ZASTOSOWANIA MATEMATYKI
APPLICATIONES MATHEMATICAE
XV, 4 (1977)

ALGORITHM 49

A. ADRABIŃSKI and J. GRABOWSKI (Wrocław)

AN ALGORITHM FOR SOLVING THE MACHINE SEQUENCING PROBLEM

1. Procedure declaration. The procedure *OPTKOLOBR* finds a sequence of machine operations such that the total time spent for processing all operations is minimal. A detailed description of the mathematical model is given in papers [3] and [4].

Data:

k — number of machines;

l — assumed number of iterations of the algorithm;

mc — number of operations, it is equal to $\sum_{i=1}^{k} N_i$;

t — number of ordered pairs of operations corresponding to the technological requirements (see [3]);

bool — Boolean variable with value **true** if an initial solution is given and **false** otherwise;

$cT[1:mc]$ — array of operation times; the array contains, in locations $1, 2, \ldots, N[1]$, the operation times of machine No. 1, and in locations $N[i-1]+1, \ldots, N[i]$ those of machine No. i ($i = 2, 3, \ldots, k$);

$N[1:k]$ — integer array containing in $N[i]$ the number of operations which are to be carried out on machine i;

$RTP, RTK[1:t]$ — integer arrays containing the ordered pairs of operations corresponding to the technological requirements of operation order RT (see [3] and [4] for details); the predecessors and successors are contained in the arrays RTP and RTK, respectively;

$SP, SK[1:ret]$ — integer arrays containing the pairs of operations of the initial solution (see [4]);

INF — maximum positive number of real type.

Results:

