A FORTRAN CODE FOR THE TRANS-SHIPMENT PROBLEM

Markku Kallio
Andras Por
Margareta Soismaa

April 1979
WP-79-26

INTERNATIONAL INSTITUTE FOR APPLIED SYSTEMS ANALYSIS
A-2361 Laxenburg, Austria

Margareta Soismaa is a Research Assistant at the Helsinki School of Economics.

ACKNOWLEDGEMENT

The authors are grateful to William Orchard-Hays for his helpful comments. Of course, however, the authors are responsible for possible remaining mistakes.

ABSTRACT

A code written in FORTRAN for PDP-11 is reported for solving the capacitated transshipment problem.

A FORTRAN CODE FOR THE TRANSSHIPMENT PROBLEM

Markku Kallio, Andras Por and
Margareta Soismaa

1. Introduction and Summary

Capacitated transshipment problems comprise an important class of structured linear programming problems. Being the most general pure network problems, they have found a wide variety of applications for problems such as transportation, manpower planning, water resources management, regional location problems, production-inventory systems, cash management, etc.

The main advantage of the transshipment problems is that the problem structure can be exploited in a very efficient way while solving the problem. This amounts to far less computer time and core requirement than what is the case when using standard LP software. For instance, network problems which are considered large in the usual LP terminology, can be handled even by the relatively small PDP-11 at IIASA. Another fundamental feature of these network problems is that the optimal solution is integral provided that the problem data is integral. This is important, as such optimal integer solutions for optimization problems are extremely difficult to obtain in general. This property is applicable, for instance, to some regional development problems within IIASA.

The code reported here has been written for solving the capacitated transshipment problem, as well as its special cases, such as transportation, assignment, maximum flow, etc. problems. The program has been written in FORTRAN for PDP-11. Currently, the maximum problem size is restricted by $1.8\ m + n \leq 3640$, where m is the number of nodes and n is the number of arcs (including slack and possible artificial arcs). Of course, this restriction can be easily relaxed if the necessary core is available.

The simplex method has been employed to solve the optimization problem. Our main goal then was to exploit the structural properties (such as the triangularity) for the basis matrices. A special scheme of a forthcoming paper [2] has been implemented for updating the triangular representation of the basis at each simplex iteration. (For an excellent presentation of network techniques and further references, see also reference [1].) Another goal was to keep the core requirement at a moderate level. However, through further polishing, core requirement can still be reduced rather easily, for instance, taking a more sophisticated approach for storing the arc list and the upper bound vector.

The input/output section has been designed as simple as possible. In actual use, one should easily be able to change these sections if necessary. Neither has much attention been paid to initialization of the algorithm when no advanced starting basis is available. Indeed, in such a case, we start with an all logical basis where the cost coefficients for slacks, as usual, have been set to zero and for artificials to a number, which is large enough to prevent them appearing in the optimal solution.

2. The Capacitated Transshipment Problem

Let N be a set of nodes j and A a set of arcs (directed pairs of indices ij) pointing from i to j, for some $i \in N$ and $j \in N$. For each node $j \in N$, define $D_j = \{jk : jk \in A\}$ and $S_j = \{ij : ij \in A\}$ as the set of arcs starting from j and ending to j, respectively. In this notation, the capacitated transshipment problem is to

$$\text{find} \quad x_{ij} \text{ for all } ij \in A \text{ to}$$

$$\text{minimize} \sum_{ij \in A} c_{ij} x_{ij} \tag{1}$$

subject to

$$\sum_{ij \in S_j} x_{ij} - \sum_{jk \in D_j} x_{jk} \mathrel{\hat{=}} d_j \qquad \text{for all } j \in N \;, \tag{2}$$

$$0 \le x_{ij} \le u_{ij} \qquad \text{for all } ij \in A \;, \tag{3}$$

Where $\hat{=}$ stands either for =, for $\le$ or for $\ge$. Here x_{ij} is the flow (or shipment) along arc ij from node i to node j, c_{ij} is the corresponding unit cost for transporting from i to j, u_{ij} is the maximum possible amount for flow x_{ij} (which may be infinite) and d_j is the external flow (out) from node j. We may also interprete d_j as an external demand at node j. If this is negative it actually means supply.

Problem (1)-(3) is a structured linear programming problem which is characterized by the following: each variable x_{ij} has two nonzero coefficients in constraints (2), one being equal to 1 and the other equal to -1. This property implies that each basis matrix corresponding to constraint (2) is a triangular matrix. This fact shall be exploited in our code for solving the problem (1)-(3).

Remark. If all constraints (2) are of equality type, then $\sum d_j$ must be 0 in order for the problem to be feasible. In this case, one or more of the equations are redundant: they can be obtained as linear combinations of other constraints. Corresponding to a redundant constraint, an artificial variable will appear in the optimal basis at zero level. For computing the final price vector, we shall, at the end, set the cost coefficients for such artificials to zero. According to the usual convention, this amounts to a dual variable equal to zero for each redundant constraint.

