```
    end i;
    if l > 1 then NIK[l-1] := 0;
    for j := l step 1 until k1 do
      NIK[j] := Q[k, j];
    l2 := if k + l - 1 > n - 1 then n - 1 else k + l - 1;
    for j:=k step 1 until l2 do
      NIK[j] := Q[k, j] × XX[j-k+1, k];
  end ENDBASIS;
  procedure BANDET (A, B, INT, n, m);
    value n, m; integer n, m; array A, B; integer array INT;
  comment 23. BANDET decomposes the 2m + 1 banded n × n
    matrix A in an upper triangular matrix A and a lower triangular
    matrix B using Gaussian elimination with complete pivoting. De-
    tails of the interchanges are stored in the array INT. The arrays are
    dimensioned as follows A[1:n, -m:m], B[1:n, 1:m], INT[1:n].
    For further details see [5];
  begin
    integer i, j, k, l; real x;
    l := m;
    for i := 1 step 1 until m do
    begin
      for j := 1 - i step 1 until m do
        A[i, j-l] := A[i, j];
      l := l - 1;
      for j := m - l step 1 until m do
        A[i, j] := 0
    end i;
    l := m;
    for k := 1 step 1 until n do
    begin
      x := A[k, -m]; i := k;
      if l < n then l := l + 1;
      for j := k + 1 step 1 until l do
        if abs(A[j, -m]) > abs(x) then
        begin x := A[j, -m]; i := j end;
      INT[k] := i;
      if i ≠ k then
      for j := -m step 1 until m do
      begin
        x := A[k, j]; A[k, j] := A[i, j]; A[i, j] := x
      end j;
      for i := k + 1 step 1 until l do
      begin
        x := A[i, -m]/A[k, -m]; B[k, i-k] := x;
        for j := 1 - m step 1 until m do
          A[i, j-1] := A[i, j] - x × A[k, j];
        A[i, m] := 0
      end i
    end k
  end BANDET;
  procedure BANSOL (A, B, C, INT, n, m);
    value n, m; integer n, m; array A, B, C; integer array INT;
  comment 24. The parameters A, B, INT, n, and m come from
    BANDET. BANSOL solves the system decomposed by BANDET
    with right-hand side C. The solution is returned in {C[i]}1n (see
    [5]);
  begin
    integer i, j, k, l; real x;
    l := m;
    for k := 1 step 1 until n do
    begin
      i := INT [k];
      if i ≠ k then
      begin x := C[k]; C[k] := C[i]; C[i] := x end;
      if l < n then l := l + 1;
      for i := k + 1 step 1 until l do
        C[i] := C[i] - B[k, i - k] × C[k]
    end k;
    l := -m;
    for i := n step -1 until 1 do
    begin
      x := C[i]; j := i + m;
      for k := 1 - m step 1 until l do
        x := x - A[i, k] × C[k + j];
      C[i] := x/A[i, -m];
      if l < m then l := l + 1
    end i
  end BANSOL;
```

Acknowledgment. We wish to thank Harold Eidson for useful suggestions and for checking the algorithm. The referees were also very helpful.

References

1. de Boor, C. On calculating with B-splines. *J. Approx. Th. 6* (1972), 50–62.
2. Lyche, Tom, and Schumaker, Larry L. Computation of smoothing and interpolating natural splines via local bases. *SIAM J. Numer. Anal. 10* (1973), 1027–1038.
3. Reinsch, C.H. Smoothing by spline functions. *Numer. Math. 10* (1967), 177–183.
4. Reinsch, C.H. Smoothing by spline functions, II. *Numer. Math. 16* (1971), 451–454.
5. Martin, R.S., and Wilkinson, J.H. Solution of symmetric and unsymmetric band equations and the calculation of eigenvectors of band matrices. *Numer. Math. 9* (1967), 279–301.
6. Woodford, C.H. An algorithm for data smoothing using spline functions. *BIT 10* (1971), 501–510.

Algorithm 481

Arrow to Precedence Network Transformation [H]

Keith C. Crandall [Recd. 15 Jan. 1973]
Department of Civil Engineering, University of California, Berkeley, CA 94704

Key Words and Phrases: critical path, networks, precedence networks
CR Categories: 3.23, 5.40
Language: Fortran

Description

Purpose. Many of the recent application programs in the area of critical path scheduling and resource allocation are written for the precedence networking convention [1, 2, 3]. Since only a few of these programs accept networks defined by the arrow convention directly, a method of transforming arrow convention networks into precedence convention is required. This algorithm generates the required transformation by producing a list of followers for each non-dummy arrow activity. New labels are produced for each transformed activity and replace the $(i - j)$ labels associated with arrow networks. (The new label is actually the activity input sequence value, but this can easily be modified to any desired notation by using the input sequence value as a subscript to any array containing the desired notation.)