$LNC[1:mc]$ — array of the earliest starting times of operations for the optimal operation sequence;

```
procedure OPTKOLOBR(k,l,mc,t,bool,cT,N,P,RTP,RTK,SP,SK,LNC,
 LRG,Lg,FIN,INF);
 value k,mc,cT;
 integer k,l,mc,t;
 real Lg,INF;
 integer array N,RTP,RTK,SP,SK;
 array P,cT,LNC,LRG;
 Boolean bool;
 label FIN;
 begin
  integer i,j,m,m1,md,n,p,puj,r,s,R,APj,AKj,APzj,AKzj,APpj,
  AKpj,Rob1,Rob2;
  real max,M;
  Boolean Fj,zFj,I;
  integer array B[1:mc];
  APj:=R:=md:=0;
  for i:=1 step 1 until k do
   begin
    p:=N[i];
    APj:=APj+p;
    md:=p×(p-1)+md
   end i;
  APj:=APj-k;
  md:=md+2;
  m:=mc+mc;
  n:=m+2;
  for i:=1 step 1 until mc do
   begin
    for j:=1 step 1 until t do
     if RTP[j]=i
```

```
        then go to E;
      R:=R+1;
      B[R]:=i;
E:  end i;
    p:=t+R+md;
    m:=m+p;
    m1:=m+1;
    begin
     integer array AKT,APT,AK,AP[1:m],pu[1:m1],AKz,APz,AKp,
     APp,Sgp,Sgk[1:md],C[1:n],A,DLT,H,HU,Kp[0:1],DLT1[1:1],F[
     1:1,1:md];
     array Lox,Loxpr,Lxz,Lxzpr[1:n],dlt[1:1],c[1:m];
     procedure CPM(pu,AP,AK,Lox);
      array Lox;
      integer array pu,AP,AK;
      begin
       integer i,j,k,u,APj,AKj,puj;
       real max;
       Booleah array sw[1:n],su[1:m];
       u:=0;
       for i:=1 step 1 until n do
        begin
         sw[i]:=su[i]:=true;
         pu[i]:=0
        end i;
       for i:=n+1 step 1 until m do
        begin
         su[i]:=true;
         pu[i]:=0
        end i;
```

```
for k:=1 step 1 until n do
 begin
  for i:=1 step 1 until n do
   if sw[i]
    then
    begin
     for j:=1 step 1 until m do
      if AK[j]=i
       then
       begin
        if sw[AP[j]]
         then go to nexti
         else
          if su[j]
           then
           begin
            u:=u+1;
            su[j]:=false;
            pu[u]:=puj:=j;
            max:=Lox[AP[puj]];
            if puj>p
             then max:=max+c[puj];
            AKj:=AK[puj];
            if I
             then
             begin
              if max>Lox[AKj]
               then
               begin
                Loxpr[AKj]:=Lox[AKj];
```

```
                    Lox[AKj]:=max;
                    C[AKj]:=puj
                   end max>Lox[AKj]
                   else
                    if max>Loxpr[AKj]
                     then Loxpr[AKj]:=max;
                  go to E1
                 end I;
                if max>Lox[AKj]
                 then Lox[AKj]:=max;
E1:            end su[j]
            end AK[j]=i;
          sw[i]:=false;
          go to nextk;
nexti:   end sw[i];
nextk:end k
    end CPM;
   s:=0;
   if bool
    then go to Z4;
   puj:=0;
   for j:=1 step 1 until k do
    begin
     APj:=N[j];
     AKj:=APj-1;
     for i:=1 step 1 until AKj do
      begin
       r:=i+puj;
       for APzj:=i+1 step 1 until APj do
        begin
```

```
          s:=s+1;
          AKp[s]:=AKzj:=2×r;
          AP[s]:=APz[s]:=AKzj+1;
          AK[s]:=AKz[s]:=APpj:=2×(APzj+puj);
          APp[s]:=APpj+1;
          c[s]:=0.0
         end APzj
       end i;
      puj:=puj+APj
     end j;
    go to Z2;
Z4:puj:=APj;
   APj:=AKj:=APzj:=1;
   AKzj:=N[1];
Z3:for j:=AKj sten 1 until AKzj do
    begin
     for i:=1,i+1 while i≤puj do
      if SK[i]=j
       then go to Z1;
     APpj:=0;
     r:=N[APzj]-1;
     for i:=1,i+1 while APpj<r do
      if SP[i]=j
       then
       begin
        AKpj:=APj+APpj;
        Sgp[AKpj]:=j;
        Sgk[AKpj]:=j:=SK[i];
        APpj:=APpj+1;
        i:=0
```

```
  end i;
for i:=1 step 1 until APpj do
 begin
  s:=s+1;
  AKpj:=i+APj-1;
  AP[s]:=APz[s]:=2×Sgp[AKpj]+1;
  AK[s]:=AKz[s]:=2×Sgk[AKpj];
  APp[s]:=AK[s]+1;
  AKp[s]:=AP[s]-1;
  r:=0;
  c[s]:=0.0;
  for r:=r+1 while r≤APpj-i do
   begin
    s:=s+1;
    AP[s]:=APz[s]:=Rob1:=AP[s-1];
    AK[s]:=AKz[s]:=Rob2:=2×Sgk[AKpj+r];
    APp[s]:=Rob2+1;
    AKp[s]:=Rob1-1;
    c[s]:=0.0
   end r
 end i;
AKpj:=APzj;
APzj:=APzj+1;
if APzj>k
 then go to Z2;
AKpj:=N[AKpj];
AKj:=AKj+AKpj;
APj:=APj+AKpj-1;
AKzj:=AKzj+N[APzj];
go to Z3;
```

```
Z1: end j;
Z2:for i:=1 step 1 until t do
    begin