3. System Description

The optimization system consists of the main program, an input subroutine INPUT, an output subroutine PRINT, two minor subroutines IDEN and ORDER, and the optimizer subroutine SLEX. A program list appears in the Appendix. The system is implemented on our in-house PDP-11/70. The load module can be called as a UNIX shell command of the following form:

```
TRNETWORK 5=input 6=+output
```

where "input" is the name of the problem input file (see below), and "output" is a printfile name. The program also uses a file for saving the basis. The name of this file has to be defined by the user as indicated shortly.

The program first needs the problem description from a Problem Input File numbered as 5 in our code. We shall give a precise description for this file shortly. If the problem is too large, the following message will be printed before termination:

THE PROBLEM IS TOO LARGE,

ADDITIONAL CORE REQUIREMENT IS *** BYTES

Otherwise, slacks and artificials are first constructed, and thereafter the maximum allowable number of iterations is requested:

ENTER MAXIMUM NUMBER OF INTERATIONS:

If nothing is inserted (press RETURN key), a default value equal to 5 times the number of nodes is used. Next an initial basis is requested:

ENTER THE NAME OF STARTING BASIS:

If nothing is inserted, the main program constructs an all logical starting basis. Otherwise, the file named for an advanced basis is loaded. The description of such a Basis File is given below. This file may have been generated during previous runs or it may have been generated by other means. The starting basis need not be feasible in the sense that the basic solution corresponding to the initial basis may not be within bounds (3). Such a basis may also include artificial variables which cannot appear at a nonzero level at any feasible solution for the transshipment problem.

Optimization may terminate in the following cases:

(a) an optimal basic solution has been found;

(b) an unbounded optimal solution has been detected;

(c) the problem is found to be infeasible;

(d) the maximum allowable number of iterations has been reached.

In each case, a name for the final basis is requested:

ENTER THE NAME OF THE FINAL BASIS:

If nothing is entered, the basis will not be saved. Otherwise, the basis is saved under the file name entered containing at most 8 characters. The format of this file is as described below for a Basis File.

The final basic solution is printed and an indication given as to which case (a to d) occurred. The format is described under Solution Printout section below.

Problem Input File

The problem file is a file with the following structure:

1 8	16	24	32	40	position
NODE	DEMAND	TYPE			
node name	demand	type			Node section
ARC	FROM	TO	COST	BOUND	
arc number	starting node name	ending node name	unit cost	upper bound on arc	Arc section
ENDATA					

Figure 1. Problem input file.

For the node section, there is a title record containing NODE in positions 5-8, and one record for each node j. The node name contains at most 8 characters. Demand d_j may be positive, negative (meaning supply), or zero. The node TYPE is defined as

EQ if constraint j in (2) is of "=" - type
LE if constraint j in (2) is of "$\leq$" - type
GE if constraint j in (2) is of "$\geq$" - type.

The FORTRAN format for a node record is (A8,I8,6x,A2).

For the arc section, there is a title record containing ARC in positions 6-8, and one record for each arc ij. The names of the starting node i and the ending node j, as well as the unit cost c_{ij} and the upper bound u_{ij} is given. The arcs will be numbered in the same order as they are entered. This running number may also be given in the ARC column. The FORTRAN format for an arc record is (I8,2A8,2I8). The arc section is terminated with a record containing ENDATA in positions 1-6.

An Example of a Problem File. Consider the following transshipment network, where the circled nodes have their numbers j inside, each arc from i to j denotes a pair ij in A. The cost coefficients c_{ij} and upper bounds u_{ij} associated with arcs have been expressed by pairs of numbers. Demand d_j at node j is associated with an arc pointing out from that node (and not pointing to another node). The node type ($\leq$, =, or $\geq$) has been written beside the node demand. For this problem we have the set of nodes N = {1,2,3,...,12} and the set of arcs A = {1-5,1-7, 1-8,1-9,2-3,2-6,2-9,2-11,2-12,3-4,3-9,3-10,4-8,4-10,5-8,6-9,6-12}.