The logic used in the transformation can also be utilized to produce a list of precedecessors if they are desirable. (This order is required by IBM [3] but is performed internally.) The role of arrays (*II* and *JJ*) would be reversed and the array (*ILOC*) would refer to (*JJ*) vice (*II*).

Method. The values of the arrow $(i - j)$ labels are utilized to trace the followers of a particular activity. Activities which have an (i) label corresponding to the (*j*) label of the activity under evalua-

tion are logical followers. The major problems rest with the arrow *DUMMY* activities. These activities are not really followers but indicate instead addition nodes that precede logical followers. The transformation routine recursively traces all possible following nodes and determines the input sequence number of all logic followers.

To perform this search with the minimum storage required the following procedure is utilized. First the arrays (*II, JJ, NLOC*) are filled by scanning the description of the arrow network and storing in input order the converted value of the (*i*) label into array (*II*); the converted value of the (*j*) label into array (*JJ*); and finally the array (*NLOC*) contains the input sequence value. To aid in determining which activities were dummies, the last two arrays (*JJ, NLOC*) have their values set negative when the corresponding activity was a dummy. Since the minimization of storage was a goal, all incoming $(i - j)$ labels were converted into a numerical sequence starting with one. The algorithm indicates the required modification if this is undesirable. (The actual conversion method is described in the routine *HASH*.) Once the arrays are filled, the transformation routine can be called.

Routine (*TRNFRM*) first sorts the array (*II*) into ascending order, maintaining the same correspondence of each element in array (*NLOC*). A sequential scan is then performed on the sorted array (*II*), and the array is overlayed by an array, (*ILOC*), containing pointers to the beginning of each different (*i*) value in the sorted array. That is element (1) of the new array points to the start of the value (1) in the sorted array; element (10) to the start of (10), and so forth. Finally the array (*JJ*) is scanned sequentially and the nonnegative values become subscripts to the pointer array (*ILOC*). This yields the beginning location and number of activities that had an (*i*) label equal to the current (*j*) value. The values stored in (*NLOC*) are the input sequence numbers of the followers. If the follower was a *DUMMY*, (*NLOC*) negative, a recursive search is performed for additional followers.

Finally for each nonnegative entry in (*JJ*), the description is retrieved from the scratch tape and the activity and its followers are output.

Test Results. Testing was performed by two additional programs which are also included in the algorithm listing in case they are desired. Routine (*TEST*) reads the arrow network filling the arrays (*II, JJ, NLOC*) as described. Routine (*HASH*) performs the required conversion to the $(i - j)$ labels during this process.

Tests include networks with sequential dummies and other unusual conditions. In each case tried, the transformation was correct. The inefficiency of the bubble up sort could adversely affect very large networks and an alternative would be to pre-sort the arrow network and eliminate the sorting portion. The following table indicates execution time versus number of activities for tests run on a CDC 6400.

Execution Times for Various Networks Tested

Number of activities	Execution time in sec.
16	0.42
44	1.68
177	2.08
461	5.81
677	10.76

The routine does not test for logical errors in the arrow network such as loops, so these would be transformed without change into the precedence notation.

References

1. Fondahl, John W. A non-computer approach to the critical path method. Tech. Rep. No. 9, Dep. of Civil Engineering, Stanford U., Stanford, Calif., 1962.
2. Baker, Wilson C. Spread and level CPM. Tech. Rep. No. 56, Dep. of Civil Engineering, Stanford U., Stanford, Calif., 1967.
3. IBM, Project Management System. Application description manual (H20-0210), 1968.

