rth row of B^{-1} and

$$K = \begin{bmatrix} -K_1 & - \\ K_2 \\ \vdots \\ K_{r-1} \\ \vdots \\ K_{m+a+r} \end{bmatrix} \quad \text{and} \quad K_r = \frac{1}{\bar{P}_{rs}} ,$$

$$K_i = -\frac{\bar{P}_{is}}{\bar{P}_{rs}} \quad \text{for} \quad i \neq r .$$

Note, KW_r is an $(m+a+r) \times (m+a+r)$ matrix since K is $(m + a + r) \times 1$ and W_r is $1 \times (m + a + r)$.

Let $B^* = B$ and $(B^*)^{-1} = B^{-1}$ and go to Step II.

Step V:

The basis B is optimal and the solution is $X_B = B^{-1}b$.

Computational Experience

The computational procedures described in the preceding section were implemented in PL-1 programming language for use in the IBM 360/50 computer. Table 1 summarizes the characteristics of three network problems that were solved for multi-commodity dynamic flows. The network structures are shown in Figures 3 and 4.

Problem No.	Network No.	No. of Nodes	No. of Arcs	Expansion Time Periods	Nodes After Expansion	Arcs After Expansion	Commodity Source-Sink Pairs	Load Avail. Times	Commodity Loads at Source	Load at Sink		Rows in B^{-1} Matrix	No. of Iterations	Run Execution Time (min)
										Desired	Rcv'd			
1	1 (Fig 3)	54	90	40	1432	1970	1SC-51TT	10, 20, 30	350, 300, 450	350, 300, 450	350, 300, 288	1988	60	23
							55SC-53TT	10, 20, 30	325, 275, 400	325, 275, 400	300, 275, 108			
							7AC-50TT	10, 20, 30	300, 100, 130	300, 100, 130	300, 100, 130			
2	2 (Fig 4)	75	98	40	1631	2210	1SC-50TT	0	2900	2900	2900	2222	129	70
							02SC-53TT	5, 10	500, 400	500, 400	432, 288			
							04SC-54TT	0, 10	900, 300	900, 300	900, 300			
3	2	75	98	60	3132	4290	1SC-50TT	0, 20	2900, 1500	2900, 1500	(1)	4310	257	240(1)
							2SC-53TT	5, 10, 15	500, 400, 1200	500, 400, 1200	(1)			
							4SC-54TT	0, 10, 20, 25	900, 300, 1000, 1200	900, 300, 1000, 1200	(1)			

(1) Run stopped after 52 time periods because estimated pre-set run time was reached.

