```
      FUNCTION PRUN(N,PROB)
      DIMENSION PROB(50), PSUM(50), TERM(50)
C     PROGRAM TO DETERMINE THE PROBABILITY OF THE
                                          UNION OF
C     N(MAXIMUM 50) INDEPENDENT, BUT NOT MUTUALLY
                                          EXCLUSIVE EVENTS
C
C     PRUN=PROBABILITY OF THE UNION
C     N=NUMBER OF EVENTS INCLUDED
C     PROB(I)=PROBABILITY OF THE I-TH EVENT
C     TERM(R)=R-TH TERM IN THE N-TERM EXPRESSION
                                          FOR PRUN
C     PSUM(I)=I-TH OF (N-R+1) PARTIAL SUMS IN TERM(R)
C
C     INITIALIZATION OF VARIABLES
      DO1J=1,N
      PSUM(J)=0.0
    1 TERM(J)=0.0
      TERM(N)=1.0
      PRUN=0.0
C
C     EVALUATION OF PARTIAL SUMS AND TERMS OF PRUN
      DO2J=1,N
      TERM(1)=TERM(1)+PROB(J)
      TERM(N)=TERM(N)*PROB(J)
    2 PSUM(J)=PROB(J)
      IF(N-2)7,7,6
C     EVALUATION OF MIDDLE (N-2) TERMS OF PRUN
    6 I2=N-1
      J2=N
      DO4I=2,I2
      J2=J2-1
      K2=J2+1
      DO4J=1,J2
      PSUM(J)=0.0
      K1=J+1
      DO3K=K1,K2
    3 PSUM(J)=PSUM(J)+PROB(J)*PSUM(K)
    4 TERM(I)=TERM(I)+PSUM(J)
C
C     SUMMATION OF N TERMS OF PRUN
    7 SIGN=-1.0
      DO5J=1,N
      SIGN=-SIGN
    5 PRUN=PRUN+SIGN*TERM(J)
      RETURN
      END
```

Fig. 1

be determined; i.e.

$$P\left(\bigcup_{\alpha=1}^{n} E_\alpha\right) = \sum_{i=1}^{n} P_i + \sum_{r=2}^{n} (-1)^{r-1} \sum_{i=1}^{n-r+1} \sum_{j=i+1}^{n-r+1} P_i S_{r-1,j}\,. \tag{7}$$

The number of arithmetic operations is reduced in (7) to

$$N = 2(n-1) + \sum_{r=2}^{n} \left\{ \left[\sum_{i=1}^{n-r+1} 2(n-r+1) + 1 \right] + (n-r) \right\}; \tag{8}$$

i.e.

$$N = 2(n-1) + \sum_{r=2}^{n} [(n-r+1)^2 + (n-r)]. \tag{9}$$

For n as small as 10 the savings in time and storage space are considerable. For the minimum storage condition 6898 arithmetic operations are required at $n = 10$. By using the algorithm presented here, this can be reduced to 339 operations. At the same time the required storage space is reduced from 252 locations under the minimum time condition to 30 here.

Figure 1 shows a listing of the algorithm coded as a Fortran FUNCTION SUBPROGRAM.

Received November, 1967; revised March, 1968

REFERENCE

1. Wilks, S. S. *Mathematical Statistics.* Wiley, New York, 1962, p. 12.

Algorithms

J. G. HERRIOT, Editor

ALGORITHM 336
NETFLOW [H]

T. A. Bray and C. Witzgall
(Recd. 2 Oct. 1967 and 20 May 1968)
Boeing Scientific Research Laboratories, Seattle, WA 98124

KEY WORDS AND PHRASES: capacitated network, linear programming, minimum-cost flow, network flow, out-of-kilter
CR CATEGORIES: 5.32, 5.41

procedure *NETFLOW* (*nodes*, *arcs*, *I*, *J*, *cost*, *hi*, *lo*, *flow*, *pi*, *INFEAS*);
value *nodes*, *arcs*; **integer** *nodes*, *arcs*;
integer array *I*, *J*, *cost*, *hi*, *lo*, *flow*, *pi*; **label** *INFEAS*;
comment This procedure determines the least-cost flow over an upper and lower bound capacitated flow network.

