The implementation of a multicommodity flow algorithm into a FORTRAN code is discussed. The algorithm is based on a gradient projection method [1] with diagonal scaling based on Hessian or Jacobian information. The flows carried by the active paths of each origin-destination (OD) pair are iterated upon one OD pair, per iteration. The data structures and memory requirements of the algorithm are discussed and are compared with those of other formulations based on link flows associated with each origin, and aggregate link flows.
IMPLEMENTATION OF AN OPTIMAL MULTICOMMODITY NETWORK FLOW
ALGORITHM BASED ON GRADIENT PROJECTION AND A PATH FLOW FORMULATION

by

Dimitri P. Bertsekas, Rob Gendron and Wei K. Tsai*

ABSTRACT

The implementation of a multicommodity flow algorithm into a FORTRAN code is discussed. The algorithm is based on a gradient projection method [1] with diagonal scaling based on Hessian or Jacobian information. The flows carried by the active paths of each origin-destination (OD) pair are iterated upon one OD pair at a time. Active paths are generated using a shortest path algorithm—one path per OD pair, per iteration. The data structures and memory requirements of the algorithm are discussed and are compared with those of other formulations based on link flows associated with each origin, and aggregate link flows.

*This research was supported by the National Science Foundation under grant NSF-ECS-8217668 and the Defense Advanced Research Projects Agency under grant ONR/N00014-75-C-1183.

*D. P. Bertsekas and W. K. Tsai are with the Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA 02139. B. Gendron is with Alphatech, Inc., Burlington, MA.
1. Optimal Multicommodity Flow Problem Formulation

We have a directed network with set of nodes $L$ and set of links $N$. Let $W$ be a collection of ordered node pairs referred to as origin-destination (OD) pairs. For each OD pair $w \in W$ we are given a positive number $r_w$ representing input flow into the network from origin to destination. Let $P_w$ be a given set of directed paths joining the origin node and destination node of OD pair $w$. ($P_w$ could be the set of all simple directed paths joining origin and destination, or it could be a restricted set of paths determined a priori on the basis of some unspecified considerations). Note that we do not exclude the possibility that two distinct OD pairs have the same origin and destination and possibly a different set of paths, but are associated with different classes or types of traffic.

Let $x_p$ be the flow carried by a generic path $p$. The optimization variables of the problem are $x_p$, $p \in P_w$, $w \in W$ and must satisfy the constraints

$$\sum_{p \in P_w} x_p = r_w , \quad \forall w \in W,$$

(1)

$$x_p \geq 0 , \quad \forall p \in P_w, w \in W.$$  

(2)

Let $x$ be the vector of all path flows

$$x = \{x_p | p \in P_w, w \in W\}$$  

(3)

For each link $(i,j)$ and OD pair $w$ we are given a continuously differentiable function $T_{ij}(x,w)$, which is to be interpreted as the length of link $(i,j)$ when the path flow vector is $x$. In data communication routing and traffic assignment problems $T_{ij}(x,w)$ usually has the interpretation of
marginal delay and travel time respectively (see [1]-[19]). We assume that for all feasible \( x \) and all \( w \in W \)

\[
T_{ij}(x,w) \geq 0, \quad \forall (i,j) \in L, \quad (4)
\]

The length of a path \( p \in P_w \) when the path flow vector is \( x \) is defined by

\[
L_p(x,w) = \sum_{(i,j) \in p} T_{ij}(x,w)
\]

i.e. it is the sum of lengths of its links.

The problem we are considering is the following:

Find a path flow vector \( x^* \) satisfying the constraints (1), (2) and such that for every \( w \in W \) and \( p' \in P_w \)

\[
x^* > 0 \implies L_p(x^*,w) \leq L_{p'}(x^*,w), \quad \forall p' \in P_w.
\]

(6)

In other words we are looking for a path flow pattern \( x^* \) whereby the only paths that carry positive flow are shortest paths with respect to the link lengths \( T_{ij}(x^*,w) \).

The problem described above includes, among others, problems of optimal routing in data networks [1]-[8] and (possibly asymmetric) traffic assignment problems in transportation networks [9]-[19]. We refer to the references just cited for extensive discussions. The survey paper [1] describes in detail the data communication context. A typical formulation there is to find a feasible path flow vector \( x \) that minimizes

\[
\sum_{(i,j)} D_{ij}(F_{ij})
\]

(7)
where $D_{ij}$ is a monotonically increasing, twice differentiable function of the total flow $F_{ij}$ of the link $(i,j)$ given by

$$F_{ij} = \sum_{w \in W} \sum_{p \in P_w} x_p \delta(p,i,j)$$

(8)

where

$$\delta(p,i,j) = \begin{cases} 
1 & \text{if link } (i,j) \text{ belong to path } p \\ 
0 & \text{otherwise.} 
\end{cases}$$

(9)

It can be shown (see e.g. [1]) that if we make the identification

$$T_{ij} = D'_{ij} : \text{The first derivative of } D_{ij}$$

(10)

the routing optimization problem falls within the framework of the general multicommodity flow problem described earlier.
2. **A Projection Method for Solving the Multicommodity Flow Problem**

The MULTIFLO and MULTIFLO1 codes given in Appendices I and II of this report implement an algorithm that solves the problem of the previous section for the case where for all OD pairs \( w \in W \)

\[ P_w = \text{Set of all simple paths joining the origin and destination of } w. \]

The set of OD pairs is divided into \( C \) groups called *commodities*. All OD pairs of a commodity have the same origin node. Furthermore the data structures of the codes can handle only the case where the lengths \( T_{ij}(x,w) \) depend on \( w \) through the corresponding commodity \( c \). That is

\[ T_{ij}(x,w) = T_{ij}(x,\bar{w}), \quad \forall (i,j) \in L, \quad \text{and OD pairs } w, \bar{w} \text{ of the same commodity } c. \]

It is also assumed that for all feasible \( F \)

\[ \frac{\partial T_{ij}}{\partial x_P} \geq 0 \quad \forall (i,j) \text{ belonging to the path } P \]

**MULTIFLO and MULTIFLO1** operate as follows:

At the beginning of the \( k \)th iteration we have for the generic OD pair \( w \in W \) a set of active paths \( P_w^k \) consisting of at most \( (k-1) \) paths. (These paths were generated in earlier iterations and it is implicitly assumed that all other paths carry zero flow). The following calculation is executed sequentially for each commodity—first for commodity 1, then for commodity 2, and so on up to the last commodity \( C \):
Step 1: A shortest path that joins the origin node for the commodity with all other nodes is calculated. The length for each link \( (i,j) \) used for this calculation is \( T_{ij}(x,w) \) where \( x \) is the current path flow vector. These shortest paths are added to the corresponding list of active paths of each OD pair of the commodity if they are not already there, so now the list of active paths for each OD pair of the commodity contains at most \( k \) paths.

Step 2: Each OD pair \( w \) of the commodity is taken up sequentially. For each active path \( p \) of \( w \) the length \( L_p \) [cf. (5)] is calculated together with an additional number \( \alpha_p \) called the stepsize (more on the choice of this later). Both \( L_p \) and \( \alpha_p \) are calculated on the basis of the current total link flow vector. Let \( p \) be the shortest path calculated in Step 1 for the OD pair. The path flows of all paths \( p \neq \bar{p} \) are updated according to

\[
x_p + \begin{cases} \max \{0, x_p - \alpha_p (L_p - L_p)\} & \text{if } L_p > L_p \\ x_p & \text{otherwise.} \end{cases}
\]

The path flow of the shortest path \( \bar{p} \) is then adjusted so that the sum of flows of all active paths equals \( r_w \) as required by the constraint (1), i.e.

\[
x_{\bar{p}} + r_w = \sum_{\text{active } p \neq \bar{p}} x_p.
\]

In other words an amount \( x_p \) or \( \alpha_p (L_p - L_\bar{p}) \) is shifted from each nonshortest path to the shortest path \( \bar{p} \)--whichever is smaller. The total link flows \( F_{ij} \) are adjusted to reflect the changes in \( x_p \) and \( x_{\bar{p}} \).

The rationale for iteration (11) is explained in [1], [6], [8], [9].
It is based on a gradient projection method [9], [21]. Note that it is possible that \( L_p < L_p^- \) for some \( p \neq \bar{p} \) even though \( \bar{p} \) was calculated earlier as a shortest path. The reason is that by the time \( L_p \) and \( L_p^- \) are computed the total link flow vector may have changed since the time the shortest path has been calculated due to iterations on the path flows of other OD pairs of the same commodity.

Regarding the choice of the stepsize \( \alpha_p \), the MULTIFLO and MULTIFLO1 codes use the following formula for all \( p \neq \bar{p} \)

\[
\alpha_p = S_p^{-1}
\]  

where

\[
S_p = \sum_{(i,j) \in L_p \setminus (i,j) \not\in \bar{p}} \Theta_{ij}
\]

and \( L_p \) is the set of links

\[
L_p = \{(i,j) \mid (i,j) \text{ belongs to either } p \text{ or } \bar{p}, \text{ but not to both } p \text{ and } \bar{p}\}.
\]

The rationale for this is as follows:

If we interpret the algorithm as one that tries to satisfy the equation

\[
\dot{L}_p - L_p^- = 0, \quad \forall p \text{ with } x_p > 0,
\]

a natural choice for \( \alpha_p \) is

\[
\alpha_p = \frac{\Delta x_p}{\Delta (L_p - L_p^-)}
\]

where \( \Delta (L_p - L_p^-) \) is the variation of \( (L_p - L_p^-) \) resulting from a small variation
Δx_p in the path flow x_p (and an attendant variation -Δx_p in the path flow x_p). This corresponds to an approximate form of Newton's method whereby only the diagonal elements of the Jacobian matrix (corresponding to the current OD pair) are taken into account while the off-diagonal terms are set to zero (see also [1] for further discussion). For Δx_p → 0 it is easily seen that (17) yields

$$\alpha_p^{-1} = \sum_{(i,j) \in P} \left( \frac{\partial T_{ij}}{\partial x_p} - \frac{\partial T_{ij}}{\partial x_{-p}} \right) + \sum_{(i,j) \in P} \left( \frac{\partial T_{ij}}{\partial x_{-p}} - \frac{\partial T_{ij}}{\partial x_p} \right).$$  \hspace{1cm} (18)

In most cases of interest we have

$$\frac{\partial T_{ij}}{\partial x_p} = \frac{\partial T_{ij}}{\partial x_{-p}} \quad \text{if } (i,j) \in P \text{ and } (i,j) \not\in P$$

$$\frac{\partial T_{ij}}{\partial x_p} = 0 \quad \text{if } (i,j) \not\in P$$

$$\frac{\partial T_{ij}}{\partial x_{-p}} = 0 \quad \text{if } (i,j) \not\in P$$

so (18) becomes approximately [c.f. (18), (14)]

$$\alpha_p^{-1} = \sum_{(i,j) \in P} \frac{\partial T_{ij}}{\partial x_p} = S_p,$$

thereby justifying the use of the stepsizes (13), (14).

If one wishes to employ the formula (18) for the stepsizes it is necessary to modify the codes. These modifications should not be too
difficult for an experienced user. Another possibility is to use a smaller value of stepsize $\alpha_p$ than the one given by (13)—for example $\alpha_p = \rho S_p^{-1}$

$\rho \in (0,1)$ is a fixed relaxation parameter. (A smaller stepsize enhances the convergence properties of the algorithm but may deteriorate its rate of convergence). This can be accomplished without any changes in the code by simply introducing the relaxation parameter $\rho$ in the subroutine that calculates $\frac{\partial I_{ij}}{\partial x_p}$ [cf. (14)].

