FORTRAN CODES FOR NETWORK OPTIMIZATION
by Dimitri P. Bertsekas

Most of these codes are essentially those listed in the appendix of the textbook
"LINEAR NETWORK OPTIMIZATION: ALGORITHMS AND CODES",
MIT PRESS,1991 by Dimitri P. Bertsekas

The theory of many of the codes is described in the survey article "Auction
Algorithms for Network Flow Problems: A Tutorial Introduction", Computational
Optimization and Applications, 1, 1992, 7-26, and in the more recent textbook

"NETWORK OPTIMIZATION: CONTINUOUS AND DISCRETE MODELS",
ATHENA SCIENTIFIC,1998 by Dimitri P. Bertsekas

The codes will be periodically updated, so your feedback will be very valuable.
Send email to

dimitrib@mit.edu

or U.S.Mail to

Prof. Dimitri P. Bertsekas
M.I.T., Rm 35-210
Cambridge, Mass. 02139
U.S.A.

****
*

USAGE:

All codes use integer arithmetic and can be
compiled with the Absoft compiler on all Macintosh computers (subject to memory limitations).The cost range of the problem being solved must be sufficiently limited to avoid integer overflow. By changing a few input and
output statements, the codes can be easily adapted for other computers and
FORTRAN compilers. The codes should not be modified except for the minimum modifications necessary to make them run on other compilers. Please state clearly your modifications when reporting research.

All codes can be used for any noncommercial purpose. In particular the codes
should not be integrated into any commercial product or be used to satisfy any
govermental or industrial deriverables

****
*

CONTENTS:

The following is a summary of the codes:

1) GRIDGEN: Generates min cost flow problems with a grid structure.

2) NGCONVERT: Converts problems from the NETGEN format to a standard format.

3) PAPE_ALL_DEST: Pape's method for shortest paths from one origin to all destinations.

4) HEAP_ALL_DEST: Binary heap method for shortest paths from one origin to all destinations.

5) HEAP_SELECT_DEST: Binary heap method for shortest paths from one origin to selected destinations.

6) AUCTION_SELECT_DEST: Auction algorithm for shortest paths from one origin to selected destinations, as per the paper "An Auction Algorithm for Shortest Paths," SIAM J. on Optimization, Vol. 1, 1991, pp. 425-447, by D. P. Bertsekas.

7) RELAX_QC: Relaxation method for min cost flow, as per the paper "A Unified Framework for Primal-Dual Methods in Minimum Cost Network Flow Problems," Math. Programming, Vol. 32, pp. 125-145, 1985, by D. P. Bertsekas.

8) AUCTION: Forward auction algorithm for symmetric assignment problems, as per the original 1979 MIT report "A Distributed Algorithm for the Assignment Problem"; see also the paper "The Auction Algorithm: A Distributed Relaxation Method for the Assignment Problem," Annals of Operations Research, Vol. 14, 1988, pp. 105-123, by D. P. Bertsekas.

9) AUCTION_FLP: Same as AUCTION but uses floating point arithmetic to deal with problems with large cost range and/or dimension. For such problems the prices may overflow the integer range in the course of the algorithm.

10) AUCTION_AS: Auction algorithm for asymmetric assignment problems.

11) AUCTION_FR: Forward/reverse auction algorithm for symmetric assignment problems.

12) NAUCTION_SP: Combined naive auction and sequential shortest path method for assignment.

13) FORD_FULKERSON: Ford-Fulkerson algorithm for max-flow.

14) E_RELAX_MF: E-relaxation algorithm for max-flow.

15) E_RELAX: E-relaxation algorithm for min cost flow as per the paper "Distributed Relaxation Methods for Linear Network Flow Problems," Proc. 25th IEEE Conference on Decision and Control, Athens, Greece, 1986, by D. P. Bertsekas.

16) E_RELAX_N: E-relaxation algorithm for min cost flow; iterates from both positive and negative surplus nodes.

17) SLF: Small Label First (SLF) algorithm for finding shortest paths from one origin to all destinations, as per the paper "A Simple and Fast Label Correcting Method for Shortest Paths," Networks, Vol. 23, 1993, pp. 703-709, by D. P. Bertsekas.

18) SLFT: Combination of Small Label First (SLF) and threshold algorithms for finding shortest paths from one origin to all destinations, as per the paper "A Simple and Fast Label Correcting Method for Shortest Paths," Networks, Vol. 23, 1993, pp. 703-709, by D. P. Bertsekas.

19) AUCTION_MF: Auction algorithm for the max-flow problem, as per the paper "An Auction Algorithm for the Max-Flow Problem", JOTA , Vol. 87, 1995, pp. 69-101, by D. P. Bertsekas.

A common format has been adopted whereby each of the codes (except for the assignment codes) does the following:
(a)
Reads the problem in a common standard form.
(b)
Converts the problem to a form suitable for the algorithm used.
(c)
Calls the algorithm to solve the problem.
(d)
Outputs the results.

[The assignment codes include a built-in random problem generator.]

The standard form adopted is the one used for the simplex method (cf.\ Section
2.4). In particular, the problem is specified by:

The number of nodes N.

The number of arcs A.

The four A-length arrays $\it START$, $\it END$, $\it COST$, $\it CAPACITY$.

The N-length array $\it SUPPLY$.\endlist

As in Section 2.4, the arrays represent the following:

START(a): The start node of arc a.

END(a): The end node of arc a.

COST(a): The cost coefficient of arc a.

CAPACITY(a): The upper flow bound of arc a.

SUPPLY(i): The supply of node i.

The standard form representation of the problem is the simplest. However, to avoid unnecessary conversions, it is
more efficient to represent the problem in a format that is tailored to the method at hand. Thus, for long term use, the reader may wish to modify
the input portion of the given codes.

****
*

LISTINGS:

****
*

Problem Generator and Conversion Codes

****
*

GRIDGEN

To provide the reader with the means to create nonbipartite test problems, we give a simple generator, called GRIDGEN, which constructs random problems with an underlying two-dimensional grid with wraparound structure. The grid arcs form a ``skeleton,'' guaranteeing problem feasibility. Additional arcs are added between randomly selected nodes, as specified by the user. The following problem parameters are required as input:

{\bf $\it DIM1$, $\it DIM2$:} These are the two dimensions of the grid; the number of nodes is $\it N=DIM1\times DIM2$.

{\bf $\it ADDARCS$:} This is the number of arcs in addition to $6N$ arcs that are automatically generated. In particular, there are four arcs per node that form the grid, two arcs per node that start at the node and end at a randomly selected node, and $\it ADDARCS$ additional arcs that connect randomly selected pairs of nodes.

{\bf $\it TOTSUPPLY$:} The total supply, that is, the sum of the supplies of the sources (nodes that have positive supply).

{\bf $\it SOURCES$, $\it SINKS$:} The number of nodes that have positive supply, and the number of nodes that have negative supply, respectively. If both $\it SOURCES$ and $\it SINKS$ are less than $N/4$, the source nodes and sink nodes are selected randomly according to a uniform distribution; otherwise nodes 1 to $\it SOURCES$ are the source nodes, and nodes are $\it (N+1-SINKS)$ to N are the sink nodes. In both cases the supply of a source is $$\it TOTSUPPLY\over SOURCES$$ and the supply of a sink is $$\it -{TOTSUPPLY\over SINKS}$$ (to the greatest possible approximation).

{\bf $\it MINCOST$, $\it MAXCOST$:} The cost coefficient of the $4N$ grid arcs is $\it MAXCOST$. All other arcs have cost coefficient selected according to a uniform distribution between $\it MINCOST$ and $\it MAXCOST$. Thus, there is an incentive to avoid the grid arcs as much as possible in an optimal solution.

{\bf $\it MINCAP$, $\it MAXCAP$:} The capacity of the grid arcs is $\it

TOTSUPPLY$. The capacity of the other arcs is selected according to a uniform
distribution
between $\it MINCAP$ and $\it MAXCAP$.
The ``skeleton'' arcs of the grid guarantee that the
problem is feasible (any source can be connected to any sink with a
path of ``capacity'' $\it TOTSUPPLY$). However, these arcs have high cost, so at the optimum, they
tend to carry little flow (if any). Thus, to generate a meaningful problem,
the reader must specify a fairly large value for $\it ADDARCS$. Note that the
generated problem cannot be unbounded since every arc has a real upper flow
bound.

```
C ********************************************************************
C
C                 MINCOST GRID PROBLEM GENERATOR
C                     BY DIMITRI P. BERTSEKAS
C                   DECEMBER 1990
C
C ********************************************************************
C
C     THIS CODE GENERATES A GRID MINCOST PROBLEM
C     AND OUTPUTS THE PROBLEM IN STANDARD FORMAT IN THE FILE "FOR013.DAT".
C
C
      IMPLICIT INTEGER (A-Z)
      LOGICAL MARK
      DIMENSION STARTN(50000),ENDN(50000),C(50000),U(50000)
      DIMENSION B(10000)
      DIMENSION MARK(10000)


C     THIS CODE INCLUDES A UNIFORM RANDOM NUMBER GENERATOR
C     WHICH RETURNS A VALUE IN (0,1)

C   RANDOM GENERATOR STUFF

      COMMON /RANDM/ MULT,MODUL,I15,I16,JRAN
      REAL RAN

C     INITIALIZE RANDOM GENERATOR

      ISEED=13502460
      CALL SETRAN(ISEED)

      PRINT*,'CREATING A 2-D GRID MINCOST FLOW PROBLEM GRAPH'
      PRINT*,'******************************************'
5     PRINT*,'ENTER # OF NODES IN EACH OF THE TWO DIMENSIONS'
      READ*,DIM1,DIM2

      IF (DIM1.LE.1) THEN
```

```
        PRINT*,'DIMENSION 1 MUST BE GREATER THAN 1'
               GO TO 5
      END IF
      IF (DIM2.LE.1) THEN
        PRINT*,'DIMENSION 2 MUST BE GREATER THAN 1'
               GO TO 5
      END IF


      N=DIM1*DIM2

      PRINT*,'NUMBER OF ARCS SO FAR ',6*N
      PRINT*,'ENTER # OF ADDITIONAL ARCS'
      READ*,ADDARCS

      PRINT*,'TOTAL SUPPLY,# OF SOURCES,# OF SINKS'
      READ*,TOTSUPPLY,SOURCES,SINKS

      PRINT*,'ENTER MINIMUM AND MAXIMUM COST COEFFICIENT'
      READ*,MINCOST,MAXCOST

      PRINT*,'ENTER MIN AND MAX ARC CAPACITY'
      READ*,MINCAP,MAXCAP

      ARC=0

      DO 10 J=1,DIM2
      DO 15 I=1,DIM1
        NODE=I+(J-1)*DIM1

C       CREATE HORIZONTAL ARCS

        IF ((I.GT.1).AND.(I.LT.DIM1)) THEN
          ARC=ARC+1
          STARTN(ARC)=NODE
          ENDN(ARC)=NODE+1
          C(ARC)=MAXCOST
          U(ARC)=TOTSUPPLY

          ARC=ARC+1
          STARTN(ARC)=NODE
          ENDN(ARC)=NODE-1
          C(ARC)=MAXCOST
          U(ARC)=TOTSUPPLY
        ELSE
          IF (I.EQ.1) THEN
            ARC=ARC+1
            STARTN(ARC)=NODE
            ENDN(ARC)=NODE+1
            C(ARC)=MAXCOST
            U(ARC)=TOTSUPPLY

            ARC=ARC+1
            STARTN(ARC)=NODE
```

```
            ENDN(ARC)=NODE+(DIM1-1)
            C(ARC)=MAXCOST
            U(ARC)=TOTSUPPLY
          ELSE
            ARC=ARC+1
            STARTN(ARC)=NODE
            ENDN(ARC)=NODE-(DIM1-1)
            C(ARC)=MAXCOST
            U(ARC)=TOTSUPPLY

            ARC=ARC+1
            STARTN(ARC)=NODE
            ENDN(ARC)=NODE-1
            C(ARC)=MAXCOST
            U(ARC)=TOTSUPPLY
          END IF
        END IF



C       CREATE VERTICAL ARCS


        IF ((J.GT.1).AND.(J.LT.DIM2)) THEN
          ARC=ARC+1
          STARTN(ARC)=NODE
          ENDN(ARC)=NODE+DIM1
          C(ARC)=MAXCOST
          U(ARC)=TOTSUPPLY

          ARC=ARC+1
          STARTN(ARC)=NODE
          ENDN(ARC)=NODE-DIM1
          C(ARC)=MAXCOST
          U(ARC)=TOTSUPPLY
        ELSE
          IF (J.EQ.1) THEN
            ARC=ARC+1
            STARTN(ARC)=NODE
            ENDN(ARC)=NODE+DIM1
            C(ARC)=MAXCOST
            U(ARC)=TOTSUPPLY

            ARC=ARC+1
            STARTN(ARC)=NODE
            ENDN(ARC)=NODE+DIM1*(DIM2-1)
            C(ARC)=MAXCOST
            U(ARC)=TOTSUPPLY
          ELSE
            ARC=ARC+1
            STARTN(ARC)=NODE
            ENDN(ARC)=I
            C(ARC)=MAXCOST
            U(ARC)=TOTSUPPLY
```

```
            ARC=ARC+1
            STARTN(ARC)=NODE
            ENDN(ARC)=NODE-DIM1
            C(ARC)=MAXCOST
            U(ARC)=TOTSUPPLY
          END IF
        END IF


C       CREATE 2 ARCS WITH RANDOM END NODE

        DO 8 K=1,2

          ARC=ARC+1
          STARTN(ARC)=NODE
7         ENDN(ARC)=1+INT(RAN()*N)
          C(ARC)=MINCOST+RAN()*(MAXCOST-MINCOST)
          U(ARC)=MINCAP+RAN()*(MAXCAP-MINCAP)
          IF (ENDN(ARC).EQ.NODE) GOTO 7
8       CONTINUE

15    CONTINUE
10    CONTINUE

C       CREATE ADDARCS NEW ARCS WITH RANDOM START AND END NODE

        DO 12 K=1,ADDARCS
          ARC=ARC+1
          NODE=1+INT(RAN()*N)
          STARTN(ARC)=NODE
13        ENDN(ARC)=1+INT(RAN()*N)
          C(ARC)=MINCOST+RAN()*(MAXCOST-MINCOST)
          U(ARC)=MINCAP+RAN()*(MAXCAP-MINCAP)
          IF (ENDN(ARC).EQ.NODE) GOTO 13
12      CONTINUE

      NA=ARC


C    GENERATE THE SINKS AND SOURCES

      DO 17 I=1,N
        B(I)=0
        MARK(I)=.FALSE.
17    CONTINUE

      SUPL=0
      DO 20 I=1,SOURCES
18        NODE=1+INT(RAN()*N)
          IF (MARK(NODE)) THEN
            GOTO 18
          ELSE
            MARK(NODE)=.TRUE.
```

```
            B(NODE)=INT(TOTSUPPLY/SOURCES)
            SUPL=SUPL+B(NODE)
            LAST=NODE
          END IF
20    CONTINUE

      B(LAST)=B(LAST)+TOTSUPPLY-SUPL

      DEM=0
      DO 25 I=1,SINKS
22        NODE=1+INT(RAN()*N)
          IF (MARK(NODE)) THEN
            GOTO 22
          ELSE
            MARK(NODE)=.TRUE.
            B(NODE)=-INT(TOTSUPPLY/SINKS)
            DEM=DEM-B(NODE)
          END IF
25    CONTINUE


C     EQUALIZE SUPPLY AND DEMAND

      IF (SUPL.GT.DEM) THEN
        DO 30 NODE=1,N
          IF (B(NODE).LT.0) THEN
            B(NODE)=B(NODE)-(SUPL-DEM)
            GOTO 38
          END IF
30      CONTINUE
       ELSE
         IF (SUPL.LT.DEM) THEN
           DO 35 NODE=1,N
             IF (B(NODE).GT.0) THEN
               B(NODE)=B(NODE)+(DEM-SUPL)
               GOTO 38
             END IF
35         CONTINUE
         END IF
       END IF

38     CONTINUE


C
      PRINT*,'*******************************************'
      PRINT*,'WRITING THE PROBLEM ON FILE FOR013.DAT'
      OPEN(13,FILE='FOR013.DAT',STATUS='NEW')
      REWIND(13)

C     WRITE NUMBER OF NODES AND ARCS

      WRITE(13,1010) N,NA
```

```
C     WRITE START, END, COST, AND CAPACITY OF EACH ARC

      DO 40 I=1,NA
        WRITE(13,1020) STARTN(I),ENDN(I),C(I),U(I)
40    CONTINUE

C     WRITE SUPPLY OF EACH NODE

      DO 50 I=1,N
        WRITE(13,1000) B(I)
50    CONTINUE

      ENDFILE(13)
      REWIND(13)

      PRINT*,'END OF WRITING'

1000  FORMAT(1I8)
1010  FORMAT(2I8)
1020  FORMAT(4I8)

      STOP
      END

      SUBROUTINE SETRAN(ISEED)
      IMPLICIT REAL*8 (A-H,O-Z) , INTEGER*4 (I-N)
C**********************************************************************
C PORTABLE CONGRUENTIAL (UNIFORM) RANDOM NUMBER GENERATOR:
C     NEXT_VALUE = [(7**5) * PREVIOUS_VALUE] MODULO[(2**31)-1]
C
C THIS GENERATOR CONSISTS OF TWO ROUTINES:
C   (1) SETRAN - INITIALIZES CONSTANTS AND SEED
C   (2) RRAN   - GENERATES A REAL RANDOM NUMBER
C
C THE GENERATOR REQUIRES A MACHINE WITH AT LEAST 32 BITS OF PRECISION.
C THE SEED (ISEED) MUST BE IN THE RANGE (1,(2**31)-1).
C**********************************************************************
      COMMON /RANDM/ MULT,MODUL,I15,I16,JRAN
      IF(ISEED.LT.1) STOP 77
      MULT=16807
      MODUL=2147483647
      I15=2**15
      I16=2**16
      JRAN=ISEED
      RETURN
      END
C
      REAL FUNCTION RAN()
      IMPLICIT REAL*4 (A-H,O-Z) , INTEGER*4 (I-N)
C**********************************************************************
C RAN GENERATES A REAL RANDOM NUMBER BETWEEN 0 AND 1
C**********************************************************************
      COMMON /RANDM/ MULT,MODUL,I15,I16,JRAN
      IXHI=JRAN/I16
```

```
      IXLO=JRAN-IXHI*I16
      IXALO=IXLO*MULT
      LEFTLO=IXALO/I16
      IXAHI=IXHI*MULT
      IFULHI=IXAHI+LEFTLO
      IRTLO=IXALO-LEFTLO*I16
      IOVER=IFULHI/I15
      IRTHI=IFULHI-IOVER*I15
      JRAN=((IRTLO-MODUL)+IRTHI*I16)+IOVER
      IF(JRAN.LT.0) JRAN=JRAN+MODUL
      RAN = FLOAT(JRAN)/FLOAT(MODUL)
      RETURN
      END
```

NGCONVERT: NETGEN to Stardard Format Conversion

There is in the public domain a popular random network problem generator called NETGEN [KNS74]. This generator uses a problem representation format that is different from the standard form described earlier. For the benefit of a reader who has access to NETGEN, we provide the conversion code NGCONVERT, which reads a problem in the NETGEN format and writes it into the standard format.

```
C     ******************** CONVERSION PROGRAM *****************
C
C     THIS PROGRAM WILL READ A PROBLEM FILE CREATED VIA
C     THE RANDOM PROBLEM GENERATOR NETGEN OR ANY GENERATOR THAT
C     USES THE NETGEN FORMAT, AND CONVERT IT INTO THE STANDARD
C     FORMAT.
C
C     ************************************************
C
      PROGRAM NGCONVERT
      IMPLICIT NONE
      INTEGER START(45000),END(45000),COST(45000),CAP(45000)
      INTEGER SUPPLY(15000)
      INTEGER I,N,IA,NA,NSORC,NSINK
C

      PRINT *,'READING NETGEN PROBLEM DATA'
      OPEN(12,FILE='FOR012.DAT',STATUS='OLD')
      REWIND(12)
      READ(12,1020) N,IA,NSORC,NSINK

      DO 10 I=1,IA
        READ(12,1020) START(I),END(I),COST(I),CAP(I)
10    CONTINUE
```

```
      DO 20 I=1,N
        SUPPLY(I)=0
20    CONTINUE

      NA=0
      DO 25 I=1,IA
        IF (START(I).EQ.N+1) THEN
          SUPPLY(END(I))=CAP(I)
        ELSE
          IF (END(I).EQ.N+2) THEN
            SUPPLY(START(I))=-CAP(I)
          ELSE
            NA=NA+1
            COST(NA)=COST(I)
            CAP(NA)=CAP(I)
            START(NA)=START(I)
            END(NA)=END(I)
          END IF
        END IF
25    CONTINUE

      PRINT*,'WRITING THE PROBLEM ON DISK IN STANDARD FORM'
      OPEN(13,FILE='FOR013.DAT',STATUS='NEW')
      REWIND(13)

C     WRITE NUMBER OF NODES AND ARCS

      WRITE(13,1010) N,NA

C     WRITE START,  END, COST, AND CAPACITY OF EACH ARC

      DO 30 I=1,NA
        WRITE(13,1020) START(I),END(I),COST(I),CAP(I)
30    CONTINUE

C     WRITE SUPPLY OF EACH NODE

      DO 40 I=1,N
        WRITE(13,1000) SUPPLY(I)
40    CONTINUE

      ENDFILE(13)
      REWIND(13)

      PRINT*,'END OF WRITING'
      STOP

1000  FORMAT(1I8)
1010  FORMAT(2I8)
1020  FORMAT(4I8)

      END
```

```
************************************************************************
*

Shortest Path Codes

************************************************************************
*


PAPE-ALL-DEST

This code implements a variation of the D'Esopo-Pape
single origin/all destinations label correcting method
(cf.\ Section 1.3.3), given in [Pal84] and [GaP88]. The code is adapted
from
the L2QUEUE code of [GaP88].

C     ***** SAMPLE CALLING PROGRAM FOR SUBROUTINE L2QUEUE    *****
C     ***             (SHORTEST PATH PROBLEM)                  ***
C     ***                                                      ***
C     ***          THE PROGRAM IS BASED ON THE PAPER           ***
C     *** G. GALLO, S. PALLOTTINO "SHORTEST PATH ALGORITHMS",  ***
C     ***       ANNALS OF OPERATIONS RESEARCH 7, 1988.         ***
C     ***                                                      ***
C     ***       INPUT AND OUTPUT PORTIONS MODIFIED AND         ***
C     ***        THE FOUT DATA STRUCTURE INTRODUCED BY         ***
C     ***             DIMITRI P. BERTSEKAS                     ***
C     ***                                                      ***
C     ***     QUESTIONS AND COMMENTS SHOULD BE DIRECTED TO     ***
C     ***     G. GALLO AND S. PALLOTTINO                       ***
C     ***     DEPT. OF COMP. SCI., UNIV. OF PISA, ITALY.       ***
C     ***     FAX 39-50-510247                                 ***
C     ***     EMAIL: PALLO@DIPISA.DI.UNIPI.IT                  ***
C     ************************************************************

C    ************************************************************
C    FINDS SHORTEST PATH FROM ONE ORIGIN TO ALL DESTINATIONS
C    ************************************************************
C
C ALL THE PARAMETERS ARE INTEGER
C
C THE ONLY MACHINE DEPENDENT CONSTANT USED IS INF
C
C***********************************************************************


      PARAMETER(MAXNODES=10000, MAXARCS=100000)

      INTEGER FOUT,NXTOUT,D,P,Q,R,HP,Y,X,T2,DEST
      REAL TT1,TT2,TCOST
      DIMENSION FOUT(MAXNODES),D(MAXNODES),P(MAXNODES),HP(MAXNODES)
      DIMENSION Q(MAXNODES)
      DIMENSION LNGT(MAXARCS),ND(MAXARCS),NXTOUT(MAXARCS)
      DATA INF/999999999/
```

```
      CALL READ(N,M,FOUT,NXTOUT,ND,LNGT)

      DO 10 I=1,N
        Q(I) = 0
        P(I) = 0
10    CONTINUE
C
C     INITIALIZE ORIGIN AND QUEUE OF DESTINATIONS
C

      PRINT*,'THE NUMBER OF NODES IS = ',N
      PRINT*,'ENTER THE ORIGIN NODE'
      READ*,R



      PRINT *,'CALLING DESOPO-PAPE TO SOLVE THE PROBLEM'
      PRINT*,'*****************************************'
C
C     GET STARTING TIME FOR THE MAC II
C
      TT1 = LONG(362)/60.0
C

        CALL L2QUE(FOUT,NXTOUT,ND,LNGT,D,P,Q,N,INF,R)
C
C     GET ENDING TIME FOR THE MAC II
C
      TT2 = LONG(362)/60.0 - TT1
      PRINT *,'FINISHED --- TOTAL CPU TIME', TT2,' SECS'
      PRINT*,'*****************************************'
      PRINT*,'THE ORIGIN NODE IS ',R
C

      PRINT*,'SH. DIST. OF DESTINATION ',N,' IS ',D(N)

45    PRINT*,'ENTER OTHER DESTINATION NODE (0 IF DONE)'
      READ*,DEST
      IF ((DEST.LT.0).OR.(DEST.GT.N)) GOTO 45
      IF (DEST.EQ.R) GOTO 45
      IF (DEST.NE.0) THEN
        PRINT*,'SHORTEST DISTANCE TO',DEST,' = ',D(DEST)
        GOTO 45
      END IF



      STOP
      END
C
C
      SUBROUTINE L2QUE(FOUT,NXTOUT,ND,LNGT,D,P,Q,N,INF,R)
C*************************************************************
```

```
C
C    ROUTINE L2QUE
C
C 1) FINDS A SHORTEST PATH TREE ROOTED AT NODE R AND THE SHORTEST
C    DISTANCES
C 2) IS BASED ON THE D'ESOPO-PAPE METHOD WITH THE SET Q IMPLEMENTED
C    AS A DOUBLE QUEUE Q(.)
C
C MEANING OF THE INPUT PARAMETERS:
C
C FOUT(I)    = POINTER TO ARC-LIST OF NODE I, I=1,2,...,N+1
C ND(J)   = ENDING NODE OF ARC J, J=1,2,...,M
C LNGT(J) = LENGTH OF ARC J, J=1,2,...,M
C NMAX    = DIMENSION OF ARRAYS A(.), D(.), P(.), Q(.)
C MMAX    = DIMENSION OF ARRAYS ND(.), LNGT(.)
C N       = NUMBER OF NODES
C INF     = VERY LARGE INTEGER VALUE (INFINITY)
C R       = ROOT
C
C MEANING OF THE OUTPUT PARAMETERS:
C
C D(I)    = SHORTEST DISTANCE FROM R TO I, I=1,2,...,N
C P(I)    = PREDECESSOR NODE OF I IN THE SHORTEST PATH TREE, I=1,2,...,N
C
C OF THE MAIN INTERNAL PARAMETERS:
C
C Q(I) = LIST OF CANDIDATE NODES; Q(I) = -1 IF I IS NOT IN Q AND IT HAS
C          ALREADY BEEN SCANNED
C        = 0 IF I IS NOT IN Q AND IT HAS
C          NOT BEEN SCANNED
C        = J IF I PRECEDES NODE J IN THE
C          LIST
C NN   = N+1
C U    = CURRENT NODE
C V    = ENDING NODE OF THE CURRENT ARC
C INIT = START-POINTER TO THE ARC-LIST OF THE CURRENT NODE
C IFIN = END-POINTER TO THE ARC-LIST OF THE CURRENT NODE
C DV   = TENTATIVE LABEL OF NODE V
C LAST = POINTER TO THE LAST NODE OF Q(.)
C PNTR = POINTER TO THE LAST NODE OF THE FIRST QUEUE OF Q(.)
C
C ALL THE PARAMETERS ARE INTEGER
C
C*****************************************************************


      PARAMETER(MAXNODES=10000, MAXARCS=100000)

      INTEGER FOUT,NXTOUT,D,P,Q,R,U,V,DV,PNTR
      DIMENSION FOUT(MAXNODES),D(MAXNODES),P(MAXNODES),Q(MAXNODES)
      DIMENSION ND(MAXARCS),LNGT(MAXARCS),NXTOUT(MAXARCS)
C
C INITIALIZE
C
```

```
      DO 10 I=1,N
        Q(I) = 0
        D(I) = INF
  10  CONTINUE
      Q(R) = - 1
      D(R)=0
      P(R) = 0
      NN = N + 1
      Q(NN) = NN
      LAST = NN
      PNTR = NN
      U = R
C
C
C EXPLORE THE FORWARD STAR OF  U
C
20    CONTINUE

      J=FOUT(U)
25    IF (J.GT.0) THEN
        V = ND(J)
        DV = D(U) + LNGT(J)
C
C CHECK WHETHER THE LABEL OF V CAN BE IMPROVED
C
        IF ( D(V) .GT. DV ) THEN
          D(V) = DV
          P(V) = U
          IF ( Q(V) ) 30,40,50
C
C IF V IS NOT IN Q AND IT HAS ALREADY BEEN SCANNED, IT IS INSERTED  AT
C THE POSITION POINTED BY PNTR
C
30        Q(V) = Q(PNTR)
          Q(PNTR) = V
          IF ( LAST .EQ. PNTR ) LAST=V
          PNTR = V
          GO TO 50
C
C IF V IS NOT IN Q AND IT WAS NEVER SCANNED, IT IS INSERTED AT THE TAIL
C OF Q
C
40        Q(LAST) = V
          Q(V) = NN
          LAST = V
50        CONTINUE
        END IF
        J=NXTOUT(J)
        GOTO 25
       END IF
C
C REMOVE THE NEW CURRENT NODE U
C
60     U = Q(NN)
```

```
       Q(NN) = Q(U)
       Q(U) = - 1
       IF ( LAST .EQ. U ) LAST = NN
       IF ( PNTR .EQ. U ) PNTR = NN
C
C CHECK WHETHER THE LIST IS EMPTY
C
       IF ( U .LE. N ) GO TO 20

       RETURN
       END

      SUBROUTINE READ(N,M,FOUT,NXTOUT,END,LNGT)
C*****************************************************************
C READS THE GRAPH DATA (STORED AS AN ADJACENCE LIST) AND THE ORIGINS
C LIST.
C*****************************************************************


      PARAMETER(MAXNODES=10000, MAXARCS=100000)

      IMPLICIT INTEGER (A-Z)
      DIMENSION FOUT(MAXNODES),LAST(MAXNODES)
      DIMENSION START(MAXARCS),END(MAXARCS),LNGT(MAXARCS)
      DIMENSION NXTOUT(MAXARCS)
C
      PRINT*,'READING THE SHORTEST PATH PROBLEM DATA'
      OPEN(13,FILE='FOR013.DAT',STATUS='OLD')
      REWIND(13)

C     READ NUMBER OF NODES AND ARCS

      READ(13,1010) N,M

C     READ ENDNODE AND LENGTH OF EACH ARC

      DO 20 I=1,M
        READ(13,1020) START(I),END(I),LNGT(I),DUMMY
20    CONTINUE

      DO 30 I=1,N
        READ(13,1000) DUMMY
30    CONTINUE

      ENDFILE(13)
      REWIND(13)

      PRINT*,'END OF READING'

1000  FORMAT(1I8)
1010  FORMAT(2I8)
1020  FORMAT(4I8)
```

```
      PRINT *, 'RESTRUCTURING THE DATA '
C
C     GENERATE FORWARD AND BACKWARD STARS FOR EACH NODE
C
      DO 60 I=1,N
        FOUT(I)=0
        LAST(I)=0
60    CONTINUE

      DO 65 ARC=1,M
        NXTOUT(ARC)=0
        NODE=START(ARC)
        IF (FOUT(NODE).NE.0) THEN
          NXTOUT(LAST(NODE))=ARC
        ELSE
          FOUT(NODE)=ARC
        END IF
        LAST(NODE)=ARC
65    CONTINUE

      RETURN

      END
```

HEAP-ALL-DEST

This is an implementation of the single origin/all destinations label setting method, using a binary heap to maintain the candidate list (cf.\ Section 1.3.2). This code is adapted from the SHEAP code of Gallo and Pallotino [GaP88].

```
C     ******* SAMPLE CALLING PROGRAM FOR SUBROUTINE SHEAP  *******
C     ***              (SHORTEST PATH PROBLEM)                  ***
C     ***                                                       ***
C     ***          THE PROGRAM IS BASED ON THE PAPER            ***
C     *** G. GALLO, S. PALLOTTINO "SHORTEST PATH ALGORITHMS",   ***
C     ***       ANNALS OF OPERATIONS RESEARCH 7, 1988.          ***
C     ***                                                       ***
C     ***       INPUT AND OUTPUT PORTIONS MODIFIED AND          ***
C     ***        THE FOUT DATA STRUCTURE INTRODUCED BY          ***
C     ***             DIMITRI P. BERTSEKAS                      ***
C     ***                                                       ***
C     ***     QUESTIONS AND COMMENTS SHOULD BE DIRECTED TO      ***
C     ***     G. GALLO AND S. PALLOTTINO                        ***
C     ***     DEPT. OF COMP. SCI., UNIV. OF PISA, ITALY.        ***
C     ***     FAX 39-50-510247                                  ***
C     ***     EMAIL: PALLO@DIPISA.DI.UNIPI.IT                   ***
C     ***********************************************************

C     ***********************************************************
C     FINDS SHORTEST PATH FROM ONE ORIGIN TO ALL DESTINATIONS
C     ***********************************************************
```

```
C
C ALL THE PARAMETERS ARE INTEGER
C
C THE ONLY MACHINE DEPENDENT CONSTANT USED IS INF
C
C*********************************************************************


      PARAMETER(MAXNODES=10000, MAXARCS=100000)

      INTEGER FOUT,NXTOUT,D,P,Q,R,HP,Y,X,T2,DEST
      REAL TT1,TT2,TCOST
      DIMENSION FOUT(MAXNODES),D(MAXNODES),P(MAXNODES)
      DIMENSION HP(MAXNODES)
      DIMENSION Q(MAXNODES)
      DIMENSION LNGT(MAXARCS),ND(MAXARCS),NXTOUT(MAXARCS)
      DATA INF/999999999/

      CALL READ(N,M,FOUT,NXTOUT,ND,LNGT)

      DO 10 I=1,N
        Q(I) = 0
        P(I) = 0
10    CONTINUE
C
C    INITIALIZE ORIGIN AND QUEUE OF DESTINATIONS
C

      PRINT*,'THE NUMBER OF NODES IS = ',N
      PRINT*,'ENTER THE ORIGIN NODE'
      READ*,R



      PRINT *,'CALLING DIJKSTRA/HEAP TO SOLVE THE PROBLEM'
      PRINT*,'******************************************'
C
C     GET STARTING TIME FOR THE MAC II
C
      TT1 = LONG(362)/60.0
C

      CALL SHEAP(FOUT,NXTOUT,ND,LNGT,D,P,Q,HP,N,INF,R)
C
C     GET ENDING TIME FOR THE MAC II
C
      TT2 = LONG(362)/60.0 - TT1
      PRINT *,'FINISHED --- TOTAL CPU TIME', TT2,' SECS'
      PRINT*,'******************************************'
      PRINT*,'THE ORIGIN NODE IS ',R
C


      PRINT*,'SHORTEST DISTANCE TO DESTINATION',N,' = ',D(N)
```

```
45    PRINT*,'ENTER OTHER DESTINATION NODE (0 IF DONE)'
      READ*,DEST
      IF ((DEST.LT.0).OR.(DEST.GT.N)) GOTO 45
      IF (DEST.EQ.R) GOTO 45
      IF (DEST.NE.0) THEN
        PRINT*,'SHORTEST DISTANCE TO',DEST,' = ',D(DEST)
        GOTO 45
      END IF


      STOP
      END
C
C
      SUBROUTINE SHEAP(FOUT,NXTOUT,ND,LNGT,D,P,Q,HP,N,INF,R)

C*********************************************************************
C
C    ROUTINE  SHEAP
C
C 1) FINDS A SHORTEST PATH TREE ROOTED AT NODE R AND THE SHORTEST
C    DISTANCES
C 2) IS BASED ON DIJKSTRA'S METHOD, WITH PRIORITY QUEUE Q IMPLEMENTED
C    AS A BINARY HEAP
C
C MEANING OF THE INPUT PARAMETERS:
C
C FOUT(I)    = POINTER TO ARC-LIST OF NODE I, I=1,2,...,N+1
C ND(J)   = ENDING NODE OF ARC J, J=1,2,...,M
C LNGT(J) = LENGTH OF ARC J, J=1,2,...,M
C N       = NUMBER OF NODES
C INF     = VERY LARGE INTEGER VALUE (INFINITY)
C R       = ROOT
C
C MEANING OF THE OUTPUT PARAMETERS:
C
C D(I)    = SHORTEST DISTANCE FROM R TO I, I=1,2,...,N
C             I IN THE SHORTEST PATH TREE, I=1,2,...,N
C
C MEANING OF THE MAIN INTERNAL PARAMETERS:
C
C Q(I) = DICTIONARY OF THE HEAP: Q(I) GIVES THE POSITION OF NODE
C        I IN THE HEAP HP(.), I=1,2,...,N
C HP(I)= I-TH NODE IN THE HEAP, I=1,2,...,NHP
C NHP  = NUMBER OF NODES IN THE HEAP (NHP<=N)
C NN   = N+1
C U    = CURRENT NODE
C V    = ENDING NODE OF THE CURRENT ARC
C INIT = START-POINTER TO THE ARC-LIST OF THE CURRENT NODE
C IFIN = END-POINTER TO THE ARC-LIST OF THE CURRENT NODE
C DV   = TENTATIVE LABEL OF NODE V
C
C ALL THE PARAMETERS ARE INTEGER
```

```
C
C*********************************************************************


      PARAMETER(MAXNODES=10000, MAXARCS=100000)

      INTEGER FOUT,D,P,Q,R,U,V,DV,HP,DP1
      INTEGER HP1,HP2,HP3
      DIMENSION FOUT(MAXNODES),D(MAXNODES),P(MAXNODES)
      DIMENSION Q(MAXNODES),HP(MAXNODES)
      DIMENSION ND(MAXARCS),LNGT(MAXARCS),NXTOUT(MAXARCS)
C
C INITIALIZE
C
      DO 10 I=1,N
        D(I) = INF
10    CONTINUE
      NHP = 0
      D(R) = 0
      P(R) = 0
      NN = N + 1
      U = R
C
C EXPLORE THE FORWARD STAR OF  U
C
20    CONTINUE

      J=FOUT(U)
25    IF (J.GT.0) THEN
        V = ND(J)
        DV = D(U) + LNGT(J)
C
C CHECK WHETHER THE LABEL OF V CAN BE IMPROVED
C
        IF ( D(V) .GT. DV ) THEN
          D(V) = DV
          P(V) = U
          IF ( Q(V) .NE. 0 ) GO TO 30
C
C INSERT NODE V INTO THE HEAP
C
          NHP = NHP + 1
          Q(V) = NHP
C
C UPDATE THE HEAP
C
30        K = Q(V)
40        K2 = K/2
          IF ( K2 .LE. 0 ) GO TO 50
          HP2 = HP(K2)
          IF ( DV .GE. D(HP2) ) GO TO 50
          HP(K) = HP2
          Q(HP2) = K
          K = K2
```

```
          GOTO 40
50        HP(K) = V
          Q(V) = K
         END IF
         J=NXTOUT(J)
         GOTO 25
        END IF
C
C REMOVE THE NEW CURRENT NODE U FROM THE HEAP
C CHECK IF IT IS THE DESTINATION
C
70    U= HP(1)
      Q(U) = 0
      NHP = NHP - 1
C
C CHECK WHETHER THE HEAP IS EMPTY
C
      IF ( NHP ) 130,20,80
C
C UPDATE  THE HEAP
C
80    HP1 = HP(NHP+1)
      DP1 = D(HP1)
      K = 1
90    K2 = 2*K
      HP2 = HP(K2)
      IF ( K2-NHP ) 100,110,120
100   HP3 = HP(K2+1)
      IF ( D(HP2) .LT. D(HP3) ) GO TO 110
      HP2 = HP3
      K2 = K2 + 1
110   IF ( DP1 .LE. D(HP2) ) GO TO 120
      HP(K) = HP2
      Q(HP2) = K
      K = K2
      GO TO 90
120   HP(K)=HP1
      Q(HP1) = K
      GO TO 20
130   CONTINUE
      RETURN
      END


      SUBROUTINE READ(N,M,FOUT,NXTOUT,END,LNGT)
C*****************************************************************
C READS THE GRAPH DATA (STORED AS AN ADJACENCE LIST) AND THE ORIGINS
C LIST.
C*****************************************************************


      PARAMETER(MAXNODES=10000, MAXARCS=100000)

      IMPLICIT INTEGER (A-Z)
```

```
      DIMENSION FOUT(MAXNODES),LAST(MAXNODES)
      DIMENSION START(MAXARCS),END(MAXARCS),LNGT(MAXARCS)
      DIMENSION NXTOUT(MAXARCS)
C
      PRINT*,'READING THE SHORTEST PATH PROBLEM DATA'
      OPEN(13,FILE='FOR013.DAT',STATUS='OLD')
      REWIND(13)

C     READ NUMBER OF NODES AND ARCS

      READ(13,1010) N,M

C     READ ENDNODE AND LENGTH OF EACH ARC

      DO 20 I=1,M
        READ(13,1020) START(I),END(I),LNGT(I),DUMMY
20    CONTINUE

      DO 30 I=1,N
        READ(13,1000) DUMMY
30    CONTINUE

      ENDFILE(13)
      REWIND(13)

      PRINT*,'END OF READING'

1000  FORMAT(1I8)
1010  FORMAT(2I8)
1020  FORMAT(4I8)


      PRINT *, 'RESTRUCTURING THE DATA '
C
C     GENERATE FORWARD AND BACKWARD STARS FOR EACH NODE
C
      DO 60 I=1,N
        FOUT(I)=0
        LAST(I)=0
60    CONTINUE

      DO 65 ARC=1,M
        NXTOUT(ARC)=0
        NODE=START(ARC)
        IF (FOUT(NODE).NE.0) THEN
          NXTOUT(LAST(NODE))=ARC
        ELSE
          FOUT(NODE)=ARC
        END IF
        LAST(NODE)=ARC
65    CONTINUE

      RETURN
```

```
      END
```

HEAP-SELECT-DEST

This code solves the shortest path problem with a
single origin and a selected number of destinations. It is the same as the
earlier code HEAP-ALL-DEST, except that it stops when all destinations
have
been permanently labeled.

```
C     ******* SAMPLE CALLING PROGRAM FOR SUBROUTINE SHEAP  *******
C     ***              (SHORTEST PATH PROBLEM)                  ***
C     ***                                                       ***
C     ***          THE PROGRAM IS BASED ON THE PAPER            ***
C     *** G. GALLO, S. PALLOTTINO "SHORTEST PATH ALGORITHMS",   ***
C     ***       ANNALS OF OPERATIONS RESEARCH 7, 1988.          ***
C     ***                                                       ***
C     ***       INPUT AND OUTPUT PORTIONS MODIFIED AND          ***
C     ***        THE FOUT DATA STRUCTURE INTRODUCED BY          ***
C     ***             DIMITRI P. BERTSEKAS                      ***
C     ***                                                       ***
C     ***     QUESTIONS AND COMMENTS SHOULD BE DIRECTED TO      ***
C     ***     G. GALLO AND S. PALLOTTINO                        ***
C     ***     DEPT. OF COMP. SCI., UNIV. OF PISA, ITALY.        ***
C     ***     FAX 39-50-510247                                  ***
C     ***     EMAIL: PALLO@DIPISA.DI.UNIPI.IT                   ***
C     ***********************************************************

C    ************************************************************
C    FINDS SHORTEST PATH FROM ONE ORIGIN TO SEVERAL DESTINATIONS
C    ************************************************************
C
C ALL THE PARAMETERS ARE INTEGER
C
C THE ONLY MACHINE DEPENDENT CONSTANT USED IS INF
C
C*********************************************************************


      PARAMETER(MAXNODES=10000, MAXARCS=100000)

      INTEGER FOUT,D,P,Q,R,HP,Y,X,T2,DEST_COUNT,SH_DIST,DEST
      REAL TT1,TT2,TCOST
      DIMENSION FOUT(MAXNODES),D(MAXNODES),P(MAXNODES)
      DIMENSION HP(MAXNODES)
      DIMENSION Q(MAXNODES),SH_DIST(MAXNODES)
      DIMENSION LNGT(MAXARCS),ND(MAXARCS),NXTOUT(MAXARCS)
      DATA INF/999999999/

      CALL READ(N,M,FOUT,NXTOUT,ND,LNGT)

      DO 10 I=1,N
```

```
        Q(I) = 0
        P(I) = 0
        SH_DIST(I)=-1
10    CONTINUE
C
C     INITIALIZE ORIGIN AND QUEUE OF DESTINATIONS
C

      PRINT*,'THE NUMBER OF NODES IS = ',N
      PRINT*,'ENTER THE ORIGIN NODE'
      READ*,R


42    PRINT*,'ENTER FIRST DESTINATION NODE'
      READ*,DEST

      IF ((DEST.LE.0).OR.(DEST.GT.N)) GOTO 42
      IF (DEST.EQ.R) GOTO 42


      IF ((DEST.LE.0).OR.(DEST.GE.N)) DEST =N
      SH_DIST(DEST)=0
      DEST_COUNT=1

45    PRINT*,'ENTER NEXT DESTINATION NODE (0 IF DONE)'
      READ*,DEST
      IF ((DEST.LT.0).OR.(DEST.GT.N).OR.(DEST.EQ.R)) GOTO 45
      IF (DEST.NE.0) THEN
        IF (SH_DIST(DEST).LT.0) THEN
          DEST_COUNT=DEST_COUNT+1
          SH_DIST(DEST)=0
        END IF
        GOTO 45
      END IF

50    CONTINUE

      PRINT*,'TOTAL NUMBER OF DESTINATIONS = ',DEST_COUNT


      PRINT *,'CALLING DIJKSTRA/HEAP TO SOLVE THE PROBLEM'
      PRINT*,'*****************************************'
C
C     GET STARTING TIME FOR THE MAC II
C
      TT1 = LONG(362)/60.0
C

      CALL SHEAP(FOUT,NXTOUT,ND,LNGT,D,P,Q,HP,N,INF,R,
     &DEST_COUNT,SH_DIST)
C
C     GET ENDING TIME FOR THE MAC II
C
      TT2 = LONG(362)/60.0 - TT1
```

```
      PRINT *,'FINISHED --- TOTAL CPU TIME', TT2,' SECS'
      PRINT*,'*****************************************'
      PRINT*,'THE ORIGIN NODE IS ',R
C

        DO 200 I=1,N
          IF (SH_DIST(I).GE.0) THEN
            IF (SH_DIST(I).NE.D(I)) THEN
              PRINT*,'ERROR'
            END IF
            PRINT*,'SH. DIST. OF DESTINATION ',I,' IS ',SH_DIST(I)
          END IF
200     CONTINUE


      PRINT*,'*****************************************'
      PRINT *, 'PROGRAM ENDED; PRESS <CR>'
      PAUSE
      STOP
      END
C
C
      SUBROUTINE SHEAP(FOUT,NXTOUT,ND,LNGT,D,P,Q,HP,N,INF,R,
     &DEST_COUNT,SH_DIST)
C**********************************************************************
C
C    ROUTINE  SHEAP
C
C 1) FINDS A SHORTEST PATH TREE ROOTED AT NODE R AND THE SHORTEST
C    DISTANCES
C 2) IS BASED ON DIJKSTRA'S METHOD, WITH PRIORITY QUEUE Q IMPLEMENTED
C    AS A BINARY HEAP
C
C MEANING OF THE INPUT PARAMETERS:
C
C FOUT(I)    = POINTER TO ARC-LIST OF NODE I, I=1,2,...,N+1
C ND(J)   = ENDING NODE OF ARC J, J=1,2,...,M
C LNGT(J) = LENGTH OF ARC J, J=1,2,...,M
C N       = NUMBER OF NODES
C INF     = VERY LARGE INTEGER VALUE (INFINITY)
C R       = ROOT
C
C MEANING OF THE OUTPUT PARAMETERS:
C
C D(I)    = SHORTEST DISTANCE FROM R TO I, I=1,2,...,N
C             I IN THE SHORTEST PATH TREE, I=1,2,...,N
C
C MEANING OF THE MAIN INTERNAL PARAMETERS:
C
C Q(I) = DICTIONARY OF THE HEAP: Q(I) GIVES THE POSITION OF NODE
C        I IN THE HEAP HP(.), I=1,2,...,N
C HP(I)= I-TH NODE IN THE HEAP, I=1,2,...,NHP
C NHP  = NUMBER OF NODES IN THE HEAP (NHP<=N)
C NN   = N+1
```

```
C U    = CURRENT NODE
C V    = ENDING NODE OF THE CURRENT ARC
C INIT = START-POINTER TO THE ARC-LIST OF THE CURRENT NODE
C IFIN = END-POINTER TO THE ARC-LIST OF THE CURRENT NODE
C DV   = TENTATIVE LABEL OF NODE V
C
C ALL THE PARAMETERS ARE INTEGER
C
C**********************************************************************


      PARAMETER(MAXNODES=10000, MAXARCS=100000)

      INTEGER FOUT,D,P,Q,R,U,V,DV,HP,DP1
      INTEGER HP1,HP2,HP3,DEST_COUNT,SH_DIST
      DIMENSION FOUT(MAXNODES),D(MAXNODES),P(MAXNODES)
      DIMENSION Q(MAXNODES),HP(MAXNODES)
      DIMENSION ND(MAXARCS),LNGT(MAXARCS),NXTOUT(MAXARCS)
      DIMENSION SH_DIST(MAXNODES)
C
C INITIALIZE
C
      DO 10 I=1,N
        D(I) = INF
10    CONTINUE
      NHP = 0
      D(R) = 0
      P(R) = 0
      NN = N + 1
      U = R
C
C EXPLORE THE FORWARD STAR OF  U
C
20    CONTINUE

      J=FOUT(U)
25    IF (J.GT.0) THEN
        V = ND(J)
        DV = D(U) + LNGT(J)
C
C CHECK WHETHER THE LABEL OF V CAN BE IMPROVED
C
        IF ( D(V) .GT. DV ) THEN
          D(V) = DV
          P(V) = U
          IF ( Q(V) .NE. 0 ) GO TO 30
C
C INSERT NODE V INTO THE HEAP
C
          NHP = NHP + 1
          Q(V) = NHP
C
C UPDATE THE HEAP
C
```

```
30          K = Q(V)
40          K2 = K/2
            IF ( K2 .LE. 0 ) GO TO 50
            HP2 = HP(K2)
            IF ( DV .GE. D(HP2) ) GO TO 50
            HP(K) = HP2
            Q(HP2) = K
            K = K2
            GOTO 40
50          HP(K) = V
            Q(V) = K
          END IF
          J=NXTOUT(J)
          GOTO 25
        END IF
C
C REMOVE THE NEW CURRENT NODE U FROM THE HEAP
C CHECK IF IT IS A DESTINATION
C
70    U= HP(1)
      IF (SH_DIST(U).GE.0) THEN
        SH_DIST(U)=D(U)
        DEST_COUNT=DEST_COUNT-1
        IF (DEST_COUNT.EQ.0) GOTO 130
      END IF
      Q(U) = 0
      NHP = NHP - 1
C
C CHECK WHETHER THE HEAP IS EMPTY
C
      IF ( NHP ) 130,20,80
C
C UPDATE  THE HEAP
C
80    HP1 = HP(NHP+1)
      DP1 = D(HP1)
      K = 1
90    K2 = 2*K
      HP2 = HP(K2)
      IF ( K2-NHP ) 100,110,120
100   HP3 = HP(K2+1)
      IF ( D(HP2) .LT. D(HP3) ) GO TO 110
      HP2 = HP3
      K2 = K2 + 1
110   IF ( DP1 .LE. D(HP2) ) GO TO 120
      HP(K) = HP2
      Q(HP2) = K
      K = K2
      GO TO 90
120   HP(K)=HP1
      Q(HP1) = K
      GO TO 20
130   CONTINUE
      RETURN
```

```
      END


      SUBROUTINE READ(N,M,FOUT,NXTOUT,END,LNGT)
C*****************************************************************
C READS THE GRAPH DATA (STORED AS AN ADJACENCE LIST) AND THE ORIGINS
C LIST.
C*****************************************************************


      PARAMETER(MAXNODES=10000, MAXARCS=100000)

      IMPLICIT INTEGER (A-Z)
      DIMENSION FOUT(MAXNODES),LAST(MAXNODES)
      DIMENSION START(MAXARCS),END(MAXARCS),LNGT(MAXARCS)
      DIMENSION NXTOUT(MAXARCS)
C
      PRINT*,'READING THE SHORTEST PATH PROBLEM DATA'
      OPEN(13,FILE='FOR013.DAT',STATUS='OLD')
      REWIND(13)

C     READ NUMBER OF NODES AND ARCS

      READ(13,1010) N,M

C     READ ENDNODE AND LENGTH OF EACH ARC

      DO 20 I=1,M
        READ(13,1020) START(I),END(I),LNGT(I),DUMMY
20    CONTINUE

      DO 30 I=1,N
        READ(13,1000) DUMMY
30    CONTINUE

      ENDFILE(13)
      REWIND(13)

      PRINT*,'END OF READING'

1000  FORMAT(1I8)
1010  FORMAT(2I8)
1020  FORMAT(4I8)


      PRINT *, 'RESTRUCTURING THE DATA '
C
C     GENERATE FORWARD AND BACKWARD STARS FOR EACH NODE
C
      DO 60 I=1,N
        FOUT(I)=0
        LAST(I)=0
60    CONTINUE
```

```
      DO 65 ARC=1,M
        NXTOUT(ARC)=0
        NODE=START(ARC)
        IF (FOUT(NODE).NE.0) THEN
          NXTOUT(LAST(NODE))=ARC
        ELSE
          FOUT(NODE)=ARC
        END IF
        LAST(NODE)=ARC
65    CONTINUE

      RETURN

      END
```

AUCTION-SELECT-DEST

This code also solves the shortest path problem
with a single origin and a selected number of destinations. It implements
the
combined forward/backward auction algorithm of Section 4.3.

```
C     ************************************************************
C     ***  SAMPLE CALLING PROGRAM FOR SUBROUTINE AUCTION_SP    ***
C     ***            (FORWARD AND BACKWARD VERSION)            ***
C     *** SHORTEST PATH PROBLEM FROM ONE ORIGIN TO A SELECTED  ***
C     ***                SET OF DESTINATIONS                   ***
C     ***      ASSUMING ALL ARC LENGTHS ARE NONNEGATIVE        ***
C     ***                                                      ***
C     ***          BY DIMITRI BERTSEKAS, AUGUST '90            ***
C     ***       READS PROBLEM FILES IN STANDARD FORM           ***
C     ***                                                      ***
C     ************************************************************


      PARAMETER(MAXNODES=10000, MAXARCS=100000)

      IMPLICIT NONE
      INTEGER FOUT,P,LNGT,PRD,END,R,DEST,START,T2,LARGE,I,N,NA
      INTEGER PRDARC,COUNT,ITER,TERM,NUM_ORIG,NUM_EXT,NUM_SUCC
      INTEGER DPRDARC,FIN,NXTIN,NXTOUT,DTERM,DCOUNT
      INTEGER EXTEND_ARC,DEXTEND_ARC,NODE,ARC
      INTEGER FRSTQUEUE,NXTQUEUE,SH_DIST,CUR_DEST
      INTEGER PREVNODE,DEST_COUNT,DUMMY
      REAL*8 TT1,TT2

      COMMON /SCALARS/  N,LARGE,R,DEST
      COMMON /STATS/    ITER,NUM_ORIG,NUM_EXT,NUM_SUCC
      COMMON /BLK1/     START
      COMMON /BLK2/     END
      COMMON /LENGTH/   LNGT
      COMMON /FRSTOUT/  FOUT
```

```
      COMMON /FRSTIN/   FIN
      COMMON /NEXTOUT/  NXTOUT
      COMMON /NEXTIN/   NXTIN
      COMMON /PRICES/   P
      COMMON /PREDARC/  PRDARC
      COMMON /PRECARC/  DPRDARC
      COMMON /EXTARC/   EXTEND_ARC
      COMMON /DEXTARC/  DEXTEND_ARC
      COMMON /DTERM/    FRSTQUEUE,PREVNODE,DEST_COUNT
      COMMON /NXTQUEUE/ NXTQUEUE
      COMMON /SH_DIST/  SH_DIST
      DIMENSION FOUT(MAXNODES),NXTOUT(MAXARCS),P(MAXNODES),LNGT(MAXARCS)
      DIMENSION START(MAXARCS),END(MAXARCS),PRDARC(MAXNODES)
      DIMENSION EXTEND_ARC(MAXNODES),DEXTEND_ARC(MAXNODES)
      DIMENSION FIN(MAXNODES),NXTIN(MAXARCS),DPRDARC(MAXNODES)
      DIMENSION NXTQUEUE(MAXNODES),SH_DIST(MAXNODES)

      PRINT*,'AUCTION/SHORT. PATHS (1 ORIG., SEVERAL DEST.)'
      PRINT*,'*********************************************'
      PRINT *,'READING PROBLEM DATA'

      OPEN(13,FILE='FOR013.DAT',STATUS='OLD')
      REWIND(13)

C     READ NUMBER OF NODES AND ARCS

      READ(13,1010) N,NA

C     READ START, END, COST, AND CAPACITY OF EACH ARC

      DO 20 I=1,NA
        READ(13,1020) START(I),END(I),LNGT(I),DUMMY
20    CONTINUE

C     READ SUPPLY OF EACH NODE

      DO 30 I=1,N
        READ(13,1000) DUMMY
30    CONTINUE

      ENDFILE(13)
      REWIND(13)

      PRINT*,'END OF READING'

1000  FORMAT(1I8)
1010  FORMAT(2I8)
1020  FORMAT(4I8)


      PRINT *, 'RESTRUCTURING THE DATA '
C
C     GENERATE FORWARD AND BACKWARD STARS FOR EACH NODE
C
```

```
      DO 60 I=1,N
        PRDARC(I)=0
        FOUT(I)=0
60    CONTINUE

      DO 65 ARC=1,NA
        NXTOUT(ARC)=0
        NODE=START(ARC)
        IF (FOUT(NODE).NE.0) THEN
          NXTOUT(PRDARC(NODE))=ARC
        ELSE
          FOUT(NODE)=ARC
        END IF
        PRDARC(NODE)=ARC
65    CONTINUE


      DO 125 I=1,N
        FIN(I)=0
        PRDARC(I)=0
125   CONTINUE
      DO 130 ARC=1,NA
        NXTIN(ARC)=0
        NODE=END(ARC)
        IF (FIN(NODE).NE.0) THEN
          NXTIN(PRDARC(NODE))=ARC
        ELSE
          FIN(NODE)=ARC
        END IF
        PRDARC(NODE)=ARC
130   CONTINUE

C     SET LARGE TO A LARGE INTEGER FOR YOUR MACHINE
C
      LARGE=200000000
C

C   INITIALIZE PRICES AND PREDECESSOR ARCS

      DO 140 I=1,N
        P(I) = 0
        PRDARC(I)=-1
        DPRDARC(I)=-1
        EXTEND_ARC(I)=-1
        DEXTEND_ARC(I)=-1
        NXTQUEUE(I)=-1
        SH_DIST(I)=-1
140   CONTINUE


C
C    INITIALIZE ORIGIN AND QUEUE OF DESTINATIONS
C
```

```
      PRINT*,'THE NUMBER OF NODES IS = ',N
150   PRINT*,'ENTER THE ORIGIN NODE'
      READ*,R
      IF ((R.LE.0).OR.(R.GT.N)) GOTO 150

41    CONTINUE


42    PRINT*,'ENTER FIRST DESTINATION NODE'
      READ*,DEST

      IF ((DEST.LE.0).OR.(DEST.GT.N)) GOTO 42
      IF (DEST.EQ.R) GOTO 42
      CUR_DEST=DEST
      FRSTQUEUE=DEST
      DEST_COUNT=1
      NXTQUEUE(DEST)=0

45    PRINT*,'ENTER NEXT DESTINATION NODE (0 IF DONE)'
      READ*,DEST
      IF ((DEST.LT.0).OR.(DEST.GT.N)) GOTO 45
      IF (DEST.EQ.R) GOTO 45
      IF (DEST.NE.0) THEN
        IF (NXTQUEUE(DEST).LT.0) THEN
          NXTQUEUE(CUR_DEST)=DEST
          CUR_DEST=DEST
          DEST_COUNT=DEST_COUNT+1
          NXTQUEUE(DEST)=0
        END IF
        GOTO 45
      ELSE
        NXTQUEUE(CUR_DEST)=FRSTQUEUE
        PREVNODE=CUR_DEST
      END IF



50    CONTINUE

      PRINT*,'TOTAL NUMBER OF DESTINATIONS = ',DEST_COUNT

      PRINT *,'CALLING SHORTEST PATH AUCTION'
      PRINT*,'****************************************'

C
C     GET STARTING TIME FOR THE MAC II
C
      TT1 = LONG(362)/60.0
      CALL AUCTION_SP
C
C     GET ENDING TIME FOR THE MAC II
C
      TT2 = LONG(362)/60.0 - TT1
      PRINT *,'FINISHED --- TOTAL CPU TIME', TT2,' SECS'
```

```
      PRINT*,'*****************************************'
      PRINT*,'THE ORIGIN NODE IS ',R
C
        DO 200 I=1,N
          IF (SH_DIST(I).GE.0) THEN
            PRINT*,'SH. DIST. OF DESTINATION ',I,' IS ',SH_DIST(I)
          END IF
200     CONTINUE

      PRINT*,'*****************************************'

      COUNT=0
      DCOUNT=0
      DO 100 I=1,N
        IF (PRDARC(I).GT.0) COUNT=COUNT+1
        IF (DPRDARC(I).GT.0) DCOUNT=DCOUNT+1
100   CONTINUE

      PRINT *, '# OF TERMINAL NODES REACHED FROM ORIGIN =',COUNT
      PRINT *, '# OF TERMINAL NODES REACHED FROM DESTINATIONS =',DCOUNT
X      PRINT *, '# OF EXTENSIONS AT THE ORIGIN =',NUM_ORIG
X      PRINT *, '# OF SPECULATIVE EXTENSION ATTEMPTS =',NUM_EXT
X      PRINT *, '# OF SPECULATIVE EXTENSION SUCCESSES =',NUM_SUCC
X      PRINT *, '# OF NODE SCANS =',ITER

      PRINT*,'*****************************************'
      PRINT*,'ENTER <0> TO STOP; <1> TO RUN WITH OTHER DESTINATIONS'
      READ*,DEST
      IF (DEST.EQ.1) THEN
        DO 400 I=1,N
          NXTQUEUE(I)=-1
          SH_DIST(I)=-1
400     CONTINUE
        GOTO 41
      END IF


      STOP

      END
C
C


      SUBROUTINE AUCTION_SP
C**********************************************************************
C
C ROUTINE AUCTION_SP
C
C FINDS A SHORTEST PATH FROM NODE R TO SEVERAL NODES
C ASSUMING ALL ARC LENGTHS ARE NONNEGATIVE
C
C MEANING OF VARIOUS VARIABLES:
C
```

```
C FOUT(I) = FIRST OUTGOING ARC FROM NODE I
C FIN(I)  = FIRST INCOMING ARC TO NODE I
C NXTOUT(J) = NEXT ARC WITH THE SAME END NODE AS J (=0 IF J IS LAST)
C NXTIN(J) = NEXT ARC WITH THE SAME END NODE AS J (=0 IF J IS LAST)
C START(J) = STARTING NODE OF ARC J
C END(J) = ENDING NODE OF ARC J
C LNGT(J) = LENGTH OF ARC J
C N = NUMBER OF NODES
C M = NUMBER OF ARCS
C LARGE = VERY LARGE INTEGER VALUE (INFINITY)
C R = ORIGIN
C DEST = CURRENT DESTINATION
C P(I) = PRICE OF I
C PRDARC(I) = PREDECESSOR OF I IN THE CURRENT FORWARD PATH
C TERM = TERMINAL NODE OF CURRENT FORWARD PATH
C DPRDARC(I) = PREDECESSOR OF I IN THE CURRENT BACKWARD PATH
C DTERM = TERMINAL NODE OF CURRENT BACKWARD PATH
C ENDNODE = ENDING NODE OF THE CURRENT ARC
C FRSTQUEUE = FIRST NODE IN THE QUEUE OF DESTINATIONS
C NXTQUEUE(I) = DESTINATION NEXT TO I IN THE QUEUE OF DESTINATIONS
C SH_DIST(I) = SHORTEST DISTANCE FROM ORIGIN TO NODE I
C EXTEND_ARC(I) = LIKELY CANDIDATE FOR OUTGOING ARC FROM I ON SHORTEST
PATH
C DEXTEND_ARC(I) = LIKELY CANDIDATE FOR INCOMING ARC TO I ON SHORTEST PATH
C
C ALL THE VARIABLES ARE INTEGER
C
C*********************************************************************


      PARAMETER(MAXNODES=10000, MAXARCS=100000)

      IMPLICIT NONE
      INTEGER FOUT,P,LNGT,PRD,SB,W,R,DEST,LARGE,N,M
      INTEGER TERM,ENDNODE,INIT,IFIN,LEVEL,NEW_LEVEL
      INTEGER I,J,START,END,ITER,NUM_ORIG,NUM_EXT,NUM_SUCC
      INTEGER EXTEND_ARC,BSTLEVEL,PTERM,PRDARC,EXTARC
      INTEGER DEXTEND_ARC,DTERM,DBSTLEVEL,DPTERM,DPRDARC,DEXTARC
      INTEGER FIN,NXTIN,NXTOUT,STARTNODE
      INTEGER FRSTQUEUE,NXTQUEUE,SH_DIST,PREVNODE,DEST_COUNT

      COMMON /SCALARS/  N,LARGE,R,DEST
      COMMON /STATS/    ITER,NUM_ORIG,NUM_EXT,NUM_SUCC
      COMMON /BLK1/     START
      COMMON /BLK2/     END
      COMMON /LENGTH/   LNGT
      COMMON /FRSTOUT/  FOUT
      COMMON /FRSTIN/   FIN
      COMMON /NEXTOUT/  NXTOUT
      COMMON /NEXTIN/   NXTIN
      COMMON /PRICES/   P
      COMMON /PREDARC/  PRDARC
      COMMON /PRECARC/  DPRDARC
      COMMON /EXTARC/   EXTEND_ARC
```

```
      COMMON /DEXTARC/  DEXTEND_ARC
      COMMON /DTERM/    FRSTQUEUE,PREVNODE,DEST_COUNT
      COMMON /NXTQUEUE/ NXTQUEUE
      COMMON /SH_DIST/  SH_DIST
      DIMENSION FOUT(MAXNODES),NXTOUT(MAXARCS),P(MAXNODES),LNGT(MAXARCS)
      DIMENSION START(MAXARCS),END(MAXARCS),PRDARC(MAXNODES)
      DIMENSION FIN(MAXNODES),NXTIN(MAXARCS),DPRDARC(MAXNODES)
      DIMENSION EXTEND_ARC(MAXNODES),DEXTEND_ARC(MAXNODES)
      DIMENSION NXTQUEUE(MAXNODES),SH_DIST(MAXNODES)


X      ITER=0
X      NUM_ORIG=0
X      NUM_EXT=0
X      NUM_SUCC=0
       DEST=FRSTQUEUE
       TERM=R
       DTERM=DEST
       GOTO 110

C  ***************************************************
C
C    ************ MAIN FORWARD ALGORITHM **********
C
C  ***************************************************


C     START OF A NEW ITERATION

20    CONTINUE

      PTERM=P(TERM)
      EXTARC=EXTEND_ARC(TERM)

      IF (EXTARC.LT.0) THEN
        EXTARC=DPRDARC(TERM)
        IF (EXTARC.LT.0) GO TO 100
      END IF

C     SPECULATIVE PATH EXTENSION ATTEMPT

X      NUM_EXT=NUM_EXT+1
      ENDNODE=END(EXTARC)
      BSTLEVEL=LNGT(EXTARC)+P(ENDNODE)
      IF (PTERM.EQ.BSTLEVEL) THEN
X         NUM_SUCC=NUM_SUCC+1
         TERM=ENDNODE
         PRDARC(TERM)=EXTARC

C     IF A NEW DESTINATION IS FOUND, SWITCH TO THE BACKWARD ALGORITHM

         IF (NXTQUEUE(TERM).GE.0) THEN
           IF (SH_DIST(TERM).LT.0) THEN
             SH_DIST(TERM)=P(R)-P(TERM)
```

```
             GOTO 110
           END IF
         END IF

C     RETURN FOR ANOTHER ITERATION

         GO TO 20
       END IF
C
C   EXTENSION ATTEMPT WAS UNSUCCESSFUL, SO SCAN TERMINAL NODE
C

100   CONTINUE
X      ITER=ITER+1
       BSTLEVEL=LARGE
       J=FOUT(TERM)
200    IF (J.GT.0) THEN
        NEW_LEVEL = P(END(J)) + LNGT(J)
        IF (NEW_LEVEL.LT.BSTLEVEL) THEN
          BSTLEVEL=NEW_LEVEL
          EXTARC=J
        END IF
        J=NXTOUT(J)
        GOTO 200
       END IF
       EXTEND_ARC(TERM)=EXTARC



       IF (BSTLEVEL.GT.PTERM) THEN

C         PATH CONTRACTION

          P(TERM)=BSTLEVEL
          IF (TERM.NE.R) THEN
            TERM=START(PRDARC(TERM))

C  RETURN FOR ANOTHER ITERATION BUT SKIP THE SPECULATIVE EXTENSION ATTEMPT

            PTERM=P(TERM)
            GO TO 100
          ELSE

C        ORIGIN REACHED; SWITCH TO THE BACKWARD ALGORITHM

X           NUM_ORIG=NUM_ORIG+1
            IF (PTERM.GE.LARGE) THEN
              PRINT *,'NO PATH TO THE DESTINATION'
              PAUSE
              STOP
            END IF
            GO TO 110
          END IF
       ELSE
```

```
C        PATH EXTENSION

         TERM=END(EXTARC)
         PRDARC(TERM)=EXTARC

C     IF A NEW DESTINATION IS FOUND, SWITCH TO THE BACKWARD ALGORITHM

         IF (NXTQUEUE(TERM).GE.0) THEN
           IF (SH_DIST(TERM).LT.0) THEN
             SH_DIST(TERM)=P(R)-P(TERM)
             GOTO 110
           END IF
         END IF


C     RETURN FOR ANOTHER ITERATION

         GO TO 20
        END IF


C  ***************************************************
C
C    ************ MAIN BACKWARD ALGORITHM **********
C
C  ***************************************************

C    CLEANUP THE QUEUE

110    CONTINUE

115    IF (SH_DIST(DEST).GE.0) THEN
         DTERM=NXTQUEUE(DEST)
         NXTQUEUE(DEST)=-1
         DEST=DTERM
         NXTQUEUE(PREVNODE)=DEST
         DEST_COUNT=DEST_COUNT-1

C     IF THE DESTINATIONS HAVE BEEN EXHAUSTED RETURN

         IF (DEST_COUNT.EQ.0) THEN
           RETURN
         ELSE
           GOTO 115
         END IF
       END IF


C     START OF A NEW ITERATION

120    CONTINUE

       DPTERM=P(DTERM)
```

```
      DEXTARC=DEXTEND_ARC(DTERM)

      IF (DEXTARC.LT.0) THEN
        DEXTARC=PRDARC(DTERM)
        IF (DEXTARC.LT.0) GO TO 1100
      END IF

C     PATH EXTENSION

X      NUM_EXT=NUM_EXT+1
      STARTNODE=START(DEXTARC)
      DBSTLEVEL=P(STARTNODE)-LNGT(DEXTARC)
      IF (DPTERM.EQ.DBSTLEVEL) THEN
X         NUM_SUCC=NUM_SUCC+1
         DTERM=STARTNODE
         DPRDARC(DTERM)=DEXTARC

C     IF THE ORIGIN IS FOUND, SWITCH TO THE FORWARD ALGORITHM

         IF (DTERM.EQ.R) THEN
           SH_DIST(DEST)=P(R)-P(DEST)
           DEST_COUNT=DEST_COUNT-1

C     IF THE DESTINATIONS HAVE BEEN EXHAUSTED RETURN

           IF (DEST_COUNT.EQ.0) THEN
             RETURN
           ELSE
             DTERM=NXTQUEUE(DEST)
             NXTQUEUE(DEST)=-1
             DEST=DTERM
             NXTQUEUE(PREVNODE)=DEST
             GOTO 20
           END IF
         END IF

C     RETURN FOR ANOTHER ITERATION

         GO TO 120
      END IF
C
C SCAN TERMINAL NODE
C
1100   CONTINUE
X      ITER=ITER+1
       DBSTLEVEL=-LARGE
       J=FIN(DTERM)
1200   IF (J.GT.0) THEN
        STARTNODE = START(J)
        NEW_LEVEL = P(STARTNODE) - LNGT(J)
        IF (NEW_LEVEL.GT.DBSTLEVEL) THEN
          DBSTLEVEL=NEW_LEVEL
          DEXTARC=J
        END IF
```

```
         J=NXTIN(J)
         GOTO 1200
        END IF
        DEXTEND_ARC(DTERM)=DEXTARC


        IF (DBSTLEVEL.LT.DPTERM) THEN

C          PATH CONTRACTION

           P(DTERM)=DBSTLEVEL
           IF (DTERM.NE.DEST) THEN
             DTERM=END(DPRDARC(DTERM))

C            RETURN FOR ANOTHER SCAN

             DPTERM=P(DTERM)
             GO TO 1100
           ELSE

C            DESTINATION REACHED; SWITCH TO THE ORIGIN

X            NUM_ORIG=NUM_ORIG+1
             IF (DPTERM.LE.-LARGE) THEN
               PRINT *,'NO PATH TO THE ORIGIN'
               PAUSE
               STOP
             END IF
             PREVNODE=DEST
             DEST=NXTQUEUE(DEST)
             DTERM=DEST
             GO TO 20
           END IF
        ELSE

C         PATH EXTENSION

          DTERM=START(DEXTARC)
          DPRDARC(DTERM)=DEXTARC

C      IF THE ORIGIN IS FOUND, SWITCH TO THE FORWARD ALGORITHM

          IF (DTERM.EQ.R) THEN
            SH_DIST(DEST)=P(R)-P(DEST)
            DEST_COUNT=DEST_COUNT-1
            IF (DEST_COUNT.EQ.0) THEN
              RETURN
            ELSE
              DTERM=NXTQUEUE(DEST)
              NXTQUEUE(DEST)=-1
              DEST=DTERM
              NXTQUEUE(PREVNODE)=DEST
              GOTO 20
            END IF
```

```
         END IF

C     RETURN FOR ANOTHER ITERATION

         GO TO 120
        END IF



       END
```

```
*************************************************************************
*
```

