ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1990
Seria: AUTOMATYKA z.100 Nr kol.1082

Eugeniusz Nowicki
Czesław Smutnicki

Politechnika Wrocławska
Instytut Cybernetyki Technicznej

ANALIZA NAJGORSZEGO PRZYPADKU ALGORYTMÓW APROKSYMACYJNYCH DLA PROBLEMU PRZEPŁYWOWEGO

Streszczenie. W pracy rozważa się permutacyjny przepływowy problem szeregowania zadań z kryterium minimalizacji maksymalnego terminu zakończenia wykonywania wszystkich zadań. Dla tego problemu podano szereg algorytmów aproksymacyjnych oraz przedstawiono, otrzymane ostatnio, wyniki analizy najgorszego przypadku.

1. Wstęp

W literaturze przedstawiono wiele algorytmów aproksymacyjnych dla jednego z najbardziej typowych problemów szeregowania, jakim jest wielomaszynowy permutacyjny problem przepływowy. Algorytmy porównywano na drodze eksperymentalnej m.in. w pracach [3], [13], [16], [17]. W wyniku tych badań stwierdzono, że kilka z nich zasługuje na szczególną uwagę ze względu na dokładność i czas obliczeń. Są to: algorytm Palmera [14], algorytm Campbella i in. [2], algorytmy RA, RACS i RAES podane przez Dannenbringa [3], algorytm Nawaza i in. [9] oraz ostatnio podany algorytm Osmana i Pottsa [13]. Spośród wymienionych, algorytm Palmera i algorytm RA Dannenbringa charakteryzują się najmniejszym czasem obliczeń. Z kolei algorytm Nawaza i algorytm Osmana i Pottsa generują najbardziej dokładne rozwiązanie.

Do niedawna brak było istotnych rezultatów dotyczących analizy najgorszego przypadku tych algorytmów. Dopiero ostatnio pojawiły się trzy prace [10], [11], [12], które w znaczący sposób wypełniły tę lukę. W niniejszej pracy przedstawiamy szczegółowo wspomniane wyżej algorytmy i omawiamy najważniejsze wyniki dotyczące analizy najgorszego przypadku zawarte w [10], [11], [12], a także inne najnowsze rezultaty badań.

Permutacyjny m-maszynowy problem przepływowy formułuje się następująco: Dany jest zbiór zadań $J=\{1,2,\ldots,n\}$, które należy wykonać na maszynach ze zbioru $M=\{1,2,\ldots,m\}$. Każde zadanie jest wykonywane kolejno na maszynach $1,2,\ldots,m$. Zadanie j na maszynie i wykonuje się w czasie $p_{ij}\geq 0$, $i\in M$, $j\in J$. Zakłada się, że: (i) wykonywanie zadania na danej maszynie nie może być przerywane, (ii) każda maszyna wykonuje nie więcej niż jedno zadanie w dowolnej chwili czasowej oraz (iii) kolejność wykonywania zadań na wszystkich maszynach jest identyczna. Poszukuje się kolejności wykonywania zadań, któ-

ra minimalizuje termin zakończenia wykonywania wszystkich zadań.

Oznaczmy przez π dowolną permutację elementów zbioru J, zaś przez Π zbiór wszystkich takich permutacji. Dalej, przez $C_{max}(\pi)$ oznaczmy termin zakończenia wykonywania wszystkich zadań dla kolejności wykonywania zadań określonej przez $\pi\in\Pi$. Permutację $\pi^*\in\Pi$, dla której zachodzi

$$C_{max}(\pi^*) = \min_{\pi\in\Pi} C_{max}(\pi) \tag{1}$$

będziemy nazywać rozwiązaniem optymalnym.

2. Algorytmy aproksymacyjne

Algorytmy aproksymacyjne dla problemu $F||C_{max}$ można podzielić na dwie następujące grupy:

(a) metody konstrukcyjne wyznaczające rozwiązania poprzez analizę pewnego problem zastępczego,

(b) metody poprawy rozwiązania poprzez lokalne przeszukiwanie zbioru rozwiązań dopuszczalnych.

W obrębie grupy (a) można wyróżnić trzy podgrupy metod:

(a.1) Metody priorytetowe: Dla każdego zadania definiuje się priorytet jako funkcję czasów wykonywania tego zadania na poszczególnych maszynach, a następnie szereguje się zadania w kolejności nierosnących wartości priorytetów.