Table 1

Characteristics of Dynamic Multi-Commodity Network Runs on IBM 360/50 Computer

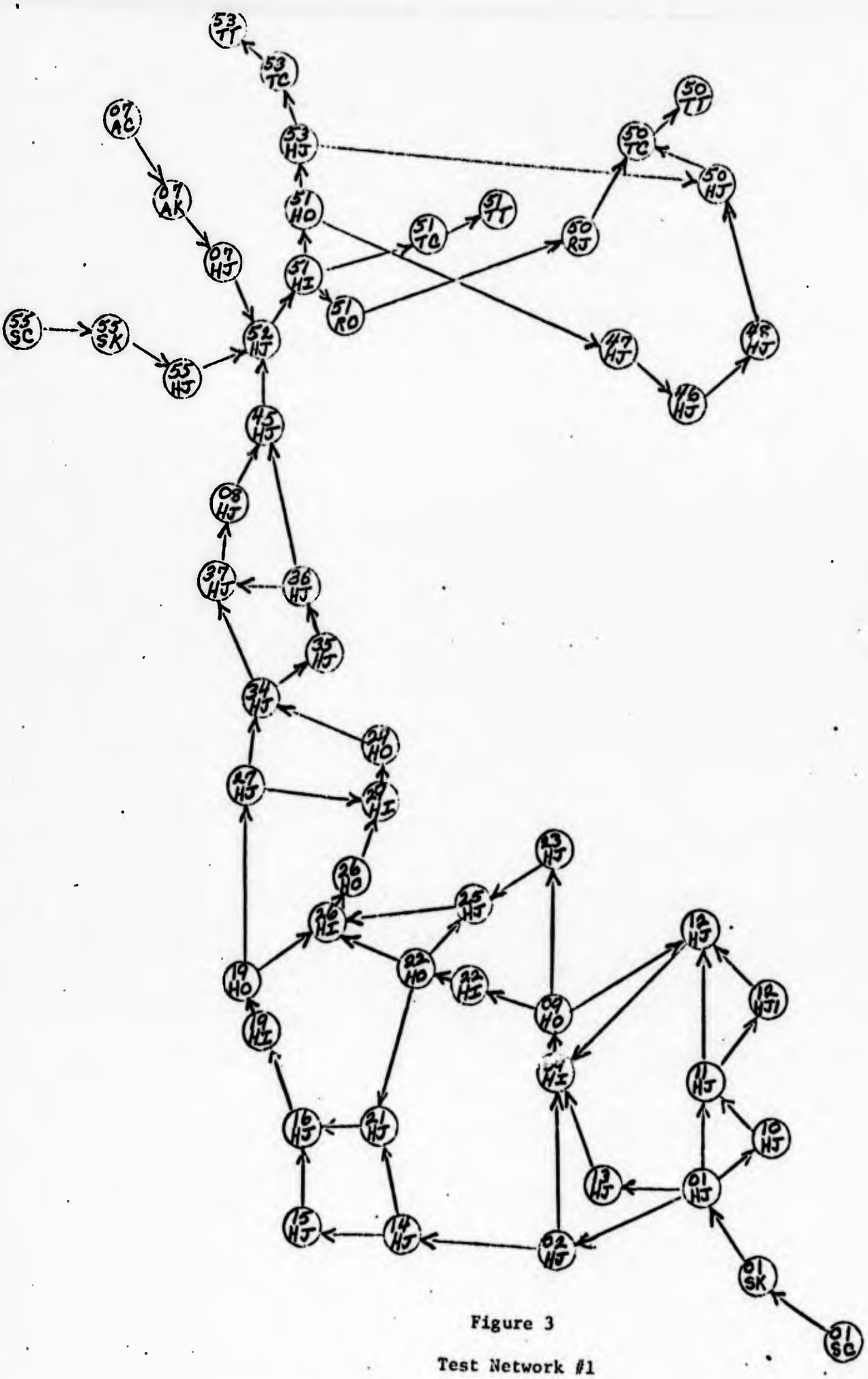


Figure 3

Test Network #1

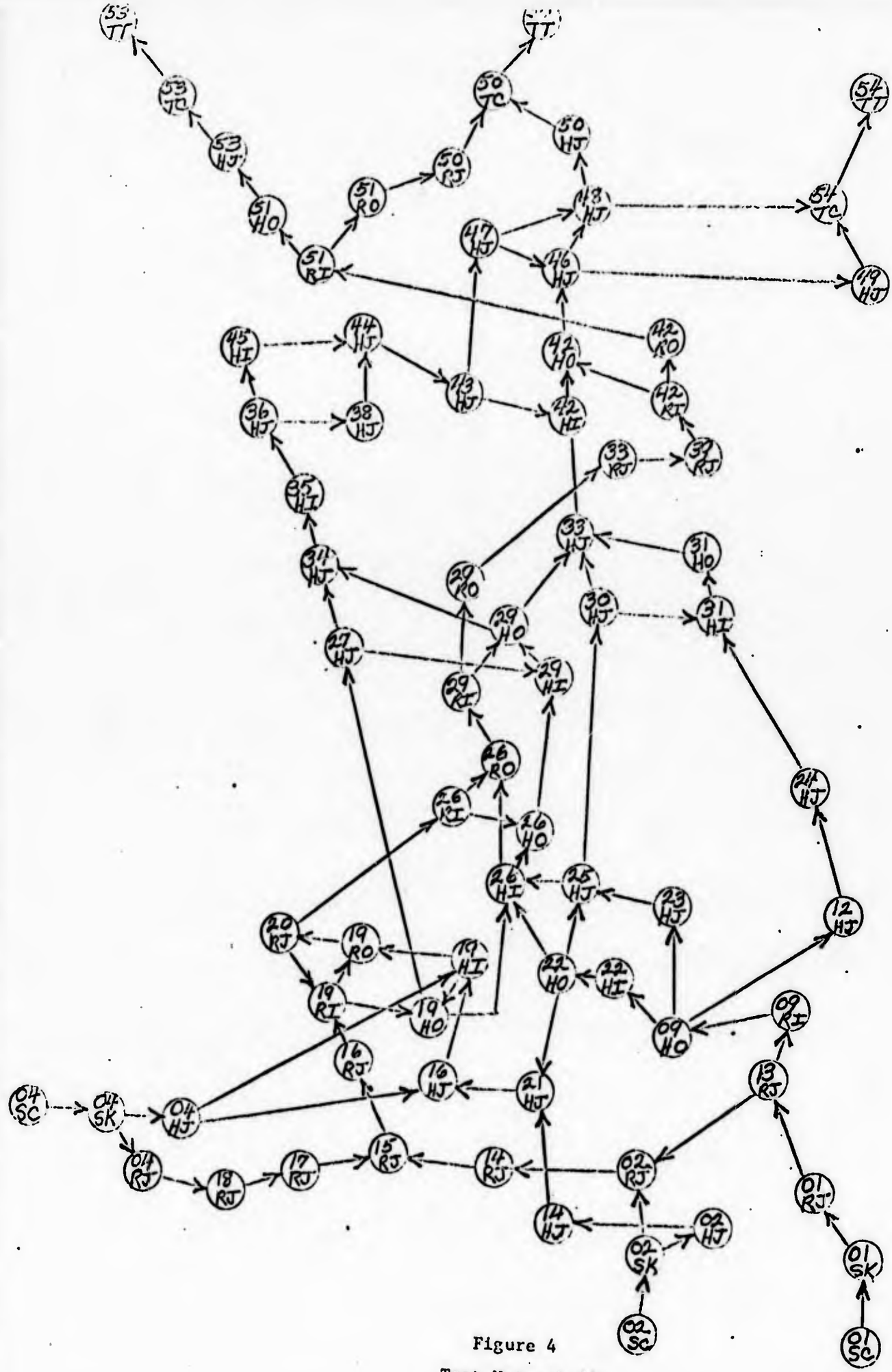


Figure 4

Test Network #2

Problem Areas in Achieving Processing Efficiency

In order to achieve a high level of processing efficiency a number of major problems had to be solved. A few of these, along with the solution method adopted, will be mentioned here to illustrate the type of problem encountered. Additional problems and details will be found in Appendix B.

A major problem was that of data retrieval for processing by the algorithm. A basic network of, say, only one-hundred arcs when time-expanded becomes a network of thousands of arcs. A network with thousands of arcs requires a direct-access external storage medium. The retrieval time for a segment of data from such external storage is many times greater than the retrieval time for data residing in the computer's high-speed internal memory. By writing an input/output subroutine that used all available computer memory for storing segments of the network and which maintained the most recently requested segments in this memory, data retrieval time was reduced substantially.

Another major problem was that a network of thousands of arcs when expressed as a matrix results in a matrix with thousands of rows and thousands of columns. An effective solution was to condense the rows to only nonzero elements. This resulted in a quite sparse matrix. However, even with the condensed matrix, there were still thousands of rows and the number of matrix computations at each iteration could be very time consuming. Fortunately, other shortcuts described in Appendix B reduced considerably the number of matrix rows processed in three out of four of the matrix computations. It is believed a future programming change described subsequently will further reduce the processing time for these computations.

Future Programming Improvements

Three proposed improvements for future programming changes with this model hold promise of contributing in a major way to additional processing efficiency. Details are given in Appendix C.

1. Expand the basic network arcs and nodes in shortest path sequence instead of by the current method of expanding it in FROM-node and TO-node sequence. This should improve the maintenance of the arc and node data records in the high-speed computer internal storage.

2. Partition the matrix so that rows corresponding to arcs can be stored separately from the rows corresponding to available loads and requirements. The less densely populated arc rows can then be condensed much more than at present, with consequent reduction in the time required to retrieve a group of rows from the external storage device.

3. Maintain an array of row indicators that specify whether or not a given row is an identity row. If a row is an identity row, it can be generated from its definition rather than retrieved from a storage device, again saving retrieval time.

2. General Program Structure

2.1 Inputs and Outputs

The multi-commodity computer model described herein was programmed in the PL/1 programming language for the IBM 360/50. The program was written as an overlay program consisting of three major executive routines and a large number of functional subroutines. The program was constructed in this manner to conserve the computer's internal storage and to facilitate modifications to the procedures as experience was gained with the algorithm's use and limitations during the developmental process.

The first executive routine is the Housekeeping routine (HSK) which controls three major initialization functions. This routine reads the processing option requested, locates and cross-references the input network, and reads the control cards that define the commodities with their loads and requirements. In general, this routine calls subroutines to perform the actual processing.

After the housekeeping routine has performed all necessary initialization, the next executive routine, the Main Processing Loop (MPL), is called in, with its subroutines, to overlay the housekeeping routine and receive control. The routine calls various subroutines to perform the iterative functions described in the algorithm. Control remains in the main processing loop until the algorithm has found the optimum solution.

After the main processing loop has completed its task, the third executive routine, End-of-Job (EOJ) is called in, with its subroutines, to overlay the main processing loop and perform end-of-job functions. This routine calls subroutines to list chains, list status of loads, and list arcs and nodes.

Before describing the subroutines of the program, the inputs and outputs to the program will be discussed. The first input to be considered is the control card file. The control card file may contain (in the order given) parameter cards, cards to modify individual expanded arc parameters, commodity source-sink cards, available load cards, requirement cards, and an end card. All cards are required except the cards to modify arc parameters.

The next input is the input time-expanded network. This input is generated by another program using the procedures described in Appendix A. Both this input and the control card file are processed during housekeeping.

