Оптимальное планирование
1969 выпуск 12

УДК 512.25 + 519.3

Т.В. ЭЙХЛЕР

АЛГОРИТМЫ УПОРЯДОЧЕНИЯ СЕТИ РАБОТ И ВЫЧИСЛЕНИЯ СЕТЕВЫХ ГРАФИКОВ

В этом сообщении излагаются два алгоритма, написанные на языке АЛГОЛ-60. Первый из них использует порядок на сети и содержит в себе соответственно переработанные процедуры сортировки [1, 3] и вычисления критического пути [4]. После упорядочения вершин сами работы линейно упорядочиваются методом Шелла [5], эффективность которого была отмечена, например, в [6]. Второй алгоритм является новым и рассчитан для применения в том случае, когда сеть ещё не построена (и вершины не определены), но для каждой работы имеется набор других (ей предшествующих) работ. При наличии контура оба алгоритма выявляют цепочку работ, образующих его. Рассмотрим комплекс работ (ориентированный граф без контуров), заданный в виде списка:

integer array i,j,d[1:n] (1)

где i[a] - начало, j[a] - конец и d[a] - продолжительность (длина) работы (дуги) a . допустим, что разных вершин всего ne*) и что имеются единственные i0 - начало сети (вершина без входящих дуг), j0 - конец сети (вершина без выходящих дуг). Определим te[1:ne] - прямой сетевой график:

$$te[i0]=0,\quad te[j]=\max_{a\in\omega^-(j)}\{te[i[a]]+d[a]\}^{**)} \qquad (2)$$

- длина самого длинного пути, оканчивающего вершиной j ,

*) Предполагается, что вершины пронумерованы в каком-то порядке от 1 до ne. Иначе их надо соответственно переименовать.

**) Через $\omega^+(\kappa)$ или $\omega^-(\kappa)$ обозначены множества дуг, имеющих вершину κ своим началом или концом.

т.е. самое ранее возможное начало всех работ, исходящих из j . Аналогично определяется tl[I:ne] -обратный сетевой график:

$$tl[j0]=te[j0], \qquad tl[i]=\min_{a\in\omega^+(i)}\{tl[j[a]]-d[a]\} \qquad (3)$$

- самое позднее допустимое окончание входящих в i работ, если весь комплекс должен быть завершён в кратчайший срок. Сетевые графики дают нам для любой работы a : es[a]=te[i[a]] её самое ранее возможное начало, ls[a]=tl[j[a]]-d[a] её самое позднее допустимое начало , ef[a]=te[i[a]]+d[a] её самое ранее возможное окончание, lf[a]=tl[j[a]] её самое позднее допустимое окончание, tf[a]=ls[a]-es[a] её полный резерв времени, ff[a]=te[j[a]]-ef[a] её свободный резерв времени. Если tf[a]=0 , то замедление работы a влечет за собой и задержку завершения всего комплекса*). Назовём такие работы критическими. Путь от i0 до j0 , состоящий только из критических работ, называется критическим. Вычислим сетевые графики по методу Форда [2] . Сначала для всех i положим te[i]:=0. Потом в цикле для всех работ подряд каждый раз, когда

$$te[j[a]] < t=te[i[a]]+d[a], \qquad (5)$$

положим te[j[a]] :=t . Как только неравенство (5) не будет иметь место ни для какой работы, вычисление te[I:ne] ,согласно формуле (2), заканчивается. Теперь, положив для всех j сначала tl[j]:=te[j0] и потом, снова в цикле, каждый раз, когда

$$tl[i[a]]>t=tl[j[a]]-d[a], \qquad (6)$$

положив tl[i[a]] :=t , в конце концов получим, согласно формуле (3), tl[I:ne] . Заметим, что исходная информация (I) никак не упорядочивалась. Но от степени беспорядка в её расположении зависит трудоёмкость (число повторений цикла) вычисления. Для того, чтобы вычисление te(tl) закончилось после первого просмотра всех дуг, необходимо и достаточно, чтобы перед рассмотрением очередной дуги были рассмотрены все предшествующие ей (следующие за ней) дуги. Допустим, что для любого a будет i[a]<j[a] и что дуги упорядочены по возрастанию их начал .

*) Ускорение критической работы не всегда уменьшает te[j0].

