Algorithms

J. G. HERRIOT, Editor

ALGORITHM 284
INTERCHANGE OF TWO BLOCKS OF DATA [K2]

WILLIAM FLETCHER (Recd. 25 Oct. 1965 and 24 Nov. 1965)
Bolt, Beranek and Newman, Inc., Cambridge, Mass.
and
ROLAND SILVER
The Mitre Corp., Bedford, Mass.

```
procedure interchange (a, m, n);
  value m, n;  integer m, n;  array a;
comment This procedure transfers the contents of a[1] ··· a[m]
  into a[n+1] ··· a[n+m] while simultaneously transferring the
  contents of a[m + 1] ··· a[m + n] into a[1] ··· a[n] without using
  an appreciable amount of auxiliary memory.
    The nonlocal procedure gcd (x, y) has value the greatest
  common divisor of the integers x and y. The nonlocal procedure
  swap (x, y) interchanges the values of the variables x and y.
    Let G be the additive group of integers modulo m+n. The
  multiples 0, n, 2n, ··· of n form a cyclic subgroup C of G. The
  order of C is r = (m + n)/d, where d is the greatest common
  divisor of m and n. The integers 1, ··· , d belong to distinct
  cosets C1 ··· Cd of C. These cosets form a disjoint covering of G.
    The interchange procedure is based on the fact that if we start
  with a member x of the coset Cx , and add n repeatedly modulo
  m + n, we will in r steps have generated each member of Cx
  just once;
begin
  integer d, i, j, k, r;
  real t;
  d := gcd (m, n);
  r := (m + n) ÷ d;
  for i := 1 step 1 until d do
  begin
    j := i;
    t := a[i];
    for k := 1 step 1 until r do
    begin
      if j ≦ m then j := j + n else j := j − m;
      swap (t, a[j])
    end k
  end i
end interchange
```


ALGORITHM 285
THE MUTUAL PRIMAL—DUAL METHOD [H]

THOMAS J. AIRD (Recd. 29 June 1964 and 5 Apr. 1965)
Wolf Research and Development Corporation
Manned Spacecraft Center
Houston, Texas

```
procedure Linearprogram (n, p, A, min, psol, dsol, bool);
  value p, n;  integer p, n;  array A, psol, dsol;  real min;
  Boolean bool;
comment This procedure solves the linear programming prob-
  lem by the Mutual Primal-Dual Simplex Method. The problem
  is assumed to be in the following form:
```

$$AX + B \leq 0$$
$$X \geq 0$$
$$\min u = d + C^T X$$

where A is $p \times n$, B is $p \times 1$ and C is $n \times 1$. The dual problem is then,

$$Y \geq 0$$
$$A^T Y + C \geq 0$$
$$\max v = d + B^T Y.$$

The matrix of coefficients, also called A is formed in the following way:

$$A = \begin{bmatrix} d & C_1 & C_2 & \cdots & C_n \\ b_1 & A_{11} & A_{12} & \cdots & A_{1n} \\ b_2 & A_{21} & A_{22} & \cdots & A_{2n} \\ \vdots & & & & \\ b_p & A_{p1} & A_{p2} & \cdots & A_{pn} \end{bmatrix}$$