Algorithm

(Note: A sample driver is included to help clarify the use of this algorithm—L.D.F.)

```
C THIS IS THE TEST PROGRAM FOR THE TRANSFORMATION ALGORITHM.
C IT READS THE ARROW NETWORK DESCRIPTION AND ESTABLISHES
C THE INPUT ARRAYS FOR THE ROUTINE (TRNFRM).
C IT IS LIMITED TO 700 ACTIVITIES IN ARROW NOTATION.
C THE ROUTINE (HASH) IS UTILIZED TO CREATE A SEQUENTIAL
C NUMBERING.
C THE ROUTINE (TRNFRM) CREATES THE ACTUAL TRANSFORMATION.
C TAPE(2) -A BINARY SCRATCH TAPE (FILE) WITH ALL DATA TO
C BE INCLUDED WITH THE TRANSFORMED ACTIVITIES.NOTE- CHANGE
C STMT 140 TO CORRESPOND WITH ACTUAL DATA STORED.
C TAPE(4) -A BINARY SCRATCH TAPE FOR TRANSFERING THE TRANS-
C FORMED DATA BACK TO THE MAIN PROGRAM FOR PRINT OUT, OR ANY
C OTHER USE. THE DATA IS IN THE FORM (I,M,FOL) WHERE I IS
C THE NEW ACTIVITY LABEL AND M IS THE NUMBER OF FOLLOWERS
C AND FOL IS AN ARRAY CONTAINING THE LABELS OF THE M
C FOLLOWERS...
      INTEGER II(700), JJ(700), NLOC(700), ACT(2), DUMMY,
     * HASH, FOL(50)
      DATA DUMMY/5HDUMMY/, IBLNK/1H /
C READ IN ARROW ACTIVITIES ACCORDING TO CURRENT FORMAT.
99999 FORMAT(1H1, 13H  INPUT ORDER, 6X, 5HLABEL, 5X, 4HDESC,
     * 7HRIPTION, 12X, 3HDUR)
99998 FORMAT(2A4, 2A10, I3, 3X, I6)
99997 FORMAT(I14, 4X, A4, 1H-, A4, 3X, 2A10, I6)
99996 FORMAT(1H1, 19HTRANSFORMED NETWORK//14H  LABEL  DESCR,
     * 6HIPTION, 10X, 3HDUR, 3X, 9HFOLLOWERS)
99995 FORMAT(1H , I7, 2X, 2A10, I4)
99994 FORMAT(1H+, 36X, 15I5/(37X, 15I5))
      WRITE (6,99999)
      NACT = 0
      NTAPE2 = 0
   10 READ (5,99998) I, J, ACT, IDUR
C FORMAT (99998) WILL VARY FOR INDIVIDUAL NEEDS.
C THE TEST FOR END OF DATA IS A BLANK CARD.
      IF (I.EQ.IBLNK) GO TO 30
      NACT = NACT + 1
C LIST THE ARROW DATA FOR REFERENCE.
      WRITE (6,99997) NACT, I, J, ACT, IDUR
C CONVERT THE ALPHANUMERIC I-J LABELS INTO SEQUENTIAL
C NUMERIC. (ROUTINE HASH PERFORMS THIS TASK.)
C STORE THE CONVERTED LABELS IN THE ARRAYS (II AND JJ).
C NOTE. THE VALUE STORED IN ARRAY (JJ) IS ALSO SAVED AS
C VARIABLE J TO ALLOW IT TO BE USED AT STMT 20 WITHOUT AN
C ARRAY REFERENCE.
      II(NACT) = HASH(I)
      J = HASH(J)
      JJ(NACT) = J
C STORE THE INCOMING INPUT SEQUENCE VALUE IN ARRAY (NLOC)
      NLOC(NACT) = NACT
C EXAMPLE OF USER CREATED LABELING, SEE ALSO COMMENTS AFTER
C STMT 140 IN ROUTINE TRNFRM.
C LABLS(NACT)=CONCATENATION OF INPUT (I-J)
C THE CONCATENATION IS PERFORMED IN ACCORDANCE WITH VALID
C FORTRAN FOR THE COMPILER IN USE.
C TEST FOR A DUMMY ACTIVITY AS IT WILL NOT BE TRANSFORMED.
      IF (ACT(1).EQ.DUMMY) GO TO 20
C SAVE ON TAPE (2) ALL INFORMATION RELATING TO THE ACTIVITY
C JUST READ THAT IS TO BE ASSOCIATED WITH THE TRANSFORMED
C ACTIVITY.(FOR THE EXAMPLES ONLY THE DESCRIPTION AND DUR
C ARE SAVED,ACTUAL USERS WILL HAVE INDIVIDUAL REQUIREMENTS)
      NTAPE2 = NTAPE2 + 1
      WRITE (2) ACT, IDUR
      GO TO 10
C IF AN ACTIVITY WAS A DUMMY, SO NOTE BY SETTING THE
C LOCATION AND JJ LABEL VECTORS NEGATIVE.
   20 NLOC(NACT) = -NACT
