ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1998
Seria: AUTOMATYKA z. 123 Nr kol. 1389

Joanna JÓZEFOWSKA, Marek MIKA, Rafał RÓŻYCKI,
Grzegorz WALIGÓRA, Jan WĘGLARZ
Politechnika Poznańska

MINIMALIZACJA MAKSYMALNEGO OPÓŹNIENIA W DYSKRETNO-CIĄGŁYCH PROBLEMACH SZEREGOWANIA - ALGORYTMY HEURYSTYCZNE

Streszczenie. W pracy rozważa się dyskretno–ciągłe problemy szeregowania zadań z kryterium minimalizacji maksymalnego opóźnienia. Zakłada się, że zadania są niezależne i niepodzielne. Zasób dyskretny stanowią identyczne i równoległe maszyny, zasób ciągły jest odnawialny, a chwilowa prędkość wykonywania zadania zależy od ilości zasobu ciągłego przydzielonego zadaniu w danej chwili. Zaproponowano algorytmy przybliżone, które porównano na podstawie eksperymentu obliczeniowego.

MINIMIZATION OF MAXIMUM LATENESS IN DISCRETE-CONTINUOUS SCHEDULING PROBLEMS - HEURISTIC ALGORITHMS

Summary. In this paper we consider discrete-continuous scheduling problems with the objective to minimize the maximum lateness. We assume that jobs are independent and nonpreemptable and there are two types of resources. The discrete resource is represented by a set of identical and parallel machines, the continuous one is renewable and the processing rate of a job at time t depends of the amount of continuous resource allotted to this job at time t. Proposed heuristics have been compared on a basis of a computational experiment.

1. Wstęp

W pracy rozważa się dyskretno-ciągłe problemy szeregowania zadań z kryterium minimalizacji maksymalnego opóźnienia. Problemy takie występują w praktyce w sytuacjach, w których zadanie wymaga do wykonania nie tylko maszyny, ale także pewnej (nie znanej a priori) dodatniej ilości zasobu podzielnego w sposób ciągły (jak np. moc elektryczna, hydrauliczna lub pneumatyczna, środki finansowe, siła robocza itp.). Często też w zadaniach o charakterze produkcyjnym czy też projektowym zleceniodawca określa żądany termin zakończenia zadania, którego przekroczenie może spowodować dodatkowe koszty.

Metodologia rozwiązywania, jak i podstawowe własności dyskretno-ciągłych problemów szeregowania zostały omówione w pracy [5]. W pracy [2] przedstawiono i udowodniono podstawowe własności rozważanego problemu dla kryterium minimalizacji maksymalnego opóźnienia, w przypadku gdy liczba zadań jest nie większa od liczby maszyn. Pewne właściwości rozwiązań optymalnych dla szczególnych przypadków oraz zbudowane na ich podstawie algorytmy dokładne o złożoności wielomianowej zostały przedstawione w pracach [3] oraz [5]. Ponadto w pracach [1] oraz [2] przedstawiono pewne algorytmy heurystyczne, a w [3] zastosowano i porównano na podstawie eksperymentu obliczeniowego trzy algorytmy metaheurystyczne, tj. algorytmy symulowanego wyżarzania, przeszukiwania tabu i genetyczny. Główną zaletą podejść metaheurystycznych jest to, że z reguły dają one rozwiązania bardzo bliskie optymalnym (o ile nie optymalne). Niestety ich główną wadą jest to, że znalezienie takiego rozwiązania jest w niektórych przypadkach bardzo czasochłonne. Ponieważ w wielu przypadkach czas obliczeń jest istotniejszy niż jakość otrzymanego rozwiązania, niekiedy konieczne jest zastosowanie prostych algorytmów heurystycznych. W niniejszej pracy przedstawiono propozycję takich algorytmów dla rozważanego problemu szeregowania wraz z wynikami eksperymentu obliczeniowego.