In the MULTIFLO code a shortest path tree is generated and stored at each iteration for each commodity. As a result the memory storage for shortest paths is proportional to the number of iterations so for large problems one cannot execute a large number of iterations without incurring a heavy penalty for disk I/O. MULTIFLO will usually find in five to ten iterations what is for most practical problems an adequate approximation to an optimal solution. This is particularly true of lightly loaded networks (e.g. with utilization of all links less than 60% at the optimum). For heavily loaded networks the number of required iterations usually tends to be larger (say 10-30). It should be a rare occasion when a user will require more than thirty iterations for his practical problem.

MULTIFLOI differs from MULTIFLO only in the method used for storing the active paths. MULTIFLOI stores explicitly all active paths in a single array rather than storing them implicitly through the generated shortest path trees. As a result the memory storage of MULTIFLOI depends on the number of active paths generated and is largely independent of the number of iterations executed. For certain problems including situations where a large number of iterations is desired MULTIFLOI may hold a storage advantage over MULTIFLO. Both codes generate identical numerical results although MULTIFLOI appears to be somewhat faster on sample test problems.
3. Data Structures for Representing the Problem

The data structures of MULTIFLO and MULTIFLO1 are described in the code documentation. The problem input structure will be illustrated here by means of the 5 node-6 link network shown in Figure 1:

Node Length Arrays (FRSTOU, LASTOU):

These arrays specify the network topology.

FRSTOU(NODE): The first link out of NODE

LASTOU(NODE): The last link out of NODE
Note that all arcs with the same head node must be grouped together in the arc list. A node with no outgoing links is recognized via FRSTOU = 0

Arc Length Arrays (STARTNODE, ENDNODE)

These arrays also specify the network topology:

STARTNODE (ARC): The head node of ARC
ENDNODE (ARC): The tail node of ARC

Commodity Length Arrays (ORGID, STARTOD)

ORGID (COMMODITY): The origin node of COMMODITY
STARTOD (COMMODITY): A pointer to the first OD pair of COMMODITY on the OD pair list
For the example of Figure 1 we will assume three commodities

<table>
<thead>
<tr>
<th>COMMODITY</th>
<th>ORGID</th>
<th>STARTOD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>

Note that it is required that OD pairs are listed sequentially by commodity, i.e. the OD pairs of commodity 1 are listed first, followed by the OD pairs of commodity 2, etc. Therefore the STARTOD array together with the total number of OD pairs specify all OD pairs associated with each commodity.

OD Pair Length Arrays (DEST, INPUT_FLOW)

DEST(OD): The destination node of OD
INPUT_FLOW(OD): The input traffic of OD

<table>
<thead>
<tr>
<th>OD</th>
<th>DEST</th>
<th>INPUT_FLOW</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>problem dependent</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>&quot;</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>&quot;</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>&quot;</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>&quot;</td>
</tr>
</tbody>
</table>

From the arrays ORGID, STARTOD and DEST together with the total number of OD pairs the set of OD pairs corresponding to each commodity is completely specified. For our example these are:
<table>
<thead>
<tr>
<th>COMMODITY</th>
<th>OD PAIRS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(2,3), (2,5)</td>
</tr>
<tr>
<td>2</td>
<td>(1,3)</td>
</tr>
<tr>
<td>3</td>
<td>(1,4), (1,5)</td>
</tr>
</tbody>
</table>

Additional input information is required to calculate the link lengths $T_{ij}$ and their first derivatives $\frac{\partial T_{ij}}{\partial x_p}$ in the subroutine DERIVS and DERIV1. This is of course problem dependent. The listing of Appendix I gives an example which is typical of routing problems in data networks [cf. equations (7)-(10)].
4. Memory Requirements - Comparisons with Other Methods

The memory storage requirements of both MULTIFLO and MULTIFLO1 are substantial, but this is true for all methods that provide as output not only the optimal total link flows but also detailed information about the optimal routing from origins to destinations (i.e. optimal path flows).

Assuming that 1 byte is allocated for a logical variable, 2 bytes are allocated for storing a node or link identification number and an iteration number, 4 bytes are allocated for storing a commodity, OD pair or path identification number, and 4 bytes are allocated for storing a real number (e.g. a path or link flow) the total array storage in bytes of MULTIFLO during execution is

\[ 6n_N + 9n_L + 6n_C + 6n_{OD} + 10n_P + 2n_I n_N n_C \]  

where:

- \( n_N \): Number of nodes
- \( n_L \): Number of links
- \( n_C \): Number of commodities
- \( n_{OD} \): Number of OD pairs
- \( n_P \): Number of active paths generated
- \( n_I \): Number of iterations.

Additional storage is required for information necessary to calculate link lengths and their derivatives but this is typically of order \( O(n_L) \) and is not significant.

The dominant array as far as storage of MULTIFLO is concerned is the
triple indexed PRED array which stores the shortest path trees generated for each commodity at each iteration. This array accounts for the last term \(2n_I n_N n_C\) in (19). The term \(10n_P\) is also substantial since the number of active paths \(n_P\) can be as large as \(n_I n_{OD}\). However, because the algorithm stores a path only once at the iteration it is first generated and does not duplicate it if it is generated again later, the actual number \(n_P\) is typically much smaller than \(n_I n_{OD}\). This was confirmed by extensive computational experimentation, that showed that except for very heavily loaded networks the actual number of active paths \(n_P\) was typically no more than \(2n_{OD}(1)\) and often considerably less. We conclude therefore that the dominant bottleneck for storage is the shortest path description array PRED requiring \(2n_I n_N n_C\) bytes.

In the MULTIFLO1 code the array PRED is not used. In its place the array PDESCR is used which requires storage of \(2n_P n_N\) at most. This calculation assumes conservatively that a path has \(n_N\) links. However in practice the actual storage for PDESCR is several times less than \(2n_P n_N\). If we adopt the rough estimate \(n_P = 2n_{OD}\) then we conclude that the storage requirements of MULTIFLO and MULTIFLO1 are roughly comparable if the number of iterations \(n_I\) is comparable to something between \(n_{OD}\) and \(\frac{n_{OD}}{4n_C}\) with MULTIFLO1 becoming definitely preferable if \(n_I = \frac{n_{OD}}{n_C}\).

MULTIFLO1 is also preferable for problems that are solved repetitively with minor variations in their data since then the knowledge of the path description array PDESCR can be fruitfully exploited. This is not possible with MULTIFLO.

In large problems where only the total link flows are of interest (e.g. traffic assignment problems) a different algorithm [e.g. the flow
Deviation (or the Frank-Wolfe) method [3], [8] or the Cantor-Gerla (or simplicial approximation) method [4], [15], may be preferable over MULTIFLO or MULTIFLO1, since then storage of order $O(n_L)$ or perhaps $O(n_L n_n L)$ is required. However when detailed routing information is of interest the memory storage requirements of MULTIFLO are competitive with those of other methods based on shortest paths including the Flow Deviation and Cantor-Gerla methods. The reason is that detailed routing information can be provided by these methods only if the shortest paths generated at each iteration are stored explicitly in an array such as PRED, and as mentioned earlier this is the main memory storage bottleneck.

There are algorithms that can solve multicommodity flow problems and provide detailed routing information without requiring the generation and storage of shortest paths. These algorithms are based on a link flow formulation [20], or the link flow fraction formulation due to Gallager [2], [5], [7] whereby the optimization variables are the flows or fractions of flow respectively for each commodity that are routed along each link. The storage requirement for these algorithms is of order $O(n_C n_L)$ and is independent of the number of iterations. When we compare this storage with the $O(n_1 n_C n_L)$ storage of algorithms based on shortest paths we see that link flow formulations hold an advantage in terms of storage for problems where a large number of iterations is desirable. The reverse is true if the number of iterations required for adequate solution of the problem is small, or if the number of links is much larger than the number of nodes.
We finally note a final advantage of the path flow formulation over link flow formulations. When the set of paths for each OD pair is restricted to be a given strict subset of the set of all possible simple paths it is extremely cumbersome to use a link flow formulation. By contrast it is straightforward to modify the MULTIFLO1 code to handle this situation.
References


APPENDIX I: MULTIFLO Code

The following FORTRAN code works on the VAX family of computers. It consists of a DRIVER program and several subroutines:

LOAD: Reads network topology and link length data from disk.

MULTIFLO: This is the main algorithm.

SP: Calculates a shortest path tree from an origin node to all other nodes.

PRFLOW: Prints out to disk problem data and algorithmic results.

DERIVS: This user supplied routine calculates for a given link (i,j) its length \( T_{ij} \) (DICAL) and the length derivative \( \frac{\partial T_{ij}}{\partial x_p} \) (D2CAL).

DERIVI: This routine is the same as DERIVS except that it calculates the length \( T_{ij} \) (DICAL) but not the length derivative \( \frac{\partial T_{ij}}{\partial x_p} \).

DELAY: This user supplied routine is useful only if the multicommodity flow problem is a routing optimization problem of the form (7)-(10) as described in Section 1. For asymmetric traffic assignment problems it has no purpose. It calculates the total delay

\[
D_{ij}(F_{ij}) = \sum_{(i,j)} D_{ij}(F_{ij})
\]

where \( D_{ij} = T_{ij} \) [cf. (7)-(10)]. The value of \( D_{ij}(F_{ij}) \) is calculated using the function DCAL.

Two versions of the shortest path routine SP are provided (SHORTPAPE and SHORTHEAP) which can be used interchangeably. SHORTHEAP is recommended for problems where there are only few destinations for each commodity. Otherwise SHORTPAPE based on [23] should be preferable.

A program (SETUP) is also provided for the purpose of creating the data describing the problem in a format that is compatible with the LOAD routine.
The routines LOAD, DERIV1, DERIVS, DELAY, and DCAL supplied in this appendix correspond to the most commonly solved optimal routing problem in data communication network applications whereby a capacity $C_{ij}$ is given for each link $(i,j)$ (this is the array BITRATE in the code) and

\[
D_{ij}(F_{ij}) = \frac{F_{ij}}{C_{ij}-F_{ij}} \quad \text{(M/M/1 Queueing Delay)} \quad (A.1)
\]

\[
T_{ij}(F_{ij}) = \frac{C_{ij}}{(C_{ij}-F_{ij})^2}
\]

\[
\frac{\partial T_{ij}(F_{ij})}{\partial F_{ij}} = \frac{2C_{ij}}{(C_{ij}-F_{ij})^3}.
\]

Because $D_{ij}(F_{ij}) \to \infty$ as $F_{ij} \to C_{ij}$, these formulas have been modified so that if $F_{ij} \geq \rho C_{ij}$, where $\rho \in (0,1)$ is a parameter set by the user, then $D_{ij}$, $T_{ij}$, and $\frac{\partial T_{ij}}{\partial F_{ij}}$ are calculated using a quadratic function which has the same value, first and second derivatives as $\frac{F_{ij}}{C_{ij}-F_{ij}}$ at the breakpoint $\rho C_{ij}$.