      JJ(NACT) = -J
C RETURN FOR NEXT INPUT ACTIVITY. TRANSFER WILL BE MADE TO
C STMT 30 WHEN LAST INPUT IS RECOGNIZED.
      GO TO 10
   30 REWIND 2
C CALL THE TRANSFORMATION ROUTINE.,DESCRIPTION OF INPUT
C ARRAYS IS FOUND IN THE (TRNFRM) ROUTINE.
      CALL TRNFRM(NACT, II, JJ, NLOC)
C PRINT OUT THE TRANSFORMED NETWORK...
      WRITE (6,99996)
      DO 40 N=1,NTAPE2
C RECOVER THE REQUIRED DATA RELATING TO THE TRANSFORMED
C ACTIVITY FROM TAPE(2) AND TAPE (4).
        READ (2) ACT, IDUR
        READ (4) I, M, FOL
        WRITE (6,99995) I, ACT, IDUR
        IF (M.LE.0) GO TO 40
        WRITE (6,99994) (FOL(MM),MM=1,M)
   40 CONTINUE
      STOP
      END



      INTEGER FUNCTION HASH(N)
C THIS ROUTINE CONVERTS THE ALPHANUMERIC ARROW LABELS INTO A
C SEQUENTIAL NUMERIC EQUIVALENT. THE MAXIMUM NUMBER OF
C SEPARATE ACTIVITY LABELS IS 500 FOR THIS TEST PACKAGE.
C THE ACTUAL INCOMING LABEL IS STORED IN ARRAY (HOLD) AND
C THE SEQUENTIAL NUMERIC EQUIVALENT IS STORED IN ARRAY
C  (SAVE)
C VARIABLE (NUM) PROVIDES THE SEQUENTIAL NUMBERS.
      INTEGER HOLD(500), SAVE(500)
```

```
      DATA NUM/0/, HOLD/500*0/
C USE A MODIFIED HASHING ROUTINE TO FIND AND STORE THE
C EQUIVALENT VALUES.
C NN IS A HASHED VALUE FOR THE INPUT VARIABLE N.
99999 FORMAT(34H EXCEEDED THE EVENT TABLE CAPACITY)
      NN = MOD(IABS(N/68719476736),375)
   10 DO 20 I=NN,500
C THE ARRAY (HOLD) IS EXAMINED STARTING WITH THE HASHED
C VALUE, IF THE ARRAY ELEMENT CONTAINS THE INPUT VARIABLE N,
C TRANSFER IS MADE TO STMT 40 AND THE EQUIVALENT SEQUENTIAL
C NUMBER IS RECALLED FROM ARRAY (SAVE). IF THE ARRAY ELEMENT
C CONTAINS A ZERO,TRANSFER IS MADE TO STMT 30 AND A
C  NUMERICAL
C EQUIVALENT IS ASSIGNED. THE SEARCH OF (HOLD) CONTINUES
C UNTIL AN OPEN ELEMENT IS FOUND...
        IF (HOLD(I).EQ.N) GO TO 40
        IF (HOLD(I).EQ.0) GO TO 30
   20 CONTINUE
C IF NO OPEN ELEMENT WAS FOUND AND NN=1 THERE ARE NO OPEN
C ELEMENTS IN THE ENTIRE ARRAY. IF NN IS NOT EQUAL TO 1, SET
C IT TO 1 AND SEARCH LOWER PART OF (HOLD)...
      IF (NN.EQ.1) GO TO 60
      NN = 1
      GO TO 10
C FOUND A NEW LABEL-GIVE IT AN EQUIVALENT SEQUENTIAL NUMBER
   30 HOLD(I) = N
      NUM = NUM + 1
      IVAL = NUM
      SAVE(I) = IVAL
C TRANSFER TO STMT 50 AND SAVE A REDUNDANT RECALL FROM
C  (SAVE)
      GO TO 50
   40 IVAL = SAVE(I)
   50 HASH = IVAL
      RETURN
C AN ERROR MESSAGE IS GENERATED IF THE NUMBER OF EVENTS
C EXCEEDS THE DIMENSION ALLOWED.
   60 WRITE (6,99999)
      STOP
      END



      SUBROUTINE TRNFRM(NACT, II, JJ, NLOC)
C ALL DATA WAS STORED IN THE ARRAYS (II-JJ-NLOC) BY THE
C CALLING ROUTINE AND COMFORMS TO THE FOLLOWING DESCRIPTION
C (NACT) -THE  NUMBER OF ARROW ACTIVITIES INCLUDING DUMMIES.
C (II) -AN ARRAY OF CONVERTED -I- LABELS STORED IN THE ARROW
C NETWORK INPUT ORDER.REFER TO THE COMMENTS AFTER STMT 140
C IF USER GERERATED LABELS ARE DESIRED.SEE ALSO COMMENTS IN
C MAIN ROUTINE.
C (JJ) -AN ARRAY LIKE (II) FOR -J- LABELS EXCEPT THAT THE
