ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1984
Seria: AUTOMATYKA, z. 74 Nr kol. 810

Jacek Błażewicz[^], Mieczysław Drabowski[*]
Jan Węglarz

[^]Politechnika Poznańska
[*]Zakłady MERA, Kraków

O NIEKTÓRYCH PROBLEMACH SZEREGOWANIA ZADAŃ WIELOPROCESOROWYCH

Streszczenie. W pracy rozpatrzono problem minimalizacji długości uszeregowania przy założeniu, iż pojedyncze zadanie do swego wykonania potrzebuje w każdej chwili czasu dwóch lub więcej procesorów. Przeanalizowano wpływ podzielności zadań na złożoność problemu.

1. Wstęp

W dotychczasowych pracach dotyczących problemów szeregowania zadań na procesorach maszyn cyfrowych zakładano, że każde zadanie może być w każdej chwili wykonywane przez co najwyżej jeden procesor [2,5,8]. Jednakże rozwój nowych systemów mikroprocesorowych sprawił, że założenie to nie zawsze jest spełnione. W szczególności, w mikroprocesorowych systemach diagnostycznych, wykonywanie zadania oznacza generowanie sygnałów pobudzających obiekt będący przedmiotem testowania i obserwowanie sygnałów wyjściowych tego obiektu. W celu wykrycia niektórych błędów obie te czynności muszą być wykonywane równocześnie. Inny przykład mogą stanowić samotestujące systemy wielomikroprocesorowe, w których pewne procesory są wykorzystywane do testowania innych. W obu powyższych przypadkach należy zatem założyć, iż pojedyncze zadanie do swego wykonania potrzebuje w każdej chwili dwóch lub więcej procesorów [1,7,10]. W pracy zostaną przeanalizowane przy tym założeniu problemy szeregowania zadań niepodzielnych i podzielnych, dla kryterium długości uszeregowania. Zanim to jednak uczynimy, zdefiniujemy rozpatrywany problem bardziej formalnie.

W ogólności rozpatrywać będziemy zbiór n niezależnych /bez ograniczeń kolejnościowych/ zadań, które należy wykonać, wykorzystując zbiór m identycznych procesorów $P = \{P_1, P_2, \ldots, P_m\}$. Zbiór zadań składa się z k podzbiorów $Z = \{Z_1, Z_2, \ldots, Z_{n_1}\}$, $W^2 = \{W_1^2, W_2^2, \ldots, W_{n_2}^2\}$, . . . , $W^k = \{W_1^k, W_2^k, \ldots, W_{n_k}^k\}$, gdzie $n = n_1 + n_2 + \ldots + n_k$. Każde zadanie Z_i, $i = 1,2,\ldots,n_1$ potrzebuje do swego wykonania dowolnego procesora, a jego czas wykonywania wynosi t_i jednostek. Z drugiej strony, każde zadanie W_i^k potrzebuje do swego wykonywania k procesorów, które wykonają to zadanie pracując jednocześnie w ciągu w_i^k jednostek. Zatem, możemy zadania ze zbioru W^k nazwać zadaniami k-procesorowymi.

Będziemy mówili o szeregowaniu podzielnym, gdy wykonywanie dowolnego zadania ze zbioru $Z \cup W^2 \cup \ldots \cup W^k$ może być przerwane w dowolnej chwili czasu i kontynuowane później bez dodatkowych kosztów /strat czasu/, na-

tomiast o szeregowaniu niepodzielnym - w przeciwnym przypadku. Uszeregowanie nazwiemy wykonalnym jeśli oprócz spełnienia zwykłych warunków [5] w każdej chwili czasu każde zadanie jednoprocesorowe /ze zbioru Z/ jest wykonywane przez 1 procesor a każde zadanie k-procesorowe przez k procesorów. Uszeregowanie wykonalne nazwiemy optymalnym, jeśli jego długość jest minimalna.

W paragrafie 2 przeanalizujemy złożoność problemu szeregowania zadań niepodzielnych, natomiast w paragrafie 3 zbadamy wpływ podzielności na złożoność algorytmów szeregowania zadań jedno- i dwuprocesorowych.

2. Szeregowanie zadań niepodzielnych

W paragrafie tym skoncentrujemy się na analizie niepodzielnego szeregowania zadań o jednostkowych czasach wykonywania. Uczynimy tak dlatego, iż w przypadku zadań o dowolnych czasach wykonywania problemu jest NP-zupełny nawet dla zbiorów zadań składających się wyłącznie z zadań jednoprocesorowych [5]. Zatem najprawdopodobniej nie można rozwiązać takiego problemu za pomocą algorytmów optymalnych o złożoności wielomianowej.

