as before and computes

$$\hat{\sigma}^2 + L\hat{\sigma}^2_{TA} + JL\hat{\sigma}^2_{PA} + KL\hat{\sigma}_T{}^2$$

for testing this source of variation. Thus instances in which exact tests exist for certain model terms can be considered a special case of the algorithm.

If no exact F test exists for a certain model term, the degrees of freedom corresponding to the denominator used in calculating the F statistic for this term can be obtained as

$$\hat{\nu} = \frac{\hat{\gamma}^2}{\sum(\hat{\gamma}_i{}^2/\nu_i)}$$

$\hat{\gamma}$ is the value used as the denominator in calculating the F value, while the $\hat{\gamma}_i$ and ν_i are the calculated mean squares and degrees of freedom, respectively of the model terms whose EMS's when combined equal the expectation of γ. See [3] for a derivation of this formula. All values for calculating $\hat{\nu}$ are known besides the $\hat{\gamma}_i$ and these are obtained by constructing from the algorithm a table denoting the structure of the EMS's of the model terms. From the table a triangular set of equations solvable by a backward solution is easily obtained. The solution vector gives the calculated mean squares which are to be used in computing $\hat{\nu}$.

All concepts and methodology presented in this paper have been fully implemented on the IBM 7074 while the authors were members of the Statistical Laboratory at Iowa State University. An attempt was made to avoid machine dependencies. As a consequence, the program has been readily converted to the IBM 360, Model 50 now in use at Iowa State University. It is intended that the algorithms developed will form part of a more extensive statistical computing system oriented toward algebraic problem specification.

Acknowledgment. The authors wish to thank E. J. Carney, Assistant Professor of Statistics, Iowa State University, for constructing a preliminary version of the variance component algorithm to handle factorial models.

RECEIVED DECEMBER 1965; REVISED JULY 1966

REFERENCES

1. HEMMERLE, W. J. Algebraic specifications of statistical models for analysis of variance computations. *J. ACM 11* (1964), 234–239.
2. SATTERTHWAITE, F. E. An approximate distribution of estimates of variance components. *Biometrics 2* (1946), 110–114.
3. SCHEFFÉ, HENRY. *The Analysis of Variance*. John Wiley, New York, 1959.
4. SCHLATER, J. E. Analysis of variance and covariance computations on a digital computer for balanced complete structures based on algebraic model specifications. M.S. Thesis, Iowa State U. Library, 1965.

Algorithms

J. G. HERRIOT, Editor

ALGORITHM 293
TRANSPORTATION PROBLEM [H]