Each directed network arc a is defined by nodes $I[a]$ and $J[a]$, has upper and lower flow bounds $hi[a]$ and $lo[a]$, and cost per unit of flow $cost[a]$. Costs and flow bounds may be any positive or negative integers. An upper flow bound must be greater than or equal to its corresponding lower flow bound for a feasible solution to exist. There may be any number of parallel arcs connecting any two nodes.

The procedure returns vectors *flow* and *pi*. $flow[a]$ is the computed optimal flow over network arc a. $pi[n]$ is a number—the dual variable—which represents the relative value of injecting one unit of flow into the network of node n. *NETFLOW* may be entered with any values in vectors *flow* and *pi* (such as those from a previous or a guessed solution) feasible or not. If the initial contents of *flow* do not conserve flow at any node, the solution values will also not conserve flow at that node, by the same amount.

This procedure is a revision (see remark by T. A. Bray and C. Witzgall [1]) of Algorithm 248 [2]. Like the original, it follows the out-of-kilter algorithm described by D. R. Fulkerson [3] and elsewhere. It follows the RAND code by R. J. Clasen (Fortran) in three instances, using a single set of labels *na*, which correspond to the *nb* of Algorithm 248, avoiding superfluous tests in the part following *BACK* (for instance, $c > 0 \wedge flow[a] < lo[a]$ is equivalent to $c > 0$ at this point of the program), and taking advantage of the fact that arcs remain in kilter and need not be rechecked again. In addition, the convention $inf = -1$ is adopted in order to permit costs and bounds of value around 99999999 without their interfering with the initiation of minimum search.

References:

1. Bray, T. A., and Witzgall, C. Remark on Algorithm 248, NETFLOW. *Comm. ACM 11* (Sept. 1968), 633.
2. Briggs, William A. Algorithm 248, NETFLOW. *Comm. ACM 8* (Feb. 1965), 103.
3. Fulkerson, D. R. An out-of-kilte method for minimal-cost flow problems. *J. Soc. Ind. Appl. Math. 9* (Mar. 1961), 18–27;

```
begin
  integer a, aok, c, cok, del, eps, inf, lab, m, n, src, snk;
  integer array na[1: nodes];
  integer procedure minp(x,y); value x,y; integer x,y;
  begin
    if x < y ∧ x ≧ 0 then minp := x else minp := y
  end minp;
  comment check feasibility of formulation;
  for a := 1 step 1 until arcs do
    if lo[a] > hi[a] then go to INFEAS;
  inf := −1;
  comment find out-of-kilter arc;
  for aok := 1 step 1 until arcs do
  begin
    cok := cost[aok] + pi[I[aok]] − pi[J[aok]];
TEST: if flow[aok] < lo[aok] ∨ (cok<0∧flow[aok]<hi[aok]) then
    begin
      src := J[aok]; snk := I[aok]; na[src] := + aok;
      go to LABL
    end;
    if flow[aok] > hi[aok] ∨ (cok>0∧flow[aok]>lo[aok]) then
    begin
      src := I[aok]; snk := J[aok]; na[src] := −aok;
      go to LABL
    end;
    comment arc aok is in kilter;
    go to NEXT;
    comment arc aok is out-of-kilter, clear all labels but source
      label, start new labeling;
LABL: for n := 1 step 1 until src − 1, src + 1 step 1 until
  nodes do na[n] := 0;
LOOP: lab := 0;
    comment switch set for determining whether a pass thru
      the list of arcs yields a new label;
    for a := 1 step 1 until arcs do
    begin
      if (na[I[a]]=0∧na[J[a]]=0) ∨ (na[I[a]]≠0∧na[J[a]]≠0) then
        go to XC;
      c := cost[a] + pi[I[a]] − pi[J[a]];
      if na[I[a]] = 0 then go to XA;
      if flow[a] ≧ hi[a] ∨ (flow[a]≧lo[a]∧c>0) then
        go to XC;
      na[J[a]] := +a; go to XB;
XA: if flow[a] ≦ lo[a] ∨ (flow[a] ≤hi[a]∧c<0) then
        go to XC;
      na[I[a]] := −a;
XB: lab := 1;
      comment node labeled, test for breakthru;
      if na[snk] ≠ 0 then go to INCR;
XC: end no breakthru;
    if lab ≠ 0 then go to LOOP;
    comment nonbreakthru, determine change to pi vector;
    del := inf;
    for a := 1 step 1 until arcs do
    begin
      if (na[I[a]]=0∧na[J[a]]=0) ∨ (na[I[a]]≠0∧na[J[a]]≠0) then
        go to XD;
      c := cost[a] + pi[I[a]] − pi[J[a]];
      if na[J[a]] = 0 ∧ flow[a] < hi[a] then
        del := minp(del,c);
      if na[J[a]] ≠ 0 ∧ flow[a] > lo[a] then
        del := minp(del,−c);
XD: end;
    if del = inf then
    begin
      if flow[aok] = hi[aok] ∨ flow[aok] = lo[aok] then
        del := abs(cok)
      else go to INFEAS
    end exit, no feasible flow;
    comment change pi vector by computed del;
    for n := 1 step 1 until nodes do
      if na[n] = 0 then pi[n] := pi[n] + del;
    comment test whether aok is now in kilter;
    if del = abs(cok) ∧ flow[aok] ≥ lo[aok] ∧ flow[aok]
      ≤ hi[aok] then
      go to NEXT;
    cok := cost[aok] + pi[I[aok]] − pi[J[aok]];
    go to LOOP;
    comment breakthru, compute incremental flow;
INCR: eps := inf; n := src;
BACK: a := na[n];
    if a > 0 then
    begin
      m := I[a];
      if cost[a] + pi[m] − pi[n] > 0 then
        eps := minp(eps, lo[a]−flow[a])
      else eps := minp(eps, hi[a]−flow[a])
    end
    else
    begin
      m := J[−a];
      if cost[−a] + pi [n] − pi [m] < 0 then
        eps := minp(eps,flow[−a]−hi[−a])
      else eps := minp(eps,flow[−a]−lo[−a])
    end;
    n := m; if n ≠ src then go to BACK;
    comment change flow by eps;
BACK2: a := na[n];
    if a > 0 then
    begin
      m := I[a]; flow[a] := flow[a] + eps
    end
    else
    begin
      m := J[−a]; flow[−a] := flow[−a] − eps
    end;
    n := m; if n ≠ src then go to BACK2;
    comment test whether aok is now in kilter;
    go to TEST;
NEXT:
  end find next out-of-kilter arc
end NETFLOW with a feasible, optimal flow
```