Relaxation Code

```
*************************************************************************
*
```

This code (RELAX_QC) implements the relaxation method of Section 3.3. The code was written by the author in collaboration with Paul Tseng (with contributions by Jon Eckstein in the initialization part of the code). There are some differences between the code given here and similar codes of the RELAX family that have been released to the research community since 1987 ([BeT88], [BeT90]).

```
C     ******* SAMPLE CALLING PROGRAM FOR RELAXT_QC *******
C
C        THIS PROGRAM WILL READ A PROBLEM FILE IN A
C        STANDARD FORM, AND SOLVE IT USING RELAXT_QC
C
C        RELAXT-QC DIFFERS FROM THE CODE RELAXT-III AND OTHER EARLIER
C        RELAXATION CODES IN THAT IT MAINTAINS
C        THE SET OF NODES WITH NONZERO DEFICIT IN A FIFO QUEUE.
C        IT ALSO HAS THE OPTION OF INITIALIZATION USING A
C        CRASH PROCEDURE; THIS INITIALIZATION IS RECOMMENDED
C        FOR PROBLEMS THAT PROVE DIFFICULT USING THE DEFAULT
C        INITIALIZATION.
C
C     ************************************************
C
      PROGRAM MAIN
      IMPLICIT INTEGER (A-Z)
      REAL*8 TCOST,TT,TIMER
      INTEGER C(20000),X(20000),U(20000),RC(20000),B(8000)
      INTEGER STARTN(20000),ENDN(20000)
      INTEGER I1(8000),I2(8000),I3(8000),I4(20000),I5(8000)
      INTEGER I6(20000),I7(20000)
      INTEGER TFSTOU(8000),TNXTOU(20000),TFSTIN(8000),TNXTIN(20000)
```

```
      INTEGER NXTQUEUE(8000)
      LOGICAL*1 L1(8000),L2(8000)
      COMMON /ARRAYS/STARTN/ARRAYE/ENDN/ARRAYU/U/ARRAYX/X
     $/ARRAY9/RC/ARRAYB/B/BLK1/I1/BLK2/I2/BLK3/I3/BLK4/I4/BLK5/I5/BLK6
     $/I6/BLK7/I7/BLK8/L1/BLK9/L2
     $/L/N,NA,LARGE,NMULTINODE,ITER,NUM_AUGM,NUM_ASCNT,NUM_SP,CRASH
      COMMON /BLK10/TFSTOU/BLK11/TNXTOU/BLK12/TFSTIN/BLK13/TNXTIN
      COMMON /BLK14/NXTQUEUE


C     SET LARGE TO A LARGE INTEGER FOR YOUR MACHINE


      LARGE=500000000
      DANGER_THRESH=LARGE/10


      PRINT*,'RELAXATION METHOD FOR MIN COST FLOW'
      PRINT*,'**********************************'
      PRINT *,'READING PROBLEM DATA'
      OPEN(13,FILE='FOR013.DAT',STATUS='OLD')
      REWIND(13)

C     READ NUMBER OF NODES AND ARCS

      READ(13,1010) N,NA

C     READ START, END, COST, AND CAPACITY OF EACH ARC

      DO 20 I=1,NA
        READ(13,1020) STARTN(I),ENDN(I),C(I),U(I)
20    CONTINUE

      FLAG1=0
      FLAG2=0
      FLAG3=0
      DO 25 I=1,NA
        IF (ABS(C(I)).GT.LARGE) FLAG1=1
        IF (U(I).GT.LARGE) FLAG2=1
        IF (ABS(C(I)).GT.DANGER_THRESH) FLAG3=1
25    CONTINUE

      IF (FLAG1.EQ.1) THEN
         PRINT*,'SOME COSTS EXCEED THE ALLOWABLE RANGE'
         PRINT *, 'PROGRAM CANNOT RUN; PRESS <CR> TO EXIT'
         PAUSE
         STOP
      END IF

      IF (FLAG2.EQ.1) THEN
         PRINT*,'SOME ARC CAPACITIES EXCEED THE ALLOWABLE RANGE'
         PRINT *, 'PROGRAM CANNOT RUN; PRESS <CR> TO EXIT'
         PAUSE
         STOP
```

```
      END IF

      IF (FLAG3.EQ.1) THEN
         PRINT *,'*******************************'
         PRINT*,'SOME COSTS ARE DANGEROUSLY LARGE'
         PRINT *, 'PROGRAM MAY NOT RUN CORRECTLY'
         PRINT *,'*******************************'
      END IF

C     READ SUPPLY OF EACH NODE; CONVERT IT TO DEMAND

      DO 30 I=1,N
        READ(13,1000) B(I)
        B(I)=-B(I)
30    CONTINUE

      ENDFILE(13)
      REWIND(13)

      PRINT*,'END OF READING'

1000  FORMAT(1I8)
1010  FORMAT(2I8)
1020  FORMAT(4I8)


      PRINT *, 'RESTRUCTURING THE DATA '
      CALL INIDAT

C     SET REDUCED COSTS EQUAL TO COSTS (INITIAL PRICES ARE 0)

      DO 40 I=1,NA
       RC(I)=C(I)
40    CONTINUE

C     ************* INITIALIZATION PARAMETER ************

C     SET CRASH EQUAL TO 1 TO ACTIVATE A SPECIAL CRASH PROCEDURE FOR
C     THE INITIAL PRICE-FLOW PAIR. THIS IS RECOMMENDED FOR DIFFICULT
C     PROBLEMS WHERE THE DEFAULT INITIALIZATION OF ZERO PRICES
C     RESULTS IN LONG RUNNING TIMES.

      CRASH=1

      PRINT *,'*********************************************'
      PRINT *, 'SOLVING THE PROBLEM '

C    READ MAC II TIME

      TIMER = LONG(362)/60.0

C    CALL MAIN RELAXATION ROUTINE

      CALL RELAXT_QC
```

```
C    READ MAC II TIME

      TT = LONG(362)/60.0 - TIMER
      PRINT*, 'TOTAL TIME = ',TT, ' SECS.'

C    CHECK CORRECTNESS OF THE ANSWER

      DO 80 NODE=1,N
        IF (B(NODE).NE.0) THEN
          PRINT*,'NONZERO SURPLUS AT NODE ',NODE
        ENDIF
80    CONTINUE
      DO 90 ARC=1,NA
        IF (X(ARC).GT.0) THEN
          IF (RC(ARC).GT.0) THEN
            PRINT*,'CS VIOLATED AT ARC ',ARC
          ENDIF
        ENDIF
        IF (U(ARC).GT.0) THEN
          IF (RC(ARC).LT.0) THEN
            PRINT*,'CS VIOLATED AT ARC ',ARC
          ENDIF
        ENDIF
90    CONTINUE

      TCOST=DBLE(FLOAT(0))
      DO 100 I=1,NA
        TCOST=TCOST+DBLE(FLOAT(X(I)*C(I)))
100   CONTINUE

      WRITE(9,1100) TCOST
1100  FORMAT(' ','OPTIMAL COST         =',F14.2)

      PRINT *,'NUMBER OF INITIAL SH. PATH AUGMENTATIONS = ',NUM_SP
      PRINT *,'NUMBER OF ITERATIONS = ',ITER
      PRINT *,'NUMBER OF MULTINODE ITERATIONS = ',NMULTINODE
      PRINT *,'NUMBER OF MULTINODE ASCENT STEPS = ',NUM_ASCNT
      PRINT *,'NUMBER OF AUGMENTATIONS = ',NUM_AUGM
      PRINT *,'*********************************************'
      PRINT *, 'PROGRAM ENDED; PRESS <CR> TO EXIT'

      PAUSE
      END




C     ***************************************************

      SUBROUTINE INIDAT
```

```
C     This subroutine uses the data arrays STARTN and ENDN
C     to construct auxiliary data arrays FOU, NXTOU, FIN, and
C     NXTIN that are required by RELAXT.  In this subroutine we
C     arbitrarily order the arcs leaving each node and store
C     this information in FOU and NXTOU.  Similarly, we arbitra-
C     rilly order the arcs entering each node and store this
C     information in FIN and NXTIN.  At the completion of the
C     construction, we have that

C         FOU(I)     = First arc leaving node I.
C         NXTOU(J)   = Next arc leaving the head node of arc J.
C         FIN(I)     = First arc entering node I.
C         NXTIN(J)   = Next arc entering the tail node of arc J.

      COMMON /ARRAYS/STARTN/ARRAYE/ENDN/BLK1/TEMPIN/BLK2/TEMPOU
     $/BLK3/FOU/BLK4/NXTOU/BLK5/FIN/BLK6/NXTIN
     $/L/N,NA
      INTEGER STARTN(1),ENDN(1),TEMPIN(1),TEMPOU(1),FOU(1)
      INTEGER NXTOU(1),FIN(1),NXTIN(1)
      LOGICAL*1 L

      DO 25 I=1,N
        FIN(I)=0
        FOU(I)=0
        TEMPIN(I)=0
25      TEMPOU(I)=0

      DO 27 I=1,NA
        NXTIN(I)=0
        NXTOU(I)=0
        I1=STARTN(I)
        I2=ENDN(I)
        IF (FOU(I1).NE.0) THEN
          NXTOU(TEMPOU(I1))=I
        ELSE
          FOU(I1)=I
        END IF
        TEMPOU(I1)=I
        IF (FIN(I2).NE.0) THEN
          NXTIN(TEMPIN(I2))=I
        ELSE
          FIN(I2)=I
        END IF
27      TEMPIN(I2)=I
      RETURN
      END



C ***********************************************************************
C
C                            SUBROUTINE RELAXT_QC
C                          VERSION OF NOVEMBER 1991
C
```

```
C        RELAXT-QC DIFFERS FROM THE CODE RELAXT-III AND OTHER EARLIER
C        RELAXATION CODES IN THAT IT MAINTAINS
C        THE SET OF NODES WITH NONZERO DEFICIT IN A FIFO QUEUE.
C        IT ALSO HAS THE OPTION OF INITIALIZATION USING A
C        CRASH PROCEDURE; THIS INITIALIZATION IS RECOMMENDED
C        FOR DIFFICULT PROBLEMS.
C
C ***************************************************************
C
C     This subroutine solves the minimum (linear) cost ordinary
C     network flow problem.
C
C     The routine implements the relaxation method of
C
C     Bertsekas, D. P., "A Unified Framework for Primal-Dual Methods ..."
C              Math. Programming, Vol. 32, 1985, pp. 125-145
C     Bertsekas, D. P., & Tseng, P., "Relaxation Methods for Minimum Cost
..."
C              Operations Research J., Vol. 36, 1988, pp. 93-114
C
C     The relaxation method is also described in the books:
c
C     Bertsekas, D. P., "Linear Network Optimization: Algorithms and
Codes"
C              M.I.T. Press, 1991
C     Bertsekas, D. P., & Tsitsiklis, J. N., "Parallel and Distributed
C              Computation: Numerical Methods" Prentice-Hall, 1989
C
C     The routine was written by Dimitri Bertsekas and Paul Tseng with
C     contributions by Jonathan Eckstein in the default initialization
C     routine.
C
C     One difference between this routine and the similar code RELAX is
C     that it maintains a data structure that gives all the balanced arcs
C     in the network.  This structure is called the "tree" for historical
C     reasons, even though it describes a subnetwork that will generally
C     be neither acyclic nor connected.  Also, the tree may contain some
C     arcs that are not balanced:  it turns out to be cheaper to purge
C     arcs that have become unbalanced only when their end nodes are being
C     scanned, as opposed to always maintaining an exact set of balanced
arcs.
C
C   ***************************************************************
C
C      All data should be in INTEGER*4. To run in limited memory systems
C      the arrays STARTN, ENDN, NXTIN, NXTOU, SAVE, FIN, FOU, LABEL,
C      PRDCSR,NXTQUEUE may be declared as INTEGER*2.
C
C          N (the number of nodes)
C          NA (the number of arcs)
C          LARGE (a very large positive integer to represent infinity.
C           All problem data should be less than LARGE in magnitude,
C           and LARGE should be less than, say, 1/4 the largest INTEGER*4
C           of the machine used. This will guard primarily against
```

```
C           overflow in uncapacitated problems where the arc capacities
C           are taken finite but very large.)
C          STARTN(NA) (the head node array)
C          ENDN(NA) (the tail node array)
C          RC(NA) (the reduced cost array)
C          X(NA) (the arc flow array)
C          U(NA) (the arc flow capacity array)
C          DFCT(N) (the deficit array)
C          FOU(N) (the first arc out array)
C          FIN(N) (the first arc in array)
C          NXTOU(NA) (the next arc out array)
C          NXTIN(NA) (the next arc in array)
C          NXTQUEUE(N) (the next node in the queue to be examined)
C
C     This subroutine places the optimal flow in the array X
C     and the corresponding reduced cost vector in the array RC.
C
C       If favorable arrays X, U, and RC are known they can
C       be passed to the routine directly. This requires that the
C       initialization portion of the routine (up to line 70) be skipped.
C
C     ****************************************************************

      SUBROUTINE RELAXT_QC
      IMPLICIT INTEGER (A-Z)
      LOGICAL*1 FEASBL,QUIT,SCAN,SWITCH,MARK,POSIT,PCHANGE
      REAL*8 TT,TIMER,TIMER1,CAPFACTOR
      COMMON/ARRAYS/STARTN/ARRAYE/ENDN/ARRAYU/U/ARRAYX/X/ARRAY9/RC
     $/ARRAYB/DFCT/BLK1/LABEL/BLK2/PRDCSR/BLK3/FOU/BLK4/NXTOU/BLK5/FIN
     $/BLK6/NXTIN/BLK7/SAVE/BLK8/SCAN/BLK9/MARK
     $/L/N,NA,LARGE,NMULTINODE,ITER,NUM_AUGM,NUM_ASCNT,NUM_SP,CRASH
      COMMON /BLK10/TFSTOU/BLK11/TNXTOU/BLK12/TFSTIN/BLK13/TNXTIN
      COMMON /BLK14/NXTQUEUE

C     Each common block contains just one array, so the arrays in
RELAXT_QC
C     can be dimensioned to 1 element and take their dimension from the
C     main calling routine. With this trick RELAXT_QC need not be
recompiled
C     if the problem dimension changes. If your FORTRAN does not support
C     this feature change the dimensions below to be the same as the ones
C     declared in your main calling program.

      DIMENSION TFSTOU(1),TNXTOU(1),TFSTIN(1),TNXTIN(1)
      DIMENSION STARTN(1),ENDN(1),U(1),X(1),RC(1),DFCT(1)
      DIMENSION LABEL(1),PRDCSR(1),SCAN(1),FOU(1),NXTOU(1)
      DIMENSION FIN(1),NXTIN(1),SAVE(1),MARK(1)
      DIMENSION NXTQUEUE(1)

C     DDPOS and DDNEG are arrays that give the directional derivatives for
C     all positive and negative single-node price changes.  These are used
C     only in the initial phase of the algorithm, before the "tree" data
C     structure comes into play.  Therefore, they are equivalenced to
C     TFSTOU and TFSTIN, which are the same size (number of nodes) and are
```

```
C     only used after the tree comes into use.

      DIMENSION   DDPOS(1),DDNEG(1)
      EQUIVALENCE (DDPOS(1),TFSTOU(1)),(DDNEG(1),TFSTIN(1))

C    Reduce arc capacity as much as possible w/out changing the problem
C    If this is a sensitivity run via routine SENSTV skip the
C    initialization by setting REPEAT to .TRUE. in the calling program.

C     The initialization (from here up to line 70)
C      can be skipped if the calling program
C      places in common user-chosen values for the arc flows, residual arc
C      capacities, nodal deficits and reduced costs. It is mandatory that
arc flows
C      and residual costs satisfy complementary slackess, on each arc,
C      and that the DFCT  array properly correspond to the initial arc
flows


       NDEF1=0
       DO 40 NODE=1,N
        NODE_DEF=DFCT(NODE)
        IF (NODE_DEF.LT.0) THEN
          NDEF1=NDEF1+1
          SAVE(NDEF1)=NODE
        END IF

C     Note that we also set up the initial DDPOS and DDNEG for each node.
C     (this is not necessary in RELAX)

         DDPOS(NODE)=NODE_DEF
         DDNEG(NODE)=-NODE_DEF

         MAXCAP=0
         SCAPOU=0
         ARC=FOU(NODE)
41       IF (ARC.GT.0) THEN
           IF (SCAPOU.LE.LARGE-U(ARC)) THEN
             SCAPOU=SCAPOU+U(ARC)
           ELSE
             GO TO 40
           END IF
           ARC=NXTOU(ARC)
           GO TO 41
         END IF
         IF (SCAPOU.LE.LARGE-NODE_DEF) THEN
           CAPOUT=SCAPOU+NODE_DEF
         ELSE
           GO TO 40
         END IF
         IF (CAPOUT.LT.0) THEN

C  ** PROBLEM IS INFEASIBLE - EXIT
```

```
        PRINT*,'EXIT DURING INITIALIZATION'
        PRINT*,'EXOGENOUS FLOW INTO NODE',NODE,'  EXCEEDS OUT CAPACITY'
        CALL PRINTFLOWS(NODE)
        GO TO 400
         END IF

         SCAPIN=0
         ARC=FIN(NODE)
43       IF (ARC.GT.0) THEN
           U(ARC)=MIN0(U(ARC),CAPOUT)
           IF (MAXCAP.LT.U(ARC)) MAXCAP=U(ARC)
           IF (SCAPIN.LE.LARGE-U(ARC)) THEN
             SCAPIN=SCAPIN+U(ARC)
           ELSE
             GO TO 40
           END IF
           ARC=NXTIN(ARC)
           GO TO 43
         END IF
         IF (SCAPIN.LE.LARGE+NODE_DEF) THEN
           CAPIN=SCAPIN-NODE_DEF
         ELSE
           GO TO 40
         END IF
         IF (CAPIN.LT.0) THEN

C  *** PROBLEM IS INFEASIBLE - EXIT

        PRINT*,'EXIT DURING INITIALIZATION'
        PRINT*,'EXOGENOUS FLOW OUT OF NODE',NODE,' EXCEEDS IN CAPACITY'
        CALL PRINTFLOWS(NODE)
        GO TO 400
        END IF

         ARC=FOU(NODE)
45       IF (ARC.GT.0) THEN
           U(ARC)=MIN0(U(ARC),CAPIN)
           ARC=NXTOU(ARC)
           GO TO 45
         END IF
40       CONTINUE




C     ******* initialize the arc flows and the nodal deficits ********
C     *** note that U(ARC) is redefined as the residual capacity of ARC
***

C     Now compute the directional derivatives for each coordinate exactly.
C     As well as computing the actual deficits.  U(ARC) is the residual
C     capacity on ARC, and X(ARC) is the flow.  These always add up to the
C     total capacity.
```

```
      DO 48 ARC=1,NA
         X(ARC) = 0
         IF (RC(ARC).LE. 0) THEN
            T  = U(ARC)
            T1 = STARTN(ARC)
            T2 = ENDN(ARC)
            DDPOS(T1) = DDPOS(T1) + T
            DDNEG(T2) = DDNEG(T2) + T
            IF (RC(ARC).LT. 0) THEN
               X(ARC) = T
               U(ARC) = 0
               DFCT(T1) = DFCT(T1) + T
               DFCT(T2) = DFCT(T2) - T
               DDNEG(T1) = DDNEG(T1) - T
               DDPOS(T2) = DDPOS(T2) - T
            END IF
         END IF
48    CONTINUE

C     DO INITIALIZATION WITH SINGLE NODE RELAXATION ITERATIONS
C     An adaptive strategy is used:  the number of single-node iteration
C     cycles attempted is a function of the average density of the
network.

      IF (NA.GT.N*10) THEN
        NUMPASSES=2
      ELSE
        NUMPASSES=3
      END IF

C     We now do 2 or 3 passes through all the nodes.  This is the initial
C     phase:  if a single node iteration is not possible, we just go on to
C     the next node.

      DO 390 PASSES = 1,NUMPASSES

      DO 300 NODE=1,N

       IF (DFCT(NODE).NE. 0) THEN

C       Price rise or price drop?  (Note: it is impossible to have both.)

        IF (DDPOS(NODE).LE. 0) THEN

C     Price rise.  Loop over breakpoints in +Price(NODE) direction.
C     On outgoing arcs, tension will rise and reduced cost will fall
C        -- so, next break comes at smallest positive reduced cost.
C     On incoming arcs, tension will fall and reduced cost will rise
C        -- so, next break comes at smallest negative reduced cost.

          DELPRC = LARGE
```

```
            ARC = FOU(NODE)
303         IF (ARC.GT.0) THEN
               TRC = RC(ARC)
               IF ((TRC.GT. 0).AND.(TRC.LT.DELPRC)) THEN
                  DELPRC = TRC
               END IF
               ARC = NXTOU(ARC)
               GOTO 303
            END IF

            ARC = FIN(NODE)
304         IF (ARC.GT.0) THEN
               TRC = RC(ARC)
               IF ((TRC.LT.0).AND.(TRC.GT.-DELPRC)) THEN
                  DELPRC = -TRC
               END IF
               ARC = NXTIN(ARC)
               GOTO 304
            END IF

C           If no breakpoints left and ascent still possible, the problem
C           is infeasible.

            IF (DELPRC.GE.LARGE) THEN
               IF (DDPOS(NODE).EQ.0) GOTO 300
               GOTO 400
            ENDIF

C           We have an actual breakpoint.  Increase price by that quantity.
C           First check the effect on all outbound arcs, which will have a
C           tension increase and reduced cost drop.

350         NXTBRK = LARGE
            ARC = FOU(NODE)
305         IF (ARC.GT.0) THEN
               TRC = RC(ARC)
               IF (TRC .EQ. 0) THEN
                  T1 = ENDN(ARC)
                  T  = U(ARC)
                  IF (T.GT.0) THEN
                     DFCT(NODE) = DFCT(NODE) + T
                     DFCT(T1) = DFCT(T1) - T
                     X(ARC) = T
                     U(ARC) = 0
                  ELSE
                     T = X(ARC)
                  END IF
                  DDNEG(NODE) = DDNEG(NODE) - T
                  DDPOS(T1) = DDPOS(T1) - T
               END IF

C              For all outgoing arcs tension rises, and reduced cost drops.

               TRC = TRC - DELPRC
```

```
               IF ((TRC.GT.0).AND.(TRC.LT.NXTBRK)) THEN
                  NXTBRK = TRC
               ELSE IF (TRC.EQ.0) THEN

C                 Arc goes from inactive to balanced.  Just change tension
C                 increase derivatives, and check for status change at
other end.

                  DDPOS(NODE) = DDPOS(NODE) + U(ARC)
                  DDNEG(ENDN(ARC)) = DDNEG(ENDN(ARC)) + U(ARC)
               END IF
               RC(ARC) = TRC
               ARC =  NXTOU(ARC)
               GOTO 305
            END IF

C           Time to check the incoming arcs into the node.
C           These arcs will have an tension decrease and a reduced cost
rise.

            ARC = FIN(NODE)
306         IF (ARC.GT.0) THEN
               TRC = RC(ARC)
               IF (TRC.EQ.0) THEN
                  T1 = STARTN(ARC)
                  T  = X(ARC)
                  IF (T.GT.0) THEN
                     DFCT(NODE) = DFCT(NODE) + T
                     DFCT(T1) = DFCT(T1) - T
                     U(ARC) = T
                     X(ARC) = 0
                  ELSE
                     T = U(ARC)
                  END IF
                  DDPOS(T1) = DDPOS(T1) - T
                  DDNEG(NODE) = DDNEG(NODE) - T
               END IF

C              Note the reduced cost rise for every arc.

               TRC = TRC + DELPRC
               IF ((TRC.LT.0).AND.(TRC.GT.-NXTBRK)) THEN
                  NXTBRK = -TRC
               ELSE IF (TRC.EQ.0) THEN

C                 Now check for movement from active to balanced.
C                 If so, tension decrease derivatives increase.

                  DDNEG(STARTN(ARC)) = DDNEG(STARTN(ARC)) + X(ARC)
                  DDPOS(NODE) = DDPOS(NODE) + X(ARC)
               END IF
               RC(ARC) = TRC
               ARC = NXTIN(ARC)
               GOTO 306
```

```
            END IF

C           We are now done with the iteration.  If the current direction
C           is still a (degenerate) ascent direction, push onward.

            IF ((DDPOS(NODE).LE.0).AND.(NXTBRK.LT.LARGE)) THEN
               DELPRC = NXTBRK
               GOTO 350
            END IF


C     Now comes the code for a price decrease at NODE.
C     On outgoing arcs, tension will drop and reduced cost will increase
C        -- so, next break comes at smallest negative reduced cost.
C     On incoming arcs, tension will increase and reduced cost will fall
C        -- so, next break comes at smallest positive reduced cost.

      ELSE IF (DDNEG(NODE).LE.0) THEN

         DELPRC = LARGE

         ARC = FOU(NODE)
307      IF (ARC.GT.0) THEN
            TRC = RC(ARC)
            IF ((TRC.LT.0).AND.(TRC.GT.-DELPRC)) THEN
               DELPRC = -TRC
            ENDIF
            ARC = NXTOU(ARC)
            GOTO 307
         ENDIF

         ARC = FIN(NODE)
308      IF (ARC.GT.0) THEN
            TRC = RC(ARC)
            IF ((TRC.GT.0).AND.(TRC.LT.DELPRC)) THEN
               DELPRC = TRC
            END IF
            ARC = NXTIN(ARC)
            GOTO 308
         END IF

C        If there is no breakpoint, the problem is infeasible,
C        unless we are making a degenerate step.

         IF (DELPRC.EQ.LARGE) THEN
            IF (DDNEG(NODE).EQ.0) GOTO 300
            GOTO 400
         END IF

C        Now we make the step to the next breakpoint.  We start with the
C        outbound arcs.  These have a tension decrease and reduced cost
C        rise.  Therefore, the possible transitions are from balanced to
C        inactive or active to balanced.
```

```
360      NXTBRK = LARGE
         ARC = FOU(NODE)
309      IF (ARC.GT.0) THEN
            TRC = RC(ARC)
            IF (TRC.EQ.0) THEN
               T1 = ENDN(ARC)
               T  = X(ARC)
               IF (T.GT.0) THEN
                  DFCT(NODE) = DFCT(NODE) - T
                  DFCT(T1) = DFCT(T1) + T
                  U(ARC) = T
                  X(ARC) = 0
               ELSE
                  T = U(ARC)
               END IF
               DDPOS(NODE) = DDPOS(NODE) - T
               DDNEG(T1) = DDNEG(T1) - T
            END IF

C           Log the reduced cost rise for all arcs.

            TRC = TRC + DELPRC
            IF ((TRC.LT.0).AND.(TRC.GT.-NXTBRK)) THEN
               NXTBRK = -TRC
            ELSE IF (TRC.EQ.0) THEN

C              Active to balanced.  Tension decrease derivs go up.

               DDNEG(NODE) = DDNEG(NODE) + X(ARC)
               DDPOS(ENDN(ARC)) = DDPOS(ENDN(ARC)) + X(ARC)
            END IF
            RC(ARC) = TRC
            ARC = NXTOU(ARC)
            GOTO 309
         END IF

C        Now do the incoming arcs.  These have a tension increase and
C        therefore a reduced cost drop.  The possible transitions are
C        from inactive to balanced and from balanced to active.

         ARC = FIN(NODE)
310      IF (ARC.GT.0) THEN
            TRC = RC(ARC)
            IF (TRC.EQ.0) THEN
               T1 = STARTN(ARC)
               T  = U(ARC)
               IF (T.GT.0) THEN
                  DFCT(NODE) = DFCT(NODE) - T
                  DFCT(T1) = DFCT(T1) + T
                  X(ARC) = T
                  U(ARC) = 0
               ELSE
                  T = X(ARC)
               END IF
```

```
               DDNEG(T1) = DDNEG(T1) - T
               DDPOS(NODE) = DDPOS(NODE) - T
            END IF
            TRC = TRC - DELPRC
            IF ((TRC.GT.0).AND.(TRC.LT.NXTBRK)) THEN
               NXTBRK = TRC
            ELSE IF (TRC.EQ.0) THEN
               DDPOS(STARTN(ARC)) = DDPOS(STARTN(ARC)) + U(ARC)
               DDNEG(NODE) = DDNEG(NODE) + U(ARC)
            END IF
            RC(ARC) = TRC
            ARC = NXTIN(ARC)
            GOTO 310
         END IF

C        OK.  Movement is done.  Is this direction still a (degenerate)
C        ascent direction.  If so, keep going.

         IF ((DDNEG(NODE).LE.0).AND.(NXTBRK.LT.LARGE)) THEN
            DELPRC = NXTBRK
            GOTO 360
         END IF

       END IF

      END IF

300   CONTINUE

390   CONTINUE

C     STORE THE NODES THAT NOW HAVE NEGATIVE DEFICIT

      NDEF=NDEF1

      DO 372 I=1,N
        IF (DFCT(I).LT.0) THEN
          NDEF=NDEF+1
          SAVE(NDEF)=I
        END IF
372   CONTINUE

C     RETURN IF FINISHED

      IF (NDEF.EQ.NDEF1) THEN
         RETURN
      END IF

C     THIS VERSION OF RELAXT USES SELECTIVELY SHORTEST PATH ITERATIONS
C     FOR INITIALIZATION. ARCS WITH SMALL CAPACITY ARE NOT TAKEN
C     INTO ACCOUNT IN THE SHORTEST PATH COMPUTATIONS, AS SPECIFIED
C     BY THE PARAMETERS CAPFACTOR AND CAPTHRESH WHICH ARE SET BELOW.

C     IF THE PARAMETER CRASH IS SET TO 1, THE
```

```
C     SHORTEST PATH INITIALIZATION PROCEDURE IS MORE EXTENSIVE
C     CRASH=1 IS RECOMMENDED FOR DIFFICULT PROBLEMS


      THRESH=15
      IF (NDEF-NDEF1.LE.THRESH) THEN
        LIMIT1=NDEF1+1
        LIMIT2=NDEF
      ELSE
        LIMIT1=1
        LIMIT2=NDEF1
      END IF

       IF (CRASH.EQ.1) THEN
          NUMPASSES=INT(N/(2*(NDEF-NDEF1)))
          LIMIT1=NDEF1+1
          LIMIT2=NDEF
          IF (NUMPASSES.GT.3) NUMPASSES=3
       ELSE
          THRESH1=5
          IF ((NDEF-NDEF1.GT.THRESH).AND.(NDEF1.GT.THRESH1)) GO TO 70
          NUMPASSES=1
       END IF

        NUM_SP=0
        CAPFACTOR=0.02

        DO 375 J=1,NUMPASSES
         IF (CRASH.EQ.1) THEN
           CAPTHRESH=INT(CAPFACTOR*MAXCAP*(J-1))
         ELSE
           CAPTHRESH=INT(CAPFACTOR*MAXCAP)
         END IF
         DO 374 I=LIMIT1,LIMIT2
           NODE=SAVE(I)
           IF (DFCT(NODE).LT.0) THEN
             CALL SHORT_PATH(NODE,CAPTHRESH)
             NUM_SP=NUM_SP+1
           END IF
374      CONTINUE
375     CONTINUE

C     RECTIFY THE FLOWS AND DEFICITS TO ACCOUNT FOR
C     SMALL CAPACITY ARCS IGNORED IN THE SHORTEST PATH COMPUTATIONS

      DO 376 ARC=1,NA
         IF ((RC(ARC).LT.0).AND.(U(ARC).GT.0)) THEN
            T  = U(ARC)
            T1 = STARTN(ARC)
            T2 = ENDN(ARC)
            X(ARC) = X(ARC)+T
            U(ARC) = 0
            DFCT(T1) = DFCT(T1) + T
            DFCT(T2) = DFCT(T2) - T
```

```
         END IF
         IF ((RC(ARC).GT.0).AND.(X(ARC).GT.0)) THEN
            T  = X(ARC)
            T1 = STARTN(ARC)
            T2 = ENDN(ARC)
            U(ARC) = U(ARC)+T
            X(ARC) = 0
            DFCT(T1) = DFCT(T1) - T
            DFCT(T2) = DFCT(T2) + T
         END IF
376    CONTINUE

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

70    CONTINUE

C     ******* initialize the tree *******

      DO 72 I=1,N
        TFSTOU(I)=0
        TFSTIN(I)=0
72    CONTINUE

      DO 74 I=1,NA
        TNXTIN(I)=-1
        TNXTOU(I)=-1
        IF (RC(I).EQ.0) THEN
          TNXTOU(I)=TFSTOU(STARTN(I))
          TFSTOU(STARTN(I))=I
          TNXTIN(I)=TFSTIN(ENDN(I))
          TFSTIN(ENDN(I))=I
        END IF
74    CONTINUE


C     *********** Initialize other variables ***********

      FEASBL=.TRUE.
      ITER=0
      NMULTINODE=0
      NDFCT=N
      NUM_AUGM=0
      NUM_ASCNT=0
      NUMNONZERO=0
      SWITCH=.FALSE.
      DO 76 I=1,N
        MARK(I)=.FALSE.
        SCAN(I)=.FALSE.
76    CONTINUE
      NLABEL=0


C     RELAXT uses an adaptive strategy for deciding whether to
C     continue the scanning process after a price change.
```

```
C     The threshold parameters TP and TS that control
C     this strategy are set in the next few lines.

      TP=10
      TS=INT(N/15)


C     THIS VERSION OF RELAXT ALSO USES A QUEUE TO MAINTAIN THE SET OF
C     NODES WITH NONZERO DEFICIT OR SURPLUS. NODES ARE PICKED UP FROM
C     THE QUEUE FOR RELAXATION.

C     QUEUE INITIALIZATION

      DO 82 NODE=1,N-1
        NXTQUEUE(NODE)=NODE+1
82    CONTINUE
      NXTQUEUE(N)=1
      NODE=N
      PREVNODE=N-1
      LASTQUEUE=N
      NDFCT=N
      NUMNONZERO=O

C     ************ START RELAXATION ALGORITHM **************

100   CONTINUE

C     CODE FOR ADVANCING THE QUEUE

          PREVNODE=NODE
          NODE=NXTQUEUE(NODE)
          DEFCIT=DFCT(NODE)
          IF (NODE.EQ.LASTQUEUE) THEN
            NDFCT=NUMNONZERO
            NUMNONZERO=0
            LASTQUEUE=PREVNODE
          END IF

C     COCE FOR DELETING A NODE FROM THE QUEUE

         IF (DEFCIT.EQ.0) THEN
            NXTNODE=NXTQUEUE(NODE)
            IF (NODE.EQ.NXTNODE) THEN
              RETURN
            ELSE
              NXTQUEUE(PREVNODE)=NXTNODE
              NXTQUEUE(NODE)=0
              NODE=NXTNODE
              GO TO 100
            END IF
          ELSE
            POSIT = (DEFCIT.GT.0)
            NUMNONZERO=NUMNONZERO+1
          END IF
```

```
C         ***** ATTEMPT A SINGLE NODE ITERATION FROM NODE ****

      ITER=ITER+1
      IF (POSIT) THEN

C    ************* CASE OF NODE W/ POSITIVE DEFICIT ********

      PCHANGE = .FALSE.
      INDEF=DEFCIT
      DELX=0
      NB=0

C     Check outgoing (probably) balanced arcs from NODE.

      ARC=TFSTOU(NODE)
500   IF (ARC.GT.0) THEN
         IF ((RC(ARC).EQ.0).AND.(X(ARC).GT.0)) THEN
            DELX = DELX + X(ARC)
            NB = NB + 1
            SAVE(NB) = ARC
         ENDIF
         ARC = TNXTOU(ARC)
         GOTO 500
      END IF

C     Check incoming arcs.

      ARC = TFSTIN(NODE)
501   IF (ARC.GT.0) THEN
         IF ((RC(ARC).EQ.0).AND.(U(ARC).GT.0)) THEN
            DELX = DELX + U(ARC)
            NB = NB + 1
            SAVE(NB) = -ARC
         ENDIF
         ARC = TNXTIN(ARC)
         GOTO 501
      END IF

C   ***** END OF INITIAL NODE SCAN *******

18     CONTINUE

C    ********* IF NO PRICE CHANGE IS POSSIBLE, EXIT **********

      IF (DELX.GT.DEFCIT) THEN
        QUIT = (DEFCIT .LT. INDEF)
        GO TO 16
      END IF

C     RELAXT SEARCHES ALONG THE ASCENT DIRECTION FOR THE
C     BEST PRICE BY CHECKING THE SLOPE OF THE DUAL COST
C     AT SUCCESSIVE BREAK POINTS.
```

```
C     WE NOW COMPUTE THE DISTANCE TO THE NEXT BREAK POINT

      DELPRC = LARGE
      ARC = FOU(NODE)
502   IF (ARC .GT. 0) THEN
         RDCOST = RC(ARC)
         IF ((RDCOST.LT.0).AND.(RDCOST.GT.-DELPRC)) THEN
            DELPRC = -RDCOST
         ENDIF
         ARC = NXTOU(ARC)
         GOTO 502
      END IF
      ARC = FIN(NODE)
503   IF (ARC .GT. 0) THEN
         RDCOST = RC(ARC)
         IF ((RDCOST.GT.0).AND.(RDCOST.LT.DELPRC)) THEN
            DELPRC = RDCOST
         ENDIF
         ARC = NXTIN(ARC)
         GOTO 503
      END IF

C     ******* check if the problem is infeasible *******

      IF ((DELX.LT.DEFCIT).AND.(DELPRC.EQ.LARGE)) THEN

C       ***** The dual cost can be decreased without bound  *****

        GO TO 400
      END IF

C    **** SKIP FLOW ADJUSTEMT IF THERE IS NO FLOW TO MODIFY ***

      IF (DELX.EQ.0) GO TO 14

C       Adjust the flow on balanced arcs incident of NODE to
C       maintain complementary slackness after the price change

      DO 13 J=1,NB
        ARC=SAVE(J)
        IF (ARC.GT.0) THEN
          NODE2=ENDN(ARC)
          T1=X(ARC)
          DFCT(NODE2)=DFCT(NODE2)+T1
          IF (NXTQUEUE(NODE2).EQ.0) THEN
            NXTQUEUE(PREVNODE)=NODE2
            NXTQUEUE(NODE2)=NODE
            PREVNODE=NODE2
          END IF
          U(ARC)=U(ARC)+T1
          X(ARC)=0
        ELSE
          NARC=-ARC
          NODE2=STARTN(NARC)
```

```
            T1=U(NARC)
            DFCT(NODE2)=DFCT(NODE2)+T1
            IF (NXTQUEUE(NODE2).EQ.0) THEN
              NXTQUEUE(PREVNODE)=NODE2
              NXTQUEUE(NODE2)=NODE
              PREVNODE=NODE2
            END IF
            X(NARC)=X(NARC)+T1
            U(NARC)=0
          END IF
13       CONTINUE
        DEFCIT=DEFCIT-DELX
14      IF (DELPRC.EQ.LARGE) THEN
          QUIT=.TRUE.
          GO TO 19
        END IF

C       NODE corresponds to a dual ascent direction.  Decrease
C       the price of NODE by DELPRC and compute the stepsize to the
C       next breakpoint in the dual cost

        NB=0
        PCHANGE = .TRUE.
        DP=DELPRC
        DELPRC=LARGE
        DELX=0
        ARC=FOU(NODE)
504     IF (ARC.GT.0) THEN
          RDCOST=RC(ARC)+DP
          RC(ARC)=RDCOST
          IF (RDCOST.EQ.0) THEN
            NB=NB+1
            SAVE(NB)=ARC
            DELX=DELX+X(ARC)
          END IF
          IF ((RDCOST.LT.0).AND.(RDCOST.GT.-DELPRC)) DELPRC=-RDCOST
          ARC=NXTOU(ARC)
          GOTO 504
        END IF
        ARC=FIN(NODE)
505     IF (ARC.GT.0) THEN
          RDCOST=RC(ARC)-DP
          RC(ARC)=RDCOST
          IF (RDCOST.EQ.0) THEN
            NB=NB+1
            SAVE(NB)=-ARC
            DELX=DELX+U(ARC)
          END IF
          IF ((RDCOST.GT.0).AND.(RDCOST.LT.DELPRC)) DELPRC=RDCOST
          ARC=NXTIN(ARC)
          GOTO 505
        END IF

C   ***** return to check if another price change is possible ******
```

```
      GO TO 18

C     ******* perform flow augmentation at NODE ****

16    DO 11 J=1,NB
        ARC=SAVE(J)
        IF (ARC.GT.0) THEN

C         *** ARC is an outgoing arc from NODE ********************

          NODE2=ENDN(ARC)
          T1=DFCT(NODE2)
          IF (T1.LT.0) THEN

C         ** Decrease the total deficit by decreasing flow of ARC **

            QUIT=.TRUE.
            T2=X(ARC)
            DX=MIN0(DEFCIT,-T1,T2)
            DEFCIT=DEFCIT-DX
            DFCT(NODE2)=T1+DX
          IF (NXTQUEUE(NODE2).EQ.0) THEN
            NXTQUEUE(PREVNODE)=NODE2
            NXTQUEUE(NODE2)=NODE
            PREVNODE=NODE2
          END IF

            X(ARC)=T2-DX
            U(ARC)=U(ARC)+DX
            IF (DEFCIT.EQ.0) GO TO 19
          END IF
        ELSE

C         *** -ARC is an incoming arc to NODE ***

          NARC=-ARC
          NODE2=STARTN(NARC)
          T1=DFCT(NODE2)
          IF (T1.LT.0) THEN

C         ** Decrease the total deficit by increasing flow of -ARC **

            QUIT=.TRUE.
            T2=U(NARC)
            DX=MIN0(DEFCIT,-T1,T2)
            DEFCIT=DEFCIT-DX
            DFCT(NODE2)=T1+DX
          IF (NXTQUEUE(NODE2).EQ.0) THEN
            NXTQUEUE(PREVNODE)=NODE2
            NXTQUEUE(NODE2)=NODE
            PREVNODE=NODE2
          END IF
            X(NARC)=X(NARC)+DX
```

```
            U(NARC)=T2-DX
            IF (DEFCIT.EQ.0) GO TO 19
          END IF
         END IF
11      CONTINUE
19      DFCT(NODE)=DEFCIT

C       Reconstruct the list of balanced arcs adjacent to this node.
C       For the neighbor nodes, we
C       add all the newly balanced arcs, but do not bother getting rid
C       of formerly balanced ones (they will be purged the next time the
C       adjacent node is scanned).

        IF (PCHANGE) THEN
           ARC = TFSTOU(NODE)
           TFSTOU(NODE) = 0
506        IF (ARC .GT. 0) THEN
              NXTARC = TNXTOU(ARC)
              TNXTOU(ARC) = -1
              ARC = NXTARC
              GOTO 506
           END IF
           ARC = TFSTIN(NODE)
           TFSTIN(NODE) = 0
507        IF (ARC .GT. 0) THEN
              NXTARC = TNXTIN(ARC)
              TNXTIN(ARC) = -1
              ARC = NXTARC
              GOTO 507
           END IF

C      Now add the currently balanced arcs to the list for this node
C      (which is now empty), and the appropriate adjacent ones.

           DO 508 J=1,NB
              ARC = SAVE(J)
              IF (ARC.LE.0) ARC=-ARC
              IF (TNXTOU(ARC) .LT. 0) THEN
                 TNXTOU(ARC) = TFSTOU(STARTN(ARC))
                 TFSTOU(STARTN(ARC)) = ARC
              END IF
              IF (TNXTIN(ARC) .LT. 0) THEN
                 TNXTIN(ARC) = TFSTIN(ENDN(ARC))
                 TFSTIN(ENDN(ARC)) = ARC
              END IF
508        CONTINUE

        END IF

C    *** end of single node iteration for a positive deficit node ***

       ELSE

C      ******* single node iteration for a negative deficit node ******
```

```
      PCHANGE = .FALSE.
      DEFCIT=-DEFCIT
      INDEF=DEFCIT
      DELX=0
      NB=0

      ARC = TFSTIN(NODE)
509   IF (ARC .GT. 0) THEN
         IF ((RC(ARC) .EQ. 0) .AND. (X(ARC) .GT. 0)) THEN
            DELX = DELX + X(ARC)
            NB = NB + 1
            SAVE(NB) = ARC
         ENDIF
         ARC = TNXTIN(ARC)
         GOTO 509
      END IF
      ARC=TFSTOU(NODE)
510   IF (ARC .GT. 0) THEN
         IF ((RC(ARC) .EQ. 0) .AND. (U(ARC) .GT. 0)) THEN
            DELX = DELX + U(ARC)
            NB = NB + 1
            SAVE(NB) = -ARC
         ENDIF
         ARC = TNXTOU(ARC)
         GOTO 510
      END IF

28     CONTINUE
      IF (DELX.GE.DEFCIT) THEN
        QUIT = (DEFCIT .LT. INDEF)
        GO TO 26
      END IF

C     Compute distance to next breakpoint.

      DELPRC = LARGE
      ARC = FIN(NODE)
511   IF (ARC .GT. 0) THEN
         RDCOST = RC(ARC)
         IF ((RDCOST .LT. 0) .AND. (RDCOST.GT.-DELPRC)) THEN
            DELPRC = -RDCOST
         ENDIF
         ARC = NXTIN(ARC)
         GOTO 511
      END IF
      ARC = FOU(NODE)
512   IF (ARC .GT. 0) THEN
         RDCOST = RC(ARC)
         IF ((RDCOST.GT.0).AND.(RDCOST.LT.DELPRC)) THEN
            DELPRC = RDCOST
         ENDIF
         ARC = NXTOU(ARC)
         GOTO 512
```

```
      END IF

C     ******* check if problem is infeasible ************************

      IF ((DELX.LT.DEFCIT).AND.(DELPRC.EQ.LARGE)) THEN
        GO TO 400
      END IF
      IF (DELX.EQ.0) GO TO 24

C     ******* flow augmentation is possible *************************

      DO 23 J=1,NB
        ARC=SAVE(J)
        IF (ARC.GT.0) THEN
          NODE2=STARTN(ARC)
          T1=X(ARC)
          DFCT(NODE2)=DFCT(NODE2)-T1
          IF (NXTQUEUE(NODE2).EQ.0) THEN
            NXTQUEUE(PREVNODE)=NODE2
            NXTQUEUE(NODE2)=NODE
            PREVNODE=NODE2
          END IF
          U(ARC)=U(ARC)+T1
          X(ARC)=0
        ELSE
          NARC=-ARC
          NODE2=ENDN(NARC)
          T1=U(NARC)
          DFCT(NODE2)=DFCT(NODE2)-T1
          IF (NXTQUEUE(NODE2).EQ.0) THEN
            NXTQUEUE(PREVNODE)=NODE2
            NXTQUEUE(NODE2)=NODE
            PREVNODE=NODE2
          END IF
          X(NARC)=X(NARC)+T1
          U(NARC)=0
        END IF
23     CONTINUE
      DEFCIT=DEFCIT-DELX
24    IF (DELPRC.EQ.LARGE) THEN
        QUIT=.TRUE.
        GO TO 29
      END IF

C     ******* price increase at NODE is possible ********

      NB=0
      PCHANGE = .TRUE.
      DP=DELPRC
      DELPRC=LARGE
      DELX=0
      ARC=FIN(NODE)
513   IF (ARC.GT.0) THEN
        RDCOST=RC(ARC)+DP
```

```
        RC(ARC)=RDCOST
        IF (RDCOST.EQ.0) THEN
          NB=NB+1
          SAVE(NB)=ARC
          DELX=DELX+X(ARC)
        END IF
        IF ((RDCOST.LT.0).AND.(RDCOST.GT.-DELPRC)) DELPRC=-RDCOST
        ARC=NXTIN(ARC)
        GOTO 513
      END IF
      ARC=FOU(NODE)
514   IF (ARC.GT.0) THEN
        RDCOST=RC(ARC)-DP
        RC(ARC)=RDCOST
        IF (RDCOST.EQ.0) THEN
          NB=NB+1
          SAVE(NB)=-ARC
          DELX=DELX+U(ARC)
        END IF
        IF ((RDCOST.GT.0).AND.(RDCOST.LT.DELPRC)) DELPRC=RDCOST
        ARC=NXTOU(ARC)
        GOTO 514
      END IF
      GO TO 28

C     ******* perform flow augmentation at NODE ****

26    DO 21 J=1,NB
        ARC=SAVE(J)
        IF (ARC.GT.0) THEN

C         *** ARC is an incoming arc to NODE ********************

          NODE2=STARTN(ARC)
          T1=DFCT(NODE2)
          IF (T1.GT.0) THEN
            QUIT=.TRUE.
            T2=X(ARC)
            DX=MIN0(DEFCIT,T1,T2)
            DEFCIT=DEFCIT-DX
            DFCT(NODE2)=T1-DX
          IF (NXTQUEUE(NODE2).EQ.0) THEN
            NXTQUEUE(PREVNODE)=NODE2
            NXTQUEUE(NODE2)=NODE
            PREVNODE=NODE2
          END IF
            X(ARC)=T2-DX
            U(ARC)=U(ARC)+DX
            IF (DEFCIT.EQ.0) GO TO 29
          END IF
        ELSE

C         *** -ARC is an outgoing arc from NODE ******
```

```
            NARC=-ARC
            NODE2=ENDN(NARC)
            T1=DFCT(NODE2)
            IF (T1.GT.0) THEN
              QUIT=.TRUE.
              T2=U(NARC)
              DX=MIN0(DEFCIT,T1,T2)
              DEFCIT=DEFCIT-DX
              DFCT(NODE2)=T1-DX
            IF (NXTQUEUE(NODE2).EQ.0) THEN
              NXTQUEUE(PREVNODE)=NODE2
              NXTQUEUE(NODE2)=NODE
              PREVNODE=NODE2
            END IF
              X(NARC)=X(NARC)+DX
              U(NARC)=T2-DX
              IF (DEFCIT.EQ.0) GO TO 29
            END IF
          END IF
21      CONTINUE
29      DFCT(NODE)=-DEFCIT

C       Reconstruct the list of balanced arcs adjacent to this node.

        IF (PCHANGE) THEN

            ARC = TFSTOU(NODE)
            TFSTOU(NODE) = 0
515         IF (ARC .GT. 0) THEN
               NXTARC = TNXTOU(ARC)
               TNXTOU(ARC) = -1
               ARC = NXTARC
               GOTO 515
            END IF
            ARC = TFSTIN(NODE)
            TFSTIN(NODE) = 0
516         IF (ARC .GT. 0) THEN
               NXTARC = TNXTIN(ARC)
               TNXTIN(ARC) = -1
               ARC = NXTARC
               GOTO 516
            END IF

C   *** Now add the currently balanced arcs to the list for this node ***
C   *** (which is now empty), and the appropriate adjacent ones.      ***

            DO 517 J=1,NB
               ARC = SAVE(J)
               IF (ARC.LE.0) ARC=-ARC
               IF (TNXTOU(ARC) .LT. 0) THEN
                  TNXTOU(ARC) = TFSTOU(STARTN(ARC))
                  TFSTOU(STARTN(ARC)) = ARC
               END IF
               IF (TNXTIN(ARC) .LT. 0) THEN
```

```
                TNXTIN(ARC) = TFSTIN(ENDN(ARC))
                TFSTIN(ENDN(ARC)) = ARC
              END IF
517         CONTINUE

        END IF

C     ***** end of single node iteration for a negative deficit node ***

        END IF

        IF (QUIT) GO TO 100

C       ******* DO A MULTINODE ITERATION FROM NODE *****************

        NMULTINODE=NMULTINODE+1
        SWITCH = (NDFCT.LT.TP)

C       ******* UNMARK NODES LABELED EARLIER *******

           DO 90 J=1,NLABEL
             NODE2=LABEL(J)
             MARK(NODE2)=.FALSE.
             SCAN(NODE2)=.FALSE.
90         CONTINUE

C       ******* INITIALIZE LABELING ******

           NLABEL=1
           LABEL(1)=NODE
           MARK(NODE)=.TRUE.
           PRDCSR(NODE)=0

C       ******** SCAN STARTING NODE *********

           SCAN(NODE)=.TRUE.
           NSCAN=1
           DM=DFCT(NODE)
           DELX=0
           DO 95 J=1,NB
            ARC=SAVE(J)
            IF (ARC.GT.0) THEN
             IF (POSIT) THEN
               NODE2=ENDN(ARC)
             ELSE
               NODE2=STARTN(ARC)
             END IF
             IF (.NOT.MARK(NODE2)) THEN
                NLABEL=NLABEL+1
                LABEL(NLABEL)=NODE2
                PRDCSR(NODE2)=ARC
                MARK(NODE2)=.TRUE.
                DELX=DELX+X(ARC)
             END IF
```

```
            ELSE
             NARC=-ARC
             IF (POSIT) THEN
               NODE2=STARTN(NARC)
             ELSE
               NODE2=ENDN(NARC)
             END IF
             IF (.NOT.MARK(NODE2)) THEN
               NLABEL=NLABEL+1
               LABEL(NLABEL)=NODE2
               PRDCSR(NODE2)=ARC
               MARK(NODE2)=.TRUE.
               DELX=DELX+U(NARC)
             END IF
            END IF
95        CONTINUE

C         **** START SCANNING LABELED NODES ****

120       NSCAN=NSCAN+1

C         ****** check to see if SWITCH needs to be set ******
C         SWITCH indicates it may now be best to continue scanning
C         until all labelled nodes are scanned

          SWITCH = SWITCH .OR.((NSCAN .GT. TS).AND.(NDFCT.LT.TS))

C    **** scan next node on the list of labeled nodes ****
C    *** scanning will continue until either an OVERESTIMATE of the
residual
C    capacity across the cut corresponding to the scanned set of nodes
(called
C    DELX) exceeds the absolute value of the total deficit of the scanned
C    nodes (called DM), or else an augmenting path is found.  Arcs that are
C    in the tree but are not balanced are purged as part of the scanning
C    process.

       I=LABEL(NSCAN)
       SCAN(I)=.TRUE.
       IF (POSIT) THEN

C      ******* scanning node I for case of positive deficit ******

       NAUGNOD=0
       PRVARC=0
       ARC = TFSTOU(I)

518    IF (ARC.GT.0) THEN

C        ***** ARC is an outgoing arc from NODE *****

         IF (RC(ARC) .EQ. 0) THEN
            IF (X(ARC) .GT. 0) THEN
               NODE2=ENDN(ARC)
```

```
               IF (.NOT. MARK(NODE2)) THEN

C       NODE2 is not in the labeled set.  Add NODE2 to the labeled set.

                  PRDCSR(NODE2)=ARC
                  IF (DFCT(NODE2).LT.0) THEN
                     NAUGNOD=NAUGNOD+1
                     SAVE(NAUGNOD)=NODE2
                  END IF
                  NLABEL=NLABEL+1
                  LABEL(NLABEL)=NODE2
                  MARK(NODE2)=.TRUE.
                  DELX=DELX+X(ARC)
               END IF
            END IF
            PRVARC = ARC
            ARC = TNXTOU(ARC)
         ELSE
            TMPARC = ARC
            ARC = TNXTOU(ARC)
            TNXTOU(TMPARC) = -1
            IF (PRVARC .EQ. 0) THEN
               TFSTOU(I) = ARC
            ELSE
               TNXTOU(PRVARC) = ARC
            END IF
         END IF
         GOTO 518
      END IF

      PRVARC = 0
      ARC=TFSTIN(I)
519   IF (ARC.GT.0) THEN

C       ***** ARC is an incoming arc into NODE *****

         IF (RC(ARC) .EQ. 0) THEN
            IF (U(ARC) .GT. 0) THEN
               NODE2=STARTN(ARC)
               IF (.NOT. MARK(NODE2)) THEN

C    * NODE2 is not in the labeled set.  Add NODE2 to the labeled set. *

                  PRDCSR(NODE2)=-ARC
                  IF (DFCT(NODE2).LT.0) THEN
                     NAUGNOD=NAUGNOD+1
                     SAVE(NAUGNOD)=NODE2
                  END IF
                  NLABEL=NLABEL+1
                  LABEL(NLABEL)=NODE2
                  MARK(NODE2)=.TRUE.
                  DELX=DELX+U(ARC)
               END IF
            END IF
```

```
            PRVARC = ARC
            ARC = TNXTIN(ARC)
          ELSE
            TMPARC = ARC
            ARC = TNXTIN(ARC)
            TNXTIN(TMPARC) = -1
            IF (PRVARC .EQ. 0) THEN
               TFSTIN(I) = ARC
            ELSE
               TNXTIN(PRVARC) = ARC
            END IF
          END IF
          GOTO 519
        END IF

C   * correct the residual capacity of the scanned nodes cut *

        ARC=PRDCSR(I)
        IF (ARC.GT.0) THEN
           DELX=DELX-X(ARC)
        ELSE
           DELX=DELX-U(-ARC)
        END IF

C  ********** end of scanning of node I for positive deficit case ****

                ELSE

C     ******* scanning node I for case of negative deficit ****

        NAUGNOD=0

        PRVARC = 0
        ARC=TFSTIN(I)
520     IF (ARC.GT.0) THEN
          IF (RC(ARC) .EQ. 0) THEN
            IF (X(ARC) .GT. 0) THEN
               NODE2=STARTN(ARC)
               IF (.NOT. MARK(NODE2)) THEN
                  PRDCSR(NODE2)=ARC
                  IF (DFCT(NODE2).GT.0) THEN
                     NAUGNOD=NAUGNOD+1
                     SAVE(NAUGNOD)=NODE2
                  END IF
                  NLABEL=NLABEL+1
                  LABEL(NLABEL)=NODE2
                  MARK(NODE2)=.TRUE.
                  DELX=DELX+X(ARC)
               END IF
            END IF
            PRVARC = ARC
            ARC = TNXTIN(ARC)
          ELSE
            TMPARC = ARC
```

```
            ARC = TNXTIN(ARC)
            TNXTIN(TMPARC) = -1
            IF (PRVARC .EQ. 0) THEN
               TFSTIN(I) = ARC
            ELSE
               TNXTIN(PRVARC) = ARC
            END IF
          END IF
          GOTO 520
        END IF

        PRVARC = 0
        ARC = TFSTOU(I)
521     IF (ARC.GT.0) THEN
          IF (RC(ARC) .EQ. 0) THEN
             IF (U(ARC) .GT. 0) THEN
                NODE2=ENDN(ARC)
                IF (.NOT. MARK(NODE2)) THEN
                   PRDCSR(NODE2)=-ARC
                   IF (DFCT(NODE2).GT.0) THEN
                      NAUGNOD=NAUGNOD+1
                      SAVE(NAUGNOD)=NODE2
                   END IF
                   NLABEL=NLABEL+1
                   LABEL(NLABEL)=NODE2
                   MARK(NODE2)=.TRUE.
                   DELX=DELX+U(ARC)
                END IF
             END IF
             PRVARC = ARC
             ARC = TNXTOU(ARC)
          ELSE
             TMPARC = ARC
             ARC = TNXTOU(ARC)
             TNXTOU(TMPARC) = -1
             IF (PRVARC .EQ. 0) THEN
                TFSTOU(I) = ARC
             ELSE
                TNXTOU(PRVARC) = ARC
             END IF
          END IF
          GOTO 521
        END IF

        ARC=PRDCSR(I)
        IF (ARC.GT.0) THEN
           DELX=DELX-X(ARC)
        ELSE
           DELX=DELX-U(-ARC)
        END IF
        END IF

C   ****** ADD DEFICIT OF NODE SCANNED TO DM ******
```

```