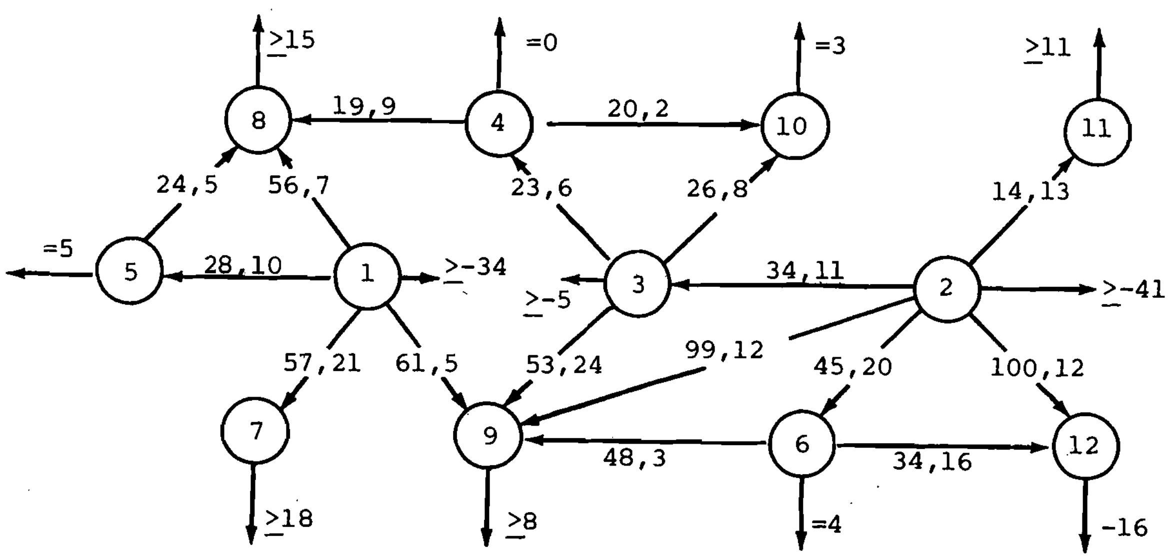

Figure 2. A transshipment network.

The problem input file is as follows:

```
   node   demand    type
      1      -34      ge
      2      -41      ge
      3       -5      eq
      4        0      eq
      5        5      eq
      6        4      eq
      7       18      ge
      8       15      ge
      9        8      ge
     10        3      eq
     11       11      ge
     12       16      eq
    arc     from      to    cost     ub
      1        2       3      34     11
      2        3       4      23      6
      3        1       5      28     10
      4        2       6      45     20
      5        1       7      57     21
      6        5       8      24      5
      7        1       8      56      7
      8        4       8      19      9
      9        1       9      61      5
     10        2       9      99     12
     11        6       9      48      3
     12        3       9      53     24
     13        3      10      26      8
     14        4      10      20      2
     15        2      11      14     13
     16        6      12      34     16
     17        2      12     100     12
endata
```

Figure 3. An example of problem input file.

Solution Printout

For the final basic solution the associated total cost as well as arc and node data is printed. For the arc-section, the following information is shown:

ARC = arc number for arc ij

FROM = starting node name of i

TO = ending node name of j

ACTIVITY = x_{ij}, activity level at basic solution

BOUND = upper bound u_{ij}

COST = unit cost c_{ij}

RESUCED COST = reduced cost for x_{ij}

The node section contains the following information:

NODE = node name

SLACK ACTIVITY = slack for an inequality type node

DUAL ACTIVITY = simplex multiplier corresponding to node

Example. The optimal solution of our example would be reported as follows:

node	slack activity	dual activity
1	0	26.00
2	0	0.00
3	0	34.00
4	0	63.00
5	0	54.00
6	0	45.00
7	0	83.00
8	0	82.00
9	0	87.00
10	0	60.00
11	0	14.00

Figure 4. A solution printout: Node section.

optimal solution at iteration number 12

arc	from	to	activity	bound	cost	reduced cost
1	2	3	10	11	34.00	0.00
2	3	4	6	6	23.00	-6.00
3	1	5	10	10	28.00	0.00
4	2	6	20	20	45.00	0.00
5	1	7	18	21	57.00	0.00
6	5	8	5	5	24.00	-4.00
7	1	8	4	7	56.00	0.00
8	4	8	6	9	19.00	0.00
9	1	9	2	5	61.00	0.00
10	2	9	0	12	99.00	12.00
11	6	9	0	3	48.00	6.00
12	3	9	6	24	53.00	0.00
13	3	10	3	8	26.00	0.00
14	4	10	0	2	20.00	23.00
15	2	11	11	13	14.00	0.00
16	6	12	16	16	34.00	0.00
17	2	12	0	12	100.00	21.00

total cost = 4358.00

Figure 4. A solution printout: Arc section.

Basis File

The basis file defines a basic solution, a basis matrix and a triangular permutation for this basis.

First there is a record for each node j containing an arc number B(j) (or its negative) and a row number W(j) in FORTRAN format (2I8). The order in which basic arcs are entered is the order of corresponding columns from left to right in the basis. Row number W(j) defines a row permutation: the constraint of node j corresponds to the W(j)th row in the permuted basis. The permuted basis determined by B(•) and W(•) has to be lower triangular. Furthermore, if the jth diagonal element of this matrix is equal to -1, then B(j) is the negative of the corresponding arc number. Otherwise, B(j) is the arc number.