In the program the parameter $\rho$ is given by the variable MAXUTI set in the subroutine LOAD to 0.99. The user may wish to change this value. The guideline is that $\rho$ should be set at a value exceeding the maximum link utilization

\[
\max_{(i,j) \in L} \frac{F_{ij}}{C_{ij}}
\]

at the optimal solution. This trick gets around situations whereby the input flows are so large that exceeding some of the link capacities during some phase of the algorithm is inevitable.
The MULTIFLO code will stop computing when one of two conditions is met: Either the maximum number of iterations (MAXITER) is exceeded or a normalized measure of deviation from the optimal solution falls below a certain tolerance (TOL). This measure is roughly equal to the percentage of input traffic of an OD pair that does not lie on a shortest path (maximized over all OD pairs), and its magnitude is not substantially affected by the size of the problem. Both convergence parameters MAXITER and TOL are set by the user in the subroutine LOAD.
'DRIVER' IS A SIMPLE EXECUTIVE TO INVOKE THE 'MULTIFLO' COMMODITY ROUTING PROGRAM. 'DRIVER' INVOKES SUBPROGRAM 'LOAD' TO READ DATA INTO 'MULTIFLO' INPUT COMMON BLOCKS. FILES READ BY 'LOAD' ARE CREATED BY A TERMINAL SESSION WITH THE USER FOR NETWORK DEFINITION THROUGH THE USE OF PROGRAM 'SETUP'.

EXECUTION STEPS FOR PROGRAM 'DRIVER'

1) ASSIGN FORTRAN UNIT 01 AS CREATED BY PROGRAM 'LOAD'
2) ASSIGN FORTRAN UNIT 02 AS CREATED BY PROGRAM 'LOAD'
3) ASSIGN FORTRAN UNIT 06 AS A DESIGNATED OUTPUT FILE

E.G.:
$ ASSIGN NETWORK.DAT FOR001
$ ASSIGN TRAFFIC.DAT FOR002
$ ASSIGN OUTPUT.DAT FOR006

PROGRAM DRIVER

LOAD FORTRAN UNIT 01 AND FORTRAN UNIT 02 FROM DISK AS CREATED FROM PROGRAM 'SETUP'

INCLUDE 'PARAM.DIM'
INCLUDE 'PATHS.BLK'
INCLUDE 'NETWRK.PRM'
INCLUDE 'CONVRG.PRM'
INTEGER COMMODITY,ORIGIN,DESTOD,OD,PAT
CALL LOAD

EXECUTE THE 'MULTIFLO' NETWORK ALGORITHM. 'MULTIFLO' SCHEDULES ITS OWN OUTPUTS TO FORTRAN UNIT 06 ON EACH ITERATION

INITIALIZE THE TIMER
CALL LIB$INIT_TIMER
CALL MULTIFLO

RECORD THE COMPUTATION TIME
CALL LIB$SHOW_TIMER

PRINT MAX LINK UTILIZATION (RELEVANT FOR M/M/1 QUEUEING DELAY OPTIMIZATION)

UMAX=0.0
DO 100 I=I,NA
   UMAX=M MAX(UMAX,FA(I)/BITRATE(I))
CONTINUE
WRITE(6,*)'MAXIMUM LINK UTILIZATION'
WRITE(6,*)UMAX

PRINT FINAL PATH FLOW INFO

WRITE(6,*)'ORIGIN / DESTINATION / PATH # / PATH FLOW'
DO 1000 COMMODITY=1,NUMCOMMOD
   ORIGIN=ORGID(COMMODITY)
   DO 500 OD=STARTOD(COMMODITY),STARTOD(COMMODITY+1)-1
DESTOD=DEST(OD)
PATH=OD
DO WHILE (PATH.GT.0)
   WRITE (6,*) ORIGIN, DESTOD, PATH, FP(PATH)
   PATH=NEXTPATH(PATH)
END DO
500    CONTINUE
1000   CONTINUE
STOP
END
LOAD

'LOAD' READS IN DATA FROM DISK CREATED WITH PROGRAM 'SETUP' FOR USE BY PROGRAM 'MULTIFLO'. NETWORK SPECIFICATION DATA RESIDES ON FORTRAN UNIT 01 AND NETWORK TRAFFIC SPECIFICATION DATA RESIDES ON FORTRAN UNIT 02.

SUBROUTINE LOAD
IMPLICIT NONE

** INCLUDE COMMON BLOCKS ********

INCLUDE 'PARAM.DIM'
INCLUDE 'NETRK.PRM'
INCLUDE 'CONVRG.PRM'

** LOCAL VARIABLE DEFINITIONS ********

INTEGER I
DO LOOP INDEX

** EXECUTABLE CODE ********

TERMINATION PARAMETERS. MAXITER GIVES THE MAX # OF ITERATIONS
TOL IS A SOLUTION ACCURACY TOLERANCE. RECOMMENDED VALUES ARE 0.01 TO 0.0001. THE PROPER VALUE OF TOL IS LARGELY INDEPENDENT OF THE PROBLEM SIZE.
MAXITER=20
TOL=0.01

THE FOLLOWING PARAMETER MAKES SENSE ONLY FOR ROUTING PROBLEMS WHERE AN M/M/1 QUEUING FORMULA IS USED FOR DELAY.
IT GIVES THE THRESHOLD FRACTION OF CAPACITY BEYOND WHICH THE DELAY FORMULA IS TAKEN TO BE QUADRATIC.
MAXUTI=0.99

LOAD THE NETWORK CONFIGURATION FROM FORTRAN UNIT 01

NODE SPECIFICATIONS

READ(1,*)NN
DO I=1,NN
   READ(1,*)FRSTOU(I),LASTOU(I)
END DO

LINK SPECIFICATIONS

READ(1,*)NA

BITRATE(I) IS A PARAMETER ASSOCIATED WITH LINK I. IN THE DATA NETWORK ROUTING CONTEXT IT HAS THE MEANING OF TRANSMISSION CAPACITY OF LINK I.

DO I=1,NA
   READ(1,*)STARTNODE(I),ENDNODE(I),BITRATE(I)
END DO

INPUT COMMODITY DATA FROM FORTRAN UNIT 02
C

READ(2, *) NUMCOMMOD
DO I=1, NUMCOMMOD
    READ(2, *) ORGID(I), STARTOD(I)
END DO
READ(2, *) NUMODPAIR
DO I=1, NUMODPAIR
    READ(2, *) DEST(I), INPUT_FLOW(I)
END DO
RETURN
END
MULTIFLO

MULTICOMMODITY FLOW ALGORITHM BASED ON A PATH FLOW FORMULATION
UPDATES THE PATH FLOWS OF OD PAIRS ONE AT A TIME ACCORDING TO
AN ITERATION OF THE PROJECTION TYPE.

DEVELOPED BY DIMITRI BERTSEKAS, BOB GENDRON, AND WEI K TSAI

BASED ON THE PAPERS:

1) BERTSEKAS, D.P., "A CLASS OF OPTIMAL ROUTING ALGORITHMS
FOR COMMUNICATION NETWORKS", PROC. OF 5TH INTERNATIONAL
CONFERENCE ON COMPUTER COMMUNICATION (ICCC-80),
ATLANTA, GA., OCT. 1980, PP. 71-76.

2) BERTSEKAS, D.P. AND GAFFI, E.M., "PROJECTION METHODS
FOR VARIATIONAL INEQUALITIES WITH APPLICATION TO
THE TRAFFIC ASSIGNMENT PROBLEM", MATH. PROGR. STUDY, 17,
D.C. SORENSEN AND J.-B. WETS (EDS), NORTH-HOLLAND,
AMSTERDAM, 1982, PP. 139-159.

3) BERTSEKAS, D.P., "OPTIMAL ROUTING AND FLOW CONTROL
METHODS FOR COMMUNICATION NETWORKS", IN ANALYSIS AND
OPTIMIZATION OF SYSTEMS, (PROC. OF 5TH INTERNATIONAL
CONFERENCE ON ANALYSIS AND OPTIMIZATION, VERSAILLES,
FRANCE), A. BENSOUSSAN AND J.L. LIONS (EDS),

4) BERTSEKAS, D.P. AND GAFFI, E.M., "PROJECTED NEWTON
METHODS AND OPTIMIZATION OF MULTICOMMODITY FLOWS",
IEEE TRANSACTIONS ON AUTOMATIC CONTROL, DEC. 1983.

SUBROUTINE MULTIFLO

IMPLICIT NONE

********** INCLUDE COMMON BLOCKS **********

INCLUIDE 'PARAM.DIM'
INCLUIDE 'NETWRK.PRM'
INCLUIDE 'CONVRG.PRM'
INCLUIDE 'PATHS.BLK'

NODE ARRAYS (LENGTH NN):
FRSTOU(NODE) - FIRST ARC OUT OF NODE
LASTOU(NODE) - LAST ARC OUT OF NODE
NOTE: THE ARC LIST MUST BE ORDERED IN SEQUENCE SO
THAT ALL ARCS OUT OF ANY NODE ARE GROUPED TOGETHER

ARC ARRAYS (LENGTH NA):
FA(ARC) - THE TOTAL FLOW OF ARC
STARTNODE(ARC) - THE HEAD NODE OF ARC
ENDNODE(ARC) - THE TAIL NODE OF ARC
COMMODITY LENGTH ARRAYS (LENGTH NUMCOMMOD):

ORGID(COMMODITY) - THE NODE ID OF THE ORIGIN OF COMMODITY
STARTOD(COMMODITY) - THE STARTING OD PAIR IN THE ODPAIR LIST
CORRESPONDING TO THE ORIGIN IN POSITION RANK

NOTE: THIS SCHEME ASSUMES THAT OD PAIRS ARE LISTED IN SEQUENCE
I.E. THE OD PAIRS CORRESPONDING TO THE COMMODITY ONE
ARE LISTED FIRST. THEY ARE
FOLLOWED BY THE OD PAIRS OF THE COMMODITY TWO
AND SO ON.

ODPAIR ARRAYS (LENGTH NUMOD):
DEST(OD) - GIVES THE DESTINATION OF ODPAIR OD
INPUT_FLOW(OD) - GIVES THE INPUT TRAFFIC OF ODPAIR OD

PATH ARRAYS (LENGTH DYNAMICALLY UPDATED):
PATHID(PATH) - THE ITERATION # AT WHICH PATH WAS GENERATED
NEXTPATH(PATH) - THE NEXT PATH FOR THE SAME OD PAIR FOLLOWING
PATH. IT EQUALS 0 IF PATH IS THE LAST FOR THAT OD PAIR
FP(PATH) - THE FLOW CARRIED BY PATH

PATH DESCRIPTION LIST ARRAY (LENGTH MAXITER*NUMCOMD*NN)
FRED(NODE,ITER,COMMODITY) - THIS TRIPLE INDEXED ARRAY SPECIFIES THE
SHORTEST PATH TREE GENERATED AT ITERATION ITER
& CORRESPONDING TO THE ORIGIN ASSOCIATED W/ COMMODITY
IT GIVES THE LAST ARC ON THE SHORTEST PATH FROM ORIGIN TO NODE.