C VALUE IS NEGATIVE FOR ALL DUMMY ACTIVITIES.
C (NLOC) -AN ARRAY INDICATING INPUT ORDER.(A SEQUENTIAL LIST
C SUCH THAT THE ABSOLUTE VALUES WOULD RANGE FROM ONE TO NACT
C )  NOTE- THE VALUE STORED IN (NLOC) IS NEGATIVE WHEN THE
C CORRESPONDING ARROW ACTIVITY WAS A -DUMMY- .
C TAPE(4) -A BINARY SCRATCH TAPE FOR TRANSFERING THE TRANS-
C FORMED DATA BACK TO THE MAIN PROGRAM FOR PRINT OUT, OR ANY
C OTHER USE. THE DATA IS IN THE FORM (I,M,FOL) WHERE I IS
C THE NEW ACTIVITY LABEL AND M IS THE NUMBER OF FOLLOWERS
C AND FOL IS AN ARRAY CONTAINING THE LABELS OF THE M
C FOLLOWERS...
C STORAGE FOR THE ARRAYS IS ALSO SPECIFIED IN THE CALLING
C PROGRAM.
      INTEGER II(1), JJ(1), NLOC(1)
      INTEGER STACK(50), FOL(50)
C THE DIMENSION STAMENTS FOR (II-JJ-NLOC) MUST BE MODIFIED
C FOR USE WITH SOME FORTRAN COMPILERS.
C DIMENSIONS ON STACK AND FOL LIMIT THE NUMBER OF FOLLOWING
C ACTIVITIES TO 50.
C STATEMENT FUNCTION TO PROVIDE OVERLAYING ARRAY (II) WITH
C ARRAY (ILOC).REFER TO THE WARNING AFTER STMT 30,IF A
C SEPERATE ARRAY (ILOC) IS UTILIZED THE STATEMENT FUNCTION
C WOULD BE DELETED.
99999 FORMAT(41H THE FOLLOWING ACTIVITY APPEARS TO HAVE M,
     * 22HORE THAN 50 FOLLOWERS )
99998 FORMAT(41H SUSPECT THE FOLLOWING ACTIVITY IS INVOLV,
     * 41HED IN A NETWORK LOOP - CHECK INPUT DATA. /I5)
      ILOC(I) = II(I)
C REWIND TAPE 4 FOR TRANSFER OF TRANSFORMED DATA.
      REWIND 4
C PLACE THE ARRAYS (II-NLOC) IN ASENDING ORDER USING (II)
C AS THE SORT VARIABLE. (THIS IS A BUBBLE UP SORT.)
      LIMIT = NACT - 1
      DO 20 M=1,LIMIT
        LL = M + 1
        DO 10 N=LL,NACT
          IF (II(M).LE.II(N)) GO TO 10
          IHOLD = II(N)
          II(N) = II(M)
          II(M) = IHOLD
          IHOLD = NLOC(N)
          NLOC(N) = NLOC(M)
          NLOC(M) = IHOLD
   10   CONTINUE
   20 CONTINUE
C REPLACE THE ARRAY (II) WITH AN INTEGER POINTER SUCH THAT
C THE (K TH) ELEMENT OF THE POINTER POINTS TO THE FIRST
C LOCATION IN THE SORTED ARRAY (II) WHICH CONTAINS THE VALUE
C (K).THE POINTER ARRAY WILL BE CALLED (ILOC) SINCE IT
C INDICATES THE BEGINNING OF SORTED ARROW NODES (ARRAY II)
C AND THESE NODES ARE NORMALLY REFERRED TO AS (I) NODES.
C THE VARIABLE (N) IS SET TO THE MINIMUM VALUE IN ARRAY (II)
C N IS ALSO A VARIABLE THAT INDICATES THE CURRENT VALUE
C UNDER INVESTIGATION IN ARRAY (II).
C L IS A POINTER TO THE ARRAY (ILOC),INDICATING THE LOCATION
C OF THE NEXT ELEMENT.IN ADDITION L ALSO INDICATES THE NEXT
C SEQUENTIAL NUMBER,AND IS USED TO FIND THE END NODES.(NODES
C WHERE THERE EXISTS NO -I- IN THE (I-J) PAIRS,AND THERE-
C FORE NO ENTRY IN THE SORTED (II) ARRAY..)
      N = 1
      L = 2
      DO 50 I=2,NACT
        IF (II(I).EQ.N) GO TO 50
        N = II(I)
   30   IF (N.EQ.L) GO TO 40
C THIS TEST FINDS THE REFERENCES TO THE END NODE WHICH WILL
C NOT BE IN THE SORTED ARRAY OF (I) NODES.
C WARNING -- ALTHOUGH INPUT ORDER IS NOT NORMALLY IMPORTANT
C REFERENCE TO END NODES,THAT IS (I-J) PAIRS WITH -J- EQUAL
