ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1988
Seria:AUTOMATYKA z.94 Nr kol.970

Eugeniusz Nowicki
Czesław Smutnicki

Politechnika Wrocławska
Instytut Cybernetyki Technicznej

ALGORYTMY PRZYBLIŻONE ROZWIĄZYWANIA ZAGADNIENIA KOLEJNOŚCIOWEGO POSTACI $1||\Sigma f_i$ */

Streszczenie. W pracy rozważany jest jednomaszynowy problem szeregowania z kryterium w postaci sumy kosztów. Zakłada się, że koszt wykonania zadania jest niemalejącą funkcją jego terminu zakończenia. Do rozwiązania tego problemu proponowane są cztery algorytmy aproksymacyjne oraz przeprowadzana jest dla nich analiza eksperymentalna.

1.Wstęp

W pracy rozważa się jednomaszynowy problem szeregowania z kryterium w postaci sumy kosztów wykonania zadań. Przyjmujemy, że koszt wykonania zadania zależy wyłącznie od terminu jego zakończenia i jest funkcją niemalejącą. Problem ten notujemy, za pracą [3], jako $1||\Sigma f_i$. W ogólnym przypadku jest on silnie NP-trudny, [6]. W celu jego rozwiązania proponowane są w literaturze dwa podejścia. Pierwsze polega na zastosowaniu metody programowania dynamicznego (np. [5]), zaś drugie na zastosowaniu metody podziału i ograniczeń (np. [1], [9]). Oba podane podejścia pozwalają na rozwiązanie w "rozsądnym" czasie przykładów o rozmiarze $n \leq 50$. Dlatego też wydaje się, że dla zagadnień o większych rozmiarach jedynym sensownym podejściem jest zastosowanie algorytmów aproksymacyjnych (przybliżonych). Aktualny stan badań dotyczących zagadnień jednomaszynowych, opracowany na podstawie 204 pozycji literaturowych, przedstawiono w pracy [4]. Z pracy tej wynika, że nie są znane żadne algorytmy aproksymacyjne dla omawianego problemu z ogólną postacią funkcji kosztu.

W prezentowanej pracy formułujemy cztery algorytmy aproksymacyjne rozwiązujące problem $1||\Sigma f_i$. Następnie przedstawiamy dolne ograniczenie minimalnej wartości funkcji celu oparte na zagadnieniu przydziału pracy (AP). Dla sformułowanych algorytmów przeprowadzamy szczegółową analizę eksperymentalną.

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

2. Sformułowanie problemu

Zagadnienie kolejnościowe postaci $1||\Sigma f_j$ formułuje się następująco. Dan jest zbiór n niezależnych zadań $J = \{1,2,\ldots,n\}$, które należy wykonać na jednej maszynie. Dla każdego zadania określony jest czas wykonywania p_j oraz niemalejąca funkcja $f_j: R^+ \cup \{0\} \to R$, $j \in J$. Funkcja f_j reprezentuje koszt związany z wykonaniem zadania $j \in J$. Mianowicie, jeżeli wykonywanie zadania j zostało zakończone w chwili C_j, to $f_j(C_j)$ jest kosztem wykonania tego zadania. Przyjmuje się, że w każdej chwili czasowej maszyna może realizować co najwyżej jedno zadanie oraz realizacji zadania nie można przerywać. Problem polega na znalezieniu uszeregowania (reprezentowanego przez zestaw terminów zakończenia C_j, $j \in J$), które minimalizuje sumaryczny koszt wykonania zadań. Łatwo pokazać, iż w zbiorze uszeregowań optymalnych istnieje uszeregowanie, w którym nie występują przerwy pomiędzy kolejno wykonywanymi zadaniami. Zatem powyższy problem formułuje się następująco.