The input matrix A is declared [0: p, 0: n], *min* is the value of the objective function, *psol* is the solution vector for the primal problem, *dsol* is the solution vector for the dual problem, *bool* will be set to **true** if an optimal solution is found, otherwise *bool* will be set to **false**;

```
begin integer array row [0:2×p,0:p], col [0:2×p,0:n], norow,
  nocol [0:2×p], index [0:n+p];
  integer i, j, k, s, t;
  procedure subschema (k);  integer k;
  comment This procedure defines an admissible sequence of
    subschema Sk+1 Sk+2 , ··· , assuming that S1 , S2 , ··· Sk ,
    have already been defined;
  begin integer count;
    for i := 1 step 1 until p do if A[i,0] > 0 then go to
      WORK;
    for j := 1 step 1 until n do if A[0,j] < 0 then go to
      WORK;  k := 0;  go to RETURN;
WORK:  if 2 × (k÷2) = k then go to EVEN else go to ODD;
EVEN:
  begin
      if k = 0 then
      begin
        for i := 1 step 1 until p do if A[i,0] > 0 then
```

```
        begin
          row[1,0] := i;   go to D3
        end;
        row[1,0] := 0;   go to D3
      end;
      for j := 1 step 1 until nocol[k] do
        if A[row[k,0],col[k,j]] = 0 then go to D1;
      go to RETURN;
D1:   for i := 1 step 1 until norow[k] do
        if A[row[k,i],col[k,0]] > 0 then go to D2;
      go to RETURN;
D2:   row[k+1,0] := row[k,i];
      col[k+1,0] := col[k,0];
      count := 0;
      for j := 1 step 1 until nocol[k] do
      if A[row[k,0],col[k,j]] = 0 then
      begin
        count := count + 1;
        col[k+1,count] := col[k,j]
      end;
      nocol[k+1] := count;
D3:   count := 0;
      for i := 1 step 1 until norow[k] do
      if A[row[k,i],col[k,0]] ≤ 0 then
      begin
        count := count + 1;
        row[k+1,count] := row[k,i]
      end;
      norow[k+1] := count;
      k := k + 1;
      go to ODD
    end EVEN;
ODD:
    begin
      for i := 1 step 1 until norow[k] do
        if A[row[k,i],col[k,0]] = 0 then go to B1;
      go to RETURN;
B1:   for j := 1 step 1 until nocol[k] do
        if A[row[k,0],col[k,j]] < 0 then go to B2;
      go to RETURN;
B2:   col[k+1,0] := col[k,j];
      row[k+1,0] := row[k,0];
      count := 0;
      for i := 1 step 1 until norow[k] do
      if A[row[k,i],col[k,0]] = 0 then
      begin
        count := count + 1;
        row[k+1,count] := row[k,i]
      end;
      norow[k+1] := count;
      count := 0;
      for j := 1 step 1 until nocol[k] do
      if A[row[k,0],col[k,j]] ≥ 0 then
      begin
        count := count + 1;
        col[k+1,count] := col[k,j]
      end;
      nocol[k+1] := count;
      k := k + 1;
      go to EVEN
    end ODD;
RETURN:
  end subschema;
  procedure pivot (s,t);   value s, t;   integer s, t;
  comment  The procedure pivot performs the usual pivot opera-
    tion on the matrix A, A[s,t] is the pivot element;
  begin integer i, j;
    A[s,t] := 1/A[s,t];
    for i := 0 step 1 until s − 1, s + 1 step 1 until p do
      begin
        A[i,t] := −A[i,t] × A[s,t];
        for j := 0 step 1 until t − 1, t + 1 step 1 until n do
          if abs(A[i,j]+A[i,t]×A[s,j])  ≤  abs(A[i,j]×10−8)  then
          A[i,j] := 0
          else A[i,j] := A[i,j] + A[i,t] × A[s,j]
      end;
      for j := 0 step 1 until t − 1, t + 1 step 1 until n do
        A[s,j] := A[s,j] × A[s,t];
      i := index[t];
      index[t] := index[n+s];
      index[n+s] := i
    end pivot;
    procedure pickapivot (k,s,t);   integer k, s, t;
    comment  The procedure pickapivot will choose a pivot ele-
      ment from S_k or S_k−1 in a manner which will guarantee im-
      provement in the goal vector;
    begin real max, test;
      if 2 × (k÷2) = k then go to EVEN else go to ODD;
ODD:
      begin
        for j := 1 step 1 until nocol[k] do
        if A[row[k,0],col[k,j]] < 0 then
        begin
          for i := 1 step 1 until norow[k] do
            if A[row[k,i],col[k,j]] > 0 then go to A1;
          s := row[k,0];
          t := col[k,j];
          k := k − 1;
          go to RETURN;
A1:
        end;
        for j := 1 step 1 until nocol[k] do
        if A[row[k,0],col[k,j]] < 0 then
        begin
          for i := 1 step 1 until norow[k] do
          if A[row[k,i],col[k,j]] > 0 then
          begin s := row[k,i];
            t := col[k,j];
            max := A[row[k,i],col[k,0]]/A[row[k,i],col[k,j]];