     s:=s+1;
     APT[s]:=AP[s]:=2×RTP[i]+1;
     AKT[s]:=AK[s]:=2×RTK[i];
     c[s]:=0.0
    end i;
   for i:=1 step 1 until R do
    begin
     s:=s+1;
     APT[s]:=AP[s]:=2×B[i]+1;
     AKT[s]:=AK[s]:=n;
     c[s]:=0.0
    end i;
   for i:=1 step 1 until mc do
    begin
     s:=s+1;
     APT[s]:=AP[s]:=puj:=2×i;
     AKT[s]:=AK[s]:=puj+1;
     c[s]:=cT[i]
    end i;
   puj:=0;
   for i:=1 step 1 until k do
    begin
     M:=P[i];
     APj:=N[i];
     for j:=1 step 1 until APj do
      begin
       s:=s+1;
```

```
        APT[s]:=AP[s]:=1;
        AKT[s]:=AK[s]:=2×(j+puj);
        c[s]:=M
       end j;
      puj:=puj+APj
     end i;
    H[1]:=HU[1]:=pu[m1]:=0;
    r:=s:=1;
    Lg:=INF;
    for j:=1 step 1 until md do
     begin
      F[1,j]:=0;
      Sgp[j]:=APz[j];
      Sgk[j]:=AKz[j]
     end j;
krok1:
    for j:=1 step 1 until md do
     begin
      Rob1:=F[r,j];
      Fj:=Rob1=0;
      zFj:=Rob1=1;
      APT[j]:=if Fj then 0 else if zFj then APz[j] else APp[
              j];
      AKT[j]:=if Fj then 0 else if zFj then AKz[j] else AKp[
              j]
     end j;
    for j:=2 step 1 until n do
     Lox[j]:=-1.0;
    Lox[1]:=0.0;
    CPM(pu,APT,AKT,Lox);
```

```
if Lox[n]≥Lg
 then
 begin
  A[r]:=A[r-1];
  go to krok4
 end Lox[n]≥Lg;
for j:=1 step 1 until n do
 begin
  Lox[j]:=Loxpr[j]:=Lxz[j]:=Lxzpr[j]:=-1.0;
  C[j]:=0
 end j;
Lox[1]:=Lxz[n]:=0.0;
I:=true;
CPM(pu,AP,AK,Lox);
I:=false;
for j:=m step -1 until 1 do
 begin
  puj:=pu[j];
  APj:=AP[puj];
  max:=Lxz[AK[puj]];
  if puj>p
   then max:=max+c[puj];
  if max>Lxz[APj]
   then
   begin
    Lxzpr[APj]:=Lxz[APj];
    Lxz[APj]:=max
   end max>Lxz[APj]
   else
    if max>Lxzpr[APj]
```

```
      then Lxzpr[APj]:=max
  end j;
max:=Lox[n];
if Lg>max
 then
 begin
  Lg:=max;
  for j:=1 step 1 until md do
   begin
    Sgp[j]:=AP[j];
    Sgk[j]:=AK[j]
   end j;
  R:=r
 end Lg>max;
A[0]:=0;
APj:=Kp[r]:=i:=0;
puj:=C[n];
for j:=0 while APj≠1 do
 begin
  if puj≤md
   then
   begin
    if F[r,puj]=0
     then
     begin
      i:=i+1;
      APzj:=APz[puj];
      AKzj:=AKz[puj];
      APpj:=APp[puj];
      AKpj:=AKp[puj];
```

```
            AKj:=A[r-1]+i;
            if AKj=1
             then go to koniec1;
            DLT1[AKj]:=puj;
            dlt[AKj]:=Loxpr[AKzj]+Lxzpr[APzj]-Lox[AKpj]-Lxz[
                      APpj]
           end F[r,puj]=0
         end puj≤md;
        APj:=AP[puj];
        puj:=C[APj]
       end j;
      Kp[r]:=i;
      APzj:=0;
      A[r]:=A[r-1]+i;
      if i=0
       then go to krok4;
  krok3:
      puj:=Kp[r];
      if puj=0
       then go to krok4;
      M:=INF;
      s:=s+1;
      for j:=1 step 1 until md do
       F[s,j]:=F[r,j];
      APj:=A[r];
      for j:=A[r-1]+1 step 1 until APj do
       if dlt[j]≠INF
        then
        begin
         max:=dlt[j];
```

```
          if max<M
           then
           begin
            M:=max;
            i:=DLT1[j];
            DLT[r]:=j
           end max<M
         end j;
       F[s,i]:=-1;
       AP[i]:=APp[i];
       AK[i]:=AKp[i];
       H[s]:=r;
       HU[s]:=i;
       r:=s;
       go to krok1;
krok4:
       puj:=H[r];
       if puj=0
        then go to koniec2
       i:=HU[r];
       r:=puj;
       puj:=Kp[r];
       Kp[r]:=puj-1;
       dlt[DLT[r]]:=INF;
       F[r,i]:=1;
       AP[i]:=APz[i];
       AK[i]:=AKz[i];
       go to krok3;
koniec1:
       APzj:=-1;
```

```
koniec2:
   for j:=1 step 1 until md do
    begin
     AP[j]:=Sgp[j];
     AK[j]:=Sgk[j];
    end j;
   for j:=1 step 1 until n do
    Lox[j]:=Lxz[j]:=-1.0;
   Lox[1]:=Lxz[n]:=0.0;
   CPM(pu,AP,AK,Lox);
   for j:=m step -1 until 1 do
    begin
     puj:=pu[j];
     APj:=AP[puj];
     max:=Lxz[AK[puj]];
     if puj>p
      then max:=max+c[puj];
     if max>Lxz[APj]
      then Lxz[APj]:=max
    end j;
   for j:=1 step 1 until mc do
    begin
     puj:=p+j;
     APj:=AP[puj];
     AKj:=AK[puj];
     M:=Lox[APj];
     max:=M+c[puj];
     LRG[j]:=Lg-max-Lxz[AKj];
     LNC[j]:=M
    end j;
```

```
if APzj=-1
 then go to FIN
end
end OPTKOLOBR
```