C TO AN END NODE,SHOULD BE POSITIONED IN THE LATER PORTION
C OF THE INPUT DATA.THIS RESTRICTION CAN BE ELIMINATED BY
C USING A SEPARATE ARRAY FOR (ILOC).
C II(L) IS SET TO ZERO TO INDICATE THAT NODE -L- IS AN END
C NODE IN THE ARROW INPUT NETWORK.
        II(L) = 0
        L = L + 1
        GO TO 30
C STORE THE SUBSCRIPT VALUE OF THE ARRAY (II) IN TO THE
C OVERLAYED ARRAY (ILOC).
   40   II(L) = I
        L = L + 1
   50 CONTINUE
C SET THE NEXT LOCATION OF THE POINTER TO ONE PAST THE LAST
C ACTIVITY NUMBER.
      MAXLST = L - 1
      II(L) = NACT + 1
C FOR ALL NON DUMMY ACTIVITIES,TRANSFORM THE ARROW LOGIC
C CONSTRAINTS INTO THE PRECEDENCE NOTATION BY GIVING THE
C ACTIVITY A LABEL EQUAL TO ITS INPUT ORDER,THEN LIST ALL
C TRANSFORMED FOLLOWERS.
      DO 160 I=1,NACT
        L = 0
        M = 0
C L INDICATES THE LENGTH OF THE STACK AND M IS THE NUMBER OF
C FOLLOWERS.THE STACK IS USED TO RECURSIVELY TRACE ALL
C DUMMIES TO FIND LOGICAL FOLLOWERS.
        N = JJ(I)
C IF N IS NEGATIVE THE ARROW ACTIVITY WAS A DUMMY.
        IF (N.LE.0) GO TO 160
   60   LOC = N
        IF (LOC.GT.MAXLST) GO TO 110
C LOC HAS A VALUE EQUAL TO THE -J- LABEL OF ACTIVITY UNDER
C TRANSFORMATION. ILOCR POINTS TO THE BEGINNING OF THAT SAME
C VALUE IN THE SORTED ARRAY (II).WHEN (LOC) EXCEEDS THE
C VALUE OF (MAXLST) THE -J- LABEL ON THE ARROW NETWORK WAS
C THE END NODE,THEREFORE THERE ARE NO FOLLOWERS.
        ILOCR = ILOC(LOC)
        IF (ILOCR.LE.0) GO TO 110
C IF ILOCR IS NEG OR ZERO THE ACTIVITY HAS NO FOLLOWERS.
   70   LOC = LOC + 1
        NN = ILOC(LOC) - ILOCR
C NN INDICATES THE NUMBER OF ELEMENTS IN ARRAY (II) WITH THE
C VALUE.
        IF (NN.LE.0) GO TO 70
        DO 100 LOOP=1,NN
          LOCS = NLOC(ILOCR)
          IF (LOCS.EQ.0) GO TO 90
          IF (LOCS.GT.0) GO TO 80
C LOCS NEGATIVE INDICATES A DUMMY AND THESE ARE HELD IN THE
C STACK FOR LATER CONTINUED SEARCH OF FOLLOWERS.
          L = L + 1
          IF (L.GT.50) GO TO 130
          STACK(L) = -LOCS
          GO TO 90
   80     M = M + 1
C A FOLLOWER HAS BEEN FOUND.STORE IT IN THE ARRAY (FOL).
          IF (M.GT.50) GO TO 120
          FOL(M) = LOCS
C INCREASE THE POINTER TO NEXT POTENTIAL FOLLOWER.
   90     ILOCR = ILOCR + 1
  100   CONTINUE
  110   IF (L.LE.0) GO TO 140
C IF (L) IS NON-ZERO,THERE ARE DUMMY LINKAGES TO BE CONSIDER
C ED. (N) WILL INDICATE FIRST OF THESE AND THE SEARCH FOR
C FOLLOWERS WILL CONTINUE.
        K = STACK(L)
        N = IABS(JJ(K))
        L = L - 1
        GO TO 60
C ERROR MESSAGES IF DIMENSIONS EXCEEDED- LOOP ASSUMED.
  120   WRITE (6,99999)
  130   WRITE (6,99998) I
  140   WRITE (4) I, M, FOL
C IF USER LABELS ARE USED THEY WOULD BE RETRIEVED THUSLY --
C       I = LABLS(I)
C       DO 150 LOOP=1,M
C         ISUB = FOL(LOOP)
C.        FOL(LOOP) = LABLS(ISUB)
C 150   CONTINUE
C WHERE LABLS WOULD BE AN ARRAY PASSED IN THE ARGUMENT LIST
  160 CONTINUE
      REWIND 4
      RETURN
      END
```