Rozpoczniemy od problemu szeregowania zadań należących jedynie do dwóch zbiorów : zbioru Z oraz zbioru W^k dla dowolnego k. Problem ten można rozwiązać, stosując poniższy algorytm.

Algorytm 1

1. Szereguj najpierw zadania ze zbioru W^k, przydzielając $\lfloor m/k \rfloor$ [1/] tych zadań w każdym przedziale czasu $[0,1], [1,2], \ldots, [l-1,l]$, gdzie
$$l = \left\lceil \frac{n_k}{\lfloor \frac{m}{k} \rfloor} \right\rceil .$$
2. Szereguj następnie zadania ze zbioru Z :
   - przydziel zadania do wolnych procesorów w przedziałach czasu $[0,1], [1,2], \ldots, [l-1,l]$;
   - przydziel pozostałe zadania ze zbioru Z do procesorów w kolejnych przedziałach czasu.

Optymalność powyższego algorytmu jest oczywista a jego złożoność wynosi $O(n)$.

Rozwiążmy teraz problem szeregowania zadań należących do zbiorów $Z, W^2, W^3, \ldots, W^k$, gdzie k jest ustaloną liczbą całkowitą. Podejście jakie możemy zastosować do rozwiązania tego problemu jest podobne do wykorzystanego w przypadku problemu szeregowania zadań niepodzielnych o jednostkowych czasach wykonywania przy ograniczeniach ze strony dodatkowych zasobów [4]. Pomiżej opiszemy to zmodyfikowane podejście.

1/ $\lfloor x \rfloor$ oznacza największą liczbę całkowitą nie większą niż x , a $\lceil x \rceil$ najmniejszą liczbę całkowitą nie mniejszą niż x.

Przez elementarną instancję rozumieć tutaj będziemy zbiór zadań $Z \cup W^2 \cup W^3 \cup \ldots \cup W^k$, dla którego spełniona jest następująca nierówność

$$n_1 + 2n_2 + 3n_3 + \ldots + kn_k \leq m$$

Zauważmy, że zadania tworzące każdą elementarną instancję mogą być wykonane w jednostce czasu. Ponadto, ponieważ k jest ustalone, więc liczba różnych elementarnych instancji K jest także ustalona.

Możemy teraz zdefiniować odpowiadające elementarnym instancjom, elementarne wektory $\bar{b}_1, \bar{b}_2, \ldots, \bar{b}_K$, których i-ta składowa oznacza liczbę i-procesorowych zadań.

Problem nasz sprowadza się zatem do znalezienia dekompozycji danej instancji /zbioru zadań/ na minimalną liczbę instancji elementarnych. Jest oczywiste, że powyższy problem jest równoważny znalezieniu dekompozycji wektora $\bar{n} = n_1, n_2, \ldots, n_k$ na liniową kombinację elementarnych wektorów $\bar{b}_1, \bar{b}_2, \ldots, \bar{b}_k$, dla której suma współczynników jest najmniejsza. Zatem otrzymujemy następujący problem :

Znaleźć całkowitoliczbowe $e_1, e_2, \ldots, e_k$ takie, że

$$\sum_{j=1}^{K} e_j \bar{b}_j = \bar{n},$$

$$e_j \geq 0,$$

oraz

$$\sum_{j=1}^{K} e_j \quad \text{osiąga minimum.}$$

Powyższy problem całkowitoliczbowego programowania liniowego posiada ustaloną liczbę zmiennych. Z pracy [9] wynika, że problem całkowitoliczbowego programowania liniowego z ustaloną liczbą zmiennych K można rozwiązać w czasie ograniczonym przez wielomian zależny od liczby ograniczeń M i log a, gdzie a jest największym współczynnikiem występującym w sformułowaniu tego problemu. Wyrażając to ściślej, złożoność tego problemu jest $O\left(2^{K^2}(KM \log a)^{c^K}\right)$ dla pewnej stałej c. W przypadku szeregowania zadań otrzymujemy końcową złożoność obliczeniową powyższego podejścia $O\left((\log n)^{c^K}\right) < O(n)$.

