ZBIGNIEW RZEMYKOWSKI

ZASTOSOWANIE METOD „JEDNOETAPOWYCH" DO OPTYMALNEJ ALOKACJI ZASOBÓW ODNAWIALNYCH MIĘDZY CZYNNOŚCI PODZIELNE

1. Wstęp

W artykule dokonano przeglądu niektórych propozycji tzw. jednoetapowych[1] metod rozwiązywania problemu optymalnego rozdziału zasobów odnawialnych między podzielne czynności przedsięwzięcia opisanego za pomocą słabo spójnej sieci czynności. Badania skoncentrowano na sprecyzowaniu istoty proponowanych metod oraz analizie ich złożoności obliczeniowej. Niezależnie od tego zaproponowano kilka algorytmów z zakresu zagadnień cząstkowych występujących przy rozwiązywaniu omawianego problemu[2].

2. Sformułowanie problemu

Dane jest przedsięwzięcie opisane za pomocą kanonicznej, uporządkowanej topologicznie oraz słabo spójnej sieci czynności $S=[P, U]$ o m czynnościach (łukach) oraz n zdarzeniach (wierzchołkach). Sieć czynności nazywa się siecią słabo spójną, jeżeli nie istnieje w niej droga (ciąg czynności uwzględniający ich orientację) wiodąca od zdarzenia początkowego do zdarzenia końcowego sieci. Dla każdej czynności $i \in U$ jest określony czas jej trwania d_i oraz wektor zapotrzebowań w jednostce czasu na q rodzajów środków $z_i=(z_{i1}, z_{i2}, \ldots, z_{iq})$.

---

[1] Pojęcie to – niezbyt dobrze odpowiadające istocie tych metod – zostało wprowadzone w pracach [15] i [16].

[2] Niniejszy artykuł stanowi zmodyfikowaną wersję niepublikowanego opracowania przygotowanego w ramach badań realizowanych w Programie CPBP 10. 09 (zob. [12]).

Zasoby tych środków są stałe w czasie i wynoszą odpowiednio $r_1, r_2, \ldots, r_q$ jednostek. Przyjmijmy, że $r_j \geqslant z_{ij}$ dla każdej czynności i oraz każdego środka j. Zakłada się ponadto, że dowolna czynność $i \in U$ jest wykonywana w chwili t wtedy i tylko wtedy, gdy przydzielono jej $z_{i1}, z_{i2}, \ldots, z_{iq}$ jednostek kolejnych środków.

Zakładamy, że wszystkie czynności przedsięwzięcia są podzielne. Oznacza to, że wykonywanie każdej czynności może być w dowolnej chwili i dowolną liczbę razy przerwane, a następnie wznowione bez jakiejkolwiek straty czasu czy środków.

Poszczególnym czynnościom należy przydzielić w czasie odpowiednią liczbę środków (tj. wyznaczyć harmonogram realizacji przedsięwzięcia) w taki sposób, aby całe przedsięwzięcie wykonać w najkrótszym czasie. Musi być przy tym zachowana określona w sieci kolejność wykonywania czynności, a łączna wielkość przydzielonych czynnościom środków nie może przekraczać w żadnym momencie ich dostępności.

Sformułowany wyżej problem rozdziału zasobów odnawialnych między czynności podzielne jest problemem *NP*-trudnym. Wynika to np. z tego, iż problem ten jest w szczególnym przypadku tożsamy z ogólnym problemem szeregowania. Wystarczy bowiem przyjąć, iż:

– jedynym środkiem są identyczne procesory ($q=1$);

– każda czynność może być wykonywana na dowolnym procesorze ($z_{i1}=1$ dla $i \in U$);

– czas wykonywania czynności i na dowolnym procesorze jest równy czasowi jej trwania.

Ponieważ ogólny problem szeregowania jest – jak wykazał Ullman w pracy [20] – problemem *NP*-trudnym, więc i omawiany tutaj problem alokacji środków w sieci czynności jest co najmniej tak samo trudny.

3. Idea metod opartych na frontach czynności

Podstawową grupę metod rozwiązywania problemu alokacji zasobów odnawialnych między czynności podzielne stanowią tzw. metody jednoetapowe[3]. Polegają one na wyznaczeniu dopuszczalnych zasobowo frontów czynności w danej sieci czynności oraz ustaleniu – poprzez rozwiązanie odpowiednich zadań – optymalnych czasów realizacji przedsięwzięcia za pomocą dopuszczalnie uporządkowanych frontów. Przypomnijmy, że:

[3] Jak już mówiliśmy, nazwa metody „jednoetapowe" została zaproponowana w pracach [15] i [16]. Metody „dwuetapowe" zostały po raz pierwszy przedstawione niezależnie w pracach [6] oraz [17]. W [13] przedstawiono jeszcze inny, nowy sposób rozwiązywania problemów rozdziału zasobów między czynności podzielne, sprowadzający się do sformułowania i rozwiązania odpowiedniego zadania programowania zero-jedynkowego, przy którym nie ma żadnego znaczenia typ analizowanej sieci. W identyczny sposób postępuje się bowiem w przypadku sieci słabo jak i silnie spójnych.

– frontem czynności w sieci S nazywa się każdy taki niepusty zbiór czynności $f=(u_1, u_2, \ldots, u_k)$, że żadne dwie czynności danego zbioru nie leżą na jednej ścieżce w S ([22], [5]),

– front dopuszczalny zasobowo charakteryzuje się natomiast tym, że łączne zapotrzebowanie wszystkich czynności danego frontu na poszczególne środki nie przekracza ich zasobów,

– porządkowanie dopuszczalne frontów zapewnia zachowanie określonej w sieci kolejności wykonywania czynności i polega na takim uporządkowaniu zbioru frontów, aby dla każdej pary frontów f_s, f_w takich, że $s<w$ front f_w nie zawierał czynności poprzedzających w sieci czynności należących do frontu f_s ([22], [5]).

Okazuje się, iż dopuszczalne uporządkowanie frontów dla sieci słabo spójnych nie zawsze jest możliwe. Jak zauważono w [22] w sieciach tego typu można utworzyć takie pary frontów $f_w=(u_{w1}, u_{w2}, \ldots, u_{wt})$ oraz $f_v=(u_{v1}, u_{v2}, \ldots, u_{vs})$, zawierających takie czynności u_{wi}, u_{wj} oraz u_{vk}, u_{vl}, że:

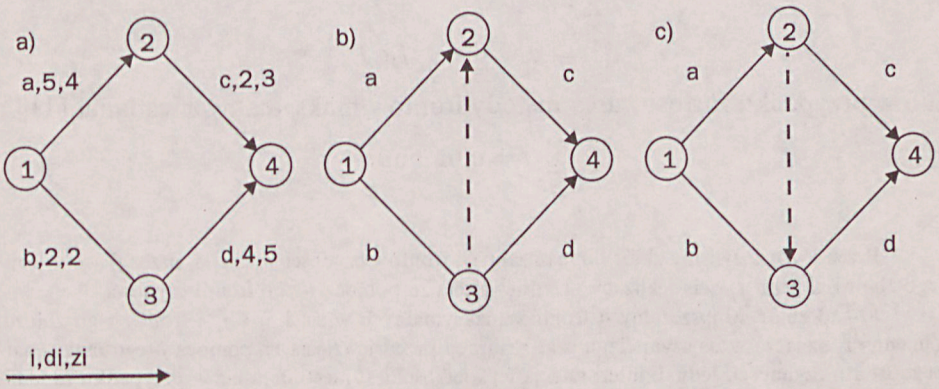
a) $u_{wi} \prec u_{vk}$

oraz

b) $u_{vl} \prec u_{wj}$.