            go to A2
          end
        end;
        go to A3;
A2:     for i := i + 1 step 1 until norow[k] do
        if A[row[k,i],col[k,j]] > 0 then
        begin
          test := A[row[k,i],col[k,0]]/A[row[k,i],col[k,j]];
          if test > max then
          begin
            s := row[k,i];
            max := test
          end
        end;
        k := k − 1;
        go to RETURN;
A3:     for j := 1 step 1 until nocol[k−1] do
        if A[row[k,0],col[k−1,j]] < 0 then
        begin
          s := row[k,0];
          t := col[k−1,j];
          max  :=  A[row[k−1,0],col[k−1,j]]/A[row[k,0],col[k−1,j]];
          go to A4
        end;
        s := row[k,0];
        t := col[k,0];
        k := k − 2;
        go to RETURN;
```

```
A4:  for j := j + 1 step 1 until nocol[k-1] do
     if A[row[k,0],col[k-1,j]] < 0 then
      begin
        test  :=  A[row[k-1,0],col[k-1,j]]/A[row[k,0],col[k-1,j]];
        if test > max then
        begin
          t := col[k-1,j];
          max := test
        end
      end;
      k := k - 2;
      go to RETURN
    end ODD;
EVEN:
    begin
      for i := 1 step 1 until norow[k] do
      if A[row[k,i],col[k,0]] > 0 then
      begin
        for j := 1 step 1 until nocol[k] do
        if A[row[k,i],col[k,j]] < 0 then
        go to B1;
        s := row[k,i];
        t := col[k,0];
        k := k - 1;
        go to RETURN;
B1:
      end;
      for i := 1 step 1 until norow[k] do
      if A[row[k,i],col[k,0]] > 0 then
      begin
        for j := 1 step 1 until nocol[k] do
        if A[row[k,i],col[k,j]] < 0 then
        begin
          s := row[k,i];
          t := col[k,j];
          max :=  A[row[k,0],col[k,j]]/A[row[k,i],col[k,j]];
          go to B2
        end
      end;
      go to B3;
B2:  for j := j + 1 step 1 until nocol[k] do
     if A[row[k,i],col[k,j]] < 0 then
     begin
       test := A[row[k,0],col[k,j]]/A[row[k,i],col[k,j]];
       if test > max then
       begin
         t := col[k,j];
         max := test
       end
     end;
     k := k - 1;
     go to RETURN;
B3:  for i := 1 step 1 until norow[k-1] do
     if A[row[k-1,i],col[k,0]] > then
     begin
       s := row[k-1,i];
       t := col[k,0];
       max := A[row[k-1,i],col[k-1,0]]/A[row[k-1,i],col[k,0]];
       go to B4
     end;
     s := row[k,0];
     t := col[k,0];
     k := k - 2;
     go to RETURN;
B4:  for i := i + 1 step 1 until norow[k-1] do
     if A[row[k-1,i],col[k,0]] > then
     begin
       test := A[row[k-1,i],col[k-1,0]]/A[row[k-1,i],col[k,0]];
          if test > max then
          begin
            s := row[k-1,i];
            max := test
          end
        end;
        k := k - 2;
        go to RETURN
      end EVEN;
RETURN:
  end pickapivot;
  for i := 1 step 1 until p + n do index[i] := i;
  for i := 0 step 1 until p do row[0,i] := i;
  for j := 0 step 1 until n do col[1,j] := j;
  norow[0] := p;  nocol[1] := n;  k := 0;
  comment  This is a check on the row constraints;
NEXTPIVOT:
  for i := 1 step 1 until p do
  begin
    if A[i,0] ≤ 0 then go to NEXTI;
    for j := 1 step 1 until n do
      if A[i,j] < 0 then go to NEXTI;
    comment  Row constraints are incompatible;
    bool := false;
    go to FINISH;
NEXTI:
  end;
  comment  This is a check on the column constraints;
  for j := 1 step 1 until n do
  begin
    if A[0,j] ≥ 0 then go to NEXTJ;
    for i := 1 step 1 until p do
      if A[i,j] > 0 then go to NEXTJ;
    comment  Column constraints are incompatible;
    bool := false;
    go to FINISH;
NEXTJ:
  end;
  subschema(k);
  if k = 0 then
  begin
    comment  k = 0 indicates that the present solution is opti-
      mal. A[0,0] is value of the objective function;
    min := A[0,0];
    for i := 1 step 1 until p + n do psol[i] := dsol[i] := 0;
    comment  Find the primal solution vector;
    for i := 1 step 1 until p do
      psol[index[n+i]] := -A[i,0];
    comment  Find the dual solution vector;
    for i := 1 step 1 until n do
      if index[i] > n then
      dsol[index[i]-n] := A[0,i]
      else
      dsol[index[i]+p] := A[0,i];
    bool := true;
    go to FINISH;
  end;
  pickapivot(k,s,t);
  if s = 0 ∨ t = 0 then
  begin
    comment  No feasible solution;
    bool := false;
    go to FINISH;
  end;
  pivot(s,t);
  go to NEXTPIVOT;
FINISH:
end Linearprogram
```