LRG[*1*:*mc*] — array of total time reserves in the optimal operation sequence;
Lg — value of the total time spent for processing all operations;
FIN — label outside of the body of the procedure *OPTKOLOBR* to which an exit is made if the assumed number of iterations *l* is smaller than that required by the algorithm.

Remarks.

(1) If *bool* ≡ **false**, then the initial solution is generated by the procedure *OPTKOLOBR*; the method has been presented in paper [2].

(2) *ret* is the number of disjunctive arcs of the initial solution. It is not a parameter of the procedure *OPTKOLOBR*. This number may be calculated as follows:

$$ret = \sum_{i=1}^{k} N_i - k \quad \text{or} \quad ret = mc - k.$$

The number of iterations of the algorithm has to satisfy the inequality

$$l \leqslant \frac{K - 8t - 14\,md - 45\,mc - 60}{md + 8},$$

where parameters t, l, mc are as described above,

$$md = \frac{1}{2} \sum_{i=1}^{k} N_i(N_i - 1),$$

and K is the number of memory cells which can be used by the parameters of the procedure.

2. Method used. The algorithm uses the method from papers [3] and [4] finding a minimaximal path in the disjunctive graph. The test step of the algorithm is based on paper [2].

3. Certification. The procedure *OPTKOLOBR* has been verified on the ODRA 1204 computer with 16K core memory for many examples. For all examples it has started from the initial solution generated inside the procedure *OPTKOLOBR*. The computational results are listed in the following table:

mc	*k*	*l*	Number of iterations performed by the procedure *OPTKOLOBR*	*Lg*	Time (in sec)	Source of example
6	2	100	12	31	22	Balas [1]
8	3	100	16	55	41	Fig. 1 (1)
13	4	100	25	13	175	Balas [1]
15	5	100	3	13	36	Fig. 2
22 (2)	5	110	110	408	1642	Fig. 3
25 (2)	5	90	90	395	1272	Fig. 4
30 (2)	3	30	30	267	206	Fig. 5

(1) In Figs. 1-5, numbers above arrows denote the operation number, and those below arrows denote the processing time of operation.

(2) The solution is suboptimal (the assumed number of iterations *l* has been exceeded).