Symbolem $\prec$ oznaczono relację poprzedzania w sieci czynności.

Z relacji a) wynika zatem, iż dla zachowania określonej w sieci kolejności wykonywania czynności front f_w powinien być zrealizowany przed frontem f_v, z relacji zaś b) – iż kolejność wykonywania tych frontów musi być odwrotna. Mamy zatem dwa sprzeczne postulaty uniemożliwiające dopuszczalne uporządkowanie zbioru frontów zawierającego takie fronty. Przez realizację frontu będziemy rozumieć realizację czynności należących do danego frontu. Każdą parę frontów spełniających przedstawione wyżej warunki nazwano w [5] frontami sprzecznymi. Najprostszą sieć zawierającą fronty sprzeczne przedstawia rysunek 1a. Spośród wszystkich możliwych do utworzenia dla tej sieci frontów czynności $F=\{(a), (a,b), (a,d), (b), (b,c), (c), (c,d), (d)\}$ mamy jedną parę frontów sprzecznych; są to fronty (a,d) oraz (b,c).



Rys. 1.

W takiej sytuacji, gdy zbiór frontów zawiera fronty sprzeczne, czas realizacji przedsięwzięcia za pomocą co najmniej jednego, dowolnego frontu z każdej pary frontów sprzecznych musi być zerowy ([22] s. 129). W literaturze można spotkać kilka propozycji postępowań zapewniających – bezpośrednio lub pośrednio – spełnienie tego warunku. Omawiamy je w następnych paragrafach opracowania.

4. Idea metody silnych uspójnień sieci

Pierwsza chronologicznie metoda wyznaczania dokładnego (optymalnego) rozwiązania omawianego problemu rozdziału zasobów odnawialnych w słabo spójnych sieciach czynności polega na:

1) utworzeniu na podstawie wyjściowej słabo spójnej sieci czynności, drogą wprowadzania do niej czynności fikcyjnych, wszystkich – dopuszczalnych ze względu na wymaganą kolejność wykonywania czynności – wariantów sieci silnie spójnych,

2) wygenerowaniu dla każdej utworzonej w 1) sieci silnie spójnej zbioru dopuszczalnych zasobowo frontów czynności lub zbioru dopuszczalnych frontów maksymalnych. Można wykazać, że żaden z utworzonych dla tych sieci zbiorów frontów nie będzie zawierał frontów sprzecznych ([5], s. 51). Wszystkie zaś utworzone w ten sposób zbiory frontów stanowią wszystkie podzbiory zbioru frontów dopuszczalnych w sieci wyjściowej powstające przez eliminację jednego frontu z każdej pary frontów sprzecznych,

3) utworzeniu i rozwiązaniu dla każdego z utworzonych w 2) zbioru frontów (dla każdego wariantu sieci silnie spójnej) zadania ([22], [5])[4]:

$$\sum_{f_j \in F} t_j = \text{minimum},$$

$$\sum_{f_j : i \in f_j} t_j = d_i \quad \text{dla} \quad i \in U,$$

$$t_j \geqslant 0 \quad \text{dla} \quad f_j \in F$$

lub w przypadku zastosowania metody frontów maksymalnych zadania [11][5]:

$$\sum_{\bar{f}_j \in \bar{F}} t_j = \text{minimum},$$

---

[4] Przez F oznaczyliśmy zbiór dopuszczalnych frontów czynności w sieci S, przez f_j – j-ty front tego zbioru, a przez t_j czas realizacji przedsięwzięcia za pomocą j-tego frontu czynności.

[5] $\bar{F}$ – to zbiór dopuszczalnych frontów maksymalnych w sieci S, a $\bar{f}_j$ – front j tego zbioru. Zmienna t_j oznacza w dalszym ciągu czas realizacji przedsięwzięcia za pomocą j-tego maksymalnego frontu czynności. Jednak interpretacja tej wielkości jest nieco inna niż w przypadku zadania poprzedniego (zob. [11]).

$$\sum_{\bar{f}_j : i \in \bar{f}_j} t_j \geqslant d_i \quad \text{dla} \quad i \in U,$$

$$t_j \geqslant 0 \quad \text{dla} \quad \bar{f}_j \in \bar{F},$$

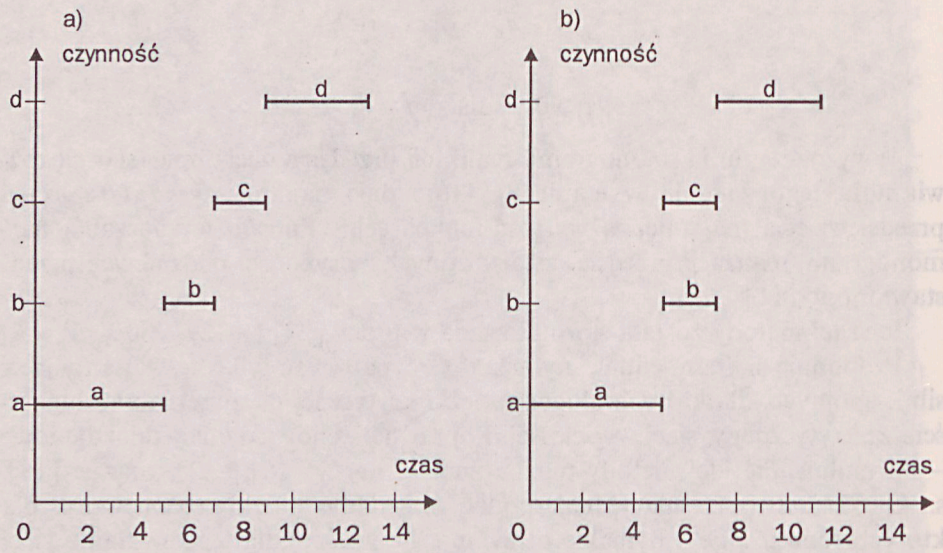
4) wyznaczeniu harmonogramu realizacji przedsięwzięcia opartego na rozwiązaniu tego zadania w etapie 3), które daje najmniejszy czas realizacji przedsięwzięcia (najmniejszą wartość funkcji celu). Sposób wyznaczania harmonogramu realizacji przedsięwzięć złożonych z czynności podzielnych przedstawiono w [11].