Znaleźć kolejność wykonywania zadań $\pi^* \in \Pi$, dla której

$$F(\pi^*) = \min_{\pi \in \Pi} F(\pi) , \qquad (1)$$

gdzie

$$F(\pi) = \sum_{j=1}^{n} f_{\pi(j)}\Big(\sum_{i=1}^{j} p_{\pi(i)} \Big)$$

oraz $\pi = (\pi(1),\pi(2),\ldots,\pi(n))$ jest permutacją elementów zbioru J, zaś Π jest zbiorem wszystkich takich permutacji.

Jak już wspominaliśmy we wstępie, problem (1) jest silnie NP-trudny. W pracach [2] oraz [7] podano odpowiednio algorytmy aproksymacyjne dla tego problemu przy założeniu, że funkcje kosztu są postaci: $f_j(t) = \max(0,t-d_j)$ $j \in J$ i $f_j(t) = w_j \max(0,t-d_j)$, $j \in J$. W następnym rozdziale przedstawimy algorytmy aproksymacyjne dla przypadku dowolnych funkcji kosztu $f_j(t)$.

3. Algorytmy aproksymacyjne

Ogólna idea przedstawianych algorytmów aproksymacyjnych polega na przeglądaniu tylko pewnego podzbioru zbioru Π. Podzbiór ten jest konstruowany w oparciu o daną początkową permutację oraz postać funkcji kosztu. Jako ta-

wiązanie algorytmy przyjmują permutację o najmniejszej wartości funkcji celu z tego podzbioru. Przed ich prezentacją zdefiniujemy kilka niezbędnych wielkości. Dla każdej permutacji $\pi \in \Pi$ oraz zadania $j \in J$ oznaczmy przez $O(\pi,j)$ podzbiór Π taki, że

$$O(\pi,j) = \{(j,\pi(1),\ldots,\pi(k-1),\pi(k+1),\ldots,\pi(n)), (\pi(1),j,\pi(2),\ldots,\pi(k-1), \pi(k+1),\ldots,\pi(n)),\ldots,(\pi(1),\ldots,\pi(k-1),\pi(k+1),\ldots,\pi(n),j)\}, \quad (2)$$

gdzie

$$k : \pi(k)=j.$$

Zbiór $O(\pi,j)$ zawiera n permutacji łącznie z π. Z definicji (2) wynika, że otrzymujemy go przez kolejne ustawianie zadania j na pozycjach $1,2,\ldots,n$ w permutacji π. Przykładowo, dla $n = 4$ oraz $\pi = (4,1,3,2)$ i $j = 1$ zbiór ten zawiera cztery permutacje $(1,4,3,2)$, $(4,1,3,2)$, $(4,3,1,2)$, $(4,3,2,1)$.

Wprowadzimy dalej funkcję $H:\Pi \to \Pi$. Wielkość $H(\pi)$ jest wyznaczana w następujący sposób:

(i) Podstaw $F^H := \infty$, $\pi_0 := \pi$.

(ii) Dla każdego $i = 1,2,\ldots,n$ wyznacz permutację $\pi_i \in O(\pi_{i-1},\pi(i))$ spełniającą

$$F(\pi_i) = \min_{\rho \in O(\pi_{i-1},\pi(i))} F(\rho)$$

oraz jeżeli $F(\pi_i) < F^H$, to podstaw $H(\pi) := \pi_i$, $F^H := F(\pi_i)$.

Zauważmy, że po zakończeniu wyznaczania wartości $H(\pi)$ zachodzi $F^H = F(H(\pi))$. W trakcie jej wyznaczania przeglądamy zbiór $O(\pi,\pi(1))$ wybierając najlepszą ze względu na wartość funkcji celu permutację π_1. Następnie przeglądamy zbiór $O(\pi_1,\pi(2))$ wybierając permutację π_2, itd. W ostatnim (n-tym) kroku przeglądamy zbiór $O(\pi_{n-1},\pi(n))$ wybierając permutację π_n. Łącznie we wszystkich krokach przeglądamy zbiór $\bigcup_{i=1}^{n} O(\pi_{i-1},\pi(i))$ zawierający nie więcej niż n^2 permutacji. W konsekwencji wyznaczona permutacja $H(\pi)$ ma najmniejszą wartość funkcji celu na tym zbiorze.