Algorithm 482

Transitivity Sets [G7]

John McKay and E. Regener* [Recd. 21 May 1973]
School of Computer Science, McGill University, Montreal, Quebec, Canada

Key Words and Phrases: transitivity, sets
CR Categories: 5.39
Language: Algol

Let $P = \{P_1, P_2, \ldots, P_k\}$ be a set of k permutations on the set $\Omega = \{1, 2, \ldots, n\}$. The transitivity set containing i (or orbit of i) under P is the set of images of i under the action of products of elements of P. This procedure computes these orbits.

On entry, *im*[*i*,*j*] is assumed to contain the image of i under P_j, for $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, k$. The procedure numbers the orbits consecutively starting at 1. On exit *ind*[*i*] contains the number of the orbit to which i belongs. The orbits appear in order in *orb*[1:*n*]. In *orb* the first element of each orbit is tagged negative. If only one permutation is input, the array *orb* contains it (tagged) in disjoint cycle form on exit.

The algorithm, which involves no searching, is related to one for finding a spanning tree of a graph [1]. The set P need not, in general, generate a group—it is sufficient that it generate a semigroup on Ω.

References

1. Cannon, J. Ph.D. Th., Sydney U., Sydney, N.S.W., Australia, 1969.

Algorithm

```
procedure orbits (ind, orb, im, n, k);
  value n, k; integer n, k;
  integer array ind, orb, im;
begin
  integer q, r, s, j, nt, ns, norb;
  for j := 1 step 1 until n do ind[j] := 0;
  norb := 0; ns := 1;
  for r := 1 step 1 until n do if ind[r] = 0 then
  begin
    norb := norb + 1; ind[r] := norb;
    nt := ns; orb[ns] := -r; s := r;
a:
    ns := ns + 1;
    for j := 1 step 1 until k do
    begin
      q := im[s,j];
      if ind[q] = 0 then
      begin
        nt := nt + 1; orb[nt] := q; ind[q] := norb
      end
    end;
    if ns ≤ nt then
    begin s := orb[ns]; go to a end
  end
end
```

* Now at Faculté de Musique, University de Montréal, Montréal, P.Q., Canada.

Remark on Algorithm 450 [E4]
Rosenbrock Function Minimization [Marek Machura and Andrzej Mulawa, *Comm. ACM 16* (Aug. 1973), 482–483]

Adhemar Bultheel [Recd. 10 Oct. 1973]
Katholieke Universiteit Leuven, Faculty of Applied Sciences, Applied Math Division, Celestijnenlaan 200 B, B-3030 Heverlee, Belgium

1. Some misprints were found in the listing of the algorithm.
(a) An *E* has to replace the *F* printed in the following statements:
The one preceding the statement labeled 70.
The one following the statement labeled 80.
The one preceding the statement labeled 100.
The one following the statement labeled 100.
(b) The digit 1 should replace the character *I* as the first index of *ALPHA* in the statement preceding the statement labeled 200.
(c) *RETA* should be read *BETA* in the statement preceding the statement labeled 260.

2. Some compilers detect an error in the calling sequence of *MONITR* in the third line following the statement labeled 70 because the fifth argument of *MONITR* is an *INTEGER*-type constant, and in the subroutine *MONITR* the fifth argument stands for the norm *B* of a vector which is obviously a *REAL*-type variable as is also assumed in the other calls of *MONITR*. One way to overcome this difficulty is to replace 0 by any *REAL* constant, say 0.

3. Since it is often useful to have the initial guess and the corresponding function value printed, an additional call to *MONITR* could be inserted just preceding the *COMMENT*
C START OF THE ITERATION LOOP
This statement could be
CALL MONITR (N, X, FO, R, 1.E 10, CON, 0).
The last argument is the monitoring index *NR*. The user of *Romin* should program *MONITR* to handle the initial guess when $NR = 0$ (printing or not, checking for convergence or not, ...). The fifth argument is chosen to be a large constant because it stands for the norm *B* of a vector. The routine *MONITR* will contain a test to see if $B < \epsilon$ with ϵ "small" and chosen by the user. If one wants to check the initial guess for convergence, then the routine would stop when *B* equals 0. .