       DM=DM+DFCT(I)

C      Check if the set of scanned nodes correspond
C      to a dual ascent direction; if yes, perform a
C      price adjustment step, otherwise continue labeling

       IF (NSCAN.LT.NLABEL) THEN
         IF (SWITCH) GO TO 210
         IF ((DELX.GE.DM).AND.(DELX.GE.-DM)) GO TO 210
       END IF

C   ************* TRY A PRICE CHANGE ***********

C   Note that since DELX-ABS(DM) is an OVERESTIMATE of ascent slope, we
C   may occasionally try a direction that is not really an ascent
direction.
C   In this case the ANCNTx routines return with QUIT set to .FALSE. .
The
C   main code, in turn, then tries to label some more nodes.

           IF (POSIT) THEN
             CALL ASCNT1(DM,DELX,NLABEL,AUGNOD,FEASBL,
     $       SWITCH,NSCAN,NODE,PREVNODE)
             NUM_ASCNT=NUM_ASCNT+1
           ELSE
             CALL ASCNT2(DM,DELX,NLABEL,AUGNOD,FEASBL,
     $       SWITCH,NSCAN,NODE,PREVNODE)
             NUM_ASCNT=NUM_ASCNT+1
           END IF
           IF (.NOT.FEASBL) GO TO 400
           IF (.NOT.SWITCH)  GO TO 100
           IF ((SWITCH).AND.(AUGNOD.GT.0)) THEN
             NAUGNOD=1
             SAVE(1)=AUGNOD
           END IF

C          CHECK IF AUGMENTATION IS POSSIBLE.
C          IF NOT RETURN TO SCAN ANOTHER NODE.

210        CONTINUE

           IF (NAUGNOD.EQ.0) GO TO 120

C          Do the augmentation.

           DO 96 J=1,NAUGNOD
           NUM_AUGM=NUM_AUGM+1
           AUGNOD=SAVE(J)
           IF (POSIT) THEN
                CALL AUGFL1(AUGNOD,NODE,PREVNODE)
           ELSE
                CALL AUGFL2(AUGNOD,NODE,PREVNODE)
           END IF
96         CONTINUE
```

```
C        ** RETURN TO TAKE UP ANOTHER NODE W/ NONZERO DEFICIT **

         GO TO 100

C     ******* problem is found to be infeasible *********************

400   PRINT *,' PROBLEM IS FOUND TO BE INFEASIBLE.'
      FEASBL = .FALSE.
      RETURN
      END




      SUBROUTINE SHORT_PATH(NODE,CAPTHRESH)

C  This routine is a modified shortest path method for finding
C  shortest augmenting paths starting at NODE. It uses essentially
C  Dijkstra's method with a binary heap data structure but it
C  ignores all arcs with capacity less than the parameter CAPTHRESH

      IMPLICIT NONE
      INTEGER STARTN,ENDN,U,X,RC,DFCT,D,PRDCSR,FOU,FIN,NXTIN,NXTOU
      INTEGER HP,Q,N,NA,LARGE,NHP,OPNODE,DV,K,K2,HP2,HP3,HP1,I,DP1
      INTEGER DX,IB,ROOT,MAXD,ARC,NARC,CURNODE,NOLIST,DK,DIFF
      INTEGER START,END,LIST,CAPTHRESH,NODE
      LOGICAL*1 MARK
      COMMON/ARRAYS/STARTN/ARRAYE/ENDN/ARRAYU/U/ARRAYX/X/ARRAY9/RC
     $/ARRAYB/DFCT/BLK1/D/BLK2/PRDCSR/BLK3/FOU/BLK4/NXTOU/BLK5/FIN
     $/BLK6/NXTIN/BLK9/MARK/BLK10/HP/BLK11/Q/BLK12/LIST
      COMMON/L/N,NA,LARGE

      DIMENSION STARTN(1),ENDN(1),U(1),X(1),RC(1),DFCT(1)
      DIMENSION D(1),FOU(1),NXTOU(1),MARK(1)
      DIMENSION FIN(1),NXTIN(1),HP(1),PRDCSR(1),Q(1),LIST(1)


      DO 10 I=1,N
        D(I)=LARGE
        Q(I)=0
        MARK(I)=.FALSE.
10    CONTINUE


      D(NODE)=0
      PRDCSR(NODE)=0
      NOLIST=0
      NHP=0
      CURNODE=NODE

18    CONTINUE
```

```
      ARC=FOU(CURNODE)
20    IF (ARC.GT.0) THEN
        IF ((RC(ARC).GE.0).AND.(U(ARC).GT.CAPTHRESH)) THEN
          OPNODE=ENDN(ARC)
          DV=D(CURNODE)+RC(ARC)
          IF (D(OPNODE).GT.DV) THEN
           D(OPNODE) = DV
           PRDCSR(OPNODE) = -ARC
           IF (Q(OPNODE).EQ.0) THEN

C     INSERT OPNODE INTO THE HEAP

             NHP=NHP+1
             Q(OPNODE)=NHP
           END IF

C     UPDATE THE HEAP

           K = Q(OPNODE)
25         K2 = K/2
           IF ( K2 .GT. 0 ) THEN
            HP2 = HP(K2)
            IF ( DV .LT. D(HP2) ) THEN
              HP(K) = HP2
              Q(HP2) = K
              K = K2
              GO TO 25
            END IF
           END IF

           HP(K) = OPNODE
           Q(OPNODE) = K
         END IF
        END IF
        ARC=NXTOU(ARC)
        GO TO 20
      END IF


      ARC=FIN(CURNODE)
30    IF (ARC.GT.0) THEN
        IF ((RC(ARC).LE.0).AND.(X(ARC).GT.CAPTHRESH)) THEN
          OPNODE=STARTN(ARC)
          DV=D(CURNODE)-RC(ARC)
          IF (D(OPNODE).GT.DV) THEN
           D(OPNODE) = DV
           PRDCSR(OPNODE) = ARC
           IF ( Q(OPNODE) .EQ. 0 ) THEN

C     INSERT OPNODE INTO THE HEAP

             NHP = NHP + 1
             Q(OPNODE) = NHP
           END IF
```

```
C     UPDATE THE HEAP

            K = Q(OPNODE)
35          K2 = K/2
            IF ( K2 .GT. 0 ) THEN
             HP2 = HP(K2)
             IF ( DV .LT. D(HP2) ) THEN
               HP(K) = HP2
               Q(HP2) = K
               K = K2
               GO TO 35
             END IF
            END IF

            HP(K) = OPNODE
            Q(OPNODE) = K
           END IF
          END IF
          ARC=NXTIN(ARC)
          GO TO 30
        END IF


C   MARK PERMANENTLY LABELED NODE

        NOLIST=NOLIST+1
        LIST(NOLIST)=CURNODE
        MARK(CURNODE)=.TRUE.


C    REMOVE THE NEW CURRENT NODE FROM THE HEAP
C    CHECK IF IT HAS POSITIVE DEFICIT

        IF (NHP.EQ.0) GO TO 130
        CURNODE = HP(1)
        Q(CURNODE) = 0
        IF (DFCT(CURNODE).GT.0) GO TO 130
        NHP = NHP - 1

C    CHECK WHETHER THE HEAP IS EMPTY

        IF (NHP.EQ.0) GO TO 18


C    UPDATE THE HEAP

80       HP1 = HP(NHP+1)
         DP1 = D(HP1)
         K = 1
90       K2 = 2*K
         HP2 = HP(K2)
         IF ( K2-NHP ) 100,110,120
100      HP3 = HP(K2+1)
```

```
       IF ( D(HP2) .LT. D(HP3) ) GO TO 110
       HP2 = HP3
       K2 = K2 + 1
110    IF ( DP1 .LE. D(HP2) ) GO TO 120
       HP(K) = HP2
       Q(HP2) = K
       K = K2
       GO TO 90
120    HP(K)=HP1
       Q(HP1) = K
       GO TO 18

130   CONTINUE

C    SHORTEST AUGMENTING PATH HAS BEEN COMPUTED

C    UPDATE THE REDUCED COSTS OF ALL ARCS

      MAXD=D(CURNODE)
      DO 140 I=1,NOLIST
        K=LIST(I)
        DK=D(K)
        DIFF=MAXD-DK
        ARC=FOU(K)
134     IF (ARC.GT.0) THEN
          END=ENDN(ARC)
          IF (.NOT.MARK(END)) THEN
            RC(ARC)=RC(ARC)-DIFF
          ELSE
            RC(ARC)=RC(ARC)-D(END)+DK
          END IF
          ARC=NXTOU(ARC)
          GOTO 134
        END IF
        ARC=FIN(K)
136     IF (ARC.GT.0) THEN
          START=STARTN(ARC)
          IF (.NOT.MARK(START)) THEN
            RC(ARC)=RC(ARC)+DIFF
          END IF
          ARC=NXTIN(ARC)
          GOTO 136
        END IF
140   CONTINUE

C    DO THE AUGMENTATION AND EXIT

      DX=MIN0(DFCT(CURNODE),-DFCT(NODE))
      IF (DX.GT.0) THEN
       IB=CURNODE
150    IF (PRDCSR(IB).NE.0) THEN
          ARC=PRDCSR(IB)
          IF (ARC.GT.0) THEN
            DX=MIN0(DX,X(ARC))
```

```
            IB=ENDN(ARC)
          ELSE
            DX=MIN0(DX,U(-ARC))
            IB=STARTN(-ARC)
          END IF
          GOTO 150
        END IF

        ROOT=IB

C     ******* update the flow and deficits *******

          DFCT(CURNODE)=DFCT(CURNODE)-DX
          DFCT(ROOT)=DFCT(ROOT)+DX
          IB=CURNODE
160       IF (IB.NE.ROOT) THEN
           ARC=PRDCSR(IB)
           IF (ARC.GT.0) THEN
             X(ARC)=X(ARC)-DX
             U(ARC)=U(ARC)+DX
             IB=ENDN(ARC)
           ELSE
             NARC=-ARC
             X(NARC)=X(NARC)+DX
             U(NARC)=U(NARC)-DX
             IB=STARTN(NARC)
           END IF
           GOTO 160
          END IF
        END IF

        RETURN
        END




      SUBROUTINE CHECKCOMPSLACK

C     This subroutine checks complementary slackness.
C     It is used for diagnostic purposes.

      IMPLICIT INTEGER (A-Z)

      COMMON/ARRAYS/STARTN/ARRAYE/ENDN/ARRAYU/U/ARRAYX/X/ARRAY9/RC
     $/ARRAYB/DFCT/BLK3/FOU/BLK4/NXTOU/BLK5/FIN
     $/BLK6/NXTIN/L/N,NA,LARGE

      DIMENSION STARTN(1),ENDN(1),U(1),X(1),RC(1),DFCT(1)
      DIMENSION FOU(1),NXTOU(1)
      DIMENSION FIN(1),NXTIN(1)
```

```
      DO 100 ARC = 1,NA
        IF ((RC(ARC) .GT. 0).AND.(X(ARC) .GT. 0)) THEN
            PRINT*,'COMPLEMENTARY SLACKNESS VIOLATED',ARC
            PRINT*,RC(ARC),X(ARC)
            PAUSE
        ELSE
          IF ((RC(ARC) .LT. 0).AND.(U(ARC).GT.0)) THEN
            PRINT*,'COMPLEMENTARY SLACKNESS VIOLATED', ARC
            PRINT*,RC(ARC),U(ARC)
            PAUSE
          END IF
        END IF
100   CONTINUE

      RETURN
      END



      SUBROUTINE PRINTFLOWS(NODE)

C     This subroutine prints the deficit and the flows of
C     arcs incident to NODE. It is used for diagnostic purposes
C     in case of an infeasible problem.
C
      IMPLICIT INTEGER (A-Z)

      COMMON/ARRAYS/STARTN/ARRAYE/ENDN/ARRAYU/U/ARRAYX/X
     $/ARRAYB/DFCT/BLK3/FOU/BLK4/NXTOU/BLK5/FIN/BLK6/NXTIN

      DIMENSION STARTN(1),ENDN(1),U(1),X(1),DFCT(1)
      DIMENSION FOU(1),NXTOU(1)
      DIMENSION FIN(1),NXTIN(1)


      PRINT*,'DEFICIT (I.E., NET FLOW OUT) OF NODE =',DFCT(NODE)
      PRINT*,'FLOWS AND CAPACITIES OF INCIDENT ARCS OF NODE',NODE
      IF (FOU(NODE).EQ.0) THEN
        PRINT*,'NO OUTGOING ARCS'
      ELSE
        ARC=FOU(NODE)
5       IF (ARC.GT.0) THEN
          PRINT*,'ARC',ARC,'  BETWEEN NODES',NODE,ENDN(ARC)
          PRINT*,'FLOW =',X(ARC)
          PRINT*,'RESIDUAL CAPACITY =',U(ARC)
          ARC=NXTOU(ARC)
          GO TO 5
        END IF
      END IF

      IF (FIN(NODE).EQ.0) THEN
        PRINT*,'NO INCOMING ARCS'
      ELSE
```

```
        ARC=FIN(NODE)
10       IF (ARC.GT.0) THEN
          PRINT*,'ARC',ARC,'  BETWEEN NODES',STARTN(ARC),NODE
          PRINT*,'FLOW =',X(ARC)
          PRINT*,'RESIDUAL CAPACITY =',U(ARC)
          ARC=NXTIN(ARC)
          GO TO 10
        END IF
      END IF

      RETURN
      END




      SUBROUTINE AUGFL1(AUGNOD,NODE,PREVNODE)

C     This subroutine performs the flow augmentation step.
C     A flow augmenting path has been identified in the scanning
C     step and here the flow of all arcs positively (negatively)
C     oriented in the flow augmenting path is decreased (increased)
C     to decrease the total deficit.

      IMPLICIT INTEGER (A-Z)
      COMMON/ARRAYS/STARTN/ARRAYE/ENDN/ARRAYU/U/ARRAYX/X
     $/ARRAYB/DFCT/BLK2/PRDCSR
      DIMENSION STARTN(1),ENDN(1),U(1),X(1),DFCT(1),PRDCSR(1)
      COMMON /BLK14/NXTQUEUE
      DIMENSION NXTQUEUE(1)


C     A flow augmenting path ending at AUGNOD has been found.
C     First determine DX, the amount of flow change.

      DX=-DFCT(AUGNOD)
      IB=AUGNOD
500   IF (PRDCSR(IB).NE.0) THEN
        ARC=PRDCSR(IB)
        IF (ARC.GT.0) THEN
          DX=MIN0(DX,X(ARC))
          IB=STARTN(ARC)
        ELSE
          DX=MIN0(DX,U(-ARC))
          IB=ENDN(-ARC)
        END IF
        GOTO 500
      END IF
      ROOT=IB
      DX=MIN0(DX,DFCT(ROOT))
      IF (DX .LE. 0) RETURN

C     ***** Update the flow by decreasing (increasing) the flow of
```

```
C     all arcs positively (negatively) oriented in the flow
C     augmenting path.  Adjust the deficits accordingly. *****

      DFCT(AUGNOD)=DFCT(AUGNOD)+DX
      IF (NXTQUEUE(AUGNOD).EQ.0) THEN
            NXTQUEUE(PREVNODE)=AUGNOD
            NXTQUEUE(AUGNOD)=NODE
            PREVNODE=AUGNOD
      END IF
      DFCT(ROOT)=DFCT(ROOT)-DX
      IB=AUGNOD
501   IF (IB.NE.ROOT) THEN
        ARC=PRDCSR(IB)
        IF (ARC.GT.0) THEN
          X(ARC)=X(ARC)-DX
          U(ARC)=U(ARC)+DX
          IB=STARTN(ARC)
        ELSE
          NARC=-ARC
          X(NARC)=X(NARC)+DX
          U(NARC)=U(NARC)-DX
          IB=ENDN(NARC)
        END IF
        GOTO 501
      END IF
      RETURN
      END




      SUBROUTINE ASCNT1(DM,DELX,NLABEL,AUGNOD,FEASBL,SWITCH,
     $NSCAN,STARTNODE,PREVNODE)

C     This subroutine essentially performs the multi-node
C     price adjustment step.  It first checks if the set
C     of scanned nodes correspond to a dual ascent direction.
C     If yes, then decrease the price of all scanned nodes.
C     There are two possibilities for price adjustment:
C     If SWITCH=.TRUE. then the set of scanned nodes
C     corresponds to an elementary direction of maximal
C     rate of ascent, in which case the price of all scanned
C     nodes are decreased until the next breakpoint in the
C     dual cost is encountered.  At this point some arc
C     becomes balanced and more node(s) are added to the
C     labeled set.
C     If SWITCH=.FALSE. then the prices of all scanned nodes
C     are decreased until the rate of ascent becomes
C     negative (this corresponds to the price adjustment
C     step in which both the line search and the degenerate
C     ascent iteration are implemented).

      IMPLICIT INTEGER (A-Z)
```

```
      LOGICAL*1 SCAN,MARK,SWITCH,FEASBL
      COMMON/ARRAYS/STARTN/ARRAYE/ENDN/ARRAYU/U/ARRAYX/X/ARRAY9/RC
     $/ARRAYB/DFCT/BLK1/LABEL/BLK2/PRDCSR/BLK3/FOU/BLK4/
     $NXTOU/BLK5/FIN/BLK6/NXTIN/BLK7/SAVE/BLK8/SCAN/BLK9/MARK
     $/L/N,NA,LARGE
      COMMON /BLK10/TFSTOU/BLK11/TNXTOU/BLK12/TFSTIN/BLK13/TNXTIN
      COMMON /ASCBLK/B
      COMMON /BLK14/NXTQUEUE
      DIMENSION NXTQUEUE(1)
      DIMENSION TFSTOU(1),TNXTOU(1),TFSTIN(1),TNXTIN(1)
      DIMENSION STARTN(1),ENDN(1),U(1),X(1),RC(1),DFCT(1),LABEL(1)
      DIMENSION PRDCSR(1),FOU(1),NXTOU(1),FIN(1),NXTIN(1)
      DIMENSION SAVE(1),SCAN(1),MARK(1)

C     Store the arcs between the set of scanned nodes and
C     its complement in SAVE and compute DELPRC, the stepsize
C     to the next breakpoint in the dual cost in the direction
C     of decreasing prices of the scanned nodes.

      DELPRC=LARGE
      DLX=0
      NSAVE=0

C   Calculate the array SAVE of arcs across the cut of scanned
C   nodes in a different way depending on whether NSCAN>N/2 or not.
C   This is done for efficiency.

      IF (NSCAN.LE.N/2) THEN
      DO 1 I=1,NSCAN
        NODE=LABEL(I)
          ARC=FOU(NODE)
500       IF (ARC.GT.0) THEN

C   ARC is an arc pointing from the set of scanned nodes to its
complement.

            NODE2=ENDN(ARC)
            IF (.NOT.SCAN(NODE2)) THEN
              NSAVE=NSAVE+1
              SAVE(NSAVE)=ARC
              RDCOST=RC(ARC)
      IF ((RDCOST.EQ.0).AND.(PRDCSR(NODE2).NE.ARC)) DLX=DLX+X(ARC)
            IF ((RDCOST.LT.0).AND.(RDCOST.GT.-DELPRC)) DELPRC=-RDCOST
            END IF
            ARC=NXTOU(ARC)
            GOTO 500
          END IF
          ARC=FIN(NODE)

501       IF (ARC.GT.0) THEN

C   ARC is an arc pointing to the set of scanned nodes from its
complement.
```

```
              NODE2=STARTN(ARC)
              IF (.NOT.SCAN(NODE2)) THEN
                NSAVE=NSAVE+1
                SAVE(NSAVE)=-ARC
                RDCOST=RC(ARC)
      IF ((RDCOST.EQ.0).AND.(PRDCSR(NODE2).NE.-ARC)) DLX=DLX+U(ARC)
                IF ((RDCOST.GT.0).AND.(RDCOST.LT.DELPRC)) DELPRC=RDCOST
              END IF
              ARC=NXTIN(ARC)
              GOTO 501
            END IF
1. CONTINUE

      ELSE

      DO 2 NODE=1,N
        IF (SCAN(NODE)) GO TO 2
          ARC=FIN(NODE)
502       IF (ARC.GT.0) THEN
            NODE2=STARTN(ARC)
            IF (SCAN(NODE2)) THEN
              NSAVE=NSAVE+1
              SAVE(NSAVE)=ARC
              RDCOST=RC(ARC)
      IF ((RDCOST.EQ.0).AND.(PRDCSR(NODE).NE.ARC)) DLX=DLX+X(ARC)
              IF ((RDCOST.LT.0).AND.(RDCOST.GT.-DELPRC)) DELPRC=-RDCOST
              END IF
              ARC=NXTIN(ARC)
              GOTO 502
            END IF
            ARC=FOU(NODE)
503         IF (ARC.GT.0) THEN
              NODE2=ENDN(ARC)
              IF (SCAN(NODE2)) THEN
                NSAVE=NSAVE+1
                SAVE(NSAVE)=-ARC
                RDCOST=RC(ARC)
      IF ((RDCOST.EQ.0).AND.(PRDCSR(NODE).NE.-ARC)) DLX=DLX+U(ARC)
                IF ((RDCOST.GT.0).AND.(RDCOST.LT.DELPRC)) DELPRC=RDCOST
              END IF
              ARC=NXTOU(ARC)
              GOTO 503
            END IF
2. CONTINUE
      END IF

C     Check if the set of scanned nodes truly corresponds
C     to a dual ascent direction.  Here DELX+DLX is the exact
C     sum of the flow on arcs from the scanned set to the
C     unscanned set plus the ( capacity - flow ) on arcs from
C     the unscanned set to the scanned set.

C     If this is not the case, we set SWITCH to .TRUE.
C     and exit the subroutine.
```

```
      IF (DELX+DLX.GE.DM) THEN
        SWITCH=.TRUE.
        AUGNOD=0
        DO 3 I=NSCAN+1,NLABEL
          NODE=LABEL(I)
          IF (DFCT(NODE).LT.0) AUGNOD=NODE
3. CONTINUE
        RETURN
      END IF
      DELX=DELX+DLX

C     Check that the problem is feasible

4. IF (DELPRC.EQ.LARGE) THEN

C       We can decrease the dual cost without bound.
C       Therefore the primal problem is infeasible.

        FEASBL=.FALSE.
        RETURN
      END IF

C     Decrease prices of the scanned nodes, add more
C     nodes to the labeled set & check if a newly labeled node
C     has negative deficit.

      IF (SWITCH) THEN
        AUGNOD=0
        DO 7 I=1,NSAVE
          ARC=SAVE(I)
          IF (ARC.GT.0) THEN
            RC(ARC)=RC(ARC)+DELPRC
            IF (RC(ARC).EQ.0) THEN
              NODE2=ENDN(ARC)
              IF (TNXTOU(ARC) .LT. 0) THEN
                TNXTOU(ARC) = TFSTOU(STARTN(ARC))
                TFSTOU(STARTN(ARC)) = ARC
              END IF
              IF (TNXTIN(ARC) .LT. 0) THEN
                TNXTIN(ARC) = TFSTIN(NODE2)
                TFSTIN(NODE2) = ARC
              END IF
              PRDCSR(NODE2)=ARC
              IF (DFCT(NODE2).LT.0) THEN
                AUGNOD=NODE2
              ELSE
                IF (.NOT.MARK(NODE2)) THEN
                 MARK(NODE2)=.TRUE.
                 NLABEL=NLABEL+1
                 LABEL(NLABEL)=NODE2
                END IF
              END IF
            END IF
```

```
          ELSE
            ARC=-ARC
            RC(ARC)=RC(ARC)-DELPRC
            IF (RC(ARC).EQ.0) THEN
              NODE2=STARTN(ARC)
              IF (TNXTOU(ARC) .LT. 0) THEN
                TNXTOU(ARC) = TFSTOU(NODE2)
                TFSTOU(NODE2) = ARC
              END IF
              IF (TNXTIN(ARC) .LT. 0) THEN
                TNXTIN(ARC) = TFSTIN(ENDN(ARC))
                TFSTIN(ENDN(ARC)) = ARC
              END IF
              PRDCSR(NODE2)=-ARC
              IF (DFCT(NODE2).LT.0) THEN
                AUGNOD=NODE2
              ELSE
               IF (.NOT.MARK(NODE2)) THEN
                MARK(NODE2)=.TRUE.
                NLABEL=NLABEL+1
                LABEL(NLABEL)=NODE2
               END IF
              END IF
            END IF
          END IF
7       CONTINUE
        RETURN

      ELSE

C     Decrease the prices of the scanned nodes by DELPRC.
C     Adjust arc flow to maintain complementary slackness with
C     the prices.

      NB = 0
      DO 6 I=1,NSAVE
        ARC=SAVE(I)
        IF (ARC.GT.0) THEN
            T1=RC(ARC)
            IF (T1.EQ.0) THEN
             T2=X(ARC)
             T3=STARTN(ARC)
             DFCT(T3)=DFCT(T3)-T2
             IF (NXTQUEUE(T3).EQ.0) THEN
                NXTQUEUE(PREVNODE)=T3
                NXTQUEUE(T3)=STARTNODE
                PREVNODE=T3
             END IF
             T3=ENDN(ARC)
             DFCT(T3)=DFCT(T3)+T2
             IF (NXTQUEUE(T3).EQ.0) THEN
                NXTQUEUE(PREVNODE)=T3
                NXTQUEUE(T3)=STARTNODE
                PREVNODE=T3
```

```
            END IF

            U(ARC)=U(ARC)+T2
            X(ARC)=0
          END IF
        RC(ARC)=T1+DELPRC
        IF (RC(ARC).EQ.0) THEN
           DELX=DELX+X(ARC)
           NB = NB + 1
           PRDCSR(NB) = ARC
        ENDIF
      ELSE
        ARC=-ARC
        T1=RC(ARC)
        IF (T1.EQ.0) THEN
          T2=U(ARC)
          T3=STARTN(ARC)
          DFCT(T3)=DFCT(T3)+T2
          IF (NXTQUEUE(T3).EQ.0) THEN
              NXTQUEUE(PREVNODE)=T3
              NXTQUEUE(T3)=STARTNODE
              PREVNODE=T3
           END IF

          T3=ENDN(ARC)
          DFCT(T3)=DFCT(T3)-T2
          IF (NXTQUEUE(T3).EQ.0) THEN
              NXTQUEUE(PREVNODE)=T3
              NXTQUEUE(T3)=STARTNODE
              PREVNODE=T3
           END IF

          X(ARC)=X(ARC)+T2
          U(ARC)=0
        END IF
        RC(ARC)=T1-DELPRC
        IF (RC(ARC).EQ.0) THEN
           DELX=DELX+U(ARC)
           NB = NB + 1
           PRDCSR(NB) = ARC
        END IF
      END IF
6     CONTINUE
      END IF

      IF (DELX.LE.DM) THEN

C       The set of scanned nodes still corresponds to a
C       dual (possibly degenerate) ascent direction.  Compute
C       the stepsize DELPRC to the next breakpoint in the
C       dual cost.

        DELPRC=LARGE
        DO 10 I=1,NSAVE
```

```
          ARC=SAVE(I)
          IF (ARC.GT.0) THEN
            RDCOST=RC(ARC)
            IF ((RDCOST.LT.0).AND.(RDCOST.GT.-DELPRC)) DELPRC=-RDCOST
          ELSE
            ARC=-ARC
            RDCOST=RC(ARC)
            IF ((RDCOST.GT.0).AND.(RDCOST.LT.DELPRC)) DELPRC=RDCOST
          END IF
10      CONTINUE
        IF ((DELPRC.NE.LARGE).OR.(DELX.LT.DM)) GO TO 4
       END IF

C    Add new balanced arcs to the superset of balanced arcs.

       DO 9 I=1,NB
         ARC=PRDCSR(I)
           IF (TNXTIN(ARC).EQ.-1) THEN
             J=ENDN(ARC)
             TNXTIN(ARC)=TFSTIN(J)
             TFSTIN(J)=ARC
           END IF
           IF (TNXTOU(ARC).EQ.-1) THEN
             J=STARTN(ARC)
             TNXTOU(ARC)=TFSTOU(J)
             TFSTOU(J)=ARC
           END IF
9      CONTINUE

       RETURN
       END




       SUBROUTINE AUGFL2(AUGNOD,NODE,PREVNODE)
       IMPLICIT INTEGER (A-Z)
       COMMON/ARRAYS/STARTN/ARRAYE/ENDN/ARRAYU/U/ARRAYX/X
      $/ARRAYB/DFCT/BLK2/PRDCSR
       COMMON /BLK14/NXTQUEUE
       DIMENSION NXTQUEUE(1)
       DIMENSION STARTN(1),ENDN(1),U(1),X(1),DFCT(1),PRDCSR(1)

       DX=DFCT(AUGNOD)
       IB=AUGNOD
500    IF (PRDCSR(IB).NE.0) THEN
         ARC=PRDCSR(IB)
         IF (ARC.GT.0) THEN
           DX=MIN0(DX,X(ARC))
           IB=ENDN(ARC)
         ELSE
           DX=MIN0(DX,U(-ARC))
           IB=STARTN(-ARC)
         END IF
```

```
        GOTO 500
       END IF
      ROOT=IB
      DX=MIN0(DX,-DFCT(ROOT))
      IF (DX .LE. 0) RETURN

C     Update the flow and deficits

      DFCT(AUGNOD)=DFCT(AUGNOD)-DX
      IF (NXTQUEUE(AUGNOD).EQ.0) THEN
            NXTQUEUE(PREVNODE)=AUGNOD
            NXTQUEUE(AUGNOD)=NODE
            PREVNODE=AUGNOD
      END IF
      DFCT(ROOT)=DFCT(ROOT)+DX
      IB=AUGNOD
501   IF (IB.NE.ROOT) THEN
        ARC=PRDCSR(IB)
        IF (ARC.GT.0) THEN
          X(ARC)=X(ARC)-DX
          U(ARC)=U(ARC)+DX
          IB=ENDN(ARC)
        ELSE
          NARC=-ARC
          X(NARC)=X(NARC)+DX
          U(NARC)=U(NARC)-DX
          IB=STARTN(NARC)
        END IF
        GOTO 501
      END IF
      RETURN
      END




      SUBROUTINE ASCNT2(DM,DELX,NLABEL,AUGNOD,FEASBL,SWITCH,
     $NSCAN,STARTNODE,PREVNODE)
      IMPLICIT INTEGER (A-Z)

      LOGICAL*1 SCAN,MARK,SWITCH,FEASBL
      COMMON/ARRAYS/STARTN/ARRAYE/ENDN/ARRAYU/U/ARRAYX/X/ARRAY9/RC
     $/ARRAYB/DFCT/BLK1/LABEL/BLK2/PRDCSR/BLK3/FOU/BLK4/
     $NXTOU/BLK5/FIN/BLK6/NXTIN/BLK7/SAVE/BLK8/SCAN/BLK9/MARK
     $/L/N,NA,LARGE
      COMMON /BLK10/TFSTOU/BLK11/TNXTOU/BLK12/TFSTIN/BLK13/TNXTIN
      COMMON /ASCBLK/B
      COMMON /BLK14/NXTQUEUE
      DIMENSION NXTQUEUE(1)
      DIMENSION TFSTOU(1),TNXTOU(1),TFSTIN(1),TNXTIN(1)
      DIMENSION STARTN(1),ENDN(1),U(1),X(1),RC(1),DFCT(1),LABEL(1)
      DIMENSION PRDCSR(1),FOU(1),NXTOU(1),FIN(1),NXTIN(1)
      DIMENSION SAVE(1),SCAN(1),MARK(1)
```

```
C     ******* augment flows across the cut & compute price rise *****

      DELPRC=LARGE
      DLX=0
      NSAVE=0
      IF (NSCAN.LE.N/2) THEN
      DO 1 I=1,NSCAN
        NODE=LABEL(I)
          ARC=FIN(NODE)
500       IF (ARC.GT.0) THEN
            NODE2=STARTN(ARC)
            IF (.NOT.SCAN(NODE2)) THEN
              NSAVE=NSAVE+1
              SAVE(NSAVE)=ARC
              RDCOST=RC(ARC)
      IF ((RDCOST.EQ.0).AND.(PRDCSR(NODE2).NE.ARC)) DLX=DLX+X(ARC)
            IF ((RDCOST.LT.0).AND.(RDCOST.GT.-DELPRC)) DELPRC=-RDCOST
            END IF
            ARC=NXTIN(ARC)
            GOTO 500
          END IF
          ARC=FOU(NODE)
501       IF (ARC.GT.0) THEN
            NODE2=ENDN(ARC)
            IF (.NOT.SCAN(NODE2)) THEN
              NSAVE=NSAVE+1
              SAVE(NSAVE)=-ARC
              RDCOST=RC(ARC)
        IF ((RDCOST.EQ.0).AND.(PRDCSR(NODE2).NE.-ARC)) DLX=DLX+U(ARC)
              IF ((RDCOST.GT.0).AND.(RDCOST.LT.DELPRC)) DELPRC=RDCOST
            END IF
            ARC=NXTOU(ARC)
            GOTO 501
          END IF
1. CONTINUE
      ELSE
      DO 2 NODE=1,N
        IF (SCAN(NODE)) GO TO 2
          ARC=FOU(NODE)
502       IF (ARC.GT.0) THEN
            NODE2=ENDN(ARC)
            IF (SCAN(NODE2)) THEN
              NSAVE=NSAVE+1
              SAVE(NSAVE)=ARC
              RDCOST=RC(ARC)
      IF ((RDCOST.EQ.0).AND.(PRDCSR(NODE).NE.ARC)) DLX=DLX+X(ARC)
            IF ((RDCOST.LT.0).AND.(RDCOST.GT.-DELPRC)) DELPRC=-RDCOST
            END IF
            ARC=NXTOU(ARC)
            GOTO 502
          END IF
          ARC=FIN(NODE)
503       IF (ARC.GT.0) THEN
            NODE2=STARTN(ARC)
```

```
            IF (SCAN(NODE2)) THEN
              NSAVE=NSAVE+1
              SAVE(NSAVE)=-ARC
              RDCOST=RC(ARC)
        IF ((RDCOST.EQ.0).AND.(PRDCSR(NODE).NE.-ARC)) DLX=DLX+U(ARC)
              IF ((RDCOST.GT.0).AND.(RDCOST.LT.DELPRC)) DELPRC=RDCOST
            END IF
            ARC=NXTIN(ARC)
            GOTO 503
          END IF
2. CONTINUE
      END IF
      IF (DELX+DLX.GE.-DM) THEN
        SWITCH=.TRUE.
        AUGNOD=0
        DO 3 I=NSCAN+1,NLABEL
          NODE=LABEL(I)
          IF (DFCT(NODE).GT.0) AUGNOD=NODE
3. CONTINUE
        RETURN
      END IF
      DELX=DELX+DLX

C     ******* check that the problem is feasible ********************

4. IF (DELPRC.EQ.LARGE) THEN
        FEASBL=.FALSE.
        RETURN
      END IF

C      ***** INCREASE PRICES *****

      IF (SWITCH) THEN
        AUGNOD=0
        DO 7 I=1,NSAVE
          ARC=SAVE(I)
          IF (ARC.GT.0) THEN
            RC(ARC)=RC(ARC)+DELPRC
            IF (RC(ARC).EQ.0) THEN
              NODE2=STARTN(ARC)
              IF (TNXTOU(ARC) .LT. 0) THEN
                TNXTOU(ARC) = TFSTOU(NODE2)
                TFSTOU(NODE2) = ARC
              END IF
              IF (TNXTIN(ARC) .LT. 0) THEN
                TNXTIN(ARC) = TFSTIN(ENDN(ARC))
                TFSTIN(ENDN(ARC)) = ARC
              END IF
              PRDCSR(NODE2)=ARC
              IF (DFCT(NODE2).GT.0) THEN
                AUGNOD=NODE2
              ELSE
               IF (.NOT.MARK(NODE2)) THEN
                 MARK(NODE2)=.TRUE.
```

```
              NLABEL=NLABEL+1
              LABEL(NLABEL)=NODE2
             END IF
            END IF
          END IF
        ELSE
          ARC=-ARC
          RC(ARC)=RC(ARC)-DELPRC
          IF (RC(ARC).EQ.0) THEN
            NODE2=ENDN(ARC)
            IF (TNXTOU(ARC) .LT. 0) THEN
              TNXTOU(ARC) = TFSTOU(STARTN(ARC))
              TFSTOU(STARTN(ARC)) = ARC
            END IF
            IF (TNXTIN(ARC) .LT. 0) THEN
              TNXTIN(ARC) = TFSTIN(NODE2)
              TFSTIN(NODE2) = ARC
            END IF
            PRDCSR(NODE2)=-ARC
            IF (DFCT(NODE2).GT.0) THEN
              AUGNOD=NODE2
            ELSE
             IF (.NOT.MARK(NODE2)) THEN
              MARK(NODE2)=.TRUE.
              NLABEL=NLABEL+1
              LABEL(NLABEL)=NODE2
             END IF
            END IF
          END IF
        END IF
7       CONTINUE
        RETURN

      ELSE

      NB = 0
      DO 6 I=1,NSAVE
        ARC=SAVE(I)
        IF (ARC.GT.0) THEN
            T1=RC(ARC)
            IF (T1.EQ.0) THEN
             T2=X(ARC)
             T3=STARTN(ARC)
             DFCT(T3)=DFCT(T3)-T2
             IF (NXTQUEUE(T3).EQ.0) THEN
                NXTQUEUE(PREVNODE)=T3
                NXTQUEUE(T3)=STARTNODE
                PREVNODE=T3
             END IF

             T3=ENDN(ARC)
             DFCT(T3)=DFCT(T3)+T2
             IF (NXTQUEUE(T3).EQ.0) THEN
                NXTQUEUE(PREVNODE)=T3
```

```
                NXTQUEUE(T3)=STARTNODE
                PREVNODE=T3
              END IF

              U(ARC)=U(ARC)+T2
              X(ARC)=0
            END IF
          RC(ARC)=T1+DELPRC
          IF (RC(ARC).EQ.0) THEN
             DELX=DELX+X(ARC)
             NB = NB + 1
             PRDCSR(NB) = ARC
          END IF
        ELSE
          ARC=-ARC
          T1=RC(ARC)
            IF (T1.EQ.0) THEN
              T2=U(ARC)
              T3=STARTN(ARC)
              DFCT(T3)=DFCT(T3)+T2
              IF (NXTQUEUE(T3).EQ.0) THEN
                NXTQUEUE(PREVNODE)=T3
                NXTQUEUE(T3)=STARTNODE
                PREVNODE=T3
              END IF

              T3=ENDN(ARC)
              DFCT(T3)=DFCT(T3)-T2
              IF (NXTQUEUE(T3).EQ.0) THEN
                NXTQUEUE(PREVNODE)=T3
                NXTQUEUE(T3)=STARTNODE
                PREVNODE=T3
              END IF

              X(ARC)=X(ARC)+T2
              U(ARC)=0
            END IF
          RC(ARC)=T1-DELPRC
          IF (RC(ARC).EQ.0) THEN
             DELX=DELX+U(ARC)
             NB = NB + 1
             PRDCSR(NB) = ARC
        END IF
      END IF
6     CONTINUE

      END IF
      IF (DELX.LE.-DM) THEN
        DELPRC=LARGE
        DO 10 I=1,NSAVE
          ARC=SAVE(I)
          IF (ARC.GT.0) THEN
            RDCOST=RC(ARC)
            IF ((RDCOST.LT.0).AND.(RDCOST.GT.-DELPRC)) DELPRC=-RDCOST
```

```
          ELSE
            ARC=-ARC
            RDCOST=RC(ARC)
            IF ((RDCOST.GT.0).AND.(RDCOST.LT.DELPRC)) DELPRC=RDCOST
          END IF
10      CONTINUE
        IF ((DELPRC.NE.LARGE).OR.(DELX.LT.-DM)) GO TO 4
      END IF

C     Add new balance arcs to the superset of balanced arcs.

      DO 9 I=1,NB
        ARC=PRDCSR(I)
          IF (TNXTIN(ARC).EQ.-1) THEN
            J=ENDN(ARC)
            TNXTIN(ARC)=TFSTIN(J)
            TFSTIN(J)=ARC
          END IF
          IF (TNXTOU(ARC).EQ.-1) THEN
            J=STARTN(ARC)
            TNXTOU(ARC)=TFSTOU(J)
            TFSTOU(J)=ARC
          END IF
9     CONTINUE

      RETURN
      END
```

```
*************************************************************************
*

Auction Codes for Assignment Problems

*************************************************************************
*

Since assignment
problems cannot be generated by the GRIDGEN program, the assignment codes
include a built-in random problem generator. This generator can be
replaced by
other code that reads an assignment problem from a file.

AUCTION

This code implements the forward auction
algorithm with $\e$-scaling for the symmetric assignment problem; cf.\
Section 4.1.

C ***************************************************************
C
C       SAMPLE CALLING PROGRAM FOR THE AUCTION ALGORITHM
```

```
C
C   THIS DRIVER CREATES A SYMMETRIC ASSIGNMENT PROBLEM
C   WITH EQUAL NUMBER OF ROWS AND COLUMNS,
C   AND CALLS THE AUCTION SUBROUTINE TO FIND AN
C   ASSIGNMENT OF MAXIMAL VALUE.
C
C ****************************************************************

        PARAMETER(MAXNODES=10000, MAXARCS=100000)

        IMPLICIT NONE
        INTEGER N,NA,A,ISMALL,BEGEPS,ENDEPS,CYCLES
        INTEGER NUMPHASES,STARTINCR,TEST,PROFIT
        INTEGER I,J,IA,ARC,NOASS,ICOST,ABSCOST,CURARC
        INTEGER CURCOL,FSTARC,LSTARC,MINCOST,MAXCOST,MCOST
        INTEGER FOUT(MAXNODES),COLASS(MAXNODES)
        INTEGER ASSIGN(MAXNODES),PCOL(MAXNODES)
        INTEGER COST(MAXARCS),END(MAXARCS)
        REAL*8 FACTOR,TT1,TT2,TCOST,AVERAGE
        COMMON/ARRAYC/COST/ARRAYS/END/ARRAYF/FOUT/BK1/N,A,ISMALL
     $  /BK2/CYCLES,AVERAGE,NUMPHASES/ARRAYA/ASSIGN/PCOLA/PCOL

C ****************************************************************
C
C     PROBLEM GENERATION CODE STARTS HERE
C     THE USER MAY REPLACE THIS CODE WITH A CODE THAT READS
C     HIS/HER PROBLEM FROM A FILE
C
C ****************************************************************

C     THIS CODE INCLUDES A UNIFORM RANDOM NUMBER GENERATOR
C     WHICH RETURNS A VALUE IN (0,1)

C     INITIALIZE RANDOM GENERATOR

      INTEGER MULT,MODUL,I15,I16,JRAN,ISEED
      REAL RAN
      COMMON /RANDM/ MULT,MODUL,I15,I16,JRAN

      ISEED=13502460
      CALL SETRAN(ISEED)

      PRINT*,'GENERATING A SYMMETRIC ASSIGNMENT PROBLEM'
      PRINT*,'****************************************'

C **** READ THE NUMBER OF ROWS N & THE NUMBER OF ARCS A ****

      PRINT*,'ENTER THE NUMBER OF ROWS (AND COLUMNS)'
      READ*,N
5.    PRINT*,'ENTER THE NUMBER OF ARCS PER ROW (>1)'
      READ*,NA
      IF (NA.LT.2) GOTO 5
      PRINT*,'ENTER THE MINIMUM AND THE MAXIMUM COST'
      READ*,MINCOST,MAXCOST
```

```
C     THE NUMBER OF ARCS IS N*NA

      A=N*NA

C      THE ARCS INCIDENT TO ROW I ARE FOUT(I) TO FOUT(I+1)-1
C      ALSO, FOR FEASIBILITY EACH ROW IS DIRECTLY CONNECTED
C      WITH THE CORRESPONDING COLUMN

      DO 20 I=1,N
        FOUT(I)=1+(I-1)*NA
20    CONTINUE
      FOUT(N+1)=A+1

C      GENERATE THE END(ARC) AND COST(ARC) WHICH ARE THE COLUMN
C      AND THE COST COEFFICIENT ASSOCIATED WITH ARC

      DO 25 ARC=1,A
        END(ARC)=1+RAN()*N
        IF ((END(ARC).GT.N).OR.(END(ARC).LT.1)) THEN
          PRINT*,'ERROR IN PROBLEM GENERATION'
          PAUSE
          STOP
        END IF
        COST(ARC)=MINCOST+RAN()*(MAXCOST-MINCOST)
25    CONTINUE

C   MODIFY THE END OF THE LAST ARC OUT OF EACH ROW FOR FEASIBILITY
C   AND SET ITS COST TO -MAXCOST

      DO 30 I=1,N
        END(FOUT(I+1)-1)=I
        COST(FOUT(I+1)-1)=-MAXCOST
30    CONTINUE


C
C ***************************************************************
C
C          PROBLEM GENERATION CODE ENDS HERE
C
C ***************************************************************

C       SCALE THE COST TO WORK WITH INTEGER EPSILON

        MAXCOST=0
        DO 35 IA=1,A
           ABSCOST=IABS(COST(IA))
           IF (ABSCOST.GT.MAXCOST) MAXCOST=ABSCOST
           COST(IA)=COST(IA)*(N+1)
35      CONTINUE

C *** ISMALL IS A VERY SMALL INTEGER FOR YOUR MACHINE ***
```

```
      ISMALL=-2000000000

      IF (MAXCOST.GT.INT(ABS(ISMALL)/(N+1))) THEN
        PRINT*,'THE COST RANGE IS TOO LARGE FOR INTEGER ARITHMETIC'
        PAUSE
        STOP
      END IF

      MAXCOST=MAXCOST*(N+1)


C     THE FOLLOWING PARAMETERS BEGEPS, FACTOR, ENDEPS, AND STARTINCR
C     ARE PASSED TO THE AUCTION ALGORITHM. VALUES BETWEEN
C      (A) MAXCOST/5 AND MAXCOST/2 FOR BEGEPS
C      (B) 4 AND 6 FOR FACTOR
C      (C) N/10 AND 1 FOR ENDEPS
C      (D) 1 AND BEGEPS FOR STARTINCR
C     HAVE WORKED WELL FOR LARGE SPARSE PROBLEMS.
C     FOR DENSE PROBLEMS IT IS RECOMMENDED THAT
C     BEGEPS BE SET TO A SMALLER VALUE,
C     ENDEPS BE SET TO 1,
C     STARTINCR BE SET TO 1.

      PRINT*,'*****************************************'
      PRINT*,'MAXIMUM COST IS ',MAXCOST
      PRINT*,'ENTER THE STARTING EPSILON'
      READ*,BEGEPS
      IF (BEGEPS.LT.1) BEGEPS=1
      PRINT*,'ENTER THE EPSILON REDUCTION FACTOR'
      READ*,FACTOR
      ENDEPS=N/10
      IF (ENDEPS.LT.1) ENDEPS=1
      IF (ENDEPS.GT.BEGEPS) ENDEPS=BEGEPS
      STARTINCR=BEGEPS/10
      IF (STARTINCR.LT.1) STARTINCR=1

      PRINT*,'*****************************************'
      PRINT*,'STARTING EPSILON = ',BEGEPS
      PRINT*,'EPSILON REDUCTION FACTOR = ',FACTOR
      PRINT*,'THRESHOLD EPSILON BEFORE IT IS SET TO 1 = ',ENDEPS
      PRINT*,'STARTING MIN BIDDING INCREMENT = ',STARTINCR
      PRINT*,'CALLING AUCTION TO SOLVE THE PROBLEM'
      PRINT*,'*****************************************'

C     GET STARTING TIME FOR THE MAC II

      TT1 = LONG(362)/60.0

      CALL AUCTION(BEGEPS,FACTOR,ENDEPS,STARTINCR)

C     GET ENDING TIME FOR THE MAC II

      TT2 = LONG(362)/60.0 - TT1
      PRINT *,'FINISHED --- TOTAL CPU TIME', TT2,' SECS'
```

```
      PRINT*,'*****************************************'

C     *** DISPLAY RESULTS ***

X        WRITE(9,2010) CYCLES
X2010    FORMAT(' NO OF AUCTION CYCLES',I7)
X        WRITE(9,2020) AVERAGE
X2020    FORMAT(' AVERAGE NUMBER OF BIDS PER CYCLE',F9.3)
        WRITE(9,2030) NUMPHASES
2030    FORMAT('NO OF EPSILON SUBPROBLEMS SOLVED =',I7)

C       CHECK OPTIMALITY OF SOLUTION & CALCULATE COST

        DO 40 I=1,N
          COLASS(I)=0
40      CONTINUE
        NOASS=0
        DO 50 J=1,N
           IF (ASSIGN(J).GT.0) THEN
             NOASS=NOASS+1
             COLASS(ASSIGN(J))=J
           END IF
50      CONTINUE
        IF (NOASS.NE.N) THEN
          PRINT*,'# OF ASSIGNED ROWS NOT EQUAL TO # OF ROWS'
        END IF
        TCOST=0
        DO 60 I=1,N
          J=COLASS(I)
          IF (J.EQ.0) THEN
            PRINT*,'ROW ',I,' IS UNASSIGNED'
          END IF
          FSTARC=FOUT(I)
          LSTARC=FOUT(I+1)-1
          MCOST=ISMALL
          DO 70 CURARC=FSTARC,LSTARC
            CURCOL=END(CURARC)
            IF (CURCOL.EQ.J) THEN
              IF (MCOST.LT.COST(CURARC)) THEN
                 MCOST=COST(CURARC)
              END IF
            END IF
70        CONTINUE
          PROFIT = MCOST-PCOL(J)
          DO 72 CURARC=FSTARC,LSTARC
             J = END(CURARC)
             TEST = COST(CURARC)-PCOL(J)-PROFIT
             IF (TEST.GT.1) THEN
                PRINT *, '1-CS VIOLATED AT ARC',CURARC
             END IF
72        CONTINUE
          TCOST=TCOST+MCOST/(N+1)
60      CONTINUE
        WRITE(9,2100) TCOST
```

```
2100    FORMAT(' ASSIGNMENT COST=',F14.2)
        PRINT *, ' PROGRAM ENDED; <CR> TO EXIT '
        PAUSE
        END



C ***************************************************************
C
C        AUCTION CODE FOR N BY N ASSIGNMENT PROBLEMS
C
C             WRITTEN BY DIMITRI P. BERTSEKAS
C              (MODIFICATIONS BY PAUL TSENG)
C
C                  VERSION 1.1, SEPT. 1990
C
C THIS CODE IMPLEMENTS THE AUCTION ALGORITHM WITH E-SCALING.
C IT SOLVES A SEQUENCE OF SUBPROBLEMS AND DECREASES
C EPSILON BY A CONSTANT FACTOR BETWEEN SUBPROBLEMS.
C THIS VERSION CORRESPONDS TO A GAUSS-SEIDEL MODE
C AND SOLVES EPSILON SUBPROBLEMS INEXACTLY.
C
C THE CODE IS AN IMPROVED VERSION OF AN EARLIER (SEPT. 1985)
C AUCTION CODE WITH E-SCALING WRITTEN BY DIMITRI P. BERTSEKAS
C
C THE CODE TREATS THE PROBLEM AS A MAXIMIZATION PROBLEM.
C TO SOLVE A MINIMIZATION PROBLEM, REVERSE THE SIGN OF THE
C ARC COSTS PRIOR TO CALLING AUCTION, AND REVERSE AGAIN
C THE SIGN OF THE OPTIMAL COST UPON RETURN FROM AUCTION.
C THIS CODE ALLOWS MULTIPLE ARCS BETWEEN A ROW AND A COLUMN.
C
C THIS VERSION OF THE AUCTION ALGORITHM IS ADAPTIVE. HERE THE MINIMAL
C BIDDING INCREMENT (STORED IN THE VARIABLE INCR) MAY BE SMALLER THAN
C EPSILON. FOR EVERY SUBPROBLEM, INCR STARTS AT THE PARAMETER VALUE
C STARTINCR (WHICH IS PASSED TO THE AUCTION ROUTINE) AND IS INCREASED
C BY A FACTOR OF 2 AT THE END OF EACH CYCLE, UP TO A MAXIMUM VALUE OF
C EPSILON. THIS ADAPTIVE FEATURE IS PARTICULARLY EFFECTIVE FOR
C DENSE PROBLEMS. IT CAN BE DEFEATED BY SELECTING STARTINCR=BEGEPS
C
C ***************************************************************
C
C THE USER MUST SUPPLY THE FOLLOWING PROBLEM DATA IN FORWARD STAR FORMAT
C (THAT IS, ALL ARCS OF THE SAME ROW ARE NUMBERED CONSECUTIVELY):
C       N=NUMBER OF ROWS (EQUALS NUMBER OF COLUMNS)
C       A=NUMBER OF ARCS
C       FOUT(ROW)=FIRST ARC COMING OUT OF ROW
C       COST(ARC)=COST OF ARC
C       END(ARC)=COLUMN CORRESPONDING TO ARC
C
C AND THE FOLLOWING PARAMETERS FOR THE AUCTION ALGORITHM:
C       BEGEPS=STARTING VALUE OF EPSILON (MUST BE NO LESS THAN 1)
C       ENDEPS=FINAL VALUE OF EPSILON BEFORE IT IS SET TO 1
C       FACTOR=FACTOR BY WHICH EPSILON IS DECREASED BETWEEN SUBPROBLEMS
C       STARTINCR=THE STARTING VALUE OF THE BIDDING INCREMENT
```

```
C       ENDEPS SHOULD NOT EXCEED BEGEPS.
C       FACTOR MUST BE GREATER THAN 1.
C
C FOUT(.) IS AN ARRAY OF LENGTH N.
C COST(.),END(.) ARE ARRAYS OF LENGTH A.
C
C THE SOLUTION IS CONTAINED IN THE ARRAY ASSIGN(.) WHERE
C    ASSIGN(COL) GIVES THE ROW ASSIGNED TO COL.
C
C THIS ALGORITHM DOES NOT CHECK FOR INFEASIBILITY OF THE PROBLEM.
C TO MAKE SURE THE PROBLEM IS FEASIBLE THE USER MAY ADD
C    ADDITIONAL VERY SMALL COST ARCS.
C
C THIS CODE MAY FAIL DUE TO INTEGER OVERFLOW IF THE NUMBER OF NODES
C OR THE COST RANGE (OR BOTH) ARE LARGE. TO CORRECT THIS SITUATION,
C THE PRICES AND OTHER RELATED VARIABLES (MAX1,MAX2,TMAX ETC) SHOULD
C BE DECLARED AS DOUBLE PRECISION REALS.
C ***************************************************************
C ALL PROBLEM DATA ARE INTEGER
C ***************************************************************


        SUBROUTINE AUCTION(BEGEPS,FACTOR,ENDEPS,STARTINCR)

        PARAMETER(MAXNODES=10000, MAXARCS=100000)

        IMPLICIT NONE
        INTEGER NONEWLIST,THRESH,INCR,STARTINCR,INCRFACTOR
        INTEGER A,K,N,I,J,CURARC,CURCOL
        INTEGER ROW,FSTARC,FSTCOL,SNDARC,SNDCOL,TMAX,TMIN,BSTCOL
        INTEGER MAX1,MAX2,TRDARC,EPSILON,BEGEPS,ENDEPS
        INTEGER M,ISMALL,ILARGE,LARGEINCR,CYCLES
        INTEGER NUMPHASES,LSTARC,NOLIST,OLDROW
        INTEGER FOUT(MAXNODES),PCOL(MAXNODES),LIST(MAXNODES)
               INTEGER ASSIGN(MAXNODES)
        INTEGER COST(MAXARCS),END(MAXARCS)
        REAL*8 AVERAGE,FACTOR
               COMMON/ARRAYC/COST/ARRAYS/END/ARRAYF/FOUT/BK1/N,A,ISMALL
     $  /BK2/CYCLES,AVERAGE,NUMPHASES/ARRAYA/ASSIGN/PCOLA/PCOL

C ***************************************************************

C   ******* CHECK VALIDITY OF PARAMETERS PASSED *******

        IF (BEGEPS.LT.1) THEN
          PRINT*,'STARTING VALUE OF EPSILON IS LESS THAN 1'
          PRINT*,'EXECUTION ABORTED'
          STOP
        END IF
        IF (ENDEPS.GT.BEGEPS) THEN
          PRINT*,'PARAMETER ENDEPS IS GREATER THAN PARAMETER BEGEPS'
          PRINT*,'ENDEPS IS SET AT THE DEFAULT VALUE OF 1'
          ENDEPS=1
        END IF
```

```
        IF ((FACTOR.LE.1).AND.(BEGEPS.GT.1)) THEN
          PRINT*,'EPSILON REDUCTION FACTOR IS NOT GREATER THAN 1'
          PRINT*,'EXECUTION ABORTED'
          STOP
        END IF
        IF (STARTINCR.LT.1) THEN
          PRINT*,'MIN BIDDING INCREMENT IS LESS THAN 1'
          PRINT*,'STARTINCR IS SET AT THE DEFAULT VALUE OF 1'
          STARTINCR=1
        END IF

C   ******* INITIALIZATION *******

        EPSILON=BEGEPS
        ILARGE=-ISMALL
        LARGEINCR=INT(ILARGE/10)
        THRESH=INT(0.2*N)
        INCRFACTOR=2
        IF (THRESH.GT.100) THRESH=100
X        CYCLES=1
X        AVERAGE=N
         NUMPHASES=1
         DO 10 J=1,N
           ASSIGN(J)=0
           PCOL(J)=ISMALL
10      CONTINUE

        FOUT(N+1)=A+1
        NOLIST=N
        DO 20 I=1,N
            LIST(I)=I
20      CONTINUE

C ***************************************************************
C
C  THIS IMPLEMENTATION OF THE AUCTION ALGORITHM OPERATES IN CYCLES.
C  EACH CYCLE CONSISTS OF ONE BID BY EACH OF THE ROWS THAT ARE
C  UNASSIGNED AT THE START OF THE CYCLE (THESE ROWS ARE STORED IN
C  THE ARRAY LIST(.)). AS THE CYCLE PROGRESSES NEW
C  ROWS BECOME UNASSIGNED; THESE ARE STORED IN LIST(.)
C  AND WILL SUBMIT A BID AT THE NEXT CYCLE.
C  NOTE THAT ONLY ONE ROW SUBMITS A BID AT A TIME; THIS IS KNOWN AS
C  THE GAUSS-SEIDEL VERSION OF THE ALGORITHM. THE VERSION WHERE ALL
C  UNASSIGNED ROWS SUBMIT A BID AT THE SAME TIME IS KNOWN AS THE JACOBI
C  VERSION OF THE ALGORITHM, AND HAS NOT BEEN IMPLEMENTED. IT GENERALLY
C  TENDS TO RUN SOMEWHAT SLOWER THAN THE GAUSS-SEIDEL VERSION, BUT
C  IT ADMITS A HIGHER DEGREE OF PARALLELIZATION. A JACOBI VERSION
C  MAY BE PREFERABLE ON A PARALLEL MACHINE, BUT IS USUALLY INFERIOR
C  TO THE GAUSS-SEIDEL VERSION ON A SERIAL MACHINE.
C
C ***************************************************************
C
C       START SUBPROBLEM (SCALING PHASE) W/ NEW EPSILON
C
```

```
C ***************************************************************

12      CONTINUE

        IF (EPSILON.EQ.1) THRESH=0
        INCR=STARTINCR
        IF (INCR.GT.EPSILON) INCR=EPSILON

C ***************************************************************
C
C       START AUCTION CYCLE WITH NEW LIST
C
C ***************************************************************

15      CONTINUE

C  INITIALIZE COUNT OF NEXT LIST OF UNASSIGNED ROWS

        NONEWLIST=0

C  CYCLE THROUGH THE CURRENT LIST OF UNASSIGNED ROWS

        DO 100 I=1,NOLIST
        ROW=LIST(I)
        FSTARC=FOUT(ROW)
        LSTARC=FOUT(ROW+1)-1
        FSTCOL=END(FSTARC)

C  FIRST TAKE CARE OF THE EXCEPTIONAL CASE WHERE ROW HAS ONLY ONE ARC

        IF (FSTARC.EQ.LSTARC) THEN
          PCOL(FSTCOL)=PCOL(FSTCOL)+LARGEINCR
            OLDROW=ASSIGN(FSTCOL)
            ASSIGN(FSTCOL)=ROW
            IF (OLDROW.GT.0) THEN
              NONEWLIST=NONEWLIST+1
              LIST(NONEWLIST)=OLDROW
            END IF
          GO TO 100
        END IF

C  NEXT TAKE CARE OF THE REGULAR CASE WHERE ROW HAS MULTIPLE ARCS

        SNDARC=FSTARC+1
        SNDCOL=END(SNDARC)
        MAX1=COST(FSTARC)-PCOL(FSTCOL)
        MAX2=COST(SNDARC)-PCOL(SNDCOL)
        IF (MAX1.GE.MAX2) THEN
           BSTCOL=FSTCOL
        ELSE
           TMAX=MAX1
           MAX1=MAX2
           MAX2=TMAX
           BSTCOL=SNDCOL
```

```
        END IF
        IF (SNDARC.LT.LSTARC) THEN
        TRDARC=SNDARC+1
        DO 40 CURARC=TRDARC,LSTARC
           CURCOL=END(CURARC)
           TMAX=COST(CURARC)-PCOL(CURCOL)
           IF (TMAX.GT.MAX2) THEN
             IF (TMAX.GT.MAX1) THEN
                  MAX2=MAX1
                  MAX1=TMAX
                  BSTCOL=CURCOL
             ELSE
                  MAX2=TMAX
             END IF
           END IF
40      CONTINUE
        END IF


C  ROW BIDS FOR BSTCOL INCREASING ITS PRICE, AND GETS ASSIGNED
C  TO BSTCOL, WHILE ANY ROW ASSIGNED TO BSTCOL BECOMES UNASSIGNED

         PCOL(BSTCOL)=PCOL(BSTCOL)+MAX1-MAX2+INCR
         OLDROW=ASSIGN(BSTCOL)
         ASSIGN(BSTCOL)=ROW
         IF (OLDROW.GT.0) THEN
           NONEWLIST=NONEWLIST+1
           LIST(NONEWLIST)=OLDROW
         END IF

100    CONTINUE

C      ************* END OF AN AUCTION CYCLE ***************

C  OPTIONALLY COLLECT STATISTICS

X        AVERAGE=(CYCLES*AVERAGE+NOLIST)/(CYCLES+1)
X        CYCLES=CYCLES+1

C  CHECK IF THERE ARE STILL 'MANY' UNASSIGNED ROWS, THAT IS, IF THE
C  NUMBER OF UNASSIGNED ROWS IS GREATER THAN
C  THE PARAMETER THRESH. IF NOT, REPLACE CURRENT LIST WITH THE NEW LIST,
C  AND GO FOR ANOTHER CYCLE. OTHERWISE, IF EPSILON > 1, REDUCE EPSILON,
C  RESET THE ASSIGNMENT TO EMPTY AND RESTART AUCTION;
C  IF EPSILON = 1 TERMINATE.
C  ALSO INCREASE THE MINIMAL BIDDING INCREMENT UP TO A MAXIMUM
C  VALUE OF EPSILON (THIS IS THE ADAPTIVE FEATURE)

        INCR=INCR*INCRFACTOR
        IF (INCR.GT.EPSILON) INCR=EPSILON
        IF (NONEWLIST.GT.THRESH) THEN
            NOLIST=NONEWLIST
            GO TO 15
        END IF
```

```
C ************************************************************
C
C               END OF SUBPROBLEM (SCALING PHASE)
C
C ************************************************************

C ****** IF EPSILON IS 1 TERMINATE ******

       IF (EPSILON.EQ.1) THEN
          RETURN
       ELSE

C   ELSE REDUCE EPSILON AND RESET THE ASSIGNMENT TO EMPTY

          NUMPHASES=NUMPHASES+1
          EPSILON=INT(EPSILON/FACTOR)
          IF (EPSILON.GT.INCR) EPSILON=INT(EPSILON/FACTOR)
          IF ((EPSILON.LT.1).OR.(EPSILON.LT.ENDEPS)) EPSILON=1
          THRESH=INT(THRESH/FACTOR)

X         PRINT*,'*** END OF A SCALING PHASE; NEW EPSILON=',EPSILON

          TMIN=ILARGE
          DO 200 J=1,N
           IF (PCOL(J).LT.TMIN) TMIN=PCOL(J)
           IF (ASSIGN(J).GT.0) THEN
            NONEWLIST=NONEWLIST+1
            LIST(NONEWLIST)=ASSIGN(J)
            ASSIGN(J)=0
           END IF
200       CONTINUE

C     RESET MINIMUM PRICE TO ISMALL

          INCR=TMIN-ISMALL
          DO 210 J=1,N
            PCOL(J)=PCOL(J)-INCR
210       CONTINUE


C     FINAL PARAMETER UPDATES BEFORE RETURNING FOR ANOTHER SCALING PHASE

          NOLIST=NONEWLIST
          IF (STARTINCR.LT.EPSILON) STARTINCR=FACTOR*STARTINCR
          GO TO 12
        END IF


        END



      SUBROUTINE SETRAN(ISEED)
```

```
      IMPLICIT REAL*8 (A-H,O-Z) , INTEGER*4 (I-N)
C*********************************************************************
C PORTABLE CONGRUENTIAL (UNIFORM) RANDOM NUMBER GENERATOR:
C     NEXT_VALUE = [(7**5) * PREVIOUS_VALUE] MODULO[(2**31)-1]
C
C THIS GENERATOR CONSISTS OF TWO ROUTINES:
C   (1) SETRAN - INITIALIZES CONSTANTS AND SEED
C   (2) RRAN   - GENERATES A REAL RANDOM NUMBER
C
C THE GENERATOR REQUIRES A MACHINE WITH AT LEAST 32 BITS OF PRECISION.
C THE SEED (ISEED) MUST BE IN THE RANGE (1,(2**31)-1).
C*********************************************************************
      COMMON /RANDM/ MULT,MODUL,I15,I16,JRAN
      IF(ISEED.LT.1) STOP 77
      MULT=16807
      MODUL=2147483647
      I15=2**15
      I16=2**16
      JRAN=ISEED
      RETURN
      END

      REAL FUNCTION RAN()
      IMPLICIT REAL*4 (A-H,O-Z) , INTEGER*4 (I-N)
C*********************************************************************
C RAN GENERATES A REAL RANDOM NUMBER BETWEEN 0 AND 1
C*********************************************************************
      COMMON /RANDM/ MULT,MODUL,I15,I16,JRAN
      IXHI=JRAN/I16
      IXLO=JRAN-IXHI*I16
      IXALO=IXLO*MULT
      LEFTLO=IXALO/I16
      IXAHI=IXHI*MULT
      IFULHI=IXAHI+LEFTLO
      IRTLO=IXALO-LEFTLO*I16
      IOVER=IFULHI/I15
      IRTHI=IFULHI-IOVER*I15
      JRAN=((IRTLO-MODUL)+IRTHI*I16)+IOVER
      IF(JRAN.LT.0) JRAN=JRAN+MODUL
      RAN = FLOAT(JRAN)/FLOAT(MODUL)
      RETURN
      END
```