ALGORITHM 337

CALCULATION OF A POLYNOMIAL AND ITS DERIVATIVE VALUES BY HORNER SCHEME [C1]

W. Pankiewicz (Recd. 28 Mar. 1968 and 16 May 1968)
Warszawa - 1, Al. 3-go Maja 2/68, Poland

KEY WORDS AND PHRASES: function evaluation, polynomial evaluation, Algol procedure, Horner's scheme
CR CATEGORIES: 5.12, 4.22

procedure *horner*(*n,a,k,r,x0,b*); **value** *n,k,x0,b*;
integer *n,k*; **real** *x0*; **Boolean** *b*; **array** *a,r*;
comment If b is **true** the procedure calculates and stores in $r[i]$ the value of

$$d^i(\sum_{j=0}^{n} a[j] \times x \uparrow j)/dx^i$$

and $x = x0$ for $i = 0, 1, \cdots, k$. If b is **false** it calculates and stores in the array r the values of the first $k+1$ coefficients of the expansion of the polynomial in a power series in the neighborhood of $x0$, i.e.

$$\sum_{j=0}^{n} a[j] \times x \uparrow j = \sum_{i=0}^{n} r[i] \times (x-x0) \uparrow i.$$

Here n is the degree of the polynomial whose coefficients are given by $a[0:n]$. It is assumed that $0 \leq k \leq n$. If $k = 0$ only the value of the polynomial is calculated. If b is **false** the choice $k = n$ would be most useful.

This algorithm is essentially equivalent to Algorithm 29 [*Comm. ACM 3* (Nov. 1960), 604] in terms of quantities computed, but the application of Horner's scheme significantly reduces the number of operations.

Example 1. For the polynomial of degree $n = 5$: $w(x) = x \uparrow 5 + 2 \times x \uparrow 4 - 3 \times x \uparrow 3 + 8 \times x \uparrow 2 - 7 \times x + 11$, $k = 2$, $x0 = 2$ and $b =$ **true,** the following was obtained: $r[0] = 69$, $r[1] = 133$, $r[2] = 236$, i.e. $w(2) = 69$, $w'(2) = 133$ and $w''(2) = 236$.