Rozpatrzmy teraz problem szeregowania różniący się od powyższego tylko tym, że k jest dowolną /a nie ustaloną/ liczbą całkowitą. Przy takim założeniu okazuje się, że problem jest obliczeniowo trudny. Można bowiem wykazać, że decyzyjna wersja tego problemu /oznaczmy ją przez Π_1/ jest silnie NP-zupełna. Uczynimy to w poniższym twierdzeniu.

Twierdzenie 1

Problem Π_1 jest silnie NP-zupełny.

Dowód

Pominiemy, jako dość oczywisty, dowód przynależności problemu Π_1 do klasy NP. W drugiej części dowodu jako znany problem silnie NP-zupeł-

ny przyjmiemy decyzyjny problem trójpodziału [6], zdefiniowany poniżej.

Instancja : Zbiór A składający się z 3q elementów, ograniczenie $B \in Z^{+1/}$, $B/4 < s(a) < B/2$ i $\sum_{a \in A} a = qB$.

Pytanie : Czy istnieje podział zbioru A na q rozłącznych podzbiorów $A_1, A_2, \ldots, A_q$ takich, że dla każdego i, i=1,2,...,q zachodzi $\sum_{a \in A_i} s(a) = B$?

Dla danej instancji problemu trójpodziału konstruuje się odpowiadającą instancję π_1 w następujący sposób.

- Liczba zadań jest równa q.
- Dla każdego elementu $a \in A$ definiuje się zadanie o jednostkowym czasie wykonywania należące do zbioru $W^{s(a)}$.
- Progową długość uszeregowania przyjmuje się równą q.

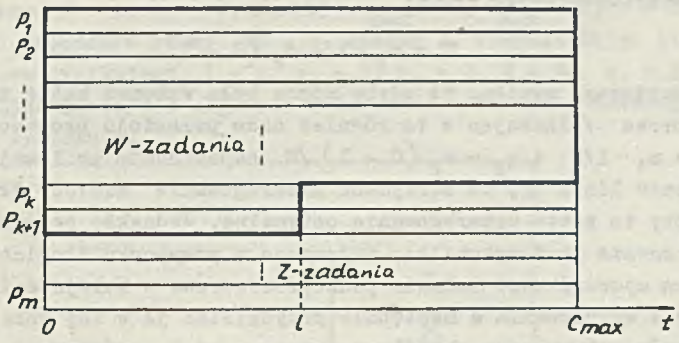
Nietrudno zauważyć, że odpowiedź dla instancji problemu trójpodziału brzmi "tak" wtedy i tylko wtedy, gdy dla instancji problemu π_1 istnieje uszeregowanie o długości nie większej niż q. Ponieważ pozostałe warunki określające transformację pseudowielomianową są także spełnione [6], więc twierdzenie zostało udowodnione. □

Z powyższego twierdzenia a także z poprzedzających je rozważań dotyczących złożoności problemów szeregowania zadań o dowolnych czasach wykonywania wynika, że dużo problemów szeregowania zadań niepodzielnych to problemy NP-zupełne, zatem nierozwiązywalne efektywnie w praktyce. Z tego względu zajmiemy się w następnym rozdziale zagadnieniem szeregowania zadań podzielnych.

3. Szeregowanie zadań podzielnych

Rozważania w tym paragrafie skoncentrujemy na analizie problemu szeregowania zadań należących do dwóch zbiorów : Z oraz W^2. Można wykazać [3], że w tym przypadku pomiędzy uszeregowaniami o minimalnej długości istnieje tzw. A-uszeregowanie, to znaczy takie, w którym najpierw do kolejnych procesorów w przedziale czasu $[0, C_{max}]$ przydzielane są wszystkie zadania dwuprocesorowe a następnie w pozostałej części uszeregowania umieszczane są zadania ze zbioru Z /por. rys. 1/. Korzystając z tego wyniku, poszukiwać będziemy optymalnego A-uszeregowania.

1/ Z^+ oznacza zbiór liczb całkowitych dodatnich.