All output goes to the printer and consists of a chain flow listing by commodity, a load-requirement-residue listing by commodity, and a listing of the time-expanded network with flows in the arcs. Output is given only at the end-of-job processing.

Four work files are required by the program, all of which must be on direct-access devices. The files are the following: (1) B^{-1} matrix storage file, (2) expanded arc file, (3) expanded node file, and (4) chain file. Special subroutines were written to handle the storage and retrieval for each file.

2.2 Program Overlay Structure

The diagram in Figure 5 shows the overlay structure of the program with each routine or subroutine location identified. The following is a description of each routine or subroutine identified in the diagram.

Multi-Commodity Time-Expansion Algorithm (MCTEA)

This is the first routine to receive control in the program. Its function is simply to call the three executive routines in sequence and control the overlay of those routines.

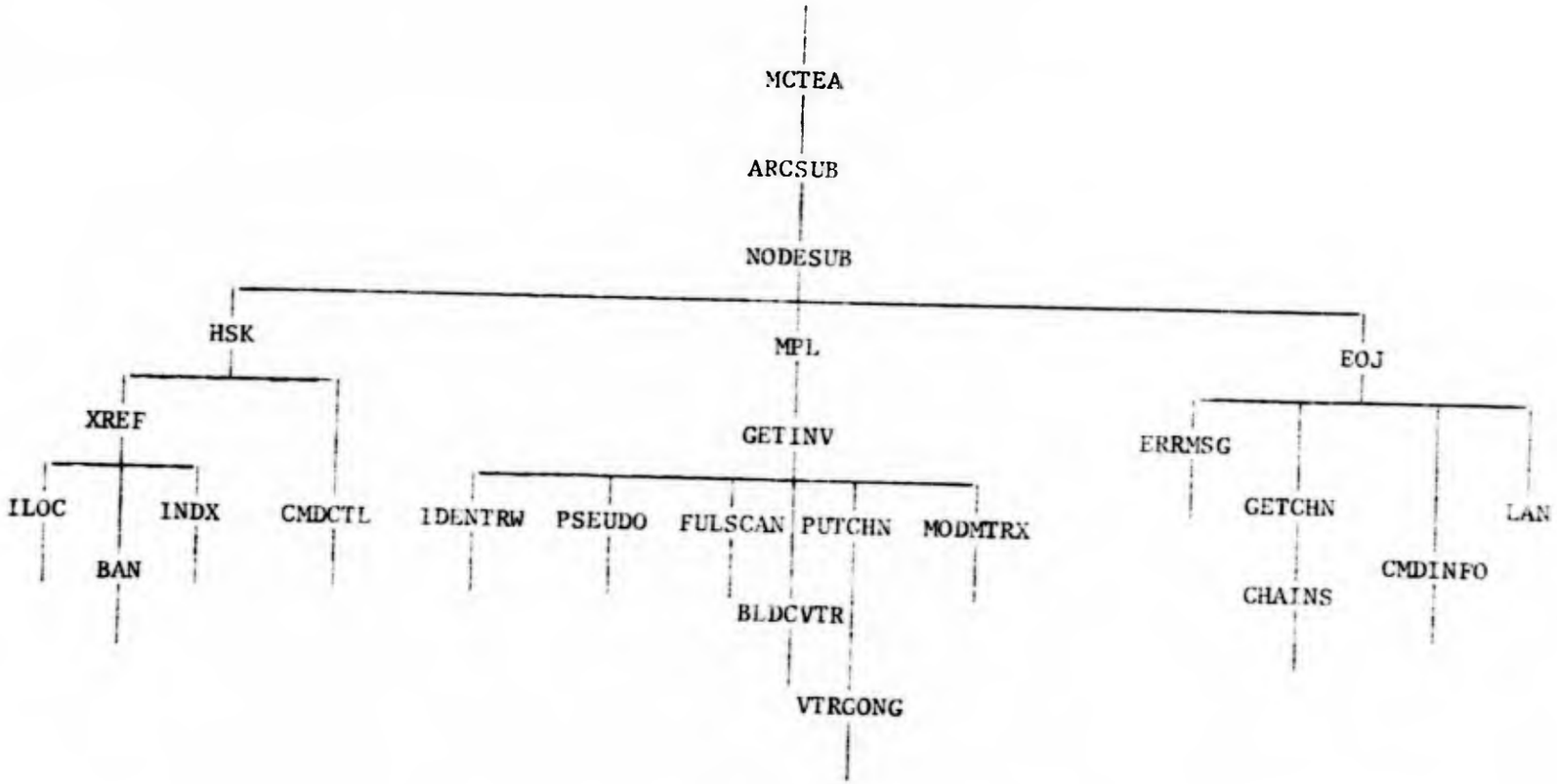


Figure 5

Multi-Commodity Time-Expansion Algorithm
(Overlay and Subroutine Diagram)

Arc Input/Output Subroutine (ARCSUB)

This subroutine controls all I/O functions connected with the arc temporary data set. It has three entry points:

(1) INITARC - the entry point that opens the arc direct-access work file for formatting, computes size of each I/O buffer according to user specified blocking factor, allocates user specified number of arc buffers, and prepares for overlap between processing and output I/O.

(2) ARCSUB - the main entry to store or retrieve an arc record. Accounting information about each arc buffer is maintained so that the most recently used subset of arcs is always in memory.

(3) CLRARC - the entry that clears out all arc buffers prior to closing the file.

Node Input/Output Subroutine (NODESUB)

This subroutine is the exact equivalent of ARCSUB except that it controls all I/O functions connected with the node temporary data set.

Housekeeping (HSK)

This is the first executive routine. It controls the initialization and processing of all input. It first reads the group of control cards that supplies the required program parameters as well as some of the optional ones, such as blocking factors, number of buffers, etc. It then calls in the routine XREF to locate and process the time-expanded network and calls CMDCTL to read and process the control cards that specify source-sink pairs, etc.

Cross-Reference (XREF)

The routine initializes certain variables and controls the calling and overlay of the subroutines ILOC, BAN, INDX.

Initialize and Locate Input (ILOC)

This subroutine basically opens the time-expanded network input file, reads the first records in which certain previously derived information is

passed from the time-expansion program and initializes certain variables based on the information that was passed.

Build Arcs and Nodes (BAN)

This subroutine reads the time-expanded arcs from the network input, makes any capacity changes specified by the control card input, builds tables for the arcs and for the nodes, and computes pointers for the arcs and nodes to tie the two tables together.

Index (INDX)

This subroutine builds nodes not previously built by BAN and adds them to the node table. It also searches the node table for TO-nodes (node y) of all arcs, computes a pointer for the node, and stores the pointer in the arc for future high-speed references.

Commodity Control (CMDCTL)

This subroutine reads, processes, and stores in tables the following cards: commodity source-sink pair cards, available load cards, and requirement cards. This subroutine is the last of the housekeeping executive functions. Control now passes back to the main control routine (MCTEA).