2. Sformułowanie problemu

Dany jest n–elementowy zbiór niezależnych i niepodzielnych zadań. Każde zadanie żąda równocześnie zasobów dwóch kategorii, tj. zasobu dyskretnego i zasobu ciągłego. Zasobem dyskretnym jest dowolna maszyna z m–elementowego zbioru identycznych i równoległych maszyn. Zasób ciągły (tj. podzielny w sposób ciągły) może zostać przydzielony zadaniu, któremu już przydzielono maszynę, w pewnej (a priori nie znanej) dodatniej ilości. Od tego, ile tego zasobu przydzielimy zadaniu w danej chwili t, zależy chwilowa prędkość wykonywania tego zadania. Jest ona określona pewną ciągłą, niemalejącą i funkcją $f_i[u_i(t)]$, $i = 1, 2, \ldots, n$, gdzie $u_i(t) \in [0, 1]$ jest ilością zasobu ciągłego przydzielonego zadaniu i w chwili t. Dla każdego zadania jest znany jego rozmiar $\tilde{x}_i$, funkcja określająca szybkość wykonywania f_i oraz linia krytyczna d_i. Czasy gotowości wszystkich zadań są identyczne i wynoszą zero, co oznacza, że wszystkie zadania są dostępne od początku procesu. Szybkość wykonywania zadania i jest opisana następującym równaniem:

$$\frac{dx_i(t)}{dt} = f_i[u_i(t)], \quad x_i(0) = 0, x_i(C_i) = \tilde{x}_i.$$

gdzie: $x_i(t)$ jest stanem zadania i w chwili t;

$u_i(t) \in [0,1]$ jest ilością zasobu ciągłego przydzieloną do zadania i w chwili t;

$f_i(\cdot)$ jest niemalejącą i ciągłą funkcją, $f_i(0) = 0$;

C_i jest (nie znaną a priori) chwilą zakończenia wykonywania zadania i;

$\tilde{x}_i$ jest znanym stanem końcowym zadania i (rozmiarem zadania i).

Problem polega na znalezieniu takiej sekwencji zadań na maszynach i równocześnie takiego przydziału zasobu ciągłego do zadań, które minimalizują maksymalne opóźnienie $L_{max} = \max_i \{C_i - d_i$.

3. Własności problemu

Niech π będzie dyskretno–ciągłym problemem szeregowania n zadań na m maszynach z kryterium L_{max}. Każde zadanie jest określone przez $\tilde{x}_i$, d_i oraz f_i (i = 1, 2, ..., n). Jak wiadomo [3], dla funkcji $f_i \le c_i u_i$, $c_i = f_i(1)$, $i = 1, 2, \ldots, n$, optymalne rozwiązanie otrzymuje się przez uszeregowanie wszystkich zadań na jednej maszynie w kolejności niemalejących linii krytycznych i przydzielenie każdemu zadaniu całej ilości zasobu ciągłego w każdej chwili. Natomiast w przypadku wklęsłych funkcji f_i, i = 1, 2, ..., n pożądane jest równoległe wykonywanie zadań. W dalszym ciągu będziemy rozważać wyłącznie wklęsłe funkcje f_i.

Przypadek n ≤ m

W pracy [2] udowodniono, że w przypadku gdy liczba zadań n nie przekracza liczby maszyn m, uszeregowanie optymalne można otrzymać przez przydzielenie każdego zadania do innej maszyny i wykonanie ich z taką ilością zasobu ciągłego, że $d_1 - C_1 = d_2 - C_2 = \ldots = d_n - C_n = \Delta$ oraz $u_1 + u_2 + \ldots + u_n = 1$.

Przypadek n > m

Zgodnie z metodologią przedstawioną w [4] uszeregowanie dopuszczalne można przedstawić za pomocą tzw. ciągu dopuszczalnego oraz podziału rozmiarów zadań. Ciąg dopuszczalny składa się z n kombinacji $Z_1, Z_2, \ldots, Z_n$ zadań takich, że:

i) każde zadanie występuje w co najmniej jednej kombinacji;

ii) jeżeli zadanie występuje w więcej niż jednej kombinacji, to, ze względu na niepodzielność zadań, muszą to być kombinacje kolejne;

iii) każda kombinacja zawiera co najwyżej m zadań (pierwsze n – m + 1 kombinacji zawiera dokładnie m zadań, następne zawierają o jedno zadanie mniej w stosunku do kombinacji poprzedniej, tak więc kombinacja Z_n składa się tylko z jednego zadania).