Idea tej metody została sformułowana w pracy [22] (s. 129 - 130).

Proponuje się tam jednak mylnie, aby rozpatrywać tylko te warianty sieci silnie spójnych, dla których długość ścieżki krytycznej nie przekracza długości ścieżki krytycznej w sieci wyjściowej. Poprawne – choć jedynie jednozdaniowe – sformułowanie idei metody można znaleźć np. w [16], s. 23, oraz w [18], s. 49 - 50. Łatwo można jednak podać przykłady takich przedsięwzięć, dla których rozwiązanie optymalne otrzymuje się właśnie dla tego wariantu sieci silnie spójnej, którego ścieżka krytyczna jest dłuższa od ścieżki w wyjściowej sieci czynności. Tak też jest i w przypadku przedsięwzięcia opisanego siecią na rysunku 1a. Długość ścieżki krytycznej w tej sieci – ścieżki $\langle 1,2\rangle$ $\langle 2,4\rangle$ – wynosi 7. Oba warianty silnego uspójnienia sieci wyjściowej przedstawiają rysunki 1b oraz 1c. W przypadku sieci przedstawionej na rysunku 1b przebieg (i długość) ścieżki krytycznej nie ulega zmianie, natomiast w przypadku sieci przedstawionej na rysunku 1c mamy nową ścieżkę krytyczną; ścieżkę $\langle 1,2\rangle$, $\langle 2,3\rangle$, $\langle 3,4\rangle$ o długości 9. Zgodnie z przytoczoną wyżej propozycją Zuchowickiego i Radczika [22] wystarczyłoby zatem rozważać jedynie sieć przedstawioną na rysunku 1b. Tymczasem najkrótszy harmonogram realizacji przedsięwzięcia – zakładając, że dostępność zasobu wynosi 5 jednostek – można zbudować na podstawie sieci przedstawionej na rysunku 1c; uwzględniając w harmonogramie wyeliminowany w sieci na rysunku 1b front (b,c). Optymalne harmonogramy dla sieci z rysunku 1b i 1c przedstawiają rysunki 2a oraz 2b (str. 60).

Szczegółowo omówimy teraz jedynie pierwszy etap postępowania; tworzenie silnie spójnych sieci czynności. Problemy występujące w pozostałych etapach są omówione dokładnie w innych opracowaniach (zob. np. [22, 5, 19, 15, 16, 11]).

Powtórzmy zatem. Dana jest słabo spójna sieć czynności. Należy utworzyć na podstawie tej sieci – drogą wprowadzania do niej odpowiednich czynności fikcyjnych – wszystkie dopuszczalne ze względu na wymaganą kolejność wykonywania czynności sieci silnie spójne. Załóżmy dodatkowo, iż żadna z sieci silnie spójnych nie zawiera czynności fikcyjnych zbędnych do silnego uspójnienia sieci; nie zawiera w zbiorze wprowadzanych do sieci czynności fikcyjnych łuków tranzytywnych. Dla sieci na rysunku 1 taką czynnością byłaby przykładowo czynność $\langle 1,4\rangle$.



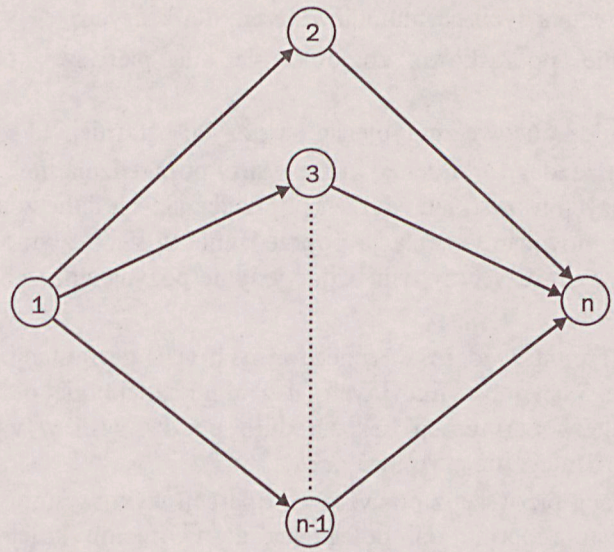
Rys. 2.

Wykażemy teraz, że liczbę możliwych do utworzenia sieci silnie spójnych nie można ograniczyć przez wielomian rozmiarów wyjściowej sieci czynności; przez wielomian liczby czynności lub liczby zdarzeń w tej sieci. W tym celu zauważmy, że silne uspójnienie sieci czynności polega na wprowadzeniu do niej czynności fikcyjnych w taki sposób, aby istniała ścieżka przechodząca przez wszystkie wierzchołki lub mówiąc inaczej, aby uporządkowanie topologiczne otrzymanej sieci było jednoznaczne (liniowe). Rozpatrzmy z kolei przedstawioną na rysunku 3 sieć czynności S o zbiorze zdarzeń $P=\{1,2,...,n\}$ oraz zbiorze czynności

$$\bar{U}=\{\langle 1,j\rangle, \langle j,n\rangle : j=2,3,...,n-1\}.$$

Zauważmy, że przez U oznaczamy zbiór czynności utożsamianych z ich numerami, a przez $\bar{U}$ – zbiór czynności definiowanych jako pary ⟨zdarzenie poprzedzające, zdarzenie następujące danej czynności⟩. Na podstawie tej sieci można utworzyć $(n-2)!$ sieci silnie spójnych[6]. Wynika to z tego, że na tyle właśnie sposobów można uporządkować liniowo zdarzenia $2,3,...,n-1$, wprowadzając za każdym razem do sieci wyjściowej $n-3$ odpowiednio zorientowanych czynności fikcyjnych łączących te zdarzenia. Wyklucza to oczywiście istnienie wielomianowych algorytmów tworzenia wszystkich sieci silnie spójnych. Mówiąc inaczej dowodzi, że omawiany tutaj problem generowania

[6] Ponieważ dla tej sieci liczba czynności m wynosi $2(n-2)$, więc w stosunku do m liczba ta wynosi $(m/2-4)!$.



Rys. 3

wszystkich sieci silnie spójnych jest problemem *NP*-złożonym. Mówiąc dokładniej – jako problem generowania jest to problem $\#P$-zupełny (zob. np. [3], rozdz. 7).

Sieć przedstawiona na rysunku 3 jest jednocześnie tą siecią, dla której można utworzyć najwięcej sieci silnie spójnych. Wynika to z tego, że:

– n wierzchołków dowolnej sieci czynności można uporządkować liniowo na $n!$ sposobów,

– zgodnie z zasadami porządkowania topologicznego zdarzenie początkowe sieci (wierzchołek minimalny) musi mieć numer 1, a zdarzenie końcowe (wierzchołek maksymalny) – numer n,

– tylko w przypadku sieci tego typu jak na rysunku 3 pozostałe $n-2$ wierzchołki można uporządkować w dowolny sposób. Sposobów tych jest właśnie $(n-2)!$.