AUCTION_FLP

This code is the same as the preceding one except that
it uses floating point arithmetic when updating prices. This is useful
when
the cost range and the dimension of the problem are large, in which case
the prices may overflow the integer range in the course of the algorithm.
However, the floating point arithmetic slows down the code somewhat.

```
C ***************************************************************
C
C       SAMPLE CALLING PROGRAM FOR THE AUCTION ALGORITHM
C
C   THIS DRIVER CREATES A SYMMETRIC ASSIGNMENT PROBLEM
C   WITH EQUAL NUMBER OF ROWS AND COLUMNS,
C   AND CALLS THE AUCTION SUBROUTINE TO FIND AN
C   ASSIGNMENT OF MAXIMAL VALUE.
C
C   MODIFIED BY DAVID CASTANON (OCT. 1991) TO WORK WITH ARBITRARILY
C   LARGE COST RANGE (UP TO THE INTEGER RANGE OF THE MACHINE);
C   TO ACCOMPLISH THIS, THE OBJECT PRICES AND EPSILON ARE
C   DOUBLE PRECISION VARIABLES, SO THE PRICES CANNOT OVERFLOW THE
C   MACHINE'S INTEGER RANGE.
C
C ***************************************************************

        PARAMETER(MAXNODES=10000, MAXARCS=100000)

        IMPLICIT NONE
        INTEGER N,NA,A,ISMALL,CYCLES
        INTEGER NUMPHASES
        INTEGER I,J,IA,ARC,NOASS,ICOST,ABSCOST,CURARC
        INTEGER CURCOL,FSTARC,LSTARC,MINCOST,MAXCOST,MCOST
        INTEGER FOUT(MAXNODES),COLASS(MAXNODES)
        INTEGER ASSIGN(MAXNODES)
        INTEGER COST(MAXARCS),END(MAXARCS)
        REAL*8 TT1,TT2,TCOST
        REAL*8 PCOL(MAXNODES)
        REAL*8 AVERAGE
        REAL*8 BEGEPS,ENDEPS,FACTOR,STARTINCR,TEST,PROFIT
        COMMON/ARRAYC/COST/ARRAYS/END/ARRAYF/FOUT/BK1/N,A,ISMALL
     $  /BK2/CYCLES,AVERAGE,NUMPHASES/ARRAYA/ASSIGN/PCOLA/PCOL

C ***************************************************************
C
C     PROBLEM GENERATION CODE STARTS HERE
C     THE USER MAY REPLACE THIS CODE WITH A CODE THAT READS
C     HIS/HER PROBLEM FROM A FILE
C
C ***************************************************************

C     THIS CODE INCLUDES A UNIFORM RANDOM NUMBER GENERATOR
C     WHICH RETURNS A VALUE IN (0,1)

C     INITIALIZE RANDOM GENERATOR

      INTEGER MULT,MODUL,I15,I16,JRAN,ISEED
      REAL RAN
      COMMON /RANDM/ MULT,MODUL,I15,I16,JRAN

      ISEED=13502460
      CALL SETRAN(ISEED)
```

```
      PRINT*,'GENERATING A SYMMETRIC ASSIGNMENT PROBLEM'
      PRINT*,'****************************************'

C **** READ THE NUMBER OF ROWS N & THE NUMBER OF ARCS A ****

      PRINT*,'ENTER THE NUMBER OF ROWS (AND COLUMNS)'
      READ*,N
6. PRINT*,'ENTER THE NUMBER OF ARCS PER ROW (>1)'
      READ*,NA
      IF (NA.LT.2) GOTO 5
      PRINT*,'ENTER THE MINIMUM AND THE MAXIMUM COST'
      READ*,MINCOST,MAXCOST

C     THE NUMBER OF ARCS IS N*NA

      A=N*NA

C      THE ARCS INCIDENT TO ROW I ARE FOUT(I) TO FOUT(I+1)-1
C      ALSO, FOR FEASIBILITY EACH ROW IS DIRECTLY CONNECTED
C      WITH THE CORRESPONDING COLUMN

      DO 20 I=1,N
        FOUT(I)=1+(I-1)*NA
20    CONTINUE
      FOUT(N+1)=A+1

C      GENERATE THE END(ARC) AND COST(ARC) WHICH ARE THE COLUMN
C      AND THE COST COEFFICIENT ASSOCIATED WITH ARC

      DO 25 ARC=1,A
        END(ARC)=1+RAN()*N
        IF ((END(ARC).GT.N).OR.(END(ARC).LT.1)) THEN
          PRINT*,'ERROR IN PROBLEM GENERATION'
          PAUSE
          STOP
        END IF
        COST(ARC)=MINCOST+RAN()*(MAXCOST-MINCOST)
25    CONTINUE

C   MODIFY THE END OF THE LAST ARC OUT OF EACH ROW FOR FEASIBILITY
C   AND SET ITS COST TO -MAXCOST

      DO 30 I=1,N
        END(FOUT(I+1)-1)=I
        COST(FOUT(I+1)-1)=-MAXCOST
30    CONTINUE


C
C ************************************************************
C
C          PROBLEM GENERATION CODE ENDS HERE
C
```

```
C ************************************************************

C       SCALE THE COST TO WORK WITH INTEGER EPSILON

        MAXCOST=0
        DO 35 IA=1,A
           ABSCOST=IABS(COST(IA))
           IF (ABSCOST.GT.MAXCOST) MAXCOST=ABSCOST
35      CONTINUE

C *** ISMALL IS A VERY SMALL INTEGER FOR YOUR MACHINE ***

        ISMALL=-2000000000

        IF (MAXCOST.GT.ABS(ISMALL)) THEN
          PRINT*,'THE COST RANGE IS TOO LARGE FOR INTEGER ARITHMETIC'
          PAUSE
          STOP
        END IF


C       THE FOLLOWING PARAMETERS BEGEPS, FACTOR, ENDEPS, AND STARTINCR
C       ARE PASSED TO THE AUCTION ALGORITHM. VALUES BETWEEN
C        (A) MAXCOST/5 AND MAXCOST/2 FOR BEGEPS
C        (B) 4 AND 6 FOR FACTOR
C        (C) 1/10 AND 1/(N+1) FOR ENDEPS
C        (D) 1 AND BEGEPS FOR STARTINCR
C       HAVE WORKED WELL FOR LARGE SPARSE PROBLEMS.
C       FOR DENSE PROBLEMS IT IS RECOMMENDED THAT
C       BEGEPS BE SET TO A SMALLER VALUE,
C       ENDEPS BE SET TO 1/(N+1),
C       STARTINCR BE SET TO 1/(N+1).

        PRINT*,'*****************************************'
        PRINT*,'MAXIMUM COST IS ',MAXCOST
        PRINT*,'ENTER THE STARTING EPSILON'
        READ*,BEGEPS
        IF (BEGEPS.LT.1.0/(N+1)) BEGEPS=1.0/(N+1)
        PRINT*,'ENTER THE EPSILON REDUCTION FACTOR'
        READ*,FACTOR
        ENDEPS=1.0/10.0
        IF (ENDEPS.LT.1.0/(N+1)) ENDEPS=1.0/(N+1)
        IF (ENDEPS.GT.BEGEPS) ENDEPS=BEGEPS
        STARTINCR=BEGEPS/10.0
        IF (STARTINCR.LT.1.0/(N+1)) STARTINCR=1.0/(N+1)

        PRINT*,'*****************************************'
        PRINT*,'STARTING EPSILON = ',BEGEPS
        PRINT*,'EPSILON REDUCTION FACTOR = ',FACTOR
        PRINT*,'THRESHOLD EPSILON BEFORE IT IS SET TO 1/(N+1) = ',ENDEPS
        PRINT*,'STARTING MIN BIDDING INCREMENT = ',STARTINCR
        PRINT*,'CALLING AUCTION TO SOLVE THE PROBLEM'
        PRINT*,'*****************************************'
```

```
C     GET STARTING TIME FOR THE MAC II

      TT1 = LONG(362)/60.0

      CALL AUCTION(BEGEPS,FACTOR,ENDEPS,STARTINCR)

C     GET ENDING TIME FOR THE MAC II

      TT2 = LONG(362)/60.0 - TT1
      PRINT *,'FINISHED --- TOTAL CPU TIME', TT2,' SECS'
      PRINT*,'*****************************************'

C     *** DISPLAY RESULTS ***

X        WRITE(9,2010) CYCLES
X2010    FORMAT(' NO OF AUCTION CYCLES',I7)
X        WRITE(9,2020) AVERAGE
X2020    FORMAT(' AVERAGE NUMBER OF BIDS PER CYCLE',F9.3)
        WRITE(9,2030) NUMPHASES
2030    FORMAT('NO OF EPSILON SUBPROBLEMS SOLVED =',I7)

C       CHECK FEASIBILITY OF SOLUTION & CALCULATE COST

        DO 40 I=1,N
          COLASS(I)=0
40      CONTINUE
        NOASS=0
        DO 50 J=1,N
           IF (ASSIGN(J).GT.0) THEN
             NOASS=NOASS+1
             COLASS(ASSIGN(J))=J
           END IF
50      CONTINUE
        IF (NOASS.NE.N) THEN
          PRINT*,'# OF ASSIGNED ROWS NOT EQUAL TO # OF ROWS'
        END IF
        TCOST=0
        DO 60 I=1,N
          J=COLASS(I)
          IF (J.EQ.0) THEN
            PRINT*,'ROW ',I,' IS UNASSIGNED'
          END IF
          FSTARC=FOUT(I)
          LSTARC=FOUT(I+1)-1
          MCOST=ISMALL
          DO 70 CURARC=FSTARC,LSTARC
            CURCOL=END(CURARC)
            IF (CURCOL.EQ.J) THEN
              IF (MCOST.LT.COST(CURARC)) THEN
                 MCOST=COST(CURARC)
              END IF
            END IF
70        CONTINUE
          PROFIT=MCOST-PCOL(J)
```

```
          DO 72 CURARC=FSTARC,LSTARC
             J=END(CURARC)
             TEST=COST(CURARC)-PCOL(J)-PROFIT
             IF (TEST.GT.1.0/N) THEN
                PRINT *,'1/(N+1)-CS VIOLATED AT ARC',CURARC
             END IF
72        CONTINUE
          TCOST=TCOST+MCOST
60      CONTINUE
        WRITE(9,2100) TCOST
2100    FORMAT(' ASSIGNMENT COST=',F22.2)
        PRINT *, ' PROGRAM ENDED; <CR> TO EXIT '
        PAUSE
        END



C ***************************************************************
C
C    FLOATING POINT AUCTION CODE FOR N BY N ASSIGNMENT PROBLEMS
C
C             WRITTEN BY DIMITRI P. BERTSEKAS
C        (MODIFICATIONS BY DAVID CASTANON AND PAUL TSENG)
C
C                  VERSION 1.2, OCT. 1991
C
C
C MODIFIED BY DAVID CASTANON (OCT. 1991) TO WORK WITH ARBITRARILY
C LARGE COST RANGE (UP TO THE INTEGER RANGE OF THE MACHINE);
C TO ACCOMPLISH THIS, THE OBJECT PRICES AND EPSILON ARE
C DOUBLE PRECISION VARIABLES, SO THE PRICES CANNOT OVERFLOW THE
C MACHINE'S INTEGER RANGE.
C
C THIS CODE IMPLEMENTS THE AUCTION ALGORITHM WITH E-SCALING.
C IT SOLVES A SEQUENCE OF SUBPROBLEMS AND DECREASES
C EPSILON BY A CONSTANT FACTOR BETWEEN SUBPROBLEMS.
C THIS VERSION CORRESPONDS TO A GAUSS-SEIDEL MODE
C AND SOLVES EPSILON SUBPROBLEMS INEXACTLY.
C
C THE CODE IS AN IMPROVED VERSION OF AN EARLIER (SEPT. 1985)
C AUCTION CODE WITH E-SCALING WRITTEN BY DIMITRI P. BERTSEKAS
C
C THE CODE TREATS THE PROBLEM AS A MAXIMIZATION PROBLEM.
C TO SOLVE A MINIMIZATION PROBLEM, REVERSE THE SIGN OF THE
C ARC COSTS PRIOR TO CALLING AUCTION, AND REVERSE AGAIN
C THE SIGN OF THE OPTIMAL COST UPON RETURN FROM AUCTION.
C THIS CODE ALLOWS MULTIPLE ARCS BETWEEN A ROW AND A COLUMN.
C
C THIS VERSION OF THE AUCTION ALGORITHM IS ADAPTIVE. HERE THE MINIMAL
C BIDDING INCREMENT (STORED IN THE VARIABLE INCR) MAY BE SMALLER THAN
C EPSILON. FOR EVERY SUBPROBLEM, INCR STARTS AT THE PARAMETER VALUE
C STARTINCR (WHICH IS PASSED TO THE AUCTION ROUTINE) AND IS INCREASED
C BY A FACTOR OF 2 AT THE END OF EACH CYCLE, UP TO A MAXIMUM VALUE OF
C EPSILON. THIS ADAPTIVE FEATURE IS PARTICULARLY EFFECTIVE FOR
```

```
C DENSE PROBLEMS. IT CAN BE DEFEATED BY SELECTING STARTINCR=BEGEPS
C
C ****************************************************************
C
C THE USER MUST SUPPLY THE FOLLOWING PROBLEM DATA IN FORWARD STAR FORMAT
C (THAT IS, ALL ARCS OF THE SAME ROW ARE NUMBERED CONSECUTIVELY):
C       N=NUMBER OF ROWS (EQUALS NUMBER OF COLUMNS)
C       A=NUMBER OF ARCS
C       FOUT(ROW)=FIRST ARC COMING OUT OF ROW
C       COST(ARC)=COST OF ARC
C       END(ARC)=COLUMN CORRESPONDING TO ARC
C
C AND THE FOLLOWING PARAMETERS FOR THE AUCTION ALGORITHM:
C       BEGEPS=STARTING VALUE OF EPSILON (MUST BE NO LESS THAN 1/(N+1))
C       ENDEPS=FINAL VALUE OF EPSILON BEFORE IT IS SET TO 1/(N+1)
C       FACTOR=FACTOR BY WHICH EPSILON IS DECREASED BETWEEN SUBPROBLEMS
C       STARTINCR=THE STARTING VALUE OF THE BIDDING INCREMENT
C       ENDEPS SHOULD NOT EXCEED BEGEPS.
C       FACTOR MUST BE GREATER THAN 1.
C
C FOUT(.) IS AN ARRAY OF LENGTH N.
C COST(.),END(.) ARE ARRAYS OF LENGTH A.
C
C THE SOLUTION IS CONTAINED IN THE ARRAY ASSIGN(.) WHERE
C       ASSIGN(COL) GIVES THE ROW ASSIGNED TO COL.
C
C THIS ALGORITHM DOES NOT CHECK FOR INFEASIBILITY OF THE PROBLEM.
C TO MAKE SURE THE PROBLEM IS FEASIBLE THE USER MAY ADD
C ADDITIONAL VERY SMALL COST ARCS.
C
C ****************************************************************
C ALL PROBLEM DATA ARE INTEGER
C ****************************************************************


        SUBROUTINE AUCTION(BEGEPS,FACTOR,ENDEPS,STARTINCR)

        PARAMETER(MAXNODES=10000, MAXARCS=100000)

        IMPLICIT NONE
        INTEGER NONEWLIST,THRESH
        INTEGER A,K,N,I,J,CURARC,CURCOL
        INTEGER ROW,FSTARC,FSTCOL,SNDARC,SNDCOL,BSTCOL
        INTEGER TRDARC
        INTEGER M,ISMALL,ILARGE,LARGEINCR,CYCLES
        INTEGER NUMPHASES,LSTARC,NOLIST,OLDROW
        INTEGER FOUT(MAXNODES),LIST(MAXNODES)
        INTEGER ASSIGN(MAXNODES)
        INTEGER COST(MAXARCS),END(MAXARCS)
        REAL*8 BEGEPS,ENDEPS,FACTOR,STARTINCR,INCRFACTOR
        REAL*8 PCOL(MAXNODES)
        REAL*8 MAX1,MAX2,TMAX,EPSILON,INCR
        REAL*8 AVERAGE
        COMMON/ARRAYC/COST/ARRAYS/END/ARRAYF/FOUT/BK1/N,A,ISMALL
```

```
     $  /BK2/CYCLES,AVERAGE,NUMPHASES/ARRAYA/ASSIGN/PCOLA/PCOL

C ***********************************************************

C   ******* CHECK VALIDITY OF PARAMETERS PASSED *******

        IF (BEGEPS.LE.1.0/(N+2)) THEN
          PRINT*,'STARTING VALUE OF EPSILON IS LESS THAN 1/(N+1)'
          PRINT*,'EXECUTION ABORTED'
          STOP
        END IF
        IF (ENDEPS.GT.BEGEPS) THEN
          PRINT*,'PARAMETER ENDEPS IS GREATER THAN PARAMETER BEGEPS'
          PRINT*,'ENDEPS IS SET AT THE DEFAULT VALUE OF 1/(N+1)'
          ENDEPS=1.0/(N+1)
        END IF
        IF ((FACTOR.LE.1).AND.(BEGEPS.GE.1.0/N)) THEN
          PRINT*,'EPSILON REDUCTION FACTOR IS NOT GREATER THAN 1'
          PRINT*,'EXECUTION ABORTED'
          STOP
        END IF
        IF (STARTINCR.LE.1.0/(N+2)) THEN
          PRINT*,'MIN BIDDING INCREMENT IS LESS THAN 1/(N+1)'
          PRINT*,'STARTINCR IS SET AT THE DEFAULT VALUE OF 1/(N+1)'
          STARTINCR=1.0/(N+1)
        END IF

C   ******* INITIALIZATION *******

        EPSILON=BEGEPS
        ILARGE=-ISMALL
        LARGEINCR=INT(ILARGE/10)
        THRESH=INT(0.2*N)
        INCRFACTOR=2.0
        IF (THRESH.GT.100) THRESH=100
X        CYCLES=1
X        AVERAGE=N
         NUMPHASES=1
         DO 10 J=1,N
           ASSIGN(J)=0
           PCOL(J)=ISMALL
10      CONTINUE

        FOUT(N+1)=A+1
        NOLIST=N
        DO 20 I=1,N
            LIST(I)=I
20      CONTINUE

C ***********************************************************
C
C  THIS IMPLEMENTATION OF THE AUCTION ALGORITHM OPERATES IN CYCLES.
C  EACH CYCLE CONSISTS OF ONE BID BY EACH OF THE ROWS THAT ARE
C  UNASSIGNED AT THE START OF THE CYCLE (THESE ROWS ARE STORED IN
```

```
C  THE ARRAY LIST(.)). AS THE CYCLE PROGRESSES NEW
C  ROWS BECOME UNASSIGNED; THESE ARE STORED IN LIST(.)
C  AND WILL SUBMIT A BID AT THE NEXT CYCLE.
C  NOTE THAT ONLY ONE ROW SUBMITS A BID AT A TIME; THIS IS KNOWN AS
C  THE GAUSS-SEIDEL VERSION OF THE ALGORITHM. THE VERSION WHERE ALL
C  UNASSIGNED ROWS SUBMIT A BID AT THE SAME TIME IS KNOWN AS THE JACOBI
C  VERSION OF THE ALGORITHM, AND HAS NOT BEEN IMPLEMENTED. IT GENERALLY
C  TENDS TO RUN SOMEWHAT SLOWER THAN THE GAUSS-SEIDEL VERSION, BUT
C  IT ADMITS A HIGHER DEGREE OF PARALLELIZATION. A JACOBI VERSION
C  MAY BE PREFERABLE ON A PARALLEL MACHINE, BUT IS USUALLY INFERIOR
C  TO THE GAUSS-SEIDEL VERSION ON A SERIAL MACHINE.
C
C ***************************************************************
C
C       START SUBPROBLEM (SCALING PHASE) W/ NEW EPSILON
C
C ***************************************************************

12      CONTINUE

        IF (EPSILON.LE.1.0/(N+1)) THRESH=0
        INCR=STARTINCR
        IF (INCR.GT.EPSILON) INCR=EPSILON

C ***************************************************************
C
C       START AUCTION CYCLE WITH NEW LIST
C
C ***************************************************************

15      CONTINUE

C  INITIALIZE COUNT OF NEXT LIST OF UNASSIGNED ROWS

        NONEWLIST=0

C  CYCLE THROUGH THE CURRENT LIST OF UNASSIGNED ROWS

        DO 100 I=1,NOLIST
        ROW=LIST(I)
        FSTARC=FOUT(ROW)
        LSTARC=FOUT(ROW+1)-1
        FSTCOL=END(FSTARC)

C  FIRST TAKE CARE OF THE EXCEPTIONAL CASE WHERE ROW HAS ONLY ONE ARC

        IF (FSTARC.EQ.LSTARC) THEN
          PCOL(FSTCOL)=PCOL(FSTCOL)+LARGEINCR
          OLDROW=ASSIGN(FSTCOL)
          ASSIGN(FSTCOL)=ROW
          IF (OLDROW.GT.0) THEN
            NONEWLIST=NONEWLIST+1
            LIST(NONEWLIST)=OLDROW
          END IF
```

```
          GO TO 100
        END IF

C  NEXT TAKE CARE OF THE REGULAR CASE WHERE ROW HAS MULTIPLE ARCS

        SNDARC=FSTARC+1
        SNDCOL=END(SNDARC)
        MAX1=COST(FSTARC)-PCOL(FSTCOL)
        MAX2=COST(SNDARC)-PCOL(SNDCOL)
        IF (MAX1.GE.MAX2) THEN
           BSTCOL=FSTCOL
        ELSE
           TMAX=MAX1
           MAX1=MAX2
           MAX2=TMAX
           BSTCOL=SNDCOL
        END IF
        IF (SNDARC.LT.LSTARC) THEN
        TRDARC=SNDARC+1
        DO 40 CURARC=TRDARC,LSTARC
           CURCOL=END(CURARC)
           TMAX=COST(CURARC)-PCOL(CURCOL)
           IF (TMAX.GT.MAX2) THEN
             IF (TMAX.GT.MAX1) THEN
                  MAX2=MAX1
                  MAX1=TMAX
                  BSTCOL=CURCOL
             ELSE
                  MAX2=TMAX
             END IF
           END IF
40      CONTINUE
        END IF


C  ROW BIDS FOR BSTCOL INCREASING ITS PRICE, AND GETS ASSIGNED
C  TO BSTCOL, WHILE ANY ROW ASSIGNED TO BSTCOL BECOMES UNASSIGNED

         PCOL(BSTCOL)=PCOL(BSTCOL)+MAX1-MAX2+INCR
         OLDROW=ASSIGN(BSTCOL)
         ASSIGN(BSTCOL)=ROW
         IF (OLDROW.GT.0) THEN
           NONEWLIST=NONEWLIST+1
           LIST(NONEWLIST)=OLDROW
         END IF

100    CONTINUE

C      ************* END OF AN AUCTION CYCLE ***************

C  OPTIONALLY COLLECT STATISTICS

X        AVERAGE=(CYCLES*AVERAGE+NOLIST)/(CYCLES+1)
X        CYCLES=CYCLES+1
```

```
C  CHECK IF THERE ARE STILL 'MANY' UNASSIGNED ROWS, THAT IS, IF THE
C  NUMBER OF UNASSIGNED ROWS IS GREATER THAN
C  THE PARAMETER THRESH. IF NOT, REPLACE CURRENT LIST WITH THE NEW LIST,
C  AND GO FOR ANOTHER CYCLE. OTHERWISE, IF EPSILON > 1, REDUCE EPSILON,
C  RESET THE ASSIGNMENT TO EMPTY AND RESTART AUCTION;
C  IF EPSILON = 1 TERMINATE.
C  ALSO INCREASE THE MINIMAL BIDDING INCREMENT UP TO A MAXIMUM
C  VALUE OF EPSILON (THIS IS THE ADAPTIVE FEATURE)

        INCR=INCR*INCRFACTOR
        IF (INCR.GT.EPSILON) INCR=EPSILON
        IF (NONEWLIST.GT.THRESH) THEN
            NOLIST=NONEWLIST
            GO TO 15
        END IF

C ************************************************************
C
C               END OF SUBPROBLEM (SCALING PHASE)
C
C ************************************************************

C ****** IF EPSILON IS 1 TERMINATE ******

       IF (EPSILON.LE.1.0/N) THEN
          RETURN
       ELSE

C   ELSE REDUCE EPSILON AND RESET THE ASSIGNMENT TO EMPTY

          NUMPHASES=NUMPHASES+1
          EPSILON=EPSILON/FACTOR
          IF (EPSILON.GT.INCR) EPSILON=EPSILON/FACTOR
          IF ((EPSILON.LT.1.0/N).OR.(EPSILON.LT.ENDEPS)) THEN
             EPSILON=1.0/(N+1)
          END IF
          THRESH=INT(THRESH/FACTOR)

X         PRINT*,'*** END OF A SCALING PHASE; NEW EPSILON=',EPSILON

          DO 200 J=1,N
           IF (ASSIGN(J).GT.0) THEN
            NONEWLIST=NONEWLIST+1
            LIST(NONEWLIST)=ASSIGN(J)
            ASSIGN(J)=0
           END IF
200       CONTINUE



C     FINAL PARAMETER UPDATES BEFORE RETURNING FOR ANOTHER SCALING PHASE

          NOLIST=NONEWLIST
```

```
          IF (STARTINCR.LT.EPSILON) STARTINCR=FACTOR*STARTINCR
          GO TO 12
        END IF


        END



      SUBROUTINE SETRAN(ISEED)
      IMPLICIT REAL*8 (A-H,O-Z) , INTEGER*4 (I-N)
C*********************************************************************
C PORTABLE CONGRUENTIAL (UNIFORM) RANDOM NUMBER GENERATOR:
C     NEXT_VALUE = [(7**5) * PREVIOUS_VALUE] MODULO[(2**31)-1]
C
C THIS GENERATOR CONSISTS OF TWO ROUTINES:
C   (1) SETRAN - INITIALIZES CONSTANTS AND SEED
C   (2) RRAN   - GENERATES A REAL RANDOM NUMBER
C
C THE GENERATOR REQUIRES A MACHINE WITH AT LEAST 32 BITS OF PRECISION.
C THE SEED (ISEED) MUST BE IN THE RANGE (1,(2**31)-1).
C*********************************************************************
      COMMON /RANDM/ MULT,MODUL,I15,I16,JRAN
      IF(ISEED.LT.1) STOP 77
      MULT=16807
      MODUL=2147483647
      I15=2**15
      I16=2**16
      JRAN=ISEED
      RETURN
      END

      REAL FUNCTION RAN()
      IMPLICIT REAL*4 (A-H,O-Z) , INTEGER*4 (I-N)
C*********************************************************************
C RAN GENERATES A REAL RANDOM NUMBER BETWEEN 0 AND 1
C*********************************************************************
      COMMON /RANDM/ MULT,MODUL,I15,I16,JRAN
      IXHI=JRAN/I16
      IXLO=JRAN-IXHI*I16
      IXALO=IXLO*MULT
      LEFTLO=IXALO/I16
      IXAHI=IXHI*MULT
      IFULHI=IXAHI+LEFTLO
      IRTLO=IXALO-LEFTLO*I16
      IOVER=IFULHI/I15
      IRTHI=IFULHI-IOVER*I15
      JRAN=((IRTLO-MODUL)+IRTHI*I16)+IOVER
      IF(JRAN.LT.0) JRAN=JRAN+MODUL
      RAN = FLOAT(JRAN)/FLOAT(MODUL)
      RETURN
      END
```

AUCTION-AS

This code implements the forward/reverse auction
algorithm with $\e$-scaling for the asymmetric assignment problem; cf.\
Section 4.2.1. It can also solve as a special case the symmetric
assignment
problem. In this case as well as in the case where $\e$-scaling is
bypassed
($\e=1$ throughout the algorithm), only the forward part of the algorithm
is
used. Note also that this code uses the ``third best'' object
implementation
described in Exercise 1.7 of Section 4.1.

```
C ***************************************************************
C
C       SAMPLE CALLING PROGRAM FOR AUCTION ALGORITHM
C          FOR THE ASYMMETRIC ASSIGNMENT PROBLEM
C
C   THIS DRIVER CREATES AN ASYMMETRIC ASSIGNMENT PROBLEM
C   WITH LESS OR EQUAL NUMBER OF ROWS THAN COLUMNS,
C   AND CALLS THE AUCTION_AS SUBROUTINE TO FIND AN
C   ASSIGNMENT OF MAXIMAL VALUE.
C
C   THIS VERSION USES A THIRD BEST VALUE
C
C ***************************************************************

        PARAMETER(MAXNODES=10000, MAXARCS=100000)

        IMPLICIT NONE
        INTEGER M,N,NA,A,IA,K,ILARGE,BEGEPS,ENDEPS,CYCLES
        INTEGER NUMPHASES,STARTINCR,LAMBDA
        INTEGER I,J,ARC,NOASS,ICOST,ABSCOST,CURARC
        INTEGER CURCOL,FSTARC,LSTARC,MINCOST,MAXCOST,MCOST
        INTEGER EXTRA,REMAINDER,INDEX,COUNT
        INTEGER COL_ASSIGNED_TO(MAXNODES),ROW_ASSIGNED_TO(MAXNODES)
        INTEGER FOUT(MAXNODES),FIN(MAXNODES),NXTIN(MAXARCS)
        INTEGER COST(MAXARCS),START(MAXARCS),END(MAXARCS)
        INTEGER PCOL(MAXNODES),PROW(MAXNODES)
        INTEGER PRDARC(MAXNODES)
        REAL*8 FACTOR,TT1,TT2,TCOST,AVERAGE
        COMMON/ARRAYC/COST/ARRAYS/START/ARRAYE/END
        COMMON/ARRAYFO/FOUT/ARRAYFI/FIN/ARRAYNI/NXTIN
        COMMON/ARRAYRA/ROW_ASSIGNED_TO/ARRAYCA/COL_ASSIGNED_TO
        COMMON/ARRAYPC/PCOL/ARRAYPR/PROW
        COMMON/BK1/M,N,A,ILARGE
        COMMON/BK2/CYCLES,AVERAGE,NUMPHASES,LAMBDA

C ***************************************************************
C
```

```
C     PROBLEM GENERATION CODE STARTS HERE
C     THE USER MAY REPLACE THIS CODE WITH A CODE THAT READS
C     HIS/HER PROBLEM FROM A FILE
C
C ***************************************************************

C     THIS CODE INCLUDES A UNIFORM RANDOM NUMBER GENERATOR
C     WHICH RETURNS A VALUE IN (0,1)

C     INITIALIZE RANDOM GENERATOR

      INTEGER MULT,MODUL,I15,I16,JRAN,ISEED
      REAL RAN
      COMMON /RANDM/ MULT,MODUL,I15,I16,JRAN

      ISEED=13502460
      CALL SETRAN(ISEED)

      PRINT*,'GENERATING AN ASYMMETRIC ASSIGNMENT PROBLEM'
      PRINT*,'******************************************'

C **** READ THE NUMBER OF ROWS M, COLUMNS N, AND ARCS A ****

      PRINT*,'ENTER THE NUMBER OF ROWS'
      READ*,M

7. PRINT*,'ENTER THE NUMBER OF COLUMNS (>= # OF ROWS)'
      READ*,N
      IF (N.LT.M) GO TO 2

8. PRINT*,'ENTER THE NUMBER OF ARCS PER ROW (>1)'
      READ*,NA
      IF (NA.LT.2) GOTO 5

      PRINT*,'ENTER THE MINIMUM AND THE MAXIMUM COST'
      READ*,MINCOST,MAXCOST

C     THE NUMBER OF ARCS IS M*NA+N-M

      A=M*NA+N-M
      PRINT*,'THE NUMBER OF ARCS IS ',A

C      THE ARCS INCIDENT TO ROW I ARE FOUT(I) TO FOUT(I+1)-1
C      ALSO, FOR FEASIBILITY EACH ROW IS DIRECTLY CONNECTED
C      WITH THE CORRESPONDING COLUMN;
C      ALSO EACH COLUMN HAS AT LEAST ONE ARC

      EXTRA=INT((N-M)/M)

      DO 20 I=1,M
        FOUT(I)=1+(I-1)*(NA+EXTRA)
20    CONTINUE
      FOUT(M+1)=A+1
```

```
      DO 22 I=1,M
        DO 23 ARC=FOUT(I),FOUT(I+1)-1
          START(ARC)=I
23      CONTINUE
22    CONTINUE

C      GENERATE THE END(ARC) AND COST(ARC) WHICH ARE THE COLUMN
C      AND THE COST COEFFICIENT ASSOCIATED WITH ARC

      DO 25 ARC=1,A
        END(ARC)=1+RAN()*N
        IF ((END(ARC).GT.N).OR.(END(ARC).LT.1)) THEN
          PRINT*,'ERROR IN PROBLEM GENERATION'
          PAUSE
          STOP
        END IF
        COST(ARC)=MINCOST+RAN()*(MAXCOST-MINCOST)
25    CONTINUE

C   MODIFY THE END OF THE LAST ARC OUT OF EACH ROW FOR FEASIBILITY
C   AND SET ITS COST TO -MAXCOST

      DO 30 I=1,M
        END(FOUT(I+1)-1)=I
        COST(FOUT(I+1)-1)=-MAXCOST
30    CONTINUE

C   MODIFY THE END OF SOME ARCS SO THAT EACH COLUMN HAS AT LEAST ONE ARC

      IF (EXTRA.GE.1) THEN
        DO 32 I=1,M
          DO 33 K=1,EXTRA
            END(FOUT(I)+K-1)=K*M+I
33        CONTINUE
32      CONTINUE
       END IF

       REMAINDER=N-M*(1+EXTRA)
       IF (REMAINDER.GE.1) THEN
         INDEX=FOUT(M)+EXTRA-1
         DO 35 J=1,REMAINDER
           END(INDEX+J)=(1+EXTRA)*M+J
35       CONTINUE
       END IF

C    CONSTRUCT THE FIN() AND NXTIN() ARRAYS

      DO 42 J=1,N
        FIN(J)=0
        PRDARC(J)=0
42    CONTINUE

      DO 43 ARC=1,A
        NXTIN(ARC)=0
```

```
        J=END(ARC)
        IF (FIN(J).NE.0) THEN
          NXTIN(PRDARC(J))=ARC
        ELSE
          FIN(J)=ARC
        END IF
        PRDARC(J)=ARC
43    CONTINUE


C ***************************************************************
C
C          PROBLEM GENERATION CODE ENDS HERE
C
C ***************************************************************



C       SCALE THE COST TO WORK WITH INTEGER EPSILON

        MAXCOST=0
        DO 45 IA=1,A
           ABSCOST=IABS(COST(IA))
           IF (ABSCOST.GT.MAXCOST) MAXCOST=ABSCOST
           COST(IA)=COST(IA)*(M+1)
45      CONTINUE

C *** ILARGE IS A VERY LARGE INTEGER FOR YOUR MACHINE ***

        ILARGE=2000000000

        IF (MAXCOST.GT.INT(ILARGE/(M+1))) THEN
          PRINT*,'THE COST RANGE IS TOO LARGE FOR INTEGER ARITHMETIC'
          PAUSE
          STOP
        END IF

        MAXCOST=MAXCOST*(M+1)

        LAMBDA=-ILARGE

C       THE FOLLOWING PARAMETERS BEGEPS, FACTOR, ENDEPS, AND STARTINCR
C       ARE PASSED TO THE AUCTION ALGORITHM. VALUES BETWEEN
C        (A) MAXCOST/5 AND MAXCOST/2 FOR BEGEPS
C        (B) 4 AND 6 FOR FACTOR
C        (C) M/10 AND 1 FOR ENDEPS
C        (D) 1 AND BEGEPS FOR STARTINCR
C       HAVE WORKED WELL FOR LARGE SPARSE PROBLEMS.
C       FOR DENSE PROBLEMS AND FOR VERY ASYMMETRIC PROBLEMS
C       IT IS RECOMMENDED THAT
C       BEGEPS BE SET TO A SMALLER VALUE (POSSIBLY 1),
C       ENDEPS BE SET TO 1,
C       STARTINCR BE SET TO 1.
```

```
C       FOR VERY ASYMMETRIC PROBLEMS, BEGEPS=1, CORRESPONDING TO
C       NO E-SCALING, MAY WORK BEST AND SHOULD BE AT LEAST TRIED.

        PRINT*,'MAXIMUM COST IS ',MAXCOST
        PRINT*,'ENTER THE STARTING EPSILON'
        READ*,BEGEPS
        IF (BEGEPS.LT.1) BEGEPS=1
        PRINT*,'ENTER THE EPSILON REDUCTION FACTOR'
        READ*,FACTOR
        ENDEPS=M/10
        IF (ENDEPS.LT.1) ENDEPS=1
        IF (ENDEPS.GT.BEGEPS) ENDEPS=BEGEPS
        STARTINCR=BEGEPS/10
        IF (STARTINCR.LT.1) STARTINCR=1

        PRINT*,'*****************************************'
        PRINT*,'STARTING EPSILON = ',BEGEPS
        PRINT*,'EPSILON REDUCTION FACTOR = ',FACTOR
        PRINT*,'THRESHOLD EPSILON BEFORE IT IS SET TO 1 = ',ENDEPS
        PRINT*,'STARTING MIN BIDDING INCREMENT = ',STARTINCR
        PRINT*,'CALLING ASYMMETRIC ASSIGNMENT AUCTION'
        PRINT*,'*****************************************'

C     GET STARTING TIME FOR THE MAC II

      TT1 = LONG(362)/60.0

      CALL AUCTION_AS(BEGEPS,FACTOR,ENDEPS,STARTINCR)

C     GET ENDING TIME FOR THE MAC II

      TT2 = LONG(362)/60.0 - TT1
      PRINT *,'FINISHED --- TOTAL CPU TIME', TT2,' SECS'
      PRINT*,'*****************************************'

C     *** DISPLAY RESULTS ***

X        WRITE(9,2010) CYCLES
X2010    FORMAT(' NO OF AUCTION CYCLES',I7)
X        WRITE(9,2020) AVERAGE
X2020    FORMAT(' AVERAGE NUMBER OF BIDS PER CYCLE',F9.3)
        WRITE(9,2030) NUMPHASES
2030    FORMAT('NO OF EPSILON SUBPROBLEMS SOLVED =',I7)

C       CHECK OPTIMALITY & CALCULATE COST

        DO 48 J=1,N
          I=ROW_ASSIGNED_TO(J)
          IF (N.GT.M) THEN
            ROW_ASSIGNED_TO(J)=0
          ELSE
            IF (I.GT.0) THEN
              COL_ASSIGNED_TO(I)=J
            END IF
```

```
          END IF
48      CONTINUE


        TCOST=0
        DO 50 I=1,M
          J=COL_ASSIGNED_TO(I)
          ROW_ASSIGNED_TO(J)=I
          IF (J.EQ.0) THEN
             PRINT*,'ROW ',I,' IS UNASSIGNED'
          END IF
          IF (PCOL(J).LT.LAMBDA) THEN
             PRINT*,'CS VIOLATION AT ASSIGNED COLUMN ',J
          END IF
          FSTARC=FOUT(I)
          LSTARC=FOUT(I+1)-1
          MCOST=-ILARGE
          DO 55 ARC=FSTARC,LSTARC
            CURCOL=END(ARC)
            IF (PROW(I)+PCOL(CURCOL).LT.COST(ARC)-1) THEN
              PRINT*,'1-CS VIOLATED AT ARC ',ARC
            END IF
            IF (CURCOL.EQ.J) THEN
              IF (MCOST.LT.COST(ARC)) THEN
                 MCOST=COST(ARC)
              END IF
            END IF
55        CONTINUE
          TCOST=TCOST+MCOST/(M+1)
          IF (PROW(I)+PCOL(J).NE.MCOST) THEN
            PRINT*,'1-CS VIOLATED AT ROW ',I
          END IF
50      CONTINUE

        COUNT=0
        DO 60 J=1,N
           IF (ROW_ASSIGNED_TO(J).EQ.0) THEN
             IF (PCOL(J).GT.LAMBDA) THEN
               PRINT*,'CS VIOLATION AT UNASSIGNED COLUMN ',J
             END IF
           ELSE
             COUNT=COUNT+1
           END IF
60      CONTINUE

        IF (COUNT.LT.M) THEN
          PRINT*,'THE NUMBER OF ASSIGNED COLUMNS IS WRONG'
        END IF


        WRITE(9,2100) TCOST
2100    FORMAT(' ASSIGNMENT COST=',F18.2)
        PRINT *, ' PROGRAM ENDED; <CR> TO EXIT '
        PAUSE
```

```
      END



C ***************************************************************
C
C     AUCTION CODE FOR ASYMMETRIC M BY N ASSIGNMENT PROBLEMS
C
C               WRITTEN BY DIMITRI P. BERTSEKAS
C
C                         DEC. 1990
C
C THIS CODE IMPLEMENTS A FORWARD/REVERSE AUCTION ALGORITHM
C WITH E-SCALING FOR THE ASYMMETRIC ASSIGNMENT PROBLEM.
C IT SOLVES A SEQUENCE OF SUBPROBLEMS AND DECREASES
C EPSILON BY A CONSTANT FACTOR BETWEEN SUBPROBLEMS.
C THIS VERSION CORRESPONDS TO A GAUSS-SEIDEL MODE.
C
C THE CODE TREATS THE PROBLEM AS A MAXIMIZATION PROBLEM.
C TO SOLVE A MINIMIZATION PROBLEM, REVERSE THE SIGN OF THE
C ARC COSTS PRIOR TO CALLING AUCTION, AND REVERSE AGAIN
C THE SIGN OF THE OPTIMAL COST UPON RETURN FROM AUCTION.
C THIS CODE ALLOWS MULTIPLE ARCS BETWEEN A ROW AND A COLUMN.
C
C THIS VERSION OF THE AUCTION ALGORITHM IS ADAPTIVE. HERE THE MINIMAL
C BIDDING INCREMENT (STORED IN THE VARIABLE INCR) MAY BE SMALLER THAN
C EPSILON. FOR EVERY SUBPROBLEM, INCR STARTS AT THE PARAMETER VALUE
C STARTINCR (WHICH IS PASSED TO THE AUCTION ROUTINE) AND IS INCREASED
C BY A FACTOR OF 2 AT THE END OF EACH CYCLE, UP TO A MAXIMUM VALUE OF
C EPSILON. THIS ADAPTIVE FEATURE IS PARTICULARLY EFFECTIVE FOR
C DENSE PROBLEMS. IT CAN BE DEFEATED BY SELECTING STARTINCR=BEGEPS
C
C ***************************************************************
C
C THE USER MUST SUPPLY THE FOLLOWING PROBLEM DATA IN FORWARD STAR FORMAT
C (THAT IS, ALL ARCS OF THE SAME ROW ARE NUMBERED CONSECUTIVELY):
C       M=NUMBER OF ROWS
C       N=NUMBER OF COLUMNS
C       A=NUMBER OF ARCS
C       FOUT(ROW)=FIRST ARC COMING OUT OF ROW
C       FIN(COL)=FIRST ARC COMING INTO COL
C       NXTIN(ARC)=NEXT ARC INCIDENT TO THE SAME COLUMN AS ARC
C       COST(ARC)=COST OF ARC
C       START(ARC)=ROW CORRESPONDING TO ARC
C       END(ARC)=COLUMN CORRESPONDING TO ARC
C
C AND THE FOLLOWING PARAMETERS FOR THE AUCTION ALGORITHM:
C       BEGEPS=STARTING VALUE OF EPSILON (MUST BE NO LESS THAN 1)
C       ENDEPS=FINAL VALUE OF EPSILON BEFORE IT IS SET TO 1
C       FACTOR=FACTOR BY WHICH EPSILON IS DECREASED BETWEEN SUBPROBLEMS
C       STARTINCR=THE STARTING VALUE OF THE BIDDING INCREMENT
C       ENDEPS SHOULD NOT EXCEED BEGEPS.
C       FACTOR MUST BE GREATER THAN 1.
```

```
C
C FOUT(.) IS AN ARRAY OF LENGTH N.
C COST(.),END(.) ARE ARRAYS OF LENGTH A.
C
C THE SOLUTION IS CONTAINED IN THE ARRAY ROW_ASSIGNED_TO(.) WHERE
C    ROW_ASSIGNED_TO(COL) GIVES THE ROW ASSIGNED TO COL.
C ALSO COL_ASSIGNED_TO(ROW) GIVES THE COLUMN ASSIGNED TO ROW.
C
C THIS ALGORITHM DOES NOT CHECK FOR INFEASIBILITY OF THE PROBLEM.
C TO MAKE SURE THE PROBLEM IS FEASIBLE THE USER MAY ADD
C    ADDITIONAL VERY SMALL COST ARCS.
C
C THIS CODE MAY FAIL DUE TO INTEGER OVERFLOW IF THE NUMBER OF NODES
C OR THE COST RANGE (OR BOTH) ARE LARGE. TO CORRECT THIS SITUATION,
C THE PRICES AND OTHER RELATED VARIABLES (MAX1,MAX2,TMAX ETC) SHOULD
C BE DECLARED AS DOUBLE PRECISION REALS.
C ***************************************************************
C ALL PROBLEM DATA ARE INTEGER
C ***************************************************************



      SUBROUTINE AUCTION_AS(BEGEPS,FACTOR,ENDEPS,STARTINCR)

      PARAMETER(MAXNODES=10000, MAXARCS=100000)

      IMPLICIT NONE
      INTEGER A,K,N,I,J,M,CURARC,CURCOL,CURROW,LAMBDA,DELTA
      INTEGER THRESH,INCR,STARTINCR,INCRFACTOR,EPS_THRESH
      INTEGER NOLIST,RNOLIST,NONEWLIST,RNONEWLIST
      INTEGER ROW,COLUMN, BSTROW,BSTCOL
      INTEGER FSTARC,FSTCOL,NXTARC,NXTROW,LSTARC,SNDARC,SNDCOL
      INTEGER MAX1,MAX2,TMAX,TMIN,TRDARC,EPSILON,BEGEPS,ENDEPS
      INTEGER ISMALL,ILARGE,LARGEINCR,CYCLES
      INTEGER NUMPHASES,OLDROW,OLDCOL
      INTEGER MAX3,BSTARC,SBSTARC
      INTEGER CUR_BARC,TRDVAL(MAXNODES)
      LOGICAL INIT
      INTEGER BEST_ARC(MAXNODES),SECD_ARC(MAXNODES)
      INTEGER COL_ASSIGNED_TO(MAXNODES),ROW_ASSIGNED_TO(MAXNODES)
      INTEGER FOUT(MAXNODES),FIN(MAXNODES),NXTIN(MAXARCS)
      INTEGER COST(MAXARCS),START(MAXARCS),END(MAXARCS)
      INTEGER LIST(MAXNODES),REV_LIST(MAXNODES)
      INTEGER PCOL(MAXNODES),PROW(MAXNODES)
      REAL*8 AVERAGE,FACTOR
      COMMON/ARRAYC/COST/ARRAYS/START/ARRAYE/END
      COMMON/ARRAYFO/FOUT/ARRAYFI/FIN/ARRAYNI/NXTIN
      COMMON/ARRAYRA/ROW_ASSIGNED_TO/ARRAYCA/COL_ASSIGNED_TO
      COMMON/ARRAYPC/PCOL/ARRAYPR/PROW
      COMMON/BK1/M,N,A,ILARGE
      COMMON/BK2/CYCLES,AVERAGE,NUMPHASES,LAMBDA

C ***************************************************************
```

```
C   ******* CHECK VALIDITY OF PARAMETERS PASSED *******

        IF (BEGEPS.LT.1) THEN
          PRINT*,'STARTING VALUE OF EPSILON IS LESS THAN 1'
          PRINT*,'EXECUTION ABORTED'
          STOP
        END IF
        IF (ENDEPS.GT.BEGEPS) THEN
          PRINT*,'PARAMETER ENDEPS IS GREATER THAN PARAMETER BEGEPS'
          PRINT*,'ENDEPS IS SET AT THE DEFAULT VALUE OF 1'
          ENDEPS=1
        END IF
        IF ((FACTOR.LE.1).AND.(BEGEPS.GT.1)) THEN
          PRINT*,'EPSILON REDUCTION FACTOR IS NOT GREATER THAN 1'
          PRINT*,'EXECUTION ABORTED'
          STOP
        END IF
        IF (STARTINCR.LT.1) THEN
          PRINT*,'MIN BIDDING INCREMENT IS LESS THAN 1'
          PRINT*,'STARTINCR IS SET AT THE DEFAULT VALUE OF 1'
          STARTINCR=1
        END IF

C   ******* INITIALIZATION *******

         EPSILON=BEGEPS
         ISMALL=-ILARGE
         LARGEINCR=INT(ILARGE/10)
         THRESH=INT(0.2*M)
         INCRFACTOR=2
         EPS_THRESH=BEGEPS/(FACTOR**3)
         IF (EPS_THRESH.LT.2)  EPS_THRESH=2
         INIT=.FALSE.
         IF (THRESH.GT.100) THRESH=100
X        CYCLES=1
X        AVERAGE=N
         NUMPHASES=1

         DO 10 I=1,N
           PCOL(I)=ISMALL
10       CONTINUE

         FOUT(M+1)=A+1

C ************************************************************
C
C  THIS IMPLEMENTATION OF THE AUCTION ALGORITHM OPERATES IN CYCLES.
C  EACH CYCLE CONSISTS OF ONE BID BY EACH OF THE ROWS THAT ARE
C  UNASSIGNED AT THE START OF THE CYCLE (THESE ROWS ARE STORED IN
C  THE ARRAY LIST(.)). AS THE CYCLE PROGRESSES NEW
C  ROWS BECOME UNASSIGNED; THESE ARE STORED IN LIST(.)
C  AND WILL SUBMIT A BID AT THE NEXT CYCLE.
C
C  THE ALGORITHM FIRST FINDS A FEASIBLE ASSIGNMENT, WHERE ALL PERSONS
```

```
C  ARE ASSIGNED, USING A COMBINED FORWARD/REVERSE AUCTION.
C  THEN THE ALGORITHM USES A MODIFIED REVERSE AUCTION TO TO SATISFY
C  THE REMAINING E-CS CONDITIONS.
C
C ***************************************************************
C
C       START SUBPROBLEM (SCALING PHASE) W/ NEW EPSILON
C
C ***************************************************************

12      CONTINUE

C     INITIALIZE ROW ASSIGNMENT LISTS

        NOLIST=M
        DO 20 I=1,M
           LIST(I)=I
20      CONTINUE

        DO 22 J=1,N
           ROW_ASSIGNED_TO(J)=0
22      CONTINUE

        INCR=STARTINCR
        IF (INCR.GT.EPSILON) INCR=EPSILON
        IF (EPSILON.EQ.1) THRESH=0


C ***************************************************************
C
C       START FORWARD AUCTION CYCLE
C
C ***************************************************************

      IF (EPSILON.LT.EPS_THRESH)  THEN

17      CONTINUE

C  INITIALIZE COUNT OF NEXT LIST OF UNASSIGNED ROWS

        NONEWLIST=0

C  CYCLE THROUGH THE CURRENT LIST OF UNASSIGNED ROWS

        DO 103 I=1,NOLIST
          ROW=LIST(I)
          FSTARC=FOUT(ROW)
          LSTARC=FOUT(ROW+1)-1
          FSTCOL=END(FSTARC)

C  FIRST TAKE CARE OF THE EXCEPTIONAL CASE WHERE ROW HAS ONLY ONE ARC

          IF (FSTARC.EQ.LSTARC) THEN
            PCOL(FSTCOL)=PCOL(FSTCOL)+LARGEINCR
```

```
            PROW(ROW)=COST(FSTARC)-PCOL(FSTCOL)
            OLDROW=ROW_ASSIGNED_TO(FSTCOL)
            ROW_ASSIGNED_TO(FSTCOL)=ROW
            IF (OLDROW.GT.0) THEN
              NONEWLIST=NONEWLIST+1
              LIST(NONEWLIST)=OLDROW
            END IF
            GO TO 103
          END IF

C  NEXT TAKE CARE OF THE REGULAR CASE WHERE ROW HAS MULTIPLE ARCS

          IF (((FSTARC+2).LE.LSTARC).AND.(INIT)) THEN

            BSTARC=BEST_ARC(ROW)
            SBSTARC=SECD_ARC(ROW)
            BSTCOL=END(BSTARC)
            SNDCOL=END(SECD_ARC(ROW))

            MAX1=COST(BSTARC)-PCOL(BSTCOL)
            MAX2=COST(SBSTARC)-PCOL(SNDCOL)
            IF((MAX1.GE.TRDVAL(ROW)).AND.(MAX2.GE.TRDVAL(ROW))) THEN

                IF (MAX1.LT.MAX2) THEN
                  TMAX=MAX1
                  MAX1=MAX2
                  MAX2=TMAX
                  BSTCOL=SNDCOL
                END IF
                GO TO 70

             END IF
          END IF

          SNDARC=FSTARC+1
          SNDCOL=END(SNDARC)
          MAX1=COST(FSTARC)-PCOL(FSTCOL)
          MAX2=COST(SNDARC)-PCOL(SNDCOL)
          IF (MAX1.GE.MAX2) THEN
              BSTARC=FSTARC
              SBSTARC=SNDARC
          ELSE
             TMAX=MAX1
             MAX1=MAX2
             MAX2=TMAX
             BSTARC=SNDARC
             SBSTARC=FSTARC
          END IF
          IF (SNDARC.LT.LSTARC) THEN
            TRDARC=SNDARC+1
            MAX3=ISMALL
            DO 43 CURARC=TRDARC,LSTARC
               CURCOL=END(CURARC)
               TMAX=COST(CURARC)-PCOL(CURCOL)
```

```
                IF (TMAX.GT.MAX2) THEN
                 IF (TMAX.GT.MAX1) THEN
                       SBSTARC=BSTARC
                       BSTARC=CURARC
                       MAX3=MAX2
                       MAX2=MAX1
                       MAX1=TMAX
                  ELSE
                       SBSTARC=CURARC
                       MAX3=MAX2
                       MAX2=TMAX
                 END IF
                ELSE
                  IF (MAX3.LT.TMAX) MAX3=TMAX
                END IF
43            CONTINUE
            END IF

          BEST_ARC(ROW)=BSTARC
          BSTCOL=END(BSTARC)
          SECD_ARC(ROW)=SBSTARC
          TRDVAL(ROW)=MAX3

70        CONTINUE


C  ROW BIDS FOR BSTCOL INCREASING ITS PRICE, AND GETS ASSIGNED
C  TO BSTCOL, WHILE ANY ROW ASSIGNED TO BSTCOL BECOMES UNASSIGNED

            PCOL(BSTCOL)=PCOL(BSTCOL)+MAX1-MAX2+INCR
            PROW(ROW)=MAX2-INCR
            OLDROW=ROW_ASSIGNED_TO(BSTCOL)
            ROW_ASSIGNED_TO(BSTCOL)=ROW
            IF (OLDROW.GT.0) THEN
              NONEWLIST=NONEWLIST+1
              LIST(NONEWLIST)=OLDROW
            END IF

103     CONTINUE

C       ************* END OF A FORWARD AUCTION CYCLE ***************

C  OPTIONALLY COLLECT STATISTICS

X         AVERAGE=(CYCLES*AVERAGE+NOLIST)/(CYCLES+1)
X         CYCLES=CYCLES+1

C  CHECK IF A SWITCH TO MODIFIED REVERSE AUCTION
C  SHOULD BE MADE (IF NONEWLIST EQUALS ZERO).
C  ALSO INCREASE THE MINIMAL BIDDING INCREMENT UP TO A MAXIMUM
C  VALUE OF EPSILON (THIS IS THE ADAPTIVE FEATURE)

          INCR=INCR*INCRFACTOR
          IF (INCR.GT.EPSILON) INCR=EPSILON
```

```
          IF (NONEWLIST.GT.THRESH) THEN
              NOLIST=NONEWLIST
              INIT=.TRUE.
              GO TO 17
          END IF

        ELSE

C ***************************************************************
C
C       START FORWARD AUCTION CYCLE
C
C ***************************************************************

15      CONTINUE

C  INITIALIZE COUNT OF NEXT LIST OF UNASSIGNED ROWS

          NONEWLIST=0

C  CYCLE THROUGH THE CURRENT LIST OF UNASSIGNED ROWS

          DO 100 I=1,NOLIST
            ROW=LIST(I)
            FSTARC=FOUT(ROW)
            LSTARC=FOUT(ROW+1)-1
            FSTCOL=END(FSTARC)

C  FIRST TAKE CARE OF THE EXCEPTIONAL CASE WHERE ROW HAS ONLY ONE ARC

            IF (FSTARC.EQ.LSTARC) THEN
              PCOL(FSTCOL)=PCOL(FSTCOL)+LARGEINCR
              PROW(ROW)=COST(FSTARC)-PCOL(FSTCOL)
              OLDROW=ROW_ASSIGNED_TO(FSTCOL)
              ROW_ASSIGNED_TO(FSTCOL)=ROW
              IF (OLDROW.GT.0) THEN
                NONEWLIST=NONEWLIST+1
                LIST(NONEWLIST)=OLDROW
              END IF
              GO TO 100
            END IF

C  NEXT TAKE CARE OF THE REGULAR CASE WHERE ROW HAS MULTIPLE ARCS

            SNDARC=FSTARC+1
            SNDCOL=END(SNDARC)
            MAX1=COST(FSTARC)-PCOL(FSTCOL)
            MAX2=COST(SNDARC)-PCOL(SNDCOL)
            IF (MAX1.GE.MAX2) THEN
               BSTCOL=FSTCOL
            ELSE
               TMAX=MAX1
               MAX1=MAX2
               MAX2=TMAX
```

```
            BSTCOL=SNDCOL
          END IF
          IF (SNDARC.LT.LSTARC) THEN
            TRDARC=SNDARC+1
            DO 40 CURARC=TRDARC,LSTARC
               CURCOL=END(CURARC)
               TMAX=COST(CURARC)-PCOL(CURCOL)
               IF (TMAX.GT.MAX2) THEN
                 IF (TMAX.GT.MAX1) THEN
                      MAX2=MAX1
                      MAX1=TMAX
                      BSTCOL=CURCOL
                 ELSE
                      MAX2=TMAX
                 END IF
               END IF
40          CONTINUE
          END IF


C  ROW BIDS FOR BSTCOL INCREASING ITS PRICE, AND GETS ASSIGNED
C  TO BSTCOL, WHILE ANY ROW ASSIGNED TO BSTCOL BECOMES UNASSIGNED

          PCOL(BSTCOL)=PCOL(BSTCOL)+MAX1-MAX2+INCR
          PROW(ROW)=MAX2-INCR
          OLDROW=ROW_ASSIGNED_TO(BSTCOL)
          ROW_ASSIGNED_TO(BSTCOL)=ROW
          IF (OLDROW.GT.0) THEN
            NONEWLIST=NONEWLIST+1
            LIST(NONEWLIST)=OLDROW
          END IF

100     CONTINUE

C       ************* END OF A FORWARD AUCTION CYCLE ***************

C  OPTIONALLY COLLECT STATISTICS

X         AVERAGE=(CYCLES*AVERAGE+NOLIST)/(CYCLES+1)
X         CYCLES=CYCLES+1

C  CHECK IF A SWITCH TO MODIFIED REVERSE AUCTION
C  SHOULD BE MADE (IF NONEWLIST IS NO MORE THAN THE THRESHOLD).
C  ALSO INCREASE THE MINIMAL BIDDING INCREMENT UP TO A MAXIMUM
C  VALUE OF EPSILON (THIS IS THE ADAPTIVE FEATURE)

          INCR=INCR*INCRFACTOR
          IF (INCR.GT.EPSILON) INCR=EPSILON
          IF (NONEWLIST.GT.THRESH) THEN
              NOLIST=NONEWLIST
              GO TO 15
          END IF

        END IF
```

```
C ************************************************************
C
C             START OF MODIFIED REVERSE AUCTION
C
C ************************************************************

       IF (EPSILON.GT.1) GOTO 400
       IF (N.EQ.M) GOTO 400

       INCR=EPSILON


C  COMPUTE LAMBDA WHICH IS THE MINIMUM COLUMN PRICE OVER ALL
C  ASSIGNED COLUMNS, AND COMPILE THE LIST OF UNASSIGNED COLUMNS

       LAMBDA=ILARGE
       RNOLIST=0
       DO 310 J=1,N
         ROW=ROW_ASSIGNED_TO(J)
         IF (ROW.GT.0) THEN
           COL_ASSIGNED_TO(ROW)=J
           IF (LAMBDA.GT.PCOL(J)) LAMBDA=PCOL(J)
         ELSE
           RNOLIST=RNOLIST+1
           REV_LIST(RNOLIST)=J
         END IF
310    CONTINUE


       IF (RNOLIST.NE.N-M) THEN
          PRINT*,'NUMBER OF UNASSIGNED ROWS IS WRONG'
          PAUSE
          STOP
       END IF


C  START OF A NEW MODIFIED REVERSE AUCTION CYCLE


315      CONTINUE

C  INITIALIZE COUNT OF NEXT LIST OF UNASSIGNED COLUMNS

        RNONEWLIST=0

C  CYCLE THROUGH THE CURRENT LIST OF UNASSIGNED COLUMNS

        DO 340 J=1,RNOLIST
          COLUMN=REV_LIST(J)
          IF (PCOL(COLUMN).LE.LAMBDA) GO TO 340
          CURARC=FIN(COLUMN)
          NXTARC=NXTIN(CURARC)
          CURROW=START(CURARC)
```

```
C  FIRST TAKE CARE OF THE EXCEPTIONAL CASE WHERE ROW HAS ONLY ONE ARC

           IF (NXTARC.EQ.0) THEN
             MAX1=COST(CURARC)-PROW(CURROW)
             IF (LAMBDA.GE.MAX1-INCR) THEN
                PCOL(COLUMN)=LAMBDA
                GO TO 340
             END IF
             PCOL(COLUMN)=LAMBDA
             PROW(CURROW)=COST(CURARC)-LAMBDA
             OLDCOL=COL_ASSIGNED_TO(CURROW)
             COL_ASSIGNED_TO(CURROW)=COLUMN
             IF (PCOL(OLDCOL).GT.LAMBDA) THEN
               RNONEWLIST=RNONEWLIST+1
               REV_LIST(RNONEWLIST)=OLDCOL
             END IF
             GO TO 340
           END IF

C  NEXT TAKE CARE OF THE REGULAR CASE WHERE ROW HAS MULTIPLE ARCS

           NXTROW=START(NXTARC)
           MAX1=COST(CURARC)-PROW(CURROW)
           MAX2=COST(NXTARC)-PROW(NXTROW)
           IF (MAX1.GE.MAX2) THEN
              BSTROW=CURROW
           ELSE
              TMAX=MAX1
              MAX1=MAX2
              MAX2=TMAX
              BSTROW=NXTROW
           END IF

           CURARC=NXTIN(NXTARC)

330        IF (CURARC.GT.0) THEN
             CURROW=START(CURARC)
             TMAX=COST(CURARC)-PROW(CURROW)
             IF (TMAX.GT.MAX2) THEN
               IF (TMAX.GT.MAX1) THEN
                    MAX2=MAX1
                    MAX1=TMAX
                    BSTROW=CURROW
               ELSE
                    MAX2=TMAX
               END IF
             END IF
             CURARC=NXTIN(CURARC)
             GO TO 330
           END IF


C  COLUMN BIDS FOR BSTROW INCREASING ITS PRICE, AND GETS ASSIGNED
```

```
C  TO BSTROW, WHILE ANY COLUMN ASSIGNED TO BSTROW BECOMES UNASSIGNED


          IF (LAMBDA.GE.MAX1-INCR) THEN
             PCOL(COLUMN)=LAMBDA
             GO TO 340
          END IF
          DELTA=MAX1-MAX2+INCR
          IF (DELTA.GT.MAX1-LAMBDA) DELTA=MAX1-LAMBDA
          PROW(BSTROW)=PROW(BSTROW)+DELTA
          PCOL(COLUMN)=MAX1-DELTA
          OLDCOL=COL_ASSIGNED_TO(BSTROW)
          COL_ASSIGNED_TO(BSTROW)=COLUMN
          IF (PCOL(OLDCOL).GT.LAMBDA) THEN
            RNONEWLIST=RNONEWLIST+1
            REV_LIST(RNONEWLIST)=OLDCOL
          END IF

340    CONTINUE

C      ************* END OF A MODIFIED REVERSE AUCTION CYCLE
***************

C  OPTIONALLY COLLECT STATISTICS

X        AVERAGE=(CYCLES*AVERAGE+RNOLIST)/(CYCLES+1)
X        CYCLES=CYCLES+1

C  CHECK IF THERE ARE STILL 'MANY' UNASSIGNED ELIGIBLE COLUMNS, THAT IS,
C  IF THE NUMBER OF ELIGIBLE UNASSIGNED COLUMNS IS GREATER THAN
C  THE PARAMETER THRESH. IF NOT, REPLACE CURRENT LIST WITH THE NEW LIST,
C  AND GO FOR ANOTHER CYCLE. OTHERWISE, IF EPSILON > 1, REDUCE EPSILON,
C  RESET THE ASSIGNMENT TO EMPTY AND RESTART AUCTION;
C  IF EPSILON = 1 TERMINATE.


         IF (RNONEWLIST.GT.THRESH) THEN
           RNOLIST=RNONEWLIST
           GO TO 315
         END IF


C ************************************************************
C
C                  END OF SUBPROBLEM (SCALING PHASE)
C
C ************************************************************

400    CONTINUE

C ****** IF EPSILON IS 1 TERMINATE ******

        IF (EPSILON.EQ.1) THEN
           RETURN
```

```
       ELSE

C   ELSE REDUCE EPSILON AND UPDATE PARAMETERS FOR THE NEXT SCALING PHASE

           NUMPHASES=NUMPHASES+1
           EPSILON=INT(EPSILON/FACTOR)
           IF (EPSILON.GT.INCR) EPSILON=INT(EPSILON/FACTOR)
           IF ((EPSILON.LT.1).OR.(EPSILON.LT.ENDEPS)) EPSILON=1
           IF (STARTINCR.LT.EPSILON) STARTINCR=FACTOR*STARTINCR
           THRESH=INT(THRESH/FACTOR)


X          PRINT*,'*** END OF A SCALING PHASE; NEW EPSILON=',EPSILON

           GO TO 12
         END IF


         END


      SUBROUTINE SETRAN(ISEED)
      IMPLICIT REAL*8 (A-H,O-Z) , INTEGER*4 (I-N)
C**********************************************************************
C PORTABLE CONGRUENTIAL (UNIFORM) RANDOM NUMBER GENERATOR:
C     NEXT_VALUE = [(7**5) * PREVIOUS_VALUE] MODULO[(2**31)-1]
C
C THIS GENERATOR CONSISTS OF TWO ROUTINES:
C   (1) SETRAN - INITIALIZES CONSTANTS AND SEED
C   (2) RRAN   - GENERATES A REAL RANDOM NUMBER
C
C THE GENERATOR REQUIRES A MACHINE WITH AT LEAST 32 BITS OF PRECISION.
C THE SEED (ISEED) MUST BE IN THE RANGE (1,(2**31)-1).
C**********************************************************************
      COMMON /RANDM/ MULT,MODUL,I15,I16,JRAN
      IF(ISEED.LT.1) STOP 77
      MULT=16807
      MODUL=2147483647
      I15=2**15
      I16=2**16
      JRAN=ISEED
      RETURN
      END
C
      REAL FUNCTION RAN()
      IMPLICIT REAL*4 (A-H,O-Z) , INTEGER*4 (I-N)
C**********************************************************************
C RAN GENERATES A REAL RANDOM NUMBER BETWEEN 0 AND 1
C**********************************************************************
      COMMON /RANDM/ MULT,MODUL,I15,I16,JRAN
      IXHI=JRAN/I16
      IXLO=JRAN-IXHI*I16
      IXALO=IXLO*MULT
      LEFTLO=IXALO/I16
```

```
      IXAHI=IXHI*MULT
      IFULHI=IXAHI+LEFTLO
      IRTLO=IXALO-LEFTLO*I16
      IOVER=IFULHI/I15
      IRTHI=IFULHI-IOVER*I15
      JRAN=((IRTLO-MODUL)+IRTHI*I16)+IOVER
      IF(JRAN.LT.0) JRAN=JRAN+MODUL
      RAN = FLOAT(JRAN)/FLOAT(MODUL)
      RETURN
      END
```