(a.2) Metody bazujące na problemie $F2||C_{max}$: Problem wyjściowy sprowadza się do jednego lub kilku pomocniczych problemów $F2||C_{max}$ poprzez zdefiniowanie dla każdego zadania czasów wykonywania na dwóch zastępczych maszynach. Po rozwiązaniu problemów pomocniczych permutacje o najmniejszej wartości C_{max} dla problemu wyjściowego przyjmuje się jako rozwiązanie przybliżone.

(a.3) Metody bazujące na permutacjach częściowych: W fazie wstępnej ustala się listę zadań przez zastosowanie pewnej metody priorytetowej. W fazie zasadniczej konstruuje się ciąg n permutacji częściowych poczynając od permutacji jednoelementowej i kończąc na permutacji n-elementowej. Kolejna permutacja częściowa jest tworzona na bazie poprzedniej i kolejnego zadania z listy. Permutacja otrzymana w wyniku fazy zasadniczej jest rozwiązaniem przybliżonym.

Typowym reprezentantem grupy (a.1) jest algorytm (P) zaproponowany przez Palmera [14]. W algorytmie tym priorytety są definiowane następująco:

$$s_j = \sum_{i=1}^{m} (2i-m-1)p_{ij}, \quad j=1,\dots,n, \tag{2}$$

zaś jego złożoność obliczeniowa jest $O(nm+n\log n)$. Ogólnie można przyjąć funkcję priorytetu równą $s_j = \sum_{i=1}^{m} \alpha_i p_{ij}$, $j=1,\dots,m$, gdzie α_i są pewnymi waga-

mi; celowe wydaje się przy tym nałożenie na α_i następujących warunków

$$\alpha_i = -\alpha_{m-i+1},\ \alpha_i \le 0,\ i=1,\dots,\lfloor m/2 \rfloor \tag{3}$$

Reprezentantami grupy (a.2) są m.in. algorytmy podane Dannenbringa [3] (D), Rocka i Schmidta [15] (R), Campbella i in. [2] (C). Algorytm (D) konstruuje jeden problem pomocniczy $F2||C_{max}$ z czasami wykonywania

$$a_j = \sum_{i=1}^{m} (m-i+1)p_{ij},\qquad b_j = \sum_{i=1}^{m} ip_{ij}\quad,\ j=1,\dots,n, \tag{4}$$

odpowiednio na pierwszej i drugiej maszynie. Algorytm (R) podobnie konstruuje jeden problem pomocniczy $F2||C_{max}$ z czasami wykonywania

$$a_j = \sum_{i=1}^{\lceil m/2 \rceil} p_{ij},\qquad b_j = \sum_{i=\lceil m/2 \rceil+1}^{m} p_{ij}\quad,\ j=1,\dots,n. \tag{5}$$

Z kolei algorytm (C) konstruuje m-1 problemów pomocniczych $F2||C_{max}$ z czasami wykonywania

$$a_j^k = \sum_{i=1}^{k} p_{ij},\qquad b_j^k = \sum_{i=m-k+1}^{m} p_{ij},\qquad j=1,\dots,n, \tag{6}$$

odpowiednio na pierwszej i drugiej maszynie dla k-tego problemu pomocniczego, $k=1,\dots,m-1$. Algorytmy (D) i (R) mają złożoność obliczeniową $O(mn+n\log n)$, zaś algorytm (C) $O(m^2n+mn\log n)$. Ogólnie można przyjąć, że czasy a_j, b_j dla problemu pomocniczego są definiowane następująco

$$a_j = \sum_{i=1}^{m} \alpha_i p_{ij},\qquad b_j = \sum_{i=1}^{m} \beta_i p_{ij}\quad,\ j=1,\dots,n, \tag{7}$$

gdzie α_i, β_i są pewnymi wagami; celowe wydaje się przy tym nałożenie na α_i β_i następujących warunków: α_i nierosnące względem i oraz

$$\alpha_i = \beta_{m-i+1},\ \alpha_i,\beta_i \ge 0,\ i=1,\dots,m. \tag{8}$$

W grupie (a.3) przedstawicielem jest algorytm wstawiania (N) podany przez Nawaza i in. [9]. W fazie wstępnej tego algorytmu tworzy się pomocniczą listę zadań β z wykorzystaniem priorytetów w następującej postaci

$$s_j = \sum_{i=1}^{m} p_{ij}\ ,\qquad j=1,\dots,n. \tag{9}$$

W k-tym kroku fazy zasadniczej $(k=1,\dots,n)$ na bazie permutacji częściowej $\sigma=(\sigma(1),\dots,\sigma(k-1))$, otrzymanej w poprzednim kroku, oraz zadania $\beta(k)$ generuje się k permutacji przez wstawianie zadania $\beta(k)$ na wszystkie możliwe pozycje w σ, tzn. permutacje postaci $(\beta(k),\sigma(1),\dots,\sigma(k-1))$,

