ZASTOSOWANIA MATEMATYKI
APPLICATIONES MATHEMATICAE
XIII, 1 (1972)

ALGORITHM 16

J. KUCHARCZYK (Wrocław)

IMPLICIT ENUMERATION ALGORITHM FOR SOLVING ZERO-ONE INTEGER LINEAR PROGRAMS

1. Procedure declaration. The procedure *ilp01SW* solves the following zero-one integer linear programming problem by implicit enumeration:

$$\min f = \sum_{j=1}^{n} c_j x_j \qquad (c_j \geqslant 0),$$

provided

$$\sum_{j=1}^{n} a_{ij} x_j \geqslant b_i \qquad (i = 1, 2, \ldots, m),$$

$$x_j = 0, 1 \qquad (j = 1, 2, \ldots, n).$$

Data:

n — number of variables,
m — number of constraints,
$a[1{:}\ m,\ 1{:}\ n]$ — coefficient matrix of the constraints,
$b[1{:}\ m]$ — right sides of the constraints,
$c[1{:}\ n]$ — coefficients of the objective function,
inf — maximum positive integer number.

Results:

$x[1{:}\ n]$ — optimum solution (if ex = **true** only, otherwise the procedure *ilp01SW* does not change the array x),
f — optimum value of the objective function (if ex = **true** only, otherwise the procedure *ilp01SW* does not change the value of f),
ex — **true** if an optimum solution exists and **false** if there is no feasible solution to the problem.

Other parameters:

bl — label to which exit from the procedure body is made if any of the elements of $c[1{:}\ n]$ is negative.

```
procedure ilp01SW(n,m,a,b,c,x,f,ex,inf,bl);
 value n,m,inf;
 integer n,m,f,inf;
 Boolean ex;
 label bl;
 integer array a,b,c,x;
 begin
  integer i,j,z,zd,W,Wg,d,sq,y,I,J,mine,ymin;
  integer array mi,xd,xx[1:n];
  zd:=inf;
  ex:=false;
  sq:=0;
  for j:=1 step 1 until n do
   begin
    xx[j]:=-1;
    mi[j]:=0;
    if c[j]<0
     then go to bl
   end j;
  for i:=1 step 1 until m do
   if b[i]>0
    then go to ntrivs;
  f:=0;
  ex:=true;
  for j:=1 step 1 until n do
   x[j]:=0;
  go to exit;
ntrivs:
  ymin:=0;
  for i:=1 step 1 until m do
```

```
    begin
     y:=-b[i];
     for j:=1 step 1 until n do
      if xx[j]>0
       then y:=y+a[i,j];
     if y<ymin
      then
      begin
       ymin:=y;
       I:=i
      end y<ymin
    end i;
   z:=0;
   for j:=1 step 1 until n do
    if xx[j]>0
     then z:=z+c[j];
   if ymin=0∧z<zd
    then
feas:
    begin
     zd:=z;
     ex:=true;
     for j:=1 step 1 until n do
      xd[j]:=if xx[j]>0 then 1 else 0,
     go to backtr
    end ymin=0∧z<zd;
   if zd<inf
    then
test1:
    begin
```

```
        mine:=inf;
        for j:=1 step 1 until n do
         if xx[j]<0
          then
          begin
           y:=c[j];
           if y<mine
            then mine:=y
          end j;
        if z+mine≥zd
         then go to backtr
       end zd<inf;
      d:=0;
      for j:=1 step 1 until n do
       if xx[j]<0
        then
        begin
         y:=a[I,j];
         if y>0
          then d:=d+y
        end j;
      W:=ymin+d;
    test2:
      if W<0
       then go to backtr;
      if W=0
       then
    maug:
       begin
        for j:=1 step 1 until n do
```

```
    if xx[j]<0
     then
     begin
      y:=a[I,j];
      if y>0
       then
       begin
        sq:=sq+1;
        mi[sq]:=-j;
        xx[j]:=1;
        z:=z+c[j]
       end y>0
       else
        if y<0
         then
         begin
          sq:=sq+1;
          mi[sq]:=-j;
          xx[j]:=0
         end y<0
    end j;
  go to if z<zd then ntrivs else backtr
 end W=0;
d:=mine:=0;
for j:=1 step 1 until n do
 if xx[j]<0
  then
  begin
   y:=a[I,j];
   if y>0
```

```
          then
          begin
           if z+c[j]<zd
            then
            begin
             d:=d+y;
             if mine<y
              then
              begin
               mine:=y;
               J:=j
              end mine<y
            end z+c[j]<zd
          end y>0
        end j;
test3:
  if d=0
   then go to backtr;
  Wg:=ymin+d;
test4:
  if Wg<0
   then go to backtr;
test5:
  if W-mine<0
   then
   begin
augm:
     sq:=sq+1;
     mi[sq]:=-J;
     xx[J]:=1;
```

```
   go to ntrivs
  end W-mine<0;
test6:
 if Wg-mine<0
  then go to augm
  else
  begin
   sq:=sq+1;
   mi[sq]:=J;
   xx[J]:=1;
   go to ntrivs
  end Wg-mine≥0;
backtr:
 if sq=0
  then go to END;
 if mi[sq]<0
  then
  begin
   xx[-mi[sq]]:=-1;
   mi[sq]:=0;
   sq:=sq-1;
   go to backtr
  end mi[sq]<0
  else
  begin
   xx[mi[sq]]:=0;
   mi[sq]:=-mi[sq];
   go to ntrivs
  end mi[sq]≥0;
END:
```

```
        f:=zd;
        for j:=1 step 1 until n do
         x[j]:=xd[j];
      exit:
       end ilp01SW;
```

