ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1983

Seria: INFORMATYKA z. 5 Nr kol. 759

Tadeusz Jan KUDŁA

POSZUKIWANIE PRZEKROJÓW MINIMALNYCH KOMPLEKSU

Streszczenie. Artykuł dotyczy algorytmu poszukującego minimalnych przekrojów kompleksu. Przedstawiona metoda tworzy przekroje minimalne kompleksu ograniczając się do tych przekrojów, które spełniają odpowiednie warunki. Pozostałe przekroje są eliminowane z rozważań, co zmniejsza angażowane do rozwiązania problemu zasoby maszynowe. W pracy rozważono tylko grafy silnie spójne - nie ogranicza to ogólności metody.

1. Wstęp

Dana jest sieć ($g \in (B^2)^A$, $f \in N^A$), gdzie g jest grafem o łukach ze zbioru A, którym przyporządkowano pary wierzchołków ze zbioru B. Funkcja f określa koszty rozcinania łuków $a \in A$, wyrażone za pomocą liczb naturalnych $f(a) \in N$. Zbiór rozciętych łuków rozrywających wszystkie cykle grafu nazywamy przekrojem grafu. Wśród przekrojów istnieją takie, które rozrywają cykle grafu przy minimalnym koszcie. Nazywamy je przekrojami minimalnymi.

Problem polega na znalezieniu rodziny przekrojów minimalnych i wyznaczeniu minimalnego kosztu decyklizacji grafu g. Jeżeli graf jest acykliczny, to rodzina jego przekrojów zawiera tylko przekrój pusty, a koszt rozcięcia wynosi 0. Gdy w grafie znajdują się pętle, to należą one do każdego przekroju grafu, a więc i do minimalnego. Jeżeli w grafie występują łuki równoległe między jakąś parą wierzchołków, to z punktu widzenia decyklizacji można je zastąpić jednym łukiem o sumarycznym koszcie rozcięcia. Z powyższych powodów, bez straty ogólności, można założyć, że mamy do czynienia z cyklicznym grafem prostym. W takim grafie istnieje podgraf $g' \in (B'^2)^{A'}$ ($A' \subset A$, $B' \subset B$, $g' = g|_{A'}$) o własnościach:

I) - dla dowolnej pary wierzchołków z B' istnieje łącząca je droga (łańcuch skierowany),

II) - nie istnieje podgraf $g'' \in (B''^2)^{A''}$ ($A'' \subset A$, $B'' \subset B$, $g'' = g|_{A''}$) różny od g i posiadający własność I oraz spełniający inkluzje $A' \subset A''$, $B' \subset B''$. Taki podgraf nazywać będziemy kompleksem[x]. Kompleks zbudowany

[x] Używa się też następujących określeń: fragment [6], zespół [3], maksymalny podgraf silnie spójny [6]; w literaturze angielskojęzycznej - maximal strongly connected component [7], irreducible subsystem [11]. Słowo kompleks zostało użyte w [2].

jest z cykli, graf może posiadać kilka kompleksów. Po skondensowaniu kompleksów do wierzchołków grafu graf staje się grafem acyklicznym. Decyklizacja grafu polega więc na decyklizacji jego kompleksów. Przekrój grafu jest sumą przekrojów jego kompleksów. Odtąd więc przez graf g rozumieć będziemy dowolny kompleks, dla którego poszukiwać będziemy przekrojów minimalnych. Idea metody jest wzięta z [2]. Tutaj przedstawiono jej pewne usprawnienia.

2. Opis metody

Kompleks g jest opisany za pomocą macierzy binarnej $G[g_{ij}]$

$$g_{ij} = \begin{cases} 1 - \text{jeżeli łuk } a_j \text{ należy do cyklu } c_i, \\ 0 - \text{w przeciwnym wypadku}, \end{cases} \tag{2.1}$$

$$i = 1,\dots, n: = \mathrm{card}(C),$$
$$j = 1,\dots, m: = \mathrm{card}(A),$$

gdzie C oznacza zbiór cykli kompleksu. Ponieważ cykl grafu wyznaczają jednoznacznie jego łuki, to odtąd przez $c \in C$ rozumieć będziemy zbiór łuków cyklu. Funkcja card określa liczbę elementów zbioru.