Rys. 1 Przykład A-uszeregowania

Określmy teraz dolne oszacowanie długości uszeregowania dla rozpatrywanego problemu. Aby to uczynić wprowadzimy poniższe oznaczenia :

$$X = \sum_{i=1}^{n_1} t_i \qquad Y = \sum_{i=1}^{n_2} w_i^2 \qquad T = X + 2Y$$

$$t_{max} = \max \{ t_i : Z_i \in Z \}, \quad w_{max} = \max \{ w_i^2 : w_i^2 \in W^2 \}.$$

Oszacowanie C_{max} ma następującą postać :

$$C_{max} \geq C = \max \{ T/m , Y/\lfloor m/2 \rfloor, t_{max}, w_{max} \} \qquad /1/$$

Jest dość oczywiste, że długość wykonalnego uszeregowania nie może być krótsza niż wartość maksymalna z czterech składników w powyższym wzorze, to znaczy ze średniego zapotrzebowania na obsługę przypadającego na jeden procesor, średniego zapotrzebowania na obsługę zadań dwuprocesorowych przypadającego na dwa procesory, maksymalnego czasu wykonywania zadania dwuprocesorowego i maksymalnego czasu wykonywania zadania dwuprocesorowego.

Korzystając z oszacowania /1/ można teraz podjąć próbę konstrukcji uszeregowania podzielnego o minimalnej długości równej C. Najpierw przydzielone są w przedziale $[0,C]$ zadania ze zbioru W^2, począwszy od procesora P_1. Otrzymane uszeregowanie częściowe przedstawia rysunek 1. Po prawej stronie pionowej linii $t = l$, m_2 procesorów może wykonywać zadania jednoprocesorowe w przedziale $[l,C]$, gdzie

$$m_2 = m - 2\lfloor Y/C \rfloor.$$

Natomiast po lewej stronie tej linii, $m_1 = m_2 - 2$ procesorów może wykonywać zadania jednoprocesorowe w przedziale $[0,l]$. Zatem łączna moc obliczeniowa procesorów, którą można przeznaczyć na wykonywanie zadań

jednoprocesorowych wynosi

$$M = m_1 \; l + m_2 \,(C - l).$$

Z powyższego wynika, iż gdyby można było wykonać każde zadanie jednoprocesorowe /włączając w to również czas przestoju procesora/ w częściach $q_1 = m_1 \cdot l/M$ i $q_2 = m_2 \,(C - l)\,/M$ odpowiednio po lewej i po prawej stronie linii l, to otrzymane uszeregowanie miałoby długość równą C. Byłoby to zatem uszeregowanie optymalne. Jednakże powyższe postępowanie nie zawsze jest wykonalne, zwłaszcza w przypadku długich zadań. Należy zatem uporządkować zadania jednoprocesorowe w kolejności nie rosnących czasów wykonywania a następnie przydzielać je w tej kolejności do procesorów, utrzymując określoną przez q_1 i q_2 równowagę pomiędzy częściami poszczególnych zadań przydzielanymi po obu stronach linii l. Rozważania te podsumowano w poniższym algorytmie.

Algorytm 2

1. Oblicz oszacowanie C zgodnie ze wzorem /1/.
2. Przydziel zadania dwuprocesorowe w dowolnej kolejności do procesorów w przedziale $[0,C]$, przerywając wykonywanie zadania w momencie C i przydzielając pozostałą jego część do następnych dwóch procesorów w momencie 0.
3. Oblicz części q_1 i q_2 na jakie należy podzielić każde zadanie jednoprocesorowe. Uporządkuj te zadania w kolejności nie rosnących czasów wykonywania. Ustaw wskaźnik równowagi pomiędzy częściami zadań przydzielonymi po prawej i lewej stronie linii l, $DT := 0$.
4. Rozpatrz pierwsze nie przydzielone zadanie na liście /powiedzmy Z_i/ i
   a/ Jeśli $DT = 0$, to przydziel część zadania Z_i równą $q_2 t_i$ po prawej stronie linii l. Jeśli jest to niemożliwe przydziel po tej stronie $C - l$ jednostek tego zadania, ustaw $DT := q_2 t_i - (C-l)$ i idź do kroku 5.
   b/ Jeśli $DT > 0$, to przydziel maksymalną możliwą część tego zadania τ po prawej stronie linii l, gdzie

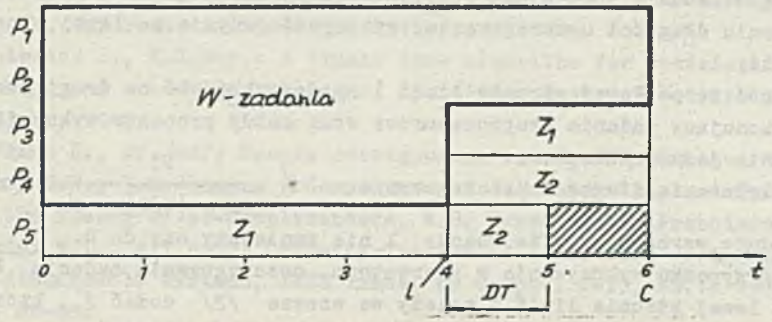
$$\tau := \min \left\{ t_i,\; C - l \,,\; DT + q_2 t_i \right\}$$

   i podstaw $DT := DT - (\tau - q_2 t_i)$.
5. Przydziel pozostałą część tego zadania po lewej stronie uszeregowania i powtórz krok 4 jeśli istnieją jeszcze nie przydzielone zadania.