Realizując ogólną ideę przedstawioną na początku rozdziału proponujemy następujący algorytm.

Algorytm H1

Krok 1. Wyznacz permutację $\pi \in \Pi$, dla której $p_{\pi(1)} \le p_{\pi(2)} \le \ldots \le p_{\pi(n)}$.

Krok 2. Wylicz $H(\pi)$ oraz podstaw $\pi^H := H(\pi)$.

Algorytm H1 wyznacza permutację π^H, którą traktujemy jako przybliżone rozwiązanie problemu (1). Złożoność obliczeniowa kroku 1 jest rzędu $O(n\log n)$, zaś krok 2 ma złożoność $O(n^3)$, jednakże może on być implementowany w formie procedury o złożoności $O(n^2)$. Stąd całkowita złożoność algorytmu jest rzędu $O(n^2)$.

Z analizy Algorytmu H1 wynika, że algorytm ten można prosto zmodyfikować. Modyfikacja ta polega na wielokrotnym wykonaniu kroku 2. Otrzymamy wtedy następujący algorytm.

Algorytm H2

Krok 1. Wyznacz permutację $\pi \in \Pi$, dla której $p_{\pi(1)} \le p_{\pi(2)} \le \dots \le p_{\pi(n)}$.
Podstaw $i := 1$, $\pi^{H2} := \pi$.

Krok 2. Wylicz $H(\pi)$ oraz podstaw $\pi^H := H(\pi)$.
Jeżeli $F(\pi^H) < F(\pi)$, to podstaw $\pi^{H2} := \pi^H$.

Krok 3. Jeżeli $F(\pi) \le F(\pi^H)$ lub $i = n$, to stop.
Podstaw $\pi := \pi^H$, $i := i+1$ oraz idź do Kroku 2.

Podobnie jak dla Algorytmu H1, Algorytm H2 wyznacza permutację π^{H2}, którą traktujemy jako przybliżone rozwiązanie problemu (1). Złożoność obliczeniowa tego algorytmu jest rzędu $O(n^2k)$, gdzie k - liczba realizacji kroku 3 ($k \le n$).

3. Dolne ograniczenie

Dla problemu (1) dolne ograniczenie LB wartości funkcji celu będziemy wyznaczać poprzez rozwiązanie odpowiednio sformułowanego zagadnienia przydziału pracy (AP). Metoda ta była z powodzeniem stosowana m.in. w pracach [1], [9]. Poniżej przedstawimy jej opis.

Niech σ będzie permutacją elementów zbioru J taką, że $p_{\sigma(1)} \le p_{\sigma(2)} \le \dots \le p_{\sigma(n)}$. Oznaczmy przez s_{ij} sumę czasów wykonywania $i-1$ zadań, wybranych kolejno z permutacji σ, poczynając od zadania $\sigma(1)$ i nie uwzględniając zadania j, $i \in J$, $j \in J$. Jako dolne ograniczenie przyjmujemy

$$LB = \min_{\pi \in \Pi} \sum_{i=1}^{n} a_{i\pi(i)} \qquad (5)$$

gdzie wielkości a_{ij} $i,j \in J$ są określone następująco

$$a_{ij} = f_j(s_{ij} + p_j), \quad i,j \in J. \qquad (6)$$

Współczynniki a_{ij} są równe minimalnej wartości kosztu związanej z wykonywaniem zadania j-tego jako i-tego w kolejności, i,j ∈ J.