Przekrojem X^n kompleksu nazywamy zbiór łuków kompleksu, który z każdym cyklem posiada łuk wspólny

$$\bigwedge_{c \in C} c \cap X^n \neq \emptyset \tag{2.2}$$

Rozcięcie więc łuków przekroju X^n przekształca kompleks w graf acykliczny. Przekrój X^n można scharakteryzować wektorem binarnym z następująco:

$$z = \begin{bmatrix} z_1 \\ z_i \\ z_m \end{bmatrix}, \quad z_i = \begin{cases} 1 - \text{jeżeli łuk } a_i \in X^n, \\ 0 - \text{jeżeli łuk } a_i \notin X^n \end{cases} \tag{2.3}$$

gdy wektor charakterystyczny z^* określa przekrój minimalny, to zachodzi

$$\sum_i g_{ji}\, z^* \geqslant 1; \quad j = 1,\dots,n, \tag{2.4}$$

co oznacza rozcięcie wszystkich cykli, oraz

$$Fz^* = \min_z Fz, \quad F = \left[f(a_1),\dots,f(a_m)\right]. \tag{2.5}$$

Poszukiwanie przekroju minimalnego przeprowadza się iteracyjnie. W tym celu oznaczmy przez $X \subset A$ dowolny zbiór łuków, zaś przez $E(X)$ zbiór cykli, dla których X jest przekrojem, tj.

$$C \supset E(X) \ni c \Longleftrightarrow c \cap X \neq \emptyset \tag{2.6}$$

według tych oznaczeń mamy więc $C = E(X^n)$.

Niech $C^k = \{c_j\}_{j=1}^{k}$ jest zbiorem k cykli, a S_k rodziną wszystkich przekrojów dla cykli z C^k

$$S_k = \left\{X_1^k, \ldots, X_{p_k}^k : \quad E(X_i^k) \supset C_k, \quad i = 1, \ldots, p_k\right\} \tag{2.7}$$

Zasady budowy rodziny przekrojów S_k można przedstawić następująco:

I. $S_1 = \left\{\{a_1\}, \ldots, \{a_{p_1}\} : \ a_i \in c_1, \quad i = 1, \ldots, p_1; \quad p_1 = \mathrm{card}(c_1)\right\}$

Na $k-1$ etapie zbudowano zbiory C_{k-1} i S_{k-1}. Przechodząc do etapu k-tego budujemy zbiór C_k, dołączając do zbioru C_{k-1} cykl c_k i budujemy zbiór S_k na podstawie zbiorów $X_i^{k-1} \in S_{k-1}$ w następujący sposób:

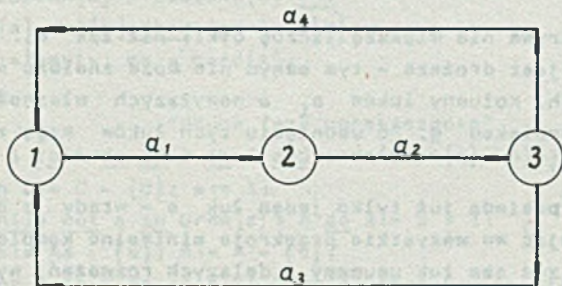
II. gdy $X_i^{k-1} \cap c_k \neq \emptyset$ dla pewnego $i = 1, \ldots, p_{k-1}$, to przekrój X_i^{k-1} wprowadzamy do rodziny S_k;

III. jeżeli zaś dla pewnego i powyższy iloczyn jest pusty, to do rodziny S_k dodajemy rodzinę zbiorów $X_i^{k-1} \cup \{a_l\}$; $a_l \in c_k$ dla $l = 1, \ldots, \mathrm{card}(c_k)$.

Utworzona w ten sposób rodzina przekrojów S_n zawiera wszystkie przekroje kompleksu g. Dla każdego przekroju można obliczyć koszt rozcięcia

$$f'(X_i^n) = \sum_{a_l \in X_i} f(a_l), \quad i = 1, \ldots, \mathrm{card}(S_n) \tag{2.8}$$

Przekroje minimalne otrzymamy wybierając najmniejsze $f'(X_i^n)$. Przekrojów minimalnych może być kilka, co widać z prostego przykładu (rys. 1).



Rys. 1. Przykład kompleksu posiadającego 3 przekroje

Jeżeli $f(a_1) = f(a_2) = f(a_3) + f(a_4)$, to przekrojami minimalnymi są, dla tego kompleksu, przekroje określone przez następujące wektory charakterystyczne:

$$z_1^* = \begin{bmatrix}1\\0\\0\\0\end{bmatrix},\quad z_2^* = \begin{bmatrix}0\\1\\0\\0\end{bmatrix},\quad z_3^* = \begin{bmatrix}0\\0\\1\\1\end{bmatrix}$$