AUCTION-FR

This code implements the combined forward/reverse auction algorithm with $\e$-scaling for the symmetric assignment problem; cf.\ Section 4.2. For good performance, it is frequently not essential to use $\e$-scaling in this code. It is therefore recommended that the code be tried without $\e$-scaling by setting the starting $\e$ to 1.

```
C ***************************************************************
C
C       SAMPLE CALLING PROGRAM FOR COMBINED FORWARD/REVERSE
C                     AUCTION ALGORITHM
C
C   THIS DRIVER CREATES A SYMMETRIC ASSIGNMENT PROBLEM
C   WITH EQUAL NUMBER OF ROWS AND COLUMNS,
C   AND CALLS THE AUCTION_FR SUBROUTINE TO FIND AN
C   ASSIGNMENT OF MAXIMAL VALUE.
C
C ***************************************************************


        PARAMETER(MAXNODES=10000, MAXARCS=100000)

        IMPLICIT NONE
        INTEGER N,NA,A,IA,ILARGE,BEGEPS,ENDEPS,CYCLES
        INTEGER NUMPHASES,STARTINCR
        INTEGER I,J,ARC,NOASS,ICOST,ABSCOST,CURARC
        INTEGER CURCOL,FSTARC,LSTARC,MINCOST,MAXCOST,MCOST
        INTEGER COL_ASSIGNED_TO(MAXNODES),ROW_ASSIGNED_TO(MAXNODES)
        INTEGER FOUT(MAXNODES),FIN(MAXNODES),NXTIN(MAXARCS)
        INTEGER COST(MAXARCS),START(MAXARCS),END(MAXARCS)
        INTEGER PCOL(MAXNODES),PROW(MAXNODES)
        INTEGER PRDARC(MAXNODES)
        REAL*8 FACTOR,TT1,TT2,TCOST,AVERAGE
        COMMON/ARRAYC/COST/ARRAYS/START/ARRAYE/END
        COMMON/ARRAYFO/FOUT/ARRAYFI/FIN/ARRAYNI/NXTIN
        COMMON/ARRAYRA/ROW_ASSIGNED_TO/ARRAYCA/COL_ASSIGNED_TO
        COMMON/ARRAYPC/PCOL/ARRAYPR/PROW
```

```
        COMMON/BK1/N,A,ILARGE/BK2/CYCLES,AVERAGE,NUMPHASES

C ************************************************************
C
C     PROBLEM GENERATION CODE STARTS HERE
C     THE USER MAY REPLACE THIS CODE WITH A CODE THAT READS
C     HIS/HER PROBLEM FROM A FILE
C
C ************************************************************

C     THIS CODE INCLUDES A UNIFORM RANDOM NUMBER GENERATOR
C     WHICH RETURNS A VALUE IN (0,1)

C     INITIALIZE RANDOM GENERATOR

      INTEGER MULT,MODUL,I15,I16,JRAN,ISEED
      REAL RAN
      COMMON /RANDM/ MULT,MODUL,I15,I16,JRAN

      ISEED=13502460
      CALL SETRAN(ISEED)

      PRINT*,'GENERATING A SYMMETRIC ASSIGNMENT PROBLEM'
      PRINT*,'****************************************'

C **** READ THE NUMBER OF ROWS N & THE NUMBER OF ARCS A ****

      PRINT*,'ENTER THE NUMBER OF ROWS (AND COLUMNS)'
      READ*,N
9. PRINT*,'ENTER THE NUMBER OF ARCS PER ROW (>1)'
      READ*,NA
      IF (NA.LT.2) GOTO 5
      PRINT*,'ENTER THE MINIMUM AND THE MAXIMUM COST'
      READ*,MINCOST,MAXCOST

C     THE NUMBER OF ARCS IS N*NA

      A=N*NA

C      THE ARCS INCIDENT TO ROW I ARE FOUT(I) TO FOUT(I+1)-1
C      ALSO, FOR FEASIBILITY EACH ROW IS DIRECTLY CONNECTED
C      WITH THE CORRESPONDING COLUMN

      DO 20 I=1,N
        FOUT(I)=1+(I-1)*NA
20    CONTINUE
      FOUT(N+1)=A+1

      DO 22 I=1,N
        DO 23 ARC=FOUT(I),FOUT(I+1)-1
          START(ARC)=I
23      CONTINUE
22    CONTINUE
```

```
C      GENERATE THE END(ARC) AND COST(ARC) WHICH ARE THE COLUMN
C      AND THE COST COEFFICIENT ASSOCIATED WITH ARC

       DO 25 ARC=1,A
         END(ARC)=1+RAN()*N
         IF ((END(ARC).GT.N).OR.(END(ARC).LT.1)) THEN
           PRINT*,'ERROR IN PROBLEM GENERATION'
           PAUSE
           STOP
         END IF
         COST(ARC)=MINCOST+RAN()*(MAXCOST-MINCOST)
25     CONTINUE

C   MODIFY THE END OF THE LAST ARC OUT OF EACH ROW FOR FEASIBILITY
C   AND SET ITS COST TO -MAXCOST

       DO 30 I=1,N
         END(FOUT(I+1)-1)=I
         COST(FOUT(I+1)-1)=-MAXCOST
30     CONTINUE

C    CONSTRUCT THE FIN() AND NXTIN() ARRAYS

       DO 32 J=1,N
         FIN(J)=0
         PRDARC(J)=0
32     CONTINUE

       DO 33 ARC=1,A
         NXTIN(ARC)=0
         J=END(ARC)
         IF (FIN(J).NE.0) THEN
           NXTIN(PRDARC(J))=ARC
         ELSE
           FIN(J)=ARC
         END IF
         PRDARC(J)=ARC
33     CONTINUE


C ***************************************************************
C
C          PROBLEM GENERATION CODE ENDS HERE
C
C ***************************************************************


C       SCALE THE COST TO WORK WITH INTEGER EPSILON

         MAXCOST=0
         DO 35 IA=1,A
            ABSCOST=IABS(COST(IA))
            IF (ABSCOST.GT.MAXCOST) MAXCOST=ABSCOST
            COST(IA)=COST(IA)*(N+1)
```

```
35      CONTINUE

C *** ILARGE IS A VERY LARGE INTEGER FOR YOUR MACHINE ***
C     THE SCALED COSTS SHOULD BE SIGNIFICANTLY SMALLER THAN
C     ILARGE AND SIGNIFICANTLY LARGER THAN -ILARGE


        ILARGE=2000000000


        IF (MAXCOST.GT.INT(ILARGE/(N+1))) THEN
          PRINT*,'THE COST RANGE IS TOO LARGE FOR INTEGER ARITHMETIC'
          PAUSE
          STOP
        END IF

        MAXCOST=MAXCOST*(N+1)


C       THE FOLLOWING PARAMETERS BEGEPS, FACTOR, ENDEPS, AND STARTINCR
C       ARE PASSED TO THE AUCTION ALGORITHM. VALUES BETWEEN
C        (A) MAXCOST/2 AND 1 FOR BEGEPS
C        (B) 4 AND 6 FOR FACTOR
C        (C) N AND 1 FOR ENDEPS
C        (D) 1 AND BEGEPS FOR STARTINCR
C       HAVE WORKED WELL FOR LARGE SPARSE PROBLEMS.
C       FOR DENSE PROBLEMS IT IS RECOMMENDED THAT
C       BEGEPS BE SET TO A SMALLER VALUE,
C       ENDEPS BE SET TO 1,
C       STARTINCR BE SET TO 1.

C       GENERALLY, THE COMBINED FORWARD/REVERSE ALGORITHM
C       WORKS BEST WITH A SMALLER VALUE OF BEGEPS THAN THE
C       FORWARD ALGORITHM.
C       IT IS WORTH TRYING BEGEPS=1, IN WHICH CASE THERE IS
C       ONLY ONE SCALING PHASE (THAT IS, NO E-SCALING IS USED).

        PRINT*,'*****************************************'
        PRINT*,'MAXIMUM COST IS ',MAXCOST
        PRINT*,'ENTER THE STARTING EPSILON'
        READ*,BEGEPS
        IF (BEGEPS.LT.1) BEGEPS=1
        PRINT*,'ENTER THE EPSILON REDUCTION FACTOR'
        READ*,FACTOR
        ENDEPS=N/10
        IF (ENDEPS.LT.1) ENDEPS=1
        IF (ENDEPS.GT.BEGEPS) ENDEPS=BEGEPS
        STARTINCR=1
        IF (STARTINCR.LT.1) STARTINCR=1

        PRINT*,'*****************************************'
        PRINT*,'STARTING EPSILON = ',BEGEPS
        PRINT*,'EPSILON REDUCTION FACTOR = ',FACTOR
        PRINT*,'THRESHOLD EPSILON BEFORE IT IS SET TO 1 = ',ENDEPS
```

```
        PRINT*,'STARTING MIN BIDDING INCREMENT = ',STARTINCR
        PRINT*,'CALLING FORWARD/REVERSE AUCTION'
        PRINT*,'****************************************'

C     GET STARTING TIME FOR THE MAC II

      TT1 = LONG(362)/60.0

      CALL AUCTION_FR(BEGEPS,FACTOR,ENDEPS,STARTINCR)

C     GET ENDING TIME FOR THE MAC II

      TT2 = LONG(362)/60.0 - TT1
      PRINT *,'FINISHED --- TOTAL CPU TIME', TT2,' SECS'
      PRINT*,'****************************************'

C     *** DISPLAY RESULTS ***

X        WRITE(9,2010) CYCLES
X2010    FORMAT(' NO OF AUCTION CYCLES',I7)
X        WRITE(9,2020) AVERAGE
X2020    FORMAT(' AVERAGE NUMBER OF BIDS PER CYCLE',F9.3)
        WRITE(9,2030) NUMPHASES
2030    FORMAT('NO OF EPSILON SUBPROBLEMS SOLVED =',I7)

C       CHECK OPTIMALITY & CALCULATE COST

        TCOST=0
        DO 40 I=1,N
          J=COL_ASSIGNED_TO(I)
          IF (J.EQ.0) THEN
              PRINT*,'ROW ',I,' IS UNASSIGNED'
          END IF
          IF (ROW_ASSIGNED_TO(J).NE.I) THEN
              PRINT*,'ASSIGNMENT MIXUP: ROW ',I,' COLUMN ',J
          END IF
          FSTARC=FOUT(I)
          LSTARC=FOUT(I+1)-1
          MCOST=-ILARGE
          DO 45 ARC=FSTARC,LSTARC
            CURCOL=END(ARC)
            IF (PROW(I)+PCOL(CURCOL).LT.COST(ARC)-1) THEN
              PRINT*,'1-CS VIOLATED AT ARC ',ARC
            END IF
            IF (CURCOL.EQ.J) THEN
              IF (MCOST.LT.COST(ARC)) THEN
                 MCOST=COST(ARC)
              END IF
            END IF
45        CONTINUE
          TCOST=TCOST+MCOST/(N+1)
          IF (PROW(I)+PCOL(J).NE.MCOST) THEN
            PRINT*,'1-CS VIOLATED AT ROW ',I
          END IF
```

```
40      CONTINUE

        DO 50 J=1,N
           IF (ROW_ASSIGNED_TO(J).EQ.0) THEN
              PRINT*,'COLUMN ',J,' IS UNASSIGNED'
           END IF
50      CONTINUE

        WRITE(9,2100) TCOST
2100    FORMAT(' ASSIGNMENT COST=',F18.2)
        PRINT *, ' PROGRAM ENDED; <CR> TO EXIT '
        PAUSE
        END



C ****************************************************************
C
C     FORWARD/REVERSE AUCTION CODE FOR N BY N ASSIGNMENT PROBLEMS
C
C             WRITTEN BY DIMITRI P. BERTSEKAS
C
C                    DEC. 1990
C
C THIS CODE IMPLEMENTS THE FORWARD/REVERSE AUCTION ALGORITHM
C WITH E-SCALING FOR SYMMETRIC N BY N ASSIGNMENT PROBLEMS.
C IT SOLVES A SEQUENCE OF SUBPROBLEMS AND DECREASES
C EPSILON BY A CONSTANT FACTOR BETWEEN SUBPROBLEMS.
C THIS VERSION CORRESPONDS TO A GAUSS-SEIDEL MODE
C AND SOLVES EPSILON SUBPROBLEMS INEXACTLY.
C
C THE CODE IS AN IMPROVED VERSION OF AN EARLIER (SEPT. 1985)
C AUCTION CODE WITH E-SCALING WRITTEN BY DIMITRI P. BERTSEKAS
C
C THE CODE TREATS THE PROBLEM AS A MAXIMIZATION PROBLEM.
C TO SOLVE A MINIMIZATION PROBLEM, REVERSE THE SIGN OF THE
C ARC COSTS PRIOR TO CALLING AUCTION, AND REVERSE AGAIN
C THE SIGN OF THE OPTIMAL COST UPON RETURN FROM AUCTION.
C THIS CODE ALLOWS MULTIPLE ARCS BETWEEN A ROW AND A COLUMN.
C
C THIS VERSION OF THE AUCTION ALGORITHM IS ADAPTIVE. HERE THE MINIMAL
C BIDDING INCREMENT (STORED IN THE VARIABLE INCR) MAY BE SMALLER THAN
C EPSILON. FOR EVERY SUBPROBLEM, INCR STARTS AT THE PARAMETER VALUE
C STARTINCR (WHICH IS PASSED TO THE AUCTION ROUTINE) AND IS INCREASED
C BY A FACTOR OF 2 AT THE END OF EACH CYCLE, UP TO A MAXIMUM VALUE OF
C EPSILON. THIS ADAPTIVE FEATURE IS PARTICULARLY EFFECTIVE FOR
C DENSE PROBLEMS. IT CAN BE DEFEATED BY SELECTING STARTINCR=BEGEPS
C
C THIS CODE MAY FAIL DUE TO INTEGER OVERFLOW IF THE NUMBER OF NODES
C OR THE COST RANGE (OR BOTH) ARE LARGE. TO CORRECT THIS SITUATION,
C THE PRICES AND OTHER RELATED VARIABLES (MAX1,MAX2,TMAX ETC) SHOULD
C BE DECLARED AS DOUBLE PRECISION REALS.
C ****************************************************************
```

```
C
C THE USER MUST SUPPLY THE FOLLOWING PROBLEM DATA IN FORWARD STAR FORMAT
C (THAT IS, ALL ARCS OF THE SAME ROW ARE NUMBERED CONSECUTIVELY):
C       N=NUMBER OF ROWS (EQUALS NUMBER OF COLUMNS)
C       A=NUMBER OF ARCS
C       FOUT(ROW)=FIRST ARC COMING OUT OF ROW
C       FIN(COL)=FIRST ARC COMING INTO COL
C       NXTIN(ARC)=NEXT ARC INCIDENT TO THE SAME COLUMN AS ARC
C       COST(ARC)=COST OF ARC
C       START(ARC)=ROW CORRESPONDING TO ARC
C       END(ARC)=COLUMN CORRESPONDING TO ARC
C
C AND THE FOLLOWING PARAMETERS FOR THE AUCTION ALGORITHM:
C       BEGEPS=STARTING VALUE OF EPSILON (MUST BE NO LESS THAN 1)
C       ENDEPS=FINAL VALUE OF EPSILON BEFORE IT IS SET TO 1
C       FACTOR=FACTOR BY WHICH EPSILON IS DECREASED BETWEEN SUBPROBLEMS
C       STARTINCR=THE STARTING VALUE OF THE BIDDING INCREMENT
C       ENDEPS SHOULD NOT EXCEED BEGEPS.
C       FACTOR MUST BE GREATER THAN 1 (UNLESS BEGEPS=1).
C
C FOUT(.) IS AN ARRAY OF LENGTH N.
C COST(.),END(.) ARE ARRAYS OF LENGTH A.
C
C THE SOLUTION IS CONTAINED IN THE ARRAY ROW_ASSIGNED_TO(.) WHERE
C    ROW_ASSIGNED_TO(COL) GIVES THE ROW ASSIGNED TO COL.
C ALSO COL_ASSIGNED_TO(ROW) GIVES THE COLUMN ASSIGNED TO ROW.
C
C THIS ALGORITHM DOES NOT CHECK FOR INFEASIBILITY OF THE PROBLEM.
C TO MAKE SURE THE PROBLEM IS FEASIBLE THE USER MAY ADD
C    ADDITIONAL VERY SMALL COST ARCS.
C
C ****************************************************************
C ALL PROBLEM DATA ARE INTEGER
C ****************************************************************


        SUBROUTINE AUCTION_FR(BEGEPS,FACTOR,ENDEPS,STARTINCR)

        PARAMETER(MAXNODES=10000, MAXARCS=100000)

        IMPLICIT NONE
        INTEGER A,K,N,I,J,M,CURARC,CURCOL,CURROW
        INTEGER THRESH,INCR,STARTINCR,INCRFACTOR
        INTEGER NOLIST,RNOLIST,NONEWLIST,RNONEWLIST
        INTEGER ROW,COLUMN, BSTROW,BSTCOL
        INTEGER FSTARC,FSTCOL,NXTARC,NXTROW,LSTARC,SNDARC,SNDCOL
        INTEGER MAX1,MAX2,TMAX,TMIN,TRDARC,EPSILON,BEGEPS,ENDEPS
        INTEGER ISMALL,ILARGE,LARGEINCR,CYCLES
        INTEGER NUMPHASES,OLDROW,OLDCOL
        INTEGER COL_ASSIGNED_TO(MAXNODES),ROW_ASSIGNED_TO(MAXNODES)
        INTEGER FOUT(MAXNODES),FIN(MAXNODES),NXTIN(MAXARCS)
        INTEGER COST(MAXARCS),START(MAXARCS),END(MAXARCS)
        INTEGER LIST(MAXNODES),REV_LIST(MAXNODES)
        INTEGER PCOL(MAXNODES),PROW(MAXNODES)
```

```
      REAL*8 AVERAGE,FACTOR
      LOGICAL READY_SWITCH,SWITCH
      COMMON/ARRAYC/COST/ARRAYS/START/ARRAYE/END
      COMMON/ARRAYFO/FOUT/ARRAYFI/FIN/ARRAYNI/NXTIN
      COMMON/ARRAYRA/ROW_ASSIGNED_TO/ARRAYCA/COL_ASSIGNED_TO
      COMMON/ARRAYPC/PCOL/ARRAYPR/PROW
      COMMON/BK1/N,A,ILARGE/BK2/CYCLES,AVERAGE,NUMPHASES

C ************************************************************

C   ******* CHECK VALIDITY OF PARAMETERS PASSED *******

      IF (BEGEPS.LT.1) THEN
        PRINT*,'STARTING VALUE OF EPSILON IS LESS THAN 1'
        PRINT*,'EXECUTION ABORTED'
        STOP
      END IF
      IF (ENDEPS.GT.BEGEPS) THEN
        PRINT*,'PARAMETER ENDEPS IS GREATER THAN PARAMETER BEGEPS'
        PRINT*,'ENDEPS IS SET AT THE DEFAULT VALUE OF 1'
        ENDEPS=1
      END IF
      IF ((FACTOR.LE.1).AND.(BEGEPS.GT.1)) THEN
        PRINT*,'EPSILON REDUCTION FACTOR IS NOT GREATER THAN 1'
        PRINT*,'EXECUTION ABORTED'
        STOP
      END IF
      IF (STARTINCR.LT.1) THEN
        PRINT*,'MIN BIDDING INCREMENT IS LESS THAN 1'
        PRINT*,'STARTINCR IS SET AT THE DEFAULT VALUE OF 1'
        STARTINCR=1
      END IF

C   ******* INITIALIZATION *******

      EPSILON=BEGEPS
      ISMALL=-ILARGE
      LARGEINCR=INT(ILARGE/10)
      THRESH=INT(0.2*N)
      INCRFACTOR=2
      IF (THRESH.GT.100) THRESH=100
X      CYCLES=1
X      AVERAGE=N
       NUMPHASES=1

C     INITIALIZE FORWARD/REVERSE SWITCH PARAMETERS

       READY_SWITCH=.FALSE.
       SWITCH=.TRUE.


       DO 10 J=1,N
         PCOL(J)=0
10     CONTINUE
```

```
         FOUT(N+1)=A+1

C ***************************************************************
C
C  THIS IMPLEMENTATION OF THE AUCTION ALGORITHM OPERATES IN CYCLES.
C  EACH FORWARD AUCTION CYCLE CONSISTS OF ONE BID BY EACH OF THE ROWS
C  THAT ARE UNASSIGNED AT THE START OF THE CYCLE (THESE ROWS ARE
C  STORED IN THE ARRAY LIST(.)). AS THE CYCLE PROGRESSES NEW
C  ROWS BECOME UNASSIGNED; THESE ARE STORED IN LIST(.)
C  AND WILL SUBMIT A BID AT THE NEXT CYCLE.
C  REVERSE AUCTION CYCLES ARE STRUCTURED SIMILARLY.
C
C ***************************************************************
C
C       START SUBPROBLEM (SCALING PHASE) W/ NEW EPSILON
C
C ***************************************************************
C
12      CONTINUE

C     INITIALIZE ROW AND COLUMN ASSIGNMENT LISTS

        NOLIST=N
        DO 20 I=1,N
           COL_ASSIGNED_TO(I)=0
           LIST(I)=I
20      CONTINUE

        RNOLIST=N
        DO 22 J=1,N
           ROW_ASSIGNED_TO(J)=0
           REV_LIST(J)=J
22      CONTINUE

        INCR=STARTINCR
        IF (INCR.GT.EPSILON) INCR=EPSILON
        IF (EPSILON.EQ.1) THRESH=0

C ***************************************************************
C
C       START FORWARD AUCTION CYCLE
C
C ***************************************************************

15      CONTINUE

C  INITIALIZE COUNT OF NEXT LIST OF UNASSIGNED ROWS

        NONEWLIST=0

C  CYCLE THROUGH THE CURRENT LIST OF UNASSIGNED ROWS

        DO 100 I=1,NOLIST
```

```
          ROW=LIST(I)
          IF (COL_ASSIGNED_TO(ROW).GT.0) GO TO 100
          FSTARC=FOUT(ROW)
          LSTARC=FOUT(ROW+1)-1
          FSTCOL=END(FSTARC)

C  FIRST TAKE CARE OF THE EXCEPTIONAL CASE WHERE ROW HAS ONLY ONE ARC

          IF (FSTARC.EQ.LSTARC) THEN
            PCOL(FSTCOL)=PCOL(FSTCOL)+LARGEINCR
            PROW(ROW)=COST(FSTARC)-PCOL(FSTCOL)
            OLDROW=ROW_ASSIGNED_TO(FSTCOL)
            ROW_ASSIGNED_TO(FSTCOL)=ROW
            COL_ASSIGNED_TO(ROW)=FSTCOL
            IF (OLDROW.GT.0) THEN
              COL_ASSIGNED_TO(OLDROW)=0
              NONEWLIST=NONEWLIST+1
              LIST(NONEWLIST)=OLDROW
            END IF
            GO TO 100
          END IF

C  NEXT TAKE CARE OF THE REGULAR CASE WHERE ROW HAS MULTIPLE ARCS

          SNDARC=FSTARC+1
          SNDCOL=END(SNDARC)
          MAX1=COST(FSTARC)-PCOL(FSTCOL)
          MAX2=COST(SNDARC)-PCOL(SNDCOL)
          IF (MAX1.GE.MAX2) THEN
             BSTCOL=FSTCOL
          ELSE
             TMAX=MAX1
             MAX1=MAX2
             MAX2=TMAX
             BSTCOL=SNDCOL
          END IF
          IF (SNDARC.LT.LSTARC) THEN
            TRDARC=SNDARC+1
            DO 40 CURARC=TRDARC,LSTARC
               CURCOL=END(CURARC)
               TMAX=COST(CURARC)-PCOL(CURCOL)
               IF (TMAX.GT.MAX2) THEN
                 IF (TMAX.GT.MAX1) THEN
                      MAX2=MAX1
                      MAX1=TMAX
                      BSTCOL=CURCOL
                 ELSE
                      MAX2=TMAX
                 END IF
               END IF
40          CONTINUE
          END IF
```

```
C  ROW BIDS FOR BSTCOL INCREASING ITS PRICE, AND GETS ASSIGNED
C  TO BSTCOL, WHILE ANY ROW ASSIGNED TO BSTCOL BECOMES UNASSIGNED

          PCOL(BSTCOL)=PCOL(BSTCOL)+MAX1-MAX2+INCR
          PROW(ROW)=MAX2-INCR
          COL_ASSIGNED_TO(ROW)=BSTCOL
          OLDROW=ROW_ASSIGNED_TO(BSTCOL)
          ROW_ASSIGNED_TO(BSTCOL)=ROW
          IF (OLDROW.GT.0) THEN
            COL_ASSIGNED_TO(OLDROW)=0
            NONEWLIST=NONEWLIST+1
            LIST(NONEWLIST)=OLDROW
          END IF

100     CONTINUE

C       ************* END OF A FORWARD AUCTION CYCLE ***************

C  OPTIONALLY COLLECT STATISTICS

X         AVERAGE=(CYCLES*AVERAGE+NOLIST)/(CYCLES+1)
X         CYCLES=CYCLES+1

C  CHECK IF THERE ARE STILL 'MANY' UNASSIGNED ROWS, THAT IS, IF THE
C  NUMBER OF UNASSIGNED ROWS IS GREATER THAN
C  THE PARAMETER THRESH. IF NOT, REPLACE CURRENT LIST WITH THE NEW LIST,
C  AND GO FOR ANOTHER CYCLE. OTHERWISE, IF EPSILON > 1, REDUCE EPSILON,
C  RESET THE ASSIGNMENT TO EMPTY AND RESTART AUCTION;
C  IF EPSILON = 1 TERMINATE.
C  ALSO INCREASE THE MINIMAL BIDDING INCREMENT UP TO A MAXIMUM
C  VALUE OF EPSILON (THIS IS THE ADAPTIVE FEATURE)


          INCR=INCR*INCRFACTOR
          IF (INCR.GT.EPSILON) INCR=EPSILON
          IF (NONEWLIST.GT.THRESH) THEN
              IF (SWITCH) THEN
                NOLIST=NONEWLIST
                GO TO 115
              END IF
              IF (NONEWLIST.LT.NOLIST) READY_SWITCH=.TRUE.
              IF ((NONEWLIST.EQ.NOLIST).AND.(READY_SWITCH)) THEN
                READY_SWITCH=.FALSE.
                GO TO 115
              ELSE
                NOLIST=NONEWLIST
                GO TO 15
              END IF
          ELSE
              GO TO 300
          END IF


C ************************************************************
```

```
C
C        START REVERSE AUCTION CYCLE
C
C ***************************************************************

115      CONTINUE

C  INITIALIZE COUNT OF NEXT LIST OF UNASSIGNED COLUMNS

         RNONEWLIST=0

C  CYCLE THROUGH THE CURRENT LIST OF UNASSIGNED COLUMNS

         DO 200 J=1,RNOLIST
           COLUMN=REV_LIST(J)
           IF (ROW_ASSIGNED_TO(COLUMN).GT.0) GO TO 200
           CURARC=FIN(COLUMN)
           NXTARC=NXTIN(CURARC)
           CURROW=START(CURARC)

C  FIRST TAKE CARE OF THE EXCEPTIONAL CASE WHERE ROW HAS ONLY ONE ARC

           IF (NXTARC.EQ.0) THEN
             PROW(CURROW)=PROW(CURROW)+LARGEINCR
             PCOL(COLUMN)=COST(CURARC)-PROW(CURROW)
             OLDCOL=COL_ASSIGNED_TO(CURROW)
             COL_ASSIGNED_TO(CURROW)=COLUMN
             ROW_ASSIGNED_TO(COLUMN)=CURROW
             IF (OLDCOL.GT.0) THEN
               ROW_ASSIGNED_TO(OLDCOL)=0
               RNONEWLIST=RNONEWLIST+1
               REV_LIST(RNONEWLIST)=OLDCOL
             END IF
             GO TO 200
           END IF

C  NEXT TAKE CARE OF THE REGULAR CASE WHERE ROW HAS MULTIPLE ARCS

           NXTROW=START(NXTARC)
           MAX1=COST(CURARC)-PROW(CURROW)
           MAX2=COST(NXTARC)-PROW(NXTROW)
           IF (MAX1.GE.MAX2) THEN
              BSTROW=CURROW
           ELSE
              TMAX=MAX1
              MAX1=MAX2
              MAX2=TMAX
              BSTROW=NXTROW
           END IF

           CURARC=NXTIN(NXTARC)

130        IF (CURARC.GT.0) THEN
             CURROW=START(CURARC)
```

```
            TMAX=COST(CURARC)-PROW(CURROW)
            IF (TMAX.GT.MAX2) THEN
              IF (TMAX.GT.MAX1) THEN
                   MAX2=MAX1
                   MAX1=TMAX
                   BSTROW=CURROW
              ELSE
                   MAX2=TMAX
              END IF
            END IF
            CURARC=NXTIN(CURARC)
            GO TO 130
          END IF


C  COLUMN BIDS FOR BSTROW INCREASING ITS PRICE, AND GETS ASSIGNED
C  TO BSTROW, WHILE ANY COLUMN ASSIGNED TO BSTROW BECOMES UNASSIGNED


          PROW(BSTROW)=PROW(BSTROW)+MAX1-MAX2+INCR
          PCOL(COLUMN)=MAX2-INCR
          ROW_ASSIGNED_TO(COLUMN)=BSTROW
          OLDCOL=COL_ASSIGNED_TO(BSTROW)
          COL_ASSIGNED_TO(BSTROW)=COLUMN
          IF (OLDCOL.GT.0) THEN
            ROW_ASSIGNED_TO(OLDCOL)=0
            RNONEWLIST=RNONEWLIST+1
            REV_LIST(RNONEWLIST)=OLDCOL
          END IF

200    CONTINUE

C      ************* END OF A REVERSE AUCTION CYCLE ***************

C  OPTIONALLY COLLECT STATISTICS

X        AVERAGE=(CYCLES*AVERAGE+RNOLIST)/(CYCLES+1)
X        CYCLES=CYCLES+1

C  CHECK IF THERE ARE STILL 'MANY' UNASSIGNED COLUMNS, THAT IS, IF THE
C  NUMBER OF UNASSIGNED COLUMNS IS GREATER THAN
C  THE PARAMETER THRESH. IF NOT, REPLACE CURRENT LIST WITH THE NEW LIST,
C  AND GO FOR ANOTHER CYCLE. OTHERWISE, IF EPSILON > 1, REDUCE EPSILON,
C  RESET THE ASSIGNMENT TO EMPTY AND RESTART AUCTION;
C  IF EPSILON = 1 TERMINATE.
C  ALSO INCREASE THE MINIMAL BIDDING INCREMENT UP TO A MAXIMUM
C  VALUE OF EPSILON (THIS IS THE ADAPTIVE FEATURE)


         INCR=INCR*INCRFACTOR
         IF (INCR.GT.EPSILON) INCR=EPSILON
         IF (RNONEWLIST.GT.THRESH) THEN
              IF (SWITCH) THEN
                SWITCH=.FALSE.
```

```
                RNOLIST=RNONEWLIST
                GO TO 15
              END IF
              IF (RNONEWLIST.LT.RNOLIST) READY_SWITCH=.TRUE.
              IF ((RNONEWLIST.EQ.RNOLIST).AND.(READY_SWITCH)) THEN
                READY_SWITCH=.FALSE.
                GO TO 15
              ELSE
                RNOLIST=RNONEWLIST
                GO TO 115
              END IF
          ELSE
              GO TO 300
          END IF


C ************************************************************
C
C             END OF SUBPROBLEM (SCALING PHASE)
C
C ************************************************************


300     CONTINUE

C ****** IF EPSILON IS 1 TERMINATE ******

        IF (EPSILON.EQ.1) THEN
           RETURN
        ELSE

C   ELSE REDUCE EPSILON AND UPDATE PARAMETERS FOR THE NEXT SCALING PHASE

           NUMPHASES=NUMPHASES+1
           EPSILON=INT(EPSILON/FACTOR)
           IF (EPSILON.GT.INCR) EPSILON=INT(EPSILON/FACTOR)
           IF ((EPSILON.LT.1).OR.(EPSILON.LT.ENDEPS)) EPSILON=1
           IF (STARTINCR.LT.EPSILON) STARTINCR=FACTOR*STARTINCR
           THRESH=INT(THRESH/FACTOR)

X          PRINT*,'*** END OF A SCALING PHASE; NEW EPSILON=',EPSILON

           GO TO 12
         END IF

         END




       SUBROUTINE SETRAN(ISEED)
       IMPLICIT REAL*8 (A-H,O-Z) , INTEGER*4 (I-N)
C*****************************************************************
C PORTABLE CONGRUENTIAL (UNIFORM) RANDOM NUMBER GENERATOR:
```

```
C     NEXT_VALUE = [(7**5) * PREVIOUS_VALUE] MODULO[(2**31)-1]
C
C THIS GENERATOR CONSISTS OF TWO ROUTINES:
C   (1) SETRAN - INITIALIZES CONSTANTS AND SEED
C   (2) RRAN   - GENERATES A REAL RANDOM NUMBER
C
C THE GENERATOR REQUIRES A MACHINE WITH AT LEAST 32 BITS OF PRECISION.
C THE SEED (ISEED) MUST BE IN THE RANGE (1,(2**31)-1).
C*********************************************************************
      COMMON /RANDM/ MULT,MODUL,I15,I16,JRAN
      IF(ISEED.LT.1) STOP 77
      MULT=16807
      MODUL=2147483647
      I15=2**15
      I16=2**16
      JRAN=ISEED
      RETURN
      END
C
      REAL FUNCTION RAN()
      IMPLICIT REAL*4 (A-H,O-Z) , INTEGER*4 (I-N)
C*********************************************************************
C RAN GENERATES A REAL RANDOM NUMBER BETWEEN 0 AND 1
C*********************************************************************
      COMMON /RANDM/ MULT,MODUL,I15,I16,JRAN
      IXHI=JRAN/I16
      IXLO=JRAN-IXHI*I16
      IXALO=IXLO*MULT
      LEFTLO=IXALO/I16
      IXAHI=IXHI*MULT
      IFULHI=IXAHI+LEFTLO
      IRTLO=IXALO-LEFTLO*I16
      IOVER=IFULHI/I15
      IRTHI=IFULHI-IOVER*I15
      JRAN=((IRTLO-MODUL)+IRTHI*I16)+IOVER
      IF(JRAN.LT.0) JRAN=JRAN+MODUL
      RAN = FLOAT(JRAN)/FLOAT(MODUL)
      RETURN
      END
```

```
*************************************************************************
*

NAUCTION_SP: Combined
Naive Auction and  Sequential Shortest Path Code

*************************************************************************
*

This code implements the
sequential shortest path method for the assignment problem, preceded by an
```

extensive initialization using the naive auction algorithm (cf.\ Section 1.2.4). The code is quite similar in structure and performance to a code of the author [Ber81] and to the code of [JoV86] and [JoV87]. These codes also combined a naive auction initialization with the sequential shortest path method.

```
C ***************************************************************
C
C COMBINED NAIVE AUCTION AND SEQUENTIAL SHORTEST PATH METHOD
C            FOR SYMMETRIC N X N ASSIGNMENT PROBLEMS
C                 FINDS AN OPTIMAL ASSIGNMENT
C
C               WRITTEN BY DIMITRI P. BERTSEKAS
C
C ***************************************************************
C
C USER MUST SUPPLY THE FOLLOWING PROBLEM DATA
C N=NUMBER OF ROWS AND NUMBER OF COLUMNS
C A=NUMBER OF ARCS
C FOUT(ROW)=1ST ARC COMING OUT OF ROW
C COST(ARC)=COST OF ARC
C END(ARC)=COLUMN CORRESPONDING TO ARC
C
C FOUT(.) IS AN N - LENGTH ARRAY
C COST(.),END(.),NXTEND(.) ARE ARRAYS OF LENGTH A
C THE OPTIMAL ASSIGNMENT IS CONTAINED IN THE ARRAY ASSIGN(.) WHERE
C    ASSIGN(COL) IS THE ROW ASSIGNED TO COL
C THIS ALGORITHM DOES NOT CHECK FOR INFEASIBILITY OF THE PROBLEM
C TO MAKE SURE THE PROBLEM IS FEASIBLE THE USER MAY ADD
C    ADDITIONAL VERY HIGH COST ARCS
C
C THIS CODE ALLOWS MULTIPLE ARCS BETWEEN A ROW AND A COLUMN.
C
C ***************************************************************
C ALL PROBLEM DATA ARE INTEGER
C ***************************************************************
C
        IMPLICIT INTEGER (A-Z)
        INTEGER AUCTNUM,A,STARC,ENDARC,CURROW,ARC,COUNT
        INTEGER HCOUNT,ROW,FSTARC,FSTCOL,SNDARC,SNDCOL,TMAX,BSTCOL
        INTEGER TMARG,TA,ROWPR,OLMARG,PRICE,TPRICE,CURCOL,DUMMY
        INTEGER FOUT(6000),PCOL(6000),PROW(6000),MARG(6000)
        INTEGER ASSIGN(6000),COLLAB(6000),ROWLAB(6000)
        INTEGER LIST(6000),SCAN(6000)
        INTEGER COST(70000),END(70000)
        LOGICAL MAXSET
        REAL THRESH,RAND,TT,TIMER,TCOST

C
C ***************************************************************
C
C     PROBLEM GENERATION CODE STARTS HERE
C     THE USER MAY REPLACE THIS CODE WITH A CODE THAT READS
C     HIS/HER PROBLEM FROM A FILE
```

```
C
C ************************************************************
C
C   RANDOM GENERATOR STUFF
C
      INTEGER MULT,MODUL,I15,I16,JRAN,ISEED
      REAL RAN
      COMMON /RANDM/ MULT,MODUL,I15,I16,JRAN
C     UNIFORM RANDOM NUMBER GENERATOR WHICH RETURNS A VALUE IN (0,1)
C
C     INITIALIZE RANDOM GENERATOR
C
      ISEED=13502460
      CALL SETRAN(ISEED)

      PRINT*,'GENERATING A SYMMETRIC ASSIGNMENT PROBLEM'
      PRINT*,'****************************************'

C **** READ THE NUMBER OF ROWS N & THE NUMBER OF ARCS A ****

      PRINT*,'ENTER THE NUMBER OF ROWS (AND COLUMNS)'
      READ*,N
10.   PRINT*,'ENTER THE NUMBER OF ARCS PER ROW (>1)'
      READ*,NA
      IF (NA.LT.2) GOTO 5
      PRINT*,'ENTER THE MINIMUM AND THE MAXIMUM COST'
      READ*,MINCOST,MAXCOST

C     THE NUMBER OF ARCS IS N*NA

      A=N*NA

C      THE ARCS INCIDENT TO ROW I ARE FOUT(I) TO FOUT(I+1)-1
C      FOR FEASIBILITY EACH ROW IS DIRECTLY CONNECTED
C      WITH THE CORRESPONDING COLUMN

      DO 20 I=1,N
        FOUT(I)=1+(I-1)*NA
20    CONTINUE

C **** GENERATE THE END(IA) AND COST(IA) WHICH ARE THE COLUMN
C      AND THE COST COEFFICIENT ASSOCIATED WITH ARC IA ****

      DO 25 IA=1,A
        END(IA)=1+RAN()*N
        IF ((END(IA).GT.N).OR.(END(IA).LT.1)) THEN
          PRINT*,'ERROR IN PROBLEM GENERATION'
          PAUSE
          STOP
        END IF
        COST(IA)=MINCOST+RAN()*(MAXCOST-MINCOST)
25    CONTINUE

C   MODIFY THE END OF THE LAST ARC OUT OF EACH ROW FOR FEASIBILITY
```

```
C   AND SET ITS COST TO -MAXCOST

      DO 30 I=1,N
        END(FOUT(I+1)-1)=I
        COST(FOUT(I+1)-1)=-MAXCOST
30    CONTINUE

C
C ***************************************************************
C
C          PROBLEM GENERATION CODE ENDS HERE
C
C ***************************************************************
C

C       MAIN ALGORITHM BEGINS HERE

C
C       THE ALGORITHM INTERNALLY TREATS THE PROBLEM AS A
C       MINIMIZATION PROBLEM.
C       TO SOLVE AN ASSIGNMENT MAXIMIZATION PROBLEM, SET
C       THE FOLLOWING PARAMETER MAXSET TO TRUE, ELSE SET IT TO FALSE

        MAXSET=.TRUE.

        IF (MAXSET) THEN
          DO 35 IA=1,A
            COST(IA)=-COST(IA)
35        CONTINUE
        END IF


        PRINT*,'NO OF ROWS (AND COLUMNS):',N
        PRINT*,'NO OF ARCS:',A
        PRINT*,'COMBINED NAIVE AUCTION AND SEQ. SH. PATH METHOD'
        PRINT *,'*******************************************'
        TIMER = LONG(362)

        ISIMPL=0
        IPRC=0
        ISMALL=-20000000
        ILARGE=-ISMALL
        COUNT=0
        HCOUNT=0
        FOUT(N+1)=A+1

C       THE FOLLOWING INITIALIZATION UP TO THE START OF THE
C       NAIVE AUCTION CYCLES WAS SUGGESTED BY JONKER AND VOLGENANT

C       INITIALIZE COLUMN PRICES

        DO 105 J=1,N
           PCOL(J)=ILARGE
105     CONTINUE
```

```
C       FIND BEST ROW FOR EACH COLUMN

          DO 120 I=1,N
            PROW(I)=0
            ROWLAB(I)=0
            FSTARC=FOUT(I)
            LSTARC=FOUT(I+1)-1
            DO 110 ARC=FSTARC,LSTARC
              J=END(ARC)
              IF (COST(ARC).LT.PCOL(J)) THEN
                PCOL(J)=COST(ARC)
                ASSIGN(J)=I
              END IF
110         CONTINUE
120       CONTINUE

C      CONSTRUCT ROW ASSIGNMENT AND DEASSIGN MULTIPLY ASSIGNED ROWS

         DO 130 J=1,N
            J0=N-J+1
            I=ASSIGN(J0)
            IF (ROWLAB(I).NE.0) THEN
              ROWLAB(I)=-ABS(ROWLAB(I))
              ASSIGN(J0)=0
            ELSE
              ROWLAB(I)=J0
            END IF
130       CONTINUE

C      CONSTRUCT CANDIDATE LIST AND CORRECT COLUMN PRICES


         NEWNOL=0
         DO 150 I=1,N
           CURCOL=ROWLAB(I)
           IF (CURCOL.EQ.0) THEN
             NEWNOL=NEWNOL+1
             LIST(NEWNOL)=I
             GOTO 150
           END IF
           IF (CURCOL.LT.0) THEN
             ROWLAB(I)=-CURCOL
           ELSE
             MAX2=ISMALL
             FSTARC=FOUT(I)
             LSTARC=FOUT(I+1)-1
             DO 140 ARC=FSTARC,LSTARC
               J=END(ARC)
               IF (J.NE.CURCOL) THEN
                 IF (PCOL(J)-COST(ARC).GT.MAX2) THEN
                    MAX2=PCOL(J)-COST(ARC)
                 END IF
               END IF
```

```
140         CONTINUE
            PROW(I)=MAX2
            PCOL(CURCOL)=PCOL(CURCOL)+MAX2
          END IF
150     CONTINUE


        IF (NEWNOL.EQ.0) GO TO 2000

C      ************************************************************

C       END OF INITIALIZATION; DO A NUMBER OF AUCTION CYCLES
C       WHICH IS SET BELOW IN THE PARAMETER AUCTNUM BASED ON THE
C       SPARSITY OF THE PROBLEM

        IF (N*N.LT.10*A) AUCTNUM=2
        IF ((N*N.GE.10*A).AND.(N*N.LT.25*A)) AUCTNUM=3
        IF ((N*N.GE.25*A).AND.(N*N.LT.50*A)) AUCTNUM=4
        IF ((N*N.GE.50*A).AND.(N*N.LT.100*A)) AUCTNUM=5
        IF (N*N.GE.100*A) AUCTNUM=6

C       TO RUN A PURE SEQUENTIAL SHORTEST PATH METHOD SET
C       AUCTNUM=0

        IF (AUCTNUM.EQ.0) GOTO 1000

C      START OF A NEW CYCLE

80      NOLIST=NEWNOL
        I=1
               NEWNOL=0

C       TAKE UP A ROW FOR ITERATION

100     ROW=LIST(I)
        I=I+1

        FSTARC=FOUT(ROW)
        LSTARC=FOUT(ROW+1)-1
        BSTCOL=END(FSTARC)
        IF (FSTARC.EQ.LSTARC) THEN
           OLDROW=ASSIGN(BSTCOL)
           ASSIGN(BSTCOL)=ROW
           ROWLAB(ROW)=BSTCOL
           PROW(ROW)=ISMALL
           PCOL(BSTCOL)=ISMALL+COST(FSTARC)
           IF (OLDROW.GT.0) THEN
             I=I-1
             LIST(I)=OLDROW
             ROWLAB(OLDROW)=0
           END IF
           ISIMPL=ISIMPL+1
           GO TO 100
        END IF
```

```
      SNDARC=FSTARC+1
      SNDCOL=END(SNDARC)
      MAX1=PCOL(BSTCOL)-COST(FSTARC)
      MAX2=PCOL(SNDCOL)-COST(SNDARC)
      IF (MAX1.LT.MAX2) THEN
         TMAX=MAX1
         MAX1=MAX2
         MAX2=TMAX
         FSTCOL=BSTCOL
         BSTCOL=SNDCOL
         SNDCOL=FSTCOL
      END IF
      IF (SNDARC.LT.LSTARC) THEN
       TRDARC=SNDARC+1
       DO 108 ARC=TRDARC,LSTARC
         CURCOL=END(ARC)
         TMAX=PCOL(CURCOL)-COST(ARC)
         IF (TMAX.GT.MAX2) THEN
           IF (TMAX.GT.MAX1) THEN
              MAX2=MAX1
              MAX1=TMAX
              SNDCOL=BSTCOL
              BSTCOL=CURCOL
           ELSE
              MAX2=TMAX
              SNDCOL=CURCOL
           END IF
         END IF
108     CONTINUE
      END IF
      PROW(ROW)=MAX2
      OLDROW=ASSIGN(BSTCOL)
      INCR=MAX1-MAX2
      IF (INCR.GT.0) THEN
        PCOL(BSTCOL)=PCOL(BSTCOL)-INCR
        ASSIGN(BSTCOL)=ROW
        ROWLAB(ROW)=BSTCOL
        IF (OLDROW.GT.0) THEN
           I=I-1
           LIST(I)=OLDROW
           ROWLAB(OLDROW)=0
           ISIMPL=ISIMPL+1
           GOTO 100
        END IF
      ELSE
        IF (OLDROW.GT.0) THEN
          BSTCOL=SNDCOL
          OLDROW=ASSIGN(BSTCOL)
        END IF
        IF (OLDROW.EQ.0) THEN
          ASSIGN(BSTCOL)=ROW
          ROWLAB(ROW)=BSTCOL
        ELSE
          NEWNOL=NEWNOL+1
```

```
            LIST(NEWNOL)=ROW
          END IF
        END IF
        ISIMPL=ISIMPL+1
        IF (I.LE.NOLIST) GOTO 100

C    END OF A NAIVE AUCTION CYCLE

        COUNT=COUNT+1

        IF (NEWNOL.EQ.0) THEN
          NASSIH=NEWNOL
          GOTO 2000
        END IF
        IF (COUNT.LT.AUCTNUM) GOTO 80



C ***************************************************************
C
C       ******* END OF NAIVE AUCTION PART *******
C
C
C       ******* START OF SEQ. SH. PATH METHOD *******
C
C ***************************************************************



1000    NASSIH=NEWNOL
X        IHPRC=0
1020    DO 180 I=1,NEWNOL
           SCAN(I)=LIST(I)
180     CONTINUE
        DO 190 J=1,N
           MARG(J)=1
190     CONTINUE
        NOSCAN=NEWNOL
        IL1=1
        IL2=NOSCAN
1030    DO 1060 I=IL1,IL2
           CURROW=SCAN(I)
           ROWPR=PROW(CURROW)
           FSTARC=FOUT(CURROW)
           LSTARC=FOUT(CURROW+1)-1
           DO 1050 ARC=FSTARC,LSTARC
                CURCOL=END(ARC)
                OLMARG=MARG(CURCOL)
                IF (OLMARG.EQ.0) GO TO 1050
                TMARG=PCOL(CURCOL)-ROWPR-COST(ARC)
           IF (TMARG.EQ.0) THEN
             IF (ASSIGN(CURCOL).EQ.0) THEN
                GO TO 1500
```

```
C     PERFORM AUGMENTATION

              ELSE
                 MARG(CURCOL)=0
                 NOSCAN=NOSCAN+1
                 SCAN(NOSCAN)=ASSIGN(CURCOL)
                 COLLAB(CURCOL)=CURROW
              END IF
            ELSE
              IF ((OLMARG.GT.0).OR.(TMARG.GT.OLMARG)) THEN
                MARG(CURCOL)=TMARG
                COLLAB(CURCOL)=CURROW
              END IF
            END IF
1050     CONTINUE
1060     CONTINUE

C        CURRENT SCAN PHASE COMPLETE; CHECK FOR MORE ROWS TO SCAN

         IF (IL2.LT.NOSCAN) THEN
            IL1=IL2+1
            IL2=NOSCAN
            GO TO 1030
         END IF


C        PRICE CHANGE


         IPRC=IPRC+1
X         IHPRC=IHPRC+1
         INCR=ISMALL
         DO 200 J=1,N
            TMARG=MARG(J)
            IF ((TMARG.LT.0).AND.(TMARG.GT.INCR)) INCR=TMARG
200      CONTINUE
         DO 210 I=1,NOSCAN
            PROW(SCAN(I))=PROW(SCAN(I))+INCR
210      CONTINUE
         DO 220 J=1,N
            IF (MARG(J).EQ.0) PCOL(J)=PCOL(J)+INCR
220      CONTINUE
         DO 1400 J=1,N
            IF (MARG(J).GE.0) GO TO 1400
             MARG(J)=MARG(J)-INCR
               IF (MARG(J).EQ.0) THEN
                 IF (ASSIGN(J).EQ.0) THEN
                   CURCOL=J
                   CURROW=COLLAB(CURCOL)

C       PERFORM AUGMENTATION

                   GO TO 1500
                 ELSE
```

```
                  NOSCAN=NOSCAN+1
                  SCAN(NOSCAN)=ASSIGN(J)
                END IF
             END IF
1400     CONTINUE
         IL1=IL2+1
         IL2=NOSCAN
         GO TO 1030

C        END PRICE CHANGE


C        AUGMENTATION STARTS HERE

1500     IF (ROWLAB(CURROW).EQ.0) THEN
           ASSIGN(CURCOL)=CURROW
           ROWLAB(CURROW)=CURCOL
           GO TO 1700
         END IF
         TA=ROWLAB(CURROW)
         ASSIGN(CURCOL)=CURROW
         ROWLAB(CURROW)=CURCOL
         CURCOL=TA
         CURROW=COLLAB(CURCOL)
         GO TO 1500
1700     IF (NEWNOL.EQ.1) GO TO 2000

         DO 240 I=1,NEWNOL-1
            IF (LIST(I).EQ.CURROW) THEN
              DO 250 K=I+1,NEWNOL
                 LIST(K-I)=LIST(K)
250           CONTINUE
              DO 260 K=1,I-1
                 LIST(NEWNOL-I+K)=SCAN(K)
260           CONTINUE
              NEWNOL=NEWNOL-1
              GO TO 1020
            END IF

240      CONTINUE

         NEWNOL=NEWNOL-1
         GO TO 1020

C        END AUGMENTATION


C ***************************************************
C
C        EXIT ROUTINE. CHECKS FOR OPTIMALITY OF THE SOLUTION
C        CALCULATES THE OPTIMAL COST, AND COMPILES SOLUTION
C        STATISTICS. THE USER MAY REPLACE THIS BY CODE THAT
C        WRITES AT THE APPROPRIATE DEVICE THE OPTIMAL SOLUTION
C        CONTAINED IN THE ARRAY ASSIGN(.).
```

```
C
C ***************************************************

2000    CONTINUE

        TT =(LONG(362) - TIMER)/60

        PRINT*, 'TOTAL TIME = ',TT, ' secs.'
        PRINT*,'NO OF NAIVE AUCTION ITERATIONS:',ISIMPL
        PRINT*,'NO OF SH. PATH ITERATIONS:',NASSIH
X        PRINT*,'NO OF PRICE CHANGES:',IHPRC

C       CHECK FEASIBILITY OF SOLUTION & CALCULATE COST

        DO 265 I=1,N
           ROWLAB(I)=0
265     CONTINUE

        NOASS=0
        DO 280 J=1,N
           IF (ASSIGN(J).GT.0) THEN
             ROWLAB(ASSIGN(J))=J
             NOASS=NOASS+1
           END IF
280     CONTINUE

        IF (NOASS.LT.N) THEN
          PRINT*,'THE NUMBER OF ASSIGNED COLUMNS IS',NOASS,'(TOO SMALL)'
        END IF

        TCOST=0
        DO 300 I=1,N
          J=ROWLAB(I)
          IF (J.EQ.0) THEN
            PRINT*,'ROW ',I,' IS UNASSIGNED'
          END IF
          FSTARC=FOUT(I)
          LSTARC=FOUT(I+1)-1
          MINCOST=ILARGE
          DO 310 ARC=FSTARC,LSTARC
           CURCOL=END(ARC)
           TMARG=PROW(I)-PCOL(CURCOL)+COST(ARC)
           IF (TMARG.LT.0) THEN
             PRINT*,'COMPL. SLACKNESS VIOLATION: ROW',I,' COLUMN',CURCOL
           END IF
           IF (CURCOL.EQ.J) THEN
             IF (MINCOST.GT.COST(ARC)) THEN
                MINCOST=COST(ARC)
                ROWMARG=TMARG
             END IF
           END IF
310       CONTINUE
          IF (ROWMARG.NE.0) THEN
            PRINT*,'COMPL. SLACK. VIOLATION AT ASSIGNED ARC OF ROW',I
```

```
          END IF
          IF (MAXSET) THEN
            TCOST=TCOST-MINCOST
          ELSE
            TCOST=TCOST+MINCOST
          END IF
300     CONTINUE

        WRITE(9,2100) TCOST
2100    FORMAT(' ASSIGNMENT COST=',F14.2)

        PRINT *,'*********************************************'
        PRINT *, 'PROGRAM ENDED; PRESS <CR>'

        PAUSE
        STOP

        END



      SUBROUTINE SETRAN(ISEED)
      IMPLICIT REAL*8 (A-H,O-Z) , INTEGER*4 (I-N)
C**********************************************************************
C PORTABLE CONGRUENTIAL (UNIFORM) RANDOM NUMBER GENERATOR:
C     NEXT_VALUE = [(7**5) * PREVIOUS_VALUE] MODULO[(2**31)-1]
C
C THIS GENERATOR CONSISTS OF TWO ROUTINES:
C   (1) SETRAN - INITIALIZES CONSTANTS AND SEED
C   (2) RRAN   - GENERATES A REAL RANDOM NUMBER
C
C THE GENERATOR REQUIRES A MACHINE WITH AT LEAST 32 BITS OF PRECISION.
C THE SEED (ISEED) MUST BE IN THE RANGE (1,(2**31)-1).
C**********************************************************************
      COMMON /RANDM/ MULT,MODUL,I15,I16,JRAN
      IF(ISEED.LT.1) STOP 77
      MULT=16807
      MODUL=2147483647
      I15=2**15
      I16=2**16
      JRAN=ISEED
      RETURN
      END
C
      REAL FUNCTION RAN()
      IMPLICIT REAL*4 (A-H,O-Z) , INTEGER*4 (I-N)
C**********************************************************************
C RAN GENERATES A REAL RANDOM NUMBER BETWEEN 0 AND 1
C**********************************************************************
      COMMON /RANDM/ MULT,MODUL,I15,I16,JRAN
      IXHI=JRAN/I16
      IXLO=JRAN-IXHI*I16
      IXALO=IXLO*MULT
      LEFTLO=IXALO/I16
```

```
      IXAHI=IXHI*MULT
      IFULHI=IXAHI+LEFTLO
      IRTLO=IXALO-LEFTLO*I16
      IOVER=IFULHI/I15
      IRTHI=IFULHI-IOVER*I15
      JRAN=((IRTLO-MODUL)+IRTHI*I16)+IOVER
      IF(JRAN.LT.0) JRAN=JRAN+MODUL
      RAN = FLOAT(JRAN)/FLOAT(MODUL)
      RETURN
      END



************************************************************************
*

Max-Flow Codes

************************************************************************
*


FORD-FULKERSON

This code implements the Ford-Fulkerson method for
solving the max-flow problem  (cf.\ Section 1.2.2).

C     ******* SAMPLE CALLING PROGRAM FOR FORD-FULKERSON *******

      PARAMETER (MAXNODES=10000, MAXARCS=40000)
      IMPLICIT INTEGER (A-Z)
      COMMON /SCALARS/  N,NA,LARGE,SOURCE,SINK
      COMMON /STATS/    NITER
      COMMON /UBOUND/   U
      COMMON /FLOW/     F
      COMMON /BLK1/     STARTN
      COMMON /BLK2/     ENDN
      COMMON /BLK3/     PRDCSR
      COMMON /BLK4/     FIN
      COMMON /BLK5/     FOUT
      COMMON /BLK6/     NXTIN
      COMMON /BLK7/     NXTOU
      COMMON /LABELS/   LABEL
      COMMON /MARKS/    MARK

      INTEGER STARTN(MAXARCS)
      INTEGER ENDN(MAXARCS)
      INTEGER U(MAXARCS)
      INTEGER F(MAXARCS)
      INTEGER PRDCSR(MAXNODES)
      INTEGER FIN(MAXNODES)
      INTEGER FOUT(MAXNODES)
      INTEGER NXTIN(MAXARCS)
      INTEGER NXTOU(MAXARCS)
```

```
      INTEGER LABEL(MAXNODES)
      LOGICAL MARK(MAXNODES)
      REAL*8 TT,TIMER


      PRINT*,'FORD-FULKERSON METHOD FOR MAX-FLOW'
      PRINT*,'**********************************'
      PRINT *,'READING PROBLEM DATA'
      OPEN(13,FILE='FOR013.DAT',STATUS='OLD')
      REWIND(13)

      LARGE=500000000


C     READ NUMBER OF NODES AND ARCS

      READ(13,1010) N,NA

C     READ START, END, COST, AND CAPACITY OF EACH ARC

      DO 20 I=1,NA
        READ(13,1020) STARTN(I),ENDN(I),DUMMY,U(I)
20    CONTINUE

      ENDFILE(13)
      REWIND(13)

1000  FORMAT(1I8)
1010  FORMAT(2I8)
1020  FORMAT(4I8)


      PRINT*,'THE NUMBER OF NODES IS = ',N

41    PRINT*,'ENTER THE SUPERSOURCE NODE'
      READ*,SOURCE
      IF ((SOURCE.LE.0).OR.(SOURCE.GT.N)) GOTO 41

42    PRINT*,'ENTER THE SUPERSINK NODE'
      READ*,SINK

      IF ((SINK.LE.0).OR.(SINK.GT.N)) GOTO 42
      IF (SINK.EQ.R) GOTO 42

C     THE SOURCE AND SINK NODES WILL NOT BE ITERATED ON



      PRINT *, 'RESTRUCTURING THE DATA '
      CALL INIDAT


C   SET FLOWS TO ZERO
```

```
      DO 50 ARC=1,NA
        F(ARC)=0
50    CONTINUE

      PRINT *,'*********************************'
      TIMER = LONG(362)
      CALL FORD_FULK
      TT = FLOAT(LONG(362) - TIMER)/60
      PRINT*, 'TOTAL TIME = ',TT, ' secs.'

C     ***** COMPUTE MAX-FLOW *****

      MAX_FLOW=0
      DO 60 ARC=1,NA
        IF (STARTN(ARC).EQ.SOURCE) MAX_FLOW=MAX_FLOW+F(ARC)
60    CONTINUE
      MAX_FLOW2=0
      DO 70 ARC=1,NA
        IF (ENDN(ARC).EQ.SINK) MAX_FLOW2=MAX_FLOW2+F(ARC)
70    CONTINUE

      PRINT *,'# OF ITERATIONS = ',NITER
      PRINT *, 'MAX-FLOW = ',MAX_FLOW2
      PRINT *,'*********************************'

      IF (MAXFLOW.NE.MAXFLOW2) THEN
         PRINT*,'SOURCE FLOW NOT EQUAL TO SINK FLOW'
      ENDIF
      PRINT *, 'PROGRAM ENDED; PRESS <CR> TO EXIT'

      PAUSE
      END




C     ************************************************
      SUBROUTINE INIDAT
C     THIS SUBROUTINE USES THE DATA ARRAYS STARTN AND ENDN
C     TO CONSTRUCT AUXILIARY DATA ARRAYS FOUT, NXTOU, FIN, AND
C     NXTIN.  IN THIS SUBROUTINE WE
C     ARBITRARILY ORDER THE ARCS LEAVING EACH NODE AND STORE
C     THIS INFORMATION IN FOUT AND NXTOU.  SIMILARLY, WE ARBITRA-
C     RILLY ORDER THE ARCS ENTERING EACH NODE AND STORE THIS
C     INFORMATION IN FIN AND NXTIN.  AT THE COMPLETION OF THE
C     CONSTRUCTION, WE HAVE
C
C         FOUT(I)    = FIRST ARC LEAVING NODE I.
C         NXTOU(J)   = NEXT ARC LEAVING THE HEAD NODE OF ARC J.
C         FIN(I)     = FIRST ARC ENTERING NODE I.
C         NXTIN(J)   = NEXT ARC ENTERING THE TAIL NODE OF ARC J.

      PARAMETER (MAXNODES=10000, MAXARCS=40000)
      IMPLICIT INTEGER (A-Z)
```

```
      COMMON /SCALARS/  N,NA,LARGE
      COMMON /BLK1/     STARTN
      COMMON /BLK2/     ENDN
      COMMON /BLK4/     FIN
      COMMON /BLK5/     FOUT
      COMMON /BLK6/     NXTIN
      COMMON /BLK7/     NXTOU

      INTEGER STARTN(MAXARCS)
      INTEGER ENDN(MAXARCS)
      INTEGER FIN(MAXNODES)
      INTEGER FOUT(MAXNODES)
      INTEGER NXTIN(MAXARCS)
      INTEGER NXTOU(MAXARCS)
      INTEGER FINALIN(MAXNODES)
      INTEGER FINALOU(MAXNODES)

      DO 20 NODE=1,N
        FIN(NODE)=0
        FOUT(NODE)=0
        FINALIN(NODE)=0
        FINALOU(NODE)=0
20    CONTINUE

      DO 30 ARC=1,NA
        START=STARTN(ARC)
        END=ENDN(ARC)
        IF (FOUT(START).NE.0) THEN
          NXTOU(FINALOU(START))=ARC
        ELSE
          FOUT(START)=ARC
        END IF
        IF (FIN(END).NE.0) THEN
          NXTIN(FINALIN(END))=ARC
        ELSE
          FIN(END)=ARC
        END IF
        FINALOU(START)=ARC
        FINALIN(END)=ARC
        NXTIN(ARC)=0
        NXTOU(ARC)=0
30    CONTINUE
C
      RETURN
      END



C  ***************************************************************
C
C     BASIC FORD-FULKERSON METHOD FOR MAX-FLOW.
C
C  ***************************************************************
```

```
      SUBROUTINE FORD_FULK
      IMPLICIT INTEGER (A-Z)
      COMMON /SCALARS/  N,NA,LARGE,SOURCE,SINK
      COMMON /STATS/    NITER
      COMMON /UBOUND/   U
      COMMON /FLOW/     F
      COMMON /BLK1/     STARTN
      COMMON /BLK2/     ENDN
      COMMON /BLK3/     PRDCSR
      COMMON /BLK4/     FIN
      COMMON /BLK5/     FOUT
      COMMON /BLK6/     NXTIN
      COMMON /BLK7/     NXTOU
      COMMON /MARKS/    MARK
      COMMON /LABELS/   LABEL

      INTEGER STARTN(1),ENDN(1),U(1),F(1),FIN(1),FOUT(1)
      INTEGER NXTIN(1),NXTOU(1),PRDCSR(1),LABEL(1)
      LOGICAL MARK(1)


      NITER=0

      DO 10 I=1,N
        MARK(I)=.FALSE.
10    CONTINUE


C    START OF NEW ITERATION

15    NLABEL=1
      NSCAN=1
      MARK(SOURCE)=.TRUE.
      LABEL(1)=SOURCE


20    CONTINUE

C     SCAN A NEW NODE

      NODE=LABEL(NSCAN)

C     SCAN OUTGOING ARCS OF NODE

      ARC=FOUT(NODE)
30    IF (ARC.GT.0) THEN
         NODE2=ENDN(ARC)
         IF ((.NOT.MARK(NODE2)).AND.(F(ARC).LT.U(ARC))) THEN
           PRDCSR(NODE2)=ARC
           IF (NODE2.EQ.SINK) THEN
             CALL AUGMENT
             NITER=NITER+1
             DO 40 I=1,NLABEL
               MARK(LABEL(I))=.FALSE.
```

```
40           CONTINUE
             GOTO 15
           ELSE
             MARK(NODE2)=.TRUE.
             NLABEL=NLABEL+1
             LABEL(NLABEL)=NODE2
           END IF
         END IF
         ARC=NXTOU(ARC)
         GOTO 30
       END IF

C      SCAN INCOMING ARCS OF NODE

       ARC=FIN(NODE)
50     IF (ARC.GT.0) THEN
         NODE2=STARTN(ARC)
         IF ((.NOT.MARK(NODE2)).AND.(F(ARC).GT.0)) THEN
           PRDCSR(NODE2)=-ARC
           IF (NODE2.EQ.SINK) THEN
             CALL AUGMENT
             NITER=NITER+1
             DO 60 I=1,NLABEL
               MARK(LABEL(I))=.FALSE.
60           CONTINUE
             GOTO 15
           ELSE
             MARK(NODE2)=.TRUE.
             NLABEL=NLABEL+1
             LABEL(NLABEL)=NODE2
           END IF
         END IF
         ARC=NXTIN(ARC)
         GOTO 50
       END IF

C      CHECK FOR TERMINATION; SCAN A NEW NODE

       IF (NSCAN.EQ.NLABEL) THEN
         RETURN
       END IF
       NSCAN=NSCAN+1
       GOTO 20

       END




       SUBROUTINE AUGMENT
       IMPLICIT INTEGER (A-Z)
       COMMON /SCALARS/  N,NA,LARGE,SOURCE,SINK
       COMMON /UBOUND/   U
```

```
      COMMON /FLOW/     F
      COMMON /BLK1/     STARTN
      COMMON /BLK2/     ENDN
      COMMON /BLK3/     PRDCSR

      INTEGER STARTN(1),ENDN(1),U(1),F(1),PRDCSR(1)


      DX=LARGE
      CURNODE=SINK
10    IF (CURNODE.NE.SOURCE) THEN
        ARC=PRDCSR(CURNODE)
        IF (ARC.GT.0) THEN
          INCR=U(ARC)-F(ARC)
          IF (DX.GT.INCR) DX=INCR
          CURNODE=STARTN(ARC)
        ELSE
          ARC=-ARC
          INCR=F(ARC)
          IF (DX.GT.INCR) DX=INCR
          CURNODE=ENDN(ARC)
        END IF
        GOTO 10
      END IF

      CURNODE=SINK
20    IF (CURNODE.NE.SOURCE) THEN
        ARC=PRDCSR(CURNODE)
        IF (ARC.GT.0) THEN
          F(ARC)=F(ARC)+DX
          CURNODE=STARTN(ARC)
        ELSE
          ARC=-ARC
          F(ARC)=F(ARC)-DX
          CURNODE=ENDN(ARC)
        END IF
        GOTO 20
      END IF

      RETURN

      END



*****************************************************************************
*

$\e$-RELAX-MF

*****************************************************************************
*

This code implements the $\e$-relaxation
```