Next, there is a list of variables at their upper bounds. At most ten such variables may be indicated in one record. The FORTRAN format for such a record is (10I8). At the end of this list there is a record containing ENDBASIS in positions 1-8. This indicates the end of a basis file. The optimal basis of our example is given in Figure 5 below.

```
       3       6
       5      12
      13       8
      -8       4
       7       1
      -9      11
      12       2
       1       5
      15       7
      16       3
       4       9
     -19      10
       2       6
endbasis
```

Figure 5. A basis file.

REFERENCES

[1] Bradley, G.H., G.G. Brown and G.W. Graves, "Design and Implementation of Large-Scale Primal Transshipment Algorithms," Management Science, 24 (1977), 1-33.

[2] Kallio and M. Soismaa, "On Basis Representation for Network Optimization Problems," forthcoming.

APPENDIX

```
c *********************************************************************
c       main program
c *********************************************************************
      implicit integer (a-z)
      real *8 dr(3000),nodes(1),nam(6),naml,nam2,drl(1550)
      real rr(1)
      integer *2 ir(1),typ(1),dem(1)
      equivalence (dr(1),rr(1)), (dr(1),ir(1)), (dr(1),nodes(1)),
     1 (dr(2000),typ(1)), (dr(3000),dem(1)), (dr(3001),drl(1))
      data naml /8h    node/, nam2 /8h     arc/, eq /2heq/, le /2hle/,
     1 ge /2hge/, icore /18200/
c icore = length of array dr(.) and array drl(.)  in words.
c           (word = 2 bytes)
      n = 0
      read (5,99999) nam(1)
      if (nam(1).eq.naml) go to 10
      write (6,99998)
      stop 22
c read node information
   10 read (5,99999) (nam(i),i=1,3)
      if (nam(1).eq.nam2) go to 20
      n = n + 1
      decode(24,99997,nam) nodes(n),dem(n),ty
      typ(n) = 0
      if (ty.eq.le) typ(n) = 1
      if (ty.eq.ge) typ(n) = -1
      go to 10
c distribution of the core defined by arrays dr(.) and drl(.)
c among arrays required by procedures input and slex.
   20 kl = 4*n
      k2 = 5*n
      do 30 i=1,n
         ir(kl+i) = typ(i)
         ir(k2+i) = dem(i)
   30 continue
      arcmax = (icore-9*n)/5
      kl = 2*n + 1
      k2 = 4*n + 1
      k3 = 2*(kl+n+arcmax-1) + 1
      k4 = 3*n + k3
      call input(dr, rr(kl), rr(kl+n), ir(k2), ir(k2+n), ir(k3),
     1 ir(k3+n), ir(2*n+k3), ir, ir(n+1), ir(k4), ir(k4+arcmax),
     2 arcmax, n)
      stop 11
99999 format (5a8)
99998 format (1x, 27hnode-section does not exist)
99997 format (a8, i8, 6x, a2)
      end
```

```
      subroutine input(nodes, pii, cost, typ, dem, x, b, w, alfa, av,
     1 ub, a, arcmax, n)
      implicit integer (a-z)
      real pii(1), cost(1), comax, max
      logical *1 nam31(8),sp
      real *8 nodes(1),nam(10),nam3,eob
      dimension a(2,1), ub(1), alfa(1), av(2,1), w(1), x(1), b(1),
     1 typ(1), dem(1), ib(10)
      equivalence (nam3,nam31(1))
      data nam3 /8hendata  /, sp /1h /, eob /8hendbasis/
      narc = 0
      narcx = 0
      do 10 i=1,arcmax
         ub(i) = 0
   10 continue
      comax = -1.e32
c read arc information
   20 read (5,99992) (nam(i),i=1,5)
      if (nam(1).eq.nam3) go to 40
      if (narc.ge.arcmax) narcx = narcx + 1
      if (narcx.ne.0) go to 30
      narc = narc + 1
   30 call iden(nodes, n, nam(2), a, narc)
      decode(40,99991,nam) a(2,narc),a(1,narc),cost(narc),ub(narc)
      if (cost(narc).lt.comax) go to 20
      comax = cost(narc)
      go to 20
   40 narcl = narc
c compute cost coefficient for artificial arcs
c construct slack and artificial arcs
      max = 2.*float(n)*comax + 1.
      do 90 i=1,n
         w(i) = i
         ii = i
         jj = 0
         nn = 2
         if (typ(i).ne.1) go to 50
         nn = 1
         ii = 0
         jj = i
   50    if (narc.ge.arcmax) narcx = narcx + 1
         if (narcx.ne.0) go to 60
         a(2,narc+1) = ii
         a(1,narc+1) = jj
         cost(narc+1) = 0
         if (typ(i).eq.0) cost(narc+1) = max
         narc = narc + 1
   60    if (typ(i).eq.-1 .and. dem(i).le.0) go to 80
         if (typ(i).eq.1 .and. dem(i).ge.0) go to 80
         if (typ(i).eq.0 .and. dem(i).le.0) go to 80
         if (narc.lt.arcmax) go to 70
         narcx = narcx + 1