*************** LOCAL VARIABLE DEFINITIONS **********************

INTEGER*2 FRED(NNN,NMAXITER,NNORIG)
PATH DESCRIPTION ARRAY - CONTAINS SHORTEST
PATH TREES FOR ALL ITERATIONS

LOGICAL SPNEW
LOGICAL INDICATING A NEW PATH FOUND

LOGICAL SAME
LOGICAL INDICATING A NEW SHORTEST PATH A\X\EADY EXISTING

INTEGER NODE
NODE IDENTIFIER

INTEGER DESTOD
THE DESTINATION NODE OF AN OD PAIR

INTEGER ARC
DO LOOP INDEX FOR ARCS

INTEGER PATH
A PATH INDEX

INTEGER NUMLIST
TOTAL NUMBER OF ACTIVE PATHS FOR OD PAIR UNDER CONSIDERATION

INTEGER ITER
SPECIFIC ITERATION

INTEGER N1,N2
TEMPORARY VARIABLES

REAL MFINDER
THE LENGTH FOR A SHORTEST PATH

REAL MINSDER
THE SECOND DERIVATIVE LENGTH FOR THE SHORTEST PATH

REAL TMINSDER
TEMPORARY VALUE FOR SECOND DERIVATIVE LENGTH OF SHORTEST PATH

REAL INCR
TEMPORARY VALUE FOR SECOND DERIVATIVE LENGTH OF SHORTEST PATH

REAL PATHINCR
SHIFT OF FLOW FOR A GIVEN PATH
REAL FLOW
C FLOW FOR A PATH
REAL FDER
C THE ACCRURED LENGTH ALONG A PATH
REAL SDER
C THE ACCRUED SECOND DERIVATIVE LENGTH ALONG A PATH
REAL TEMPERRO
C TEMPORARY STORAGE FOR CONVERGENCE ERROR
REAL EDLENGTH(NMAXITER)
C ARRAY OF LENGTHS OF PATHS FOR AN OD PAIR
REAL SDLENGTH(NMAXITER)
C ARRAY OF SECOND DERIVATIVE LENGTHS OF PATHS FOR AN OD PAIR
INTEGER PATHLIST(NMAXITER)
C ARRAY OF ACTIVE PATHS FOR AN OD PAIR
INTEGER COMMODITY
C DO LOOP INDEX FOR THE OD PAIR ORIGINS
INTEGER ORIGIN
C SPECIFIC ORIGIN
INTEGER I
C DO LOOP INDEX
INTEGER OD
C OD DO LOOP INDEX
INTEGER K
C DO LOOP INDEX
INTEGER SHORTEST
C THE SHORTEST PATH
LOGICAL MEMBER(NNA)
C LOGICAL FOR AN ARC INCLUDED IN THE SHORTEST PATH
REAL DLENGTH
C DIFFERENCE IN PATH LENGTHS FOR THE TRAFFIC
REAL DICAL
C ARC LENGTH
REAL D2CAL
C DERIVATIVE OF ARC LENGTH

****************************************************************************** EXECUTABLE CODE ******************************************************************************

****************************************************************************** INITIALIALIZATION ******************************************************************************

DO 5 ARC=1,NA
   FA(ARC)=0.0
5 CONTINUE
DO I=1,NUMODPAIR
   FP(I)=INPUT_FLOW(I)
ENDDO
STARTOD(NUMCOMMOD+1)=NUMODPAIR+1
NUMPATH=0
NUMITER=1
DO 100 COMMODITY=1,NUMCOMMOD
   ORIGIN=ORGID(COMMODITY)
   CALL SP(ORIGIN,COMMODITY)
   DO 10 I=1,NN
      PRED(I,1,COMMODITY)=PA(I)
   10 CONTINUE

LOOP OVER OD PAIRS OF COMMODITY
N1=STARTOD(COMMODITY)
N2=STARTOD(COMMODITY+1)-1
DO 50 OD=N1,N2
  NUMPATH=NUMPATH+1
  PATHID(NUMPATH)=1
  NEXTPATH(NUMPATH)=0
  FLOW=FP(NUMPATH)
  NODE=DEST(OD)
  DO WHILE (NODE.NE.ORIGIN)
    ARC=P(A(NODE))
    FA(ARC)=FA(ARC)+FLOW
    NODE=STARTNODE(ARC)
  END DO
50 CONTINUE
100 CONTINUE

C C INITIALIZE THE MEMBER ARRAY
C DO 70 ARC=1,NA
  MEMBER(ARC)=.FALSE.
70 CONTINUE

C C INITIALIZE THE TOTAL DELAY
C CALL DELAY(DTOT(NUMITER))

C C OUTPUT THE CURRENT INFORMATION TO DISK
C CALL PRFLOW

C ********************************************
C * END OF INITIALIZATION
C ********************************************

C ***** START NEW ITERATION *****

110 NUMITER=NUMITER+1
  CURERROR=0

C **** LOOP OVER ALL COMMODITIES ****
D 1000 COMMODITY=1,NUMCOMMOD
  ORIGIN=ORCID(COMMODITY)
  CALL SP(ORIGIN,COMMODITY)
  DO 150 I=1,NN
    PRED(I,NUMITER,COMMODITY)=PA(I)
  CONTINUE
150

C **** LOOP OVER OD PAIRS OF COMMODITY

C N1=STARTOD(COMMODITY)
N2=STARTOD(COMMODITY+1)-1
DO 500 OD=N1,N2
C C CHECK IF THERE IS ONLY ONE ACTIVE PATH AND IF SO SKIP
C THE ITERATION
C IF (NEXTPATH(OD).EQ.0) THEN
  NODE=DEST(OD)
  DO WHILE (NODE.NE.ORIGIN)
ARC=PA(NODE)
IF (ARC.NE.PRED(NODE,1,COMMODITY)) GO TO 180
NODE=STARTNODE(ARC)
END DO
GO TO 500
END IF

C

CONTINUE

C

MARK THE ARCS OF THE SHORTEST PATH

C

DESTOD=DEST(OD)
NODE=DESTOD
DO WHILE (NODE.NE.ORIGIN)
   ARC=PA(NODE)
   MEMBER(ARC) = .TRUE.
   NODE=STARTNODE(ARC)
END DO

C

GENERATE LIST OF ACTIVE PATHS FOR OD PAIR

C

NUMLIST=1
PATHLIST(1)=OD
PATH=NEXTPATH(OD)
DO WHILE (PATH.GT.0)
   NUMLIST=NUMLIST+1
   PATHLIST(NUMLIST)=PATH
   PATH=NEXTPATH(PATH)
END DO

C

DETERMINE 1ST & 2ND DERIVATIVE LENGTH OF ACTIVE PATHS

C
ALSO DETERMINE WHETHER THE CALCULATED SHORTEST PATH

C
IS ALREADY IN THE LIST

C

SPNEW=.TRUE.
DO 200 K=1,NUMLIST
   SAME=.TRUE.
   FDER=0
   SDER=0
   TMINSDER=0
   PATH=PATHLIST(K)
   ITER=PATHID(PATH)
   NODE=DESTOD
   DO WHILE (NODE.NE.ORIGIN)
      ARC=PRED(NODE,ITER,COMMODITY)
      CALL DERIVS(COMMODITY,FA(ARC),ARC,D1CAL,D2CAL)
      FDER=FDER+D1CAL
      IF (.NOT.MEMBER(ARC)) THEN
         SDER=SDER4D2CAL
         SAME=.FALSE.
      ELSE
         SDER=SDER4D2CAL
         TMINSDER=TMINSDER+D2CAL
      END IF
      NODE=STARTNODE(ARC)
   END DO
   IF (SAME) THEN
      SPNEW=.FALSE.
      SHORTEST=PATH
      FDLENGTH(K)=FDER
   IF (SAME) THEN
      SPNEW=.FALSE.
      SHORTEST=PATH
      FDLENGTH(K)=FDER
   END IF

END DO

200
MINFDER=FDER
MINSDER=TMINSDER
ELSE
FDLENGTH(K)=FDER
SDLENGTH(K)=SDER
END

200 CONTINUE

*** INSERT SHORTEST PATH IN PATH LIST IF IT IS NEW ***

IF (SPNEW) THEN
NUMPATH=NUMPATH+1
SHORTEST=NUMPATH
PATHID(NUMPATH)=NUMITER
NEXTPATH(PATHLIST(NUMLIST))=NUMPATH
NEXTPATH(NUMPATH)=0
MINFDER=0
MINSDER=0
NODE=DESTOD
DO WHILE (NODE.NE.ORIGIN)
ARC=PA(NODE)
CALL DERIVS(COMMODITY,FA(ARC),ARC,D1CAL,D2CAL)
MINFDER=MINFDER+D1CAL
MINSDER=MINSDER+D2CAL
NODE=STARTNODE(ARC)
END DO
END IF

**** UPDATE PATH & LINK FLOWS ****

INCR=0
TEMPERROR=0
DO 250 K=1,NUMLIST
DLENGTH=FDLENGTH(K)-MINFDER
IF (DLENGTH.GT.0) THEN
PATH=PATHLIST(K)
FLOW=FP(PATH)
IF ((FLOW.EQ.0.0).AND.(K.GT.1)) THEN
NEXTPATH(PATHLIST(K-i))=NEXTPATH(PATH)
GO TO 250
END IF
PATHINCR=DLENGTH/(SDLENGTH(K)+MINSDER)
IF (FLOW.LE.PATHINCR) THEN
FP(PATH)=0.0
PATHINCR=FLOW
ELSE
FP(PATH)=FLOW-PATHINCR
END IF
INCR=INCR+PATHINCR
TEMPERROR=TEMPERROR+FLOW*DLENGTH/FDLENGTH(PATH)
ITER=PATHID(PATH)
NODE=DESTOD
DO WHILE (NODE.NE.ORIGIN)
ARC=PRED(NODE,ITER,COMMODITY)
FA(ARC)=FA(ARC)-PATHINCR
NODE=STARTNODE(ARC)
END DO
END IF

250 CONTINUE
*** UPDATE THE ERROR CRITERION ***

CURERROR=AMAX1(CURERROR,TEMPERROR/INPUT_FLOW(OD))

**** UPDATE FLOWS FOR SHORTEST PATH ****

FP(SHORTEST)=FP(SHORTEST)+INCR
NODE=DESTOD
DO WHILE (NODE.NE.ORIGIN)
ARC=PA(NODE)
FA(ARC)=FA(ARC)+INCR
MEMBER(ARC)=.FALSE.
NODE=STARTNODE(ARC)
END DO

500 CONTINUE

***** END OF LOOP FOR OD PAIRS CORRESPONDING TO COMMODITY
***** UPDATE TOTAL DELAY

CALL DELAY(DTOT(NUMITER))

1000 CONTINUE

CHECK IF THE # OF ACTIVE PATHS EXCEED THE ALLOCATED NUMBER
IF (NUMPATH.GT.NNUMPATH) THEN
WRITE(6,*)'MAX # OF ALLOCATED PATHS EXCEEDED'
STOP
END IF

OUTPUT THE CURRENT SOLUTION TO DISK
CALL PRFLOW

***** END OF ITERATION *****

*** IF THE ERROR IS SMALLER THAN TOL, OR THE LIMIT ON
 THE NUMBER OF ITERATIONS IS REACHED RETURN
ELSE GO FOR ANOTHER ITERATION

IF ((CURERROR.LT.TOL).OR.(NUMITER.EQ.MAXITER)) THEN
RETURN
ELSE
GO TO 110
END IF

END

************** END OF MULTIFLO **************
SHORTHEAP

'SHORTHEAP' SOLVES THE SHORTEST PATH PROBLEM BY
DIJKSTRA'S ALGORITHM AND A HEAP DATA STRUCTURE.
THIS ALGORITHM SHOULD BE USED WHEN THE NUMBER OF
DESTINATIONS FOR EACH COMMODITY IS SMALL RELATIVE
TO THE TOTAL NUMBER OF NODES.

