ALGORITHM 16

J. KUCHARCZYK (Wroclaw)

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:

\[ \min f = \sum_{j=1}^{n} c_j x_j \quad (c_j \geq 0), \]

provided

\[ \sum_{j=1}^{n} a_{ij} x_j \geq b_i \quad (i = 1, 2, \ldots, m), \]

\[ x_j = 0, 1 \quad (j = 1, 2, \ldots, n). \]

Data:

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

Results:

- \( x[I: n] \) — optimum solution (if \( ex = \text{true} \) only, otherwise the procedure *ilp01SW* does not change the array \( x \)),
- \( f \) — optimum value of the objective function (if \( ex = \text{true} \) only, otherwise the procedure *ilp01SW* does not change the value of \( f \)),
- \( ex \) — \text{true} if an optimum solution exists and \text{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 \( c[I: n] \) 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,d,W,Wg,d,eq,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:
mine:=0;
for i:=1 step 1 until m do
begin
  y := \text{\texttt{b}[i]};
  \textbf{for} j := 1 \textbf{step} 1 \textbf{until} n \textbf{do}
    \textbf{if} xx[j] > 0
      \textbf{then} y := y + \text{\texttt{a}[i,j]};
    \textbf{if} y < y_{\text{\texttt{min}}}
      \textbf{then}
      \textbf{begin}
        y_{\text{\texttt{min}}} := y;
        I := i
      \textbf{end}
      \textbf{y < y_{\text{\texttt{min}}}}
    \textbf{end}
  \textbf{1};
  z := 0;
  \textbf{for} j := 1 \textbf{step} 1 \textbf{until} n \textbf{do}
    \textbf{if} xx[j] > 0
      \textbf{then} z := z + c[j];
    \textbf{if} y_{\text{\texttt{min}}} = 0 \land z < zd
      \textbf{then}
      \textbf{feas:}
      \textbf{begin}
        zd := z;
        ex := \text{\texttt{true}};
        \textbf{for} j := 1 \textbf{step} 1 \textbf{until} n \textbf{do}
          xd[j] := \text{\texttt{if} xx[j] > 0 \textbf{then} 1 \textbf{else} 0},
          \textbf{go to} backtr
      \textbf{end}
      \textbf{y_{\text{\texttt{min}}} = 0 \land z < zd};
    \textbf{if} zd < \text{\texttt{inf}}
      \textbf{then}
      \textbf{test1:}
      \textbf{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 \( xx[j] \leq 0 \)
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 \( xx[j] \leq 0 \)
then
begin
\( y := a[I,j]; \)
if \( y > 0 \)
then
begin
if \( z+c[j] < z_d \) then
begin
begin
\( d := d + y \).
if \( m_{ine} < y \) then
begin
mine := y;
J := j
end
end
end
end
end
\( y > 0 \)
end
j:
test3:
if \( d = 0 \) then go to backtr;
Wg := y_{min} + d;
test4:
if \( Wg < 0 \) then go to backtr;
test5:
if \( W - m_{ine} < 0 \) then
begin
begin
augm:
\( sq := sq + 1 \);
mi[sq] := J;
xx[J] := 1;
Algorithm 16

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;
end Wg-mine>0;
end:

backtr:
if sq=0
then go to END;
if mi[sq]<0
then
begin
xx[-mi[sq]]:=-1;
mi[sq]:=0;
sq:=sq-1;
end mi[sq]<0
else
begin
xx[mi[sq]]:=0;
mi[sq]:=mi[sq];
go to ntrivs
end mi[sq]>0;
end:

END:
\( f := zd; \)
\[
\text{for } j := 1 \text{ step } 1 \text{ until } n \text{ do } \\
\quad x[j] := xd[j];  \\
\text{exit;}
\]
\end{ilp01SW;}

2. Method used. The procedure \( \text{ilp01SW} \) is an implementation of an implicit enumeration method for zero-one linear programming problems and incorporates some of the ideas proposed by Sehrage 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 \( \text{ilp01SW} \).

The already classical backtracking procedure, as applied by Glover [3], Cyklowski and Kucharecyk [2], and others, is used in the algorithm. For enumeration, two vectors \( \mu = (\mu_1, \mu_2, \ldots, \mu_n) \) and \( x = (x_1, x_2, \ldots, x_n) \) are provided. \( \mu \) is the proper enumeration vector indicating the enumeration stage, and \( x \) contains the partial solution corresponding to \( \mu \). 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 \( x \) 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 \( \mu \) and \( x \) are suitably changed. While backtracking, the right most positive element of \( \mu \) is negated and any negative elements to the right of it are set equal to zero; also an appropriate change in \( x \) is performed.

The enumeration is completed when all components of \( \mu \) are non-positive.

Procedure \( \text{ilp01SW} \) works as follows:

Notation: \( \mathbf{x}^o \) — optimum solution vector, \( \mathbf{x} \) — 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 \( f = 0, \mathbf{x}^o = (0, 0, \ldots, 0) \) and exit.
2. If not, set \( \mu = (0, 0, \ldots, 0), \mathbf{x} = (-1, -1, \ldots, -1), s = 0 \) and \( \hat{z} = \infty \).
3. Calculate
\[ \min_{1 \leq i \leq m} y_i = y_{i_0}, \quad \text{where} \quad y_i = \sum_{k=1}^{s} a_{ik} x_k - b_i \]
determines whether constraint \( i \) is satisfied (\( y_i \geq 0 \)) or not (\( y_i < 0 \)).

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

5. If \( y_{i_0} \geq 0 \), 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 \( \hat{x} = x \) (changing in \( x \) 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 \( z + \min_{j \in F} c_j \geq \hat{z} \), 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 \( d = \sum_{k \in F, a_{ik} > 0} a_{ik} \), 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 \( a_{ik} > 0 \) and \( x_k = 0 \) for \( a_{ik} \leq 0 \). 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 \( d^* = \sum_{k \in T} a_{ik} \), where \( T = \{j \mid j \in F, a_{ij} > 0, z + c_j < \hat{z}\} \).

**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 \( \max_{k \in T} a_{ik} = a_{i_0 j_0} \) 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 $f = \tilde{z}$ and $x^0 = \tilde{x}$, and exit.

3. Certification. Procedure \textit{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


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

Received on 11. 9. 1971

J. KUCHARCZYK (Wrocław)

ALGORYTM ROZWIĄZYWANIA ZERO-JEDYNKOWYCH PROGRAMÓW LINIOWYCH METODĄ DEDUKCYJNEGO WYLCZANIA

STRESZCZENIE

Procedura \textit{ilp01SW} rozwiązuje następujący zero-jedynkowy program liniowy metodą dedukcyjnego wyliczania:

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

gdy

$$\sum_{j=1}^{n} a_{ij} x_j \geq h_i \quad (i = 1, 2, \ldots, m),$$

$$x_j = 0, 1 \quad (j = 1, 2, \ldots, n).$$
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 \( ex = \text{true} \), w przeciwnym razie procedura \( ilp01SW \) nie zmienia tablicy \( x \)),
\( f \) — optymalna wartość funkcji celu (tylko gdy \( ex = \text{true} \), w przeciwnym razie procedura \( ilp01SW \) nie zmienia wartości \( f \)),
\( ex \) — \text{true}, gdy rozwiązanie optymalne istnieje, i \text{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.