$(\sigma(1),\beta(k),\sigma(2),\ldots,\sigma(k-1)),\ldots,(\sigma(1),\ldots,\sigma(k-1),\beta(k))$. Permutacja o najmniejszej wartości C_{max}, wybrana spośród wygenerowanych, jest przyjmowana jako permutacja częściowa do następnego kroku tej fazy. Złożoność obliczeniowa algorytmu (N) jest $O(mn^3)$. Możliwa jest pewna modyfikacja (N1) tego algorytmu, zmniejszająca jego złożoność obliczeniową do $O(mn^2)$. Polega ona na tym, że w k-tym kroku fazy zasadniczej generuje się tylko 2 permutacje postaci $(\beta(k),\sigma(1),\ldots,\sigma(k-1))$, $(\sigma(1),\ldots,\sigma(k-1),\beta(k))$.

W obrębie grupy (b) można wyróżnić dwie podgrupy metod:

(b.1) Szukanie zstępujące: Dla każdej permutacji n-elementowej definiuje się jej otoczenie. Proces szukania rozpoczyna się od pewnej permutacji początkowej i polega na poszukiwaniu w jej otoczeniu permutacji z mniejszą wartością C_{max}. Jeżeli takiej permutacji nie znajdziemy, to proces poszukiwania się kończy. W przeciwnym wypadku, dla otrzymanej permutacji konstruujemy otoczenie i proces poszukiwań jest kontynuowany. Możliwe są dwie alternatywne metody przeszukiwania: (i) metoda pierwszej poprawy, (ii) metoda największej poprawy. W metodzie (i) przeglądamy otoczenie do chwili znalezienia pierwszej "lepszej" permutacji; przeglądanie można przeprowadzać wg wcześniej ustalonej kolejności (takiej samej dla każdego otoczenia) lub losowo. W metodzie (ii) przeglądane jest zawsze całe otoczenie i wybierana jest permutacja o najmniejszej wartości C_{max}.

(b.2) Błądzenie losowe: W metodach tych proces przeglądania otoczenia danej permutacji π, polega na poszukiwaniu permutacji o wartości C_{max} nie większej niż $C_{max}(\pi)+\delta$, gdzie $\delta>0$ jest wybierane losowo. Dla permutacji tej konstruujemy otoczenie i proces poszukiwań jest kontynuowany. Podobnie jak w (b.1), przeglądanie otoczenia można przeprowadzać wg ustalonej kolejności lub losowo.

W metodach z p. (b) zakłada się, że pierwsza permutacja początkowa została dostarczona z zewnątrz, tzn. ustalona arbitralnie, wylosowana lub wyznaczona przy użyciu jednej z metod opisanych w p. (a). Zastosowany warunek stopu powoduje, że zwykle przeglądaniu podlega bardzo duża liczba otoczeń, co z kolei wpływa niekorzystnie na czas obliczeń. Z tego też względu wprowadza się pomocnicze kryterium stopu zatrzymujące poszukiwania po przeglądnięciu ustalonej liczby otoczeń. Otoczenie permutacji π definiuje się jako pewien podzbiór $O(\pi)\subseteq\Pi$. Typowo rozważane są dwa rodzaje otoczeń: otoczenie wymian oraz otoczenie przesuwań. Otoczenie wymian $O_w(\pi)$ jest tworzone poprzez wybór pozycji i,j, $1\le i<j\le n$ oraz zamianę miejscami zadań $\pi(i),\pi(j)$, tzn.

$$O_w(\pi) = \{(\pi(1),\ldots,\pi(i-1),\pi(j),\pi(i+1),\ldots,\pi(j-1),\pi(i),\pi(j+1),\ldots,\pi(n)):\ 1\le i<j\le n\}.$$

Specyficznym rodzajem otoczenia wymian jest otoczenie wymian sąsiednich $O_{ws}(\pi)$ zawierające n-1 permutacji, tzn.

$$O_{ws}(\pi) = \{(\pi(1),\ldots,\pi(i-1),\pi(i+1),\pi(i),\pi(i+2),\ldots,\pi(n)):\ 1\le i<n\}.$$