W celu wykazania, że wielkość określona wzorem (3) jest dolnym ograniczeniem, zauważmy, iż dla każdego π ∈ Π oraz i ∈ J zachodzi

$$\sum_{k=1}^{i-1} p_{\pi(k)} \geq s_{i\pi(i)} .$$

Stąd, z definicji F(π), z (4) oraz z faktu, że funkcje f_j są niemalejące, otrzymujemy

$$F(\pi) = \sum_{i=1}^{n} f_{\pi(i)}\Big(\sum_{j=1}^{i-1} p_{\pi(j)} + p_{\pi(i)}\Big) \geq$$

$$\geq \sum_{i=1}^{n} f_{\pi(i)}\big(s_{i\pi(i)} + p_{\pi(i)}\big) = \sum_{i=1}^{n} a_{i\pi(i)} .$$

Ostatecznie dostajemy następującą nierówność

$$\min_{\pi\in\Pi} F(\pi) \geq \min_{\pi\in\Pi} \sum_{i=1}^{n} a_{i\pi(i)} = LB$$

co potwierdza, że wielkość LB jest dolnym ograniczeniem dla problemu (1).

Jak już wspominaliśmy, zagadnienie kolejnościowe definiujące LB znane jest w literaturze jako problem przydziału pracy (AP) z macierzą kosztów określoną przez (4). Do wyznaczania współczynników a_{ij}, i,j ∈ J proponujemy zastosowanie następującego algorytmu.

Algorytm A

Krok 1. Wyznacz permutację σ elementów zbioru J taką, że $p_{\sigma(1)} \leq p_{\sigma(2)} \leq \ldots \leq p_{\sigma(n)}$.

Krok 2. Dla każdego zadania j ∈ J wyznacz pozycję μ_j, na której zadanie to występuje w permutacji σ. W tym celu wykonaj podstawienie $\mu_{\sigma(j)} := j$, j=1,2,...n.

Krok 3. Podstaw s:= 0 oraz i:= 1.

Krok 4. Dla każdego j = 1,2,...,n wykonaj:
jeżeli $\mu_j \geq i$, to podstaw $a_{ij} := f_j(s+p_j)$,
jeżeli $\mu_j < i$, to podstaw $a_{ij} := f_j(s+p_{\sigma(i)})$.

Krok 5. Jeżeli i=n, to stop.

W przeciwnym przypadku podstaw $s := s+p_{\sigma(i)}$, $i := i+1$ oraz przejdź do kroku 4.

Powyższy algorytm został oparty na spostrzeżeniu, że zgodnie z definicją s_{ij} , $i,j \in J$ zachodzi

$$s_{ij} = \begin{cases} \sum_{k=1}^{i-1} p_{\sigma(k)} & \text{jeśli } \mu_j \geq i, \\ \sum_{k=1}^{i-1} p_{\sigma(k)} - p_j + p_{\sigma(i)} & \text{jeśli } \mu_j < i. \end{cases}$$

W konsekwencji drogą prostych przekształceń łatwo otrzymać zależność rekurencyjną wykorzystywaną w Algorytmie A. Algorytm ten, wyznaczający n^2 liczb s_{ij} , ma złożoność obliczeniową rzędu $O(n^2)$. Z kolei złożoność obliczeniowa zagadnienia (AP) wynosi $O(n^{2.5})$. Ostatecznie złożoność obliczeniowa metody wyznaczania dolnego ograniczenia jest rzędu $O(n^{2.5})$.

4. Wyniki obliczeń

Algorytmy H1 i H2 zostały zaimplementowane w języku FORTRAN i przetestowane na minikomputerze IBM PC/XT. Wyliczenie wartości $H(\pi)$ występujące w Kroku 2 tych algorytmów zostało zaimplementowane w formie procedury o złożoności obliczeniowej rzędu $O(n^2)$. W procedurze korzysta się z faktu, że zamiana dwóch sąsiednich zadań (występująca przy badaniu permutacji ze zbioru $O(\pi_{i-1},\pi(i))$) nie zmienia terminów zakończenia wykonywania wszystkich pozostałych zadań. W konsekwencji do wyliczania wartości $F(\rho)$, dla kolejnych $\rho \in O(\pi_{i-1},\pi(i))$, można zastosować odpowiednią formułę rekurencyjną, zamiast algorytmu o złożoności $O(n)$. Uzyskujemy w ten sposób złożoność całej procedury $O(n^2)$ zamiast oczekiwanej $O(n^3)$.