Jeżeli przez K_i (i = 1, 2, ...,n) oznaczymy zbiór indeksów wszystkich kombinacji, w których występuje zadanie i, to rozmiar $\bar{x}_i$ zadania i można podzielić na części $\bar{x}_{ik} \geq 0$, (i = 1, 2, ...,n; $k \in K_i$), które określają, jaka część zadania i będzie wykonana w kombinacji Z_k. Dla rozważanego kryterium optymalizacji oraz dla danego ciągu dopuszczalnego możemy znaleźć optymalne wartości $\bar{x}_{ik}$ rozwiązując następujący problem programowania matematycznego[3]:

Problem P.

Zminimalizować z

przy ograniczeniach:

$$\sum_{k \in K_i} \bar{x}_{ik} = \bar{x}_i, \qquad i = 1, 2, ..., n$$

$$\sum_{i \leq k} M_i^* - d_k \leq z, \qquad k = 1, 2, ..., n$$

$$\bar{x}_{ik} \geq 0 \qquad i = 1, 2, ..., n; k \in K_i$$

gdzie M_i^* jest jedynym dodatnim pierwiastkiem równania $\sum_{i \in Z_k} f_i^{-1}\left(\frac{\bar{x}_{ik}}{M_k^*}\right) = 1$.

Tak otrzymane uszeregowanie nazywamy semioptymalnym, czyli uszeregowaniem, w którym dla danego ciągu dopuszczalnego znaleziono optymalny rozdział zasobu ciągłego. Proces znajdowania uszeregowania można więc podzielić na dwa etapy. Pierwszy z nich ma charakter kombinatoryczny i polega na zbudowaniu ciągu dopuszczalnego. Drugi polega na znalezieniu optymalnego rozdziału zasobu ciągłego w danym ciągu dopuszczalnym przez rozwiązanie problemu P. W ogólności jednak, aby znaleźć rozwiązanie optymalne tego problemu, musimy znaleźć uszeregowania semioptymalne dla wszystkich ciągów dopuszczalnych i spośród nich wybrać to, które optymalizuje rozważane kryterium. Podejście takie jest jednak nieefektywne obliczeniowo, ze względu na to, że liczba ciągów dopuszczalnych rośnie wykładniczo wraz ze wzrostem liczby zadań. W związku z tym proponujemy podejścia, w których analizujemy tylko jeden ciąg dopuszczalny.

4. Analiza szczególnych przypadków

Przeanalizujmy dwa przypadki szczególne rozważanego problemu. Pierwszy, w którym mamy do czynienia z tzw. „luźnymi" liniami krytycznymi, został przedstawiony w pracy [3], gdzie również podano dla niego algorytm optymalny o złożoności wielomianowej. Z problemem takim mamy do czynienia zawsze wtedy, gdy spełniona jest nierówność $(d_i - d_{i-1}) \cdot f_i(1) \geq \tilde{x}_i$, i = 2, 3, ..., n. Wówczas rozwiązanie optymalne otrzymamy wykonując wszystkie zadania na jednej maszynie, w kolejności niemalejących linii krytycznych, przy użyciu maksymalnej dostępnej ilości zasobu ciągłego.

Drugi przypadek zachodzi, gdy mamy do czynienia z tzw. „ciasnymi" liniami krytycznymi. W szczególności wszystkie linie krytyczne mogą być identyczne, tj. $d_1 = d_2 = ... = d_n$. Wówczas, jak łatwo zauważyć, uszeregowanie, które stanowi rozwiązanie problemu minimalizacji długości uszeregowania C_{max}, jest również rozwiązaniem problemu minimalizacji maksymalnego opóźnienia L_{max}.