Wyróżnia się dwa typy otoczeń przesuwań: otoczenie przesuwań w prawo $O_{pp}(\pi)$ i otoczenie przesuwań w lewo $O_{pl}(\pi)$. Otoczenie przesuwań w prawo (w lewo) jest tworzone poprzez wybór pozycji i,j, $1 \le i < j \le n$ oraz przesunięcie zadania $\pi(i)$ ($\pi(j)$) bezpośrednio za zadanie $\pi(j)$ (bezpośrednio przed zadanie $\pi(i)$), tzn.

$$O_{pp}(\pi) = \{(\pi(1),..,\pi(i-1),\pi(i+1),..,\pi(j),\pi(i),\pi(j+1),..,\pi(n)): 1 \le i < j \le n\}.$$

$$O_{pl}(\pi) = \{(\pi(1),..,\pi(i-1),\pi(j),\pi(i),..,\pi(j-1),\pi(j+1),..,\pi(n)): 1 \le i < j \le n\}.$$

Otoczenia $O_w(\pi)$, $O_{pp}(\pi)$, $O_{pl}(\pi)$ zawierają $n(n-1)/2$ permutacji. Zwykle otoczenie przesuwań (łączne) definiuje się jako $O_p(\pi)=O_{pp}(\pi)\cup O_{pl}(\pi)$.

Reprezentantami grupy (b.1) są m.in. algorytmy (D1) i (D2) podane przez Dannenbringa [3]. W algorytmach tych stosowane są otoczenia wymian sąsiednich, zaś permutacja początkowa jest permutacja wyznaczona algorytmem (D) Dannenbringa. Przeszukaniu podlega całe otoczenie zgodnie z metodą największej poprawy. Algorytm (D1) kończy działanie po przeszukaniu otoczenia permutacji początkowej, zaś algorytm (D2) kontynuuje przeszukania kolejnych otoczeń do chwili znalezienia permutacji lokalnie optymalnej. Z kolei w pracy [13] badano dwa warianty algorytmu składającego się z dwóch faz: przeszukanie otoczenia wymian $O_w(\pi)$ oraz przeszukanie otoczenia przesuwań $O_p(\pi)$. Permutacja początkowa dla fazy pierwszej była wyznaczana algorytmem (N) (dla pierwszego wariantu) lub połączeniem algorytmów (P+C+D) (dla drugiego wariantu). Permutacja początkowa dla fazy drugiej jest permutacją lokalnie optymalną wyznaczoną w pierwszej fazie. Otoczenia są przeszukiwane do pierwszej poprawy wg ustalonej kolejności. Korzystając z wyników pracy [5], można uzyskać znaczną poprawę efektywności przeszukiwania otoczeń (np. [12]). Inne sposoby definiowania otoczenia podano m.in. w [7], [18].

Reprezentantami grupy (b.2) są 4 warianty algorytmu opisane w [13]. Warianty te różnią się między sobą sposobem budowy otoczenia (otoczenie wymian, otoczenie przesuwań) oraz kolejnością przeglądania permutacji z otoczenia (uporządkowane, losowe). Oznaczmy przez π' aktualnie przeglądaną permutację z otoczenia permutacji π, zaś przez Δ różnicę $C_{max}(\pi')-C_{max}(\pi)$. Permutacja π' jest akceptowana jako nowa permutacja generująca otoczenie, jeżeli $\Delta \le 0$ lub $\Delta > 0$ i $R \le e^{-\Delta/T}$, gdzie R jest liczbą wylosowaną wg rozkładu równomiernego [0,1], zaś T jest parametrem zwanym "temperaturą". Parametr T ma charakter dynamiczny i maleje wraz ze wzrostem liczby przeglądanych otoczeń. Dodatkowy warunek stopu powoduje zatrzymanie obliczeń po przeglądnieciu ustalonej liczby otoczeń. We wszystkich wariantach permutację początkową wybiera się arbitralnie, np. (1,2,...,n).