Do testowania przyjęto funkcję kosztu wykonania zadania w postaci $f_j(t) = w_j \max(0,t-d_j)$, $j \in J$. Wykorzystując jej specyfikę zaproponowano dwa dodatkowe warianty H2.1 oraz H2.2 Algorytmu H2. W Algorytmie H2.1 permutacja $\pi \in \Pi$ z Kroku 1 jest wyznaczana wg niemalejących wartości w_j . W Algorytmie H2.2 permutacja $\pi \in \Pi$ jest wyznaczana według niemalejących wartości $w_j/p_j + w_j/d_j$.

Dla każdego $n = 20, 50, 80, 100$ oraz $t = 0.2, 0.4, 0.6, 0.8$ wylosowano 10 przykładów problemu (1). Dane do przykładów w_j, p_j, d_j, jej wylosowano jako liczby całkowite odpowiednio przez:

(a) generator o rozkładzie równomiernym na przedziale [4,15],

(b) generator o rozkładzie równomiernym ze średnią równą 100n(1-t) i wariancją 4n,

(c) generator o rozkładzie normalnym ze średnią 100 i wariancją 25.

Podany sposób losowania przyjęto za pracą [8]. Dla każdego wylosowanego przykładu wyznaczano LB wg (3), π^H, $F(\pi^H)$ oraz błąd

$$\eta^H = \begin{cases} \dfrac{F(\pi^H)-LB}{F(\pi^H)} * 100\ \%, & \text{gdy } F(\pi^H) \neq 0, \\ 0\ \%, & \text{gdy } F(\pi^H) = 0, \end{cases}$$

$H \in (H1, H2, H2.1, H2.2)$. Powyższa definicja błędu została zaczerpnięta z pracy [2] i reprezentuje wielkość określającą "potencjalną" możliwość zmniejszenia wartości funkcji celu dla otrzymanego uszeregowania, $0 \leq \eta^H \leq 100$ ($\eta^H = 0$ dla uszeregowania optymalnego). Wyniki obliczeń przedstawiono w Tabeli 1.

Analiza wyników z Tabeli 1 prowadzi do następujących spostrzeżeń:

(i) średnia wartość błędu η^H waha się w granicach 3.38% ÷ 7.85%, zaś maksymalna wartość błędu η^H w granicach 5.41% ÷ 13.51%,

(ii) średnia wartość błędu η^H maleje ze wzrostem t dla ustalonego n; nie obserwuje się natomiast wpływu zmiany n przy ustalonym t na tę średnią,

(iii) maksymalna wartość błędu η^H maleje ze wzrostem n przy ustalonym t,

(iv) we wszystkich badanych przykładach Algorytm H1 dawał tylko niewiele gorsze rezultaty (w sensie wartości η^H) od pozostałych algorytmów,

(v) tylko dla n = 20 Algorytm H1 generował czasami tak dobre rozwiązania (w sensie wartości 1) jak inne algorytmy; dla pozostałych n wpływ "zapętlenia" kroku 2 Algorytmu H1 jest znaczący dla wartości 1,

(vi) różnice, w sensie wartości η^H oraz 1, pomiędzy Algorytmami H2, H2.1, H2.2 są niewielkie,

(vii) Algorytmy H2, H2.1, H2.2 realizowały co najwyżej 5 razy Krok 2 (wynika to z podanych czasów obliczeń), mimo iż teoretycznie mogły go wykonywać n razy.

Uwzględniając powyższe uwagi, wyciągamy następujące wnioski:

- wszystkie zaproponowane Algorytmy są "zadowalająco dobre" dla testowanej klasy przykładów losowych,
- wpływ permutacji początkowej jest nieznaczny; wynika to ze spostrzeżeń (iv) i (vi).