Example 2. For the polynomial of degree $n = 7$: $w(x) = x \uparrow 7 - 7 \times x \uparrow 5 + 6 \times x \uparrow 4 + 4 \times x \uparrow 3 - x \uparrow 2 - 2 \times x - 9$, $k = 7$, $x0 = 2$ and $b =$ **false** the following vector r was obtained: 15, 122, 279, 332, 216, 77, 14, 1, i.e., the given polynomial can be expressed in the form: $w(x) = 15 + 122 \times (x-2) + 279 \times (x-2) \uparrow 2 + 332 \times (x-2) \uparrow 3 + 216 \times (x-2) \uparrow 4 + 77 \times (x-2) \uparrow 5 + 14 \times (x-2) \uparrow 6 + (x-2) \uparrow 7$;

```
begin
  integer i, j, l;  real rr;
  rr := a[0];
  for i := 0 step 1 until k do
    r[i] := rr;
  for j := 1 step 1 until n do
  begin
    r[0] := r[0] × x0 + a[j];
    l := if n − j > k then k else n − j;
    for i := 1 step 1 until l do
      r[i] := r[i] × x0 + r[i−1]
  end;
  if b then
  begin
    l := 1;
    for i := 2 step 1 until k do
    begin
      l := l × i;
      r[i] := r[i] × l
    end
  end
end horner
```

REMARK ON ALGORITHM 248 [H]

NETFLOW [William A. Briggs, *Comm. ACM 8* (Feb. 1965), 103]

J. H. Henderson, R. M. Knapp, and M. E. Volberding
(Recd. 7 Apr. 1966)
Northern Natural Gas Company, Omaha, Neb.

KEY WORDS AND PHRASES: capacitated network, linear programming, minimum-cost flow, network flow, out-of-kilter
CR CATEGORIES: 5.32, 5.41

Algorithm 248 was transcribed into Burroughs Extended Algol for the Burroughs B5500. After modification it has been used successfully. Before modification it was found to give erroneous values of *pi* for transportation problems and nonoptimal solutions for networks representing multitime level trans-shipment problems. This was caused by the method utilized within the procedure for exiting with the best solution. The difficulty was circumvented by inserting a statement just before label *SKIP* reading:

if *nb* [*src*] = *arcs* **then go to** *FINI*;

This statement enables the user to exit the procedure without a pass through the *pi* incrementation block and a final pass through the out-of-kilter arc-finding block, saving a significant amount of time on sizeable problems. With the arcs arranged so that the arc directed from the "super sink" to the "super source" is the last one in the arc array, it must be the last arc remaining out-of-kilter. Therefore, by the time the search block discovers it as an out-of-kilter arc, an optimal solution has already been found.

[Algorithm 336 [*Comm. ACM 11* (Sept. 1968), 631–632] is an improved version of Algorithm 248, which by its very construction bypasses this error.—J.G.H.]

REMARK ON ALGORITHM 248 [H]

NETFLOW [William A. Briggs, *Comm. ACM 8* (Feb. 1965), 103]

T. A. Bray and C. Witzgall
(Recd. 2 Oct. 1967 and 20 May 1968)
Boeing Scientific Research Laboratories, Seattle, WA 98124

KEY WORDS AND PHRASES: capacitated network, linear programming, minimum-cost flow, network flow, out-of-kilter
CR CATEGORIES: 5.32, 5.41

We found that

1. in the statement
   c := *cost*[*a*] − *abs*(*pi*[*ni*]−*pi*[*nj*]) × *sign*(*nb*[*ni*]);
   on page 103, column 2, line 3 from below,
   the "*abs*" should be deleted.
2. in the statement
   LABL: **if** *a* = *aok* ∧ *na*[*src*] ≠ 0 **then go to** *SKIP*;
   on page 103, column 2, line 13 from above,
   the value of *na*[*src*] may be undefined.

The algorithm worked satisfactorily after the corresponding changes had been made. We acknowledge a correspondence with R. M. Van Slyke and R. D. Sanderson of the University of California, Berkeley, on the subject.

Algorithm 336 [*Comm. ACM 11* (Sept. 1968), 631–632] is an improved version of Algorithm 248 incorporating these changes.