3. Analiza najgorszego przypadku

Oznaczmy przez π^A permutację generowaną przez odpowiedni algorytm aprok-

symacyjny (A). Niech $C^A=C_{max}(\pi^A)$ oraz $C^*=C_{max}(\pi^*)$. Poniżej podano przegląd rezultatów dotyczących analizy najgorszego przypadku algorytmów C, D, D1, D2, N, N1, R, P oraz algorytmu C-k zdefiniowanego następująco: algorytm (C-k) jest uproszczoną wersją algorytmu (C), w której uwzględnia się tylko jeden k-ty problem pomocniczy. Przy omawianiu przykładów osiągalności będziemy wielokrotnie zakładać, że zbiór zadań rozbija się na $v\geq 1$ podzbiorów identycznych zadań $J_1,\ldots,J_v$. Wtedy dla uproszczenia notacji każde zadanie z podzbioru J_j indeksowane będzie przez j. Dodatkowo, dla określania kolejności wykonywania zadań wprowadzony został symbol $(\beta)_k$ na oznaczenie k-krotnego powtórzenia permutacji β (częściowa permutacja zbioru $\{1,\ldots,v\}$); przykładowo $(3,1)_2$ przy v=3 określa sekwencję: jedno zadanie ze zbioru J_3, jedno zadanie ze zbioru J_1, jedno zadanie ze zbioru J_3, jedno zadanie ze zbioru J_1.

Omawianie rezultatów rozpoczniemy od algorytmu (C-k).

Twierdzenie 1. [10] Dla algorytmu (C-k), k=1,...,m-1 zachodzi

$$C^{C-k}/C^* \leq |m/2-k| + m/2$$

i ograniczenie to jest osiągalne.

Z powyższego twierdzenia wynika natychmiast następujący wniosek.

Wniosek 1. [10] Dla algorytmu (C) zachodzi

$$C^C/C^* \leq \lceil m/2 \rceil$$

i ograniczenie to jest osiągalne.

W tabeli 1 podano przykład osiągalności oszacowania z wniosku 1 ograniczając się do przypadku m=5 maszyn. Przykład ten składa się z trzech podzbiorów zadań o licznościach odpowiednio s^2, s, 1, gdzie s liczba naturalna wiek-

podzbiór	1	2	3
m = 1			
2			
3	1	1	1+s
4		s	
5			s^2
n_j	s^2	s	1

Tab. 1. Przykład dla algorytmu (C): n_j jest licznością j-tego podzbioru; pole puste oznacza zerowy czas wykonywania zadania na maszynie.

sza od jeden. Zadania w każdym podzbiorze są identyczne i indeksowane nume-

rem tego podzbioru. Zauważmy, że sekwencja $(1)_s(2)_s(3)$ jest sekwencją optymalną dla każdego problemu pomocniczego w algorytmie (C). Stąd mamy $\pi^C=(1)_s(2)_s(3)$ oraz $C^C=3s^2+1$. Z kolei sekwencja optymalna jest $\pi^*=(3)(2)_s(1)_s$ oraz $C^*=s^2+2s+1$. Tak więc, C^C/C^* dąży do $3=\lceil 5/2\rceil$, przy s dążącym do nieskończoności. Odpowiednie przykłady osiągalności oszacowań z Tw.1 oraz Wn.1 dla dowolnego m podano w pracy [10].

Twierdzenie 2. [15] Dla algorytmu (R) zachodzi

$$C^R/C^* \le \lceil m/2\rceil$$

i ograniczenie to jest osiągalne. ■

Przykład, dla którego ograniczenie jest osiągalne, ma n=m zadań. Czas wykonywania i-tego zadania na i-tej maszynie jest równy jeden, a pozostałe czasy zero. Z algorytmu (R) wynika, że $\pi^R=(\lceil m/2\rceil+1,\dots,m,1,2,\dots,\lceil m/2\rceil)$, zaś $C^R=\lceil m/2\rceil$. Z kolei $\pi^*=(m,m-1,\dots,1)$ i $C^*=1$. Stąd otrzymujemy $C^R/C^*=\lceil m/2\rceil$.

Prezentacja kolejnych rezultatów wymaga uprzedniego zdefiniowania niezbędnych pojęć. Podobnie jak w pracach [11], [12] definujemy dla $t\in\{1,\dots,r\}$, $r=\lfloor m/2\rfloor$ funkcje

$$g_o(t)=t+\frac{r^2+r}{2t}, \qquad g_e(t)=t+\frac{r^2-t}{2t-1}$$

oraz oznaczamy odpowiednie minima przez $g_z^*=\min\{g_z(t):\ t=1,\dots,r\}$, $z\in\{o,e\}$. W wymienionych pracach pokazano, że funkcja $g_z(t)$ osiąga swe minimum dla $t=\lceil t_z^*\rceil$ $z\in\{o,e\}$, gdzie

$$t_o^*=(\sqrt{2r^2+2r+1}-1)/2, \qquad t_e^*=r/\sqrt{2}.$$

oraz że wartości g_o^*, g_e^* są obie równe w przybliżeniu $m/\sqrt{2}$.