Main Processing Loop (MPL)

This is the second and most important executive routine. It overlays all of housekeeping and its associated subroutines. It first calls GETINV with a request to initialize the B^{-1} matrix data set, then it calls IDENTRW to initialize the B^{-1} matrix to an identity matrix. After the matrix is initialized, Step I of the algorithm is complete. The iterative linear program processing now begins (Steps II-IV of the algorithm) as follows:

a. MPL calls PSEUDO to compute the linear program "pseudo-costs" in vector $\bar{c}$ and to store these element pseudo-costs in their associated arc, load and requirement records. If a pseudo-cost is found to be negative, its corresponding slack vector becomes P_s .

b. If P_s is now a slack vector, MPL calls VTRGONG bypassing FULSCAN and BLDCVTR (which derive the shortest commodity chain and build a chain vector as P_s).

c. If P_s is not a slack vector, MPL calls FULSCAN to derive the shortest chain with the greatest capacity over all source-sink pairs and then calls BLDCVTR to build the chain vector P_s for that chain.

d. If no P_s can be found, the optimum solution has been found and control returns to the main control routine (MCTEA).

e. MPL calls VTRGONG to determine the vector leaving the basis (P_r) as in Step III of the algorithm.

f. Finally MPL calls MODMTRX to compute the new B^{-1} matrix and to update the solution vector X_B.

g. An iteration is complete and control passes back to (a) to begin next iteration.

B-Inverse Input/Output Subroutine (GETINV)

This subroutine controls all I/O functions connected with the B^{-1} matrix temporary data set. It has three entry points:

1. INITINV - the entry point that opens the direct-access work file for formatting, computes the size of each I/O buffer according to user specified blocking factor, allocates the buffers, and prepares for overlap between processing and I/O.

2. GETINV - the main entry point that controls the storage and retrieval of individual matrix rows. According to the setting of control switches, three sets of logic are employed:

   a. random retrieval with "look-ahead" - this set of logic provides "look-ahead" logic for retrieving B^{-1} matrix rows during the matrix computation, $\overline{C} = C_B B^{-1}$. This "look-ahead" is done by scanning row vector C_B for the next nonzero element that requires its corresponding B^{-1} matrix row to be retrieved from the external storage device on which the B^{-1} matrix resides; therefore, even though the retrieval is random, processing and I/O overlap are accomplished.

b. sequential retrieval with overlap - this set of logic provides anticipatory buffering for the sequential retrieval operation required by the matrix computation, $\bar{P}_{is} = B^{-1}P_s$.

c. random storage and retrieval - this set of logic provides for overlap between the outputting of the previously modified B^{-1} matrix row and the modifications to the current B^{-1} matrix row.

3. CLRINV - this entry point clears all buffers for the B^{-1} matrix and reinitializes control switches.

Write Identity Rows (IDENTRW)

This subroutine initializes B^{-1} to an identity matrix and initializes the solution elements to the values of the corresponding constraints.

Compute "Pseudo-Costs" (PSEUDO)

This subroutine computes the linear programming "pseudo-costs" by executing the matrix computation, $\bar{C} = C_B B^{-1}$. The subroutine then stores the elements of $\bar{C}$ in their associated records and, if any element is negative, then its corresponding slack vector is generated and stored in P_s , the vector to enter the basis. Control returns to MPL.

Fullscan Shortest Chain Algorithm (FULSCAN)

This subroutine generates the column that most improves the objective function by executing the procedures described in Step II of the algorithm. The chain of least cost and greatest capacity is generated over all source-sink pairs. If no chain of cost less than $p + 1$ (time span + 1) is found, a switch is set to indicate a solution has been reached; control returns to MPL which will signal end-of-job.

Build Chain Vector (BLDCVTR)

This subroutine builds the chain vector $\hat{P}_j^{(k)}$. The chain vector is built from the information generated by FULSCAN. Control returns to MPL.

Put Chain (PUTCHN)

This subroutine is called by VTRGONG to store the chain P_s in the basis column P_r as computed using the procedures given in the algorithm. Information about the chain is also stored for the future purpose of editing for print. Control returns to VTRGONG.

Compute Vector Leaving Basis (VTRGONG)

This subroutine computes the vector P_r to leave the basis according to the procedures described in the algorithm, Step III. Once P_r is computed, PUTCHN is called to replace P_r by P_s ; the rth element of C_B is updated; and the number r along with $\bar{P}_{rs}$ is stored for use while modifying B^{-1}. Control returns to MPL.

Modify B^{-1} Matrix (MODMTRX)

This subroutine first computes column vector K using vector $\bar{P}_{is}$, pivot element $\bar{P}_{rs}$, and pivot number r. Next the rth row of B^{-1} is brought into computer memory for use in modifying B^{-1}. Each nonzero element i of vector K is multiplied times each nonzero element j of the rth row of B^{-1}, W_r ; each resulting product is added to corresponding element j of the ith row of B^{-1}. After the entire ith row of B^{-1} is modified, it is used to recompute the ith solution element which is attached to B^{-1} row i ; following which the newly modified B^{-1} row i is written back on the external device housing the B^{-1} matrix. After all necessary modifications to B^{-1} and the solution elements have been made, control returns to MPL to begin the next iteration of processing.

End-of-Job (EOJ)

This is the last executive routine, and it controls all end-of-job processing. If the job is to be terminated because of an unrecoverable error condition, ERRMSG is called to print the diagnostics, then the output subroutines are called to list any output that may have been derived before the error occurred. If the job is terminating normally, the output subroutines are called:

1. List Chains (CHAINS) - to list all derived chain schedules by commodity;

2. List Commodity Information (CMDINFO) - to list all commodity loads and requirements with the amount of each that was delivered; and

3. Optionally, List Arcs and Nodes (LAN) - to print all arcs and nodes in readable format. The flows in the arcs are shown in the listing.

After all outputs are printed, control returns to the main control routine (MCTEA) to signal the termination of the job.

Error Message (ERRMSG)

This subroutine prints the diagnostic message set up by the subroutine that discovered the error condition and signalled the error. Control returns to EOJ.

Get Chain (GETCHN)

This subroutine is called by CHAINS to retrieve a chain from the chain work file for a specified commodity. Control returns to CHAINS.

List Chains (CHAINS)

This subroutine is called by EOJ to list all chain schedules derived by the solution. The chains are listed by commodity with the amount of flow, cost of chain, and accumulated flow. Control returns to EOJ.

List Commodity Information (CMDINFO)

This subroutine is called by EOJ to list, by commodity, the available loads with their residue and the required loads with load delivered. In addition the source departure and sink arrival times are given for each commodity load. Control returns to EOJ.

List Arcs and Nodes (LAN)

This is an optional output subroutine that lists all arcs with their flows and all nodes. Control returns to EOJ which then returns to the main control routine (MCTEA) to terminate computer run.

REFERENCES

[1] BOYER, DONALD D. and HUNT, DONALD J. (1968). A modified simplex algorithm for solving the multi-commodity maximum flow problem. Technical Paper Serial T-211. Logistics Research Project, The George Washington University.