Докажем от противного, что после одного просмотра всех дуг в этом порядке неравенство (5) уже не будет иметь места ни для какой дуги. Пусть имеется такая дуга b , что te[i[b]]> >te[j[b]]- d[a]. Значит, существует такое $\bar{a}>b$, что $j[\bar{a}]=i[b]$. Тогда $i[\bar{a}]<i[b]$, но отсюда следует $\bar{a}<b$, что доказывает наше утверждение. То же самое верно и для (6), если вычисление tl ведётся в обратном порядке. Вместе с предварительным упорядочением удобно проверять наличие контура в списке (I). Назовем рангом вершины i число вершин в самом длинном пути от 10 до i . Для вычисления рангов может быть применён тот же метод Форда [3] . Допустим, что после некоторого шага вычислений получен ранг bt и найдено $k\in\{1,2,\ldots,bt-1\}$, которому не соответствует ни один ранг. Тогда подграф, порожденный вершинами с рангом > k , обязательно содержит контур. Если контура нет, то, используя видоизмененный метод Шелла [5], можно упорядочить дуги по возрастанию рангов их начал. Приведем программу в виде процедуры на языке АЛГОЛ-60, которая выявляет контур (если такой имеется), упорядочивает сеть и вычисляет оба сетевых графика. Так как первоначальный список работ (I) будет многократно преобразован, введём ещё массив na[I:n] - наименования работ, сохраняющий соответствие списка (I) с упорядоченной сетью.

```
procedure pertsort (na,i,j,d,n,ne,inf,te,tl);
value n,ne,inf ; integer array na,i,j,d,te,tl; integer n,ne,inf;
 comment  Вводимым списком работ является  na,i,j,d[I:n],
а результатом  te,tl[I:ne]  - сетевые графики. n -число работ,  ne  - число событий, а  inf  - верхняя граница для длины критического пути;
begin integer a,b,t,bt; integer array e,r,c[I:ne];
 comment  e - список событий,  r  - соответствующие ранги, а  C
служит для обнаружения контура. Все события будут сначала переименованы натуральными числами от I до ne;
 e[I]:=i[I];  i[I]:=ne:=I;
 for a:=2 step I until n do
    begin for b:=I step I until ne do
          if i[a]=e[b]then begin i[a]:=b;go to rI end;
    ne:=ne+I;  e[ne]:=i[a];  i[a]:=ne;
 rI:end a;
 for a:=I step I until n do
     begin for b:=I step I until ne do
           if j[a]=e[b]then begin j[a]:=b; go to r2 end;
     ne:=ne+I; e[ne]:=j[a];  j[a]:=ne ;
```

```
r2: end a;
comment    Вычислим теперь r[I:ne]    , а если обнаружим
контур,то перейдем к метке   loop  ;
      bt:=I;for a:=I step I until ne do  r[a]:=c[a]:=I;
pass:  t:=0;
      For a:=I step I until n do
       begin if r[j[a]] > r[i[a]]   then go to fill;
        t:=0;      r[j[a]] :=r[i[a]] +I;
        if r[j[a]] > bt      then bt:=r[j[a]] ;
fill:  c[r[j[a]]]  :=c[r[i[a]]]  :=0
      end a;
    if t:=0 then begin for a:=I step I until bt do
                        if c[a]=I then go to loop
                                  else c[a]:=I;
                  go to pass
                 end t=0;
  comment   Теперь упорядочим события ;
      for a:=I step I until bt do
       for b:=I step I until ne do
        if r[b]=a then begin e[b]:=t;  t:=t+I end;
comment  Для любой работы  a   и  ,  теперь место e[i[a]]<e[j[a]];
      for a:=I step I until n d.
       begin t:=i[a] ;   i[a]:=n+e[t];
        if j[a]=ne then j[a]:=n+ne;
        for b:=I step I until n do
         if j[b]=t then j[b]:=i[a]
       end a;
      for a:=I step I until n do begin i[a]:=i[a]-n;
                                       j[a]:=j[a]-n
                                 end a;
        begin integer k,m,wi,wj,wd,wn;
comment       Этот блок упорядочивает список работ по  i ;
      for a:=I step a until n do m:=2×m-I;
      for m:=m÷2 while m≠0 do
       begin k:=n-m;
        for b:=I step I until k do
         begin wi:=i[b+m] ; wj:=j[b+m];
          wd:=d[b+m] ; wn:=na[b+m];
          for a:=b step -m until I do
           begin if wi ⩾ i[a]  then go to h ;
            i[a+m]:=i[a] ;  d[a+m] :=j[a] ;
```

```
        j[a+m]:=j[a] ;  na[a+m]:=na[a]
      end a;
  h:  i[a+m]:=wi;  j[a+m]:=wj;
      d[a+m]:=wd;  na[a+m]:=wn
    end b
   end m
  end sorting;
  for a:=I step I until ne do begin te[a]:=0;
                                    tl[a]:=inf
                                  end a;
  for a:=I step I until n do
   begin b:=te[i[a]] +d[a];
    if te[j[a]]>b  then tl[j[a]] :=b
   end a;
  tl[ne]:=te[ne];
  for a:=n step -I until I do
   begin b:=tl[j[a]] -d[a];
    if tl[i[a]]>b  then tl[i[a]] :=b
   end a;
loop: begin integer k; integer array c[I:n];
  comment  Если обнаружен контур, то этот блок выявляет его
  в виде последовательности работ,  b -ая и последняя из
  которых совпадают ;
      k:=0; for b:=I step I until n do
              if r[j[b]]>a  then go to p;
   p: k:=k+I;  c[k]:=bt:=t; t:=i[b];
      for b:=k-I step -I until I do
       if c[b]=bt then begin вывод (c,b); stop end;
      for b:=I step I until n do
       if j[b]=t∧r[j[b]]>a  then go to p
    end loop
   end pertsort
```