Jest oczywiste, że jeśli po wykonaniu tego algorytmu $DT = 0$, to skonstruowane uszeregowanie jest wykonalne a co za tym idzie - optymalne. Nie jest to jednakże zawsze możliwe, jak pokazano w poniższym przykładzie.

Rozważmy następujące dane $n = 5$, $n_1 = 2$, $n_2 = 3$, $m = 5$, $\bar{t} = [6,3]$

$\overline{w}^2$= [3,3,4], Zatem X = 9, Y = 10, T = 29, t_{max} = 6, w_{max} = 4, C = 6 a czas przestoju procesora równy mC - T, wynosi 1. Przydzielając zadania dwuprocesorowe uzyskujemy l = 4, m_1 = 1, m_2 = 3, M = 10, q_1 = 2/5, q_2 = 3/5. Wykonując następnie algorytm 2 do końca uzyskujemy uszeregowanie pokazane na rysunku 2.



Rys. 2 Przykład uszeregowania niewykonalnego.

Ponieważ DT = 1 zatem uszeregowanie to jest niewykonalne.

Zachowanie się uszeregowań takie jak przedstawiono powyżej wynika z dwóch faktów : z niesymetrycznych wartości mocy obliczeniowych procerorów po prawej i lewej stronie linii l oraz z istnienia długich zadań, które powinny być wykonywane dłużej niż to możliwe po prawej stronie linii l. Te przyczyny sprawiają, że część ostatniego zadania nie może być wykonana po lewej stronie linii l z powodu braku mocy obliczeniowej procesorów. Część ta nie może być również wykonana po prawej stronie linii l, gdyż inna część tego zadania została już po tej stronie przydzielona. Wynika stąd, że część tego zadania nie może być w ogóle wykonana w uszeregowaniu skonstruowanym przez algorytm 2 ; część ta tworzy zatem tak zwany czas martwy (DT). Aby zatem wykonać wszystkie zadania musimy zwiększyć długość uszeregowania. Nową długość określa poniższe twierdzenie.

Twierdzenie 2

Jeśli w uszeregowaniu skonstruowanym przez algorytm 2 DT > 0, to długość uszeregowania optymalnego wyraża się wzorem

$$C^*_{max} = C + \frac{DT + \mathcal{E}}{m - \lfloor Y/C \rfloor - 1}, \qquad /2/$$

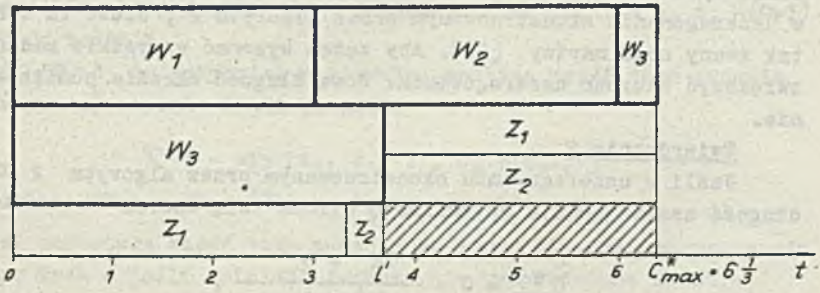
gdzie $\mathcal{E} = \max\{0, 2DT + t_i + C - 2l - \frac{DT}{m - \lfloor Y/C \rfloor - 1}$, jeśli zadanie Z_i powodujące czas martwy jest wykonywane po lewej stronie linii l, a przeciwnym razie $\mathcal{E} = 0$,

Nie będziemy tu przytaczać całego dowodu, który jest dosyć skomplikowany [3]. Podamy jednak główne punkty rozumowania.