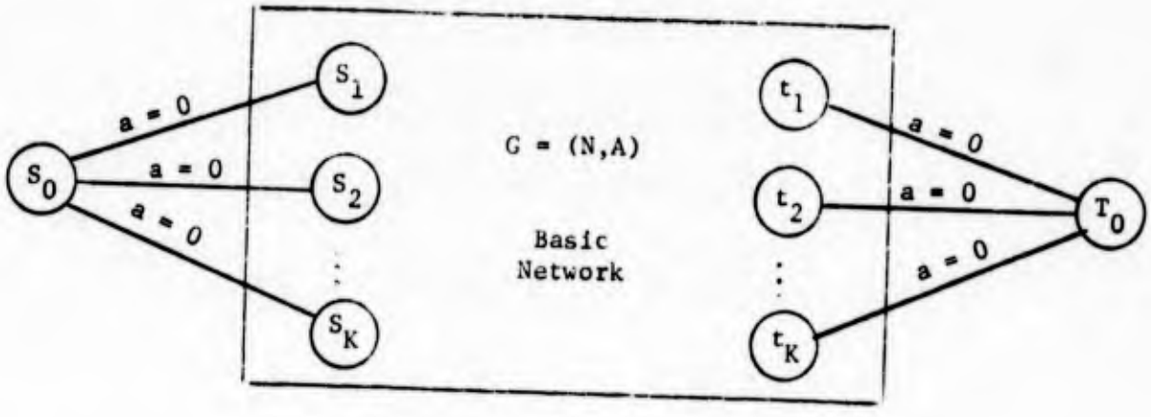
[2] FORD, L. and FULKERSON, D. R. (1958). A suggested computation for maximal multi-commodity network flows. Management Sci. 5 97-101.

[3] FORD, L. and FULKERSON, D. R. (1962). Flows in Networks. Princeton University Press, New Jersey.

APPENDIX A

Time-Expansion Procedures

Consider the basic network $G = (N,A)$, where N is a finite set of points called nodes and A is a set of pairs of points from N called arcs, each of which have two values associated with them -- a capacity and a traversal time. A set of sources $S \subset N$ and a set of sinks $T \subset N$ are given. Let us construct a dummy source node S_0 and a dummy sink node T_0; construct arcs from S_0 to each S_i of set S; construct arcs from each t_i of set T to T_0, and assign a traversal time of zero to each such constructed arc. The resulting basic network will look like the following.



Next, time-expand over p time periods from node S_0 to node T_0.

Before the time-expansion procedures are given, the following clarifying definitions of symbols are presented.

a. In the basic network G, an arc is represented by (x,y); in the time-expanded network $G(p)$, the replicated arcs of (x,y) are represented by $[x(\pi_x), y(\pi_x + a(x,y))]$ with capacity $c(x,y)$ and traversal time $a(x,y)$.

b. Here π_x, the node price, is the time unit at which node x will be reached from S_0.

The mechanics of expanding the basic network G to the time-expanded network $G(p)$ may be outlined as follows.

1. Establish p, the time span of interest.

2. For each node x of the node set N, determine the length (time), π_{0x}, of the shortest path [3] from the source, S_0. If the shortest path from S_0 to T_0, π_{0t}, is greater than p, the network will expand to the empty set of arcs.

3. For each node x of the node set N, determine the length (time), π_{xt}, of the shortest path to the sink, T_0.

4. For the real sources, S_i of the set S, generate source holdover arcs $[S_i(0), S_i(1)]$, $[S_i(1), S_i(2)]$, ..., $[S_i(n-1), S_i(n)]$ such that $n \leq p - \pi_{0t}$. Set $c(x,y) = \infty$ and $a(x,y) = 1$ for these holdovers.

5. For all arcs (x,y) of G, generate new arcs $[x(\pi_x), y(\pi_y)]$, where π_x and π_y take on values such that $\pi_x + a(x,y) = \pi_y$; $\pi_{0x} \leq \pi_x \leq p - \pi_{xt}$; and $\pi_{0y} \leq \pi_y \leq p - \pi_{yt}$. Capacity and traversal time for these replicated arcs are $c(x,y)$ and $a(x,y)$ respectively.

6. For the real sinks, t_i of set T, generate sink holdover arcs $[t_i(n), t_i(n+1)]$, $[t_i(n+1), t_i(n+2)]$, ..., $[t_i(p-1), t_i(p)]$ such that $n = \pi_{0x}$ where π_{0x} is the length (time) of the shortest path from S_0 to the real sink t_i. Set $c(x,y) = \infty$ and $a(x,y) = 0$ for these holdovers.

The resulting time-expanded network $G(p)$ will be as in the schematic diagram of Figure 1.

APPENDIX B

Approaches to Problems of Processing Efficiency

The PL-1 program as written allows a maximum of 32,000 arcs in the time-expanded network. An expanded network of even a fraction of the maximum size allowed presents many problems related to storage sizes and processing times. For such a large network, the number of records accesses to the arc and node data sets is extremely high. The arc data set and, especially, the node data set are very volatile random access files. Because of this volatility, it was necessary to program input/output subroutines that would take advantage of all the internal storage space available in an effort to maintain as large a percentage as possible of the arcs and nodes in the computer's high-speed storage area. In addition to avoid having stagnant data in this high-speed "memory," an accounting field was set up for each segment of data in "memory." These accounting fields gave the input/output subroutine the ability to execute the following decision: "In order to find room for a new needed segment of data, locate and replace the segment of data in 'memory' that has been used the least recently time-wise, but, if that segment was modified, write it back onto the external storage device to update the data set." Although no overlap is possible between <u>input</u> and processing, all <u>write</u> operations are done with processing overlap for the arc and node data files.

The problem of the very large B^{-1} matrix will now be discussed. A B^{-1} matrix of the allowed maximum (32,000+) number of rows, or even a fraction of that number, could produce prohibitive computer run times. There are several severe problems associated with a matrix of that size:

1. The number of columns to a row is equal to the number of rows ($m + a + r$, or a maximum 32,000+);
2. The four linear program steps that involve the B^{-1} matrix at each iteration:

   a. $\bar{C} = C_B B^{-1}$,

   b. $\bar{P}_{is} = B^{-1} P_s$,

c. $\bar{b}_i = B^{-1}b = X_B$, and

d. $(B^*)^{-1} = B^{-1} + KW_r$ where KW_r is a square matrix of the same dimensions as B^{-1} ;

3. The vectors $\bar{C}$, b , X_B , C_B , P_s , and the temporary work vectors all have $m + a + r$ elements.

For (1) it was found that the matrix B^{-1} is a very sparse matrix. So sparse, in fact, that no problem run to date has had more than 80 nonzero columns to a row. The sparsity of the rows made it possible to "shrink" the rows to only the nonzero elements with column numbers for each element, thereby making it possible to group the rows on the tracks of the direct access external storage device used by the program. This compact representation of the rows greatly decreases the time required to retrieve the matrix from the external storage device and allows much more overlap between the processing and retrieval of the rows.