5. Podejścia heurystyczne

W pracy [2] przedstawiono propozycję algorytmu, który polegał na uporządkowaniu zadań według niemalejących linii krytycznych i następnie przydzieleniu pierwszych m zadań z tak otrzymanej listy do wolnych maszyn i wykonaniu ich przy użyciu optymalnej ilości zasobu ciągłego. Po wykonaniu ostatniego z tych m zadań, następne m zadań z listy jest wykonywane w taki sam sposób. Całość jest powtarzana, aż do wykonania ostatniego zadania z listy. Jak łatwo zauważyć, zasadniczą wadą tego algorytmu jest fakt, że nie wszystkie maszyny są wykorzystywane przez cały czas trwania procesu.

Ulepszoną wersję tego algorytmu przedstawiono w [1] pod nazwą algorytmu dynamicznego. Można go przedstawić następująco:

1. Posortuj zadania według niemalejących d_i;
2. z := 0;
3. Uszereguj pierwszych m zadań z listy na m maszynach (każde zadanie na innej maszynie);
4. z := z + 1;
5. Wykonywanym zadaniom przydziel zasób ciągły w sposób optymalny;

6. Wykonuj zadania na przydzielonych maszynach z przydzielonymi zasobami do momentu zakończenia pierwszego z nich (tj. zadania z);
7. Oblicz, jaka część pozostałych zadań (z + 1, ..., z + m - 1) nie została jeszcze wykonana w momencie zakończenia zadania z. Przydziel te fragmenty zadań do maszyn, na których te zadania były dotąd wykonywane;
8. Sprawdź, czy na liście istnieje jakieś nie wykonywane jeszcze zadanie, tj. czy z + m < n. Jeśli nie, to przejdź do punktu 10;
9. Uszereguj zadanie o indeksie z + m na wolnej maszynie, tj. na maszynie o indeksie z mod m. Przejdź do punktu 4;
10. Sprawdź, czy z = n. Jeśli tak, to STOP (wszystkie zadania zostały wykonane) w przeciwnym przypadku przejdź do punktu 4.

Jak widać, uszeregowanie otrzymane przy zastosowaniu tego algorytmu nie zawiera przestojów maszyn w czasie realizacji procesu. Nie jest to jednak algorytm optymalny. Przy dokładniejszej analizie można zauważyć, że zawsze prowadzi on do ciągu dopuszczalnego postaci:

$$\{1, 2, 3, ..., m\}, \{2, 3, ..., m, m + 1\}, ..., \{n - m+1, n - m+2, ..., n - 1, n\}, ..., \{n - 1, n\}\ \{n\}$$

Zauważmy, że znajdując rozwiązanie optymalne Problemu P dla powyższego ciągu dopuszczalnego, zawsze znajdziemy rozwiązanie nie gorsze pod względem rozważanego kryterium od rozwiązania otrzymanego przy użyciu omawianego algorytmu. Dlatego też pierwszy z proponowanych przez nas algorytmów wykorzystuje powyższą obserwację.

ALGORYTM EDF4DC

Algorytm EDF4DC jest adaptacją dobrze znanego algorytmu EDF (Earliest-Deadline-First) dla dyskretno-ciągłych problemów szeregowania. W tej wersji algorytmu przy budowaniu ciągu dopuszczalnego rozmiary zadań nie zostały uwzględnione, dlatego ciąg ten zawsze będzie miał identyczną postać. Algorytm można przedstawić następująco:

1. Uporządkuj zadania według niemalejących linii krytycznych;
2. Zbuduj ciąg dopuszczalny wg następujących zasad:
   a) przydziel m pierwszych zadań do kombinacji Z_1;
   b) kombinację Z_i (i = 2, 3, ..., n) stwórz kopiując kombinację Z_{i-1}, usuwając z niej zadanie o najmniejszej linii krytycznej i umieszczając na jego miejscu zadanie

o najmniejszej linii krytycznej spośród zadań jeszcze nie przydzielonych, o ile takowe istnieje.