3. Uproszczenie problemu

Opisana procedura tworzenia rodziny S_n przekrojów kompleksu g nie eliminuje przekrojów, o których już w trakcie budowania rodzin S_k wiadomo, że nie wejdą w skład przekrojów minimalnych. Redukcja tych przekrojów na każdym poziomie k dawałaby oszczędność pamięci i czasu pracy algorytmu. Podobnie, w niektórych przypadkach, możliwe jest wstępne zmniejszenie absorbowanych przez problem zasobów komputerowych. Analizę wstępnej redukcji rozmiaru problemu rozpoczniemy od zbadania łuków zewnętrznych cykli, tj. takich łuków, które wchodzą tylko do jednego cyklu. Jeżeli a jest łukiem zewnętrznym cyklu c i jeżeli

$$f(a) > \min_{b \in c'} f(b), \tag{3.1}$$

c' - zbiór łuków cyklu c bez łuków zewnętrznych,

to łuk a nie należy do żadnego przekroju minimalnego i jego kolumnę można usunąć z macierzy G. Podobnie można analizować łuki należące do dwóch cykli, do trzech itp. Uogólniając powyższe, można stwierdzić, że jeżeli

$$E(\{a_i\}) \subset E(\{a_j\}), \quad i \neq j, \tag{3.2}$$

$$f(a_j) < f(a_i), \tag{3.3}$$

to łuk a_i rozrywa nie większą liczbę cykli niż łuk a_j, a jednocześnie jego rozcięcie jest droższe - tym samym nie może znaleźć się w przekrojach minimalnych. Kolumny łuków a_i o powyższych własnościach usuwamy z macierzy G kompleksu g. Po usunięciu tych łuków mogą zaistnieć następujące sytuacje:

I. Cykl c posiada już tylko jeden łuk a - wtedy z góry wiadomo, że łuk ten musi wejść we wszystkie przekroje minimalne kompleksu - fakt ten zapamiętujemy, zaś sam łuk usuwamy z dalszych rozważań, wykreślając z macierzy G jego kolumnę oraz wiersze $E(\{a\})$.

II. Zbiór łuków pewnego cyklu c_i zawiera się w zbiorze łuków innego cyklu c_j ($c_i \subset c_j$), wtedy wiersz cyklu c_j wykreślamy z macierzy G.

Obie powyższ opisane sytuacje nie są sytuacjami uniwersalnymi, jak to sugeruje się w [2]. Wystąpić one mogą dopiero po wykonaniu redukcji wg równań 3.2 i 3.3. Nie zachodzi bowiem nigdy, by w kompleksie prostym (a taki jest założony) cykl mógł posiadać jeden łuk lub by łuki jednego cyklu stanowiły podzbiór właściwy zbioru łuków innego cyklu. Algorytm wstępnego uproszczenia problemu poszukiwania minimalnych przekrójów kompleksu przedstawiono poniżej jako procedurę "simplification". Procedura ta określa rozmiar ms*ns nowej tablicy binarnej Gs, która tworzy się w procesie upraszczania pierwotnej tablicy G kompleksu. Algorytm w zbiorze A przechowuje informacje, które łuki uległy redukcji, zaś w zbiorze binarnym C pokazuje, które cykle zostały usunięte z rozważań. Zbiór As jest zbiorem pojedynczych łuków cykli, które nie zostały zredukowane w pierwszej fazie upraszczania, a które na końcu zostaną dodane do każdego przekroju minimalnego (sytuacja I). Pierwsza faza upraszczania obejmuje działania wg równań 3.2 i 3.3. Jeżeli w tej fazie nie zostaną dokonane żadne uproszczenia, to algorytm - zgodnie z powyższym - omija fazę drugą i trzecią. Druga faza upraszczania to budowa zbioru As, trzecia natomiast jest wykonywana w celu sprawdzenia czy nie ma cykli zawierających się w innych (sytuacja II). Algorytm jest opisany metodą zawartą w [4] i uzupełnioną o operacje stworzone dla uproszczenia działań na zbiorach (teoriomnogościowych).

A1. Algorytm uproszczenia problemu poszukiwania minimalnego przekroju kompleksu

```
procedure simplification
  var a,b: arc; c,d: cycle
begin A:= space;                "pierwsza faza upraszczania"
   for a:= 1 to m - 1 do if a in A then
      for b:= a + 1 to m do if b in A then
         if Gcolumn[a] < Gcolumn[b] and
            f[b] < f[a] then A:= A - {a} else
            if Gcolumn[b] < Gcolumn[a] and
               f[a] < f[b] then A:= A - {b};
   C:= space; As:= empty; ms:= card(A);
   if ms < m then
      begin                     "druga faza upraszczania"
         for c:= 1 to n do if c in C and card(Grow[c] * A)=1 then
            begin C:= C - {c}; a:= 1;
               while not a in Grow[c] * A do a:= a + 1;
               As:= As + {a}; A:= A - {a};
               for d in C do if a in Grow[d] then C:= C - {d}
            end;
```

```
        ms:= card(A);          "trzecia faza upraszczania"
        for c:= 1 to n - 1 do if c in C then
           for d:= c + 1 to n do if d in C then
              if Grow[c]*A > Grow[d]*A then C:= C - {c} else
                 if Grow[c]*A < Grow[d]*A then C:= C - {d};
        ns:= card(C)
      end
end
```