The fact that the B^{-1} matrix is manipulated so many times per linear program iteration as shown in problem statement (2) poses still another problem. In particular 2.a. shows the computation $\bar{C} = C_B B^{-1}$ which normally requires B^{-1} to be processed column by column. To process B^{-1} by columns would be impractical since all other computations shown under (2) require B^{-1} to be processed row by row. Fortunately, it is unnecessary to perform computation $\bar{C} = C_B B^{-1}$ with columns. Since C_B is a row vector that is initially all zeros and is modified at the rate of one element per iteration, a much faster method of computing is available. Consider the following examples.

$$C_B = (00301)$$

$$B^{-1} = \begin{vmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & -1 & 0 & 1 \end{vmatrix}$$

If "$\bar{C} = C_B B^{-1}$" is computed using the usual column method, the resulting row vector will be

$$\bar{C} = (-3 \quad 1 \quad 2 \quad 0 \quad 1)$$

Let us perform the same operation in the following manner:

a. Initialize row vector $\bar{C}$ to zeros

$$\bar{C} = (0 \quad 0 \quad 0 \quad 0 \quad 0) ;$$

b. Locate the first nonzero element i in C_B and multiply that element by each nonzero element j in the ith row of B^{-1} adding each resulting matrix subproduct to each element j of $\bar{C}$. Using the third element "3" of C_B in our example and multiplying it by each element of row 3 of B^{-1}, the resulting "intermediate" $\bar{C}$ is

$$\bar{C} = (-3 \quad 0 \quad 3 \quad 0 \quad 0) ;$$

c. And continuing as in (b) until all nonzero elements of C_B are applied to the method, the final vector $\bar{C}$ will be

$$\bar{C} = (-3 \quad 1 \quad 2 \quad 0 \quad 1) ,$$

which is exactly the result obtained with the usual column method.

This method is particularly attractive when we realize that there will never be more nonzero elements in C_B than the number of iterations to the solution.

Let us look now at the computation shown in problem statement 2.b., namely "$\bar{P}_{is} = B^{-1}P_s$". There is no shortcut available here as there was for vector $\bar{C}$ in 2.a. since <u>every</u> row of B^{-1} must be multiplied by P_s. The program as written does nothing more than provide overlap between the row times P_s matrix multiplication and the retrieval of the rows from the external storage device. In spite of the overlap of processing and retrieval, matrix operation "$B^{-1}P_s$" continues to be the most time-consuming in the program.

Another B^{-1} matrix computation is "$X_B = B^{-1}b = \bar{b}_i$ in B^{-1} problem statement 2.c. There is a shortcut available as will be pointed out here and discussed more in the next paragraph. Consider the initial contents of the solution vector X_B which is equal to the constraint vector b as stated in Step I of the algorithm. For example, initially,

$$B^{-1} = I = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad b = \begin{bmatrix} 1 \\ 3 \\ 1 \\ 2 \\ 5 \end{bmatrix} \qquad X_B = b = \begin{bmatrix} 1 \\ 3 \\ 1 \\ 2 \\ 5 \end{bmatrix}.$$

Continuing our example, let us assume that row 4 of B^{-1} was modified during the first iteration of the linear program. Now let us compute $X_B = B^{-1}b$ using the new B^{-1} and see what changes have occurred in X_B as a result of the change of row 4 of B^{-1}.

$$B^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad b = \begin{bmatrix} 1 \\ 3 \\ 1 \\ 2 \\ 5 \end{bmatrix} \qquad X_B = B^{-1}b = \begin{bmatrix} 1 \\ 3 \\ 1 \\ 1 \\ 5 \end{bmatrix}.$$

The fourth solution element of X_B changed, and <u>only</u> that element, as a result of the change in the fourth row of B^{-1}. Keeping this in mind, let us consider the computations necessary to modify the B^{-1} matrix.

The final computation involving B^{-1} is stated in 2.d. This is the computation that modifies the old B^{-1} to form the new B^{-1} to be used in the next iteration of the linear program. The statement of the computation,

$$(B^*)^{-1} = B^{-1} + KW_r ,$$

assumes that the value of r and the column vector K have been previously computed. Let us assume that $r = 1$ and $W_r = W_1$, which is the first row of the old B^{-1}. Let us further assume that the vector K has been computed as follows.

$$B^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad K = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -1 \\ 0 \end{bmatrix} \qquad W_r = W_1 = (1 \quad 0 \quad 0 \quad 0 \quad 0)$$

Fortunately, it is not necessary to fully compute KW_r and store the resulting matrix, which is of the same dimensions as the B^{-1}, before performing the matrix addition $B^{-1} + KW_r$. Further, if any element i of K is zero, all elements in the ith row of KW_r will be zero, and the matrix addition of the ith row of B^{-1} and the ith row of KW_r will produce no change to the ith row of B^{-1}. Therefore, the only rows of B^{-1} that need to participate in the computation to produce the new B^{-1}, $(B^*)^{-1}$, are the rows that have a nonzero corresponding element in vector K. In the above example, the first and only nonzero element in K is the fourth element ("-1"), and the only B^{-1} row that needs to be retrieved, modified, and put back is the fourth row. Following the discussion of the previous paragraph, the modified fourth B^{-1} row is immediately multiplied with the constraint vector b to compute the new fourth solution element in X_B.

$$(B^*)^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad b = \begin{bmatrix} 1 \\ 3 \\ 1 \\ 2 \\ 5 \end{bmatrix} \qquad X_B = \begin{bmatrix} 1 \\ 3 \\ 1 \\ 1 \\ 5 \end{bmatrix}$$

If the B^{-1} matrix is modified in this manner and the solution vector is updated at the same time, the operation $B^{-1}b$ is essentially eliminated and the number of storage and retrieval operations from an external storage device is kept to a bare minimum.

The solution for the problem (3) of storing the large vectors that are required by the linear program has been suggested in the previous discussions concerning the B^{-1} matrix. That solution is to "shrink" all vectors down to

their nonzero elements. This solution can effectively be applied to all vectors and work spaces with the exception of the constraint vector b and the solution vector X_B. These two vectors are almost 100 percent nonzero. The least costly way to maintain these two vectors was determined to be the following.

1. To store the constraints in the arc and load records, i.e., store arc capacity i in arc i, etc.

2. To store the elements of the solution vector in the same record as the corresponding row of B^{-1}, i.e., store solution element i with B^{-1} row i.

APPENDIX C

Proposed Future Program Improvements

1. Although the input/output subroutine for the arc and node data sets is "tailored" to network flow problems, the current sequence [FROM-node (x) , TO-node (y)] of the expanded arcs and nodes as generated by the procedure in Appendix A is believed to be a major cause of the high computer run times for the program. A suggested future improvement to this sequence would enable the arc and node input/output subroutines to maintain a much better set of arcs and nodes in the computer's high-speed internal storage area. Compare the currently used sequence of the following arcs taken from the network in Figure 6, as they are time-expanded and the proposed sequence.

Figure 6 arcs: $\underline{(x,y)}$, $\underline{a(x,y)}$ $\underline{c(x,y)}$

(a) (0,1) , a(0,1) = 3 , c(0,1) = 50 ;

(b) (0,2) , a(0,2) = 6 , c(0,2) = 30 ;

(c) (0,3) , a(0,3) = 8 , c(0,3) = 15 ;

(d) (1,2) , a(1,2) = 2 , c(1,2) = 50 ;

(e) (1,4) , a(1,4) = 2 , c(1,4) = 25 ; and

(f) (2,3) , a(2,3) = 2 , c(2,3) = 15 .