(Algorithms are continued on page 354)

set, being formed of two proper tight sets; elements such as A [4, 8] belong to neither so that further applications of EXPAND are needed to determine the feasibility of elements in rows 4 through 9.

6. Timing

Suppose the array A has n rows with an average of m nonzero elements per row, i.e., there are (mn) nonzero elements in A. In the worst conceivable case each element of A would require a separate application of EXPAND and each application of EXPAND would involve the examination of the whole matrix, i.e., the number of operations would be of the order of $(m\,n)^2$. However there is a complex interaction between the actual number of calls on EXPAND and the average number of rows searched in a single application of EXPAND—the higher the average number of rows searched per application, the less the average number of applications, and vice versa; so that it is difficult to estimate the exact dependence on m and n. Empirical results seem to indicate that the above estimate is conservative.

For the timetable problem m is always bounded by the number of hours in a school day (about ten) and decreases steadily during the calculation, and hence the number of operations required is of the order of n^2.

7. Conclusion

A practical algorithm based on the Hungarian Method of H. Kuhn has been described for carrying out the examination and reduction of 2-dimensional arrays as required in Gotlieb's method for the solution of the timetable problem. In addition, various devices to improve the efficiency of the algorithm have been described.

Received December, 1965

REFERENCES

1. Gotlieb, C. C. The construction of class-teacher timetables. Proc. IFIP Congress 62 (Munich), North Holland Publ. Co., 1963, 73–77.
2. Ackoff, R. L. (ed.) *Progress in Operations Research*. Wiley, 1961, 149–150.
3. Kuhn, H. W. The Hungarian Method for the assignment problem. *Nav. Res. Log. Quart. 2* (1955), 83–97.
4. Friedman, L. F., and Yaspan, A. J. An analysis of stewardess requirements and scheduling for a major domestic airline: Annex A. The Assignment Problem technique. *Nav. Res. Log. Quart. 1* (1954) 223–229.
5. Hall, P. On representatives of subsets. *J. Lond. Math. Soc. 10* (1935), 26–30.
6. Csima, J. Investigations on a timetable problem. Ph.D. Thesis, U. of Toronto, 1965.

APPENDIX

```
procedure expand (A, rowsolution, columnsolution, row, n,
  infeasible);
value row, n; integer row, n; Boolean infeasible;
integer array A, rowsolution, columnsolution;
  begin
comment This procedure performs one iteration of the Hun-
  garian Method on the array A. A partial solution which does
  not include an element in the row "row" is defined by the array
  "columnsolution" (and equivalently by the array "rowsolu-
  tion"). The partial solution is rearranged to allow an additional
  element taken from the designated, previously unrepresented
  row to be incorporated in a new enlarged partial solution, when
  this is possible. If the latter is not possible, the Boolean vari-
  able "infeasible" is set to true;
integer j, k, marknext, marknew;
integer array reference [1:n], rowlist [1:n];
  infeasible := false; marknext := marknew := 0;
  for j := 1 step 1 until n do reference [j] := 0;
newrow: for j := 1 step 1 until n do
  begin if A [row, j] ≠ 0 ∧ reference [j] = 0 then
    begin if columnsolution [j] = 0 then goto backtrack;
      reference [j] := row;
      marknew := marknew + 1;
      rowlist [marknew] := columnsolution [j]
    end
  end;
  if marknext ≥ marknew then goto nosolution;
  marknext := marknext + 1;
  row := rowlist [marknext];
  goto newrow;
backtrack: k = rowsolution [row];
  columnsolution [j] := row; rowsolution [row] := j;
  if k = 0 then goto finis;
  j := k; row := reference [k];
  goto backtrack;
nosolution: infeasible := true;
finis: end of expand
```

ALGORITHMS—cont'd from page 328

CERTIFICATION OF ALGORITHM 271 (M1)

QUICKERSORT [R. S. Scowen, *Comm. ACM 8* (Nov. 1965), 669]

Charles R. Blair (Recd. 11 Jan. 1966)
Department of Defense, Washington, D.C.

QUICKERSORT compiled and ran without correction through the ALDAP translator for the CDC 1604A. Comparison of average sorting times, shown in Table I, with other recently published algorithms demonstrates *QUICKERSORT*'s superior performance.

TABLE I. Average Sorting Times in Seconds

Number of items	Algorithm 201 Shellsort		Algorithm 207 Stringsort		Algorithm 245 Treesort 3		Algorithm 271 Quickersort	
	Integers	Reals	Integers	Reals	Integers	Reals	Integers	Reals
10	0.01	0.01	0.03	0.03	0.02	0.02	0.01	0.01
20	0.02	0.02	0.05	0.05	0.04	0.04	0.02	0.02
50	0.08	0.08	0.20	0.20	0.11	0.12	0.06	0.06
100	0.19	0.22	0.39	0.40	0.26	0.27	0.13	0.13
200	0.48	0.53	1.0	1.1	0.59	0.62	0.28	0.30
500	1.5	1.7	2.8	2.9	1.7	1.8	0.80	0.85
1000	3.7	4.2	6.6	6.9	3.7	4.0	1.8	1.9
2000	9.1	10.	13.	14.	8.2	8.7	3.9	4.1
5000	27.	30.	40.	41.	23.	24.	11.	12.
10000	65.	72.	93.	97.	49.	52.	23.	25.