1. Czas martwy może wystąpić jedynie po prawej stronie uszeregowania.
2. Czas ten jest spowodowany przez jedno zadanie /powiedzmy Z_1/.
3. Część zadania powodującą powstanie czasu martwego należy, po zwiększeniu długości uszeregowania, wykonywać jedynie po lewej stronie linii 1.
4. Część tę po lewej stronie linii 1 może wykonywać co drugi procesor wykonujący zadania dwuprocesorowe oraz każdy procesor wykonujący zadania jednoprocesorowe.
5. Zwiększenie długości uszeregowania o $\theta = \frac{DT}{m - \lfloor Y/C \rfloor - 1}$ nie spowoduje wzrostu m_1, to znaczy 1 nie zmniejszy się do 0.
6. W przypadku wykonywania w pierwotnym uszeregowaniu zadania Z_1 także po lewej stronie linii 1 należy we wzorze /2/ dodać ε, którego wartość zdefiniowano w tezie twierdzenia 2.

Korzystając z powyższego twierdzenia można łatwo skonstruować uszeregowanie optymalne. W tym celu trzeba wykonać najpierw algorytm 2 i jeśli w wyniku jego wykonania $DT > 0$, to należy zwiększyć długość uszeregowania w sposób podany w twierdzeniu 2 i następnie wykonać jeszcze raz algorytm 2. Optymalność tego podejścia wynika z twierdzenia 2 i z dyskusji poprzedzającej je. Jego złożoność jest $O(n_1 \cdot \log n_1)$, gdyż jego najbardziej złożoną czynnością jest uporządkowanie n_1 zadań jednoprocesorowych.

Możemy teraz powrócić do rozpatrywanego przykładu. Po wykonaniu określonych powyżej czynności uzyskujemy uszeregowanie, które pokazano na rysunku 3 .



Rys. 3 Uszeregowanie optymalne dla rozpatrywanego przykładu.

LITERATURA

[1]. Avizienis A.: Fault tolerance : the survival attribute of digital

systems. Proc. of the IEEE 66, No.10, 1978, pp. 1109-1125

[2]. Baker K.R.: Introduction to Sequencing and Scheduling, J.Wiley and Sons, New York 1974.

[3]. Błażewicz J., M.Drabowski, J.Węglarz : Scheduling independent 2-processor tasks in microprocessor systems to minimize schedule length, Information Processing Letters, w druku.

[4]. Błażewicz J., K.Ecker,: A linear time algorithm for restricted bin packing and scheduling problems, Operations Research Letters 2, No.2 1983, pp.80-83.

[5]. Coffman G., Jr /ed/; Teoria szeregowania zadań, WNT, Warszawa 1982.

[6]. Garey M.R., D.S.Johnson : Computers and Intractability : A Guide to the Theory of NP-Completeneses, W.H. Freeman, San Francisco 1979.

[7]. Hakimi S.L., A.T.Amin: Characterization of connection assignment of diagnosable systems, IEEE Trans. on Comput. C-23, No.1. 1974, pp. 86-88.

[8]. Lawler E.L., J.K.Lenstra, A.G.H. Rinnooy Kan : Recent developments in deterministic sequencing and scheduling : a survey. Report BW 146, Mathematisch Centrum, Amsterdam 1981.

[9]. Lenstra H.W.,Jr.; Integer programming with a fixed number of variables, Report, University of Amsterdam 1981.

[10]. Preperate F,P., G.Metze, R.T.Chien : On the connection assignment problem of diagnosable system, IEEE Trans. on Electronic Comput. EC-16, No.6, 1967, pp. 848-854.

[11]. Węglarz J., J.Błażewicz, W.Cellary, R.Słowiński : An automatic revised simplex method for constrained resource network scheduling, ACM Trans. on Mathematical Software 3, No.3 1977, pp.295-300.

Recenzent:Doc.dr hab.inż.Józef Grabowski
Wpłynęło do Redakcji do 30.03.1984r.

О ПРОБЛЕМЕ СОСТАВЛЕНИЯ РАСПИСАНИЙ МНОГОПРОЦЕССОРНЫХ ЗАДАЧ

Р е з ю м е

В работе рассмотрена проблема минимизации длины расписания.Предполагается, что некоторые задачи необходимо выполнять на двух или больше процессорах.Обсуждается также влияние прерывания задач на вычислительную сложность проблемы.

ON CERTAIN PROBLEMS OF SCHEDULING MULTIPROCESSOR TASKS

S u m m a r y

The problem to be considered is one of scheduling tasks to minimize a schedule length. It is assumed that certain tasks require for their processing two or more processors at a time. The impact of task divisibility on the complexity of the problem is also studied.