Twierdzenie 3. [11],[12] Dla algorytmu (A), $A\in\{P,D,D1,D2\}$ zachodzi:

(i) dla m nieparzystego

$$C^A/C^* \le g_o^*$$

i oszacowanie to jest osiągalne,

(ii) dla m parzystego

$$C^A/C^* \le g_e^* + \frac{1}{2(2\lceil t_e^*\rceil-1)}$$

oraz istnieje przykład, dla którego $C^A/C^* \ge g_e^*$. ■

W tabeli 2 podano przykład osiągalności oszacowania z Tw.3 dla algorytmu (P) ograniczając się do przypadku m=5 maszyn. Przykład ten składa się z pięciu podzbiorów identycznych zadań o liczności s każdy, gdzie s liczba naturalna większa od jeden.

podzbiór	1	2	3	4	5
m = 1	1/4	1/2			1/4
2				1	
3			1		
4		1			
5	1/4			1/2	1/4
n_j	s	s	s	s	s

Tab. 2. Przykład dla algorytmu (P): n_j jest licznością j-tego podzbioru; pole puste oznacza zerowy czas wykonywania zadania na maszynie.

Z algorytmu (P) dostaniemy $\pi^P=(5)_s(4)_s(3)_s(2)_s(1)_s$ oraz $C^P\geq 3.5s$. Z kolei dla sekwencji $\pi^o=(1,2,3,4,5)_s$ mamy $C_{max}(\pi^o)\leq s+2$. Tak więc, $C^P/C^* \geq C^P/C_{max}(\pi^o) \geq 3.5s/(s+2)$ i przy $s\to\infty$ wartość $C^P/C^* \to 3.5$, ponieważ $C^P/C^*\leq$ ≤ 3.5; odpowiednia wartość g_o^* dla m=5 wynosi 3.5.

Z kolei w tabeli 3 podano przykład osiągalności oszacowania z Tw.3 dla algorytmów D,D1,D2, ograniczając się do przypadku m=5 maszyn. Przykład ten składa się z pięciu podzbiorów identycznych zadań o licznościach s,4s,4s, 8s,8s odpowiednio, gdzie s liczba naturalna większa od jeden. Z algorytmu

podzbiór	1	2	3	4	5
m = 1	1/4	1/8			1/32
2				1/8	
3			1/4		
4		1/4			
5	1/4			1/16	1/32
n_j	s	4s	4s	8s	8s

Tab. 3. Przykład dla algorytmów D,D1,D2: n_j jest licznością j-tego podzbioru; pole puste oznacza zerowy czas wykonywania zadania na maszynie.

(D) mamy $\pi^D=(5)_{8s}(4)_{8s}(3)_{4s}(2)_{4s}(1)_s$ oraz $C^D\geq 3.5s$. Z kolei dla sekwencji $\pi^o=((1)_1(2)_4(3)_4(4)_8(5)_8)_s$ mamy $C_{max}(\pi^o)\leq s+2$. Stąd, podobnie jak w poprzednim przykładzie, wartość $C^D/C^*\to 3.5$ przy $s\to\infty$. Algorytm (D1) dostarczy np. sekwencję $\pi^{D1}=(5)_{8s}(4)_{8s}(3)_{4s}(2)_{4s-1}(1)_1(2)_1(1)_{s-1}$, dla której $C^{D1}\geq 3.5s-1/4$, co prowadzi do otrzymania identycznego rezultatu na oszacowanie C^{D1}/C^*. Z kolei algorytm (D2) dostarcza sekwencję

$n^{D2}=(5)_{8s-4}(4)_1(5)_4(4)_{8s-3}(3)_1(4)_2(3)_{4s-2}(2)_1(3)_1(2)_{4s-2}(1)_1(2)_1(1)_{s-1}$,
dla której $C^{D2}\geq 3.5s-7/8$, co również nie zmienia końcowego wyniku oszacowania. Odpowiednie przykłady ogólne (dla dowolnego m) przedstawiono w pracach [11],[12]; tam też podano wzmocnienie dolnych oszacowań współczynnika najgorszego przypadku dla algorytmu (P) w przypadku m parzystego.

Z przedstawionej analizy wynika, że wpółczynniki najgorszego przypadku algorytmów C,D,D1,D2,P są nie mniejsze niż $\lceil m/2\rceil$. Dodatkowo, dla algorytmów priorytetowych (a.1) można pokazać następujące twierdzenie, które podajemy bez dowodu.