Otwartym problemem pozostaje skonstruowanie wyrażenia pozwalającego na dokładne ustalenie – dla danej sieci słabo spójnej – liczby możliwych do utworzenia sieci silnie spójnych oraz zakwalifikowaniu problemu ustalania tej liczby (problemu zliczania sieci silnie spójnych) do problemów łatwych (problemów klasy $\#P$) bądź trudnych (problemów $\#P$-zupełnych).

5. Algorytm generowania silnie spójnych sieci czynności

Opierając się na wynikach uzyskanych w poprzednim paragrafie, zbudujmy teraz algorytm tworzenia dla słabo spójnej sieci czynności wszystkich sieci silnie spójnych. Jego istota sprowadza się do dwu operacji:

1) Generowania tych permutacji zdarzeń, dla których:

a) zdarzenie początkowe znajduje się na pierwszej pozycji każdej permutacji,

b) zdarzenie końcowe znajduje się na pozycji ostatniej,

c) pozycja każdego zdarzenia, które w sieci poprzedzają inne zdarzenia, jest wyższa od pozycji wszystkich zdarzeń poprzedzających dane zdarzenie. Ponieważ – jak już mówiliśmy – relacja poprzedzania w sieci czynności jest relacją przechodnią, wystarczy przy tym badać jedynie pozycję zdarzeń bezpośrednio poprzedzających.

2) Rozpatrzeniu każdej z wygenerowanych w 1) permutacji oraz wprowadzeniu do sieci S czynności fikcyjnych $\langle a, b\rangle$ dla każdej takiej pary zdarzeń a, b, że b następuje w permutacji bezpośrednio po a – jeśli w wyjściowej sieci czynności nie istnieje już czynność $\langle a, b\rangle$.

Dla realizacji pierwszej z powyższych operacji wykorzystamy znaną procedurę generowania permutacji polegającą na tworzeniu kolejnych uporządkowań drogą pojedynczej transformacji sąsiednich elementów ([7], s. 27-28). Za pomocą tej procedury utworzymy wszystkie permutacje zbioru zdarzeń, wybierając od razu uporządkowania spełniające postulat sformułowany w punkcie 1c). Porządek zdarzeń spełniający ten warunek – a więc każde takie uporządkowanie zdarzeń, że dla $\langle i, j\rangle \in U$ zdarzenie i poprzedza w uporządkowaniu zdarzenie j – będziemy dalej nazywać dopuszczalnym uporządkowaniem zdarzeń.

W przedstawionym niżej algorytmie – zapisanym w konwencji zbliżonej do języka PASCAL – stosujemy (poza zmiennymi pomocniczymi) następujące oznaczenia:

P[i] – numer zdarzenia stojącego w bieżącej permutacji na pozycji i,

C[i] – bieżąca pozycja zdarzenia i względem zdarzeń i+1, i+2,..., n,

PR[i] – zmienna logiczna przyjmująca wartość true, gdy zdarzenie i jest przesuwane – w stosunku do permutacji poprzedniej – na pozycję wyższą, a wartość false przy przesuwaniu na pozycję niższą,

OK – zmienna logiczna informująca o tym, czy wygenerowany porządek zdarzeń jest dopuszczalny,

i(l), j(l) – zdarzenie poprzedzające i zdarzenie następujące l-tej z kolei czynności w zbiorze U,

Ũ – zbiór czynności w aktualnie generowanej sieci czynności,

a, b – para zdarzeń, dla których w danej permutacji nie jest zachowana kolejność niezbędna dla dopuszczalnego ich uporządkowania; kolejność a poprzedza b.

Danymi algorytmu jest zbiór czynności Ũ oraz liczba wierzchołków (n) i łuków (m) w sieci.

Algorytm generowania silnie spójnych sieci czynności

```
Procedure Nowa_sieć (P);
begin
    Ũ := Ū;
    for j := 1 to n−1 do
        if ⟨j,j+1⟩ ∉ Ū then
            Ũ := Ũ + ⟨j,j+1⟩;
end;
Procedure Sprawdz_porzadek (P, a, b, OK);
begin
    l := 0; {nr czynności w U}
    repeat
        l := l+1; k := 0; OK := false;
        repeat
            k := k+1;
            if k = i(l) then OK := true;
            (* Spełnienie tego warunku oznacza, iż zdarzenie i(l) poprzedza
w bieżącej permutacji zdarzenie j(l) *)
        until OK or (k = j(l));
        (* Jeżeli dla OK = false zachodzi k = j(l), to w bieżącej permutacji
zdarzenie j(l) „pojawiło się" przed zdarzeniem i(l) *);
    until not OK or (l = m);
    if not OK then begin
        a := i(l); b := j(l);
    end;
end;
Procedure Permutacje (Ū, n, m);
begin
    OK := true;
    for i := 1 to n do begin
        P[i] := i; C[i] := 1; PR[i] := true;
    end;
    C[n−1] := 0; Nowa_siec (P); i := 2;
    while i < n−1 do begin
        i := 2; × := 0;
    while C[i] = n−1 do begin
        PR[i] := not PR[i]; C[i] := 1;
        if PR[i] then × := × + 1;
        i := i+1;
    end;
    if i < n−1 then begin {transpozycja}
```

```
        if PR [i] then k := C[i] + x
            else k := n - i - C[i] + x;
        y := P[k]; P[k] := P[k+1]; P[k+1] := y;
        C[i] := C[i] + 1;
        if OK then begin
            if ⟨P[k+1], P[k]⟩ ∈ Ū then
                begin {b przestawiono przed a dla ⟨a, b⟩ ∈ Ū}
                    OK := false; a := P[k+1]; b := P[k];
                end else
                    Nowa_siec (P)
            else
                if (P[k] = a) and (P[k+1] = b) then
                    begin {przywrócono właściwy porządek zdarzeń a, b}
                        Sprawdz_porzadek (P, a, b, OK);
                        if OK then Nowa_siec (P);
                    end;
            end; {if OK}
        end;     {transpozycji}
    end;         {while i < n - 1}
end;
begin (Ū, n, m) {początek algorytmu}
    Permutacje (U, n, m)
end.
```

Główną procedurą algorytmu jest procedura Permutacje. Generuje ona kolejne permutacje zdarzeń, sprawdza, czy dokonywana transpozycja zdarzeń spowoduje niedopuszczalność uporządkowania oraz czy powodująca niedopuszczalność kolejność zdarzeń *a* oraz *b* została w wyniku wykonanej transformacji zachowana, czy też uległa zmianie. Procedura ta korzysta z dwu procedur pomocniczych. W procedurze Nowa_siec wprowadza się do sieci odpowiednie czynności fikcyjne, natomiast w procedurze Sprawdz_porzadek ustala się, czy zapisane w tablicy P uporządkowanie zdarzeń jest dopuszczalne.