INPUT:
S - THE STARTING NODE
COMMODITY - THE CORRESPONDING COMMODITY

OUTPUT:
PA(I) - THE LAST ARC ON THE SHORTEST PATH ENDING AT NODE I
DIST(I) - THE SHORTEST DISTANCE TO NODE I

SUBROUTINE SP(S,COMMODITY)
IMPLICIT NONE

************** INCLUDE COMMON BLOCKS **************
INCLUDE 'PARAM.DIM'
INCLUDE 'NETWRK.PRM'
INCLUDE 'PATHS.BLK'

************** LOCAL VARIABLE DEFINITIONS **************

REAL MIN
TEMPORARY MINIMUM VALUE
REAL D1,D2,DP
NODE DISTANCE
REAL XLARGE
BIG X BY DEFAULT
INTEGER S
INPUT NODE
INTEGER COMMODITY
INPUT COMMODITY
INTEGER P
NODE ALONG THE PATH OF S TO DESTINATIONS
INTEGER I
DO LOOP INDEX
INTEGER J
DO LOOP INDEX
INTEGER ARC
DO LOOP INDEX
INTEGER ND
A NODE INDEX
INTEGER DNUMBER
# OF DESTINATIONS FOR COMMODITY
INTEGER N1
TEMPORARY VARIABLE
INTEGER N2
TEMPORARY VARIABLE
INTEGER UPNODE,DOWNNODE,DOWNNODE1,LASTNODE
VARIABLES USED IN UPDATING THE HEAP ARRAY
INTEGER CURRANK,NEWRANK
C VARIABLES USED IN UPDATING THE HEAP ARRAY
C INTEGER ENDSHEAP
C MARKS THE LAST ELEMENT OF THE HEAP ARRAY
C INTEGER RANK(NNN)
C RANK(NODE) GIVES THE RANK OF NODE IN THE HEAP
C INTEGER NRANK(NNN)
C NRANK(I) GIVES THE NODE OF RANK I IN THE HEAP
C REAL DICAL
C FIRST DERIVATIVE OF DELAY WITH RESPECT TO LOAD
C LOGICAL FIRSTITER
C TRUE IF THIS IS THE FIRST ITERATION
C LOGICAL SCAN(NNN)
C LOGICAL INDICATING THAT A NODE HAS BEEN SCANNED
C LOGICAL DSTATUS(NNN)
C LOGICAL SPECIFYING IF A NODE IS A DESTINATION
C
C EXECUTABLE CODE
C
C XLARGE=1E15
C DICAL=1.0
C P=S
C DO 10 I=1,NN
C DIST(I)=XLARGE
C SCAN(I)=.FALSE.
C DSTATUS(I)=.FALSE.
C 10 CONTINUE
C DIST(S)=0
C IF (NUMITER.EQ.1) THEN
C FIRSTITER=.TRUE.
C ELSE
C FIRSTITER=.FALSE.
C END IF
C MARK THE DESTINATION NODES
C N1=STARTOD(COMMODITY)
C N2=STARTOD(COMMODITY+1)-1
C DNUMBER=N2-N1+1
C DO 15 I=N1,N2
C DSTATUS(DEST(I))=.TRUE.
C CONTINUE
C INITIALIZE THE HEAP FLOOR
C ENDHEAP=0
C
C SCAN NODE P
C
C CONTINUE
C SCAN(P)=.TRUE.
C IF (DSTATUS(P)) THEN
C IF (DNUMBER.EQ.1) RETURN
C DNUMBER=DNUMBER-1
C END IF
C IF (FRSTOU(P).NE.0) THEN
C DP=DIST(P)
C DO 20 ARC=FRSTOU(P),LASTOU(P)
C ND=ENDNODE(ARC)
C IF (.NOT.SCAN(ND)) THEN
C IF (.NOT.FIRSTITER) THEN
CALL DERIV1(COMMODITY,FA(ARC),ARC,D1CAL)
END IF
D2=DIST(ND)

C IF ND HAS NOT BEEN LABELLED INSERT IT IN THE HEAP
IF (D2.EQ.XLARGE) THEN
ENDHEAP=ENDHEAP+1
RANK(ND)=ENDHEAP
NRANK(ENDHEAP)=ND
END IF
D1=DP+D1CAL
IF (D1.LT.D2) THEN
PA(ND)=ARC
DIST(ND)=D1
CURRANK=RANK(ND)
END IF
50
NEWRANK=INT(CURRANK/2)
IF (NEWRANK.GE.1) THEN
UPNODE=NRANK(NEWRANK)
IF (D1.LT.DIST(UPNODE)) THEN
NRANK(CURRANK)=UPNODE
RANK(UPNODE)=CURRANK
CURRANK=NEWRANK
GO TO 50
END IF
END IF
NRANK(CURRANK)=ND
RANK(ND)=CURRANK
END IF
END IF
20 CONTINUE
END IF
C
C ******** FIND NEXT NODE TO SCAN ********
C
TEST FOR ERROR
IF (ENDHEAP.EQ.0) THEN
WRITE(6,*) 'ERROR IN THE SHORTEST PATH POUTINE'
STOP
END IF
P=NRANK(1)
C
RESTRUCTURE HEAP ARRAYS

C
LASTNODE=NRANK(ENDHEAP)
ENDHEAP=ENDHEAP-1
D1=DIST(LASTNODE)
CURRANK=1
100
NEWRANK=CURRANK+CURRANK
CURRANK=1
IF (NEWRANK.LE.ENDHEAP) THEN
DOWNNODE=NRANK(NEWRANK)
IF (NEWRANK.EQ.ENDHEAP) THEN
DOWNNODE1=DOWNNODE
ELSE
DOWNNODE1=NRANK(NEWRANK+1)
END IF
IF (DIST(DOWNNODE).LE.DIST(DOWNNODE1)) THEN
IF (D1.GT.DIST(DOWNNODE)) THEN
NRANK(CURRANK)=DOWNNODE
RANK(DOWNNODE)=CURRANK
CURRANK=NEWRANK
GO TO 100
ENDIF
ELSE
  IF (D1.GT.DIST(DOWNNODE1)) THEN
    NRANK(CURRANK) = DOWNNODE1
    RANK(DOWNNODE1) = CURRANK
    CURRANK = NEWRANK + 1
    GO TO 100
  END IF
ENDIF
ENDIF
NRANK(CURRANK) = LASTNODE
RANK(LASTNODE) = CURRANK
GO TO 1000
END
'SHORTPAPE' solves the shortest path problem by Pepe's modification of Bellman's algorithm.

**INPUT:**
- S - The starting node
- commodity - The corresponding commodity

**OUTPUT:**
- PA(I) - The last arc on the shortest path ending at node I
- DIST(I) - The shortest distance to node I

---

SUBROUTINE SP(S, COMMODITY)

IMPLICIT NONE

INCLUDE COMMON BLOCKS

INCLUDE 'PARAM.DIM'
INCLUDE 'NETWRK.PRM'
INCLUDE 'PATHS.BLK'

****************
LOCAL VARIABLE DEFINITIONS
****************

REAL D1, DP
NODE DISTANCE
REAL XLARGE
BIG X BY DEFAULT
INTEGER ILARGE
INTEGER LARGER THAN THE NUMBER OF NODES
INTEGER S
INPUT NODE
INTEGER COMMODITY
INPUT COMmodity
INTEGER P
NODE PRESENTLY SCANNED
INTEGER I
DO LOOP INDEX
INTEGER ARC
DO LOOP INDEX
INTEGER ND
A NODE INDEX
INTEGER N1
TEMPORARY VARIABLE
INTEGER N2
TEMPORARY VARIABLE
INTEGER ENQUEUE
MARKS THE LAST ELEMENT OF THE QUEUE ARRAY
REAL DICAL
FIRST DERIVATIVE OF DELAY WITH RESPECT TO FLOW
LOGICAL FIRSTITER
TRUE IF THIS IS THE FIRST ITERATION
INTEGER Q(NNN)
QUEUE OF NODES TO BE SCANNED

EXECUTABLE CODE

---
XIARGE=1E15
1LARGE=NNN+1
DICAL=1.0
DO 10 I=1,NN
   DIST(I)=XIARGE
   Q(I)=0
10 CONTINUE
IF (NUMITER.EQ.1) THEN
   FIRSTITER=.TRUE.
ELSE
   FIRSTITER=.FALSE.
END IF
DIST(S)=0
Q(S)=ILARGE
ENDQUEUE=S
P=S

********** START OF MAIN ALGORITHM **********
100 CONTINUE

***** SCAN NODE P *****

N1=FRSTOU(P)
IF (N1.EQ.0) GO TO 201
N2=LASTOU(P)
DP=DIST(P)
DO 200 ARC=N1,N2
   ND=ENDNODE(ARC)
   IF (.NOT.FIRSTITER) THEN
      CALL DERIV1(COMMODITY,FA(ARC),ARC,DICAL)
   END IF
   DI=DP+DICAL
   IF (D1.GE.DIST(ND)) GO TO 200
   PA(!-)=ARC
   DIST(ND)=D1
   IF (Q(ND)) 160,140,200
   Q(ENDQUEUE)=ND
   ENDQUEUE=ND
   Q(ND)=ILARGE
   GO TO 200
140 Q(ND)=Q(P)
   Q(P)=ND
   IF (ENDQUEUE.EQ.P) ENDQUEUE=ND
200 CONTINUE

*** IF ND HAS ALREADY BEEN SCANNED ADD IT AT THE BEGINNING OF THE QUEUE ***
160 Q(P)=Q(ND)
   Q(ND)=P
   IF (ENDQUEUE.EQ.P) ENDQUEUE=ND
CONTINUE

*** GET NEXT NODE FROM THE TOP OF THE QUEUE ***
201 N1=Q(P)

*** FLAG P AS HAVING BEEN SCANNED ***
Q(P)=-1
P=N1
*** IF THE QUEUE IS NOT EMPTY GO BACK TO SCAN NEXT NODE ***
IF (P.LT.ILARGE) GO TO 100
RETURN
END
SUBROUTINE DELAY (DT)
IMPLICIT NONE

*************** INCLUDE COMMON BLOCKS ***************

INCLUDE 'PARAM.DIM'
INCLUDE 'PATHS.BLK'
INCLUDE 'NETWRK.PRM'
INCLUDE 'CONVRG.PRM'

*************** ARGUMENT DEFINITIONS ***************

ON OUTPUT:

REAL DT
TOTAL SYSTEM DELAY

*************** EXTERNAL FUNCTIONS REFERENCED ***************

REAL DCAL
DELAY AS A FUNCTION OF FLOW

*************** LOCAL VARIABLE DEFINITIONS ***************

INTEGER K
DO LOOP INDEX

*************** EXECUTABLE CODE ***************

LOOP OVER ALL LINKS AND ACCRUE TOTAL DELAY

DT=0.
DO 50 K=1,NA
   DT=DT+DCAL(FA(K), K)
CONTINUE

RETURN
END
DCAL

'DCAL' COMPUTES THE DELAY ACROSS A SPECIFIED ARC GIVEN THE FLOW. THE DELAY IS ASSUMED TO BE CONSISTENT WITH M/M/1 QUEUEING FOR FLOWS BELOW A MAXIMUM UTILIZATION AND QUADRATIC BEYOND WITH CONTINUITY IN THE DERIVATIVES AT THE MAXIMUM UTILIZATION.