G. BAYER (Recd. 9 July 1965 and 22 Aug. 1966)
Technische Hochschule, Braunschweig, Germany

```
procedure transp1 (m, n, inf, c, a, b, x, kw); value m, n, inf;
  integer m, n, inf, kw; integer array c, a, b, x;
comment transp1 is derived from Algorithm 258, transport,
  [Comm. ACM 8 (June 1965), 381] in order to reduce running time
  by about 50 percent. The following notation is used.
  c   m, n-matrix of unit costs,
  a   array of quantities available,
  b   array of quantities required, following the usual description of the transportation problem,
  inf greatest positive integer within machine capacity,
  x   m, n-matrix of flows,
  kw  optimal total costs (computed by procedure).
  c, a, b are disturbed by the procedure. Sum of a[i] = sum of b[i].
  Multiple solutions are left out of account. [Ref.: G. Hadley,
  Linear Programming, Reading, London, 1962, p. 351];
begin integer i, j, u, v, k, l, s, t, gd, h, p, cij, xij, ai, bj, lsvj, nlvi;
  Boolean zg;
  integer array g, listu, nlv[1:m], r, listv[1:n], ls[0:m+n-1],
    nl[1:m×n], lsv[0:n];
  comment in the for-statement u := ··· after s33, operate on
    all pairs i, j with c[i,j] = 0. To win time the array nl supervises
    those zeros; the j-indices of zeros in row i are kept in
    nl[(i-1)×n+1] ··· nl[nlv[i]]. In the for-statement v := ···
    after s33, operate on all pairs i, j with x[i,j] ≠ 0 (and c[i,j]=0).
    ls supervises those essential zeros, the i-indices of essential
    zeros in column j are kept in ls[lsv[j-1]+1] ··· ls[lsv[j]
    Procedure in adds to list ls, procedure out takes out from list
    ls an essential zero in position i, j;
  procedure in;
  begin
    lsvj := lsv[j];
    for t := lsv[n] step -1 until lsvj do ls[t+1] := ls[t];
    for t := j step 1 until n do lsv[t] := lsv[t] + 1;
    ls[lsvj+1] := i
  end ;
  procedure out ;
  begin
    lsvj := lsv[j];
    for t := lsv[j-1]+1 step 1 until lsvj do
    begin
      if ls[t] ≠ i then go to next;
      s := t; go to ex;
  next:
    end ;
  ex:
    for t := j step 1 until n do lsv[t] := lsv[t]-1;
    lsvj := lsv[n];
    for t := s step 1 until lsvj do ls[t] := ls[t+1]
  end ;
  for i := 1 step 1 until m do
    for j := 1 step 1 until n do x[i,j] := 0;
  for i := 1 step 1 until m do nlv[i] := (i-1)×n;
```

```
    lsv[0] := 0;
    for j := 1 step 1 until n do
    begin
      listv[j] := 1;
      lsv[j] := 0
    end ;
s1: kw := gd := 0;
    comment gd is the defect, i.e., the sum of quantities not yet
      transported;
    for i := 1 step 1 until m do
    begin
      h := inf;
      for j := 1 step 1 until n do
        if c[i, j] < h then h := c[i, j];
      for j := 1 step 1 until n do
      begin
        cij := c[i, j] := c[i, j] - h;
        if cij = 0 then
        begin
          listv[j] := 0;
          nlvi := nlv[i] := nlv[i] + 1;
          nl[nlvi] := j
        end
      end;
      kw := h × a[i] + kw
    end see next comment;
    for j := 1 step 1 until n do
    begin
      if listv[j] = 0 then go to nextj1;
      h := inf;
      for i := 1 step 1 until m do
        if c[i, j] < h then h := c[i, j];
      for i := 1 step 1 until m do
      begin
        cij := c[i, j] := c[i, j] - h;
        if cij = 0 then
        begin
          nlvi := nlv[i] := nlv[i] + 1;
          nl[nlvi] := j
        end
      end;
      kw := h × b[j] + kw;
nextj1:
    end;
    comment in step 1 the usual reduction of the matrix of costs
      is achieved (dual problem), zeros are listed in nl;
s2: for i := 1 step 1 until m do
    begin
      ai := a[i]; nlvi := nlv[i];
      for u := (i-1) × n + 1 step 1 until nlvi do
      begin
        if ai = 0 then go to nexti2;
        j := nl[u];
        bj := b[j];
        if bj = 0 then go to nextj4;
        h := x[i, j] := if ai < bj then ai else bj;
        ai := ai - h; b[j] := bj - h; in;
nextj4:
      end;
nexti2:
      a[i] := ai; gd := gd + ai
    end;
    comment applying a usual rule to all zeros we get an initial
      flow (restricted primal problem) in step 2;
s31:
    if gd = 0 then go to s6;
    comment problem is solved if defect has become zero;
s32:
    for j := 1 step 1 until n do r[j] := 0;
    k := 0;
    for i := 1 step 1 until m do
    begin
      if a[i] ≠ 0 then
      begin
        k := k + 1; listu[k] := i; g[i] := inf
      end
      else g[i] := 0
    end;
    comment r[j] = 0 if column j is unlabeled, = i if labeled
      from row i. g[i] = 0 if row i is unlabeled, = inf if a[i] ≠ 0,
      i.e., a[i] is a possible source of flow. The indices i of labeled
      rows are kept in listu[1] ··· listu[k]. In step 3, consisting of
      step 32 and step 33, the maximal flow is found by the la-
      beling process. Labeling ends in only two ways: (a) a column j
      with b[j] > 0 has been labeled: go to step 4, (b) all labeling is
      done, but a positive flow has not been found: go to s5;
s33: l := 0;
    for u := 1 step 1 until k do
    begin
      i := listu[u]; nlvi := nlv[i];
      begin
        j := nl[s];
        if r[j] ≠ 0 then go to nextj5;
        r[j] := i; l := l + 1; listv[l] := j;
        if b[j] > 0 then go to s4;
nextj5:
      end
    end in each newly labeled row, see listu, look for zeros in
      unlabeled columns, list them in listv;
    if l = 0 then go to s5;
    k := 0;
    for v := 1 step 1 until l do
    begin
      j := listv[v]; lsvj := lsv[j];
      for s := lsv[j-1]+1 step 1 until lsvj do
      begin
        i := ls[s];
        if g[i] = 0 then
        begin
          g[i] := j; k := k + 1;
          listu[k] := i
        end
      end
    end in each newly labeled column, see listv, look for essential
      zeros in unlabeled rows, label these rows, list them in listu;
    if k = 0 then go to s5;
    go to s33;
    comment step 4. A column j with b[j] has been labeled, b[j]
      is the sink of a possible positive flow, the path of which is
      indicated by labels. Find the minimum flow h along the path;
    h := b[j]; p := j;
mark:
    i := r[j]; j := g[i];
    if j = inf then
    begin
      if a[i] < h then h := a[i]; go to re
    end;
    if x[i, j] < h then h := x[i, j];
    go to mark;
```

```
re: ;
  comment flow h along the labeled path thus reduces defect
    without changing total costs. Correct list of essential zeros
    if necessary. Start labeling anew, optimizing the restricted
    primal problem;
  j := p; b[j] := b[j] − h; a[i] := a[i] − h;
  gd := gd − h;
re1:
  i := r[j]; xij := x[i, j]; x[i, j] := xij + h;
  if xij = 0 then in;
  j := g[i];
  if j = inf then go to s31;
  xij := x[i, j] := x[i, j] − h;
  if xij = 0 then out;
  go to re1;
s5: ;
  comment step 5. Flow is maximal. To find a new solution to
    the dual, take the part of matrix c which is the intersection
    of labeled rows and unlabeled columns, reduce matrix in a
    certain way;
  k := 0; l := n + 1;
  for j := 1 step 1 until n do
  begin
    if r[j] = 0 then
    begin
      k := k + 1; listv[k] := j
    end
    else
    begin
      l := l − 1; listv[l] := j
    end
  end list all labeled resp. unlabeled columns in listv;
  h := inf;
  for i := 1 step 1 until m do
  begin
    if g[i] = 0 then go to nexti6;
    for s := 1 step 1 until k do
    begin
      j := listv[s];
      if c[i, j] < h then h := c[i, j]
    end;
nexti6:
  end find minimum h in partial matrix;
  for i := 1 step 1 until m do
  begin
    zg := g[i] ≠ 0; nlvi := (i−1) × n;
    for s := 1 step 1 until n do
    begin
      j := listv[s];
      if zg then cij := c[i, j]
      else
      cij := c[i, j] := c[i, j] + h;
      if cij = 0 then
      begin
        nlvi := nlvi + 1;
        nl[nlvi] := j
      end
    end;
    for s := 1 step 1 until k do
    begin
      j := listv[s];
      if zg then cij := c[i, j] := c[i, j] − h
      else cij := c[i, j];
      if cij = 0 then
      begin
        nlvi := nlvi + 1;
        nl[nlvi] := j
      end
    end;
    nlv[i] := nlvi
  end reduction, add h to labeled columns, subtract h from
    labeled rows. Construct new list of zeros;
  kw := h × gd + kw;
  comment total costs for new solution of dual;
  go to s32;
s6: ;
  comment solution, defect has become zero;
end
```