Poprawność algorytmu wynika z następujących przesłanek:

a) algorytm generuje wszystkie permutacje,

b) z przyjętego w paragrafie 2 założenia o topologicznym uporządkowaniu czynności wynika, że wyjściowa permutacja zdarzeń jest dopuszczalna (stanowi dopuszczalne uporządkowanie zdarzeń),

c) kolejne z generowanych uporządkowań są dopuszczalne tak długo, dopóki nie dokona się transpozycji takich sąsiednich zdarzeń *a* oraz *b*, że $\langle a, b\rangle \in \bar{U}$,

d) kolejne z generowanych uporządkowań są niedopuszczalne tak długo, dopóki:

– nie dokona się transpozycji tej pary zdarzeń *a* oraz *b*, których uprzednia transpozycja spowodowała niedopuszczalność poprzednich uporządkowań,

– uporządkowanie pozostałych zdarzeń (a jest to sprawdzane za pomocą procedury Sprawdz_porzadek) jest dopuszczalne.

Dla zobrazowania sposobu działania algorytmu w tabeli 1 podajemy kolejne wartości niektórych zmiennych dla sieci czynności na rysunku 1a.

Tabela 1

P[1]	P[2]	P[3]	P[4]	OK	a	b	Wywołane procedury
1	2	3	4	true			Nowa_siec
2	1	3	4	false	1	2	
2	3	1	4	false	1	2	
2	3	4	1	false	1	2	
3	2	4	1	false	1	2	
3	2	1	4	false	1	2	
3	1	2	4	true			Sprawdz_porzadek Nowa_siec
1	3	2	4	true			Nowa_siec
1	3	4	2	true			Nowa_siec
3	1	4	2	true			Nowa_siec
3	4	1	2	true			Nowa_siec
3	4	2	1	false	1	2	
4	3	2	1	false	1	2	
4	3	1	2	false	3	4	Sprawdz_porzadek
4	1	3	2	false	3	4	
1	4	3	2	false	3	4	
1	4	2	3	false	3	4	

Zbadajmy teraz złożoność obliczeniową całego algorytmu. Dla procedury Permutacje – z racji generowania wszystkich permutacji zbioru $n-2$ elementowego – otrzymujemy (bez uwzględnienia wywołań procedur pomocniczych) złożoność rzędu $O(n-2!)$. W procedurze Nowa_siec przegląda się raz wszystkie zdarzenia; ma więc złożoność $O(n)$. W procedurze Sprawdz_porzadek – w krańcowym przypadku – dla każdej czynności przegląda się wszystkie zdarzenia; daje to złożoność $O(nm)$. Zakładając zatem, że przy każdej z wygenerowanych permutacji są wywoływane obie procedury pomocnicze, otrzymujemy złożoność algorytmu rzędu $O(m(n-1)!)$. Pewne zmniejszenie tej złożoności można uzyskać, modyfikując przedstawione procedury. Otwartą natomiast sprawą pozostaje skonstruowanie algorytmu o złożoności liniowej względem liczby generowanych sieci, a więc algorytmu generującego nie wszystkie (jak czyni to powyższy algorytm), lecz jedynie dopuszczalne uporządkowania zdarzeń.

6. Istota metody eliminacji frontów sprzecznych

Idea kolejnej – a zaproponowanej w [2] – metody wyznaczania optymalnego rozwiązywania omawianego problemu rozdziału zasobów odnawialnych między czynności podzielne w słabo spójnej sieci czynności wynika bezpośrednio ze sformułowanej w paragrafie 3 uwagi, mówiącej, że czas realizacji przedsięwzięcia za pomocą co najmniej jednego, dowolnego frontu z każdej pary frontów sprzecznych musi być zerowy. Spełnienie tego warunku w omawianej metodzie osiąga się przez:

1) Utworzenie dla wyjściowej sieci czynności wszystkich dopuszczalnych zasobowo frontów czynności,

2) Wyznaczenie w utworzonym w 1 zbiorze frontów wszystkich par frontów sprzecznych. Wstępną wersję algorytmu generującego w dowolnym zbiorze frontów czynności rodzinę wszystkich par frontów sprzecznych przedstawiono w [12].

3) Sformułowanie i rozwiązanie zadania (por. [2]):

$$\sum_{f_j \in F} t_j = \text{minimum},$$

$$\sum_{f_j : i \in f_j} t_j = d_i \quad \text{dla} \quad i \in U,$$

$$t_i * t_j = 0 \quad \text{dla} \quad (f_i, f_j) \in FS,$$

$$t_j \geqslant 0 \quad \text{dla} \quad f_j \in F,$$

gdzie:

F jest zbiorem dopuszczalnych frontów czynności w sieci S,

f_j jest j-tym frontem tego zbioru,

t_j jest czasem realizacji przedsięwzięcia za pomocą frontu f_j,

FS jest zbiorem par wszystkich par frontów sprzecznych w zbiorze frontów F.

Zauważmy, że optymalne rozwiązanie przedstawionego wyżej zadania gwarantuje, iż wśród frontów o niezerowych czasach trwania nie będzie frontów sprzecznych. Ten sam efekt można oczywiście osiągnąć, zastępując warunki $t_i * t_j = 0$ jednym warunkiem

$$\sum_{(f_i, f_j) \in FS} (t_i * t_j) = 0.$$

Ostatnim etapem postępowania będzie oczywiście, jak we wszystkich metodach opartych na frontach czynności:

4) Wyznaczanie harmanogramu realizacji przedsięwzięcia.

7. Wyznaczanie frontów w słabo spójnych sieciach czynności

Problem generowania frontów czynności w słabo spójnych sieciach czynności jest problemem *NP*-trudnym [16]. Wynika to przykładowo z tego, że liczby frontów w takiej sieci nie można ograniczyć wielomianem rozmiarów sieci. Tak jest również w odniesieniu do sieci przedstawionej na rysunku 3. Liczba wszystkich frontów w tej sieci wynosi $3^{m/2}-1$ [8]. Wykazano również, iż *NP*-trudny jest także problem wyznaczania liczby frontów w dowolnej sieci [1], [8].

Tutaj skonstruujemy algorytm wyznaczania dopuszczalnych frontów czynności w dowolnej sieci czynności. Skorzystamy przy tym z przedstawionych w [10] propozycji dotyczących budowy takiego algorytmu. Inny algorytm wyznaczania frontów czynności w dowolnej sieci został opisany w [9]. Jest to algorytm „macierzowy". Stosując go generuje się najpierw wszystkie fronty jednoelementowe, następnie wszystkie dwuelementowe, potem trójelementowe itd. W [1] przedstawiono natomiast algorytm generowania wszystkich antyłańcuchów w wierzchołkowej sieci czynności o złożoności $O(ml)$, gdzie m jest liczbą łuków, a l – liczbą antyłańcuchów. Ponieważ łatwo można wykazać, iż każdemu antyłańcuchowi w sieci wierzchołkowej odpowiada wzajemnie jednoznacznie pewien front czynności w sieci łukowej (a takie tutaj rozpatrujemy), więc także i ten algorytm – po transformacji sieci łukowej w sieć wierzchołkową – można wykorzystać do generowania frontów czynności.