Представим, что составление списка работ, необходимых для достижения каких-то определенных целей, находится на самом первом этапе. Каждая работа обозначается своим порядковым номером i (порядок может быть случайным), и для неё, кроме продолжительности d_i, указано только множество A_i других (предшествующих) работ, завершение которых является достаточным условием для начала работы i. Отдельный человек, за-

полнявший i -ую строчку списка работ *)

$$< i , d_i , |A_i| , A_i > , \qquad (7)$$

может не знать отношений предшествования между элементами из A_i , и поэтому среди них могут быть лишние.

Пусть $N=\{1,2,\dots,n\}$ - множество работ и $\mathcal{A}=\{A_i \mid i=1,\dots,n\}$ - семейство подмножеств множества N . Будем говорить, что $L \vdash K$ тогда и только тогда, когда либо $K \in A_i$, либо существует $m \in N$ такое, что $L \vdash m$ и $m \vdash K$. Если ни для одного $m \in N$ не имеет место, что $m \vdash m$, то $(N, \vdash)$ - строго частично упорядоченное множество.

До сих пор предполагалось, что список задан в виде сети (I), т.е. введены какие-то события (вершины) и каждая работа представлена как пара таких вершин, т.е. как дуга сети. Поскольку такая пара может быть обозначением только одной дуги, то для устранения кратных дуг вводят ещё вспомогательные события и фиктивные работы. Для правильного изображения отношения предшествования между работами приходится также вводить большое число фиктивных работ. Понятно, что вследствие этого объем информации, необходимый для обработки данного комплекса, заметно растет. Причём пока что всё это делалось вручную. Ниже предлагается алгоритм, который сразу, исходя из списка (7), упорядочивает работы или выявляет возможный контур.

А. Найдем множество $M := \{ m \in N \mid A_m = \varnothing \}$, состоящее из всех минимальных элементов из N .

Б. Если M пусто, то переходим к Е .

В. Элементы из M запишем в любом порядке (они все одного ранга) .

Г. Положим $N := N \setminus M$; $\mathcal{A} := \mathcal{A} \setminus \{A_m \mid m \in M\}$ и для всех i положим $A_i := A_i \setminus M$.

Д. Если N непусто, то вернёмся к А. В противном случае сортировка закончена и переходим к Х .

Е. Всем $i \in N$ соответствуют непустые A_i - значит, имеется контур и надо его выявлять:

Возьмем $c_1 \in N$, $c_2 \in A_{c_1}$, $c_3 \in A_{c_2}$, ... ; контур будет иметь вид : $c_j \vdash \dots \vdash c_{j+p} = c_j$. Такие j, p существуют и не больше n . Записав этот контур, переходим к Х .

Х. Конец алгоритма.