4. With these corrections and changes the algorithm was successfully used under a WATFIV compiler on the IBM 370-155 computer of the Computing Centre of the University of Leuven. For the example of the parabolic valley function given by the authors of the algorithm and with the same starting point the following results were obtained: in a single-precision version 202 function evaluations were needed to reach $F = 0.299986.10^{-4}$, and in a double-precision version 194 function evaluations to reach $F = 0.297742.10^{-4}$ and 290 function evaluations gave $F = 0.489134.10^{-13}$.

Remark on Algorithm 454 [E4]
The Complex Method for Constrained Optimization [Joel A. Richardson and J.L. Kuester, *Comm. ACM 16* (Aug. 1973), 487–489]

Kenneth D. Shere [Recd. 8 Oct. 1973]
Mathematical Analysis Division, Naval Ordnance Laboratory, Silver Spring, MD 20910

This algorithm can result in an infinite loop. This happens whenever the "corrected point," the centroid of the remaining "complex" points, and every point on the line segment joining these two points all have functional values lower than the functional values at each of the remaining complex points. Two examples for which this algorithm fails are [1] and [2]:

1. maximize $f(x) = -100(x_2-x_1^2)^2 - (1-x_1)^2$
   $-10 \leq x_1, x_2 \leq 10$, initial value $(x_1, x_2) = (-2.5, 5.0)$

and

2. maximize
   $f(\theta, \phi) = 0.2\ (sin\ (\theta_0)\ cos\ (\phi_0)\ sin\ (\theta)\ cos\ (\phi) + sin\ (\theta_0)\ sin\ (\phi_0)\ sin\ (\theta)\ sin\ (\phi) + cos\ (\theta_0)\ cos\ (\theta)) - 1.0\ (sin^2\ (\theta)\ cos^2\ (\theta) + cos^2\ (\phi)\ sin^2\ (\phi)\ sin^4\ (\theta))$
   $0 \leq \theta, \phi \leq \pi/2$, $(\theta_0, \phi_0) = (.8726, .0873)$,
   initial $(\theta, \phi) = (\pi/4, \pi/4)$

Also, there is no difference in usage between *M* and *L*.

A similar method is the "simplex method" [3]. A modification to the "complex method" which uses the ideas of [3] has been programmed. The modified *JCONSX* solves each of the above problems in under 5 CP sec on a CDC 6400. The modified routine is available to interested parties upon request.

It is also worth noting that the variable *IA*, which appears in the second statement after 70 *CONTINUE* is not used elsewhere.

References

1. Rosenbrock, H.H. An automatic method for finding the greatest or least value of a function. *Comput. J. 3* (1960), 175–184.
2. Ferguson, R.E. An electromagnetism problem. (Private communication.)
3. Parkinson, J.M., and Hutchinson, D. An investigation into the efficiency of variants on the simplex method. In *Numerical Methods for Nonlinear Optimization*, F.A. Lootsma, Ed., Academic Press, New York, 1972.

Numerical Mathematics
R.A. Willoughby Editor

Gauss Harmonic Interpolation Formulas

A.H. Stroud
Texas A&M University

Let R be an open, bounded, simply connected region in the (x,y)-plane and let (x_*,y_*) be a point in R. Assuming R is starlike with respect to (x_*,y_*), we discuss a method for computing Gauss harmonic interpolation formulas for R and the point (x_*,y_*). Such formulas approximate a harmonic function at (x_*,y_*) in terms of a linear combination of its values at certain selected points on the boundary of R. Such formulas are useful for approximating the solution of the Dirichlet problem for R.

Key Words and Phrases: interpolation, quadrature, harmonic interpolation, harmonic quadrature, Dirichlet problem

CR Categories: 5.13, 5.16, 5.17

1. Introduction

We consider the two-dimensional Dirichlet problem. This is the problem of finding a function $u(x,y)$ so that

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 \quad \text{in} \quad R, \qquad u(x, y) \text{ given on } B. \quad (1)$$

Here R is an open, bounded, simply connected region in the (x,y)-plane and B is the boundary of R.

If (x_*, y_*) is a given point in R, it is well known (see, for example [7, p. 85]) that

$$u(x_*, y_*) = \int_B w(x_*, y_*; x, y) u(x, y)\, d\sigma \quad \text{where}$$

$$w(x_*, y_*; x, y) \equiv -\frac{\partial G}{\partial n}(x_*, y_*; x, y) \quad (2)$$

is the normal derivative of Green's function for R.

Author's address: College of Science, Department of Mathematics, Texas A&M University, College Station, TX 77843.