         if (typ(i).eq.0) narcx = narcx - 1
         go to 90
   70    if (typ(i).eq.0) narc = narc - 1
         a(2,narc+1) = jj
```

```
          a(l,narc+l) = ii
          cost(narc+l) = max
          narc = narc + 1
          nn = 1
          if (typ(i).eq.l) nn = 2
c define initial basis using slacks and artificials
   80     x(i) = iabs(dem(i))
          b(i) = narc
          if (nn.eq.2) b(i) = -b(i)
   90 continue
      if (narcx.eq.0) go to 100
c run is abandoned due to violation of the available core size
      narcx = 10*narcx
      write (6,99999) narcx
      stop
  100 write (7,99998)
      read (1,99997) niter
      if (niter.eq.0) niter = 5*n
      write (7,99996)
      read (1,99994) nam3
      do 110 i=1,8
         if (nam31(l).ne.sp) go to 120
  110 continue
      phase = 2
      go to 210
c restore basis from the file whose name is contained in variable nam3
  120 call setfil(8, nam3)
      do 130 i=l,n
         read (8,99993) b(i), w(i)
  130 continue
  140 read (8,99994) nam
      if (nam(l).eq.eob) go to 160
      decode(80,99993,nam) ib
      do 150 i=1,10
         if (ib(i).eq.0) go to 160
         il = ib(i)
         ub(il) = -ub(il)
  150 continue
      go to 140
  160 call closef(8)
c setup of basis solution
      phase = 2
      do 170 i=l,narc
         if (ub(i).ge.0) go to 170
         il = a(2,i)
         i2 = a(l,i)
         dem(il) = dem(il) - ub(i)
         dem(i2) = dem(i2) + ub(i)
  170 continue
      do 200 i=l,n
         i2 = iabs(b(i))
         il = a(2,i2)
         x(i) = -dem(il)
         if (b(i).le.0) go to 180
         il = a(l,i2)
         x(i) = dem(il)
```

```
  180     if (x(i).lt.0 .or. x(i).gt.ub(i2)) phase = 1
          do 190 j=1,2
             if (a(j,i2).eq.0) go to 190
             dem(a(j,i2)) = dem(a(j,i2)) + x(i)*(-1)**j
  190     continue
  200 continue
c save node-names on file "trnames"
  210 call setfil(2, 7htrnames)
      write (2) (nodes(i),i=1,n)
      call closef(2)
c optimize
      call slex(a, cost, ub, pii, x, b, w, n, narc, niter, ind, colin,
     1 av, alfa, phase, max)
      write (7,99995)
      read (1,99994) nam3
      do 220 i=1,8
         if (nam31(i).ne.sp) go to 230
  220 continue
      go to 270
c save basis on the file whose name is contained in variable nam3
  230 call setfil(8, nam3)
      do 240 i=1,n
         write (8,99993) b(i), w(i)
  240 continue
      il = 0
      do 250 i=1,narc
         if (ub(i).ge.0) go to 250
         il = il + 1
         ib(il) = i
         if (il.lt.10) go to 250
         write (8,99993) ib
         il = 0
  250 continue
      if (il.eq.0) go to 260
      write (8,99993) (ib(i),i=1,il)
  260 write (8,99994) eob
  270 call order(x, b, n, nnl, narcl)
c print solution
      call print(a, nodes, x, b, w, cost, ub, pii, n, narcl, nnl, n,
     1 ind, colin, niter)
99999 format (4x, 25hthe problem is too large,/4x, 17hadditional core r,
     1 14hequirement is , i5, 6h bytes)
99998 format (4x, 35henter maximum number of iterations:)
99997 format (i6)
99996 format (4x, 37henter the name of the starting basis:)
99995 format (4x, 34henter the name of the final basis:)
99994 format (10a8)
99993 format (10i8)
99992 format (5a8)
99991 format (8x, 2i8, f8.0, i8)
      end
```

```
      subroutine print(a, nodes, x, b, w, cost, ub, pii, n, narc, mml,
     1 mm2, ind, colin, niter)
      implicit integer (a-z)
      real *8 nodes(1)
      real cost(1), pii(1), px, costt
      dimension x(1), b(1), ub(1), a(2,1000), w(1)
      costt = 0.0
c restore node-names from file "trnames"
      call setfil(2, 7htrnames)
      read (2) (nodes(i),i=1,n)
      call closef(2)
c print type of solution
      if (ind.ne.3) go to 10
      write (6,99999) niter
      go to 40
   10 if (ind.ne.4) go to 20
      write (6,99998) niter
      go to 40
   20 if (ind.ne.2) go to 30
      write (6,99997) niter
      go to 40
   30 write (6,99996) niter
c print arc section
   40 lmax = 50
      lin = narc/lmax
      lre = narc - lin*lmax
      if (lre.ne.0) lin = lin + 1
      ii = 1
      jj = 1
      do 110 i=1,lin
         write (6,99995)
         lpr = lmax
         if (i.eq.lin) lpr = lre
         do 100 j=1,lpr