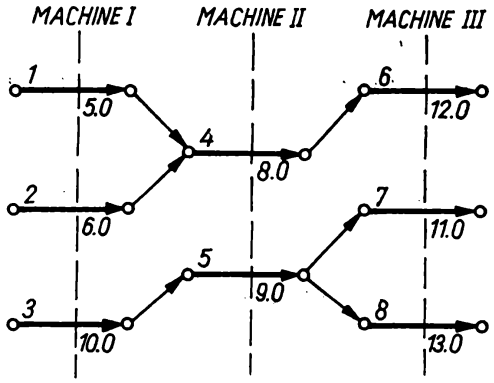

Fig. 1

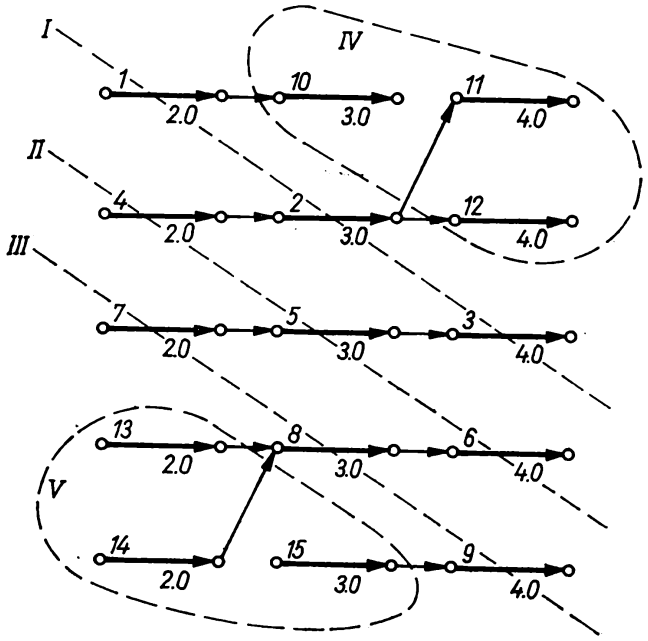

Fig. 2

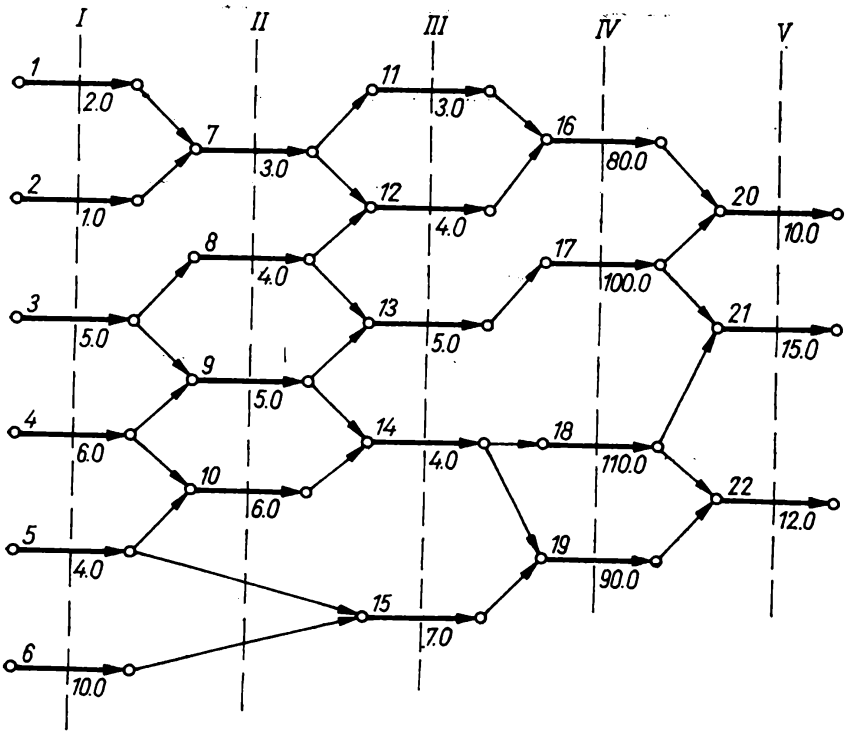
I
II
III
IV
V
1
2.0
2
1.0
3
5.0
4
6.0
5
4.0
6
10.0
7
3.0
8
4.0
9
5.0
10
6.0
11
3.0
12
4.0
13
5.0
14
4.0
15
7.0
16
80.0
17
100.0
18
110.0
19
90.0
20
10.0
21
15.0
22
12.0