REAL FUNCTION DCAL(X,ARC)
IMPLICIT NONE

*************** INCLUDE COMMON BLOCKS ***************

INCLUDE 'PARAM.DIM'
INCLUDE 'NETWRK.PRM'
INCLUDE 'CONVRG.PRM'
INCLUDE 'PATHS.BLK'

*************** ARGUMENT DEFINITIONS ***************

REAL X
INPUT FLOW FOR THE ARC

INTEGER ARC
INPUT ARC

*************** LOCAL VARIABLE DEFINITIONS ***************

REAL RATE
MAXIMUM LINK CAPACITY

REAL Y
TEMPORARY VARIABLE

REAL Z
TEMPORARY VARIABLE

REAL Q0
ZEROTH ORDER TERM IN THE QUADRATIC APPROXIMATION FOR OVERLOADED LINKS

REAL Q1
FIRST ORDER TERM IN THE QUADRATIC APPROXIMATION

REAL Q2
SECOND ORDER TERM IN THE QUADRATIC APPROXIMATION

REAL EXCESS
FLOW BEYOND THE MAXIMUM ALLOWABLE UTILIZATION

*************** EXECUTABLE CODE ***************

RATE=BITRATE(ARC)
Y=MAXUTI*RATE

M/M/1 DELAY

IF (X.LT.Y) THEN
   DCAL=X/(RATE-X)
ELSE
   QUADRATIC APPROXIMATION TO AVOID OVERFLOWS
   EXCESS=X-Y
Z = RATE - Y
Q0 = Y / Z
Q1 = Q0 / (MAXUTI * Z)
Q2 = Q1 / Z
DCAL = Q0 + Q1 * EXCESS + Q2 * EXCESS ** 2
ENDIF
RETURN
END

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C
C 'DERIVS' COMPUTES THE DERIVATIVES OF DELAY WITH RESPECT TO FLOW FOR
C LINKS. BELOW A MAXIMUM UTILIZATION, M/M/1 DELAY IS ASSUMED TO APPLY
C WHEREAS A QUADRATIC APPROXIMATION IS ASSUMED FOR UTILIZATIONS BEYOND
C THE MAXIMUM. THE DERIVATIVES ARE CONTINUOUS AT THE MAXIMUM
C UTILIZATION.
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
SUBROUTINE DERIVS (COMMODITY, X, ARC, DICAL, D2CAL)
IMPLICIT NONE
C
C ********************
INCLUDE COMMON BLOCKS ,
C
INCLUDE 'PARAM.DIM'
INCLUDE 'NETWRK.PRM'
INCLUDE 'CONVRG.PRM'
INCLUDE 'PATHS.BLK'
C
C
C ********************
ARGUMENT DEFINITIONS
C
C ON INPUT:
INTEGER COMMODITY
THE CORRESPONDING COMMODITY
C
REAL X
FLOW IN THE SPECIFIED LINK
C
INTEGER ARC
THE SPECIFIED LINK
C
ON OUTPUT:
C
REAL DICAL
ARC LENGTH (1ST DERIVATIVE OF DELAY)
C
REAL D2CAL
FIRST DERIVATIVE OF ARC LENGTH
C
C
C
C
C

*************** LOCAL VARIABLE DEFINITIONS ***************
C
REAL MAXI
MAXIMUM ALLOWABLE FLOW FOR LINK FOR M/M/1 QUEUEING DELAY
C
REAL RATE
THE MAXIMUM FLOW CAPACITY FOR THE LINK
C
REAL EXCESS
FLOW BEYOND THE MAXIMUM ALLOWABLE FLOW
C
REAL D1
TEMPORARY VARIABLE
C
REAL T
TEMPORARY VARIABLE

*************** INCLUDE COMMON BLOCKS ***************
**EXECUTABLE CODE**

RATE=BITRATE(ARC)
MAXI=MAXUTI RATE
EXCESS=X-MAXI

IF(EXCESS.LE.0.0) THEN

DERIVATIVES OF M/M/1 QUEUEING DELAY

T=RATE-X
D1CAL=RATE/T**2
D2CAL=2.0*D1CAL/T

ELSE

DERIVATIVES OF THE QUADRATIC APPROXIMATION

T=RATE-MAXI
D1=RATE/T**2
D2CAL=2.0*D1/T
D1CAL=D1+D2CAL*EXCESS

END IF
RETURN
END

---

'DERIV1' COMPUTES THE FIRST DERIVATIVE OF DELAY WITH RESPECT TO FLOW FOR LINKS. BELOW A MAXIMUM UTILIZATION, M/M/1 DELAY IS ASSUMED TO APPLY WHEREAS A QUADRATIC APPROXIMATION IS ASSUMED FOR UTILIZATIONS BEYOND THE MAXIMUM. THE DERIVATIVES ARE CONTINUOUS AT THE MAXIMUM UTILIZATION.

SUBROUTINE DERIV1(COMMODITY,X,ARC,D1CAL)
IMPLICIT NONE

**INCLUDE COMMON BLOCKS**

INCLUDE 'PARAM.DIM'
INCLUDE 'NETWRK.PRM'
INCLUDE 'CONVRG.PRM'
INCLUDE 'PATHS.BLK'

**ARGUMENT DEFINITIONS**

ON INPUT:

INTEGER COMMODITY
THE CORRESPONDING COMMODITY

REAL X
FLOW IN THE SPECIFIED LINK

INTEGER ARC
THE SPECIFIED ARC

ON OUTPUT:
REAL DICAL  
ARC LENGTH (1ST DERIVATIVE OF DELAY)  

*************** LOCAL VARIABLE DEFINITIONS  ***************  

REAL MAXI  
MAXIMUM ALLOWABLE FLOW FOR LINK FOR M/M/1 QUEUEING DELAY  
REAL RATE  
The maximum flow capacity for the link  
REAL EXCESS  
FLOW BEYOND THE MAXIMUM ALLOWABLE FLOW  
REAL D1  
TEMPORARY VARIABLE  
REAL T  
TEMPORARY VARIABLE  
REAL D2CAL  
TEMPORARY VARIABLE  

*************** EXECUTABLE CODE  ***************  

RATE=BITRATE(ARC)  
MAXI=MAXUTI*RATE  
EXCESS=X-MAXI  

IF (EXCESS.LE.0.0) THEN  

DERIVATIVE OF M/M/1 QUEUEING DELAY  
T=RATE-X  
D1CAL=RATE/T**2  
ELSE  

DERIVATIVE OF THE QUADRATIC APPROXIMATION  
END IF  
RETURN  
END
`PRFLOW` outputs intermediate results in the Multiflo algorithm. Iteration #, delay, number of active paths generated and convergence are the primary outputs.

Subroutine PRFLOW

IMPLICIT NONE

C INCLUDE COMMON BLOCKS '-----'
INCLUDE 'PARAM.DIM'
INCLUDE 'NETWRK.PRM'
INCLUDE 'CONVRG.PRM'
INCLUDE 'PATHS.BLK'

************ LOCAL VARIABLE DEFINITIONS ************

LOGICAL FIRFLG
FIRST PASS FLAG FOR OUTPUT CONTROL
INTEGER I
DO LOOP INDEX

************ LOCAL DATA INITIALIZATION ************

DATA FIRFLG/.TRUE./

ON THE VERY FIRST PASS, OUTPUT THE CONTENTS OF INPUT BLOCKS TO FILE

IF(FIRFLG) THEN
WRITE(6,*)'***********************************************************
WRITE(6,*)'MULTIFLO SUMMARY '*
WRITE(6,*)'***********************************************************
WRITE(6,*)'
WRITE(6,*)'***********************************************************
WRITE(6,*)'INITIALIZATION DATA '*
WRITE(6,*)'***********************************************************
WRITE(6,*)'
WRITE(6,*)'NETWORK SPECIFICATION DATA:'
WRITE(6,*)'
WRITE(6,*)'NODE SPECIFICATIONS'
WRITE(6,*)'NUMBER OF NODES:',NN
WRITE(6,*)'NODE # FRSTOU LASTOU'
DO I=1,NN
WRITE(6,*)I,FRSTOU(I),LASTOU(I)
END DO
WRITE(6,*)'
WRITE(6,*)'LINK SPECIFICATIONS:'
WRITE(6,*)'NUMBER OF LINKS:',NA
WRITE(6,*)'LINK # STARTNODE ENDNODE BITRATE'
DO I=1,NA
WRITE(6,*)I,STARTNODE(I),ENDNODE(I),BITRATE(I)
END DO
WRITE(6,*)'
WRITE(6,*)'COMMODITY SPECIFICATIONS'
WRITE(6,*)'NUMBER OF COMMODITIES:',NUMCOMMOD
WRITE(6,*)'COMMOD #    ORGID    STARTOD'
DO I=1,NUMCOMMOD
   WRITE(6,*)I,ORGID(I),STARTOD(I)
END DO
WRITE(6,*)'
WRITE(6,*)'OD PAIR SPECIFICATIONS'
WRITE(6,*)'NUMBER OF OD PAIRS: ',NUMODPAIR
WRITE(6,*)'OD PAIR #    DEST INPUT FLOW'
DO I=1,NUMODPAIR
   WRITE(6,*)I,DEST(I),INPUT_FLOW(I)
END DO
WRITE(6,*)'
WRITE(6,*)'********************************************************************'
WRITE(6,*)'* MULTIFLO DATA BY ITERATION '*
WRITE(6,*)'********************************************************************'
WRITE(6,*)'ITERATION #    TOTAL DELAY    CONVERGENCE    NUMBER OF'
WRITE(6,*)'ERROR    ACTIVE    PATHS'
FIRFLG=.FALSE.
END IF
IF (NUMITER.GT.0) THEN
   WRITE(6,*)NUMITER,DTOT(NUMITER),CURERROR,NUMPATH
END IF
RETURN
END
'INCLUDE' FILE PARAM.DIM

'PARAM.DIM' CONTAINS THE ARRAY DIMENSIONS

*************** NETWORK PARAMETERS ***************

PARAMETER NNN=100
        MAXIMUM NUMBER OF NODES
PARAMETER NNA=500
        MAXIMUM NUMBER OF ARCS
PARAMETER NNUMOD=1000
        MAXIMUM NUMBER OF OD PAIRS
PARAMETER NNUMPATH=10000
        MAXIMUM NUMBER OF PATHS FOR CONSIDERATION
PARAMETER NMAXITER=50
        MAXIMUM NUMBER OF ITERATIONS ALLOWED
PARAMETER NNORIG=100
        MAXIMUM NUMBER OF COMMODITIES
PARAMETER NINDEX=100000
        MAXIMUM NUMBER OF ELEMENTS OF PATH
PARAMETER DESCRIPTION ARRAY (USED IN MULTIFLO1)
'INCLUDE' FILE NETWRK.PRM

'NETWRK.PRM' CONTAINS THE NETWORK SPECIFICATION PARAMETERS

COMMON /NETWORK/
  &  NN,FRSTOU,LASTOU,
  &  NA,STARTNODE,ENDNODE,BITRATE,
  &  NUMCOMMOD,ORGID,STARTOD,
  &  NUMODPAIR,DEST,INPUT_FLOW

INTEGER*2  NN
  NUMBER OF NODES IN THE NETWORK

INTEGER*2  FRSTOU(NNN)
  THE FIRST ARC EMANATING FROM A NODE

INTEGER*2  LASTOU(NNN)
  THE FINAL ARC EMANATING FROM A NODE

INTEGER*2  NA
  NUMBER OF LINKS (ARCS) IN THE NETWORK

INTEGER*2  STARTNODE(NNA)
  THE START NODE FOR AN ARC

INTEGER*2  ENDNODE(NNA)
  THE END NODE FOR AN ARC

REAL  BITRATE(NNA)
  THE LINK CAPACITY IN BITS/SECOND

INTEGER*2  NUMCOMMOD
  THE NUMBER OF COMMODITIES IN THE NETWORK

INTEGER*2  ORGID(NNORIG)
  THE NODE NUMBER OF THE ORIGIN

INTEGER*2  STARTOD(NNORIG)
  THE POINTER TO THE STARTING NODE IN AN OD PAIR

INTEGER*2  NUMODPAIR
  THE NUMBER OF OD PAIRS

INTEGER*2  DEST(NNUMOD)
  THE DESTINATION NODE OF TRAFFIC IN AN OD PAIR

REAL  INPUTFLOW(NNUMOD)
  THE INPUT TRAFFIC TO THE NODE IN BITS/SECOND
'INCLUDE' FILE CONVRG.PRM

'CONVRG.PRM' CONTAINS THE CONVERGENCE PARAMETERS FOR THE
NETWORK FLOW PROBLEM

COMMON /CONVRG/
    &
    MAXITER,TOL,MAXUTI,OUTPFL

INTEGER MAXITER
    MAXIMUM NUMBER OF ITERATIONS IN THE SOLUTION
REAL TOL
    TOLERANCE ON SOLUTION ACCURACY
REAL MAXUTI
    MAXIMUM UTILIZATION FOR M/M/1 QUEUE DELAY
LOGICAL OUTPFL
    OUTPUT CONTROL VARIABLE
'INCLUDE' FILE PATHS.BLK

'PATHS.BLK' DEFINES THE ARRAYS NEEDED TO MAINTAIN PATH FLOWS AND DESCRIPTION.