---

*) Через $|X|$ обозначено число элементов множества X

Ниже приведенная программа на языке АЛГОЛ-60 реализует предложенный алгоритм и одновременно вычисляет (по работам) прямой сетевой график. Она не оформлена как процедура, потому что множества A_i вводятся в виде длинного массива $a[1:m]$ только после вычисления $m := \sum_{i=1}^{n} |A_i|$.

```
 begin integer n; ввод ( n );
comment n является числом всех работ, а массивы d,mi[1:n]
соответствуют значениям d_i и |A_i| из списка (7 );
 begin integer array d,mi[1:n]; ввод ( d,mi ) ;
 begin integer k,m; m:=0;
  comment После суммирования m:=Σ_{k=1}^{n} mi[k] вводится массив
   a[1:m], состоящий из элементов всех множеств A_i ;
  for k:=1 step 1 until n do m:=m+mi[k];
 begin integer array a[1:m]; ввод (a);
 begin integer i,r,t,w; integer array r1,r2,i,s,f[1:n];
  comment r1,r2[k] указывает место множества A_k в мас -
сиве a[1:m] , т.е. A_k=a[r1[k]:r2[k]]. i[1:n] оу-
дет упорядоченным списком работ, а s,f[t] являются соответ-
ственно самыми ранними началом и концом работы i[t] ;
      t:=1; w:=0; fork:=1 step 1 until n do
                   begin r1[k]:=w+1;
                    r2[k]:=w:=w+mi[k]
                   end k;
m1:   r:=t; for k:=1 step=1 until n do
            begin if r2[k]<0 then go to m2;
             for L:=r1[k] step 1 until r2[k] do
              if a[L]>0 then go to m2;
              w:=0;
             for L:=r1[k] step 1 until r2[k] do
              if a[i]<w then w:=a[i];
             i[t]:=k; s[t]:=-w; f[t]:=d[k]-w;
             r2[k]:=-1; t:=t+1;
      m2:   end k;
    if r=t then go to loop;
      for L:=r step 1 until t-1 do
       for k:=1 step 1 until n do
        if a[k]=i[L] then a[k]:=-f[L];
      if t<n then go to m1;
вывод (i,s,f);          stop;
loop: begin integer i,j; integer array c[1:n];
   comment Этот олок выявляет обнаруженный контур в виде
```

последовательности работ, j - ая и последняя из которых совпадают;

```
      i:=0;
      for L:=I step 1 until n do if r2[L]>0 then go to p;
p:    i:=i+I; c[i]:=k:=L;
      for j:=i-I step -1 until 1 do
       if c[j]=k then begin вывод (c,j); stop end ;
      for L:=rI[k] step I until r2[k] do
       if a[L]>0 then begin L:=a[L]; go to p end
    end loop
  end end end end end
```

По существу, работы в предложенном методе рассматриваются как вершины, а не как дуги некоторого ориентированного графа. После обработки списка (7) нашей программой можно уже по оси времени построить соответствующую сеть. Но и сразу, не построив сети, можно приступить к вычислению обратного сетевого графика и тем самым получить для каждой работы все данные (4). Оба алгоритма проверены с помощью транслятора ТА-1М. По ним составлены АЛЬФА-транслятором (с учетом его особенностей) рабочие программы.

Литература

1. I. Boothroyd. Algorithm 201 Shellsort, - "Comm.ACM", v.6, N 8, 1963, p. 445 .
2. Л.Р. Форд, Д.Р.Фалкерсон., Потоки в сетях. Изд-во "Мир", М., 1966.
3. R.H.Kase. Algorithm 219. Topological ordering for PERT Networks, -"Comm ACM", v.6, № 12, 1963, p. 739.
4. B.Leavenworth. Algorithm 40. Critical path scheduling, - "Comm. ACM", v. 4, № 3, 1961 p.952. Русский перевод. В сб. "Алгоритмы (1-50)" под ред. М.И. Агеева. Изд-во ВЦ АН СССР, М., 1966.
5. D.L.Shell. A high-speed sorting procedure, - "Comm. ACM", v.2, N 7, 1959, p. 30-32 .
6. Л.С.Лозинский Внутренняя сортировка информации при ограниченной памяти, - "Кибернетика", т.58,№ 3, 1965.

Поступила в редакцию
15.XI. 1967 г.