Time-Expansion of Figure 6 to Time 17

Expansion 1 (Current) Arcs: $\underline{(x/t,y/t)}$	Expansion 2 (Proposed) Arcs: $\underline{(x/t,y/t)}$
(0/0,1/3)	(0/0,1/3)
(0/0,2/6)	(0/0,2/6)
(0/0,3/8)	(0/0,3/8)
(0/1,1/4)	(1/3,2/5)
(0/1,2/7)	(1/3,4/5)
(0/1,3/9)	(2/5,3/7)
(0/2,1/5)	(0/1,1/4)

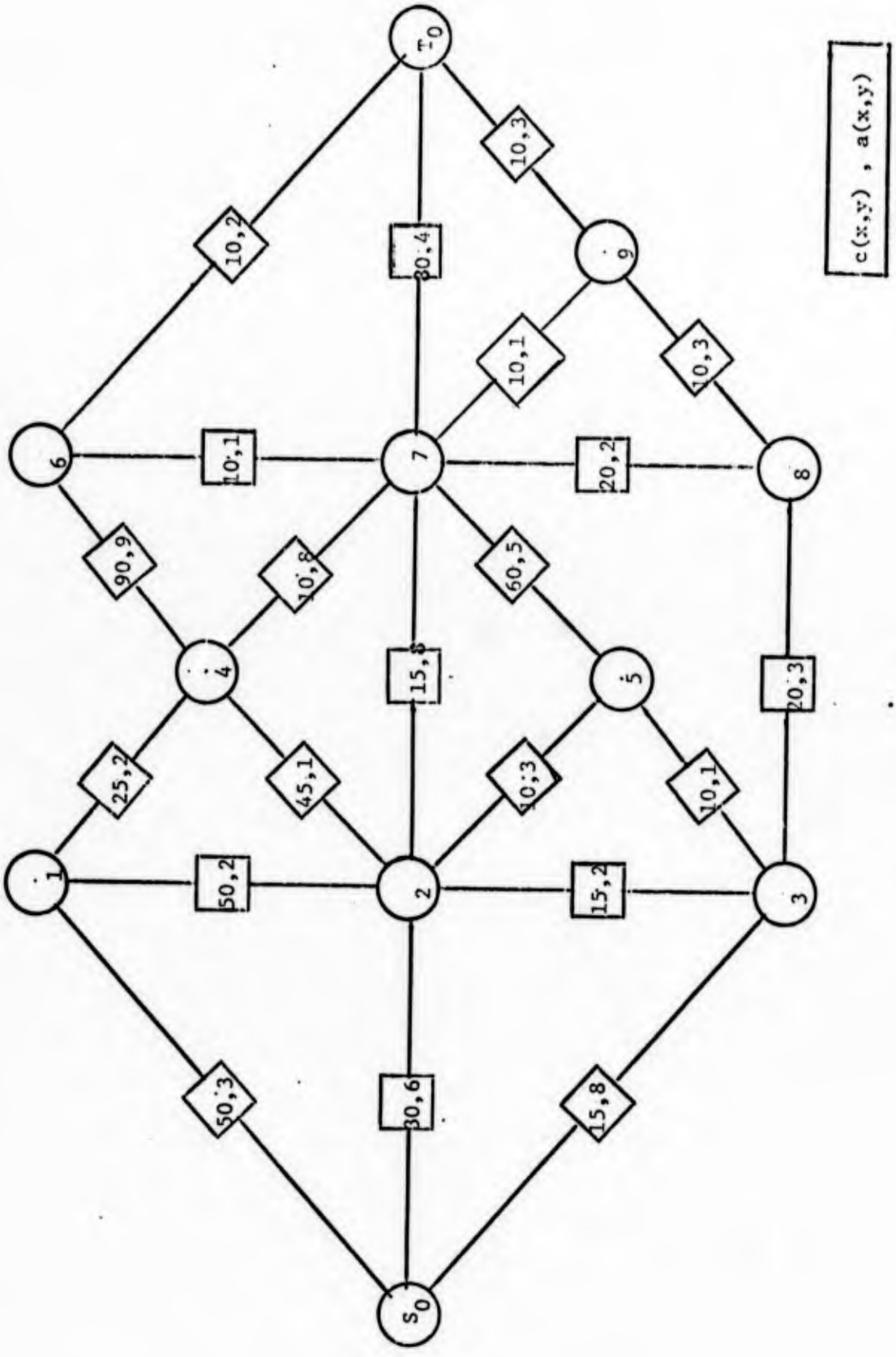

T0
S0
1
2
3
4
5
6
7
8
9
10,2
30,4
10,3
10,1
10,3
10,1
20,2
90,9
10,8
60,5
15,8
20,3
25,2
45,1
10,3
10,1
50,2
15,2
50,3
30,6
15,8
c(x,y) , a(x,y)

Figure 6

Expansion 1 Arcs: (Cont'd)	Expansion 2 Arcs: (Cont'd)
(1/3,2/5)	(0/1,2/7)
(1/3,4/5)	(0/1,3/9)
(1/4,2/6)	(1/4,2/6)
(1/4,4/6)	(1/4,4/6)
(1/5,2/7)	(2/6,3/8)
(2/5,3/7)	(0/2,1/5)
(2/6,3/8)	(1/5,2/7)
(2/7,3/9)	(2/7,3/9)

Nodes: n/τ	Nodes: n/τ
0/0	0/0
0/1	1/3
0/2	2/5
1/3	3/7
1/4	4/5
1/5	0/1
2/5	1/4
2/6	2/6
2/7	3/8
3/7	4/6
3/8	0/2
3/9	1/5
4/5	2/7
4/6	3/9

Let us assume that because of the space available the expanded arcs can be brought into computer "memory" only in groups of six and the expanded nodes can be brought into "memory" in groups of three. For the first group of three nodes in Expansion 1, an examination of the labeling method explained in Step II(2) under algorithm will show that only <u>one</u> node of that group will be usable before another group of three will be required in its place. For the first group of three nodes in Expansion 2, <u>two</u> nodes of the group will be usable before another group will be needed. For the first group of six arcs in Expansion 1, only three arcs will be usable before the next group of arcs will be required in its place. For the first group of six arcs in Expansion 2, <u>five</u> arcs will be usable before the next group will be needed. For a more practical sized expansion, the number of usable arcs and nodes in the groups will get less in Expansion 1, but could get greater in Expansion 2. To achieve the sequence in Expansion 2, expand all arcs in the basic network for the first time they will exist, expand all arcs in the basic network for the second time they exist, ... third, etc.

2. The first m rows of the B^{-1} matrix correspond to the m arcs of the expanded network and require fewer spaces for nonzero elements, perhaps as few as ten. The last $a + r$ rows that correspond to the $a + r$ available loads and requirements have been found to require at least 80 spaces for the nonzero elements. If the B^{-1} matrix were partitioned so that the first m rows could be stored separately from the last $a + r$ rows, the first m rows could be "shrunk" more than the last $a + r$ rows; and therefore, grouped on the tracks of a direct access device in greater numbers, thereby reducing matrix retrieval time greatly.