COMMON /PATHS/
  & PA, FA, PATHID, NEXTPATH, FP, DIST, DTOT, CURERROR,
  & NUMPATH, NUMITER

INTEGER*2 PA(NNN)
THE LAST ARC ON A SHORTEST PATH TO A NODE

REAL FA(NNA)
THE FLOW IN ANY GIVEN ARC

INTEGER PATHID(NNUMPATH)
THE PATH IDENTIFIER FOR ANY GIVEN PATH

INTEGER NEXTPATH(NNUMPATH)
THE NEXT PATH FOR THE SAME OD PAIR

REAL FP(NNUMPATH)
THE FLOW OF A PATH

REAL DIST(NNN)
SHORTEST DISTANCE TO A NODE FROM THE ORIGIN

REAL DTOT(NMAXITER)
THE TOTAL DELAY BY ITERATION

INTEGER NUMITER
CURRENT ITERATION NUMBER

REAL CURERROR
CONVERGENCE ERROR (NORMALISED % OF FLOW NOT ON A SHORTEST PATH)

INTEGER NUMPATH
NUMBER OF GENERATED PATHS
'SETUP' ACCEPTS INPUTS FROM THE TERMINAL AND CREATES DATA SETS
THAT REPRESENTS NETWORKS AND LOADS IN A FORM SUITABLE FOR
PROGRAM 'MULTIFLO'

PROGRAM SETUP
IMPLICIT NONE

INCLUDE COMMON BLOCKS

INCLUDE 'PARAM.DIM'
INCLUDE 'NETWRK.PRM'

LOCAL VARIABLE DEFINITIONS

INTEGER TERMINAL_NODE
THE END NODE OF A LINK
INTEGER DESTOD
THE DESTINATION NODE OF AN OD PAIR
REAL BPS
MAXIMUM LINK CAPACITY
INTEGER NUMARC
NUMBER OF OUTGOING ARCS FOR A NODE IN THE NETWORK
REAL TRAFFIC
SPECIFIED INPUT TO AN OD PAIR
INTEGER I
DO LOOP INDEX
INTEGER J
DO LOOP INDEX
INTEGER NOD
NUMBER OF OD PAIRS ASSOCIATED WITH A COMMODITY

GET THE NODE SPECIFICATIONS

NA=0
WRITE(6,*)'INPUT THE # OF NODES'
READ(5,*)NN
DO I=1,NN
  WRITE(6,*)FOR NODE',I,' ENTER # OF ARCS EXITING THE NODE'
  READ(5,*,ERR=200)NUMARC
  IF(NUMARC.GE.0) THEN
    DO J=1,NUMARC
      WRITE(6,*)'FOR ARC',J,' AT NODE',I,' ENTER TERMINAL NODE',
      &
      ' ANDMAXIMUM BITS/S'
      READ(5,*,ERR=100)TERMINAL_NODE,BPS
      IF(TERMINAL_NODE.GT.NN) THEN
        WRITE(6,*)'TERMINAL NODE OUT OF BOUNDS'
        GO TO 100
      ELSE
        END
      END IF
    END DO
  END IF
END DO
ENTER LINK BEGIN AND END NODES

NA=NA+1
ENDDO(NA)=TERMINAL_NODE
BITRATE(NA)=BPS
END IF
STARTNODE(NA)=I
END DO
FRSTOU(I)=NA-NUMARC+1
LASTOU(I)=NA
ELSE
WRITE(6,*)'NEGATIVE ARCS ILLEGAL'
GO TO 200
END IF
END DO

OD PAIRS SETUP

WRITE(6,*)'ENTER THE NUMBER OF COMMODITIES IN THE NETWORK'
READ(5,*,ERR=1000)NUMCOMMOD
NUMODPAIR=0
DO I=1,NUMCOMMOD
WRITE(6,*)'ENTER THE ORIGIN ID AND NUMBER OF DESTINATIONS FOR ',
'COMMODITY',I
READ(5,*,ERR=300)ORCID(I),NOD
IF(ORCID(I).GE.NN) THEN
DO J=1,NOD
WRITE(6,*)'ENTER THE DESTINATION',J,' AND TRAFFIC FOR ',
'COMMODITY'
END IF
END DO
ELSE
WRITE(6,*)'ORIGIN IS OUT OF BOUNDS, MAX ORIGIN=',NN
GO TO 300
END IF
STARTOD(I)=NUMODPAIR-NOD+1
END DO

OUTPUT OF CONNECTIVITY DATA FOR DIRECT INPUT INTO 'MULTIFLO'
COMMON BLOCKS

WRITE(1,*)NN
DO I=1,NN
WRITE(1,*)FRSTOU(I),LASTOU(I)
END DO
WRITE(1,*)NA
DO I=1,NA
WRITE(1,*)STARTNODE(I),ENDDO(I),BITRATE(I)
END DO
OUTPUT OF OD TRAFFIC DATA FOR DIRECT INPUT INTO 'MULTIFLO'
COMMON BLOCKS

WRITE(2,*),NUMCOMMOD
DO I=1,NUMCOMMOD
   WRITE(2,*),ORGID(I),STARTOD(I)
END DO
WRITE(2,*),NUMODPAIR
DO I=1,NUMODPAIR
   WRITE(2,*),DEST(I),INPUT_FLOW(I)
END DO
STOP
END
APPENDIX II: MULTIFLO1 Code

The only differences between MULTIFLO and MULTIFLO1 are in the DRIVER program and in the main algorithm subroutine MULTIFLO. These two routines called DRIVER1 and MULTIFLO1, are listed below.
DRIVER1

'DRIVER1' IS A SIMPLE EXECUTIVE TO INVOKE THE 'MULTIFLO1' COMMODITY ROUTING PROGRAM. 'DRIVER1' INVOKES SUBPROGRAM 'LOAD' TO READ DATA INTO 'MULTIFLO1' INPUT COMMON BLOCKS. FILES READ BY 'LOAD' ARE CREATED BY A TERMINAL SESSION WITH THE USER FOR NETWORK DEFINITION THROUGH THE USE OF PROGRAM 'SETUP'.

EXECUTION STEPS FOR PROGRAM 'DRIVER1'

1) ASSIGN FORTRAN UNIT 01 AS CREATED BY PROGRAM 'LOAD'
2) ASSIGN FORTRAN UNIT 02 AS CREATED BY PROGRAM 'LOAD'
3) ASSIGN FORTRAN UNIT 06 AS A DESIGNATED OUTPUT FILE

E.G.:
$ ASSIGN NETWORK.DAT FOR001
$ ASSIGN TRAFFIC.DAT FOR002
$ ASSIGN OUTPUT.DAT FOR006

PROGRAM DRIVER1

LOAD FORTRAN UNIT 01 AND FORTRAN UNIT 02 FROM DISK AS CREATED FROM PROGRAM 'SETUP'

INCLUDE 'PARAM.DIM'
INCLUDE 'PATHS.BLK'
INCLUDE 'NETWRK.PRM'
INCLUDE 'CONVRG.PRM'
INTEGER COMMODITY,ORIGIN,DESTOD,OD,PATH
CALL LOAD

EXECUTE THE 'MULTIFLO1' NETWORK ALGORITHM. 'MULTIFLO1' SCHEDULES ITS OWN OUTPUTS TO FORTRAN UNIT 06 ON EACH ITERATION

INITIALIZE THE TIMER
CALL LIB$INIT_TIMER
CALL MULTIFLO1

RECORD THE COMPUTATION TIME
CALL LIB$SHOW_TIMER

PRINT MAX LINK UTILIZATION (RELEVANT FOR M/M/1 QUEUEING DELAY OPTIMIZATION)
UMAX=0.0
DO 100 I=1,NA
   UMAX=MAX(UMAX,FA(I)/BITRATE(I))
CONTINUE
WRITE(6,'(*)') 'MAXIMUM LINK UTILIZATION'
WRITE(6,'(*)') UMAX

PRINT FINAL PATH FLOW INFO
WRITE(6,'(*)') 'ORIGIN / DESTINATION / PATH # / PATH FLOW'
DO 1000 COMMODITY=1,NUMCOMMOD
   ORIGIN=ORGID(COMMODITY)
   DO 500 OD=STARTOD(COMMODITY),STARTOD(COMMODITY+1)-1
DESTOD = DEST(OD)
PATH = OD
DO WHILE (PATH.GT.0)
   WRITE (6, *) ORIGIN, DESTOD, PATH, FP (PATH)
   PATH = NEXTPATH (PATH)
END DO
500    CONTINUE
1000   CONTINUE
STOP
END
MULTICOMMODITY FLOW ALGORITHM BASED ON A PATH FLOW FORMULATION
 UPDATES THE PATH FLOWS OF OD PAIRS ONE AT A TIME ACCORDING TO AN ITERATION OF THE PROJECTION TYPE.

DEVELOPED BY DIMITRI BERTSEKAS, BOB GENDRON, AND WEI K TSAI

BASED ON THE PAPERS:

1) BERTSEKAS, D.P., "A CLASS OF OPTIMAL ROUTING ALGORITHMS FOR COMMUNICATION NETWORKS", PROC. OF 5TH INTERNATIONAL CONFERENCE ON COMPUTER COMMUNICATION (ICCC-80), ATLANTA, GA., OCT. 1980, PP. 71-76.


SUBROUTINE MULTIFLO1

IMPLICIT NONE

**************** INCLUDE COMMON BLOCKS ****************

INCLUDE 'PARAM.DIM'
INCLUDE 'NETWRK.PRM'
INCLUDE 'CONVRC.PRM'
INCLUDE 'PATHS.BLK'

NODE ARRAYS (LENGTH NN):
FRSTO(NODE) - FIRST ARC OUT OF NODE
LASTO(NODE) - LAST ARC OUT OF NODE
NOTE: THE ARC LIST MUST BE ORDERED IN SEQUENCE SO THAT ALL ARCS OUT OF ANY NODE ARE GROUPED TOGETHER

ARC ARRAYS (LENGTH NA):
FA(ARC) - THE TOTAL FLOW OF ARC
STARTO(ARC) - THE HEAD NODE OF ARC
ENDO(ARC) - THE TAIL NODE OF ARC
COMMODITY LENGTH ARRAYS (LENGTH NUMCOMMOD):

ORGID(COMMODITY) - THE NODE ID OF THE ORIGIN OF COMMODITY
STARTOD(COMMODITY) - THE STARTING OD PAIR IN THE ODPAIR LIST
CORRESPONDING TO THE ORIGIN IN POSITION RANK

NOTE: THIS SCHEME ASSUMES THAT OD PAIRS ARE LISTED IN SEQUENCE
I.E. THE OD PAIRS CORRESPONDING TO THE COMMODITY ONE
ARE LISTED FIRST. THEY ARE
FOLLOWED BY THE OD PAIRS OF THE COMMODITY TWO
AND SO ON.