Tabela 1.

Wyniki eksperymentów numerycznych

n	t	H1 av	H1 max	H2 av	H2 max	H2.1 av	H2.1 max	H2.2 av	H2.2 max
20	.2	7.85	13.51	7.15	11.55	7.30	11.55	7.15	11.55
	.4	5.94	11.48	5.33	11.45	5.30	11.54	5.24	11.45
	.6	5.13	10.34	4.57	9.72	4.57	9.70	4.56	9.72
	.8	3.55	6.42	3.40	6.42	3.38	6.34	3.39	6.34
50	.2	7.32	11.22	6.81	10.50	6.77	10.50	6.74	10.50
	.4	7.58	9.29	7.37	9.26	7.39	9.26	7.38	9.26
	.6	7.16	8.88	6.52	7.79	6.52	7.81	6.53	7.80
	.8	4.63	6.31	4.03	5.41	4.02	5.40	4.02	5.41
80	.2	6.98	10.56	6.67	10.18	6.69	10.33	6.66	10.24
	.4	7.61	9.60	7.49	9.46	7.50	9.46	7.51	9.46
	.6	5.89	7.79	5.52	7.73	5.52	7.73	5.52	7.73
	.8	5.26	6.68	4.77	6.27	4.77	6.27	4.76	6.27
100	.2	7.26	8.59	6.95	8.23	6.88	8.20	6.89	8.23
	.4	7.36	8.76	7.27	8.71	7.27	8.74	7.26	8.71
	.6	6.17	7.01	5.69	6.73	5.69	6.73	5.69	6.73
	.8	5.98	6.81	5.52	6.12	5.52	6.12	5.49	6.12

n	t	l	cpu	l	cpu	l	cpu	l	cpu
20	.2	7	1.0	10	2.3	9	1.4	10	1.6
	.4	1	1.0	8	2.3	6	1.7	9	1.7
	.6	1	0.9	8	2.3	9	2.1	9	1.8
	.8	3	1.0	8	2.3	8	1.9	9	2.1
50	.2	0	5.8	7	15.5	8	12.6	9	11.8
	.4	0	5.6	9	17.5	7	15.6	8	13.5
	.6	0	5.7	8	23.5	8	16.0	7	14.1
	.8	0	5.6	6	21.2	7	15.4	6	15.2
80	.2	0	14.3	4	48.0	4	29.2	4	44.1
	.4	0	14.4	7	46.7	6	37.0	4	30.6
	.6	0	14.1	6	59.9	2	41.4	7	53.1
	.8	0	14.5	6	69.5	5	49.6	8	46.7
100	.2	0	22.1	4	70.2	8	72.6	5	60.4
	.4	0	22.2	5	83.4	5	62.3	9	68.5
	.6	0	22.1	5	102.2	6	74.9	6	79.1
	.8	0	22.2	5	105.3	5	80.1	6	85.2

av, max - średnia arytmetyczna oraz maksymalna z 10 wyznaczonych wartości błędu η^H; w %,

l - liczba przykładów, dla których dany algorytm znalazł rozwiązanie o wartości funkcji celu równej

$$\min (F(\eta^{H1}), F(\eta^{H2}), F(\eta^{H2.1}), F(\eta^{H2.2})),$$

cpu - średni czas obliczeń na IBM PC/XT; w sec.

- fakt, że liczba powtórzeń Kroku 2 w Algorytmach H2, H2.1, H2.2 jest niewielka oraz różnica w ocenie η^H jest względnie mała, sugeruje, że w wielu praktycznych sytuacjach wystarczające jest użycie Algorytmu H1.