method for the max-flow problem (cf.\ Section 4.5). Here, $\e=1$ throughout the algorithm.

```
C     ******* SAMPLE CALLING PROGRAM FOR E-RELAX/MAX-FLOW *******
C
C     THIS VERSION USES A CYCLIC QUEUE, A SHORTEST PATH
C     (BREADTH FIRST) INITIALIZATION
C     AND A ROUTINE THAT DOES GLOBAL RELABELING AND ALSO
C     DETECTS A SATURATED CUT AND TERMINATES EARLY
C     IT THEN FINDS A MAXIMUM FLOW USING A "REVERSE'ALGORITHM
C
C     THIS CODE IS A SOMEWHAT DIFFERENT VERSION OF THE ONE
C     IN THE BOOK "LINEAR NETWORK OPTIMIZATION: ALGORITHMS AND
C     CODES, M.I.T. PRESS, 1991, BY DIMITRI P. BERTSEKAS
C
C     ************************************************
      PARAMETER (MAXNODES=20000, MAXARCS=120000)
      IMPLICIT INTEGER (A-Z)
      COMMON /SCALARS/  N,NA,LARGE,SOURCE,SINK
      COMMON /STATS/    NITER,CUTEXIT,CUTSEARCH,CUTCOUNT,NAUG
      COMMON /CHECK/    CHECKTIME,TOT_TIME
      COMMON /BLK1/     STARTN
      COMMON /BLK2/     ENDN
      COMMON /UBOUND/   U
      COMMON /FLOW/     F
      COMMON /PRICES/   P
      COMMON /BLK3/     SURPLUS
      COMMON /BLK4/     FIN
      COMMON /BLK5/     FOUT
      COMMON /BLK6/     NXTIN
      COMMON /BLK7/     NXTOU
      COMMON /BLK11/    FPUSHF
      COMMON /BLK12/    NXTPUSHF
      COMMON /BLK13/    FPUSHB
      COMMON /BLK14/    NXTPUSHB
      COMMON /LABELS/   LABEL
      COMMON /MARKS/    MARK
      COMMON /QUEUE/    NXTQUEUE
      COMMON /PRED/     PRED

C   RANDOM GENERATOR STUFF

      COMMON /RANDM/ MULT,MODUL,I15,I16,JRAN
      REAL RAN

      INTEGER NXTQUEUE(MAXNODES)
      INTEGER STARTN(MAXARCS)
      INTEGER ENDN(MAXARCS)
      INTEGER U(MAXARCS)
      INTEGER F(MAXARCS)
      INTEGER SURPLUS(MAXNODES)
      INTEGER FIN(MAXNODES)
      INTEGER FOUT(MAXNODES)
      INTEGER NXTIN(MAXARCS)
```

```
      INTEGER NXTOU(MAXARCS)
      INTEGER P(MAXNODES)
      INTEGER FPUSHF(MAXNODES)
      INTEGER NXTPUSHF(MAXARCS)
      INTEGER FPUSHB(MAXNODES)
      INTEGER NXTPUSHB(MAXARCS)
      INTEGER LABEL(MAXNODES)
      INTEGER PRED(MAXNODES)
      LOGICAL MARK(MAXNODES)
      REAL*8 TT,TOT_TIME,TIMER,TOTAL,TOTAL_C,CHECKTIME


      PRINT*,'E-RELAXATION METHOD FOR MAX-FLOW'
      PRINT*,'USES FIFO QUEUE AND GLOBAL RELABELING'
      PRINT*,'****************************************'

      LARGE=1000000000


C     ************************************************************
C
C     READ THE PROBLEM DATA
C
C     THE PROBLEM IS SPECIFIED BY THE FOLLOWING VARIABLES:
C
C     N:  THE NUMBER OF NODES
C     NA: THE NUMBER OF ARCS
C     SOURCE: THE SOURCE NODE OF THE MAX-FLOW PROBLEM
C     SINK: THE SINK NODE OF THE MAX-FLOW PROBLEM
C     STARTN(ARC): THE START NODE OF ARC
C     ENDN(ARC): THE END NODE OF ARC
C     U(ARC): THE CAPACITY OF ARC (IT IS LATER CHANGED TO RESIDUAL
CAPACITY)
C
C
      PRINT*,'READING PROBLEM DATA'

      OPEN(13,FILE='FOR013.DAT',STATUS='OLD')
      REWIND(13)

C     READ NUMBER OF NODES AND ARCS

      READ(13,*) N,NA

      DO 5 I=1,NA
        READ(13,*) STARTN(I),ENDN(I),DUMMY,U(I)
11.   CONTINUE


      ENDFILE(13)
      REWIND(13)


      SOURCE=1
```

```
      SINK=N

      PRINT*,'*****************************************'
      PRINT*,'NUMBER OF NODES =',N,'   NUMBER OF ARCS =',NA
C



C
C    END OF READING THE PROBLEM DATA
C

C  INITIALIZE PRICES & SURPLUSES

      DO 20 NODE=1,N
        FPUSHF(NODE)=0
        FPUSHB(NODE)=0
        P(NODE)=N
        SURPLUS(NODE)=0
20    CONTINUE


C  SET FLOWS TO UPPER BOUND OR LOWER BOUND TO SATISFY E-CS

      DO 50 ARC=1,NA
        IF (STARTN(ARC).EQ.SOURCE) THEN
          F(ARC)=U(ARC)
          SURPLUS(ENDN(ARC))=SURPLUS(ENDN(ARC))+F(ARC)
          SURPLUS(SOURCE)=SURPLUS(SOURCE)-F(ARC)
        ELSE
          F(ARC)=0
        END IF
50    CONTINUE


C     CREATE NECESSARY DATA STRUCTURES

      CALL INIDAT


      TIMER = LONG(362)
      CALL E_RELAX_MF
      TT = FLOAT(LONG(362) - TIMER)/60

      PRINT*,'TOTAL TIME = ',TT, ' secs'
      PRINT*,'TIME TO FIND A MIN CUT = ',TOT_TIME
      PRINT*,'TIME TO CHECK FOR A MIN CUT = ',CHECKTIME

C     ***** COMPUTE MAX-FLOW *****

      MAX_FLOW=0
      ARC=FIN(SINK)
70    IF (ARC.GT.0) THEN
        MAX_FLOW=MAX_FLOW+F(ARC)
```

```
        ARC=NXTIN(ARC)
        GOTO 70
      END IF
      IF (MAX_FLOW.NE.SURPLUS(SINK)) THEN
        PRINT*,'ERROR IN THE CALCULATION OF THE MAX-FLOW'
      END IF

      PRINT *, 'MAX-FLOW = ',MAX_FLOW
      PRINT *,'# OF FLOW CHANGES PER ARC = ',FLOAT(NAUG)/FLOAT(NA)
      PRINT *,'# OF PRICE RISES PER NODE = ',FLOAT(NITER)/FLOAT(N)
      PRINT *,'# OF SEARCHES FOR A SATURATED CUT = ',CUTCOUNT
      PRINT *,'***********************************************'

C     ***** CHECK THAT ALL SURPLUSES ARE ZERO ****

        DO 80 NODE=1,N
          IF ((NODE.NE.SOURCE).AND.(NODE.NE.SINK)) THEN
            IF (SURPLUS(NODE).NE.0) THEN
              PRINT*,'SURPLUS OF NODE ',NODE,' IS NONZERO'
            END IF
          END IF
80      CONTINUE


      PRINT *, 'PROGRAM ENDED; PRESS <CR> TO EXIT'
      PAUSE

      END




C  ***************************************************************
C
C     BASIC E-RELAXATION FOR MAX-FLOW.
C     THIS VERSION USES A QUEUE TO SELECT NODES TO ITERATE ON,
C     AND MAINTAINS PUSH LISTS. NODES JOIN THE QUEUE AT THE BOTTOM.
C     THIS VERSION USES A SHORTEST PATH (BREADTH FIRST) INITIALIZATION
C     AND A ROUTINE THAT DETECTS A SATURATED CUT AND
C     PERFORMS GLOBAL RELABELING
C
C     THIS CODE IS A SOMEWHAT DIFFERENT VERSION OF THE ONE
C     IN THE BOOK "LINEAR NETWORK OPTIMIZATION: ALGORITHMS AND
C     CODES, M.I.T. PRESS, 1991, BY DIMITRI P. BERTSEKAS
C
C  ***************************************************************

      SUBROUTINE E_RELAX_MF
      PARAMETER (MAXNODES=20000, MAXARCS=120000)
      IMPLICIT INTEGER (A-Z)
      COMMON /SCALARS/  N,NA,LARGE,SOURCE,SINK
```

```
      COMMON /STATS/    NITER,CUTEXIT,CUTSEARCH,CUTCOUNT,NAUG
      COMMON /CHECK/    CHECKTIME,TOT_TIME
      COMMON /BLK1/     STARTN
      COMMON /BLK2/     ENDN
      COMMON /UBOUND/   U
      COMMON /FLOW/     F
      COMMON /PRICES/   P
      COMMON /BLK3/     SURPLUS
      COMMON /BLK4/     FIN
      COMMON /BLK5/     FOUT
      COMMON /BLK6/     NXTIN
      COMMON /BLK7/     NXTOU
      COMMON /BLK11/    FPUSHF
      COMMON /BLK12/    NXTPUSHF
      COMMON /BLK13/    FPUSHB
      COMMON /BLK14/    NXTPUSHB
      COMMON /MARKS/    MARK
      COMMON /LABELS/   LABEL
      COMMON /QUEUE/    NXTQUEUE
      COMMON /PRED/     PRED

      INTEGER NXTQUEUE(MAXNODES)
      INTEGER STARTN(MAXARCS)
      INTEGER ENDN(MAXARCS)
      INTEGER U(MAXARCS)
      INTEGER F(MAXARCS)
      INTEGER SURPLUS(MAXNODES)
      INTEGER FIN(MAXNODES)
      INTEGER FOUT(MAXNODES)
      INTEGER NXTIN(MAXARCS)
      INTEGER NXTOU(MAXARCS)
      INTEGER P(MAXNODES)
      INTEGER FPUSHF(MAXNODES)
      INTEGER NXTPUSHF(MAXARCS)
      INTEGER FPUSHB(MAXNODES)
      INTEGER NXTPUSHB(MAXARCS)
      INTEGER LABEL(MAXNODES)
      INTEGER PRED(MAXNODES)
      LOGICAL MARK(MAXNODES)
      REAL*8  TIMER,TOT_TIME,TIMER1,CHECKTIME
      LOGICAL OUTFLAG,RELABEL

      TIMER1=LONG(362)

C     INITIALIZE COUNT OF NUMBER OF UP ITERATIONS PERFORMED

      NITER = 0
      NAUG=0
      COUNT=0
      CUTCOUNT=0
      CUTEXIT=0
      THRESH=N
      CHECKTIME=0
      TOT_TIME=0
```

```
      RELABEL=.TRUE.


C   INITIALIZE PRICES BY A BREADTH FIRST SEARCH METHOD

        DO 51 NODE=1,N
          MARK(NODE)=.FALSE.
51      CONTINUE

        MARK(SOURCE)=.TRUE.
        MARK(SINK)=.TRUE.
        P(SOURCE)=N
        P(SINK)=0

        NLABEL=1
        LABEL(1)=SINK
        NSCAN=0

55      CONTINUE
        NSCAN=NSCAN+1
        NODE=LABEL(NSCAN)
        PRICE=P(NODE)+1

        ARC=FIN(NODE)
58      IF (ARC.GT.0) THEN
          START=STARTN(ARC)
          IF (.NOT.MARK(START)) THEN
            NLABEL=NLABEL+1
            LABEL(NLABEL)=START
            P(START)=PRICE
            MARK(START)=.TRUE.
          END IF
          ARC=NXTIN(ARC)
          GOTO 58
        END IF

        IF (NLABEL.GT.NSCAN) GOTO 55

C     ***** QUEUE INITIALIZATION *****

70      CONTINUE

        FIRST=0
        DO 82 I=1,NLABEL
          NODE=LABEL(I)
          IF ((NODE.NE.SOURCE).AND.(NODE.NE.SINK)) THEN
            IF (SURPLUS(NODE).GT.0) THEN
              IF (FIRST.EQ.0) THEN
                FIRST=NODE
              ELSE
                NXTQUEUE(LAST)=NODE
              END IF
```

```
              LAST=NODE
            ELSE
              NXTQUEUE(NODE)=0
            END IF
          ELSE
            NXTQUEUE(NODE)=-1
          END IF
82      CONTINUE
        NXTQUEUE(LAST)=FIRST

        IF (FIRST.EQ.0) RETURN

        I=FIRST
        PREVNODE=LAST

C ***********************************************************
C
C     START OF THE MAIN ALGORITHM.
C     DO SINGLE NODE ITERATIONS UNTIL TERMINATION.
C
C ***********************************************************

100   CONTINUE

C     TAKE UP NEXT NODE (NODE I) FOR ITERATION

        SURPI=SURPLUS(I)
        PRICE = P(I)

        IF ((SURPI .GT. 0).AND.(PRICE.LT.N))  THEN
C          PRINT*,'NODE ',I,'   START PRICE = ',PRICE
           ARCF = FPUSHF(I)
           ARCB = FPUSHB(I)

C   ********************** D - PUSHES ************************

C   START BY TRYING TO PUSH AWAY FLOW ON ARCS THAT WERE
C   ADMISSIBLE AT THE END OF THE LAST ITERATION (IF ANY)
C   AT THIS NODE. WE MUST CHECK THAT THEY ARE STILL ADMISSIBLE.

120               IF ((SURPI .GT. 0) .AND. (ARCF .GT. 0)) THEN
                     RESID = U(ARCF) - F(ARCF)
                     J = ENDN(ARCF)
                    IF ((PRICE-P(J).EQ.1).AND.(RESID.GT.0)) THEN
                        NAUG=NAUG+1
                        IF (SURPI .GE. RESID) THEN
                           F(ARCF) = U(ARCF)
                           SURPI = SURPI - RESID
                           SURPLUS(J) = SURPLUS(J) + RESID
                           IF (NXTQUEUE(J).EQ.0) THEN
                            IF (SURPLUS(J).GT.0) THEN
                             NXTQUEUE(PREVNODE)=J
                             NXTQUEUE(J)=I
                             PREVNODE=J
```

```
                    END IF
                  END IF
                  ARCF = NXTPUSHF(ARCF)
               ELSE
                  F(ARCF) = F(ARCF) + SURPI
                  SURPLUS(J) = SURPLUS(J) + SURPI
                  IF (NXTQUEUE(J).EQ.0) THEN
                   IF (SURPLUS(J).GT.0) THEN
                    NXTQUEUE(PREVNODE)=J
                    NXTQUEUE(J)=I
                    PREVNODE=J
                   END IF
                  END IF
                  SURPI = 0
               ENDIF
            ELSE
               ARCF = NXTPUSHF(ARCF)
            ENDIF
            GOTO 120
         ENDIF
121      IF ((SURPI .GT. 0) .AND. (ARCB .GT. 0)) THEN
            J = STARTN(ARCB)
           IF ((PRICE-P(J).EQ.1).AND.(F(ARCB).GT.0)) THEN
               NAUG=NAUG+1
               IF (SURPI .GE. F(ARCB)) THEN
                  SURPI = SURPI - F(ARCB)
                  SURPLUS(J) = SURPLUS(J) + F(ARCB)
                  IF (NXTQUEUE(J).EQ.0) THEN
                   IF (SURPLUS(J).GT.0) THEN
                    NXTQUEUE(PREVNODE)=J
                    NXTQUEUE(J)=I
                    PREVNODE=J
                   END IF
                  END IF
                  F(ARCB) = 0
                  ARCB = NXTPUSHB(ARCB)
               ELSE
                  F(ARCB) = F(ARCB) - SURPI
                  SURPLUS(J) = SURPLUS(J) + SURPI
                  IF (NXTQUEUE(J).EQ.0) THEN
                   IF (SURPLUS(J).GT.0) THEN
                    NXTQUEUE(PREVNODE)=J
                    NXTQUEUE(J)=I
                    PREVNODE=J
                   END IF
                  END IF
                  SURPI = 0
               ENDIF
            ELSE
               ARCB = NXTPUSHB(ARCB)
            ENDIF
            GOTO 121
         ENDIF
```

```
C                NOW WE ENTER A REPEAT...UNTIL SORT OF LOOP UNTIL WE
C                HAVE THE SURPLUS REDUCED TO ZERO.

C    ********************* PRICE RISE *********************

129              CONTINUE

C                FIRST, TRY A (POSSIBLY DEGENERATE) PRICE RISE.

130              IF ((ARCF .EQ. 0) .AND. (ARCB .EQ. 0)) THEN
                    NITER = NITER + 1
                    PRICE = LARGE
                    ARC = FOUT(I)
131                 IF (ARC .GT. 0) THEN
                       IF (F(ARC) .LT. U(ARC)) THEN
                          XP = P(ENDN(ARC)) + 1
                          IF (XP .LT. PRICE) THEN
                             PRICE = XP
                             ARCF = ARC
                             NXTPUSHF(ARC)=0
                          ELSE
                           IF (XP .EQ. PRICE) THEN
                             NXTPUSHF(ARC) = ARCF
                             ARCF = ARC
                           END IF
                          ENDIF
                       ENDIF
                       ARC = NXTOU(ARC)
                       GOTO 131
                    ENDIF
                    ARC = FIN(I)
132                 IF (ARC .GT. 0) THEN
                       IF (F(ARC) .GT. 0) THEN
                          XP = P(STARTN(ARC))+1
                          IF (XP .LT. PRICE) THEN
                             PRICE = XP
                             ARCB = ARC
                             ARCF = 0
                             NXTPUSHB(ARC) = 0
                          ELSE
                           IF (XP .EQ. PRICE) THEN
                             NXTPUSHB(ARC) = ARCB
                             ARCB = ARC
                           END IF
                          ENDIF
                       ENDIF
                       ARC = NXTIN(ARC)
                       GOTO 132
                    ENDIF

                 ENDIF

C     ********************* D - PUSHES *********************
```

```
C                IF THE STEP WAS NOT DEGENERATE, TRY AND DO SOME
C                PUSHING.  WE DO NOT REUSE THE CODE ABOVE BECAUSE IT IS
C                NOT NECESSARY TO CHECK IF THE ARCS USED ARE ADMISSIBLE.

                 IF (SURPI .GT. 0) THEN
141                 IF (ARCF .GT. 0) THEN
                       RESID = U(ARCF) - F(ARCF)
                       NAUG=NAUG+1
                       J = ENDN(ARCF)
                       IF (SURPI .GE. RESID) THEN
                          F(ARCF) = U(ARCF)
                          SURPI = SURPI - RESID
                          SURPLUS(J) = SURPLUS(J) + RESID
                          IF (NXTQUEUE(J).EQ.0) THEN
                           IF (SURPLUS(J).GT.0) THEN
                            NXTQUEUE(PREVNODE)=J
                            NXTQUEUE(J)=I
                            PREVNODE=J
                           END IF
                          END IF
                          ARCF = NXTPUSHF(ARCF)
                          IF (SURPI .GT. 0) GOTO 141
                       ELSE
                          F(ARCF) = F(ARCF) + SURPI
                          SURPLUS(J) = SURPLUS(J) + SURPI
                          IF (NXTQUEUE(J).EQ.0) THEN
                           IF (SURPLUS(J).GT.0) THEN
                            NXTQUEUE(PREVNODE)=J
                            NXTQUEUE(J)=I
                            PREVNODE=J
                           END IF
                          END IF
                          SURPI = 0
                       ENDIF
                    END IF

142                 IF ((SURPI .GT. 0).AND.(ARCB .GT. 0)) THEN
                       J = STARTN(ARCB)
                       NAUG=NAUG+1
                       IF (SURPI .GE. F(ARCB)) THEN
                          SURPI = SURPI - F(ARCB)
                          SURPLUS(J) = SURPLUS(J) + F(ARCB)
                          IF (NXTQUEUE(J).EQ.0) THEN
                            NXTQUEUE(PREVNODE)=J
                            NXTQUEUE(J)=I
                            PREVNODE=J
                          END IF
                          F(ARCB) = 0
                          ARCB = NXTPUSHB(ARCB)
                       ELSE
                          F(ARCB) = F(ARCB) - SURPI
                          SURPLUS(J) = SURPLUS(J) + SURPI
                          IF (NXTQUEUE(J).EQ.0) THEN
                            NXTQUEUE(PREVNODE)=J
```

```
                              NXTQUEUE(J)=I
                              PREVNODE=J
                            END IF
                            SURPI = 0
                          ENDIF
                          GOTO 142
                        ENDIF

C  WE'VE DONE ALL THE PUSHING WE CAN AT THIS PRICE
C  LEVEL.  TRY TO DO A (POSSIBLY DEGENERATE) PRICE
C  INCREASE.

                        GOTO 129
                      ENDIF

C  THIS IS THE END OF THE UP ITERATION.  IF WE GET HERE, THE
C  SURPLUS OF I IS 0, AND WE HAVE DONE OUR LAST PRICE RISE.

                  SURPLUS(I) = 0
                  P(I) = PRICE
                  FPUSHF(I) = ARCF
                  FPUSHB(I) = ARCB

               ENDIF


C  ADVANCE THE QUEUE.
C  IF THE QUEUE IS EMPTY RETURN ALL SURPLUS TO THE SOURCE AND EXIT

            NXTNODE=NXTQUEUE(I)
            IF (I.EQ.NXTNODE) THEN
              IF (CUTEXIT.EQ.1) THEN
                RETURN
              ELSE
                TOT_TIME= TOT_TIME+FLOAT(LONG(362) - TIMER1)/60
                GOTO 600
              END IF
            ELSE
              NXTQUEUE(PREVNODE)=NXTNODE
              NXTQUEUE(I)=0
              I=NXTNODE
            END IF

            IF (COUNT.LT.THRESH) THEN
              COUNT=COUNT+1
              GO TO 100
            END IF


C  **************** PERIODICALLY LOOK FOR A SATURATING CUT
***************
C      AND RECALCULATE THE PRICES TO BE EQUAL TO THE SHORTEST DISTANCES

500     CONTINUE
```

```
X       PRINT*,'** CHECKING FOR EARLY TERMINATION **'

        TIMER = LONG(362)

C    UPDATE THE THRESHOLD

        THRESH=INT(1.1*THRESH)

C    REINITIALIZE THE COUNT OF ITERATIONS TO THE NEXT CHECK FOR A MIN CUT

        COUNT=0
        CUTCOUNT=CUTCOUNT+1
        DO 510 L=1,NLABEL
          MARK(LABEL(L))=.FALSE.
510     CONTINUE

        OUTFLAG=.FALSE.

        NLABEL=1
        LABEL(1)=SINK
        NSCAN=0

550     CONTINUE
        NSCAN=NSCAN+1
        NODE=LABEL(NSCAN)
        MARK(NODE)=.TRUE.
        ARC=FIN(NODE)
580     IF (ARC.GT.0) THEN
           START=STARTN(ARC)

C   IF THIS IS AN ARC ALONG WHICH FLOW CAN BE PUSHED TO NODE
C   CHECK IF THE OPP. NODE HAS 0 SURPLUS AND IF SO LABEL THE NODE

            IF (F(ARC).LT.U(ARC)) THEN
              IF (.NOT.MARK(START)) THEN
                IF (RELABEL) THEN
                  IF (SURPLUS(START).GT.0) THEN
                    OUTFLAG=.TRUE.
                  END IF
                  P(START)=P(NODE)+1
                ELSE
                  IF (SURPLUS(START).GT.0) THEN
                    CHECKTIME= CHECKTIME+FLOAT(LONG(362) - TIMER)/60
                    GOTO 100
                  END IF
                END IF
                NLABEL=NLABEL+1
                LABEL(NLABEL)=START
                MARK(START)=.TRUE.
              END IF
            END IF
            ARC=NXTIN(ARC)
            GOTO 580
```

```
        END IF

        ARC=FOUT(NODE)
585     IF (ARC.GT.0) THEN
           END=ENDN(ARC)

C   IF THIS IS AN ARC ALONG WHICH FLOW CAN BE PUSHED TO NODE
C   CHECK IF THE OPP. NODE HAS 0 SURPLUS AND IF SO LABEL THE NODE

           IF (F(ARC).GT.0) THEN
             IF (.NOT.MARK(END)) THEN
               IF (RELABEL) THEN
                 IF (SURPLUS(END).GT.0) THEN
                   OUTFLAG=.TRUE.
                 END IF
                 P(END)=P(NODE)+1
               ELSE
                 IF (SURPLUS(END).GT.0) THEN
                   CHECKTIME= CHECKTIME+FLOAT(LONG(362) - TIMER)/60
                   GOTO 100
                 END IF
               END IF
               NLABEL=NLABEL+1
               LABEL(NLABEL)=END
               MARK(END)=.TRUE.
             END IF
           END IF
           ARC=NXTOU(ARC)
           GOTO 585
        END IF

        IF (NLABEL.GT.NSCAN) GOTO 550

C   IF A NODE W/ POSITIVE SURPLUS IS REACHABLE FROM THE SINK
C   RELABEL THE UNREACHABLE NODES TO N AND GO BACK FOR MORE ITERATION

        IF (OUTFLAG) THEN

          IF (RELABEL) THEN
            DO 595 NODE=1,N
              IF (.NOT.MARK(NODE)) THEN
                IF (P(NODE).LT.N) P(NODE)=N
              END IF
595           CONTINUE
          END IF

X         PRINT*,'OUT OF CHECK FOR EARLY TERMINATION'
          CHECKTIME= CHECKTIME+FLOAT(LONG(362) - TIMER)/60
          GOTO 100
        ELSE
X         PRINT*,'SATURATED CUT FOUND'
X         PRINT*,'# OF NODES ON SINK SIDE OF THE CUT =',NLABEL
          CHECKTIME= CHECKTIME+FLOAT(LONG(362) - TIMER)/60
          TOT_TIME= TOT_TIME+FLOAT(LONG(362) - TIMER1)/60
```

```
          GOTO 600
        END IF

C
C   ********************** FIND A MAX-FLOW **************************
C
C   SATURATED CUT HAS BEEN FOUND BUT SOME NODES MAY HAVE NONZERO SURPLUS.
C   TO FIND A MAX-FLOW, WE RETURN THE EXCESS FLOW TO THE SOURCE BY
C   USING THE SAME ALGORITHM
C

600    CONTINUE

       CUTEXIT=1
       RELABEL=.FALSE.
       COUNT=0
       THRESH=LARGE

        DO 620 NODE=1,N
          MARK(NODE)=.FALSE.
620     CONTINUE

        MARK(SOURCE)=.TRUE.
        MARK(SINK)=.TRUE.
        P(SOURCE)=0

        NLABEL=1
        LABEL(1)=SOURCE
        NSCAN=0

650     CONTINUE
        NSCAN=NSCAN+1
        NODE=LABEL(NSCAN)
        PRICE=P(NODE)+1

        ARC=FIN(NODE)
660     IF (ARC.GT.0) THEN
          IF (U(ARC).GT.0) THEN
            START=STARTN(ARC)
            IF (.NOT.MARK(START)) THEN
              NLABEL=NLABEL+1
              LABEL(NLABEL)=START
              P(START)=PRICE
              MARK(START)=.TRUE.
            END IF
          END IF
          ARC=NXTIN(ARC)
          GOTO 660
        END IF

        ARC=FOUT(NODE)
670     IF (ARC.GT.0) THEN
          IF (F(ARC).GT.0) THEN
            END=ENDN(ARC)
```

```
            IF (.NOT.MARK(END)) THEN
              NLABEL=NLABEL+1
              LABEL(NLABEL)=END
              P(END)=PRICE
              MARK(END)=.TRUE.
            END IF
          END IF
          ARC=NXTOU(ARC)
          GOTO 670
        END IF

        IF (NLABEL.GT.NSCAN) GOTO 650

        GOTO 70

        END

C     ************************************************
      SUBROUTINE INIDAT
C     THIS SUBROUTINE USES THE DATA ARRAYS STARTN AND ENDN
C     TO CONSTRUCT AUXILIARY DATA ARRAYS FOUT, NXTOU, FIN, AND
C     NXTIN THAT ARE REQUIRED BY E-RELAX-MF.  IN THIS SUBROUTINE WE
C     ARBITRARILY ORDER THE ARCS LEAVING EACH NODE AND STORE
C     THIS INFORMATION IN FOUT AND NXTOU.  SIMILARLY, WE ARBITRA-
C     RILLY ORDER THE ARCS ENTERING EACH NODE AND STORE THIS
C     INFORMATION IN FIN AND NXTIN.  AT THE COMPLETION OF THE
C     CONSTRUCTION, WE HAVE
C
C         FOUT(I)    = FIRST ARC LEAVING NODE I.
C         NXTOU(J)   = NEXT ARC LEAVING THE HEAD NODE OF ARC J.
C         FIN(I)     = FIRST ARC ENTERING NODE I.
C         NXTIN(J)   = NEXT ARC ENTERING THE TAIL NODE OF ARC J.

      PARAMETER (MAXNODES=20000, MAXARCS=120000)
      IMPLICIT INTEGER (A-Z)
      COMMON /SCALARS/  N,NA,LARGE,SOURCE,SINK
      COMMON /BLK1/     STARTN
      COMMON /BLK2/     ENDN
      COMMON /BLK4/     FIN
      COMMON /BLK5/     FOUT
      COMMON /BLK6/     NXTIN
      COMMON /BLK7/     NXTOU
      INTEGER STARTN(MAXARCS)
      INTEGER ENDN(MAXARCS)
      INTEGER FIN(MAXNODES)
      INTEGER FOUT(MAXNODES)
      INTEGER NXTIN(MAXARCS)
      INTEGER NXTOU(MAXARCS)
      INTEGER FINALIN(MAXNODES),FINALOU(MAXNODES)
C
      DO 20 NODE=1,N
        FIN(NODE)=0
        FOUT(NODE)=0
        FINALIN(NODE)=0
```

```
        FINALOU(NODE)=0
20    CONTINUE
C
      DO 30 ARC=1,NA
        START=STARTN(ARC)
        END=ENDN(ARC)
        IF (FOUT(START).NE.0) THEN
          NXTOU(FINALOU(START))=ARC
        ELSE
          FOUT(START)=ARC
        END IF
        IF (FIN(END).NE.0) THEN
          NXTIN(FINALIN(END))=ARC
        ELSE
          FIN(END)=ARC
        END IF
        FINALOU(START)=ARC
        FINALIN(END)=ARC
        NXTIN(ARC)=0
        NXTOU(ARC)=0
30    CONTINUE
C
      RETURN
      END

C     ***************************************************


      SUBROUTINE SETRAN(ISEED)
      IMPLICIT REAL*8 (A-H,O-Z) , INTEGER*4 (I-N)
C*********************************************************************
C PORTABLE CONGRUENTIAL (UNIFORM) RANDOM NUMBER GENERATOR:
C     NEXT_VALUE = [(7**5) * PREVIOUS_VALUE] MODULO[(2**31)-1]
C
C THIS GENERATOR CONSISTS OF TWO ROUTINES:
C   (1) SETRAN - INITIALIZES CONSTANTS AND SEED
C   (2) RRAN   - GENERATES A REAL RANDOM NUMBER
C
C THE GENERATOR REQUIRES A MACHINE WITH AT LEAST 32 BITS OF PRECISION.
C THE SEED (ISEED) MUST BE IN THE RANGE (1,(2**31)-1).
C*********************************************************************
      COMMON /RANDM/ MULT,MODUL,I15,I16,JRAN
      IF(ISEED.LT.1) STOP 77
      MULT=16807
      MODUL=2147483647
      I15=2**15
      I16=2**16
      JRAN=ISEED
      RETURN
      END
C
      REAL FUNCTION RAN()
      IMPLICIT REAL*4 (A-H,O-Z) , INTEGER*4 (I-N)
C*********************************************************************
```

```
C RAN GENERATES A REAL RANDOM NUMBER BETWEEN 0 AND 1
C*********************************************************************
      COMMON /RANDM/ MULT,MODUL,I15,I16,JRAN
      IXHI=JRAN/I16
      IXLO=JRAN-IXHI*I16
      IXALO=IXLO*MULT
      LEFTLO=IXALO/I16
      IXAHI=IXHI*MULT
      IFULHI=IXAHI+LEFTLO
      IRTLO=IXALO-LEFTLO*I16
      IOVER=IFULHI/I15
      IRTHI=IFULHI-IOVER*I15
      JRAN=((IRTLO-MODUL)+IRTHI*I16)+IOVER
      IF(JRAN.LT.0) JRAN=JRAN+MODUL
      RAN = FLOAT(JRAN)/FLOAT(MODUL)
      RETURN
      END


**************************************************************************
*

$\e$-Relaxation Codes

**************************************************************************
*


$\e$-RELAX

This code implements the $\e$-relaxation method with
$\e$-scaling for the minimum cost flow problem (cf.\ Section 4.5).

C     ** SAMPLE CALLING PROGRAM FOR E-RELAX **
C
C     THIS PROGRAM WILL READ A PROBLEM FILE IN STANDARD FORMAT
C     AND SOLVE IT USING E-RELAX.
C
C     THIS IS AN E-SCALED VERSION THAT USES A CYCLIC QUEUE
C
C     ************************************************
C
      PARAMETER (MAXNODES=8000, MAXARCS=25000)
      IMPLICIT INTEGER (A-Z)
      COMMON /SCALARS/  N,NA,LARGE,FACTOR,EPS,
     &THRESHSURPLUS,SFACTOR
      COMMON /STATS/    NCYC,NITER,TOTALITER
      COMMON /BLK1/     STARTN
      COMMON /BLK2/     ENDN
      COMMON /UBOUND/   U
      COMMON /FLOW/     F
      COMMON /PRICES/   P
      COMMON /BLK3/     SURPLUS
      COMMON /BLK4/     FIN
```

```
      COMMON /BLK5/     FOUT
      COMMON /BLK6/     NXTIN
      COMMON /BLK7/     NXTOU
      COMMON /BLK11/    FPUSHF
      COMMON /BLK12/    NXTPUSHF
      COMMON /BLK13/    FPUSHB
      COMMON /BLK14/    NXTPUSHB
      COMMON /COST/     COST
      COMMON /QUEUE/    NXTQUEUE
      INTEGER NXTQUEUE(MAXNODES)
      INTEGER STARTN(MAXARCS)
      INTEGER ENDN(MAXARCS)
      INTEGER U(MAXARCS)
      INTEGER F(MAXARCS)
      INTEGER SURPLUS(MAXNODES)
      INTEGER FIN(MAXNODES)
      INTEGER FOUT(MAXNODES)
      INTEGER NXTIN(MAXARCS)
      INTEGER NXTOU(MAXARCS)
      INTEGER P(MAXNODES)
      INTEGER FPUSHF(MAXNODES)
      INTEGER NXTPUSHF(MAXARCS)
      INTEGER FPUSHB(MAXNODES)
      INTEGER NXTPUSHB(MAXARCS)
      INTEGER COST(MAXARCS)
      REAL*8 TT,TIMER
      DOUBLE PRECISION TCOST,PRODUCT


      PRINT*,'E-RELAXATION METHOD FOR MIN COST FLOW'
      PRINT*,'************************************'
      PRINT *,'READING PROBLEM DATA'
      OPEN(13,FILE='FOR013.DAT',STATUS='OLD')
      REWIND(13)

C     READ NUMBER OF NODES AND ARCS

      READ(13,1010) N,NA

C     READ START, END, COST, AND CAPACITY OF EACH ARC
C     AND GENERATE MAX COST VALUE

      MAXCOST=0

      DO 5 I=1,NA
        READ(13,1020) STARTN(I),ENDN(I),COST(I),U(I)
        COST(I)=COST(I)*(N+1)
        IF (ABS(COST(I)).GT.MAXCOST) MAXCOST=ABS(COST(I))
12.   CONTINUE

C     READ SUPPLY OF EACH NODE

      DO 8 I=1,N
        READ(13,1000) SURPLUS(I)
```

```
8     CONTINUE

      ENDFILE(13)
      REWIND(13)

      PRINT*,'END OF READING'

1000  FORMAT(1I8)
1010  FORMAT(2I8)
1020  FORMAT(4I8)

C
      LARGE=1500000000

C    INITIALIZE PRICES & PUSH LISTS

      DO 10 NODE=1,N
        FPUSHF(NODE)=0
        FPUSHB(NODE)=0
        P(NODE)=-INT(LARGE/10)
10    CONTINUE

C     IN THE FOLLOWING STATEMENTS, THE E-SCALING PARAMETERS ARE SET.
C     THE FOLLOWING RANGES ARE RECOMMENDED:
C     STARTING EPSILON: BETWEEN MAXCOST/20 TO MAXCOST/2
C     SCALING FACTOR: BETWEEN FOUR AND TEN

25    CONTINUE
      PRINT*,'ENTER STARTING EPSILON'
      PRINT*,'SHOULD BE BETWEEN 1 & ',MAXCOST
      READ*,EPS
      IF (EPS.LT.1) GO TO 25
      PRINT*,'ENTER SCALING FACTOR'
30    CONTINUE
      READ*,FACTOR
      IF ((EPS.GT.1).AND.(FACTOR.LE.1)) THEN
        PRINT*,'ENTER SCALING FACTOR; SHOULD BE GREATER THAN 1'
        GO TO 30
      END IF
C
        MAXSURPLUS=-LARGE
        DO 920 NODE=1,N
          IF (SURPLUS(NODE).GT.MAXSURPLUS) THEN
            MAXSURPLUS=SURPLUS(NODE)
          END IF
920     CONTINUE

C     SET THE PARAMETER SSCALE TO 1 TO ALLOW SURPLUS SCALING
C     A HEURISTIC SCHEME IS USED TO SET THE SFACTOR AND
C     THRESHSURPLUS PARAMETERS THAT CONTROL SURPLUS SCALING

      SSCALE=1

      IF (SSCALE.EQ.1) THEN
```

```
        SFACTOR=2+INT(FACTOR*MAXSURPLUS/MAXCOST)
        IF (SFACTOR.LT.4) SFACTOR=4
        THRESHSURPLUS=INT(MAXSURPLUS/SFACTOR)
        IF (EPS.EQ.1) THRESHSURPLUS=0
      ELSE
        THRESHSURPLUS=0
      END IF

      PRINT*,'INITIALIZING DATA STRUCTURES'
      CALL INIDAT

      PRINT *,'*********************************'
      PRINT *,'CALLING E-RELAX FOR MCF (+ SURPLUS NODES ITERATED ONLY)'
      TIMER = LONG(362)
      CALL EPS_RELAX
      TT =(LONG(362) - TIMER)/60
      PRINT*, 'TOTAL TIME = ',TT, ' secs.'

      TCOST=0
      DO 330 I=1,NA
        COST(I)=COST(I)/(N+1)
        PRODUCT=FLOAT(F(I)*COST(I))
        TCOST=TCOST+PRODUCT
330   CONTINUE

      WRITE(9,1100) TCOST
1100  FORMAT(' ','OPTIMAL COST         =',F14.2)

      PRINT *,'*********************************'
      PRINT *,'# OF ITERATIONS = ',TOTALITER
      PRINT *,'# OF S. N. PRICE RISES = ',NITER
      PRINT *,'*********************************'

C     CHECK CORRECTNESS OF THE ANSWER

      IF (EPS.NE.1) THEN
        PRINT*,'* CAUTION * THE FINAL EPSILON IS EQUAL TO ',EPS
      END IF
      DO 80 NODE=1,N
        IF (SURPLUS(NODE).NE.0) THEN
          PRINT*,'NONZERO SURPLUS AT NODE ',NODE
        ENDIF
80    CONTINUE
      DO 90 ARC=1,NA
        COST(ARC)=COST(ARC)*(N+1)
        IF (F(ARC).GT.0) THEN
          IF (P(STARTN(ARC))-P(ENDN(ARC)).LT.-EPS+COST(ARC)) THEN
            PRINT*,'E-CS VIOLATED AT ARC ',ARC
          ENDIF
        ENDIF
        IF (F(ARC).LT.U(ARC)) THEN
          IF (P(STARTN(ARC))-P(ENDN(ARC)).GT.EPS+COST(ARC)) THEN
            PRINT*,'E-CS VIOLATED AT ARC ',ARC
          ENDIF
```

```
        ENDIF
90    CONTINUE
      PRINT *, ' '
      PRINT *, 'PROGRAM ENDED; PRESS <CR>'
      PAUSE
      STOP
      END



C     ***************************************************
      SUBROUTINE INIDAT
C     THIS SUBROUTINE USES THE DATA ARRAYS STARTN AND ENDN
C     TO CONSTRUCT AUXILIARY DATA ARRAYS FOUT, NXTOU, FIN, AND
C     NXTIN THAT ARE REQUIRED BY E-RELAX.  IN THIS SUBROUTINE WE
C     ARBITRARILY ORDER THE ARCS LEAVING EACH NODE AND STORE
C     THIS INFORMATION IN FOUT AND NXTOU.  SIMILARLY, WE ARBITRA-
C     RILLY ORDER THE ARCS ENTERING EACH NODE AND STORE THIS
C     INFORMATION IN FIN AND NXTIN.  AT THE COMPLETION OF THE
C     CONSTRUCTION, WE HAVE THAT
C
C         FOUT(I)    = FIRST ARC LEAVING NODE I.
C         NXTOU(J)   = NEXT ARC LEAVING THE HEAD NODE OF ARC J.
C         FIN(I)     = FIRST ARC ENTERING NODE I.
C         NXTIN(J)   = NEXT ARC ENTERING THE TAIL NODE OF ARC J.

      PARAMETER (MAXNODES=8000, MAXARCS=25000)
      IMPLICIT INTEGER (A-Z)
      COMMON /SCALARS/  N,NA,LARGE,FACTOR,EPS
      COMMON /BLK1/     STARTN
      COMMON /BLK2/     ENDN
      COMMON /BLK4/     FIN
      COMMON /BLK5/     FOUT
      COMMON /BLK6/     NXTIN
      COMMON /BLK7/     NXTOU
      INTEGER STARTN(MAXARCS)
      INTEGER ENDN(MAXARCS)
      INTEGER FIN(MAXNODES)
      INTEGER FOUT(MAXNODES)
      INTEGER NXTIN(MAXARCS)
      INTEGER NXTOU(MAXARCS)
      INTEGER FINALIN(MAXNODES),FINALOU(MAXNODES)

      DO 20 NODE=1,N
        FIN(NODE)=0
        FOUT(NODE)=0
        FINALIN(NODE)=0
        FINALOU(NODE)=0
20    CONTINUE

      DO 30 ARC=1,NA
        START=STARTN(ARC)
        END=ENDN(ARC)
        IF (FOUT(START).NE.0) THEN
```

```
          NXTOU(FINALOU(START))=ARC
        ELSE
          FOUT(START)=ARC
        END IF
        IF (FIN(END).NE.0) THEN
          NXTIN(FINALIN(END))=ARC
        ELSE
          FIN(END)=ARC
        END IF
        FINALOU(START)=ARC
        FINALIN(END)=ARC
        NXTIN(ARC)=0
        NXTOU(ARC)=0
30    CONTINUE

      RETURN
      END



C  ***************************************************************
C
C     SCALED E-RELAXATION
C     THIS VERSION USES A QUEUE TO SELECT NODES.
C     NODES JOIN THE QUEUE AT THE BOTTOM.
C
C  ***************************************************************

      SUBROUTINE EPS_RELAX
      IMPLICIT NONE
      INTEGER N,NA,LARGE,FACTOR,EPS,ARC,ARCF,ARCB
      INTEGER NCYC,NITER,NODE,PRICE,PRICEINCR
      INTEGER I,J,K,LN,CAPOUT,CAPIN,PREVNODE,LASTQUEUE
      INTEGER SURPI,REJI,NEXT,START,END,PSTART,PEND,MAXPRICE
      INTEGER RESID,DEL,FLOW,NXTNODE,TOTALITER
      INTEGER REFPRICE,PRJ,XP,REFPRJ,PRICEJ,THRESHSURPLUS,SFACTOR
      INTEGER NXTQUEUE(1)
      INTEGER STARTN(1),ENDN(1),U(1),F(1),SURPLUS(1),FIN(1),FOUT(1)
      INTEGER NXTIN(1),NXTOU(1),P(1)
      INTEGER FPUSHF(1),NXTPUSHF(1),FPUSHB(1),NXTPUSHB(1)
      INTEGER COST(1)

      COMMON /SCALARS/  N,NA,LARGE,FACTOR,EPS,
     &THRESHSURPLUS,SFACTOR
      COMMON /STATS/    NCYC,NITER,TOTALITER
      COMMON /BLK1/     STARTN
      COMMON /BLK2/     ENDN
      COMMON /UBOUND/   U
      COMMON /FLOW/     F
      COMMON /PRICES/   P
      COMMON /BLK3/     SURPLUS
      COMMON /BLK4/     FIN
      COMMON /BLK5/     FOUT
      COMMON /BLK6/     NXTIN
```

```
      COMMON /BLK7/     NXTOU
      COMMON /BLK11/    FPUSHF
      COMMON /BLK12/    NXTPUSHF
      COMMON /BLK13/    FPUSHB
      COMMON /BLK14/    NXTPUSHB
      COMMON /COST/     COST
      COMMON /QUEUE/    NXTQUEUE


C     INITIALIZE COUNT OF NUMBER OF UP ITERATIONS PERFORMED

      NITER = 0
      TOTALITER=0
      NCYC=0
      MAXPRICE =LARGE

C     REDUCE ARC CAPACITIES

         DO 40 NODE=1,N
            CAPOUT=0
            ARC=FOUT(NODE)
41          IF (ARC.GT.0) THEN
               CAPOUT=MIN(LARGE,CAPOUT+U(ARC))
               ARC=NXTOU(ARC)
               GO TO 41
            END IF
            CAPOUT=MIN(LARGE,CAPOUT-SURPLUS(NODE))
            IF (CAPOUT.LT.0) GOTO 400
            CAPIN=0
            ARC=FIN(NODE)
43          IF (ARC.GT.0) THEN
               IF (U(ARC) .GT. CAPOUT) THEN
                  U(ARC)=CAPOUT
               ENDIF
               CAPIN=MIN(LARGE,CAPIN+U(ARC))
               ARC=NXTIN(ARC)
               GO TO 43
            END IF
            CAPIN=MIN(LARGE,CAPIN+SURPLUS(NODE))
            IF (CAPIN.LT.0) GOTO 400
            ARC=FOUT(NODE)
45          IF (ARC.GT.0) THEN
               IF (U(ARC) .GT. CAPIN) THEN
                  U(ARC)=CAPIN
               ENDIF
               ARC=NXTOU(ARC)
               GO TO 45
            END IF
40       CONTINUE

C   SET ARC FLOWS TO SATISFY E-CS

        DO 49 ARC=1,NA
          START=STARTN(ARC)
```

```
          END=ENDN(ARC)
          PSTART=P(START)
          PEND=P(END)
          IF (PSTART.GE.PEND+COST(ARC)+EPS) THEN
            SURPLUS(START)=SURPLUS(START)-U(ARC)
            SURPLUS(END)=SURPLUS(END)+U(ARC)
            F(ARC)=U(ARC)
          ELSE
            F(ARC)=0
          END IF
49      CONTINUE

C       *********** START OF A NEW SCALING PHASE ***********

60        CONTINUE

C       ***** QUEUE INITIALIZATION *****

          DO 82 NODE=1,N-1
              NXTQUEUE(NODE)=NODE+1
82        CONTINUE
          NXTQUEUE(N)=1
          I=1
          PREVNODE=N
          LASTQUEUE=N

C      ************ START A NEW CYCLE OF UP ITERATIONS ************

100    CONTINUE

C      TAKE UP NEXT NODE (NODE I) FOR ITERATION

          SURPI=SURPLUS(I)
          IF (SURPI.GT.THRESHSURPLUS)  THEN

C     ARCF & ARCB ARE THE CURRENT VALUES OF THE STARTING ARCS OF THE
C     PUSH LISTS OF I

            TOTALITER=TOTALITER+1
            ARCF = FPUSHF(I)
            ARCB = FPUSHB(I)

            PRICE = P(I)
C           PRINT*,'NODE ',I,'   START PRICE = ',PRICE

115       IF ((ARCF .GT. 0) .OR. (ARCB .GT. 0)) THEN

C           START BY TRYING TO PUSH AWAY FLOW ON ARCS THAT WERE
C           ADMISSIBLE AT THE END OF THE LAST ITERATION (IF ANY)
C           AT THIS NODE.  WE MUST CHECK THAT THEY ARE STILL
C           ADMISSIBLE.


120               IF ((SURPI .GT. 0) .AND. (ARCF .GT. 0)) THEN
```

```
                J = ENDN(ARCF)
                IF (PRICE-P(J)-COST(ARCF).EQ.EPS) THEN
                   RESID = U(ARCF) - F(ARCF)
                   IF (RESID.GT.0) THEN
                    IF (SURPI .GE. RESID) THEN
                       F(ARCF) = U(ARCF)
                       SURPI = SURPI - RESID
                       SURPLUS(J) = SURPLUS(J) + RESID
                       ARCF = NXTPUSHF(ARCF)
                    ELSE
                       F(ARCF) = F(ARCF) + SURPI
                       SURPLUS(J) = SURPLUS(J) + SURPI
                       SURPI = 0
                    ENDIF
                    IF (SURPLUS(J).GT.THRESHSURPLUS) THEN
                     IF (NXTQUEUE(J).EQ.0) THEN
                      NXTQUEUE(PREVNODE)=J
                      NXTQUEUE(J)=I
                      PREVNODE=J
                     END IF
                    END IF
                  ELSE
                    ARCF = NXTPUSHF(ARCF)
                  ENDIF
                ELSE
                   ARCF = NXTPUSHF(ARCF)
                ENDIF
                GOTO 120
               ENDIF

121           IF ((SURPI .GT. 0) .AND. (ARCB .GT. 0)) THEN
               J = STARTN(ARCB)
               IF (PRICE-P(J)+COST(ARCB).EQ.EPS) THEN
                  RESID=F(ARCB)
                  IF (RESID.GT.0) THEN
                     IF (SURPI .GE. RESID) THEN
                        SURPI = SURPI - RESID
                        SURPLUS(J) = SURPLUS(J) + RESID
                        F(ARCB) = 0
                        ARCB = NXTPUSHB(ARCB)
                     ELSE
                        F(ARCB) = F(ARCB) - SURPI
                        SURPLUS(J) = SURPLUS(J) + SURPI
                        SURPI = 0
                     ENDIF
                     IF (SURPLUS(J).GT.THRESHSURPLUS) THEN
                      IF (NXTQUEUE(J).EQ.0) THEN
                       NXTQUEUE(PREVNODE)=J
                       NXTQUEUE(J)=I
                       PREVNODE=J
                      END IF
                     END IF
                   ELSE
                     ARCB = NXTPUSHB(ARCB)
```

```
                  ENDIF
               ELSE
                  ARCB = NXTPUSHB(ARCB)
               ENDIF
               GOTO 121
            ENDIF
         ENDIF

C   ***** END OF D-PUSHES; CHECK IF PRICE RISE IS NEEDED *****

        IF ((ARCF .EQ. 0) .AND. (ARCB .EQ. 0)) THEN

C    ******************DO A PRICE RISE *********************

               REFPRICE=PRICE
               PRICE = LARGE
               NITER=NITER+1
               ARC = FOUT(I)
111            IF (ARC .GT. 0) THEN
                 IF (F(ARC) .LT. U(ARC)) THEN
                    XP = P(ENDN(ARC))+COST(ARC)
                    IF (XP.LT.PRICE) THEN
                       PRICE = XP
                       ARCF = ARC
                       NXTPUSHF(ARC)=0
                    ELSE
                     IF (XP.EQ.PRICE) THEN
                       NXTPUSHF(ARC) = ARCF
                       ARCF = ARC
                     END IF
                    ENDIF
                 ENDIF
                 ARC = NXTOU(ARC)
                 GOTO 111
               ENDIF
               ARC = FIN(I)

112            IF (ARC .GT. 0) THEN
                 IF (F(ARC) .GT. 0) THEN
                    XP = P(STARTN(ARC))-COST(ARC)
                    IF (XP.LT.PRICE) THEN
                       PRICE = XP
                       ARCB = ARC
                       ARCF=0
                       NXTPUSHB(ARC)=0
                    ELSE
                     IF (XP.EQ.PRICE) THEN
                       NXTPUSHB(ARC) = ARCB
                       ARCB = ARC
                     END IF
                    ENDIF
                 ENDIF
                 ARC = NXTIN(ARC)
                 GOTO 112
```

```
              ENDIF

C     IF PRICE=LARGE, THE PUSH LIST IS LIKELY EMPTY, SO SET PRICE TO AT
LEAST THE
C     LEVEL OF THE BEST ARC

           IF (PRICE.EQ.LARGE) THEN
                ARCF=0
                ARCB=0

                ARC = FOUT(I)
211             IF (ARC.GT.0) THEN
                 XP = P(ENDN(ARC))+COST(ARC)
                 IF (XP.LT.PRICE) THEN
                   PRICE = XP
                 ELSE
                   IF (XP.GE.LARGE) THEN
                    PRINT*,'PRICE OF ',I,'HAS EXCEEDED THE INT. RANGE'
                    PAUSE
                    STOP
                   END IF
                 ENDIF
                 ARC = NXTOU(ARC)
                 GOTO 211
               ENDIF
               ARC = FIN(I)

212            IF (ARC.GT.0) THEN
                 XP = P(STARTN(ARC))-COST(ARC)
                 IF (XP.LT.PRICE) THEN
                    PRICE = XP
                 ELSE
                   IF (XP.GE.LARGE) THEN
                    PRINT*,'PRICE OF ',I,'HAS EXCEEDED THE INT. RANGE'
                    PAUSE
                    STOP
                   END IF
                 ENDIF
                 ARC = NXTIN(ARC)
                 GOTO 212
               ENDIF

               IF (SURPI.GT.0) GOTO 400
               IF (PRICE.LT.INT(LARGE/10)) PRICE=INT(LARGE/10)

           END IF


C     SET PRICE OF NODE I


           PRICE=PRICE+EPS
           P(I) = PRICE
           FPUSHF(I) = ARCF
```

```
            FPUSHB(I) = ARCB

X           PRICEINCR=PRICE-REFPRICE
X           IF (PRICEINCR.LE.0) THEN
X             PRINT*,'NONPOSITIVE PRICE INCREMENT=',PRICEINCR
X             PAUSE
X             STOP
X           ENDIF

C            END OF PRICE RISE; IF THERE IS STILL A LOT OF SURPLUS,
C            TRY AND DO SOME PUSHING.

            IF (SURPI.GT.THRESHSURPLUS) GOTO 115

          ELSE

C           IF NO PRICE RISE TOOK PLACE RESET THE START OF THE PUSH LISTS

            FPUSHF(I) = ARCF
            FPUSHB(I) = ARCB
          ENDIF

C       DO THE FINAL BOOKKEEPING OF THE ITERATION


          SURPLUS(I) = SURPI

C       IF THE PRICE IS TOO HIGH, THEN SOMETHING IS WRONG

X         IF (PRICE.GT.MAXPRICE) THEN
X           GO TO 400
X         ENDIF

         ENDIF

C        ********** END OF UP ITERATION STARTING AT NODE I *********

C        CHECK FOR THE END OF THE QUEUE. THIS STEP IS NOT NECESSARY FOR
C        THE ALGORITHM; IT IS INCLUDED FOR COLLECTING STATISTICS ABOUT
C        THE ALGORITHM'S BEHAVIOR, E.G. COUNTING THE NUMBER OF CYCLES

X          IF (I.EQ.LASTQUEUE) THEN
X           LASTQUEUE=PREVNODE
X           NCYC=NCYC+1
X           PRINT*,'CYCLE # ',NCYC,'   # OF PRICE RISES ',NITER
X          END IF


C        CHECK FOR TERMINATION OF SCALING PHASE. IF SCALING PHASE IS
C        NOT FINISHED, ADVANCE THE QUEUE AND RETURN TO TAKE ANOTHER NODE.

          NXTNODE=NXTQUEUE(I)
          IF (I.NE.NXTNODE) THEN
            NXTQUEUE(I)=0
```

```
          NXTQUEUE(PREVNODE)=NXTNODE
          I=NXTNODE
          GO TO 100
        END IF

C ************** END OF SUBPROBLEM (SCALING PHASE) **************

X      PRINT*,'END OF SCALING PHASE'

C      DO A DIAGNOSTIC CHECK

X      LN=0
X      DO 500 NODE=1,N
X        IF (SURPLUS(NODE).NE.0) THEN
X         LN=LN+1
X       ENDIF
500    CONTINUE
X      PRINT*,'NONZERO SURPLUS AT ',LN,' NODES '
X     DO 600 ARC=1,NA
X        IF (F(ARC).GT.0) THEN
X        IF (P(STARTN(ARC))-P(ENDN(ARC)).LT.-EPS+COST(ARC)) THEN
X          PRINT*,'E-CS VIOLATED AT ARC ',ARC
X        ENDIF
X       ENDIF
X        IF (F(ARC).LT.U(ARC)) THEN
X        IF (P(STARTN(ARC))-P(ENDN(ARC)).GT.EPS+COST(ARC)) THEN
X          PRINT*,'E-CS VIOLATED AT ARC ',ARC
X        ENDIF
X       ENDIF
600    CONTINUE
X       PRINT*,'END OF CHECK OF OLD PHASE'

C ****** IF EPSILON IS 1 TERMINATE; ELSE REDUCE EPSILON******

        IF (EPS.EQ.1) THEN
           RETURN
        ELSE
           EPS=INT(EPS/FACTOR)
           IF (EPS.LT.1) EPS=1
        END IF
        THRESHSURPLUS=INT(THRESHSURPLUS/SFACTOR)
        IF (EPS.EQ.1) THRESHSURPLUS=0

C      RESET THE FLOWS & THE PUSH LISTS; FIND THE MINIMAL PRICE

       XP=LARGE
       DO 800 NODE=1,N
         FPUSHF(NODE)=0
         FPUSHB(NODE)=0
         IF (P(NODE).LT.XP) XP=P(NODE)
800    CONTINUE

C      REDUCE ALL PRICES TO REDUCE DANGER OF OVERFLOW
```

```
       DEL=XP+INT(LARGE/10)

       IF (DEL.LT.0) DEL=0
       DO 810 NODE=1,N
         P(NODE)=P(NODE)-DEL
810    CONTINUE

X      PRINT*,'MODIFYING ARC FLOWS TO SATISFY E-CS'
       DO 900 ARC=1,NA
         START=STARTN(ARC)
         END=ENDN(ARC)
         PSTART=P(START)
         PEND=P(END)
         IF (PSTART.GT.PEND+EPS+COST(ARC)) THEN
           RESID=U(ARC)-F(ARC)
           IF (RESID.GT.0) THEN
             SURPLUS(START)=SURPLUS(START)-RESID
             SURPLUS(END)=SURPLUS(END)+RESID
             F(ARC)=U(ARC)
           END IF
         ELSE
          IF (PSTART.LT.PEND-EPS+COST(ARC)) THEN
            FLOW=F(ARC)
            IF (FLOW.GT.0) THEN
              SURPLUS(START)=SURPLUS(START)+FLOW
              SURPLUS(END)=SURPLUS(END)-FLOW
              F(ARC)=0
            END IF
          END IF
         END IF
900     CONTINUE


C     RETURN FOR ANOTHER PHASE

X     PRINT*,'CHECKING E-CS BEFORE STARTING NEW PHASE'
X      DO 700 ARC=1,NA
X        IF (F(ARC).GT.0) THEN
X         IF (P(STARTN(ARC))-P(ENDN(ARC)).LT.-EPS+COST(ARC)) THEN
X           PRINT*,'E-CS VIOLATED AT ARC ',ARC
X         ENDIF
X       ENDIF
X        IF (F(ARC).LT.U(ARC)) THEN
X         IF (P(STARTN(ARC))-P(ENDN(ARC)).GT.EPS+COST(ARC)) THEN
X           PRINT*,'E-CS VIOLATED AT ARC ',ARC
X         ENDIF
X       ENDIF
700    CONTINUE
       GO TO 60

400    PRINT*,'PROBLEM IS INFEASIBLE'
       PAUSE
       STOP
```

```
       END
```