2. Method used. The procedure *ilp01SW* is an implementation of an implicit enumeration method for zero-one linear programming problems and incorporates some of the ideas proposed by Schrage and Woiler in [4]. The general outline of the algorithm will be presented below, and the details may be looked up by inspecting the body of *ilp01SW*.

The already classical backtracking procedure, as applied by Glover [3], Cylkowski and Kucharczyk [2], and others, is used in the algorithm. For enumeration, two vectors $\boldsymbol{\mu} = (\mu_1, \mu_2, \ldots, \mu_n)$ and $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ are provided. $\boldsymbol{\mu}$ is the proper enumeration vector indicating the enumeration stage, and $\boldsymbol{x}$ contains the partial solution corresponding to $\boldsymbol{\mu}$. If at some computation stage the partial solution is composed of s variables with indices $j_1, j_2, \ldots, j_s$, selected in that order, then

$$\mu_k = \begin{cases} j_k, & \text{if } x_{j_k} = 1 \text{ and its complement has not yet been considered,} \\ -j_k, & \text{if } x_{j_k} = 0 \text{ or } x_{j_k} = 1, \text{ and their respective complements have already been considered,} \\ 0, & \text{otherwise, i.e. for } k > s. \end{cases}$$

The component x_k of $\boldsymbol{x}$ is equal to 0 or 1 if that value has been assigned in the partial solution to variable x_k, and is equal to -1 if variable x_k is a free variable, i.e. is not contained in the partial solution. While augmenting, new variables are selected to enter into the partial solution and the vectors $\boldsymbol{\mu}$ and $\boldsymbol{x}$ are suitably changed. While backtracking, the right most positive element of $\boldsymbol{\mu}$ is negated and any negative elements to the right of it are set equal to zero; also an appropriate change in $\boldsymbol{x}$ is performed. The enumeration is completed when all components of $\boldsymbol{\mu}$ are non-positive.

Procedure *ilp01SW* works as follows:

Notation: $\boldsymbol{x}^0$ – optimum solution vector, $\hat{\boldsymbol{x}}$ – vector of the so far best feasible solution, $\hat{z}$ – so far best value of the objective function.

Initialization:

1. Try if the trivial solution $(0, 0, \ldots, 0)$ is feasible; that is so if all b_i are non-positive. If so, set $f = 0$, $\boldsymbol{x}^0 = (0, 0, \ldots, 0)$ and exit.

2. If not, set $\boldsymbol{\mu} = (0, 0, \ldots, 0)$, $\boldsymbol{x} = (-1, -1, \ldots, -1)$, $s = 0$ and $\hat{z} = \infty$.

3. Calculate

$$\min_{1 \leqslant i \leqslant m} y_i = y_{i_0}, \quad \text{where } y_i = \sum_{k=1}^{s} a_{ij_k} x_{j_k} - b_i$$

determines whether constraint i is satisfied $(y_i \geqslant 0)$ or not $(y_i < 0)$.

4. Calculate $z = \sum_{k=1}^{s} c_{j_k} x_{j_k}$, the value of the objective function for the partial solution $\{j_1, j_2, \ldots, j_s\}$.

5. If $y_{i_0} \geqslant 0$, then a feasible solution has been found, and if, in addition, it is better, i.e. if $z < \hat{z}$, then set $\hat{z} = z$ and $\hat{\boldsymbol{x}} = \boldsymbol{x}$ (changing in $\boldsymbol{x}$ the components equal to -1 into 0). Go to backtracking.

Test 1 (used only when a feasible solution has already been found):

6. Check if $z + \min_{j \in F} c_j \geqslant \hat{z}$, where F is the set of free variables $F = \{1, 2, \ldots, n\} \setminus \{j_1, j_2, \ldots, j_s\}$. If so, go to backtracking.

7. Calculate $d = \sum_{k \in F, a_{i_0 k} > 0} a_{i_0 k}$, the sum of all positive coefficients for the free variables in the selected constraint i_0.