Twierdzenie 4. W grupie metod priorytetowych z funkcją priorytetu postaci $s_j=\Sigma_{i=1}^{m}\alpha_i p_{ij}$, najmniejszą wartość współczynnika najgorszego przypadku, równą $\lceil m/2\rceil$, ma algorytm, dla którego $\alpha_i=-1$, $i=1,\ldots,\lfloor m/2\rfloor$, $\alpha_i=1$, $i=\lceil m/2\rceil+1,\ldots,m$ oraz $\alpha_{\lceil m/2\rceil}=0$ w przypadku m nieparzystego. ■

Podobnie, można pokazać, że dla algorytmów z grupy (a.2) najmniejsza osiągalna wartość współczynnika najgorszego przypadku jest równa $\lceil m/2\rceil$.

Wyniki badań nad algorytmami z grupy (a.3) sugerują, że niektóre z nich mogą mieć współczynnik najgorszego przypadku mniejszy niż $\lceil m/2\rceil$. Szczególnie interesujący wydaje się tutaj algorytm (N). Poniżej podano aktualne częściowe wyniki badań nad tym algorytmem oraz nad pewnymi jego mutacjami.

Twierdzenie 5. Dla algorytmu (N) zachodzi

$$C^N/C^* \leq (m+1)/2.$$

Istnieje przykład, z $C^N/C^* \geq 1/4+\sqrt{m/2+1/16}$, $m=k(k+1)/2$, $k=1,2,\ldots$.■

W tabeli 4 podano przykład dla algorytmu (N) ograniczając się do przypadku m=6 maszyn. Przykład ten składa się z ośmiu podzbiorów identycznych zadań o

podzbiór	1	2	3	4	5	6	7	8
m = 1								2
2					1/2	1		
3					1/2		1	
4	1/8	1/2						
5	1/8		1/2					
6	1/8			1/2				
n_j	8	2	2	2	2	1	1	1

Tab. 4. Przykład dla algorytmu (N): n_j jest licznością j-tego podzbioru; pole puste oznacza zerowy czas wykonywania zadania na maszynie.

licznościach 8,2,2,2,2,1,1,1 odpowiednio. Przygotowana w fazie wstępnej algorytmu lista ma postać $(8)_1(7)_1(6)_1(5)_2(4)_2(3)_2(2)_2(1)_8$. Dla ustalenia uwagi przyjmijmy, że w k-tym kroku fazy zasadniczej, przy równych wartościach C_{max} zadanie jest wstawiane na pierwszą możliwą pozycję. Zatem algorytm (N) dostarczy $\pi^N=(1)_8(2)_2(3)_2(4)_2(5)_2(6)_1(7)_1(8)_1$ oraz $C^N=4$. Jak łatwo sprawdzić, sekwencja $\pi^*=(4)_2(3)_1(1)_8(3)_1(2)_2(7)_1(5)_2(6)_1(8)_1$ jest optymalna oraz $C^*=2$ i stąd $C^N/C^*=2$. Możliwe jest skonstruowanie przykładów dla dowolnego m (niekoniecznie spełniającego warunek $m=k(k+1)/2$, $k=1,2,\ldots$), dla których wartość C^N/C^* jest w przybliżeniu równa $\sqrt{m/2}$. Do tej pory nie udało się jednak skonstruować przykładu o większej wartości C^N/C^*. Wszystko wskazuje na to, iż współczynnik najgorszego przypadku algorytmu (N) jest rzędu $\sqrt{m/2}$. Dość istotna w algorytmie (N) jest faza wstępna, bowiem dla algorytmu N' (jest to algorytm N z opuszczoną fazą wstępną) można podać przykład z $C^{N'}/C^*=\lceil m/2\rceil$. Niestety, podobny przykład można też podać dla algorytmu (N1) (algorytm ten ma n razy mniejszą złożoność obliczeniową niż algorytm (N)). Z kolei dla algorytmu (N1') (algorytm (N1) bez fazy wstępnej) prawdziwe jest następujące twierdzenie, które podajemy bez dowodu.

Twierdzenie 6. Dla algorytmu (N1') zachodzi

$$C^{N1'}/C^* \le (m+1)/2$$

i ograniczenie to jest osiągalne. ∎

Otrzymane wyniki analizy wskazują, że z punktu widzenia analizy najgorszego przypadku najlepszy jest algorytm (N) oraz dobre są algorytmy (C), (C-$\lfloor m/2\rfloor$), (C-$\lceil m/2\rceil$), (R), i priorytetowy z wagami określonymi w Tw. 4. Interesujący jest fakt, że niektóre metody szukań zstępujących (D1, D2) są przy tym wyraźnie gorsze. Problem konstrukcji algorytmu ze współczynnikiem najgorszego przypadku niezależnym od m pozostaje w dalszym ciągu otwarty.