$\e$-RELAX-N

This code is the same as $\e$-RELAX but allows
relaxation iterations starting from nodes with negative as well as
positive
surplus.

```
C     ** SAMPLE CALLING PROGRAM FOR E-RELAX **
C
C     THIS PROGRAM WILL READ A PROBLEM FILE IN STANDARD FORMAT
C     AND SOLVE IT USING E-RELAX.
C
C     THIS IS AN E-SCALED VERSION THAT USES A CYCLIC QUEUE
C
C     ITERATIONS ARE DONE FROM BOTH POSITIVE AND NEGATIVE
C     SURPLUS NODES. AN ADAPTIVE MECHANISM IS USED TO DECIDE
C     WHETHER TO ITERATE FROM A NEGATIVE SURPLUS NODE
C
C     ************************************************
C
      PARAMETER (MAXNODES=8000, MAXARCS=25000)
      IMPLICIT INTEGER (A-Z)
      COMMON /SCALARS/  N,NA,LARGE,FACTOR,EPS,
     &THRESHSURPLUS,SFACTOR
      COMMON /STATS/    NCYC,NITER,TOTALITER
      COMMON /BLK1/     STARTN
      COMMON /BLK2/     ENDN
      COMMON /UBOUND/   U
      COMMON /FLOW/     F
      COMMON /PRICES/   P
      COMMON /BLK3/     SURPLUS
      COMMON /BLK4/     FIN
      COMMON /BLK5/     FOUT
      COMMON /BLK6/     NXTIN
      COMMON /BLK7/     NXTOU
      COMMON /BLK11/    FPUSHF
      COMMON /BLK12/    NXTPUSHF
      COMMON /BLK13/    FPUSHB
      COMMON /BLK14/    NXTPUSHB
      COMMON /COST/     COST
      COMMON /QUEUE/    NXTQUEUE
      COMMON /STATUS/   STATUS
      INTEGER NXTQUEUE(MAXNODES)
      INTEGER STARTN(MAXARCS)
      INTEGER ENDN(MAXARCS)
      INTEGER U(MAXARCS)
      INTEGER F(MAXARCS)
      INTEGER SURPLUS(MAXNODES)
      INTEGER FIN(MAXNODES)
```

```
      INTEGER FOUT(MAXNODES)
      INTEGER NXTIN(MAXARCS)
      INTEGER NXTOU(MAXARCS)
      INTEGER P(MAXNODES)
      INTEGER FPUSHF(MAXNODES)
      INTEGER NXTPUSHF(MAXARCS)
      INTEGER FPUSHB(MAXNODES)
      INTEGER NXTPUSHB(MAXARCS)
      INTEGER COST(MAXARCS)
      INTEGER STATUS(MAXNODES)
      REAL*8 TCOST,TT,TIMER


      PRINT*,'E-RELAXATION METHOD FOR MIN COST FLOW'
      PRINT*,'************************************'
      PRINT *,'READING PROBLEM DATA'
      OPEN(13,FILE='FOR013.DAT',STATUS='OLD')
      REWIND(13)

C     READ NUMBER OF NODES AND ARCS

      READ(13,1010) N,NA

C     READ START, END, COST, AND CAPACITY OF EACH ARC
C     AND GENERATE MAX COST VALUE

      MAXCOST=0

      DO 5 I=1,NA
        READ(13,1020) STARTN(I),ENDN(I),COST(I),U(I)
        COST(I)=COST(I)*(N+1)
        IF (ABS(COST(I)).GT.MAXCOST) MAXCOST=ABS(COST(I))
13.   CONTINUE

C     READ SUPPLY OF EACH NODE

      DO 8 I=1,N
        READ(13,1000) SURPLUS(I)
8     CONTINUE

      ENDFILE(13)
      REWIND(13)

      PRINT*,'END OF READING'

1000  FORMAT(1I8)
1010  FORMAT(2I8)
1020  FORMAT(4I8)

C
      LARGE=1500000000

C  INITIALIZE NODE PRICES, SURPLUSES, ETC
```

```
      DO 10 NODE=1,N
        FPUSHF(NODE)=0
        FPUSHB(NODE)=0
        P(NODE)=0
        STATUS(NODE)=0
10    CONTINUE

C     IN THE FOLLOWING STATEMENTS, THE E-SCALING PARAMETERS ARE SET.
C     THE FOLLOWING RANGES ARE RECOMMENDED:
C     STARTING EPSILON: BETWEEN MAXCOST/20 TO MAXCOST/2
C     SCALING FACTOR: BETWEEN FOUR AND TEN

25    CONTINUE
      PRINT*,'ENTER STARTING EPSILON'
      PRINT*,'SHOULD BE BETWEEN 1 & ',MAXCOST
      READ*,EPS
      IF (EPS.LT.1) GO TO 25
      PRINT*,'ENTER SCALING FACTOR'
30    CONTINUE
      READ*,FACTOR
      IF ((EPS.GT.1).AND.(FACTOR.LE.1)) THEN
        PRINT*,'ENTER SCALING FACTOR; SHOULD BE GREATER THAN 1'
        GO TO 30
      END IF
C
        MAXSURPLUS=-LARGE
        DO 920 NODE=1,N
          IF (SURPLUS(NODE).GT.MAXSURPLUS) THEN
            MAXSURPLUS=SURPLUS(NODE)
          END IF
920     CONTINUE

C     SET THE PARAMETER SSCALE TO 1 TO ALLOW SURPLUS SCALING
C     A HEURISTIC SCHEME IS USED TO SET THE SFACTOR AND
C     THRESHSURPLUS PARAMETERS THAT CONTROL SURPLUS SCALING

      SSCALE=1

      IF (SSCALE.EQ.1) THEN
        SFACTOR=2+INT(FACTOR*MAXSURPLUS/MAXCOST)
        IF (SFACTOR.LT.4) SFACTOR=4
        THRESHSURPLUS=INT(MAXSURPLUS/SFACTOR)
        IF (EPS.EQ.1) THRESHSURPLUS=0
      ELSE
        THRESHSURPLUS=0
      END IF

      PRINT*,'INITIALIZING DATA STRUCTURES'
      CALL INIDAT

      PRINT *,'*********************************'
      PRINT *,'CALLING E-RELAX FOR MCF (+ AND - SURPLUS NODES ITERATED)'
      TIMER = LONG(362)
      CALL EPS_RELAX_N
```

```
      TT =(LONG(362) - TIMER)/60
      PRINT*, 'TOTAL TIME = ',TT, ' secs.'

      TCOST=0
      DO 330 I=1,NA
330     TCOST=TCOST+F(I)*COST(I)/(N+1)
      WRITE(9,1100) TCOST
1100  FORMAT(' ','OPTIMAL COST         =',F14.2)

      PRINT *,'*********************************'
      PRINT *,'# OF ITERATIONS = ',TOTALITER
      PRINT *,'# OF S. N. PRICE RISES = ',NITER
      PRINT *,'*********************************'

C     CHECK CORRECTNESS OF THE ANSWER

      IF (EPS.NE.1) THEN
        PRINT*,'* CAUTION * THE FINAL EPSILON IS EQUAL TO ',EPS
      END IF
      DO 80 NODE=1,N
        IF (SURPLUS(NODE).NE.0) THEN
          PRINT*,'NONZERO SURPLUS AT NODE ',NODE
        ENDIF
80    CONTINUE
      DO 90 ARC=1,NA
        IF (F(ARC).GT.0) THEN
          IF (P(STARTN(ARC))-P(ENDN(ARC)).LT.-EPS+COST(ARC)) THEN
            PRINT*,'E-CS VIOLATED AT ARC ',ARC
          ENDIF
        ENDIF
        IF (F(ARC).LT.U(ARC)) THEN
          IF (P(STARTN(ARC))-P(ENDN(ARC)).GT.EPS+COST(ARC)) THEN
            PRINT*,'E-CS VIOLATED AT ARC ',ARC
          ENDIF
        ENDIF
90    CONTINUE
      PRINT *, ' '
      PRINT *, 'PROGRAM ENDED; PRESS <CR>'
      PAUSE
      STOP
      END



C     ************************************************
      SUBROUTINE INIDAT
C     THIS SUBROUTINE USES THE DATA ARRAYS STARTN AND ENDN
C     TO CONSTRUCT AUXILIARY DATA ARRAYS FOUT, NXTOU, FIN, AND
C     NXTIN THAT ARE REQUIRED BY E-RELAX-N.  IN THIS SUBROUTINE WE
C     ARBITRARILY ORDER THE ARCS LEAVING EACH NODE AND STORE
C     THIS INFORMATION IN FOUT AND NXTOU.  SIMILARLY, WE ARBITRA-
C     RILLY ORDER THE ARCS ENTERING EACH NODE AND STORE THIS
C     INFORMATION IN FIN AND NXTIN.  AT THE COMPLETION OF THE
C     CONSTRUCTION, WE HAVE THAT
```

```
C
C         FOUT(I)    = FIRST ARC LEAVING NODE I.
C         NXTOU(J)   = NEXT ARC LEAVING THE HEAD NODE OF ARC J.
C         FIN(I)     = FIRST ARC ENTERING NODE I.
C         NXTIN(J)   = NEXT ARC ENTERING THE TAIL NODE OF ARC J.

      PARAMETER (MAXNODES=8000, MAXARCS=25000)
      IMPLICIT INTEGER (A-Z)
      COMMON /SCALARS/  N,NA,LARGE,FACTOR,EPS
      COMMON /BLK1/     STARTN
      COMMON /BLK2/     ENDN
      COMMON /BLK4/     FIN
      COMMON /BLK5/     FOUT
      COMMON /BLK6/     NXTIN
      COMMON /BLK7/     NXTOU
      INTEGER STARTN(MAXARCS)
      INTEGER ENDN(MAXARCS)
      INTEGER FIN(MAXNODES)
      INTEGER FOUT(MAXNODES)
      INTEGER NXTIN(MAXARCS)
      INTEGER NXTOU(MAXARCS)
      INTEGER FINALIN(MAXNODES),FINALOU(MAXNODES)

      DO 20 NODE=1,N
        FIN(NODE)=0
        FOUT(NODE)=0
        FINALIN(NODE)=0
        FINALOU(NODE)=0
20    CONTINUE

      DO 30 ARC=1,NA
        START=STARTN(ARC)
        END=ENDN(ARC)
        IF (FOUT(START).NE.0) THEN
          NXTOU(FINALOU(START))=ARC
        ELSE
          FOUT(START)=ARC
        END IF
        IF (FIN(END).NE.0) THEN
          NXTIN(FINALIN(END))=ARC
        ELSE
          FIN(END)=ARC
        END IF
        FINALOU(START)=ARC
        FINALIN(END)=ARC
        NXTIN(ARC)=0
        NXTOU(ARC)=0
30    CONTINUE

      RETURN
      END
```

```
C  ***************************************************************
C
C     SCALED E-RELAXATION
C     THIS VERSION USES A QUEUE TO SELECT NODES.
C     NODES JOIN THE QUEUE AT THE BOTTOM.
C     USES BOTH POSITIVE AND NEGATIVE SURPLUS ITERATIONS.
C
C  ***************************************************************

      SUBROUTINE EPS_RELAX_N
      IMPLICIT NONE
      INTEGER N,NA,LARGE,FACTOR,EPS,ARC,ARCF,ARCB
      INTEGER NCYC,NITER,NODE,PRICE,PRICEINCR
      INTEGER I,J,K,LN,CAPOUT,CAPIN,PREVNODE,LASTQUEUE
      INTEGER SURPI,REJI,NEXT,START,END,PSTART,PEND,MAXPRICE
      INTEGER RESID,DEL,FLOW,NXTNODE,TOTALITER,PARSTAT
      INTEGER REFPRICE,PRJ,XP,REFPRJ,PRICEJ,THRESHSURPLUS,SFACTOR
      INTEGER NXTQUEUE(1)
      INTEGER STARTN(1),ENDN(1),U(1),F(1),SURPLUS(1),FIN(1),FOUT(1)
      INTEGER NXTIN(1),NXTOU(1),P(1)
      INTEGER FPUSHF(1),NXTPUSHF(1),FPUSHB(1),NXTPUSHB(1)
      INTEGER COST(1),STATUS(1)

      COMMON /SCALARS/  N,NA,LARGE,FACTOR,EPS,
     &THRESHSURPLUS,SFACTOR
      COMMON /STATS/    NCYC,NITER,TOTALITER
      COMMON /BLK1/     STARTN
      COMMON /BLK2/     ENDN
      COMMON /UBOUND/   U
      COMMON /FLOW/     F
      COMMON /PRICES/   P
      COMMON /BLK3/     SURPLUS
      COMMON /BLK4/     FIN
      COMMON /BLK5/     FOUT
      COMMON /BLK6/     NXTIN
      COMMON /BLK7/     NXTOU
      COMMON /BLK11/    FPUSHF
      COMMON /BLK12/    NXTPUSHF
      COMMON /BLK13/    FPUSHB
      COMMON /BLK14/    NXTPUSHB
      COMMON /COST/     COST
      COMMON /QUEUE/    NXTQUEUE
      COMMON /STATUS/   STATUS


C
C     INITIALIZE COUNT OF NUMBER OF UP ITERATIONS PERFORMED
C
      NITER = 0
      TOTALITER=0
      NCYC=0
      MAXPRICE =LARGE
C
C     REDUCE ARC CAPACITIES
```

```
C
         DO 40 NODE=1,N
            CAPOUT=0
            ARC=FOUT(NODE)
41          IF (ARC.GT.0) THEN
               CAPOUT=MIN(LARGE,CAPOUT+U(ARC))
               ARC=NXTOU(ARC)
               GO TO 41
            END IF
            CAPOUT=MIN(LARGE,CAPOUT-SURPLUS(NODE))
            IF (CAPOUT.LT.0) GOTO 400
            CAPIN=0
            ARC=FIN(NODE)
43          IF (ARC.GT.0) THEN
               IF (U(ARC) .GT. CAPOUT) THEN
                  U(ARC)=CAPOUT
               ENDIF
               CAPIN=MIN(LARGE,CAPIN+U(ARC))
               ARC=NXTIN(ARC)
               GO TO 43
            END IF
            CAPIN=MIN(LARGE,CAPIN+SURPLUS(NODE))
            IF (CAPIN.LT.0) GOTO 400
            ARC=FOUT(NODE)
45          IF (ARC.GT.0) THEN
               IF (U(ARC) .GT. CAPIN) THEN
                  U(ARC)=CAPIN
               ENDIF
               ARC=NXTOU(ARC)
               GO TO 45
            END IF
40       CONTINUE
C
C   SET ARC FLOWS TO SATISFY E-CS
C
        DO 49 ARC=1,NA
          START=STARTN(ARC)
          END=ENDN(ARC)
          PSTART=P(START)
          PEND=P(END)
          IF (PSTART.GE.PEND+COST(ARC)+EPS) THEN
            SURPLUS(START)=SURPLUS(START)-U(ARC)
            SURPLUS(END)=SURPLUS(END)+U(ARC)
            F(ARC)=U(ARC)
          ELSE
            F(ARC)=0
          END IF
49      CONTINUE

C       *********** START OF A NEW SCALING PHASE ***********

60        CONTINUE

C      ***** QUEUE INITIALIZATION *****
```

```
        DO 82 NODE=1,N-1
            NXTQUEUE(NODE)=NODE+1
82      CONTINUE
        NXTQUEUE(N)=1
        I=1
        PREVNODE=N
        LASTQUEUE=N

C   SET THE STATUS PARAMETER FOR ALLOWING A NEG. SURPLUS ITERATION

      PARSTAT=0

C     ************ START A NEW CYCLE OF UP ITERATIONS ************

100   CONTINUE

C     TAKE UP NEXT NODE (NODE I) FOR ITERATION

        SURPI=SURPLUS(I)
        IF (SURPI.GT.THRESHSURPLUS)  THEN

C    ARCF & ARCB ARE THE CURRENT VALUES OF THE STARTING ARCS OF THE
C    PUSH LISTS OF I

           TOTALITER=TOTALITER+1
           ARCF = FPUSHF(I)
           ARCB = FPUSHB(I)
           STATUS(I)=STATUS(I)+1

           PRICE = P(I)

115      IF ((ARCF .GT. 0) .OR. (ARCB .GT. 0)) THEN

C     START BY TRYING TO PUSH AWAY FLOW ON ARCS THAT WERE
C     ADMISSIBLE AT THE END OF THE LAST ITERATION (IF ANY)
C     AT THIS NODE.  WE MUST CHECK THAT THEY ARE STILL
C     ADMISSIBLE.


120             IF ((SURPI .GT. 0) .AND. (ARCF .GT. 0)) THEN
                  J = ENDN(ARCF)
                  IF (PRICE-P(J)-COST(ARCF).EQ.EPS) THEN
                     RESID = U(ARCF) - F(ARCF)
                     IF (RESID.GT.0) THEN
                      IF (SURPLUS(J).LT.0) PARSTAT=PARSTAT+1
                      IF (SURPI .GE. RESID) THEN
                         F(ARCF) = U(ARCF)
                         SURPI = SURPI - RESID
                         SURPLUS(J) = SURPLUS(J) + RESID
                         ARCF = NXTPUSHF(ARCF)
                      ELSE
                         F(ARCF) = F(ARCF) + SURPI
                         SURPLUS(J) = SURPLUS(J) + SURPI
```

```
                        SURPI = 0
                      ENDIF
                      IF (ABS(SURPLUS(J)).GT.THRESHSURPLUS) THEN
                       IF (NXTQUEUE(J).EQ.0) THEN
                        NXTQUEUE(PREVNODE)=J
                        NXTQUEUE(J)=I
                        PREVNODE=J
                       END IF
                      END IF
                    ELSE
                      ARCF = NXTPUSHF(ARCF)
                    ENDIF
                  ELSE
                    ARCF = NXTPUSHF(ARCF)
                  ENDIF
                  GOTO 120
                ENDIF

121           IF ((SURPI .GT. 0) .AND. (ARCB .GT. 0)) THEN
                J = STARTN(ARCB)
                IF (PRICE-P(J)+COST(ARCB).EQ.EPS) THEN
                   RESID=F(ARCB)
                   IF (RESID.GT.0) THEN
                      IF (SURPLUS(J).LT.0) PARSTAT=PARSTAT+1
                      IF (SURPI .GE. RESID) THEN
                         SURPI = SURPI - RESID
                         SURPLUS(J) = SURPLUS(J) + RESID
                         F(ARCB) = 0
                         ARCB = NXTPUSHB(ARCB)
                      ELSE
                         F(ARCB) = F(ARCB) - SURPI
                         SURPLUS(J) = SURPLUS(J) + SURPI
                         SURPI = 0
                      ENDIF
                      IF (ABS(SURPLUS(J)).GT.THRESHSURPLUS) THEN
                       IF (NXTQUEUE(J).EQ.0) THEN
                        NXTQUEUE(PREVNODE)=J
                        NXTQUEUE(J)=I
                        PREVNODE=J
                       END IF
                      END IF
                    ELSE
                      ARCB = NXTPUSHB(ARCB)
                    ENDIF
                  ELSE
                     ARCB = NXTPUSHB(ARCB)
                  ENDIF
                  GOTO 121
                ENDIF
           ENDIF

C   ***** END OF D-PUSHES; CHECK IF PRICE RISE IS NEEDED *****

          IF ((ARCF .EQ. 0) .AND. (ARCB .EQ. 0)) THEN
```

```
C    ******************DO A PRICE RISE *********************

                REFPRICE=PRICE
                PRICE = LARGE
                NITER=NITER+1
                ARC = FOUT(I)
111             IF (ARC .GT. 0) THEN
                  IF (F(ARC) .LT. U(ARC)) THEN
                     XP = P(ENDN(ARC))+COST(ARC)
                     IF (XP.LT.PRICE) THEN
                        PRICE = XP
                        ARCF = ARC
                        NXTPUSHF(ARC)=0
                     ELSE
                      IF (XP.EQ.PRICE) THEN
                        NXTPUSHF(ARC) = ARCF
                        ARCF = ARC
                       END IF
                     ENDIF
                  ENDIF
                  ARC = NXTOU(ARC)
                  GOTO 111
               ENDIF
               ARC = FIN(I)

112             IF (ARC .GT. 0) THEN
                  IF (F(ARC) .GT. 0) THEN
                     XP = P(STARTN(ARC))-COST(ARC)
                     IF (XP.LT.PRICE) THEN
                        PRICE = XP
                        ARCB = ARC
                        ARCF=0
                        NXTPUSHB(ARC)=0
                     ELSE
                      IF (XP.EQ.PRICE) THEN
                        NXTPUSHB(ARC) = ARCB
                        ARCB = ARC
                       END IF
                     ENDIF
                  ENDIF
                  ARC = NXTIN(ARC)
                  GOTO 112
               ENDIF

C     IF PRICE=LARGE, THE PUSH LIST IS LIKELY EMPTY, SO SET PRICE TO AT
LEAST THE
C     LEVEL OF THE BEST ARC

           IF (PRICE.EQ.LARGE) THEN
                ARCF=0
                ARCB=0

                ARC = FOUT(I)
```

```
211             IF (ARC.GT.0) THEN
                  XP = P(ENDN(ARC))+COST(ARC)
                  IF (XP.LT.PRICE) THEN
                     PRICE = XP
                  ELSE
                     PRINT*,'PRICE OF ',I,'HAS EXCEEDED THE INT. RANGE'
                     PAUSE
                     STOP
                  ENDIF
                  ARC = NXTOU(ARC)
                  GOTO 211
                ENDIF
                ARC = FIN(I)

212             IF (ARC.GT.0) THEN
                  XP = P(STARTN(ARC))-COST(ARC)
                  IF (XP.LT.PRICE) THEN
                     PRICE = XP
                  ELSE
                     PRINT*,'PRICE OF ',I,'HAS EXCEEDED THE INT. RANGE'
                     PAUSE
                     STOP
                  ENDIF
                  ARC = NXTIN(ARC)
                  GOTO 212
                ENDIF

                IF (SURPI.GT.0) GOTO 400
                IF (PRICE.LT.INT(LARGE/10)) PRICE=INT(LARGE/10)

            END IF



C     SET PRICE OF NODE I

            PRICE=PRICE+EPS
            P(I) = PRICE
            FPUSHF(I) = ARCF
            FPUSHB(I) = ARCB

C            END OF PRICE RISE; IF THERE IS STILL A LOT OF SURPLUS,
C            TRY AND DO SOME PUSHING.

            IF (SURPI.GT.THRESHSURPLUS) GOTO 115

          ELSE

C           IF NO PRICE RISE TOOK PLACE RESET THE START OF THE PUSH LISTS

            FPUSHF(I) = ARCF
            FPUSHB(I) = ARCB
          ENDIF
```

```
C     DO THE FINAL BOOKKEEPING OF THE ITERATION


        SURPLUS(I) = SURPI

C         IF THE PRICE IS TOO HIGH, THEN SOMETHING IS WRONG

X       IF (PRICE.GT.MAXPRICE) THEN
X         GO TO 400
X       ENDIF

       ENDIF


C    CHECK FOR A NEGATIVE SURPLUS ITERATION

       IF (SURPI.LT.(-THRESHSURPLUS)) THEN

           TOTALITER=TOTALITER+1
           ARCF = FPUSHF(I)
           ARCB = FPUSHB(I)

           PRICE = P(I)

1115       IF ((ARCF .GT. 0) .OR. (ARCB .GT. 0)) THEN

C     START BY TRYING TO PUSH AWAY FLOW ON ARCS THAT WERE
C     ADMISSIBLE AT THE END OF THE LAST ITERATION (IF ANY)
C     AT THIS NODE.  WE MUST CHECK THAT THEY ARE STILL
C     ADMISSIBLE.


1120             IF ((SURPI .LT. 0) .AND. (ARCF .GT. 0)) THEN
                  J = ENDN(ARCF)
                  IF (PRICE-P(J)-COST(ARCF).EQ.(-EPS)) THEN
                    RESID = F(ARCF)
                    IF (RESID.GT.0) THEN
                     IF (SURPLUS(J).GT.0) PARSTAT=PARSTAT+1
                     IF ((-SURPI) .GE. RESID) THEN
                        F(ARCF) = 0
                        SURPI = SURPI + RESID
                        SURPLUS(J) = SURPLUS(J) - RESID
                        ARCF = NXTPUSHF(ARCF)
                     ELSE
                        F(ARCF) = F(ARCF) + SURPI
                        SURPLUS(J) = SURPLUS(J) + SURPI
                        SURPI = 0
                     ENDIF
                     IF (ABS(SURPLUS(J)).GT.THRESHSURPLUS) THEN
            IF ((NXTQUEUE(J).EQ.0).AND.(STATUS(J).LE.(PARSTAT/2))) THEN
                       NXTQUEUE(PREVNODE)=J
                       NXTQUEUE(J)=I
                       PREVNODE=J
                      END IF
```

```
                    END IF
                   ELSE
                    ARCF = NXTPUSHF(ARCF)
                   ENDIF
                 ELSE
                   ARCF = NXTPUSHF(ARCF)
                 ENDIF
                 GOTO 1120
               ENDIF

1121          IF ((SURPI .LT. 0) .AND. (ARCB .GT. 0)) THEN
              J = STARTN(ARCB)
              IF (PRICE-P(J)+COST(ARCB).EQ.(-EPS)) THEN
                 RESID=U(ARCB)-F(ARCB)
                 IF (RESID.GT.0) THEN
                    IF (SURPLUS(J).GT.0) PARSTAT=PARSTAT+1
                    IF ((-SURPI) .GE. RESID) THEN
                       SURPI = SURPI + RESID
                       SURPLUS(J) = SURPLUS(J) - RESID
                       F(ARCB) = U(ARCB)
                       ARCB = NXTPUSHB(ARCB)
                    ELSE
                       F(ARCB) = F(ARCB) - SURPI
                       SURPLUS(J) = SURPLUS(J) + SURPI
                       SURPI = 0
                    ENDIF
                    IF (ABS(SURPLUS(J)).GT.THRESHSURPLUS) THEN
          IF ((NXTQUEUE(J).EQ.0).AND.(STATUS(J).LE.(PARSTAT/2))) THEN
                      NXTQUEUE(PREVNODE)=J
                      NXTQUEUE(J)=I
                      PREVNODE=J
                     END IF
                    END IF
                  ELSE
                    ARCB = NXTPUSHB(ARCB)
                  ENDIF
               ELSE
                  ARCB = NXTPUSHB(ARCB)
               ENDIF
               GOTO 1121
             ENDIF
          ENDIF

C   ***** END OF D-PUSHES; CHECK IF PRICE RISE IS NEEDED *****

         IF ((ARCF .EQ. 0) .AND. (ARCB .EQ. 0)) THEN

C    ******************DO A PRICE RISE ********************

                REFPRICE=PRICE
                PRICE = -LARGE
                NITER=NITER+1
                ARC = FOUT(I)
1111             IF (ARC .GT. 0) THEN
```

```
                  IF (F(ARC) .GT. 0) THEN
                     XP = P(ENDN(ARC))+COST(ARC)
                     IF (XP.GT.PRICE) THEN
                        PRICE = XP
                        ARCF = ARC
                        NXTPUSHF(ARC)=0
                     ELSE
                      IF (XP.EQ.PRICE) THEN
                        NXTPUSHF(ARC) = ARCF
                        ARCF = ARC
                      END IF
                     ENDIF
                  ENDIF
                  ARC = NXTOU(ARC)
                  GOTO 1111
               ENDIF
               ARC = FIN(I)

1112            IF (ARC .GT. 0) THEN
                 IF (F(ARC) .LT. U(ARC)) THEN
                    XP = P(STARTN(ARC))-COST(ARC)
                    IF (XP.GT.PRICE) THEN
                       PRICE = XP
                       ARCB = ARC
                       ARCF=0
                       NXTPUSHB(ARC)=0
                    ELSE
                     IF (XP.EQ.PRICE) THEN
                       NXTPUSHB(ARC) = ARCB
                       ARCB = ARC
                     END IF
                    ENDIF
                 ENDIF
                 ARC = NXTIN(ARC)
                 GOTO 1112
              ENDIF

C     IF PRICE=-LARGE, THE PUSH LIST IS LIKELY EMPTY, SO SET PRICE TO AT
MOST THE
C     LEVEL OF THE BEST ARC

           IF (PRICE.EQ.-LARGE) THEN
                ARCF=0
                ARCB=0

                ARC = FOUT(I)
1211            IF (ARC .GT. 0) THEN
                  XP = P(ENDN(ARC))+COST(ARC)
                  IF (XP.GT.PRICE) THEN
                   PRICE = XP
                  ELSE
                   IF (XP.LE.-LARGE) THEN
                    PRINT*,'PRICE OF ',I,'HAS EXCEEDED THE INT. RANGE'
                    PAUSE
```

```
                    STOP
                   END IF
                  END IF
                  ARC = NXTOU(ARC)
                  GOTO 1211
                END IF
                ARC = FIN(I)

1212           IF (ARC .GT. 0) THEN
                 XP = P(STARTN(ARC))-COST(ARC)
                 IF (XP.GT.PRICE) THEN
                   PRICE = XP
                 ELSE
                   IF (XP.LE.-LARGE) THEN
                    PRINT*,'PRICE OF ',I,'HAS EXCEEDED THE INT. RANGE'
                    PAUSE
                    STOP
                   END IF
                 END IF
                 ARC = NXTIN(ARC)
                 GOTO 1212
                ENDIF

                IF (SURPI.LT.0) GOTO 400
                IF (PRICE.GT.-INT(LARGE/10)) PRICE=-INT(LARGE/10)

           END IF



C     SET PRICE OF NODE I

           PRICE=PRICE-EPS
           P(I) = PRICE
           FPUSHF(I) = ARCF
           FPUSHB(I) = ARCB

C     END OF PRICE RISE; IF THERE IS STILL A LOT OF SURPLUS,
C     TRY AND DO SOME PUSHING.

           IF (SURPI.LT.(-THRESHSURPLUS)) GOTO 1115

         ENDIF


C     DO THE FINAL BOOKKEEPING OF THE ITERATION


          SURPLUS(I) = SURPI

        END IF
```

```
C      ********** END OF ITERATION STARTING AT NODE I *********

C      CHECK FOR THE END OF THE QUEUE. THIS STEP IS NOT NECESSARY FOR
C      THE ALGORITHM; IT IS INCLUDED FOR COLLECTING STATISTICS ABOUT
C      THE ALGORITHM'S BEHAVIOR, E.G. COUNTING THE NUMBER OF CYCLES

         IF (I.EQ.LASTQUEUE) THEN
          LASTQUEUE=PREVNODE
          NCYC=NCYC+1
X         PRINT*,'CYCLE # ',NCYC,'   # OF PRICE RISES ',NITER
         END IF

C      CHECK FOR TERMINATION OF SCALING PHASE. IF SCALING PHASE IS
C      NOT FINISHED, ADVANCE THE QUEUE AND RETURN TO TAKE ANOTHER NODE.

         NXTNODE=NXTQUEUE(I)
         IF (I.NE.NXTNODE) THEN
           NXTQUEUE(I)=0
           NXTQUEUE(PREVNODE)=NXTNODE
           I=NXTNODE
           GO TO 100
         END IF

C ************** END OF SUBPROBLEM (SCALING PHASE) **************

X      PRINT*,'END OF SCALING PHASE'

C      DO A DIAGNOSTIC CHECK

X      LN=0
X      DO 500 NODE=1,N
X        IF (SURPLUS(NODE).NE.0) THEN
X         LN=LN+1
X       ENDIF
500    CONTINUE
X      PRINT*,'NONZERO SURPLUS AT ',LN,' NODES '
X     DO 600 ARC=1,NA
X        IF (F(ARC).GT.0) THEN
X        IF (P(STARTN(ARC))-P(ENDN(ARC)).LT.-EPS+COST(ARC)) THEN
X          PRINT*,'E-CS VIOLATED AT ARC ',ARC
X        ENDIF
X       ENDIF
X        IF (F(ARC).LT.U(ARC)) THEN
X        IF (P(STARTN(ARC))-P(ENDN(ARC)).GT.EPS+COST(ARC)) THEN
X          PRINT*,'E-CS VIOLATED AT ARC ',ARC
X        ENDIF
X       ENDIF
600    CONTINUE
X       PRINT*,'END OF CHECK OF OLD PHASE'

C ****** IF EPSILON IS 1 TERMINATE; ELSE REDUCE EPSILON******

         IF (EPS.EQ.1) THEN
            RETURN
```

```
         ELSE
            EPS=INT(EPS/FACTOR)
            IF (EPS.LT.1) EPS=1
         END IF
         THRESHSURPLUS=INT(THRESHSURPLUS/SFACTOR)
         IF (EPS.EQ.1) THRESHSURPLUS=0

C      RESET THE FLOWS & THE PUSH LISTS

        DO 800 NODE=1,N
          FPUSHF(NODE)=0
          FPUSHB(NODE)=0
          STATUS(NODE)=0
800     CONTINUE

        DO 900 ARC=1,NA
          START=STARTN(ARC)
          END=ENDN(ARC)
          PSTART=P(START)
          PEND=P(END)
          IF (PSTART.GT.PEND+EPS+COST(ARC)) THEN
            RESID=U(ARC)-F(ARC)
            IF (RESID.GT.0) THEN
              SURPLUS(START)=SURPLUS(START)-RESID
              SURPLUS(END)=SURPLUS(END)+RESID
              F(ARC)=U(ARC)
            END IF
          ELSE
           IF (PSTART.LT.PEND-EPS+COST(ARC)) THEN
             FLOW=F(ARC)
             IF (FLOW.GT.0) THEN
               SURPLUS(START)=SURPLUS(START)+FLOW
               SURPLUS(END)=SURPLUS(END)-FLOW
               F(ARC)=0
             END IF
           END IF
          END IF
900      CONTINUE


C     RETURN FOR ANOTHER PHASE

X     PRINT*,'CHECKING E-CS BEFORE STARTING NEW PHASE'
X      DO 700 ARC=1,NA
X        IF (F(ARC).GT.0) THEN
X         IF (P(STARTN(ARC))-P(ENDN(ARC)).LT.-EPS+COST(ARC)) THEN
X           PRINT*,'E-CS VIOLATED AT ARC ',ARC
X         ENDIF
X       ENDIF
X        IF (F(ARC).LT.U(ARC)) THEN
X         IF (P(STARTN(ARC))-P(ENDN(ARC)).GT.EPS+COST(ARC)) THEN
X           PRINT*,'E-CS VIOLATED AT ARC ',ARC
X         ENDIF
X       ENDIF
```

```
700    CONTINUE
       GO TO 60

400    PRINT*,'PROBLEM IS INFEASIBLE'
       PAUSE
       STOP
       END
```

SLF

Small Label First (SLF) code for finding shortest path from one origin to all destinations, as per the paper "A Simple and Fast Label Correcting Method for Shortest Paths," Networks, Vol. 23, 1993, pp. 703-709, by D. P. Bertsekas.

```
C    ************************************************************
C    SLF ALGORITHM TO FIND SHORTEST PATH FROM ONE ORIGIN TO ALL
DESTINATIONS
C    ************************************************************
C
C ALL THE PARAMETERS ARE INTEGER
C
C THE ONLY MACHINE DEPENDENT CONSTANT USED IS INF
C
C*********************************************************************


      PARAMETER(MAXNODES=10000, MAXARCS=100000)

      INTEGER FOUT,NXTOUT,D,P,Q,R,HP,Y,X,T2,DEST,COUNT
      REAL TT1,TT2,TCOST
      DIMENSION FOUT(MAXNODES),D(MAXNODES),P(MAXNODES),HP(MAXNODES)
      DIMENSION Q(MAXNODES)
      DIMENSION LNGT(MAXARCS),ND(MAXARCS),NXTOUT(MAXARCS)
      DATA INF/999999999/

      CALL READ(N,M,FOUT,NXTOUT,ND,LNGT)

      DO 10 I=1,N
        Q(I) = 0
        P(I) = 0
10    CONTINUE
C
C    INITIALIZE ORIGIN AND QUEUE OF DESTINATIONS
C

      PRINT*,'THE NUMBER OF NODES IS = ',N
      PRINT*,'ENTER THE ORIGIN NODE'
      READ*,R
```

```
      PRINT *,'CALLING SLF ALGORITHM TO SOLVE THE PROBLEM'
      PRINT*,'****************************************'
C
C     GET STARTING TIME FOR THE MAC II
C
      TT1 = LONG(362)/60.0
C

        CALL SLF(FOUT,NXTOUT,ND,LNGT,D,P,Q,N,INF,R,COUNT)
C
C     GET ENDING TIME FOR THE MAC II
C
      TT2 = LONG(362)/60.0 - TT1
      PRINT *,'FINISHED --- TOTAL CPU TIME', TT2,' SECS'
      PRINT*,'****************************************'
      PRINT*,'THE ORIGIN NODE IS ',R
C

      PRINT*,'NUMBER OF ITERATIONS ',COUNT
      PRINT*,'SH. DIST. OF DESTINATION ',N,' IS ',D(N)

45    PRINT*,'ENTER OTHER DESTINATION NODE (0 IF DONE)'
      READ*,DEST
      IF ((DEST.LT.0).OR.(DEST.GT.N)) GOTO 45
      IF (DEST.EQ.R) GOTO 45
      IF (DEST.NE.0) THEN
        PRINT*,'SHORTEST DISTANCE TO',DEST,' = ',D(DEST)
        GOTO 45
      END IF



      STOP
      END
C
C
      SUBROUTINE SLF(FOUT,NXTOUT,ND,LNGT,D,P,Q,N,INF,R,COUNT)
C*****************************************************************
C
C    ROUTINE SLF
C
C    SLF ALGORITHM BASED ON PAPER:
C    "A SIMLPLE AND FAST LABEL CORRECTING ALGORITHM FOR SHORTEST PATHS"
C     NETWORKS, VOL. 23, 1993, PP. 703-709, BY
C     DIMITRI P. BERTSEKAS
C
C*****************************************************************
C 1) FINDS A SHORTEST PATH TREE ROOTED AT NODE R AND THE SHORTEST
C    DISTANCES
C 2) IS BASED ON THE SLF METHOD WITH THE SET Q IMPLEMENTED
C    AS A SINGLE QUEUE Q(.)
C
C MEANING OF THE INPUT PARAMETERS:
C
```

```
C FOUT(I)    = POINTER TO ARC-LIST OF NODE I, I=1,2,...,N+1
C ND(J)   = ENDING NODE OF ARC J, J=1,2,...,M
C LNGT(J) = LENGTH OF ARC J, J=1,2,...,M
C NMAX    = DIMENSION OF ARRAYS A(.), D(.), P(.), Q(.)
C MMAX    = DIMENSION OF ARRAYS ND(.), LNGT(.)
C N       = NUMBER OF NODES
C INF     = VERY LARGE INTEGER VALUE (INFINITY)
C R       = ROOT
C
C MEANING OF THE OUTPUT PARAMETERS:
C
C D(I)    = SHORTEST DISTANCE FROM R TO I, I=1,2,...,N
C P(I)    = PREDECESSOR NODE OF I IN THE SHORTEST PATH TREE, I=1,2,...,N
C
C OF THE MAIN INTERNAL PARAMETERS:
C
C Q(I) = LIST OF CANDIDATE NODES; Q(I) = -1 IF I IS NOT IN Q AND IT HAS
C          ALREADY BEEN SCANNED
C        = 0 IF I IS NOT IN Q AND IT HAS
C          NOT BEEN SCANNED
C        = J IF I PRECEDES NODE J IN THE
C          LIST
C NN   = N+1
C U    = CURRENT NODE
C V    = ENDING NODE OF THE CURRENT ARC
C INIT = START-POINTER TO THE ARC-LIST OF THE CURRENT NODE
C IFIN = END-POINTER TO THE ARC-LIST OF THE CURRENT NODE
C DV   = TENTATIVE LABEL OF NODE V
C LAST = POINTER TO THE LAST NODE OF Q(.)
C
C ALL THE PARAMETERS ARE INTEGER
C
C*****************************************************************


      PARAMETER(MAXNODES=10000, MAXARCS=100000)

      INTEGER FOUT,NXTOUT,D,P,Q,R,U,V,DV,COUNT,FIRST,SECOND,THIRD
      DIMENSION FOUT(MAXNODES),D(MAXNODES),P(MAXNODES),Q(MAXNODES)
      DIMENSION ND(MAXARCS),LNGT(MAXARCS),NXTOUT(MAXARCS)
C
C INITIALIZE
C
      DO 10 I=1,N
        Q(I) = 0
        D(I) = INF
  10  CONTINUE
      Q(R) = - 1
      D(R)=0
      P(R) = 0
      NN = N + 1
      FIRST = NN
      LAST = NN
      COUNT=0
```

```
      U = R
C
C
C EXPLORE THE FORWARD STAR OF  U
C
20    CONTINUE


      COUNT=COUNT+1
      J=FOUT(U)
25    IF (J.GT.0) THEN
        V = ND(J)
        DV = D(U) + LNGT(J)
C
C CHECK WHETHER THE LABEL OF V CAN BE IMPROVED
C
        IF ( D(V) .GT. DV ) THEN
          D(V) = DV
          P(V) = U
          IF ( Q(V) ) 30,30,50
C
C IF V IS NOT IN Q AND ITS LABEL IS SMALLER THAN THE LABEL OF THE TOP
NODE,
C IT IS INSERTED  AT THE START OF Q
C
30       IF (LAST.EQ.NN) THEN
          Q(V) = NN
          FIRST = V
          LAST = V
          GO TO 50
         END IF

          IF (DV.LE.D(FIRST))THEN
            Q(V) = FIRST
            FIRST = V
          ELSE
            Q(LAST) = V
            Q(V) = NN
            LAST = V
          END IF

50        CONTINUE
        END IF
        J=NXTOUT(J)
        GOTO 25
       END IF
C
C REMOVE THE NEW CURRENT NODE U
C
60     U = FIRST
       FIRST = Q(U)
       Q(U) = - 1
       IF ( LAST .EQ. U ) LAST = NN
C
```

```
C CHECK WHETHER THE LIST IS EMPTY
C
       IF ( U .LE. N ) GO TO 20

       RETURN
       END

      SUBROUTINE READ(N,M,FOUT,NXTOUT,END,LNGT)
C*****************************************************************
C READS THE GRAPH DATA (STORED AS AN ADJACENCE LIST) AND THE ORIGINS
C LIST.
C*****************************************************************


      PARAMETER(MAXNODES=10000, MAXARCS=100000)

      IMPLICIT INTEGER (A-Z)
      DIMENSION FOUT(MAXNODES),LAST(MAXNODES)
      DIMENSION START(MAXARCS),END(MAXARCS),LNGT(MAXARCS)
      DIMENSION NXTOUT(MAXARCS)
C
      PRINT*,'READING THE SHORTEST PATH PROBLEM DATA'
      OPEN(13,FILE='FOR013.DAT',STATUS='OLD')
      REWIND(13)

C     READ NUMBER OF NODES AND ARCS

      READ(13,1010) N,M

C     READ ENDNODE AND LENGTH OF EACH ARC

      DO 20 I=1,M
        READ(13,1020) START(I),END(I),LNGT(I),DUMMY
20    CONTINUE

      DO 30 I=1,N
        READ(13,1000) DUMMY
30    CONTINUE

      ENDFILE(13)
      REWIND(13)

      PRINT*,'END OF READING'

1000  FORMAT(1I8)
1010  FORMAT(2I8)
1020  FORMAT(4I8)


      PRINT *, 'RESTRUCTURING THE DATA '
C
C     GENERATE FORWARD AND BACKWARD STARS FOR EACH NODE
C
      DO 60 I=1,N
```

```
        FOUT(I)=0
        LAST(I)=0
60    CONTINUE

      DO 65 ARC=1,M
        NXTOUT(ARC)=0
        NODE=START(ARC)
        IF (FOUT(NODE).NE.0) THEN
          NXTOUT(LAST(NODE))=ARC
        ELSE
          FOUT(NODE)=ARC
        END IF
        LAST(NODE)=ARC
65    CONTINUE

      RETURN

      END
```