Test 2:

8. If $W = y_{i_0} + d < 0$, go to backtracking.

9. If $W = 0$, a multiple augmentation of the partial solution may be performed by setting $x_k = 1$ for $a_{i_0 k} > 0$ and $x_k = 0$ for $a_{i_0 k} \leqslant 0$. If the new value of z is better than $\hat{z}$, try if a new feasible solution has been found which then must be memorized, afterwards, and also otherwise, backtrack.

10. If $W > 0$, calculate $d^* = \sum_{k \in T} a_{i_0 k}$, where $T = \{j \mid j \in F,\ a_{i_0 j} > 0,\ z + c_j < \hat{z}\}$.

Test 3:

11. If $d^* = 0$, i.e. T is void, then go to backtracking.

Test 4:

12. If $W^* = y_{i_0} + d^* < 0$, then go to backtracking.

Test 5:

13. Calculate $\max_{k \in T} a_{i_0 k} = a_{i_0 j_0}$ and test if $W - a_{i_0 j_0} < 0$. If so, an augmentation of the partial solution may be made by assigning to x_{j_0} the value 1 and eliminating the complementary value $x_{j_0} = 0$. Afterwards go to step 3.

Test 6:

14. If $W^* - a_{i_0 j_0} < 0$, then the same augmentation of the partial solution as described in step 13 may be made, otherwise augment the partial solution by setting $x_{j_0} = 1$, yet not eliminating the value $x_{j_0} = 0$. Then go to step 3.

15. If during backtracking it is found that the enumeration process is at its end, assign $f = \hat{z}$ and $\boldsymbol{x}^0 = \hat{\boldsymbol{x}}$, and exit.

3. Certification. Procedure *ilp01SW* has been extensively tested on the Odra 1204 computer. Many examples have been solved, among them also those given by Balas in [1]. A forthcoming paper will give more detailed results of this experimentation concerning computer running times, number of iterations performed, efficacy of the tests, modifications, etc.

References

[1] E. Balas, *An additive algorithm for solving linear programs with zero-one variables* Operat. Res. 13 (1965) p. 517-546.

[2] Z. Cylkowski and J. Kucharczyk, *Solution of zero-one integer linear programming problems by Balas' method,* Zastosow. Matem. 11 (1969), p. 111-116.

[3] F. Glover, *A multiphase dual algorithm for the 0-1 integer programming problem,* Operat. Res. 13 (1965), p. 879-919.

[4] L. Schrage and S. Woiler, *A general structure for implicit enumeration,* Technical Report, Stanford University, Stanford, California 1967 (mimeographed).

DEPT. OF STATISTICS AND OPERATIONS RESEARCH
INSTITUTE OF ADMINISTRATIVE SCIENCES
UNIVERSITY OF WROCŁAW

Received on 11. 9. 1971

J. KUCHARCZYK (Wrocław) ALGORYTM 16

ALGORYTM ROZWIĄZYWANIA ZERO-JEDYNKOWYCH PROGRAMÓW LINIOWYCH METODĄ DEDUKCYJNEGO WYLICZANIA

STRESZCZENIE

Procedura *ilp01SW* rozwiązuje następujący zero-jedynkowy program liniowy metodą dedukcyjnego wyliczania:

$$\min f = \sum_{j=1}^{n} c_j x_j \qquad (c_j \geqslant 0),$$

gdy

$$\sum_{j=1}^{n} a_{ij} x_j \geqslant b_i \qquad (i = 1, 2, \ldots, m),$$

$$x_j = 0, 1 \qquad (j = 1, 2, \ldots, n).$$

Dane:

n — liczba zmiennych,
m — liczba ograniczeń,
$a[1 : m, 1 : n]$ — macierz współczynników przy ograniczeniach,
$b[1 : m]$ — prawe strony ograniczeń,
$c[1 : n]$ — współczynniki funkcji celu,
inf — największa dodatnia liczba całkowita.

Wyniki:

$x[1 : n]$ — rozwiązanie optymalne (tylko gdy ex = **true**, w przeciwnym razie procedura *ilp01SW* nie zmienia tablicy x),
f — optymalna wartość funkcji celu (tylko gdy ex = **true**, w przeciwnym razie procedura *ilp01SW* nie zmienia wartości f),
ex — **true**, gdy rozwiązanie optymalne istnieje, i **false**, gdy brak rozwiązania dopuszczalnego.

Inne parametry:

bl — etykieta, do której następuje skok z treści procedury, jeżeli którakolwiek z liczb $c[1 : n]$ jest ujemna.

Procedura *ilp01SW* wykorzystuje pomysły, zawarte w [4], i została sprawdzona na wielu przykładach na maszynie cyfrowej Odra 1204.