5. Uwagi końcowe

Przedstawione w pracy algorytmy aproksymacyjne mogą być łatwo rozszerzone na jednomaszynowe problemy z dodatkowymi ograniczeniami kolejnościowymi lub niezerowymi terminami gotowości zadań do wykonywania, tzn. problemy postaci $1|prec|\Sigma f_j$ oraz $1|r_j|\Sigma f_j$. W przypadku pierwszego problemu początkowa permutacja $\pi \in \Pi$ powinna być dodatkowo dopuszczalna, tzn. zgodna z występującymi ograniczeniami kolejnościowymi. Ponadto w zbiorze $O(\pi_{i-1}, \pi(i))$ badane są tylko permutacje dopuszczalne. Kontrola dopuszczalności permutacji nie zwiększa złożoności algorytmów. Z kolei w przypadku drugiego z wymienionych problemów zmianie ulega sposób wyliczania wartości $F(\pi)$ oraz Krok 2 algorytmów nie może być implementowany w formie procedury o złożoności $O(n^2)$. Zatem złożoność Algorytmu H1 jest wtedy rzędu $O(n^3)$, a Algorytmu H2 $O(n^3k)$.

Algorytmy H1 oraz H2 mogą być też stosunkowo łatwo zaadaptowane do rozwiązywania problemów wielomaszynowych, takich jak np. problemy przepływowe $F||\Sigma f_j$ czy też problemy gniazdowe $J||\Sigma f_j$.

LITERATURA

[1] Adrabiński A., Grabowski J., Wodecki M.: Algorytm rozwiązania zagadnienia kolejnościowego postaci $n|1|\Sigma w_i T_i$, Raport Instytutu Cybernetyki Technicznej Politechniki Wrocławskiej, Seria Preprinty nr 79/87, Wydawnictwo Politechniki Wrocławskiej, Wrocław 1987.

[2] Baker K.R., Martin J.B.: An experimental comparison of solution algorithms for single machine tardiness problem, Naval Res. Logist. Quart. 21, 1974, 187-199.

[3] Graham R.L. et al.: Optimization and approximation in deterministic sequencing and scheduling: a survey, Ann. Discrete Math. 5, 1979, 287-326.

[4] Gupta S.K., Kyparisis J.: Single machine scheduling research, OMEGA Int. J. of Mgmt Sci. 15(3), 1987, 207-227.

[5] Lawler E.L.: On scheduling problems with deferral costs, Mgmt Sci. 11, 1964, 280-288.

[6] Lenstra J.K., Rinnooy Kan A.H.G., Brucker P.: Complexity of machine scheduling problems, Ann. Discrete Math. 1, 1977, 343-362.

[7] Montagne E.R. Jr: Sequencing with time delay costs, Arizona State Uni-

versity Industrial Engineering Research Bulletin 5, 1989, 20-31.
[8] Rinnooy Kan A.H.G.: Machine Scheduling Problems: Classification, Complexity and Computations, Nijhoff The Hague 1976.
[9] Rinnooy Kan A.H.G., Lageweg B.J., Lenstra J.K.: Minimizing total costs in one-machine scheduling, Oper. Res. 23, 1975, 908-927.

Recenzent:Doc.dr hab.inż.J.Klemka
Wpłynęło do Redakcji do 1988-04-30.

ПРИБЛИЖЕННЫЕ АЛГОРИТМЫ ДЛЯ ЗАДАЧ ЧЕРЕДОВАНИЯ

Р е з ю м е

В статье рассматривается одномашинная задача чередования с критерием суммы стоимости выполнения задач.Предполагается , что стоимость выполнения задачи является неубывающей функцией её времени выполнения. Формулируются четыре аппроксимационных алгоритма и представляются результаты вычислительных экспериментов, проведённых для случайно избранных примеров.

APPROXIMATION ALGORITHMS OF $1||\Sigma f_i$ SEQUENCING PROBLEM

S u m m a r y

The paper deals with one-machine scheduling problem with total cost criterion. It is assumed that the job processing cost is given by a nondecreasing function and depends only on job completion time. Four approximation algorithms are proposed for this problem. The experimental analysis of the algorithms is also carried out.