SLFT

Combination of Small Label First (SLF) code with threshold algorithm for finding
shortest path from one origin to all destinations, as per the paper "A Simple and
Fast Label Correcting Method for Shortest Paths," Networks, Vol. 23, 1993, pp.
703-709, by D. P. Bertsekas.

```
C    ************************************************************
C    COMBINED SLF-THRESHOLD ALGORITHM TO FIND SHORTEST PATHS
C    FROM ONE ORIGIN TO ALL DESTINATIONS
C    ************************************************************
C
C ALL THE PARAMETERS ARE INTEGER
C
C THE ONLY MACHINE DEPENDENT CONSTANT USED IS INF
C
C*********************************************************************


      PARAMETER(MAXNODES=10000, MAXARCS=100000)

      INTEGER FOUT,NXTOUT,D,P,HP,Y,X,T2,DEST,COUNT,CC
      INTEGER A,Q1,Q2,T,R
      REAL TT1,TT2,TCOST
      DIMENSION FOUT(MAXNODES),D(MAXNODES),P(MAXNODES)
      DIMENSION Q1(MAXNODES),Q2(MAXNODES)
      DIMENSION LNGT(MAXARCS),ND(MAXARCS),NXTOUT(MAXARCS)
      DATA INF/999999999/
```

```
      CALL READ(N,M,FOUT,NXTOUT,ND,LNGT)

C  LMAX IS THE MAXIMUM ARC LENGTH USED IN SETTING THE THRESHOLD



      LMAX=-INF

      DO 5 ARC=1,M
        IF (LNGT(ARC).GT.LMAX) LMAX=LNGT(ARC)
14.   CONTINUE


      CALL THINCR(N,M,LMAX,T)

      DO 10 I=1,N
        P(I) = 0
        Q1(I) = 0
        Q2(I) = 0
10    CONTINUE
C
C    INITIALIZE ORIGIN AND QUEUE OF DESTINATIONS
C

      PRINT*,'THE NUMBER OF NODES IS = ',N
      PRINT*,'ENTER THE ORIGIN NODE'
      READ*,R



      PRINT *,'CALLING SLF-THRESHOLD ALGORITHM TO SOLVE THE PROBLEM'
      PRINT*,'***********************************************'
C
C     GET STARTING TIME FOR THE MAC II
C
      TT1 = LONG(362)/60.0
C

        CALL SLFT(FOUT,NXTOUT,ND,LNGT,D,P,Q1,Q2,N,INF,T,R,COUNT,CC)
C
C     GET ENDING TIME FOR THE MAC II
C
      TT2 = LONG(362)/60.0 - TT1
      PRINT *,'FINISHED --- TOTAL CPU TIME', TT2,' SECS'
      PRINT*,'***********************************************'
      PRINT*,'THE ORIGIN NODE IS ',R
C

      PRINT*,'NUMBER OF ITERATIONS ',COUNT
      PRINT*,'NUMBER OF REPARTITIONS ',CC
      PRINT*,'SH. DIST. OF DESTINATION ',N,' IS ',D(N)

45    PRINT*,'ENTER OTHER DESTINATION NODE (0 IF DONE)'
```

```
      READ*,DEST
      IF ((DEST.LT.0).OR.(DEST.GT.N)) GOTO 45
      IF (DEST.EQ.R) GOTO 45
      IF (DEST.NE.0) THEN
        PRINT*,'SHORTEST DISTANCE TO',DEST,' = ',D(DEST)
        GOTO 45
      END IF



      STOP
      END
C
C


      SUBROUTINE SLFT(FOUT,NXTOUT,ND,LNGT,D,P,Q1,Q2,N,INF,T,R,COUNT,CC)
C*****************************************************************
C
C    ROUTINE SLFT
C
C    COMBINED SLF-THRESHOLD ALGORITHM BASED ON PAPER:
C    "A SIMLPLE AND FAST LABEL CORRECTING ALGORITHM FOR SHORTEST PATHS"
C     NETWORKS, VOL. 23, 1993, PP. 703-709, BY
C     DIMITRI P. BERTSEKAS
C
C*****************************************************************
C 1) FINDS A SHORTEST PATH TREE ROOTED AT NODE R AND THE SHORTEST
C    DISTANCES
C 2) IS BASED ON THE SLF METHOD WITH THE SET Q
C    IMPLEMENTED AS A PAIR OF LISTS: Q1(.) AS A QUEUE AND
C    Q2(.) AS A LINKED-LIST. THE PARTITION IS BASED ON A
C    THRESHOLD VALUE
C
C MEANING OF THE INPUT PARAMETERS:
C
C FOUT(I)    = POINTER TO ARC-LIST OF NODE I, I=1,2,...,N+1
C ND(J)   = ENDING NODE OF ARC J, J=1,2,...,M
C LNGT(J) = LENGTH OF ARC J, J=1,2,...,M
C NMAX    = DIMENSION OF ARRAYS A(.), D(.), P(.), Q(.)
C MMAX    = DIMENSION OF ARRAYS ND(.), LNGT(.)
C N       = NUMBER OF NODES
C INF     = VERY LARGE INTEGER VALUE (INFINITY)
C R       = ROOT
C
C MEANING OF THE OUTPUT PARAMETERS:
C
C D(I)    = SHORTEST DISTANCE FROM R TO I, I=1,2,...,N
C P(I)    = PREDECESSOR NODE OF I IN THE SHORTEST PATH TREE, I=1,2,...,N
C
C MEANING OF THE MAIN INTERNAL PARAMETERS:
C
C Q1(I) = LIST OF CANDIDATE NODES       Q1(I) = 0 IF I IS NOT IN Q1(.)
C         HAVING THEIR LABEL LESS THAN        = J IF I PRECEDES NODE J
```

```
C         OR EQUAL TO THE CURRENT                 IN THE LIST
C         THRESHOLD
C
C Q2(I) = LIST OF THE OTHER CANDIDATE   Q2(I) = 0 IF I IS NOT IN Q2(.)
C         NODES AND OLD COPIES OF NODES       = J IF I PRECEDES NODE J
C         INSERTED IN Q1(.)                     IN THE LIST
C
C NN   = N+1
C U    = CURRENT NODE
C V    = ENDING NODE OF THE CURRENT ARC
C INIT = START-POINTER TO THE ARC-LIST OF THE CURRENT NODE
C IFIN = END-POINTER TO THE ARC-LIST OF THE CURRENT NODE
C DV   = TENTATIVE LABEL OF NODE V
C LAST = POINTER TO THE LAST NODE OF Q(.)
C PNTR = POINTER TO THE LAST NODE OF THE FIRST QUEUE OF Q(.)
C
C ALL THE PARAMETERS ARE INTEGER
C
C*********************************************************************


      PARAMETER(MAXNODES=10000, MAXARCS=100000)

      INTEGER Q1,Q2,T,THRS,T1,FIRST
      INTEGER FOUT,NXTOUT,D,P,R,U,V,DV,COUNT,CC,LAST,LAST2
      DIMENSION FOUT(MAXNODES),D(MAXNODES),P(MAXNODES)
      DIMENSION ND(MAXARCS),LNGT(MAXARCS),NXTOUT(MAXARCS)
      DIMENSION Q1(MAXNODES),Q2(MAXNODES)
C
C INITIALIZE
C
      DO 10 I=1,N
        D(I) = INF
10    CONTINUE
      D(R) = 0
      P(R) = 0
      NN = N + 1
      Q1(NN) = NN
      Q2(NN) = NN
      THRS = -1
      LAST = NN
      LAST2 = NN
      COUNT=0
      CC=0
      U = R
C
C EXPLORE THE FORWARD STAR OF U
C
20    CONTINUE

      COUNT=COUNT+1
      J=FOUT(U)
25    IF (J.GT.0) THEN
        V = ND(J)
```

```
        DV = D(U) + LNGT(J)
C
C CHECK WHETHER THE LABEL OF V CAN BE IMPROVED
C
        IF ( D(V) .LE. DV ) GO TO 50
        IF ( DV .GT. THRS ) GO TO 30
        IF ( Q1(V) .GT. 0 ) GO TO 40
C
C INSERT V AT THE HEAD OR TAIL OF Q1(.)
C
        IF (LAST.EQ.NN) THEN
          Q1(V) = NN
          FIRST = V
          LAST = V
          GO TO 40
         END IF

         IF (DV.LE.D(FIRST)) THEN
          Q1(V) = FIRST
          FIRST = V
        ELSE
          Q1(LAST) = V
          Q1(V) = NN
          LAST = V
        END IF

        GO TO 40

30      IF ( Q2(V) .GT. 0 ) GO TO 40
C
C IF V IS NOT IN Q2(.), IT IS INSERTED AT THE HEAD OR THE TAIL OF Q2(.)
C
        IF (LAST2.EQ.NN) THEN
          Q2(V) = NN
          Q2(NN) = V
          LAST2 = V
          GO TO 40
         END IF

         IF (DV.LE.D(Q2(NN))) THEN
          Q2(V) = Q2(NN)
          Q2(NN) = V
        ELSE
          Q2(LAST2) = V
          Q2(V) = NN
          LAST2 = V
        END IF

C
C  UPDATE THE LABEL OF V
C

40      D(V) = DV
        P(V) = U
```

```
50      CONTINUE

        J=NXTOUT(J)
        GOTO 25
       END IF
C
C CHECK WHETHER Q1(.) IS EMPTY
C
60    IF (LAST .EQ. NN) GO TO 80
C
C REMOVE THE NEW CURRENT NODE U FROM THE HEAD OF Q1(.)
C
70    U = FIRST
      FIRST = Q1(U)
      Q1(U) = 0
      IF ( LAST .EQ. U ) LAST = NN
      GO TO 20
C
C CHECK WHETHER ALSO Q2(.) IS EMPTY
C
80    IF (LAST2 .EQ. NN ) RETURN


C
C COMPUTE THE NEW TENTATIVE THRESHOLD VALUE T1
C
      MIN = INF
      T1 = THRS + 1 + T
      LAST2=NN
      I = NN
      J = Q2(I)
C
C SCAN Q2(.) IN ORDER TO COMPUTE MIN AND TO REMOVE COPIES OF NODES
C ALREADY REMOVED
C


90    IF ( D(J) .LE. T1 ) GO TO 100
C
C UPDATE MIN
C
      LAST2=J
      MIN = MIN0(MIN, D(J))
      I = J
      GO TO 110
C
C  REMOVE J FROM Q2(.)
C
100   Q2(I) = Q2(J)
      Q2(J) = 0
C
C CHECK WHETHER J MUST BE INSERTED IN Q1(.)
C
```

```
      IF ( D(J) .LE. THRS ) GO TO 110

      IF (LAST.EQ.NN) THEN
        Q1(J) = NN
        FIRST = J
        LAST = J
        GO TO 110
       END IF

      IF (D(J).LE.D(FIRST)) THEN
        Q1(J) = FIRST
        FIRST = J
      ELSE
        Q1(LAST) = J
        Q1(J) = NN
        LAST = J
      END IF

110   J = Q2(I)
      IF ( J .NE. NN ) GO TO 90
C
C UPDATE THE THRESHOLD VALUE
C
      THRS = T1
C
C CHECK WHETHER Q1(.) IS STILL EMPTY
C
      IF ( LAST .NE. NN ) THEN
        CC=CC+1
        GO TO 70
      END IF
C
C IF Q2(.) WAS EMPTIED THEN RETURN
C
      IF ( LAST2 .EQ. NN ) RETURN
C
C INCREASE THE THRESHOLD VALUE THRS AND SCAN AGAIN Q2(.)
C
      THRS = MIN + T
      LAST2=NN
      I = NN
      J = Q2(I)

120   IF ( D(J) .GT. THRS ) GO TO 130

C
C  MOVE J FROM Q2(.) TO Q1(.)
C

      Q2(I) = Q2(J)
      Q2(J) = 0

      IF (LAST.EQ.NN) THEN
        Q1(J) = NN
```

```
        FIRST = J
        LAST = J
        GO TO 140
      END IF

      IF (D(J).LE.D(FIRST)) THEN
        Q1(J) = FIRST
        FIRST = J
      ELSE
        Q1(LAST) = J
        Q1(J) = NN
        LAST = J
      END IF

      GO TO 140

C
C  CONTINUE THE SCANNING OF Q2(.)
C

130   I = J
      LAST2=J

140   J = Q2(I)
      IF ( J .NE. NN ) GO TO 120
      CC=CC+1
      GO TO 70
      END




      SUBROUTINE THINCR(N,M,LMAX,T)
C*********************************************************************
C  COMPUTE THE INCREMENT OF THE THRESHOLD AS PROPOSED IN "F.GLOVER; R.
C  GLOVER; D.KLINGMAN, CENTER FOR CYBERNETIC STUDIES, UNIVERSITY OF
C  TEXAS, AUSTIN TX, USA. RESEARCH REPORT CCS 419, JUNE 1981.
C*********************************************************************
      INTEGER T,S

C  CHOOSE X2=1.5 FOR GRID GRAPHS AND X2=0.25 FOR RANDOM GRAPHS

      X2=0.25

      S = MIN0(35,M/N)
      ALMAX = LMAX
      T = X2 * ALMAX
      AT = T
      AS = S
      IF ( S .GT. 7 ) T = AT * 7. / AS
      IF ( T .LE. 0 ) T = 1
      RETURN
100   FORMAT(F4.2)
```

```
      END


      SUBROUTINE READ(N,M,FOUT,NXTOUT,END,LNGT)
C*****************************************************************
C READS THE GRAPH DATA (STORED AS AN ADJACENCE LIST) AND THE ORIGINS
C LIST.
C*****************************************************************


      PARAMETER(MAXNODES=10000, MAXARCS=100000)

      IMPLICIT INTEGER (A-Z)
      DIMENSION FOUT(MAXNODES),LAST(MAXNODES)
      DIMENSION START(MAXARCS),END(MAXARCS),LNGT(MAXARCS)
      DIMENSION NXTOUT(MAXARCS)
C
      PRINT*,'READING THE SHORTEST PATH PROBLEM DATA'
      OPEN(13,FILE='FOR013.DAT',STATUS='OLD')
      REWIND(13)

C     READ NUMBER OF NODES AND ARCS

      READ(13,1010) N,M

C     READ ENDNODE AND LENGTH OF EACH ARC

      DO 20 I=1,M
        READ(13,1020) START(I),END(I),LNGT(I),DUMMY
20    CONTINUE

      DO 30 I=1,N
        READ(13,1000) DUMMY
30    CONTINUE

      ENDFILE(13)
      REWIND(13)

      PRINT*,'END OF READING'

1000  FORMAT(1I8)
1010  FORMAT(2I8)
1020  FORMAT(4I8)


      PRINT *, 'RESTRUCTURING THE DATA '
C
C     GENERATE FORWARD AND BACKWARD STARS FOR EACH NODE
C
      DO 60 I=1,N
        FOUT(I)=0
        LAST(I)=0
60    CONTINUE
```

```
      DO 65 ARC=1,M
        NXTOUT(ARC)=0
        NODE=START(ARC)
        IF (FOUT(NODE).NE.0) THEN
          NXTOUT(LAST(NODE))=ARC
        ELSE
          FOUT(NODE)=ARC
        END IF
        LAST(NODE)=ARC
65    CONTINUE

      RETURN

      END
```

```
*****************************************************************************
*

Auction Code for Max-Flow Problems

*****************************************************************************
*

Auction algorithm for the max-flow problem, as per the paper
"An Auction Algorithm for the Max-Flow Problem", JOTA , Vol. 87, 1995, pp.
69-101, by D. P.
Bertsekas.

AUCTION_MAX_FLOW


C     ******* AUCTION/MAX-FLOW CODE*******
C
C     CONTAINS SOME TIMING STATEMENTS THAT NEED TO BE ADAPTED TO THE
TARGET SYSTEM
C
C     ************************************************
      PARAMETER (MAXNODES=20001, MAXARCS=120000)
      IMPLICIT INTEGER (A-Z)
      COMMON /SCALARS/  N,NA,LARGE,SOURCE,SINK
      COMMON /STATS/    NITER
      COMMON /STATS1/   NAUG,EXT_ATT,EXT_SUCC,SB_SUCC,NUM_EXT
      COMMON /CHECK/    CH_TIME,EXT_TIME,SB_TIME,AUG_TIME,MF_TIME
      COMMON /CHECK1/   IT_TIME,INIT_TIME,CEL_TIME,PL_TIME,TOT_TIME
      COMMON /BLK1/     STARTN
      COMMON /BLK2/     ENDN
      COMMON /UBOUND/   U
      COMMON /FLOW/     F
      COMMON /PRICES/   P
      COMMON /BLK3/     SURPLUS
```

```
      COMMON /BLK4/     FIN
      COMMON /BLK5/     FOUT
      COMMON /BLK6/     NXTIN
      COMMON /BLK7/     NXTOU
      COMMON /PRED/     PRDARC



      INTEGER STARTN(MAXARCS)
      INTEGER ENDN(MAXARCS)
      INTEGER U(MAXARCS)
      INTEGER F(MAXARCS)
      INTEGER SURPLUS(MAXNODES)
      INTEGER FIN(MAXNODES)
      INTEGER FOUT(MAXNODES)
      INTEGER NXTIN(MAXARCS)
      INTEGER NXTOU(MAXARCS)
      INTEGER P(MAXNODES)
      INTEGER PRDARC(MAXNODES)
      REAL*8 TT,TIMER,TOTAL,TOTALT,AVERAGE_P
      REAL*8 CH_TIME,EXT_TIME,SB_TIME,AUG_TIME,MF_TIME,IT_TIME
      REAL*8 INIT_TIME,CEL_TIME,PL_TIME,TOT_TIME


      PRINT*,'AUCTION ALGORITHM FOR MAX-FLOW'
      PRINT*,'**************************************'

      TOTAL=0
      TOTALT=0
      TOT_ITER=0
      TOT_AUG=0

      LARGE=1000000000


C     ************************************************************
C
C     START OF PROBLEM INPUT CODE
C
C
C     THE PROBLEM IS SPECIFIED BY THE FOLLOWING VARIABLES:
C
C     N:  THE NUMBER OF NODES
C     NA: THE NUMBER OF ARCS
C     SOURCE: THE SOURCE NODE OF THE MAX-FLOW PROBLEM
C     SINK: THE SINK NODE OF THE MAX-FLOW PROBLEM
C     STARTN(ARC): THE START NODE OF ARC
C     ENDN(ARC): THE END NODE OF ARC
C     U(ARC): THE CAPACITY OF ARC (IT IS LATER CHANGED TO RESIDUAL
CAPACITY)
C
      PRINT*,'READING PROBLEM DATA'

      OPEN(13,FILE='FOR013.DAT',STATUS='OLD')
```

```
      REWIND(13)

C     READ NUMBER OF NODES AND ARCS

      READ(13,*) N,NA

      DO 5 I=1,NA
        READ(13,*) STARTN(I),ENDN(I),DUMMY,U(I)
15.   CONTINUE


      ENDFILE(13)
      REWIND(13)


      SOURCE=1
      SINK=N

C




C
C    END OF PROBLEM INPUT CODE
C    ************************************************************
C

C   INITIALIZE FLOWS

      DO 30 ARC=1,NA
        F(ARC)=0
X     IF (U(ARC).EQ.0) THEN
X       PRINT*,'ARC',ARC,' HAS ZERO CAPACITY'
X       PAUSE
X       STOP
X     END IF
30    CONTINUE

C   INITIALIZE PRICES AND SURPLUSES

        DO 50 NODE=1,N
             P(NODE)=N
             SURPLUS(NODE)=0
50      CONTINUE

      P(SINK)=0


      CALL INIDAT

      TIMER = LONG(362)


C ********** CALL MAIN ALGORITHM ********
```

```
      CALL AUCT_MF

C ***********************************************


      TT = FLOAT(LONG(362) - TIMER)/60

      PRINT*,'TIME TO FIND A MIN CUT = ',TOT_TIME
      PRINT *,'*********************************************'
      PRINT*,'TOTAL TIME = ',TT, ' secs'
X      PRINT*,'TIME FOR INITIALIZATION = ',INIT_TIME
X      PRINT*,'TIME TO BUILD PUSH LISTS = ',PL_TIME
X      PRINT*,'TIME FOR EXTENSIONS = ',EXT_TIME
X      PRINT*,'TIME FOR SECOND BEST LOGIC = ',SB_TIME
X      PRINT*,'TIME FOR NODE SCANS = ',IT_TIME
X      PRINT*,'TIME FOR CONTR/EXT LOGIC = ',CEL_TIME
X      PRINT*,'TIME FOR AUGMENTATIONS  = ',AUG_TIME
X      PRINT*,'TIME TO CHECK FOR A MIN CUT = ',CH_TIME

C     ***** COMPUTE MAX-FLOW *****

      MAX_FLOW=0
      ARC=FIN(SINK)
70    IF (ARC.GT.0) THEN
        MAX_FLOW=MAX_FLOW+F(ARC)
        ARC=NXTIN(ARC)
      GOTO 70
      END IF
X      IF (MAX_FLOW.NE.SURPLUS(SINK)) THEN
X        PRINT*,'ERROR IN THE CALCULATION OF THE MAX-FLOW'
X      END IF

      PRINT *, 'MAX-FLOW = ',MAX_FLOW
      PRINT *,'# OF FLOW PUSHES/ARC = ',FLOAT(NAUG)/FLOAT(NA)
      PRINT *,'# OF PRICE RISES/NODE = ',FLOAT(NITER)/FLOAT(N)
X      PRINT *,'# OF EXTENSIONS/NODE = ',FLOAT(NUM_EXT)/FLOAT(N)
X      PRINT *,'# OF SPEC. EXTENSION ATTEMPTS = ',EXT_ATT
X      PRINT *,'# OF SPEC. EXTENSION SUCCESSES = ',EXT_SUCC
X      PRINT *,'# OF SECOND BEST SUCCESSES = ',SB_SUCC
      PRINT *,'*********************************************'


C     ***** CHECK THAT ALL SURPLUSES ARE ZERO ****

        DO 80 NODE=1,N
             IF ((NODE.NE.SOURCE).AND.(NODE.NE.SINK)) THEN
               IF (SURPLUS(NODE).NE.0) THEN
                 PRINT*,'SURPLUS OF NODE ',NODE,' IS NONZERO'
               END IF
             END IF
80      CONTINUE

C     WRITE TIME
```

```
      OPEN(16,FILE='AUCTION.OUT',STATUS='NEW')
      REWIND(16)

      WRITE(16,1020) TOT_TIME
1020  FORMAT('TIME FOR MIN CUT = ',F11.4,' secs')
      WRITE(16,1021) TT
1021  FORMAT('TOTAL TIME = ',F11.4,' secs')
X      WRITE(16,1022) CH_TIME
X1022  FORMAT('TIME TO CHECK FOR A MIN CUT = ',F9.3)
X      WRITE(16,1023) EXT_TIME
X1023  FORMAT('TIME FOR EXTENSIONS = ',F9.3)
X      WRITE(16,1024) IT_TIME
X1024  FORMAT('TIME FOR NODE SCANS = ',F9.3)
X      WRITE(16,1025) SB_TIME
X1025  FORMAT('TIME FOR SECOND BEST LOGIC = ',F9.3)
X      WRITE(16,1026) CEL_TIME
X1026  FORMAT('TIME FOR CONTR/EXT LOGIC = ',F9.3)
X      WRITE(16,1027) AUG_TIME
X1027  FORMAT('TIME FOR AUGMENTATIONS = ',F9.3)


      ENDFILE(16)
      REWIND(16)

      PRINT *, 'PROGRAM ENDED; PRESS <CR> TO EXIT'
      PAUSE
      END




C  ***************************************************************
C
C     MAIN AUCTION ALGORITHM FOR MAX-FLOW.
C     THIS VERSION USES A BREADTH FIRST INITIALIZATION
C     AND A ROUTINE THAT DETECTS A SATURATED CUT.
C
C                THIS CODE WAS WRITTEN BY
C                  DIMITRI P. BERTSEKAS
C                      FEBRUARY 1993
C
C  ***************************************************************

      SUBROUTINE AUCT_MF
      PARAMETER (MAXNODES=20001, MAXARCS=120000)
      IMPLICIT NONE
      INTEGER N,NA,L,LARGE,FACTOR,EPS,ARC
      INTEGER NAUG,NITER,EXT_ATT,EXT_SUCC,SB_SUCC,NODE
      INTEGER I,J,K,LN,CAPOUT,CAPIN
      INTEGER START,END
      INTEGER FLOW,INCR,PRICE,LIMIT,F_LIMIT,ARC_COUNT,NUM_EXT
      INTEGER BSTLEVEL,NEW_LEVEL,LAST,EXCESS,NEWINCR
```

```
      INTEGER ROOT,TERM,PTERM,EXTARC,PREVARC,PRD
      INTEGER SECLEVEL,SECARC,N_PURG,CUR_LEVEL,CUR_NODE
      INTEGER PREVNODE,NXTNODE,PR_TERM,PREVLEVEL,TWO_NA
      INTEGER ISUCC,IPRED,GAPDIST,INEXT,IFIRST,IDIST,LEVEL,PLEVEL
      INTEGER SOURCE,SINK
      INTEGER NLIST,NSCAN
      REAL*8  TIMER,TIMER1,CH_TIME,EXT_TIME,SB_TIME,AUG_TIME,MF_TIME
      REAL*8  IT_TIME,INIT_TIME,CEL_TIME,PL_TIME,TOT_TIME


      INTEGER STARTN(MAXARCS),ENDN(MAXARCS)
      INTEGER U(MAXARCS),F(MAXARCS)
      INTEGER FIN(MAXNODES),FOUT(MAXNODES)
      INTEGER NXTIN(MAXARCS),NXTOU(MAXARCS),P(MAXNODES)
      INTEGER FPUSHF(MAXNODES),NXTPUSHF(MAXARCS)
      INTEGER FPUSHB(MAXNODES),NXTPUSHB(MAXARCS)
      INTEGER PRDARC(MAXNODES),EXTEND_ARC(MAXNODES)
      INTEGER SB_ARC(MAXNODES)
      INTEGER SURPLUS(MAXNODES)
      INTEGER LIST(MAXNODES),SB_LEVEL(MAXNODES)
      INTEGER FIRST(0:MAXNODES),SUCC(0:MAXNODES),PRED(0:MAXNODES)
      INTEGER PFIRST(0:MAXNODES),PSUCC(0:MAXNODES),PPRED(0:MAXNODES)


      COMMON /SCALARS/  N,NA,LARGE,SOURCE,SINK
      COMMON /STATS/    NITER
      COMMON /STATS1/   NAUG,EXT_ATT,EXT_SUCC,SB_SUCC,NUM_EXT
      COMMON /CHECK/    CH_TIME,EXT_TIME,SB_TIME,AUG_TIME,MF_TIME
      COMMON /CHECK1/   IT_TIME,INIT_TIME,CEL_TIME,PL_TIME,TOT_TIME
      COMMON /BLK1/     STARTN
      COMMON /BLK2/     ENDN
      COMMON /UBOUND/   U
      COMMON /FLOW/     F
      COMMON /PRICES/   P
      COMMON /BLK3/     SURPLUS
      COMMON /BLK4/     FIN
      COMMON /BLK5/     FOUT
      COMMON /BLK6/     NXTIN
      COMMON /BLK7/     NXTOU
      COMMON /PRED/     PRDARC

C   ******* DATA STRUCTURE THAT DETECTS A GAP IN PRICES *********

C     FIRST(D): FIRST NODE AT LABEL D
C     SUCC(NODE): SUCCESSOR OF NODE AT THE SAME DISTANCE (=0 IF NODE IS
LAST)
C     PRED(NODE): PREDECESSOR NODE AT THE SAME DISTANCE (=0 IF NODE IS
FIRST)

C   ************************************************************

C     PLEVEL: THE LABEL OF THE HIGHEST LABEL NODE W/ POS. SURPLUS

C   ************************************************************
```

```
      TIMER1 = LONG(362)

      TOT_TIME=0

C     INITIALIZE COUNT OF NUMBER OF UP ITERATIONS PERFORMED


X      TIMER = LONG(362)


      NITER = 0
      NAUG=0
X      NUM_EXT=0

X      CH_TIME=0
X      EXT_TIME=0
X      SB_TIME=0
X      AUG_TIME=0
X      MF_TIME=0
X      IT_TIME=0
X      INIT_TIME=0
X      CEL_TIME=0
X      PL_TIME=0


X      EXT_ATT=0
X      EXT_SUCC=0
X      SB_SUCC=0

C   SET PATH LENGTH LIMIT PARAMETERS

      F_LIMIT=1

C   INITIALIZE FLOWS

X      DO 30 ARC=1,NA
X        F(ARC)=0
X     IF (U(ARC).EQ.0) THEN
X       PRINT*,'ARC',ARC,' HAS ZERO CAPACITY'
X       PAUSE
X       STOP
X     END IF
X30    CONTINUE

C   INITIALIZE DATA STRUCTURES

        DO 50 NODE=1,N
C       P(NODE)=N
C       SURPLUS(NODE)=0
              FIRST(NODE)=0
              PFIRST(NODE)=0
              PPRED(NODE)=0
```

```
              PSUCC(NODE)=0
              SB_LEVEL(NODE)=-LARGE
50       CONTINUE

C   INITIALIZE PRICES BY A BREADTH FIRST SEARCH METHOD

       P(SOURCE)=N
       P(SINK)=0
       LEVEL=0

         NLIST=1
       LIST(1)=SINK
       NSCAN=0

60       CONTINUE
         NSCAN=NSCAN+1
       NODE=LIST(NSCAN)
       PRICE=P(NODE)+1
       ARC=FIN(NODE)
65     IF (ARC.GT.0) THEN
              START=STARTN(ARC)
              IF (P(START).EQ.N) THEN
                NLIST=NLIST+1
                LIST(NLIST)=START
                EXTEND_ARC(START)=0
                P(START)=PRICE
                IF (LEVEL.LT.PRICE) LEVEL=PRICE
                IFIRST=FIRST(PRICE)
                FIRST(PRICE)=START
                SUCC(START)=IFIRST
                PRED(START)=0
                PRED(IFIRST)=START
              END IF
              ARC=NXTIN(ARC)
              GOTO 65
       END IF


         IF (NLIST.GT.NSCAN) GOTO 60


       P(SOURCE)=N
X      N_PURG=1


C  INITIALIZE PLEVEL, FLOWS, AND SURPLUSES

         PLEVEL=0

       ARC=FOUT(SOURCE)
70     IF (ARC.GT.0) THEN
              END=ENDN(ARC)
              PRICE=P(END)
              IF (PLEVEL.LT.PRICE) PLEVEL=PRICE
```

```
              IF (SURPLUS(END).EQ.0) THEN
                IFIRST=PFIRST(PRICE)
                PFIRST(PRICE)=END
                PSUCC(END)=IFIRST
                PPRED(IFIRST)=END
              END IF
              INCR=U(ARC)
              F(ARC)=INCR
              U(ARC)=0
              SURPLUS(END)=SURPLUS(END)+INCR
              ARC=NXTOU(ARC)
              GOTO 70
      END IF


X       INIT_TIME= INIT_TIME+FLOAT(LONG(362) - TIMER)/60


C  ***  START AUGMENTATIONS  ***


200   CONTINUE

      ROOT=PFIRST(PLEVEL)

X         IF (SURPLUS(ROOT).EQ.0)  THEN
X           PRINT*,'SURPLUS OF ROOT IS ZERO'
X           PAUSE
X           STOP
X         END IF

       TERM=ROOT

       ARC_COUNT=0

C  *********************************************************
C
C   *** MAIN ALGORITHM WITH ROOT AS ORIGIN ***
C
C  *********************************************************

500    CONTINUE


X        TIMER = LONG(362)

C     START OF A NEW FORWARD ITERATION

      PTERM=P(TERM)
      EXTARC=EXTEND_ARC(TERM)

X      IF (PTERM.GE.N) THEN
X        PRINT*,'*** CAUTION: NODE W/ LARGE PRICE IS TERMINAL'
X      PAUSE
```

```
X      STOP
X      END IF

       IF (EXTARC.EQ.0) THEN

X         TIMER = LONG(362)

C   BUILD THE LIST OF ARCS W/ ROOM FOR PUSHING FLOW

       FPUSHF(TERM)=0
       ARC=FOUT(TERM)
510      IF (ARC.GT.0) THEN
               IF (U(ARC).GT.0) THEN
                 IF (FPUSHF(TERM).EQ.0) THEN
                   FPUSHF(TERM)=ARC
                   NXTPUSHF(ARC)=0
                   LAST=ARC
                 ELSE
                   NXTPUSHF(LAST)=ARC
                   NXTPUSHF(ARC)=0
                   LAST=ARC
                 END IF
               END IF
               ARC=NXTOU(ARC)
               GO TO 510
       END IF

       FPUSHB(TERM)=0
         ARC=FIN(TERM)
520      IF (ARC.GT.0) THEN
               IF (F(ARC).GT.0) THEN
                 IF (FPUSHB(TERM).EQ.0) THEN
                   FPUSHB(TERM)=ARC
                   NXTPUSHB(ARC)=0
                   LAST=ARC
                 ELSE
                   NXTPUSHB(LAST)=ARC
                   NXTPUSHB(ARC)=0
                   LAST=ARC
                 END IF
               END IF
               ARC=NXTIN(ARC)
               GO TO 520
       END IF

X      PL_TIME= PL_TIME+FLOAT(LONG(362) - TIMER)/60
       GO TO 600
       END IF


C      SPECULATIVE PATH EXTENSION ATTEMPT
C      NOTE: ARC>0 MEANS THAT ARC IS ORIENTED FROM THE SOURCE TO THE SINK
C      ARC<0 MEANS THAT ARC IS ORIENTED FROM THE SINK TO THE SOURCE
```

```
X      EXT_ATT=EXT_ATT+1

       IF (EXTARC.GT.0) THEN
         IF (U(EXTARC).EQ.0) THEN
               SECLEVEL=SB_LEVEL(TERM)
X         EXT_TIME= EXT_TIME+FLOAT(LONG(362) - TIMER)/60
               GO TO 580
       END IF
       END=ENDN(EXTARC)
         BSTLEVEL=P(END)
         IF (PTERM.GE.BSTLEVEL) THEN
X        EXT_SUCC=EXT_SUCC+1
X          NUM_EXT=NUM_EXT+1
           TERM=END
           PRDARC(TERM)=EXTARC

C    IF PATH IS LONG, DO AN AUGMENTATION

           IF (ARC_COUNT.GE.F_LIMIT) THEN
X          EXT_TIME= EXT_TIME+FLOAT(LONG(362) - TIMER)/60
                GOTO 2000
              END IF

C    IF SINK IS FOUND, DO AN AUGMENTATION

              IF (TERM.EQ.SINK) THEN
X          EXT_TIME= EXT_TIME+FLOAT(LONG(362) - TIMER)/60
                GOTO 2000
              END IF

C     RETURN FOR ANOTHER ITERATION

              ARC_COUNT=ARC_COUNT+1
X        EXT_TIME= EXT_TIME+FLOAT(LONG(362) - TIMER)/60
           GO TO 500
         END IF
       ELSE
         EXTARC=-EXTARC
         IF (F(EXTARC).EQ.0) THEN
               SECLEVEL=SB_LEVEL(TERM)
X          EXT_TIME= EXT_TIME+FLOAT(LONG(362) - TIMER)/60
               GO TO 580
       END IF
       START=STARTN(EXTARC)
         BSTLEVEL=P(START)
         IF (PTERM.GE.BSTLEVEL) THEN
X        EXT_SUCC=EXT_SUCC+1
X           NUM_EXT=NUM_EXT+1
           TERM=START
           PRDARC(TERM)=-EXTARC

C    IF PATH IS LONG, DO AN AUGMENTATION

           IF (ARC_COUNT.GE.F_LIMIT) THEN
```

```
X         EXT_TIME= EXT_TIME+FLOAT(LONG(362) - TIMER)/60
               GOTO 2000
              END IF

C     IF SINK IS FOUND, DO AN AUGMENTATION

              IF (TERM.EQ.SINK) THEN
X         EXT_TIME= EXT_TIME+FLOAT(LONG(362) - TIMER)/60
               GOTO 2000
              END IF

C      RETURN FOR ANOTHER ITERATION

              ARC_COUNT=ARC_COUNT+1
X       EXT_TIME= EXT_TIME+FLOAT(LONG(362) - TIMER)/60
          GO TO 500
         END IF
       END IF

C     SECOND BEST LOGIC TEST APPLIED TO SAVE A FULL NODE SCAN
C    IF OLD BEST LEVEL CONTINUES TO BE BEST GO FOR ANOTHER CONTRACTION

550       CONTINUE

X         TIMER = LONG(362)

          SECLEVEL=SB_LEVEL(TERM)
          IF (BSTLEVEL.LE.SECLEVEL) THEN
X          SB_SUCC=SB_SUCC+1
X          SB_TIME= SB_TIME+FLOAT(LONG(362) - TIMER)/60
               GOTO 800
          END IF

C     IF SECOND BEST CAN BE USED, DO EITHER A CONTRACTION
C     OR START OVER WITH A SPECULATIVE EXTENSION

580     IF (SECLEVEL.GT.-LARGE) THEN
              EXTARC=SB_ARC(TERM)
           IF (EXTARC.GT.0) THEN
                 IF (U(EXTARC).EQ.0) THEN
X             SB_TIME= SB_TIME+FLOAT(LONG(362) - TIMER)/60
                GOTO 600
                 END IF
                 BSTLEVEL=P(ENDN(EXTARC))
              ELSE
                 IF (F(-EXTARC).EQ.0) THEN
X             SB_TIME= SB_TIME+FLOAT(LONG(362) - TIMER)/60
                GOTO 600
                 END IF
              BSTLEVEL=P(STARTN(-EXTARC))
              END IF
              IF (BSTLEVEL.EQ.SECLEVEL) THEN
X          SB_SUCC=SB_SUCC+1
               SB_LEVEL(TERM)=-LARGE
```

```
               EXTEND_ARC(TERM)=EXTARC
X         SB_TIME= SB_TIME+FLOAT(LONG(362) - TIMER)/60
               GOTO 800
             END IF
       END IF

C
C   EXTENSION ATTEMPT WAS UNSUCCESSFUL, SO SCAN TERMINAL NODE
C

600   CONTINUE

X       TIMER = LONG(362)


       BSTLEVEL=LARGE
       SECLEVEL=LARGE

       ARC=FPUSHF(TERM)
700    IF (ARC.GT.0) THEN
       NEW_LEVEL = P(ENDN(ARC))
       IF (NEW_LEVEL.LT.SECLEVEL) THEN
               IF (NEW_LEVEL.LT.BSTLEVEL) THEN
                 SECLEVEL=BSTLEVEL
                 BSTLEVEL=NEW_LEVEL
                 SECARC=EXTARC
                 EXTARC=ARC
               ELSE
                 SECLEVEL=NEW_LEVEL
                 SECARC=ARC
               END IF
       END IF
       ARC=NXTPUSHF(ARC)
       GOTO 700
       END IF

       ARC=FPUSHB(TERM)
710    IF (ARC.GT.0) THEN
              NEW_LEVEL = P(STARTN(ARC))
              IF (NEW_LEVEL.LT.SECLEVEL) THEN
                IF (NEW_LEVEL.LT.BSTLEVEL) THEN
                  SECLEVEL=BSTLEVEL
                  BSTLEVEL=NEW_LEVEL
                  SECARC=EXTARC
                  EXTARC=-ARC
                  ELSE
                  SECLEVEL=NEW_LEVEL
                  SECARC=-ARC
                END IF
              END IF
              ARC=NXTPUSHB(ARC)
              GOTO 710
       END IF
```

```
        SB_LEVEL(TERM)=SECLEVEL
        SB_ARC(TERM)=SECARC
        EXTEND_ARC(TERM)=EXTARC

X        IT_TIME= IT_TIME+FLOAT(LONG(362) - TIMER)/60


C    *************  END OF NODE SCAN  *************

C     IF THE TERMINAL NODE IS THE ROOT, ADJUST ITS PRICE AND CHANGE ROOT

800      CONTINUE


            NEW_LEVEL=BSTLEVEL+1

            P(TERM)=NEW_LEVEL

C     CHECK FOR PRICE CHANGE AND A GAP

              IF (PTERM.LT.NEW_LEVEL) THEN

             NITER=NITER+1

X            TIMER = LONG(362)

C     IF TERM HAS POSITIVE SURPPLUS ADJUST THE POS. SURPLUS BUCKETS

             IF (SURPLUS(TERM).GT.0) THEN

                  ISUCC=PSUCC(TERM)
                  IPRED=PPRED(TERM)
                  PSUCC(IPRED)=ISUCC
                  PPRED(ISUCC)=IPRED
               IF (IPRED.EQ.0) PFIRST(PTERM)=ISUCC

                  IF (NEW_LEVEL.LT.N) THEN

C     ADJUST LEVEL AND PLACE TERM AT THE TOP OF THE NEW PRICE BUCKET

                  IF (PLEVEL.LT.NEW_LEVEL) PLEVEL=NEW_LEVEL
                  NODE=PFIRST(NEW_LEVEL)
                  PFIRST(NEW_LEVEL)=TERM
                  PPRED(NODE)=TERM
                  PSUCC(TERM)=NODE
                  PPRED(TERM)=0
                  ELSE
                          IF (PLEVEL.EQ.PTERM) THEN
                   IF ((IPRED.EQ.0).AND.(ISUCC.EQ.0)) THEN

C     TERM IS PURGED AND IS THE ONLY NODE AT THE MAX POS. SURPLUS PRICE
LEVEL
C     FIND THE NEW PLEVEL
```

```
850             PLEVEL=PLEVEL-1
                           IF (PLEVEL.EQ.0) THEN
X                        CH_TIME= CH_TIME+FLOAT(LONG(362) - TIMER)/60
                               TOT_TIME= TOT_TIME+FLOAT(LONG(362) -
TIMER1)/60
                               GOTO 3000
                             END IF
                             IF (PFIRST(PLEVEL).EQ.0) GOTO 850
                             END IF
                          END IF
                  END IF
              END IF

C     ********************************************

C      REMOVE TERM FROM THE CURRENT PRICE BUCKET

              ISUCC=SUCC(TERM)
              IPRED=PRED(TERM)
                 SUCC(IPRED)=ISUCC
                 PRED(ISUCC)=IPRED

                 IF (NEW_LEVEL.LT.N) THEN

C      ADJUST LEVEL AND PLACE TERM AT THE TOP OF THE NEW PRICE BUCKET

                   IF (LEVEL.LT.NEW_LEVEL) LEVEL=NEW_LEVEL
                   NODE=FIRST(NEW_LEVEL)
                   FIRST(NEW_LEVEL)=TERM
                   PRED(NODE)=TERM
                   SUCC(TERM)=NODE
                   PRED(TERM)=0
                 ELSE
X             N_PURG=N_PURG+1
X               PRINT*,'NUMBER PURGED =',N_PURG
                  END IF

              IF (IPRED.EQ.0) THEN
                FIRST(PTERM)=ISUCC
                   IF (ISUCC.EQ.0) THEN

C  ************  GAP HAS BEEN OBTAINED; ADJUST DATA STRUCTURES

                  IF (F_LIMIT.LT.5) F_LIMIT=F_LIMIT+1

C     PURGE THE NODES ABOVE THE GAP BY SETTING THEIR PRICE TO N
C     AND CLEAN UP THE BUCKETS ABOVE THE GAP

                  IF (LEVEL.GT.PTERM) THEN
                    DO 920 CUR_LEVEL=LEVEL,PTERM+1,-1
                             CUR_NODE=FIRST(CUR_LEVEL)
900               IF (CUR_NODE.GT.0) THEN
X                     N_PURG=N_PURG+1
                               P(CUR_NODE)=N
```

```
                                    CUR_NODE=SUCC(CUR_NODE)
                                  GOTO 900
                                END IF
                                FIRST(CUR_LEVEL)=0
                                PFIRST(CUR_LEVEL)=0
920                CONTINUE

X                  PRINT*,'GAP AT PRICE ',PTERM,' # PURGED =',N_PURG
                 END IF

C     FIND THE NEW LEVEL AND NEW PLEVEL, AND START ANOTHER PATH

                 PLEVEL=PTERM

950           PLEVEL=PLEVEL-1
                   IF (PLEVEL.EQ.0) THEN
X                    CH_TIME= CH_TIME+FLOAT(LONG(362) - TIMER)/60
                           TOT_TIME= TOT_TIME+FLOAT(LONG(362) - TIMER1)/60
                           GOTO 3000
                   END IF
                   IF (PFIRST(PLEVEL).EQ.0) GOTO 950

                 LEVEL=PTERM

980           LEVEL=LEVEL-1
                   IF (LEVEL.EQ.0) THEN
X                    CH_TIME= CH_TIME+FLOAT(LONG(362) - TIMER)/60
                     TOT_TIME= TOT_TIME+FLOAT(LONG(362) - TIMER1)/60
                     GOTO 3000
                   END IF
                   IF (FIRST(LEVEL).EQ.0) GOTO 980

X                  CH_TIME= CH_TIME+FLOAT(LONG(362) - TIMER)/60
                           GOTO 200
                END IF
                  END IF
                END IF


C     ********** START OF CONTRACTION/EXTENSION LOGIC **********

X           TIMER=LONG(362)

C     DO A CONTRACTION AT ROOT IF TERM=ROOT


            IF (TERM.EQ.ROOT) THEN
X              CEL_TIME= CEL_TIME+FLOAT(LONG(362) - TIMER)/60
                 GO TO 200
               END IF

C     CHECK WHETHER EXTENSION OR CONTRACTION

                 PRD=PRDARC(TERM)
```

```
                IF (PRD.GT.0) THEN
                   PR_TERM=STARTN(PRD)
                ELSE
                   PR_TERM=ENDN(-PRD)
                END IF
                PREVLEVEL=P(PR_TERM)

                IF (PREVLEVEL.GT.BSTLEVEL) THEN

C    PATH EXTENSION


X             NUM_EXT=NUM_EXT+1
                 IF (EXTARC.GT.0) THEN
                 END=ENDN(EXTARC)
                 TERM=END
                 ELSE
                 START=STARTN(-EXTARC)
                 TERM=START
                 END IF
                 PRDARC(TERM)=EXTARC

C    IF SINK IS FOUND, DO AN AUGMENTATION

                 IF (TERM.EQ.SINK) THEN
X                CEL_TIME= CEL_TIME+FLOAT(LONG(362) - TIMER)/60
                        GOTO 2000
                 END IF

                 ARC_COUNT=ARC_COUNT+1

C    RETURN FOR ANOTHER ITERATION

X              CEL_TIME= CEL_TIME+FLOAT(LONG(362) - TIMER)/60
                       GO TO 500
               ELSE

C    PATH CONTRACTION

                 TERM=PR_TERM
                 ARC_COUNT=ARC_COUNT-1
                 PTERM=P(TERM)
                 EXTARC=PRD
                 BSTLEVEL=BSTLEVEL+1

C   DO A SECOND BEST TEST AND IF THAT FAILS, DO A FULL NODE SCAN

X              CEL_TIME= CEL_TIME+FLOAT(LONG(362) - TIMER)/60
                 GOTO 550
            END IF
```

```
C
**********************************************************************
**

C    DO AUGMENTATION FROM ROOT, CORRECT PUSH LISTS, AND POSITIVE SURPLUS
BUCKETS

2000    CONTINUE

X        TIMER = LONG(362)

C    INITIALIZE THE AUGMENTATION

        INCR=0
      NODE = ROOT

2100    EXTARC=EXTEND_ARC(NODE)
        NAUG=NAUG+1
        IF (EXTARC.GT.0) THEN
             EXCESS=SURPLUS(NODE)
             IF (EXCESS.GT.U(EXTARC)) THEN
               NEWINCR=U(EXTARC)
             ELSE
               NEWINCR=EXCESS
             END IF

C    INSERT/REMOVE NODE IN THE POSITIVE SURPLUS BUCKETS

             IF ((NEWINCR.LT.INCR).AND.(INCR.EQ.EXCESS)) THEN

C    INSERT NODE IN THE BUCKET OF ITS PRICE

                PRICE=P(NODE)
                IFIRST=PFIRST(PRICE)
                PFIRST(PRICE)=NODE
                PPRED(NODE)=0
                PSUCC(NODE)=IFIRST
                PPRED(IFIRST)=NODE
                IF (PLEVEL.LT.PRICE) PLEVEL=PRICE
           ELSE
                 IF ((NEWINCR.EQ.EXCESS).AND.(INCR.LT.EXCESS)) THEN

C     REMOVE NODE FROM THE CURRENT POS. SURPLUS BUCKET

                 ISUCC=PSUCC(NODE)
                 IPRED=PPRED(NODE)
                 PSUCC(IPRED)=ISUCC
                 PPRED(ISUCC)=IPRED

                 IF (IPRED.EQ.0) THEN
                   PFIRST(P(NODE))=ISUCC
                   IF (ISUCC.EQ.0) THEN

C     GAP IN THE POS. SURPLUS BUCKETS HAS BEEN OBTAINED
```

```
C     FIND THE NEW POS. SURPLUS PRICE LEVEL AND START ANOTHER PATH

                              IF (PLEVEL.EQ.P(NODE)) THEN
2170             PLEVEL=PLEVEL-1
                              IF (PFIRST(PLEVEL).EQ.0) GOTO 2170
                            END IF
                            END IF
                          END IF
               END IF
             END IF
             INCR=NEWINCR
             END=ENDN(EXTARC)

C    ADD ARC TO THE REDUCED GRAPH

             IF (F(EXTARC).EQ.0) THEN
               NXTPUSHB(EXTARC)=FPUSHB(END)
               FPUSHB(END)=EXTARC
               NEW_LEVEL=P(NODE)
               IF (SB_LEVEL(END).GT.NEW_LEVEL) THEN
              SB_LEVEL(END)=NEW_LEVEL
                 SB_ARC(END)=-EXTARC
               END IF
             END IF

             F(EXTARC)=F(EXTARC)+INCR
             U(EXTARC)=U(EXTARC)-INCR

          SURPLUS(END)=SURPLUS(END)+INCR
          SURPLUS(NODE)=SURPLUS(NODE)-INCR

C    REMOVE ARC FROM THE REDUCED GRAPH

             IF (U(EXTARC).EQ.0) THEN
           ARC=FPUSHF(NODE)
               IF (ARC.EQ.EXTARC) THEN
                 FPUSHF(NODE)=NXTPUSHF(ARC)
               ELSE
                 PREVARC=ARC
            ARC=NXTPUSHF(ARC)
2200         IF (ARC.GT.0) THEN
               IF (ARC.EQ.EXTARC) THEN
                 NXTPUSHF(PREVARC)=NXTPUSHF(ARC)
                 GO TO 2250
               END IF
               PREVARC=ARC
               ARC=NXTPUSHF(ARC)
               GOTO 2200
            END IF
             END IF
X         IF (SB_LEVEL(NODE).GT.(-LARGE)) THEN
X           IF (EXTARC.EQ.SB_ARC(NODE)) THEN
X             SB_LEVEL(NODE)=-LARGE
X           PRINT*,'**CAUTION** SECARC = EXTARC IN AUGMENTATION'
```

```
X           END IF
X         END IF
              END IF

2250       NODE=END

      ELSE
              EXTARC=-EXTARC
              EXCESS=SURPLUS(NODE)
              IF (EXCESS.GT.F(EXTARC)) THEN
                NEWINCR=F(EXTARC)
              ELSE
                NEWINCR=EXCESS
              END IF

C     INSERT/REMOVE NODE IN THE THE POSITIVE SURPLUS BUCKETS

              IF ((NEWINCR.LT.INCR).AND.(INCR.EQ.EXCESS)) THEN

C     INSERT NODE IN THE BUCKET OF ITS PRICE

                 PRICE=P(NODE)
                 IFIRST=PFIRST(PRICE)
                 PFIRST(PRICE)=NODE
                 PPRED(NODE)=0
                 PSUCC(NODE)=IFIRST
                 PPRED(IFIRST)=NODE
                 IF (PLEVEL.LT.PRICE) PLEVEL=PRICE
           ELSE
                  IF ((NEWINCR.EQ.EXCESS).AND.(INCR.LT.EXCESS)) THEN

C      REMOVE NODE FROM THE CURRENT POS. SURPLUS BUCKET

                  ISUCC=PSUCC(NODE)
                  IPRED=PPRED(NODE)
                  PSUCC(IPRED)=ISUCC
                  PPRED(ISUCC)=IPRED

                  IF (IPRED.EQ.0) THEN
                    PFIRST(P(NODE))=ISUCC
                    IF (ISUCC.EQ.0) THEN

C      GAP IN THE POS. SURPLUS BUCKETS HAS BEEN OBTAINED
C      FIND THE NEW POS. SURPLUS PRICE LEVEL AND START ANOTHER PATH

                               IF (PLEVEL.EQ.P(NODE)) THEN
2270              PLEVEL=PLEVEL-1
                             IF (PFIRST(PLEVEL).EQ.0) GOTO 2270
                           END IF
                           END IF
                         END IF
                  END IF
                END IF
                INCR=NEWINCR
```

```
              START=STARTN(EXTARC)

C     ADD ARC TO THE REDUCED GRAPH

              IF (U(EXTARC).EQ.0) THEN
                NXTPUSHF(EXTARC)=FPUSHF(START)
                FPUSHF(START)=EXTARC
                NEW_LEVEL=P(NODE)
                IF (SB_LEVEL(START).GT.NEW_LEVEL) THEN
               SB_LEVEL(START)=NEW_LEVEL
                   SB_ARC(START)=EXTARC
                END IF
              END IF

              U(EXTARC)=U(EXTARC)+INCR
              F(EXTARC)=F(EXTARC)-INCR

           SURPLUS(START)=SURPLUS(START)+INCR
           SURPLUS(NODE)=SURPLUS(NODE)-INCR

C     REMOVE ARC FROM THE REDUCED GRAPH

              IF (F(EXTARC).EQ.0) THEN
             ARC=FPUSHB(NODE)
                 IF (ARC.EQ.EXTARC) THEN
                   FPUSHB(NODE)=NXTPUSHB(ARC)
                 ELSE
                   PREVARC=ARC
               ARC=NXTPUSHB(ARC)
2300           IF (ARC.GT.0) THEN
                  IF (ARC.EQ.EXTARC) THEN
                    NXTPUSHB(PREVARC)=NXTPUSHB(ARC)
                    GO TO 2350
                  END IF
                  PREVARC=ARC
                  ARC=NXTPUSHB(ARC)
                  GOTO 2300
               END IF
                 END IF
X          IF (SB_LEVEL(NODE).GT.(-LARGE)) THEN
X            IF ((-EXTARC).EQ.SB_ARC(NODE)) THEN
X              SB_LEVEL(NODE)=-LARGE
X            PRINT*,'**CAUTION** SECARC = EXTARC IN AUGMENTATION'
X            END IF
X          END IF
              END IF

2350       NODE=START

       END IF

       IF (NODE.NE.TERM) GOTO 2100

X      IF (INCR.LE.0) THEN
```

```
X       PRINT*,'INVALID AUGMENTATION'
X       PAUSE
X       STOP
X     END IF


C    CHECK IF TERMINAL NODE CHANGED FROM ZERO TO POS. SURPLUS

      IF (SURPLUS(NODE).EQ.INCR) THEN

C    INSERT NODE IN THE BUCKET OF ITS PRICE

              PRICE=P(NODE)
              IFIRST=PFIRST(PRICE)
              PFIRST(PRICE)=NODE
              PPRED(NODE)=0
              PSUCC(NODE)=IFIRST
              PPRED(IFIRST)=NODE
              IF (PLEVEL.LT.PRICE) PLEVEL=PRICE
      END IF

X       AUG_TIME= AUG_TIME+FLOAT(LONG(362) - TIMER)/60

C     *************** END OF AUGMENTATION CYCLE **************

C    CHECK FOR TERMINATION

      IF (PLEVEL.EQ.0) THEN
              TOT_TIME= TOT_TIME+FLOAT(LONG(362) - TIMER1)/60
              GOTO 3000
      END IF

C    RETURN TO START ANOTHER PATH


      GOTO 200

C    ************************************************************
C
C    ********* START OF MAX-FLOW RECTIFICATION ***********
C    ******** RETURN EXCESS FLOW TO THE SOURCE ***********
C
C    ************************************************************


3000    CONTINUE

X       PRINT*,'MINIMUM CUT FOUND'

        F_LIMIT=2

C   INITIALIZE DATA STRUCTURES

        DO 3010 NODE=1,N
```

```
              P(NODE)=N
              PFIRST(NODE)=0
3010     CONTINUE

C   INITIALIZE PRICES BY A BREADTH FIRST SEARCH FROM SINK

        NLIST=1
      P(SINK)=0
      LIST(1)=SINK
      NSCAN=0

3020     CONTINUE
        NSCAN=NSCAN+1
      NODE=LIST(NSCAN)

      ARC=FIN(NODE)
3030  IF (ARC.GT.0) THEN
              IF (U(ARC).GT.0) THEN
                START=STARTN(ARC)
                IF (P(START).EQ.N) THEN
                  NLIST=NLIST+1
                  LIST(NLIST)=START
                  P(START)=0
                END IF
              END IF
              ARC=NXTIN(ARC)
              GOTO 3030
      END IF

      ARC=FOUT(NODE)
3040  IF (ARC.GT.0) THEN
       IF (F(ARC).GT.0) THEN
              END=ENDN(ARC)
              IF (P(END).EQ.N) THEN
                NLIST=NLIST+1
                LIST(NLIST)=END
                P(END)=0
           END IF
       END IF
       ARC=NXTOU(ARC)
       GOTO 3040
      END IF

        IF (NLIST.GT.NSCAN) GOTO 3020

X       PRINT*,'# OF NODES ON SINK SIDE OF THE CUT =',NLIST

C   INITIALIZE PRICES BY A BREADTH FIRST SEARCH FROM SOURCE

      P(SOURCE)=0
      PLEVEL=0

      NLIST=1
      LIST(1)=SOURCE
```

```
      NSCAN=0

3060    CONTINUE
      NSCAN=NSCAN+1
      NODE=LIST(NSCAN)
      SB_LEVEL(NODE)=-LARGE
      PRICE=P(NODE)+1

      ARC=FIN(NODE)
3065  IF (ARC.GT.0) THEN
       IF (U(ARC).GT.0) THEN
              START=STARTN(ARC)
              IF (P(START).EQ.N) THEN
                NLIST=NLIST+1
                LIST(NLIST)=START
                IF (EXTEND_ARC(START).NE.0) EXTEND_ARC(START)=ARC
                P(START)=PRICE
                IF (SURPLUS(START).GT.0) THEN
                   IF (PLEVEL.LT.PRICE) PLEVEL=PRICE
                   IFIRST=PFIRST(PRICE)
                   PFIRST(PRICE)=START
                   PSUCC(START)=IFIRST
                   PPRED(START)=0
                   PPRED(IFIRST)=START
                ELSE
                   PSUCC(START)=0
                   PPRED(START)=0
                END IF
           END IF
       END IF
       ARC=NXTIN(ARC)
       GOTO 3065
      END IF

      ARC=FOUT(NODE)
3070  IF (ARC.GT.0) THEN
       IF (F(ARC).GT.0) THEN
              END=ENDN(ARC)
              IF (P(END).EQ.N) THEN
                NLIST=NLIST+1
                LIST(NLIST)=END
                IF (EXTEND_ARC(END).NE.0) EXTEND_ARC(END)=-ARC
                P(END)=PRICE
                IF (SURPLUS(END).GT.0) THEN
                   IF (PLEVEL.LT.PRICE) PLEVEL=PRICE
                   IFIRST=PFIRST(PRICE)
                   PFIRST(PRICE)=END
                   PSUCC(END)=IFIRST
                   PPRED(END)=0
                   PPRED(IFIRST)=END
                ELSE
                   PSUCC(END)=0
                   PPRED(END)=0
                END IF
```

```
          END IF
        END IF
        ARC=NXTOU(ARC)
        GOTO 3070
       END IF


         IF (NLIST.GT.NSCAN) GOTO 3060


C    CHECK FOR TERMINATION

       IF (PLEVEL.EQ.0) RETURN

X        INIT_TIME= INIT_TIME+FLOAT(LONG(362) - TIMER)/60


C  ***  START AUGMENTATIONS  ***


3200   CONTINUE

        ROOT=PFIRST(PLEVEL)

X          IF (SURPLUS(ROOT).EQ.0)  THEN
X            PRINT*,'SURPLUS OF ROOT IS ZERO'
X            PAUSE
X            STOP
X          END IF

        TERM=ROOT

        ARC_COUNT=0

C  *******************************************************
C
C   *** MAIN ALGORITHM WITH ROOT AS ORIGIN ***
C
C  *******************************************************

3500    CONTINUE


X         TIMER = LONG(362)

C     START OF A NEW FORWARD ITERATION

       PTERM=P(TERM)
       EXTARC=EXTEND_ARC(TERM)

X       IF (PTERM.GE.N) THEN
X         PRINT*,'*** CAUTION: NODE W/ LARGE PRICE IS TERMINAL'
X       PAUSE
X       STOP
```

```
X      END IF

       IF (EXTARC.EQ.0) THEN

X         TIMER = LONG(362)

C   BUILD THE LIST OF ARCS W/ ROOM FOR PUSHING FLOW

       FPUSHF(TERM)=0
       ARC=FOUT(TERM)
3510     IF (ARC.GT.0) THEN
               IF (U(ARC).GT.0) THEN
                 IF (FPUSHF(TERM).EQ.0) THEN
                   FPUSHF(TERM)=ARC
                   NXTPUSHF(ARC)=0
                   LAST=ARC
                 ELSE
                   NXTPUSHF(LAST)=ARC
                   NXTPUSHF(ARC)=0
                   LAST=ARC
                 END IF
               END IF
               ARC=NXTOU(ARC)
               GO TO 3510
       END IF

       FPUSHB(TERM)=0
         ARC=FIN(TERM)
3520     IF (ARC.GT.0) THEN
               IF (F(ARC).GT.0) THEN
                 IF (FPUSHB(TERM).EQ.0) THEN
                   FPUSHB(TERM)=ARC
                   NXTPUSHB(ARC)=0
                   LAST=ARC
                 ELSE
                   NXTPUSHB(LAST)=ARC
                   NXTPUSHB(ARC)=0
                   LAST=ARC
                 END IF
               END IF
               ARC=NXTIN(ARC)
               GO TO 3520
       END IF

X      PL_TIME= PL_TIME+FLOAT(LONG(362) - TIMER)/60
       GO TO 3600
       END IF

C      SPECULATIVE PATH EXTENSION ATTEMPT

X       EXT_ATT=EXT_ATT+1

       IF (EXTARC.GT.0) THEN
         IF (U(EXTARC).EQ.0) THEN
```

```
               SECLEVEL=SB_LEVEL(TERM)
X         EXT_TIME= EXT_TIME+FLOAT(LONG(362) - TIMER)/60
               GO TO 3580
       END IF
          END=ENDN(EXTARC)
          BSTLEVEL=P(END)
          IF (PTERM.GE.BSTLEVEL) THEN
X         EXT_SUCC=EXT_SUCC+1
X           NUM_EXT=NUM_EXT+1
            TERM=END
            PRDARC(TERM)=EXTARC

C     IF PATH IS LONG, DO AN AUGMENTATION

            IF (ARC_COUNT.GE.F_LIMIT) THEN
X           EXT_TIME= EXT_TIME+FLOAT(LONG(362) - TIMER)/60
                 GOTO 5000
               END IF

C     IF SOURCE IS FOUND, DO AN AUGMENTATION

               IF (TERM.EQ.SOURCE) THEN
X           EXT_TIME= EXT_TIME+FLOAT(LONG(362) - TIMER)/60
                 GOTO 5000
               END IF

C      RETURN FOR ANOTHER ITERATION

               ARC_COUNT=ARC_COUNT+1
X         EXT_TIME= EXT_TIME+FLOAT(LONG(362) - TIMER)/60
            GO TO 3500
          END IF
       ELSE
          EXTARC=-EXTARC
          IF (F(EXTARC).EQ.0) THEN
                SECLEVEL=SB_LEVEL(TERM)
X           EXT_TIME= EXT_TIME+FLOAT(LONG(362) - TIMER)/60
                GO TO 3580
       END IF
          START=STARTN(EXTARC)
          BSTLEVEL=P(START)
          IF (PTERM.GE.BSTLEVEL) THEN
X         EXT_SUCC=EXT_SUCC+1
X            NUM_EXT=NUM_EXT+1
            TERM=START
            PRDARC(TERM)=-EXTARC

C     IF PATH IS LONG, DO AN AUGMENTATION

            IF (ARC_COUNT.GE.F_LIMIT) THEN
X           EXT_TIME= EXT_TIME+FLOAT(LONG(362) - TIMER)/60
                 GOTO 5000
               END IF
```

```
C    IF SOURCE IS FOUND, DO AN AUGMENTATION

             IF (TERM.EQ.SOURCE) THEN
X         EXT_TIME= EXT_TIME+FLOAT(LONG(362) - TIMER)/60
               GOTO 5000
             END IF

C     RETURN FOR ANOTHER ITERATION

             ARC_COUNT=ARC_COUNT+1
X       EXT_TIME= EXT_TIME+FLOAT(LONG(362) - TIMER)/60
          GO TO 3500
        END IF
       END IF


C    SECOND BEST LOGIC TEST APPLIED TO SAVE A FULL NODE SCAN
C   IF OLD BEST LEVEL CONTINUES TO BE BEST GO FOR ANOTHER CONTRACTION

3550      CONTINUE

X         TIMER = LONG(362)

          SECLEVEL=SB_LEVEL(TERM)
          IF (BSTLEVEL.LE.SECLEVEL) THEN
X          SB_SUCC=SB_SUCC+1
X          SB_TIME= SB_TIME+FLOAT(LONG(362) - TIMER)/60
               GOTO 3800
          END IF

C    IF SECOND BEST CAN BE USED, DO EITHER A CONTRACTION
C    OR START OVER WITH A SPECULATIVE EXTENSION

3580     IF (SECLEVEL.GT.-LARGE) THEN
              EXTARC=SB_ARC(TERM)
           IF (EXTARC.GT.0) THEN
                 IF (U(EXTARC).EQ.0) THEN
X             SB_TIME= SB_TIME+FLOAT(LONG(362) - TIMER)/60
                GOTO 3600
                 END IF
                 BSTLEVEL=P(ENDN(EXTARC))
              ELSE
                 IF (F(-EXTARC).EQ.0) THEN
X             SB_TIME= SB_TIME+FLOAT(LONG(362) - TIMER)/60
                GOTO 600
                 END IF
              BSTLEVEL=P(STARTN(-EXTARC))
              END IF
              IF (BSTLEVEL.EQ.SECLEVEL) THEN
X          SB_SUCC=SB_SUCC+1
                 SB_LEVEL(TERM)=-LARGE
                 EXTEND_ARC(TERM)=EXTARC
X          SB_TIME= SB_TIME+FLOAT(LONG(362) - TIMER)/60
                 GOTO 3800
```

```
            END IF
       END IF

C
C   EXTENSION ATTEMPT WAS UNSUCCESSFUL, SO SCAN TERMINAL NODE
C

3600   CONTINUE

X       TIMER = LONG(362)


       BSTLEVEL=LARGE
       SECLEVEL=LARGE


        ARC=FPUSHF(TERM)
3700    IF (ARC.GT.0) THEN
             NEW_LEVEL = P(ENDN(ARC))
             IF (NEW_LEVEL.LT.SECLEVEL) THEN
               IF (NEW_LEVEL.LT.BSTLEVEL) THEN
                 SECLEVEL=BSTLEVEL
                 BSTLEVEL=NEW_LEVEL
                 SECARC=EXTARC
                 EXTARC=ARC
               ELSE
                 SECLEVEL=NEW_LEVEL
                 SECARC=ARC
               END IF
             END IF
             ARC=NXTPUSHF(ARC)
             GOTO 3700
       END IF

       ARC=FPUSHB(TERM)
3710   IF (ARC.GT.0) THEN
       NEW_LEVEL = P(STARTN(ARC))
       IF (NEW_LEVEL.LT.SECLEVEL) THEN
              IF (NEW_LEVEL.LT.BSTLEVEL) THEN
                  SECLEVEL=BSTLEVEL
                  BSTLEVEL=NEW_LEVEL
                  SECARC=EXTARC
                  EXTARC=-ARC
                ELSE
                  SECLEVEL=NEW_LEVEL
                  SECARC=-ARC
              END IF
       END IF
       ARC=NXTPUSHB(ARC)
       GOTO 3710
       END IF

       SB_LEVEL(TERM)=SECLEVEL
       SB_ARC(TERM)=SECARC
```

```
       EXTEND_ARC(TERM)=EXTARC
X       IT_TIME= IT_TIME+FLOAT(LONG(362) - TIMER)/60

C    *************  END OF NODE SCAN  *************
C     IF THE TERMINAL NODE IS THE ROOT, ADJUST ITS PRICE AND CHANGE ROOT
3800    CONTINUE

           NEW_LEVEL=BSTLEVEL+1
           P(TERM)=NEW_LEVEL
C     CHECK FOR PRICE CHANGE
           IF (PTERM.LT.NEW_LEVEL) THEN
              NITER=NITER+1
X            TIMER = LONG(362)
C     IF TERM HAS POSITIVE SURPPLUS ADJUST THE POS. SURPLUS BUCKETS
              IF (SURPLUS(TERM).GT.0) THEN

                   ISUCC=PSUCC(TERM)
                   IPRED=PPRED(TERM)
                   PSUCC(IPRED)=ISUCC
                   PPRED(ISUCC)=IPRED
                IF (IPRED.EQ.0) PFIRST(PTERM)=ISUCC

                   IF (PLEVEL.LT.NEW_LEVEL) PLEVEL=NEW_LEVEL

                   NODE=PFIRST(NEW_LEVEL)
                   PFIRST(NEW_LEVEL)=TERM
                   PPRED(NODE)=TERM
                   PSUCC(TERM)=NODE
                   PPRED(TERM)=0
              END IF
           END IF
C     ********** START OF CONTRACTION/EXTENSION LOGIC **********
X          TIMER=LONG(362)
C     DO A CONTRACTION AT ROOT IF TERM=ROOT

           IF (TERM.EQ.ROOT) THEN
X             CEL_TIME= CEL_TIME+FLOAT(LONG(362) - TIMER)/60
                GO TO 3200
```

```
                END IF

C     CHECK WHETHER EXTENSION OR CONTRACTION

                PRD=PRDARC(TERM)

                IF (PRD.GT.0) THEN
                   PR_TERM=STARTN(PRD)
                ELSE
                   PR_TERM=ENDN(-PRD)
                END IF
                PREVLEVEL=P(PR_TERM)

                IF (PREVLEVEL.GT.BSTLEVEL) THEN

C    PATH EXTENSION


X             NUM_EXT=NUM_EXT+1
                  IF (EXTARC.GT.0) THEN
                  END=ENDN(EXTARC)
                  TERM=END
                  ELSE
                  START=STARTN(-EXTARC)
                  TERM=START
                  END IF
                  PRDARC(TERM)=EXTARC

C    IF SOURCE IS FOUND, DO AN AUGMENTATION

                  IF (TERM.EQ.SOURCE) THEN
X               CEL_TIME= CEL_TIME+FLOAT(LONG(362) - TIMER)/60
                          GOTO 5000
                  END IF

                  ARC_COUNT=ARC_COUNT+1

C    RETURN FOR ANOTHER ITERATION

X             CEL_TIME= CEL_TIME+FLOAT(LONG(362) - TIMER)/60
                        GO TO 3500
               ELSE

C    PATH CONTRACTION

                  TERM=PR_TERM
                  ARC_COUNT=ARC_COUNT-1
                  PTERM=P(TERM)
                  EXTARC=PRD
                  BSTLEVEL=BSTLEVEL+1

C   DO A SECOND BEST TEST AND IF THAT FAILS, DO A FULL NODE SCAN

X             CEL_TIME= CEL_TIME+FLOAT(LONG(362) - TIMER)/60
```

```
                GOTO 3550

           END IF


C   ***********************************************************
C    DO AUGMENTATION FROM ROOT, CORRECT POSITIVE SURPLUS BUCKETS

5000    CONTINUE

X        TIMER = LONG(362)

C    INITIALIZE THE AUGMENTATION

        INCR=0
      NODE = ROOT

5100    EXTARC=EXTEND_ARC(NODE)
      NAUG=NAUG+1
      IF (EXTARC.GT.0) THEN
             EXCESS=SURPLUS(NODE)
             IF (EXCESS.GT.U(EXTARC)) THEN
               NEWINCR=U(EXTARC)
             ELSE
               NEWINCR=EXCESS
             END IF

C    INSERT/REMOVE NODE IN THE POSITIVE SURPLUS BUCKETS

             IF ((NEWINCR.LT.INCR).AND.(INCR.EQ.EXCESS)) THEN

C    INSERT NODE IN THE BUCKET OF ITS PRICE

                PRICE=P(NODE)
                IFIRST=PFIRST(PRICE)
                PFIRST(PRICE)=NODE
                PPRED(NODE)=0
                PSUCC(NODE)=IFIRST
                PPRED(IFIRST)=NODE
                IF (PLEVEL.LT.PRICE) PLEVEL=PRICE
           ELSE
                 IF ((NEWINCR.EQ.EXCESS).AND.(INCR.LT.EXCESS)) THEN

C     REMOVE NODE FROM THE CURRENT POS. SURPLUS BUCKET

                 ISUCC=PSUCC(NODE)
                 IPRED=PPRED(NODE)
                 PSUCC(IPRED)=ISUCC
                 PPRED(ISUCC)=IPRED

                 IF (IPRED.EQ.0) THEN
                   PFIRST(P(NODE))=ISUCC
                   IF (ISUCC.EQ.0) THEN
```

```
C     GAP IN THE POS. SURPLUS BUCKETS HAS BEEN OBTAINED
C     FIND THE NEW POS. SURPLUS PRICE LEVEL

                              IF (PLEVEL.EQ.P(NODE)) THEN
5170             PLEVEL=PLEVEL-1
                              IF (PFIRST(PLEVEL).EQ.0) GOTO 5170
                            END IF
                            END IF
                          END IF
               END IF
              END IF
              INCR=NEWINCR
              END=ENDN(EXTARC)

C    ADD ARC TO THE REDUCED GRAPH

              IF (F(EXTARC).EQ.0) THEN
                NXTPUSHB(EXTARC)=FPUSHB(END)
                FPUSHB(END)=EXTARC
                NEW_LEVEL=P(NODE)
                IF (SB_LEVEL(END).GT.NEW_LEVEL) THEN
               SB_LEVEL(END)=NEW_LEVEL
                  SB_ARC(END)=-EXTARC
                END IF
              END IF


              F(EXTARC)=F(EXTARC)+INCR
              U(EXTARC)=U(EXTARC)-INCR

           SURPLUS(END)=SURPLUS(END)+INCR
           SURPLUS(NODE)=SURPLUS(NODE)-INCR

C    REMOVE ARC FROM THE REDUCED GRAPH

              IF (U(EXTARC).EQ.0) THEN
             ARC=FPUSHF(NODE)
                IF (ARC.EQ.EXTARC) THEN
                  FPUSHF(NODE)=NXTPUSHF(ARC)
                ELSE
                  PREVARC=ARC
               ARC=NXTPUSHF(ARC)
5200           IF (ARC.GT.0) THEN
                  IF (ARC.EQ.EXTARC) THEN
                    NXTPUSHF(PREVARC)=NXTPUSHF(ARC)
                    GO TO 5250
                  END IF
                  PREVARC=ARC
                  ARC=NXTPUSHF(ARC)
                  GOTO 5200
               END IF
                END IF
X          IF (SB_LEVEL(NODE).GT.(-LARGE)) THEN
X            IF (EXTARC.EQ.SB_ARC(NODE)) THEN
```

```
X             SB_LEVEL(NODE)=-LARGE
X           PRINT*,'**CAUTION** SECARC = EXTARC IN AUGMENTATION'
X           END IF
X         END IF
              END IF

5250       NODE=END


      ELSE
              EXTARC=-EXTARC
              EXCESS=SURPLUS(NODE)
              IF (EXCESS.GT.F(EXTARC)) THEN
                NEWINCR=F(EXTARC)
              ELSE
                NEWINCR=EXCESS
              END IF

C     INSERT/REMOVE NODE IN THE THE POSITIVE SURPLUS BUCKETS

              IF ((NEWINCR.LT.INCR).AND.(INCR.EQ.EXCESS)) THEN

C     INSERT NODE IN THE BUCKET OF ITS PRICE

                  PRICE=P(NODE)
                  IFIRST=PFIRST(PRICE)
                  PFIRST(PRICE)=NODE
                  PPRED(NODE)=0
                  PSUCC(NODE)=IFIRST
                  PPRED(IFIRST)=NODE
                  IF (PLEVEL.LT.PRICE) PLEVEL=PRICE
            ELSE
                   IF ((NEWINCR.EQ.EXCESS).AND.(INCR.LT.EXCESS)) THEN

C      REMOVE NODE FROM THE CURRENT POS. SURPLUS BUCKET

                   ISUCC=PSUCC(NODE)
                   IPRED=PPRED(NODE)
                   PSUCC(IPRED)=ISUCC
                   PPRED(ISUCC)=IPRED

                   IF (IPRED.EQ.0) THEN
                     PFIRST(P(NODE))=ISUCC
                     IF (ISUCC.EQ.0) THEN

C      GAP IN THE POS. SURPLUS BUCKETS HAS BEEN OBTAINED
C      FIND THE NEW POS. SURPLUS PRICE LEVEL

                                IF (PLEVEL.EQ.P(NODE)) THEN
5270                PLEVEL=PLEVEL-1
                              IF (PFIRST(PLEVEL).EQ.0) GOTO 5270
                            END IF
                            END IF
                          END IF
```

```
                END IF
              END IF
              INCR=NEWINCR
              START=STARTN(EXTARC)

C     ADD ARC TO THE REDUCED GRAPH

              IF (U(EXTARC).EQ.0) THEN
                NXTPUSHF(EXTARC)=FPUSHF(START)
                FPUSHF(START)=EXTARC
                NEW_LEVEL=P(NODE)
                IF (SB_LEVEL(START).GT.NEW_LEVEL) THEN
               SB_LEVEL(START)=NEW_LEVEL
                   SB_ARC(START)=EXTARC
                END IF
              END IF


              U(EXTARC)=U(EXTARC)+INCR
              F(EXTARC)=F(EXTARC)-INCR

           SURPLUS(START)=SURPLUS(START)+INCR
           SURPLUS(NODE)=SURPLUS(NODE)-INCR

C     REMOVE ARC FROM THE REDUCED GRAPH

              IF (F(EXTARC).EQ.0) THEN
             ARC=FPUSHB(NODE)
                 IF (ARC.EQ.EXTARC) THEN
                   FPUSHB(NODE)=NXTPUSHB(ARC)
                 ELSE
                   PREVARC=ARC
               ARC=NXTPUSHB(ARC)
5300           IF (ARC.GT.0) THEN
                  IF (ARC.EQ.EXTARC) THEN
                    NXTPUSHB(PREVARC)=NXTPUSHB(ARC)
                    GO TO 5350
                  END IF
                  PREVARC=ARC
                  ARC=NXTPUSHB(ARC)
                  GOTO 5300
               END IF
                 END IF
X          IF (SB_LEVEL(NODE).GT.(-LARGE)) THEN
X            IF ((-EXTARC).EQ.SB_ARC(NODE)) THEN
X              SB_LEVEL(NODE)=-LARGE
X            PRINT*,'**CAUTION** SECARC = EXTARC IN AUGMENTATION'
X            END IF
X          END IF
                 END IF

5350       NODE=START

       END IF
```

```
      IF (NODE.NE.TERM) GOTO 5100

X     IF (INCR.LE.0) THEN
X       PRINT*,'INVALID AUGMENTATION'
X       PAUSE
X       STOP
X     END IF


C     CHECK IF TERMINAL NODE CHANGED FROM ZERO TO POS. SURPLUS

      IF (SURPLUS(NODE).EQ.INCR) THEN

C     INSERT NODE IN THE BUCKET OF ITS PRICE

              PRICE=P(NODE)
              IFIRST=PFIRST(PRICE)
              PFIRST(PRICE)=NODE
              PPRED(NODE)=0
              PSUCC(NODE)=IFIRST
              PPRED(IFIRST)=NODE
              IF (PLEVEL.LT.PRICE) PLEVEL=PRICE
      END IF

X       AUG_TIME= AUG_TIME+FLOAT(LONG(362) - TIMER)/60

C     *************** END OF AUGMENTATION CYCLE **************

C     CHECK FOR TERMINATION

      IF (PLEVEL.EQ.0) RETURN

C     RETURN TO START ANOTHER PATH


      GOTO 3200

      END




C     ***************************************************
      SUBROUTINE INIDAT
C     THIS SUBROUTINE USES THE DATA ARRAYS STARTN AND ENDN
C     TO CONSTRUCT AUXILIARY DATA ARRAYS FOUT, NXTOU, FIN, AND
C     NXTIN THAT ARE REQUIRED BY AUCT_MF.  IN THIS SUBROUTINE WE
C     ARBITRARILY ORDER THE ARCS LEAVING EACH NODE AND STORE
C     THIS INFORMATION IN FOUT AND NXTOU.  SIMILARLY, WE ARBITRA-
C     RILY ORDER THE ARCS ENTERING EACH NODE AND STORE THIS
C     INFORMATION IN FIN AND NXTIN.  AT THE COMPLETION OF THE
C     CONSTRUCTION, WE HAVE
```

```
C
C         FOUT(I)    = FIRST ARC LEAVING NODE I.
C         NXTOU(J)   = NEXT ARC LEAVING THE HEAD NODE OF ARC J.
C         FIN(I)     = FIRST ARC ENTERING NODE I.
C         NXTIN(J)   = NEXT ARC ENTERING THE TAIL NODE OF ARC J.

      PARAMETER (MAXNODES=20001, MAXARCS=120000)
      IMPLICIT INTEGER (A-Z)
      COMMON /SCALARS/  N,NA,LARGE,SOURCE,SINK
      COMMON /BLK1/     STARTN
      COMMON /BLK2/     ENDN
      COMMON /BLK4/     FIN
      COMMON /BLK5/     FOUT
      COMMON /BLK6/     NXTIN
      COMMON /BLK7/     NXTOU
      INTEGER STARTN(MAXARCS)
      INTEGER ENDN(MAXARCS)
      INTEGER FIN(MAXNODES),FOUT(MAXNODES)
      INTEGER NXTIN(MAXARCS),NXTOU(MAXARCS)
      INTEGER FINALIN(MAXNODES),FINALOU(MAXNODES)
C
      DO 20 NODE=1,N
        FIN(NODE)=0
        FOUT(NODE)=0
        FINALIN(NODE)=0
        FINALOU(NODE)=0
20    CONTINUE
C
      DO 30 ARC=1,NA
        START=STARTN(ARC)
        END=ENDN(ARC)
        IF (FOUT(START).NE.0) THEN
          NXTOU(FINALOU(START))=ARC
        ELSE
          FOUT(START)=ARC
        END IF
        IF (FIN(END).NE.0) THEN
          NXTIN(FINALIN(END))=ARC
        ELSE
          FIN(END)=ARC
        END IF
        FINALOU(START)=ARC
        FINALIN(END)=ARC
        NXTIN(ARC)=0
        NXTOU(ARC)=0
30    CONTINUE
C
      RETURN
      END
```