W procedurze "simplification" typy wartości arc i cycle oznaczają okrojony typ całkowity odpowiednio od 1 do m i od 1 do n. Definicje tych typów zawiera segment główny "minisections". Wartości zmiennych tych typów służą do wskazywania elementów zbiorów: A - łuków i C - cykli. Macierz G traktowane jest jak macierz binarna, do której jest dostęp przez wskazanie wiersza - Grow[c], przez wskazanie kolumny Gcolumn[a] (wiersz i kolumnę traktuje się wtedy jak zbiory odpowiednio łuków i cykli) oraz istnieje konwencjonalny dostęp do każdego elementu tablicy G przez podanie indeksu G[c,a]. Zmienne A, As, C deklarowane są w bloku głównym jako zbiory odpowiednich mocy i operatory działające na nie lub zachodzące między nimi relacje są definiowane jak na zbiorach, np. relacja As < A jest równoważna relacji As ⊂ A = As (inkluzja zbiorów).

Jeżeli w procedurze "simplification" nastąpiło zredukowanie choćby jednego łuku, to program tworzy nową macierz binarną Gs i nową tablicę fs powstałą z tablicy f po usunięciu elementów odpowiadających zredukowanym łukom. Dokonuje tego procedura "rewrite 1", która ma następującą budowę:

A2. Tworzenie uproszczonej macierzy binarnej kompleksu - algorytm "rewrite 1"

```
procedure rewrite 1
   type arc s = 1..ms; cycle s = 1..ns;
        set as = set[arc s] of bit; set cs = set[cycle s] of bit;
        complex s = set[cycle] of set as
   var Gs: complex s; fs: array[arc s] of integer; a: arc;
       c: cycle; b: arc s; d: cycle s
begin b:= 1;
   for a in A do
      begin d:= 1; fs[b]:= f[a];
         for c in C do
            begin Gs[d,b]:= G[c,a]; d:= d + 1 end;
         b:= b + 1
   end;
creation(ms,ns,fs,Gs)
end
```

Po przepisaniu do nowych tablic wielkości odpowiadających niezredukowanym łukom i cyklom procedura "rewrite 1" wywołuje procedurę "creation", która opisana jest dalej.

4. Redukcja liczby przekrojów

Załóżmy, że utworzono rodzinę przekrojów S_{k-1} dla cykli z C_{k-1}. Na k-tym etapie buduje się rodzinę S_k według zasad II i III punktu 2. Niech X i Y są tak utworzonymi przekrojami:

$$C_k \subset E(X) \cap E(Y) \tag{4.1}$$

Utwórzmy zbiór U łuków nie badanych jeszcze dotąd cykli

$$U = \bigcup_{j=k+1}^{n} c_j$$

Zbiór $(X - Y) \cap U$ zawiera więc łuki zbioru X nie należące do zbioru Y, których rozcięcie przerywa cykle dotąd nie analizowane, natomiast zbiór $E((X - Y) \cap U)$ oznacza właśnie te cykle. Gdyby do zbioru Y dodać łuki $(X - Y) \cap U$, to ten zbiór łuków będzie rozcinał co najmniej tyle samo cykli co zbiór X:

$$E(X) \subset E(((X - Y) \cap U) \cup Y) \tag{4.2}$$

co najmniej tyle samo - gdyż mogą istnieć cykle rozcinane przez zbiór Y a nie rozcinane przez zbiór X, może bowiem zachodzić

$$E(Y) - E(X) \neq \emptyset \tag{4.3}$$

Wyrażenie 4.2 można zapisać w formie

$$E(X) \subset E(Y) \cup E((X - Y) \cap U), \tag{4.4}$$

gdyż zbiory Y i $(X - Y) \cap U$ są zbiorami rozłącznymi. Koszty rozcięcia tych zbiorów oznaczmy odpowiednio przez $f(X)$ i $f((X - Y) \cap U)$. Koszt rozcięcia łuków zbioru $Y \cup ((X - Y) \cap U)$ wynosi więc

$$f(Y) + f((X - Y) \cap U). \tag{4.5}$$

Jeżeli dla pewnych przekrojów X i Y zachodzi

$$f(X) > f(Y) + f((X - Y) \cap U), \tag{4.6}$$

to wobec 4.4 łuki ze zbioru X rozcinają nie więcej cykli niż łuki ze zbioru $Y \cup ((X - Y) \cap U)$, a ich koszt rozcięcia jest wyższy. W ten sposób przekrój X może zostać usunięty z dalszych rozważań - jest on gorszy od przekroju Y i dlatego nie jest podzbiorem minimalnego przekroju kompleksu. Jeżeli wprowadzimy funkcję kryterialną $F(X,Y) = f(Y) - f(X) + f((X - Y) \cap U)$ i jeżeli $F(X,Y) < 0$, to przekrój X w świetle powyższych rozważań nie może wejść do rodziny S_k. Funkcję kryterialną możemy przepisać w formie

$$F(X,Y) = f(Y - X) - f(X - Y) + f((X - Y) \cap U) \tag{4.7}$$

i w ten sposób funkcję tę liczy odpowiedni algorytm.