3. There are two facts about the B^{-1} matrix that should be noted:

   a. B^{-1} is initially an identity matrix, and

   b. Any row i of an identity matrix has only one nonzero element, that element has the value '+1', and that element is in the ith column of the row.

These two facts make it possible to avoid the costly task of retrieving a given row of B^{-1} if that row still has the contents it had initially. In the algorithm, Step IV, the operation of computing a new B^{-1} matrix is described. In this operation a small finite number of rows in the B^{-1} are modified to form the new B^{-1} for the next iteration. In each iteration, B^{-1} is modified more and more. If in the beginning, an array of switches were set up equal in number to the number of rows in B^{-1} and initialized to all ones (meaning all rows of B^{-1} are members of the initial identity matrix), each switch i could be set to zero as its corresponding B^{-1} row i is modified by Step IV so that the row is no longer a member of the initial identity matrix. For example, we have the initial B^{-1} and array of switches shown,

$$B^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad \text{Switches} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

and suppose Step IV modified B^{-1} rows 3 and 5 as follows,

$$B^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & -1 & 0 & 1 \end{bmatrix} \qquad \text{Switches} = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} .$$

Any subsequent operations involving the B^{-1} matrix may examine this array of switches and determine that it is necessary to retrieve from the external storage device only B^{-1} rows 3 and 5 and that B^{-1} rows 1, 2, and 4 can be "generated" using their definition. Even though a few thousand positions of the valuable high-speed internal computer storage would have to be reserved for the array of switches, it is believed there would be a 50 to 75 percent reduction in the overall computer time now necessary for the computation "$B^{-1}P_s$" .

NONE

Security Classification

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)*

THE GEORGE WASHINGTON UNIVERSITY
Logistics Research Project

2a. REPORT SECURITY CLASSIFICATION

NONE

2b. GROUP

3. REPORT TITLE

A DYNAMIC MULTI-COMMODITY, MULTI-MODE NETWORK FLOW MODEL

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

Scientific

5. AUTHOR(S) *(Last name, first name, initial)*

Hunt, Donald J. and Rosholdt, Erling F.

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
30 June 1969	43	3

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
N00014-67-A-0214	T-225
b. PROJECT NO. NR 047 001	9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*
c.	
d.	

10. AVAILABILITY/LIMITATION NOTICES

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

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
	Office of Naval Research

13. ABSTRACT

This paper describes a dynamic multi-commodity, multi-mode network flow model which permits time phasing of commodity load inputs and derives delivery schedules to the respective destinations over a time span of interest to the user. The model makes use of a time-expanded network. Methodology for time expanding a basic network is described and an algorithm for determining the commodity flow allocations is provided. Implementation of the model has been made in PL-1 programming language for use with the IBM 360/50 computer. A number of innovative programming steps which make possible very efficient processing are described and computing experience with several different network problems is reported.

DD FORM 1 JAN 64 1473

NONE

Security Classification

NONE

Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Dynamic Multi-Commodity						
Multi-Mode						
Network Flow Model						
Time-Expanded Network						
PL-1 Programming Language						

INSTRUCTIONS

1. ORIGINATING ACTIVITY: Enter the name and address of the contractor, subcontractor, grantee, Department of Defense activity or other organization *(corporate author)* issuing the report.

2*a*. REPORT SECURITY CLASSIFICATION: Enter the overall security classification of the report. Indicate whether "Restricted Data" is included. Marking is to be in accordance with appropriate security regulations.

2*b*. GROUP: Automatic downgrading is specified in DoD Directive 5200.10 and Armed Forces Industrial Manual. Enter the group number. Also, when applicable, show that optional markings have been used for Group 3 and Group 4 as authorized.

3. REPORT TITLE: Enter the complete report title in all capital letters. Titles in all cases should be unclassified. If a meaningful title cannot be selected without classification, show title classification in all capitals in parenthesis immediately following the title.

4. DESCRIPTIVE NOTES: If appropriate, enter the type of report, e.g., interim, progress, summary, annual, or final. Give the inclusive dates when a specific reporting period is covered.

5. AUTHOR(S): Enter the name(s) of author(s) as shown on or in the report. Enter last name, first name, middle initial. If military, show rank and branch of service. The name of the principal author is an absolute minimum requirement.

6. REPORT DATE: Enter the date of the report as day, month, year; or month, year. If more than one date appears on the report, use date of publication.

7*a*. TOTAL NUMBER OF PAGES: The total page count should follow normal pagination procedures, i.e., enter the number of pages containing information.

7*b*. NUMBER OF REFERENCES: Enter the total number of references cited in the report.

8*a*. CONTRACT OR GRANT NUMBER: If appropriate, enter the applicable number of the contract or grant under which the report was written.

8*b*, 8*c*, & 8*d*. PROJECT NUMBER: Enter the appropriate military department identification, such as project number, subproject number, system numbers, task number, etc.

9*a*. ORIGINATOR'S REPORT NUMBER(S): Enter the official report number by which the document will be identified and controlled by the originating activity. This number must be unique to this report.

9*b*. OTHER REPORT NUMBER(S): If the report has been assigned any other report numbers *(either by the originator or by the sponsor)*, also enter this number(s).

10. AVAILABILITY/LIMITATION NOTICES: Enter any limitations on further dissemination of the report, other than those imposed by security classification, using standard statements such as:

(1) "Qualified requesters may obtain copies of this report from DDC."

(2) "Foreign announcement and dissemination of this report by DDC is not authorized."

(3) "U. S. Government agencies may obtain copies of this report directly from DDC. Other qualified DDC users shall request through

_____."

(4) "U. S. military agencies may obtain copies of this report directly from DDC. Other qualified users shall request through

_____."

(5) "All distribution of this report is controlled. Qualified DDC users shall request through

_____."

If the report has been furnished to the Office of Technical Services, Department of Commerce, for sale to the public, indicate this fact and enter the price, if known.

11. SUPPLEMENTARY NOTES: Use for additional explanatory notes.

12. SPONSORING MILITARY ACTIVITY: Enter the name of the departmental project office or laboratory sponsoring *(paying for)* the research and development. Include address.

13. ABSTRACT: Enter an abstract giving a brief and factual summary of the document indicative of the report, even though it may also appear elsewhere in the body of the technical report. If additional space is required, a continuation sheet shall be attached.

It is highly desirable that the abstract of classified reports be unclassified. Each paragraph of the abstract shall end with an indication of the military security classification of the information in the paragraph, represented as *(TS)*, *(S)*, *(C)* or *(U)*.

There is no limitation on the length of the abstract. However, the suggested length is from 150 to 225 words.

14. KEY WORDS: Key words are technically meaningful terms or short phrases that characterize a report and may be used as index entries for cataloging the report. Key words must be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location, may be used as key words but will be followed by an indication of technical context. The assignment of links, roles, and weights is optional.

DD FORM 1 JAN 64 1473 (BACK)

NONE

Security Classification