Fig. 3

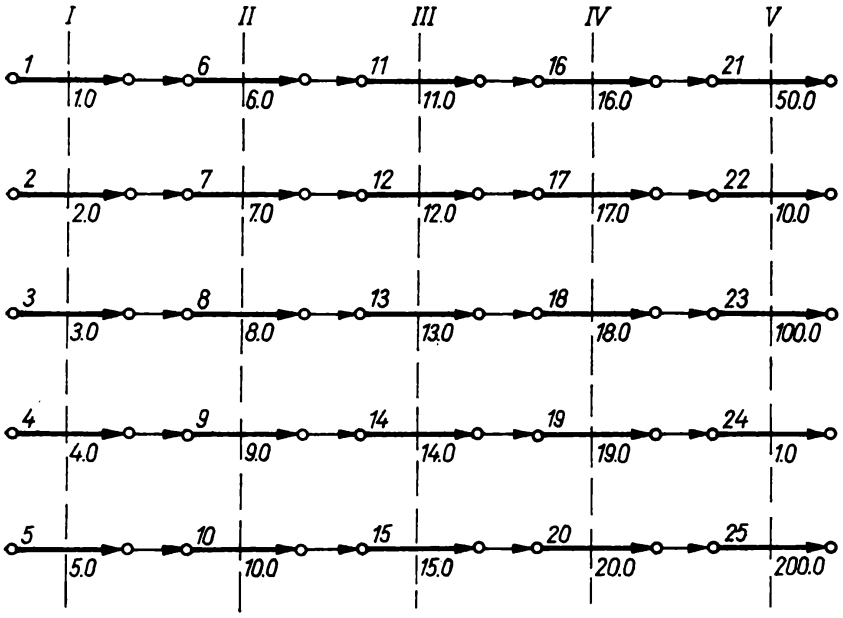
I
II
III
IV
V
1
1.0
6
6.0
11
11.0
16
16.0
21
50.0
2
2.0
7
7.0
12
12.0
17
17.0
22
10.0
3
3.0
8
8.0
13
13.0
18
18.0
23
100.0
4
4.0
9
9.0
14
14.0
19
19.0
24
1.0
5
5.0
10
10.0
15
15.0
20
20.0
25
200.0

Fig. 4

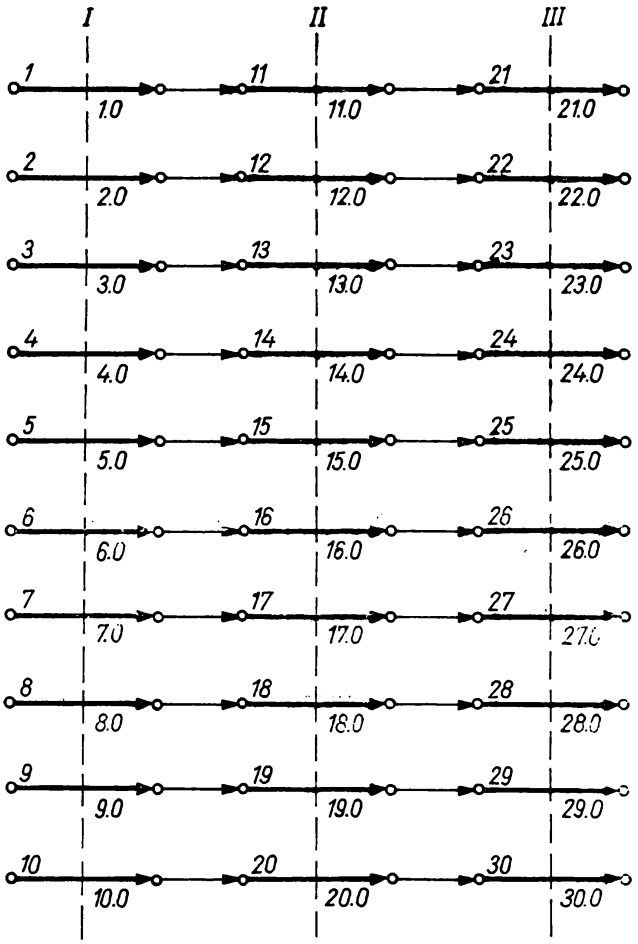

Fig. 5

References

[1] E. Balas, *Discrete programming by the filter method*, Opns. Res. 15 (1976), p. 915-967.