Na wstępie załóżmy – bez ograniczenia ogólności rozważań – że czynności należące do każdego frontu $f=(i_1, i_2, \ldots, i_w)$ są uporządkowane według rosnących numerów:

$$i_1 < i_2 < \ldots < i_w.$$

Zauważmy z kolei, że fronty $(k+1)$-elementowe $(k \geqslant 1)$ możemy tworzyć przez dodanie dla frontu k-elementowego takiej czynności, która nie leży z żadną czynnością frontu k-elementowego na jednej ścieżce. Jeżeli zatem $f=\{i_1^*, i_2^*, \ldots, i_k^*\}$ jest frontem czynności, to czynnością $k+1$ takiego frontu $(k+1)$-elementowego, że

$$f=\{i^*, i^*, \ldots, i^*, i\}$$

musi być taka czynność i, że:

$$i_{k+1} \in \bigcap_{r=1}^{k} C_{i_r}^*,$$

gdzie

$$C_i=\{j: j \in U \wedge \sim(i \prec j) \wedge i<j\}.$$

Korzystając z tego spostrzeżenia, można zbudować wiele algorytmów różniących się szczegółowymi rozwiązaniami. W [10] zaprezentowano ideę algorytmu generującego najpierw wszystkie fronty jednoelementowe, następnie wszystkie (niekoniecznie dopuszczalne) fronty dwuelementowe, wszystkie trójelementowe itd. Tutaj przedstawimy algorytm generujący dopuszczalne fronty czynności w kolejności leksykograficznej.

Główne etapy postępowania są następujące:

1. Tworzenie macierzy incydencji wierzchołków $M=[m_{ij}]_{n\times n}$, gdzie

$$m_{ij}=\begin{cases}1, & \text{gdy } \langle i,j\rangle \in \bar{U},\\ 0, & \text{gdy } \langle i,j\rangle \notin \bar{U}.\end{cases}$$

2. Wyznaczanie macierzy wierzchołkowego domknięcia tranzytowego $T=[t_{ij}]_{n\times n}$, gdzie

$$t_{ij}=\begin{cases}1, & \text{gdy } p_i \prec p_j,\\ 0, & \text{gdy } \sim(p_i \prec p_j).\end{cases}$$

Symbolem $\prec$ oznaczyliśmy relację poprzedzania w sieci czynności. Tak więc t_{ij} jest równe 1, gdy w sieci S istnieje droga od zdarzenia p_i do zdarzenia p_j; $t_{ij}=0$, gdy takiej drogi nie ma.

3. Wyznaczanie dla każdej czynności $\langle i,j\rangle \in \bar{U}$ zdefiniowanego wyżej zbioru $C_{\langle i,j\rangle}$; zbioru jej „nienastępników":

$$C_{\langle i,j\rangle}=\{\langle l,k\rangle: \langle l,k\rangle \in U \wedge [(l=i \wedge k>j) \vee (l>i \wedge t_{jl}=0)]\}.$$

Zbiory $C_{\langle i,j\rangle}$ można także wyznaczać bezpośrednio z macierzy tranzytywnego domknięcia łukowego. Wówczas $C_{\langle i,j\rangle}=\{j:\tilde{t}_{ij}=0\}$, gdzie $\tilde{T}$ jest macierzą tranzytywnego domknięcia łukowego. Ponieważ liczba łuków we wszystkich nietrywialnych sieciach czynności jest większa od liczby wierzchołków, więc dla zmniejszenia złożoności obliczeniowej odpowiednich procedur korzystamy tutaj z macierzy domknięcia wierzchołkowego.

4. Wyznaczanie dopuszczalnych frontów czynności w kolejności wynikającej z reguł porządkowania leksykograficznego.

Istotę procedury wyznaczania frontów przedstawimy na przykładzie frontów „rozpoczynających" się czynnością i (na początku $i=1$). Po wyznaczeniu wszystkich frontów tego typu przechodzi się do wyznaczania frontów „rozpoczynających" się czynnością $i+1$, a potem czynnością $i+2$, $i+3,\ldots,$ m:

a) utwórz front zawierający czynność i; zapisz go, ponieważ z założenia jest to front dopuszczalny; jeżeli $C_i=\emptyset$, przejdź do generowania frontów „zaczynających" się czynnością $i:=i+1$. Jeżeli i było równe m – koniec procedury;

b) sprawdź, czy będzie dopuszczalny utworzony uprzednio front rozszerzony o pierwszą nieanalizowaną czynność ze zbioru C_i; zbioru „nienastępników" czynności i (na początku jest to pierwsza czynność z tego zbioru); jeżeli tak, to zapisz ten front i przejdź do c); jeżeli nie, powtórz analizę dla następnej czynności ze zbioru C_i; jeżeli dla żadnej czynności z tego zbioru nie można utworzyć frontu dopuszczalnego, przyjmij $i := i+1$ oraz przejdź do a);

c) sprawdź, czy pierwsza z nieanalizowanych w tej serii badań czynność ze zbioru C_i (niech będzie to czynność j^*) należy do zbioru „nienastępników" wszystkich czynności należących już do tworzonego frontu (oczywiście poza i). Będzie to oznaczało, że czynność ta nie leży na jednej ścieżce z żadną z poprzednio wprowadzonych do frontu czynności; tworzy więc z nimi front w sieci S; jeżeli tak, przejdź do d) – jeżeli nie, weź do analizy następną czynność ze zbioru C_i; gdy analizowana czynność była ostatnią czynnością zbioru C_i:

– usuń z tworzonego frontu ostatnią z wprowadzonych do niego czynności (gdy nie jest to ostatnia czynność w C_i) lub dwie ostatnie czynności (gdy ostatnia czynność w generowanym froncie jest także ostatnią czynnością w C_i),

– rozpocznij analizę w c) od tej czynności, która w zbiorze C_i jest następna po ostatniej z czynności usuniętych z tworzonego frontu,

d) sprawdź dopuszczalność frontu poszerzonego o czynność j^*; jeżeli jest dopuszczalny, to go zapisz; przejdź do c).

PASCAL-o podobny zapis odpowiednich procedur zamieszczamy poniżej. Danymi algorytmu są:

$\bar{U}=\{\langle i, j\rangle\}$ – zbiór czynności w sieci S,
m, n – liczba czynności i zdarzeń w sieci,
D=D[j] – wektor dostępności środków,
q – liczba środków,
Z=Z[i, j] – macierz zapotrzebowań czynności na środki,
PW=PW[k] – wektor numerów pierwszych czynności wychodzących z kolejnych zdarzeń. Z założenia PW[n]=m+1.