Praca była częściowo finansowana przez RP.I.02 "Teoria sterowania i optymalizacji układów dynamicznych i procesów dyskretnych".

LITERATURA

[1] Barany I.: A vector sum theorem and its application to improving flow-shop guarantees, Math. Oper. Res. 6, 1981, 445-452.

[2] Campbell H.G., Dudek R.A., Smith M.L.: A heuristic algorithm for the n ob m machine sequencing problem, Mgmt Sci. 16, 1970, B630-637.

[3] Dannenbring D.G.: An evaluation of flow-shop sequencing heuristics, Mgmt Sci. 23, 1977, 1174-1182.

[4] Gonzales T., Sahni S.: Flow shop and job shop schedules: complexity and approximation, Oper. Res. 26, 1978, 36-52.

[5] Grabowski J., Skubalska E., Smutnicki C.: On flow shop scheduling with release and due dates to minimize maximum lateness, J. Oper. Res. Soc. 34, 1983, 615-620.

[6] Johnson S.M.: Optimal two- and tree-stage production schedules with setup time included, Naval Res. Log. Quart. 1, 1954, 61-68.

[7] Krone M.J., Steiglitz K.: Heuristic programming solution of a flow--shop problem, Operations Research 22, 1974, 629-638.

[8] Lawler E.J., Lenstra J.K., Rinnoy Kan A.G.H., Shmoys D.B.: Sequencing and Scheduling: Algorithms and complexity", Report BS-R8909, Department of Operations Research, Statistics and System Theory, 1989.

[9] Nawaz M., Enscore Jr. E.E., Ham I.: A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem, OMEGA Int. J. of Mgmt Sci. 11, 1983, 91-95.

[10] Nowicki E., Smutnicki C.: Worst-case analysis of an approximation algorithm for flow-shop scheduling, Operations Research Letters 8, 1989, 171-177.

[11] Nowicki E., Smutnicki C.: Worst-case analysis of Dannenbring's algorithm for flow-shop scheduling, Operations Research Letters, 1989, (being published).

[12] Nowicki E., Smutnicki C.: New results in the worst-case analysis for flow-shop scheduling, Mathematics of Operations Research , 1990, (being published).

[13] Osman I.H., Potts C.N.: Simulating Annealing for Permutation Flow-Shop Scheduling, Preprint OR17, Faculty of Mathematical Studies, University of Southampton, 1989.

[14] Palmer D.S.: Sequencing Jobs Through a Multi-Stage Process in the Minimum Total Time - A Quick Method of Obtaining Near Optimum, Operations Research Quart. 16, 1965, 101-107.

[15] Rock H., Schmidt G.: Machine Aggregation Heuristics in Shop Scheduling, Methods of Operations Research 45, 1982, 303-314.

[16] Sietiaputra W.: A survey of flow-shop permutation scheduling techniques and an evaluation of heuristic solution method, Master's Thesis, Pensylvania State University, 1980.

[17] Turner S., Booth D.: Comparison of heuristics for flow shop sequencing, OMEGA Int. J. of Mgmt Sci. 15, 1987, 75-78.

[18] Wala K.: Metody lokalnej optymalizacji z zagadnieniu harmonogramowania, Zeszyty Naukowe Pol. Śląs., Seria: Automatyka 95, 1988, 179-190.

Recenzent: Doc.dr h.inż.E.Toczyłowski

Wpłynęło do Redakcji do 1990-04-30.

WORST-CASE ANALYSIS OF APPROXIMATION ALGORITHMS
FOR FLOW-SHOP PROBLEM

S u m m a r y

The paper deals with the permutation flow-shop problem. Some approximation algorithms are formulated. Recent results of the worst-case analysis are presented.

АНАЛИЗ НАИХУДШЕГО СЛУЧАЯ АППРОКСИМАЦИОННЫХ АЛГОРИТМОВ ДЛЯ ПОТОЧНОЙ ПРОБЛЕМЫ

Р е з ю м е

В работе рассмотрена поточная проблема упорядочнения задач на m-машинах. Для этой проблемы сформулированы аппроксимационные алгоритмы и проведен анализ наихудшего случая.