            xx = 0
            px = cost(ii)
            ij = 0
            kv = a(1,ii)
   50       if (kv.eq.0) go to 60
            kv = w(kv)
            if (ij.eq.0) px = px - pii(kv)
            if (ij.eq.1) px = px + pii(kv)
   60       ij = ij + 1
            if (ij.gt.1) go to 70
            kv = a(2,ii)
            go to 50
   70       if (ub(ii).ge.0) go to 80
            ub(ii) = -ub(ii)
            xx = ub(ii)
   80       if (jj.gt.mml) go to 90
            bjj = b(jj)
            if (bjj.lt.0) bjj = -bjj
            if (bjj.ne.ii) go to 90
            xx = x(jj)
            jj = jj + 1
   90       il = a(2,ii)
```

```
            i2 = a(1,ii)
            costt = costt + xx*cost(ii)
            if (ub(ii).ne.0) write (6,99994) ii, nodes(i1), nodes(i2),
     1       xx, ub(ii), cost(ii), px
            if (ub(ii).eq.0) write (6,99993) ii, nodes(i1), nodes(i2),
     1       xx, cost(ii), px
            ii = ii + 1
  100    continue
  110 continue
c print total cost
      write (6,99992) costt
c print node section
      lin = n/lmax
      lre = n - lin*lmax
      if (lre.ne.0) lin = lin + 1
      ii = 1
      do 140 i=1,lin
         write (6,99991)
         lpr = lmax
         if (i.eq.lin) lpr = lre
         do 130 j=1,lpr
            xx = 0
            kv = w(ii)
            px = pii(kv)
            if (jj.gt.mm2) go to 120
            ind = b(jj)
            if (ind.lt.0) ind = -ind
            ind1 = a(2,ind)
            if (ind1.eq.0) ind1 = a(1,ind)
            if (ind1.ne.ii) go to 120
            xx = x(jj)
            jj = jj + 1
  120       write (6,99990) nodes(ii), xx, px
            ii = ii + 1
  130    continue
  140 continue
      return
99999 format (1h1//4x, 28hsolution at iteration number, i6//)
99998 format (1h1//4x, 38hunbounded solution at iteration number,
     1 i6/4x, 10hray-arc is, i6//)
99997 format (1h1//4x, 36hoptimal solution at iteration number, i6//)
99996 format (1h1//4x, 32hinfeasible solution at iteration, i6//)
99995 format (//1x, 3harc, 6x, 4hfrom, 8x, 2hto, 3x, 8hactivity, 11x,
     1 5hbound, 12x, 4hcost, 4x, 12hreduced cost//)
99994 format (1x, i3, 2(2x, a8), 3x, i8, 4x, i12, 4x, f12.2, 4x, f12.2)
99993 format (1x, i3, 2(2x, a8), 3x, i8, 4x, 8x, 4hnone, 4x, f12.2, 4x,
     1 f12.2)
99992 format (//4x, 13htotal cost = , g13.6)
99991 format (1h1//8x, 4hnode, 4x, 14hslack activity, 4x, 10hdual activ,
     1 3hity)
99990 format (4x, a8, 4x, i14, 4x, f13.2)
      end
```

```
      subroutine iden(noden, n, nam, a, narc)
      integer a(2,1000)
      real *8 noden(1),nam(1)
c set from-  and to-node numbers for arcs
      ic = 0
      do 30 i=1,n
         if (ic.eq.1) go to 10
         if (nam(1).ne.noden(i)) go to 10
         ic = ic + 1
         a(2,narc) = i
         go to 20
   10    if (ic.eq.2) go to 30
         if (nam(2).ne.noden(i)) go to 30
         ic = ic + 2
         a(1,narc) = i
   20    if (ic.eq.3) go to 40
   30 continue
      narc = narc - 1
      write (6,99999) nam(1), nam(2)
   40 return
99999 format (1x, 21hundefined node-names:, 2(2x, a8))
      end
```

```
      subroutine order(x, b, nrow, nnl, narcl)
      implicit integer (a-z)
      dimension x(1), b(1)
c reorder basic variables
         do 20 i=1,nrow-1
            do 10 j=i+1,nrow
            if (iabs(b(i)).lt.iabs(b(j))) go to 10
            bb = b(i)
            b(i) = b(j)
            b(j) = bb
            bb = x(i)
            x(i) = x(j)
            x(j) = bb
   10    continue
         if (nnl.ne.0) go to 20
         if (iabs(b(i)).le.narcl) go to 20
         nnl = i - 1
   20 continue
      if (nnl.ne.0) go to 30
      nnl = nrow
      if (b(nrow).gt.narcl) nnl = nnl - 1
   30 return
      end
```

```
      subroutine slex(a, cost, ub, pii, x, b, w, nnode, narc, niter,
     1 ind, colin, av, alfa, phase, max)
c***********************************************************************
c
c     symbol description
c     ==================
c
c     narc = number of arcs including slacks and artificials
c     nnode = number of nodes excluding dummy node
c             ( n in the main program )
c     a(k,j) = to - node for arc j ; for k=1
c              from - node for arc j ; for k=2