A3. Algorytm obliczania funkcji kryterialnej

```
function F(X,Y: set ac): integer
   var a1: arc c; U: set ac; k1: cycle c
begin F:= 0;
   for k1:= k + 1 to nc do U:= U + Gcrow[k1];
   for a1 in Y - X do F:= F + fc[a1];
   for a1 in (X - Y) - U do F:= F - fc[a1]
end
```

Funkcja $F(X,Y)$ wprowadza wśród kandydatów do wejścia do rodziny S_k częściowy porządek R

$$(Y,X) \in R \iff F(X,Y) < 0 \tag{4.8}$$

Stwierdzenie to jest łatwe do wykazania

- przeciwzwrotność jest oczywista,
- antysymetria: $F(Y,X) = f((X - Y) \cap U) + f((Y - X) \cap U) - F(X,Y)$, więc jeżeli $F(X,Y)$ jest ujemne, to $F(Y,X)$ jest dodatnie. Zanim wykażemy przechodność, zauważmy, że

$$X - Z \subset (X - Y) \cup (Y - Z) \tag{4.9}$$

$$(X - Y) \cap (Y - Z) = \emptyset \tag{4.10}$$

więc prawdziwa jest nierówność

$$f((X - Z) \cap U) \leqslant f((X - Y) \cap U) + f((Y - Z) \cap U) \tag{4.11}$$

- przechodność: jeżeli $F(X,Y) < 0$ i $F(Y,Z) < 0$, to ponieważ

$$F(X,Z) \leqslant f(Z) - f(Y) + f((Y - Z) \cap U) + f(Y) -$$
$$f(X) + f((X - Y) \cap U) = F(Y,Z) + F(X,Y) \qquad (4.12)$$

więc $F(X,Z)$ jest ujemne - co kończy dowód.

Algorytm kwalifikujący przekroje do rodziny S_k winien więc wybierać wyłącznie elementy minimalne w sensie relacji R. Takie postępowanie daje najpełniejsze odsiewanie przekrojów gorszych. Aby je przeprowadzić, należy dokonać sprawdzenia czy $F(X,Y) < 0$ i jeśli relacja ta zachodzi, to X odrzucamy, w przeciwnym wypadku sprawdzamy czy $F(Y,X) < 0$ i wtedy odrzucamy Y; natomiast jeżeli druga nierówność nie jest spełniona, to elementy X i Y są nieporównywalne w sensie relacji R. W ten sposób budowane są najmniejsze rodziny przekrojów, na czym oszczędza się pamięć komputera i czas jaki trzeba by poświęcić na analizę w dalszych krokach przekrojów, które mogłyby się wśliznąć do rodziny S_k przy jednokrotnym porównywaniu, jakie proponowane jest w [2] i [5]. Można wykazać, że po wstępnym uproszczeniu problemu poszukiwania przekrojów minimalnych kompleksu, każdy cykl reprezentowany jest w tablicy binarnej kompleksu G przez łuki, które są elementami minimalnymi w sensie relacji R. Aby to wykazać, przypuśćmy, że łuki a, b należą do cyklu c.

$$a,b \in c = \left\{a_i : \; i = 1,\dots,p\right\} \qquad (4.13)$$

Wtedy $F(\{a\},\{b\}) = f(b) - f(a) + f((\{a\} - \{b\}) \cap U) = f(b) - f(a) + f(\{a\} \cap U)$

i $F(\{a\},\{b\}) < 0$ mogłoby zachodzić wyłącznie wtedy, gdy $a \notin U$ i $f(a) > f(b)$, a to jest równoważne ze stwierdzeniem, że

$$E(\{a\}) \subset E(\{b\}) \quad \text{i} \quad f(a) > f(b), \qquad (4.14)$$

co przeczy założeniu, że przeprowadzono wstępne uproszczenie problemu (3.2, 3.3). Podobny wniosek otrzymamy, badając nierówność $F(\{b\},\{a\}) < 0$. Stąd wynika, że budowę rodzin S_k $(k = 1,\dots,n)$ można rozpocząć od dowolnego cyklu i że na każdym poziomie k można gromadzić w S_k wyłącznie elementy minimalne. Algorytm redukcji ma postać następującą:

A4. Algorytm redukcji liczby przekrojów

```
procedure reduction
  var i1,j1: P
begin i1:= 1;
  while i1 < p do
    begin
      if F(i1,X,Y) < 0 then
        begin p:= p - 1;
          for j1:= i1 to p do j1.X:= (j1 + 1).X;
             i1:= i1 - 1; nil((p + 1).X)
        end else
        if F(Y,i1,X) < 0 then begin Y:= empty; i1:= p end;
      i1:= i1 + 1
    end;
  if not Y = empty then
    begin p:= p + 1;
      new(p,X); p.X:= Y
    end
  end
```

W algorytmie tym użyto dynamicznej rezerwacji miejsca na zbiory X. Przekrój zredukowany traci miejsce w pamięci - nil((p + 1).X), niezredukowany przekrój Y reaktywuje nowe miejsce w pamięci - new(p,X). Liczba zbiorów X na każdym poziomie k nie jest znana a priori, tym samym albo się zastosuje efektywną rezerwację dynamiczną na każdy j-ty zbiór X (j.X), albo zapewni się statyczną rezerwację miejsca na całą rodzinę $\{X\}$ zbiorów - oczywiście z pewnym oszacowanym nadmiarem. Decyzja należy do programisty, algorytm A4 pokazuje natomiast, w których momentach może być użyta rezerwacja dynamiczna.

5. Algorytm tworzenia rodziny przekrojów minimalnych

Na każdym etapie k tworzenia rodziny przekrojów $S_k = \{1.S,...,p_k.S\}$ jej rozmiar jest zmienny i tym samym zmienny jest obszar pamięci poświęcony na jej przechowywanie. Aby to wskazać translatorowi, zdefiniowano zmienne typu P jako wskaźniki pojawiających się lub znikających obszarów pamięci (o tych samych rozmiarach) przeznaczonych na magazynowanie nowych przekrojów.

A5. Algorytm tworzenia rodziny przekrojów minimalnych "creation"

```
procedure creation (mc,nc,fc,Gc)
   type arc c = 1..mc; cycle c = 1..nc;
        set ec = set[arc c] of bit;
        set cc = set [cycle c] of bit;
        complex c = set [cycle c] of set ac;
        P = pointer of X,S: set ec
   var Gc: complex c; fc: array [arc c] of integer; k: integer;
      i,j,p,q: P; a: arc c
begin i:= 1;
   for a in Gcrow[1] do
      begin new(i.X); i.X:= {a}; i:= i + 1 end;
   p:= p - 1; q:= 0;
   for k:= 1 to nc do
      begin "przepisanie rodziny zbiorów  X  do rodziny  S"
        if q < p then
          begin
             for i:= 1 to q do i.S:= i.X;
             for i:= q + 1  to p do
                begin new(i.S); i.S:= i.X end
          end else
          begin
             for i:= 1 to p do i.S:= i.X;
             for i:= p + 1 to q do nil(i.S)
          end;
        q:= p;
        for i:= 1 to p do nil(i.X);
        p:= 0; "generowanie nowej rodziny zbiorów  X"
        for i:= 1 to q do
          if not i.S * Gcrow[k] = empty then
             begin Y:= i.S; reduction end else
             for a in Gcrow[k] do
                begin Y:= Gcrow[k] + {a}; reduction end
      end;
   rewrite 2
end
```

Algorytm rozpoczyna tworzenie rodziny przekrojów od pierwszego cyklu reprezentowanego przez Gcrow[1]. Na końcu procedura "creation" wywołuje procedurę "rewrite 2", która tworzy statyczny zbiór S w miejsce dynamicznych zbiorów X i S, których rezerwację kasuje - opisane ona jest dalej. Kiedy powyższy algorytm utworzy rodzinę przekrojów S_n, to składa się ona wyłącznie z przekrojów minimalnych. Wynika to z tego, że zbiór U jest pusty i wtedy

$$F(X,Y) = f(Y - X) - f(X - Y) = -F(Y,X), \qquad (5.1)$$

ponadto wiadomo, że algorytm kwalifikuje do rodziny S_n tylko takie zbiory X i Y, dla których nie zachodzi ani $F(X,Y) < 0$, ani $F(Y,X) < 0$, więc

$$F(X,Y) = F(Y,X) = 0 \iff f(Y - X) = f(X - Y), \qquad (5.2)$$

co z kolei jest równoważne stwierdzeniu, że

$$f(X) = f(Y) \qquad (5.3)$$

tym samym wybór przekrojów minimalnych z rodziny S_n jest trywialny - reprezentuje ona wyłącznie przekroje minimalne utworzone automatycznie przez procedurę "creation" w wyniku naprzemiennego stosowania akcji syntezy i redukcji przekrojów. Zadaniem ostatniej procedury jest przepisanie przekrojów minimalnych do formy zawierającej wszystkie łuki kompleksu, tj. do tablicy Ss, dodanie do każdego przekroju łuków zbioru As i obliczenie kosztu rozcięcia:

A6. Algorytm wypełniania tablicy przekrojów minimalnych "rewrite 2"

```
procedure rewrite 2
  var b: arc c
begin s:= p;
  for i:= 1 to s do
     begin b:= 1;
        for a:= 1 to m do
           if a in A then
              begin
                 Ss[i,a]:= i.X[b]; b:= b + 1
              end else
              Ss[i,a]:= As[a]
     end;
  for i:= 1 to p do nil(i.X);
  for i:= 1 to q do nil(i.S);
  koszt:= 0;
  for a in Ssrow[1] do koszt:= koszt + f(a)
end
```

Ostatecznie główny blok algorytmu poszukującego rodzinę przekrojów minimalnych ma następującą postać:

A7. Algorytm "minisections"

```
program minisections
   var m,ms,n,ns: integer
   type arc = 1..m; cycle = 1..n; sec = 1..s;
        set a = set[arc] of bit;
        set c = set [cycle] of bit;
        complex = set[cycle] of set a;
        fam sec = set[sec] of set a
   var koszt: integer; A,As: set a; C: set c; G: complex;
       Ss: fam sec; h: array[arc] of integer
begin
   read(m,n,G,h);
   simplification;
   if ms < m then
     begin
        rewrite 1;
        creation(ms,ns,fs,Gs)
     end else
     creation(m,n,f,G);
   print(Ss,koszt)
end
```

Procedury "read" i "print" nie są tutaj opisane - służą one jedynie do wskazania danych wejściowych i wyjściowych. Rezerwacja pamięci odbywa się w następującej kolejności: po wprowadzeniu m i n rezerwuje się pamięć dla zbiorów A, As, C i G. Dla tablicy Ss pamięć jest rezerwowana po określeniu s w procedurze "creation".

6. Wnioski

6.1. Wprowadzając w algorytmie "simplification" możliwości obejścia drugiej i trzeciej fazy upraszczania, zmniejsza się w szczególnych wypadkach czas realizacji programu o czas zbędnego usiłowania dokonania dalszych uproszczeń.

6.2. Wykorzystanie w algorytmie "reduction" właściwości elementów minimalnych w zbiorach częściowo uporządkowanych posiada szereg zalet:

- redukuje się do minimum obszar pamięci zajmowanej przez rodziny S_k przekrojów,
- zmniejsza się czas wykonywania programu o czas tworzenia i porównywania przekrojów nie będących kandydatami na przekroje minimalne kompleksu,
- na poziomie $k = n$ tworzy się automatycznie rodzina S_n przekrojów minimalnych.

6.3. Dodatkową oszczędność mocy obliczeniowej komputera można osiągnąć przez dynamiczną rezerwację obszaru pamięci poświęconego na rodziny S_k. Momenty, gdzie w opisanych wyżej algorytmach wydaje się to być celowe opisano według maniery z monografii [4].

Dodatek

Opis języka algorytmów

Algorytmy w niniejszej pracy są pisane w języku Pascal według definicji tego języka zawartej w pracy [4]. Algorytmy operują tutaj danymi konwencjonalnymi, jak: liczby całkowite, zmienne typu integer oraz ich tablice. Działania na nich nie wymagają objaśnień. Obok tych typów wielkości na użytek tej pracy zdefiniowano dodatkowo typy, takie jak: zbiory i rodziny zbiorów. Schematy definicji tych typów są następujące - najpierw definiuje się typ całkowity okrojony dla określenia mocy zbioru i zakresu zmienności wskaźnika elementów zbioru, a następnie sam zbiór lub rodzinę zbiorów:

- typ całkowity okrojony

```
var m: integer
type element = 1..m
```

- zbiór

```
type set = set[element] of bit
```

- rodzina zbiorów

```
var n: integer
type el fam = 1..n
     family = set[el fam] of set
```

Zmienne powyższych typów reprezentowane są przez tablice binarne - w przypadku zbioru jednowymiarowe o wymiarze m, w przypadku rodziny zbiorów dwuwymiarowe o wymierze n * m.