ODPAIR ARRAYS (LENGTH NUMOD):
DEST(OD) - GIVES THE DESTINATION OF ODPAIR OD
INPUTFLOW(OD) - GIVES THE INPUT TRAFFIC OF ODPAIR OD

PATH ARRAYS (LENGTH DYNAMICALLY UPDATED):
PATHID(PATH) - POINTER TO THE BLOCK DESCRIBING PATH
IN THE PATH DESCRIPTION ARRAY
NEXTPATH(PATH) - THE NEXT PATH FOR THE SAME OD PAIR FOLLOWING
PATH. IT EQUALS 0 IF PATH IS THE LAST FOR THAT OD PAIR
FP(PATH) - THE FLOW CARRIED BY PATH

PATH DESCRIPTION LIST ARRAY (LENGTH DYNAMICALLY UPDATED)
PDESCR(INDEX) - THIS LONG ARRAY EXPLICITLY DESCRIBES ALL
ACTIVE PATHS. FOR ANY PATH, PATHID(PATH) IS A POINTER
TO PDESCR. IT GIVES THE ELEMENT
OF THE PDESCR ARRAY CONTAINING THE # OF ARCS IN THE PATH
(CALL IT NUMARC). THE ELEMENTS PATHID(PATH)-NUMARC TO
PATHID(PATH)-1 OF THE ARRAY PDESCR CONTAIN THE ARCS THAT
MAKE UP PATH STARTING FROM THE DESTINATION AND GOING TOWARDS
THE ORIGIN OF PATH.

********************** LOCAL VARIABLE DEFINITIONS **********************
INTEGER*2 PDESCR (NINDEX)
PATH DESCRIPTION ARRAY - CONTAINS EXPLICIT
DESCRIPTION OF ALL ACTIVE PATHS.
LOGICAL SPNEW
LOGICAL INDICATING A NEW PATH FOUND
LOGICAL SAME
LOGICAL INDICATING A NEW SHORTEST PATH ALREADY EXISTING
INTEGER NODE
NODE IDENTIFIER
INTEGER DESTOD
THE DESTINATION NODE OF AN OD PAIR
INTEGER ARC
DO LOOP INDEX FOR ARCS
INTEGER PATH
A PATH INDEX
INTEGER NUMLIST
TOTAL NUMBER OF ACTIVE PATHS FOR OD PAIR UNDER CONSIDERATION
INTEGER ITER
SPECIFIC ITERATION
INTEGER N1,N2
TEMPORARY VARIABLES
REAL MINFDER
THE LENGTH FOR A SHORTEST PATH
REAL MINSDER
THE SECOND DERIVATIVE LENGTH FOR THE SHORTEST PATH
REAL TMINSDER
TEMPORARY VALUE FOR SECOND DERIVATIVE LENGTH OF SHORTEST PATH
**EXECUTABLE CODE**

**INITIALIZATION**

```fortran
DO 5 ARC=1,NA
    FA(ARC)=0.0
5 CONTINUE
```

```fortran
DO 1=1(NUMODPAIR)
    FP(I)=INPUT_FLOW(I)
1 ENDDO
```

```fortran
STARTOD (NUMCOMMOD+1)=NUMODPAIR+1
NUMPATH=0
```
INDEX=0
NUMITER=1
DO 100 COMMODITY=1,NUMCOMMOD
   ORIGIN=ORGID(COMMODITY)
   CALL SP(ORIGIN,COMMODITY)
   INDEX=INDEX+1
   LOOP OVER OD PAIRS OF COMMODITY
   N1=STARTOD(COMMODITY)
   N2=STARTOD(COMMODITY+1)-1
   DO 50 OD=N1,N2
      NUMPATH=NUMPATH+1
      NEXTPATH(NUMPATH)=0
      FLOW=FP(NUMPATH)
      INDEX=INDEX+1
      NUMARC=0
      NODE=DEST(OD)
      DO WHILE (NODE.NE.ORIGIN)
         ARC=PA(NODE)
         FA(ARC)=FA(ARC)+FLOW
         PDESCR(INDEX)=ARC
         NUMARC=NUMARC+1
         INDEX=INDEX+1
         NODE=STARTNODE(ARC)
      END DO
      PATHID(NUMPATH)=INDEX
      PDESCR(INDEX)=NUMARC
   CONTINUE
50 CONTINUE
100 CONTINUE
C
C INITIALIZE MEMBER ARRAY
C
DO 70 ARC=1,NA
   MEMBER(ARC)=.FALSE.
CONTINUE
C
C INITIALIZE THE TOTAL DELAY
C
CALL DELAY(DTOT(NUMITER))
C
C OUTPUT THE CURRENT INFORMATION TO DISK
C
CALL PREAD
C
******************************************************************************
* END OF INITIALIZATION
******************************************************************************
C
***** START NEW ITERATION *****
C
110 NUMITER=NUMITER+1
CURERROR=0
C
***** LOOP OVER ALL COMMODITIES ****
C
DO 1000 COMMODITY=1,NUMCOMMOD
   ORIGIN=ORGID(COMMODITY)
   CALL SP(ORIGIN,COMMODITY)
   LOOP OVER OD PAIRS OF COMMODITY
N1 = STARTOD(COMMODITY)
N2 = STARTOD(COMMODITY + 1) - 1
DO 500 OD = N1, N2

CHECK IF THERE IS ONLY ONE ACTIVE PATH AND IF SO SKIP THE ITERATION

IF (NEXTPATH(OD).EQ.0) THEN
  NODE = DEST(OD)
  POINT = PATHID(OD)
  NUMARC = PDESCR(POINT)
  DO 150 I = POINT - NUMARC, POINT - 1
    ARC = PDESCR(I)
    IF (ARC.NE.PA(NODE)) GO TO 180
    NODE = STARTNODE(ARC)
  150 CONTINUE
  GO TO 500
END IF

180 CONTINUE

MARK THE ARCS OF THE SHORTEST PATH

DESTOD = DEST(OD)
NODE = DESTOD
DO WHILE (NODE.NE.ORIGIN)
  ARC = PA(NODE)
  MEMBER(ARC) = .TRUE.
  NODE = STARTNODE(ARC)
END DO

GENERATE LIST OF ACTIVE PATHS FOR OD PAIR

NUMLIST = 1
PATHLIST(1) = OD
PATH = NEXTPATH(OD)
DO WHILE (PATH.GT.0)
  NUMLIST = NUMLIST + 1
  PATHLIST(NUMLIST) = PATH
  PATH = NEXTPATH(PATH)
END DO

DETERMINE 1ST & 2ND DERIVATIVE LENGTH OF ACTIVE PATHS
ALSO DETERMINE WHETHER THE CALCULATED SHORTEST PATH IS ALREADY IN THE LIST

SPNEW = .TRUE.
DO 200 K = 1, NUMLIST
  SAME = .TRUE.
  FDER = 0
  SDER = 0
  TMINSDER = 0
  PATH = PATHLIST(K)
  POINT = PATHID(PATH)
  NUMARC = PDESCR(POINT)
  DO 210 I = POINT - NUMARC, POINT - 1
    ARC = PDESCR(I)
    CALL DERIVS(COMMODITY, FA(ARC), ARC, D1CAL, D2CAL)
  210 CONTINUE
FER=FER+D1CAL
IF (.NOT.MEMBER (ARC)) THEN
   SDER=SDER+D2CAL
   SAME=.FALSE.
ELSE
   SDER=SDER-D2CAL
   TMINSDER=TMINSDER+D2CAL
END IF
CONTINUE
IF (SAME) THEN
   SPNEW=.FALSE.
   SHORTEST=PATH
   FDLENGTH(K)=FER
   MINFER=FER
   MINS=MINSDER
ELSE
   FDLENGTH(K)=FER
   SDLKETH(K)=SDER
END IF
CONTINUE

*** INSERT SHORTEST PATH IN PATH LIST IF IT IS NEW ***

IF (SPNEW) THEN
   NUMPATH=NUMPATH+1
   SHORTEST=NUMPATH
   NEXTPATH(PATHLIST(NUMPATH))=NUMPATH
   NEXTPATH(NUMPATH)=0
   MINS=0
   MINS=0
   INDEX=INDEX+1
   NUMARC=0
   NODE=DESTOD
   DO WHILE (NODE.NE.ORIGIN)
      ARC=PA(NODE)
      PDESCR(INDEX)=ARC
      NUMARC=NUMARC+1
      INDEX=INDEX+1
      CALL DERIVS(COMMODITY,FA(ARC),ARC,D1CAL,D2CAL)
      MINS=MINFER+D1CAL
      MINS=MINSDER+D2CAL
      NODE=STARTNODE(ARC)
   END DO
   PATHID(NUMPATH)=INDEX
   PDESCR(INDEX)=NUMARC
END IF

*** UPDATE PATH & LINK FLOWS ***

INCR=0
TEMPERROR=0
DO 250 K=1,NUMLIST
   DLENGTH=FDLENGTH(K)-MINFER
   IF (DLENGTH.GT.0) THEN
      PATH=PATHLIST(K)
      FLOW=FP(PATH)
      IF ((FLOW.EQ.0.0).AND.(K.GT.1)) THEN
         NEXTPATH(PATHLIST(K-1))=NEXTPATH(PATH)
      ELSE
         NEXTPATH(PATHLIST(K))=NEXTPATH(PATH)
      END IF
   END IF
END DO 250
PATHINCR = DLENGTH / (SDLENGTH(K) * MINSDER)
IF (FLOW .LE. PATHINCR) THEN
  FP(PATH) = 0.0
  PATHINCR = FLOW
ELSE
  FP(PATH) = FLOW - PATHINCR
END IF
INCR = INCR + PATHINCR
TEMPERROR = TEMPERROR + FLOW * DLENGTH / EDLENGTH(K)
POINT = PATHID(PATH)
NUMARC = PDESCR(POINT)
DO 220 I = POINT - NUMARC, POINT - 1
  ARC = PDESCR(I)
  FA(ARC) = FA(ARC) - PATHINCR
220 CONTINUE
END IF
CONTINUE

*** UPDATE THE ERROR CRITERION ***
CURERROR = AMAX1(CURERROR, TEMPERROR / INPUT_FLOW(OD))

**** UPDATE FLOWS FOR SHORTEST PATH ****
FP(SHORTEST) = FP(SHORTEST) + INCR
POINT = PATHID(SHORTEST)
NUMARC = PDESCR(POINT)
DO 300 I = POINT - NUMARC, POINT - 1
  ARC = PDESCR(I)
  FA(ARC) = FA(ARC) + INCR
  MEMBER(ARC) = .FALSE.
300 CONTINUE
CONTINUE

***** END OF LOOP FOR OD PAIRS CORRESPONDING TO COMMODITY *****
***** UPDATE TOTAL DELAY *****
CALL DELAY(DTOT(NUMITER))

1000 CONTINUE

CHECK IF THE # OF ACTIVE PATHS EXCEED THE ALLOCATED NUMBER
IF (NUMPATH.GT.NNUMPATH) THEN
  WRITE(6,*) 'MAX # OF ALLOCATED PATHS EXCEEDED'
  STOP
END IF
IF (INDEX.GT.NINDEX) THEN
  WRITE(6,*) 'DIMENSION OF PDESCR ARRAY EXCEEDED'
  STOP
END IF

OUTPUT THE CURRENT SOLUTION TO DISK
CALL PRFLOW

***** END OF ITERATION *****
*** IF THE ERROR IS SMALLER THAN TOL, OR THE LIMIT ON
THE NUMBER OF ITERATIONS IS REACHED RETURN
ELSE GO FOR ANOTHER ITERATION

IF ((CURERROR.LT.TOL) .OR. (NUMITER.EQ.MAXITER)) THEN
  WRITE(6,'*') 'FINAL STORAGE OF PATH DESCRIPTION LIST'
  WRITE(6,'*') INDEX
  RETURN
ELSE
  GO TO 110
END IF

END

********** END OF MULTIFLO1 **********