c              0 if the respective node is dummy
c     cost(j) = cost coefficient for arc j
c               0 for slack arcs
c               max for artificial arc j , where
c     max = 2*nnode*(maximum cost coefficient) + 1
c     ub(j) = upper bound for arc j if variable j is currently on its
c                   lower bound or is basic
c            -(upper bound for arc j) if variable j is currently on its
c                   upper bound
c             0 if arc j does not have an upper bound
c     phase  = 1  for phase I
c              2  for phase II
c     iternr = current iteration number
c     niter = maximum allowable number of iterations
c                   given by the user
c     b(.) = current list of basic arcs: the ith arc in the basis is b(i).
c            the sign of b(i) is the sign of the ith diagonal element
c            of the triangularized basis
c     x(.) = current values for basic arcs
c     w(.) = row permutation which lower triangularizes the basis
c            when the column permutation is given by b(.)
c     pii(.) = vector of simplex multipliers
c     redc = reduced cost for an arc
c     rc = reduced cost for the arc entering basis
c     colin = arc entering basis
c     alfa(.) = alfa-column of entering arc: basis inverse * column of
c                    entering arc
c     mini = step size for current simplex iteration
c     colout = column leaving the basis
c     ind = -1 if entering arc is to be decreased
c            1 if entering arc is to be increased
c            2 if optimal solution has been obtained
c            3 if max. allowable iterationnr. has been reached
c            4 if unbounded optimal solution has been detected
c            5 if the problem is found to be infeasible
c
c     input and output parameters
c     ===========================
c
c     the transshipment problem (including slacks and artificials) is
c     defined by arc list a(.), cost vector cost(.), upper bound
c     vector ub(.) as well as by nnode, the number of nodes, and narc,
c     the number of arcs. the maximum allowable number of iterations is
c     given by niter. an initial basic solution has been given by
```

```
c     b(.), w(.) and x(.) as well as by ub(.) indicating arcs at their
c     lower or upper bounds at the initial solution. after executing
c     the subroutine, these parameters define the final solution found
c     with simplex iterations and pii(.) gives the corresponding
c     simplex multipliers. ind indicates the type of this solution.
c
c************************************************************************
      implicit integer (a-z)
      real max, pii(1), cost(1), rc, redc, costi
      integer alfa(1), nnode, w(1), b(1), a(2,1), colin, mini, colout,
     1 x(1), iternr, di, nf, ns, begin, ub(1), av(2,1)
      ind = 0
      max = max - 1.
      iternr = -1
c compute price vector pii
   10 k = 2
      do 40 ii=1,nnode
         i = nnode - ii + 1
         call eb(b(i), mja, n)
         pii(i) = cost(mja)
         if (phase.eq.2) go to 30
         pii(i) = 0.
         if (x(i).ge.0) go to 20
         pii(i) = -1.
         k = 1
         go to 30
   20    if (ub(mja).eq.0 .or. x(i).le.ub(mja)) go to 30
         k = 1
         pii(i) = 1.
   30    ed = 3 - n
         if (ed.eq.1) pii(i) = -pii(i)
         if (a(ed,mja).eq.0) go to 40
         j = a(ed,mja)
         j = w(j)
         pii(i) = pii(i) + pii(j)
   40 continue
c did the phase change ?
      if (k.eq.phase) go to 50
      phase = k
      go to 10
   50 if (ind.eq.2) go to 140
      iternr = iternr + 1
   60 ind = 1
c maximum allowable number of iterations reached
      if (iternr.le.niter) go to 70
      ind = 3
      return
c compute reduced cost and choose entering variable
   70 rc = 0.
      do 120 i=1,narc
         costi = cost(i)
         if (costi.ge.max) go to 120
         y1 = a(1,i)
         y2 = a(2,i)
         if (phase.eq.1) costi = 0.
         if (y1.eq.0) go to 80
```

```
        if (y2.eq.0) go to 90
        y1 = w(y1)
        y2 = w(y2)
        redc = costi - pii(y1) + pii(y2)
        go to 100
 80     y2 = w(y2)
        redc = costi + pii(y2)
        go to 100
 90     y1 = w(y1)
        redc = costi - pii(y1)