Pozostałe oznaczenia:

M=M[i, j] – macierz incydencji wierzchołków w sieci S,
T=T[i, j] – macierz tranzytywnego domknięcia wierzchołkowego,
C=C[l] – zbiory „nienastępników" kolejnych czynności.

Zakładamy, że czynności w tych zbiorach są uporządkowane według rosnącej numeracji, a C[l, j] jest j-tą czynnością w zbiorze „nienastępników" czynności l.

Przez |C[l]| oznaczamy liczebność zbioru „nienastępników" tej czynności,

F – aktualnie generowany front czynności. F[k] jest k-tą czynnością tego frontu,

dfc – liczba czynności należących w danym momencie do frontu F,

i(l), j(l) – zdarzenie poprzedzające i następujące czynności l.

Algorytm generowania frontów czynności

(* Opracowany przez autora program obliczeniowy „FRONTY" pozwalający wyznaczyć za pomocą mikrokomputerów typu IBM PC m. in. wszystkie lub tylko dopuszczalne fronty czynności w dowolnych sieciach czynności jest dostępny w Centrum Informatyki AE Poznań. Opis programu jest natomiast zawarty w [14]*)

```
Procedure Macierz_Incydencji:
begin
  for i := 1 to n do
    for j := 1 to n do
      if (⟨i,j⟩ ∈ Ū) or (i=j) then
        M[i,j] := 1 else M[i,j] := 0;
end;

Procedure Domknięcie_Tranzytywne;
  (* Procedura ta oparta jest na znanym algorytmie Warshalla [21]*)
begin
  for i := 1 to n do
    for j := 1 to n do
      if T[j,i] <> 0 then
        for k := 1 to n do begin
          T[j,k] := T[j,k]+T[i,k];
          if T[j,k] <> 0 then T[j,k] := 1;
        end;
end;

Procedure Nienastępniki;
begin
  for l := 1 to m do begin
    i1 := i(l)+1;
    for k := l+1 to PW[i1] - 1 do
      C[l] := C[l] ∪ k;
    j := j(l);
    for k := i1 to n do
      if T[j,k]=0 then
        for i := PW[k] to PW[k+1]-1 do
          C[l] := C[l] ∪ i;
  end;
end;

Function Pozycja (C[i],j): integer;
{Pozycja czynności j w zbiorze C[i]}
begin
  if j=0 then Pozycja := 0
    else begin
```

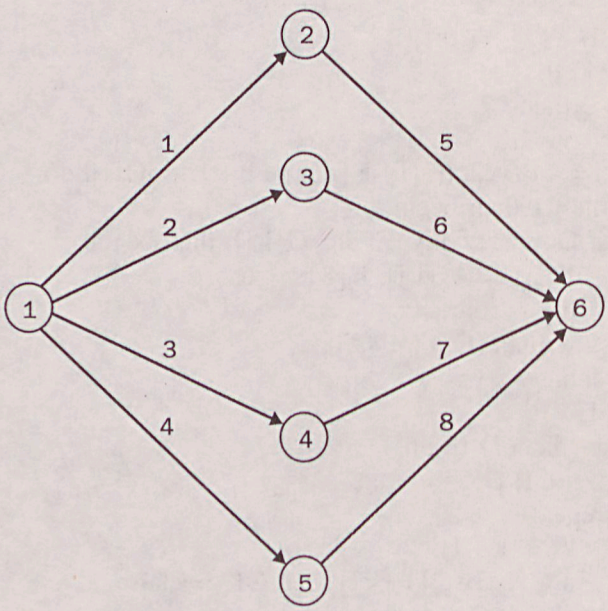
```
      k := 0; l := 0;
      repeat
        k := k+1;
        if j=C[i,k] then l := k
      until l <> 0;
      Pozycja := 1;
    end;
end;

Function Dopuszczalny (F, dfc, k):boolean;
{Sprawdzanie dopuszczalności frontu F ∪ k}
begin
  OK := true; j := 0;
  while OK and (j <=q) do begin
    j := j+1;
    if Σ Z[i,j]+Z[k,j]>D[j] then OK := false
      i∈F
  end;
  Dostępny := OK;
end;

Procedure Fronty_Czynności;
begin
  for l := 1 to m do begin
    dfc := 1;
    F[1] := 1; F[2] := 0;
    writeln (F[1]);
    if |C[l]|>0 then
      repeat
        for j := Pozycja (C[l], F[dfc+1])+1 to |C[l]| do
          if dfc=1 then begin
            if Dopuszczalny (F, dfc, C[l,j]) then begin
              F[2] := C[l,j]; F[3] := 0;
              dfc := dfc+1;
              writeln (F[1], F[2]);
            end;
          end
          else {dfc>1} begin
            k := 1; OK := true;
            repeat
              k := k+1;
              if C[l,j] ∉ C[F[k]] then OK := false
            until not OK or (k=dfc)
            if OK and Dopuszczalny (F, dfc, C[l,j]) then begin
```

```
                dfc := dfc+1;
                F[dfc] := C[l,j]; F[dfc+1] := 0;
                writeln (F[1],..., F[dfc]);
              end;
            end;
            if F[dfc]=max C[l,j] then
                          j
              dfc := dfc-2 else dfc := dfc-1;
      until dfc <=0;
    end;
end;
begin (m, n, Ū, D, q, Z, PW) {Początek algorytmu}
  Macierz_Incydencji;
  T := M;
  Domknięcie_Tranzytywne;
  Nienastępniki;
  Fronty_Czynności;
end.
```

Dla zobrazowania przebiegu obliczeń wykonanych za pomocą procedury Fronty_Czynności w tabeli 2 podajemy wartości kilku najważniejszych zmiennych dla początkowych iteracji przy wyznaczaniu frontów dla sieci przedstawionej na rysunku 4.



Rys. 4.

Przed analizą tabeli zauważmy, że:
$C_1=\{2, 3, 4, 6, 7, 8\}$; $C_2=\{3, 4, 5, 7, 8\}$;
$C_3=\{4, 5, 6, 8\}$; $C_4=\{5, 6, 7\}$;
$C_5=\{6, 7, 8\}$; $C_6=\{7, 8\}$;
$C_7=\{8\}$; $C_8=\emptyset$.