[2] — *Machine sequencing via disjunctive graphs: an implicit enumeration algorithm*, ibidem 17 (1969), p. 941-957.

[3] J. Grabowski, *A new formulation and solution of the sequencing problem: Mathematical model*, Zastosow. Matem. 15 (1976), p. 325-343.

[4] — *A new formulation and solution of the sequencing problem: Algorithm*, this fascicle, p. 463-474.

Received on 26. 1. 1976

ALGORYTM 49

A. ADRABIŃSKI i J. GRABOWSKI (Wrocław)

ALGORYTM ROZWIĄZANIA ZAGADNIENIA KOLEJNOŚCIOWEGO

STRESZCZENIE

Procedura *OPTKOLOBR* znajduje minimaksymalną drogę w grafie dysjunktywnym.

Dane:

k — liczba różnych maszyn;
l — liczba iteracji algorytmu;
mc — liczba operacji;
t — liczba par operacji, wyrażających wymagania technologiczne porządku operacji;
bool — zmienna logiczna; ma ona wartość **true**, jeżeli istnieje rozwiązanie początkowe, i **false** w przeciwnym wypadku;
cT[*1*:*mc*] — tablica czasów wykonania operacji; *cT*[*1*] ÷ *cT*[*N*[*1*]] — czasy wykonania operacji na maszynie 1; *cT*[*N*[*1*]+*1*] ÷ *cT*[*N*[*2*]] — czasy wykonania operacji na maszynie 2; itd.;
N[*1*:*k*] — tablica liczb operacji; *N*[*i*] — zawiera liczbę operacji, które będą wykonane na maszynie *i*;
RTP, *RTK*[*1*:*t*] — tablice zawierające pary operacji, które wyrażają wymagania technologiczne porządku operacji *RT*; tablica *RTP* zawiera numery poprzedników każdej pary operacji, tablica *RTK* zaś numery następników każdej pary operacji;
SP, *SK*[*1*:*ret*] — tablice zawierające pary operacji, które wyrażają rozwiązanie początkowe (patrz [4]);
INF — maksymalna dodatnia liczba typu rzeczywistego.

Wyniki:

LNC[*1*:*mc*] — tablica najwcześniejszych momentów rozpoczęcia operacji dla optymalnej kolejności ich wykonania; *LNC*[*j*] — najwcześniejszy moment rozpoczęcia operacji *j*;
LRG[*1*:*mc*] — tablica ogólnych rezerw czasów wykonania operacji; *LRG*[*j*] — wartość ogólnej rezerwy czasu operacji *j*;
Lg — wartość całkowitego czasu obróbki wszystkich operacji;
FIN — etykieta skoku z procedury *OPTKOLOBR*, jeżeli liczba iteracji *l* jest mniejsza niż liczba wymagana przez algorytm.

Uwagi:

(1) Jeżeli *bool* ≡ **false**, rozwiązanie początkowe generowane jest wewnątrz procedury *OPTKOLOBR*; metoda otrzymywania tego rozwiązania podana jest w pracy [2].

(2) *ret* jest liczbą łuków dysjunktywnych rozwiązania początkowego. Nie występuje jako parametr procedury. Można ją wyznaczyć następująco:

$$ret = \sum_{i=1}^{k} N_i - k \quad \text{lub} \quad ret = mc - k.$$

Liczba iteracji algorytmu powinna spełniać nierówność

$$l \leqslant \frac{K - 8t - 14md - 45mc - 60}{md + 8},$$

gdzie parametry t, l, mc mają znaczenie jak opisano powyżej,

$$md = \frac{1}{2} \sum_{i=1}^{k} N_i(N_i - 1),$$

a K jest liczbą komórek pamięci, które mogą być używane przez parametry procedury.

Obliczenia kontrolne, wykonane na maszynie cyfrowej ODRA 1204, wykazały poprawność procedury *OPTKOLOBR*.