100     if (redc.lt.rc .and. ub(i).ge.0) go to 110
        if (redc.le.-rc .or. ub(i).ge.0) go to 120
        ind = -1
        rc = ind*redc
        colin = i
        go to 120
110     ind = 1
        rc = redc
        colin = i
120  continue
     if (rc.lt.-1.e-10) go to 150
c optimal solution or infeasibility has been detected
     do 130 i=1,nnode
        call eb(b(i), ib, n)
        if (cost(ib).lt.max) go to 130
        if (x(i).gt.0) ind = 5
        if (ind.ne.5) ind = 2
        cost(ib) = 0.
130  continue
     if (ind.eq.2) go to 10
     if (ind.eq.5) go to 140
     ind = 2
140  niter = iternr
     if (phase.eq.1) ind = 5
     return
c compute alfa column = (basis inverse)*(entering column)
150  do 160 j=1,nnode
        av(1,j) = 0
        av(2,j) = 0
        alfa(j) = 0
160  continue
     i = a(1,colin)
     j = a(2,colin)
     begin = nnode
     if (i.ne.0) begin = w(i)
     if (j.eq.0) go to 170
     j = w(j)
     if (j.lt.begin) begin = j
170  t = 1
180  ia = a(t,colin)
     if (ia.eq.0) go to 200
     ia = w(ia)
     call eb(b(ia), ib, n)
     ic = 3 - t - t
190  id = ic
     n3 = 3 - n
```

```
      if (n3.eq.1) id = -ic
      av(t,ia) = id
      alfa(ia) = alfa(ia) + id
      if (a(n3,ib).eq.0) go to 200
      ia = a(n3,ib)
      ia = w(ia)
      call eb(b(ia), ib, n)
      go to 190
  200 t = t + 1
      if (t.le.2) go to 180
c carry out minimum ratio test for determining step
c size mini and leaving column colout
      mini = 30000
      do 280 i=1,nnode
         izz = 31000
         pivot = alfa(i)
         izx = ind*pivot
         call eb(b(i), j, n)
         if (izx) 220, 280, 210
  210    izz = x(i)
         go to 230
  220    izz = ub(j) - x(i)
  230    if (izz) 280, 240, 250
  240    if (izx.lt.0 .and. x(i).eq.0) go to 280
  250    if (izz-mini) 270, 260, 280
  260    if (cost(j).lt.max) go to 280
  270    int = izx
         mini = izz
         colout = i
         if (mini.eq.0) go to 320
  280 continue
c test whether the bound on entering arc determines
c the step size
      if (ub(colin).eq.0 .or. ind*ub(colin).gt.mini) go to 290
      call eb(b(colout), kk, 11)
      if (cost(kk).ge.max .and. ind*ub(colin).eq.mini) go to 290
      mini = ind*ub(colin)
      ub(colin) = -ub(colin)
      colout = 0
  290 if (mini.ne.30000) go to 300
c unbounded optimal solution has been detected
      ind = 4
      niter = iternr
      return
c update solution vector x(.)
  300 mini = mini*ind
      do 310 i=1,nnode
         if (i.ne.colout) x(i) = x(i) - alfa(i)*mini
  310 continue
c test whether the basis has to be changed
      if (colout.eq.0) go to 60
  320 if (ind.eq.1) x(colout) = mini
      if (ind.eq.-1) x(colout) = -ub(colin) + mini
      ub(colin) = ub(colin)*ind
      call eb(b(colout), j, n)
      ub(j) = ub(j)*int
```

```
      b(colout) = colin
c start updating
      di = 1
      if (av(1,colout).eq.0) di = 2
      nf = 0
      ns = -1
      do 330 j=begin,colout
         if (av(di,j).ne.0) ns = ns + 1
         if (av(3-di,j).ne.0) nf = nf + 1
  330 continue
c retriangularize the basis
      av(1,colout-nf) = colin*(3-2*di)
      av(2,colout-nf) = x(colout)
      if (nf.eq.0) go to 350
      ib = colin
      in = di
      do 340 i=1,nf
         j = a(3-in,ib)
         j = w(j)
         call eb(b(j), ib, in)
         av(1,colout-nf+i) = b(j)
         av(2,colout-nf+i) = x(j)
  340 continue
  350 if (ns.le.0) go to 370
      ib = colin
      in = di
      do 360 i=1,ns
         j = a(in,ib)
         j = w(j)
         call eb(b(j), ib, n)
         in = 3 - n
         av(1,colout-nf-i) = -b(j)
         av(2,colout-nf-i) = x(j)
  360 continue
  370 in = 0
      do 380 i=begin,colout
         if (alfa(i).ne.0) go to 380
         av(1,begin+in) = b(i)
         av(2,begin+in) = x(i)
         in = in + 1
  380 continue
      do 390 i=begin,colout
         b(i) = av(1,i)
         x(i) = av(2,i)
         call eb(b(i), ib, n)
         j = a(n,ib)
         w(j) = i
  390 continue
c end of simplex iteration
      go to 10
      end
```

```
      subroutine eb(ibi, ib, in)
      if (ibi.gt.0) go to 10
      ib = -ibi
      in = 2
      return
   10 ib = ibi
      in = 1
      return
      end
```