Przykład definiowania typów zmiennych:

```
var R: family; j: el fam; A,B,C: set; i: element
```

Zakłada się, że efektem określenia typu zmiennej jest przyznanie jej miejsca w pamięci, gdy tylko określona zostanie jej moc. Zakłada się dalej, że dostęp do zmiennej obok powołania się na jej nazwę (dostęp globalny) może odbywać się również poprzez wskazanie jej składowej w następujący sposób:

- dla zbioru dostęp do i-tego bitu określa A[i], przy czym jeżeli bit ten jest w stanie "1", to przyjęto, że i-ty element należy do zbioru A,
- dla rodziny zbiorów dostęp do i-tego elementu j-tego zbioru określa R[j,i], dostęp do j-tego zbioru określa się za pomocą wskazania wiersza Rrow[j] oraz zakładamy, że istnieje jeszcze dostęp do i-tej kolumny wyrażony przez Rcolumn[i]. Następnym skutkiem określenia zmiennej (jej ty-

pu) jest możliwość wykonania pewnych operacji i sprawdzenia relacji zachodzących między nimi. Zakłada się, że możliwe są następujące operacje przypisania i operacje boolowskie

- A:= space, ustawienie wszystkich bitów w pozycji "1",
- A:= empty, jak wyżej tylko w pozycji "0",
- A:= A + B, na każdej pozycji pojawia się logiczna suma bitów,
- A:= A * B, jak wyżej tylko iloczyn logiczny bitów,
- A:= A - B, jak wyżej tylko różnica logiczna bitów,

ponadto zakłada się, że dostępne jest sprawdzanie zachodzenia relacji inkluzji między zbiorami,

- A < B, co jest równoważne sprawdzaniu następującej relacji równości A = = A * B, relację przynależności elementu do zbioru określa się przez symbol <u>in</u>: a <u>in</u> A. W pracy przewidziano możliwość dynamicznej rezerwacji miejsca w pamięci - odbywa się to przez wyznaczenie zbioru wskaźników, określających dla którego zbioru rezerwacja będzie dokonywana i przez określenie pojemności rezerwowanej pamięci za pomocą definicji typu wielkości, np.:

```
var X: set
type wskaźnik = pointer of X
var r: wskaźnik
```

Powyższa informacja mówi, że rezerwowana będzie pamięć dla zmiennej typu zbiór o nazwie "X" i wskazywanej przez wskaźnik r. Rozkaz powołujący miejsce w pamięci ma postać new(r.X), rozkaz likwidujący miejsce w pamięci ma postać nil(r.X). Dla każdego r-tego zbioru X rezerwuje się jednakową pamięć równą m bitom.

LITERATURA

[1] Rasiowa H.: Wstęp do matematyki współczesnej. PWN, Warszawa 1971.

[2] Ostrowskij G.M., Wolin Ju.M.: Modelirowanie słożnych chemiko-technologiczeskich schiem. Chimia, 1975.

[3] Ostrowski G.M., Wolin J.M.: Optymalizacja złożonych systemów technologii chemicznej. WNT, Warszawa 1974.

[4] Wirth N.: Algorytmy + struktury danych = programy. WNT, Warszawa 1980.

[5] Wolin Ju.M.: Chem. Eng. Sci., 1973 r. vol. 28.

[6] Deo N.: Teoria grafów i jej zastosowanie w technice i informatyce. PWN Warszawa 1980.

[7] Berge C.: Graphs and Hypergraphs. North-Holland Publisching Company, 1976.

[8] Reingold E.M., Nievergelt J., Deo N.: Combinatorial Algorithms. Theory and Practice. Tłum. ros. Mir, 1980.

[9] Aho A.V., Hopcroft J.E., Ullman J.D.: The Design and Analysis of Computer Algorithms. Tłum. ros. Mir, 1979.

[10] Himmelblau D.M.: Decomposition of Large-Scale Problems. North-Holland Publisching Company, 1973.

[11] Rudd D.F., Watson Ch.C.: Strategy of Process Engineering. John Wiley and Sons Inc. 1968.

Recenzent: Doc. dr hab. inż. Józef Grabowski

Wpłynęło do Redakcji 15.11.1981 r.

ПОИСК МИНИМАЛЬНЫХ СЕЧЕНИИ КОМПЛЕКСА

Р е з ю м е

В статье представлен алгоритм поиска минимального сечения цикла графа. Представленный метод образует минимальные сечения комплекса ограничиваясь до сечений выполняющих определенные условия. Остальные сечения не анализируются, что уменшает привлеченные к выполнению задания машинные ресурсы. Анализ ограничен до строго связанных графов, что не ограничивает общности метода.

MINIMAL CUTS RESEARCH FOR A COMPLEX

S u m m a r y

The paper concerns with an algorithm for finding a set of minimal graph cuts. Presented method forms minimal graph cuts by consideration those cuts only which satisfy certain conditions. Remaining cuts are eliminated from further considerations and the computer power needed to solve above problem decreases. Additional introductory problem simplification has been brought in. Without loss of generosity only strongly connected graphs have been considered here.