Tabela 2

j	C[1,j]	F[1]	F[2]	F[3]	F[4]	OK	dfc	Uwagi
		1	0				1	front 1
1	2	1	2	0		true	2	front 2
2	3	1	2	3	0	true	3	front 3
3	4	1	2	3	4	true	4	front 4
4	6	1	2	3	4	false		$6 \notin C_2$
5	7	1	2	3	4	false		$7 \notin C_3$
6	8	1	2	3	4	false		$8 \notin C_4$
		koniec pętli j					3	
5	7	1	2	3	4	false		$7 \notin C_3$
6	8	1	2	3	8	true	4	front 5
		koniec pętli j					2	
3	4	1	2	4	0	true	3	front 6
4	6	1	2	4	0	false		$6 \notin C_2$
5	7	1	2	4	7	true	4	front 7
6	8	1	2	4	7	false		$8 \notin C_4$
		koniec pętli j					3	
6	8	1	2	4	7	false		$8 \notin C_4$
		koniec pętli j						
5	7	1	2	7	0	true	3	front 8

Poprawność przedstawionego wyżej algorytmu wyznaczania frontów czynności wynika z dwu faktów:

1. w [10] udowodniono, że algorytmy generowania frontów czynności oparte na wyznaczaniu przekrojów zbiorów C_i – zbiorów „nienastępników" poszczególnych czynności – a tego typu jest też prezentowany wyżej algorytm, wyznaczają wszystkie fronty czynności w dowolnych, kanonicznych sieciach czynności,

2. prezentowany tutaj algorytm tworzy wszystkie fronty jednoelementowe oraz tworzy i bada pod względem dopuszczalności zasobowej wszystkie z tych frontów $(k+1)$-elementowych $f=\{i_1^*, i_2^*, \ldots, i_k^*, i\}$ (dla $k \geqslant 1$):

a) które można wyznaczyć, korzystając z reguły mówiącej, że:

$$i_{k+1} \in \bigcap_{r=1}^{k} C_{i_r}^*,$$

b) dla których dopuszczalny jest front $f=\{i_1^*, i_2^*, \ldots, i_k^*\}$.

Mówiąc inaczej, algorytm ten tworzy wszystkie te dopuszczalne fronty czynności, które można utworzyć, korzystając z reguły podanej w 2a). Omawiany algorytm nie jest z oczywistych względów algorytmem wielomianowym względem rozmiarów wyjściowej sieci czynności. Jest natomiast o tyle efektywny, że jego złożoność obliczeniowa jest prawie liniowa w stosunku do liczby frontów w sieci S. Dokładnie rzecz biorąc, jego złożoność jest równa sumie złożoności procedury Fronty_Czynności – a złożoność tej procedury wraz z wywołanymi przez nią procedurami pomocniczymi jest właśnie liniowa względem liczby frontów – oraz złożoności procedur Macierz_Incydencji, Domknięcie_Tranzytywne i Nienastępniki – a złożoność tych procedur wynosi $O(n^3)$. Złożoność całego algorytmu wynosi zatem $O(kn^3)$, gdzie k jest liczbą frontów czynności, a n liczbą wierzchołków w sieci S.

Literatura

[1] **Ball M. O., Provan J. S.,** Calculating bounds on reachability and connectedness in stochastic networks, Networks 13 (1983).

[2] **Dubina A.,** Minimalizacja czasu realizacji przedsięwzięcia opisanego siecią słabo spójną przy uwzględnieniu rozdziału zasobów, Zeszyty Naukowe Instytutu Organizacji Przemysłu Maszynowego 2/88 (maszynopis artykułu).

[3] **Garey M. R.,** Computers and Intractability – A Guide of the Theory of NP-Completness, W. Freeman, San Francisko 1979.

[4] **Gedymin O., Dubina A.,** Zagadnienia alokacji środków w planowaniu sieciowym, Wydawnictwa Uczelniane Politechniki Lubelskiej, Lublin 1983.

[5] **Ignasiak E.,** Programowanie sieciowe, PWE, Warszawa 1972.

[6] **Lawler E. L., Labetoulle J.,** On preemptive scheduling of unrelated processors by linear programming, Journal of the ACM, 4 (1978).

[7] **Lipski W.,** Kombitoryka dla programistów, WNT, Warszawa 1982.

[8] **Kamburowski J.,** O złożoności obliczeniowej wybranych problemów budowy i analizy sieciowego modelu przedsięwzięcia, w: Problemy budowy i zastosowania modeli ekonomicznych, SGPiS, Warszawa 1987.

[9] **Kobus K.,** O pewnym algorytmie leksykograficznego porządkowania frontów w zagadnieniach programowania sieciowego, Przegląd Statystyczny R. XXIX, 1982, z. 1/2.

[10] **Rzemykowski Z.,** Algorytm tworzenia i porządkowania frontów czynności, Przegląd Statystyczny, R. XXVII, 1980, z. 1/2.

[11] **Rzemykowski Z.,** O metodzie frontów maksymalnych, Przegląd Statystyczny, R. XXXIII, 1986, z. 4.

[12] **Rzemykowski Z.,** „Jednoetapowe" metody optymalnej alokacji zasobów odnawialnych w słabo spójnych sieciach czynności, w: Teoria grafów i jej zastosowanie do optymalizacji przedsięwzięć inwestycyjnych i badawczo-rozwojowych. Opracowanie w ramach CPBP 10.09, AE Poznań 1988 (maszynopis).

[13] Problem optymalnego rozdziału zasobów odnawialnych między czynności podzielne jako zadanie PCL (przy pojedynczych sposobach wykonywania czynności), w: Teoria grafów i jej zastosowanie do optymalizacji przedsięwzięć inwestycyjnych i badawczo-rozwojowych. Opracowanie w ramach CPBP 10. 09, AE Poznań 1990 (maszynopis).

[14] Nowa wersja programu fronty czynności, w: Teoria grafów i jej zastosowanie do optymalizacji przedsięwzięć inwestycyjnych i badawczo-rozwojowych. Opracowanie w ramach CPBP 10.09, AE Poznań 1990 (maszynopis).

[15] **Słowiński R.**, Two approaches to problem of resources allocation among project activities – a comparative study, Journal of the Operational Research Society, vol. 8 (1980).

[16] **Słowiński R.**, Algorytmy sterowania rozdziałem zasobów różnych kategorii w kompleksie operacji, Seria Rozprawy nr 114, Wydawnictwo Politechniki Poznańskiej, 1980.

[17] **Słowiński R., Węglarz J.**, Minimalno-czasowy model sieciowy z różnymi sposobami wykonywania czynności, Przegląd Statystyczny, R. XXIV, z. 3 (1977).

[18] **Węglarz J.**, Sterowanie w systemach typu kompleks operacji, PWN, Warszawa-Poznań 1981.

[19] **Węglarz J., Błażewicz J., Cellary W., Słowiński R.**, An automatic revised simplex method for constrained resource network scheduling, ACM Transactions and Mathematical Software, vol. 3 (1977).

[20] **Ullman J. D.**, Złożoność obliczeniowa problemów szeregowania, w: Teoria szeregowania zadań, red. E. F. Coffman, WNT, Warszawa 1980.

[21] **Warshall S. A.**, Theorem on Boolean Matrices, Journal of the Association of Computing Machinery, vol. 9 (1962).

[22] **Zuchowicki S. I., Radczik I. A.**, Matematiczeskije metody setewogo planirowania, Nauka, Moskwa 1965.