CERTIFICATION OF ALGORITHM 257 [D1] HAVIE INTEGRATOR [Robert N. Kubik, *Comm. ACM 8* (June 1965), 381]

I. Farkas (Recd. 29 Apr. 1966 and 18 Aug. 1966)
Institute of Computer Science, University of Toronto, Toronto 5, Ont., Canada

Havieintegrator was translated with some modifications into Fortran IV and was run on the IBM 7094 II at the Institute of Computer Science, University of Toronto. To reduce the effect of roundoff, the calculations were carried through in double precision internally and the result was rounded to single precision. The main change made was that the parameters *x* and *integrand* in *havieintegrator* were replaced by a single parameter of type FUNCTION in Fortran IV. The other change was that *mask* was removed. The maximum order of approximation was kept less than or equal to 25, and convergence was obtained in every case.

The results obtained for the two test cases were in agreement with the author's result. Besides, 14 other successful tests were made and those shown in Table I are typical.

TABLE I

Integrand	*A*	*B*	*True value*	*eps*	*Error* × 10^3	*Order required*
e^x	0.0	1.0	1.7182818	10^{-6}	0	3
				10^{-4}	240	2
				10^{-2}	3700	2
x^{12}	0.01	1.1	.26555932	10^{-6}	−2	4
				10^{-4}	59	3
				10^{-2}	36041	2
$\sqrt{x}$	0.0	1.0	.66666667	10^{-6}	−27	3
				10^{-4}	−1982	2
				10^{-2}	−126848	2
$1/\sqrt{x}$	0.01	1.0	1.8000000	10^{-6}	0	3
				10^{-4}	140	2
				10^{-2}	790	2

Like other integration algorithms that determine sample points in the interval in a deterministic manner, *havieintegrator* may fail in certain instances. For example, any integrand with the property that $f(a) = f(b) = f[(a + b)/2)]$ will lead to the value $(b - a)f(a)$ which will in general not be an acceptable approximation to $\int_a^b f(x)\,dx$. Thus $\int_0^{2\pi} \sin^2 x\,dx$ leads to 0. Moreover, $\int_0^{90} xe^{-x}\,dx$ leads to "almost zero" (in fact, 5.7966×10^{-17}).

Please turn the page to the 1966 Algorithms Index.

Index By Subject TC Algorithms, 1966

Key—1st column: A1, B1, B3, etc. is the key to the underlined Modified Share Classification heading each group of algorithms; 2d column: number of the algorithm in *CACM*; 3d column: title of algorithm: 4th column: month, year and page (in parens) in *CACM*, or reference elsewhere. This index supplements the previously published indexes: Index by Subject to Algorithms: 1960–1963 [*CACM* 7 (Mar. 1964), 146–149]; 1964 [*CACM* 7 (Dec. 1964), 703]; and 1965 [*CACM* 8 (Dec. 1965), 791].