3. Dla tak zbudowanego ciągu dopuszczalnego rozwiąż optymalnie Problem P.

ALGORYTMY EDF4DC(ρ)

Jest to grupa algorytmów, również opartych na algorytmie EDF, w których przy budowaniu ciągu dopuszczalnego wykorzystano informację o rozmiarze zadania $\tilde{x}_i$ oraz o postaci funkcji f_i. Przebieg algorytmu z tej grupy jest następujący:

1. Uporządkuj zadania wg niemalejących linii krytycznych;
2. Każdemu zadaniu przydziel pewną ilość zasobu ciągłego $u_i = \rho$, $i = 1, \ldots, n$, której wartość dla poszczególnych algorytmów zostanie podana poniżej;
3. Dla każdego zadania oblicz wstępne czasy przetwarzania ze wzoru $p_i = \dfrac{\tilde{x}_i}{f_i(\rho)}$;
4. Korzystając z algorytmu EDF zbuduj uszeregowanie pomocnicze w oparciu o wstępne czasy przetwarzania;
5. Na podstawie otrzymanego uszeregowania pomocniczego zbuduj ciąg dopuszczalny, w którym kolejne kombinacje będą określone przez czasy zakończeń poszczególnych zadań w uszeregowaniu pomocniczym.
6. Dla tak zbudowanego ciągu dopuszczalnego rozwiąż optymalnie Problem P.

Wartość ρ dla poszczególnych wersji algorytmu przedstawiono w tablicy 1

Tablica 1

Wartość ρ dla różnych wersji algorytmu EDF4DC(ρ)

Algorytm	EDF4DC(1/2)	EDF4DC(1/m)	EDF4DC(1/n)	EDF4DC(1/α)
ρ	1/2	1/m	1/n	$1/\alpha_i$

ALGORYTMY LPT4DC i SPT4DC

Algorytmy te są adaptacją dobrze znanych algorytmów LPT (Longest Processing Time) oraz SPT (Shortest Processing Time) dla dyskretno-ciągłych problemów szeregowania. Ponieważ czasy przetwarzania poszczególnych zadań nie są znane a priori, przy stosowaniu algorytmu LPT(SPT) wykorzystano zamiast nich rozmiary zadań $\tilde{x}_i$. Algorytm LPT4DC (SPT4DC) można przedstawić następująco:

1. Uporządkuj zadania wg nierosnących (niemalejących) rozmiarów zadań $\tilde{x}_i$;

2. Korzystając z algorytmu LPT (SPT) zbuduj uszeregowanie pomocnicze w oparciu o rozmiary zadań $\tilde{x}_i$;
3. Na podstawie otrzymanego uszeregowania pomocniczego zbuduj ciąg dopuszczalny, w którym kolejne kombinacje będą określone przez czasy zakończeń poszczególnych zadań w uszeregowaniu pomocniczym.
4. Dla tak zbudowanego ciągu dopuszczalnego rozwiąż optymalnie Problem P.

6. Eksperyment obliczeniowy

Jakość przedstawionych algorytmów sprawdzono na podstawie eksperymentu obliczeniowego na zbiorze losowo wygenerowanych instancji. Wszystkie algorytmy zostały zaimplementowane w języku C++ i następnie wykonane na komputerze SGI PowerChallenge XL w Poznańskim Centrum Superkomputerowo-Sieciowym. Do rozwiązywania Problemu P użyto przystosowanego do rozważanego problemu kodu solvera CFSQP (A C Code for Solving (Large Scale) Constrained Nonlinear (Minimax) Optimization Problems, Generating Iterates Satisfying All Inequality Constraints) 2.3 [6].

Eksperyment został przeprowadzony dla funkcji prędkości przetwarzania zadań określonej wzorem $f_i = u_i^{1/\alpha_i}$. Wartości $\tilde{x}_i$ oraz d_i zostały wygenerowane losowo z przedziału [1,100] zgodnie z rozkładem równomiernym, natomiast wartości parametru α_i zostały wygenerowane również losowo ze zbioru $\{1, 2\}$ z równym prawdopodobieństwem. Testy przeprowadzono dla liczby zadań n = 10 oraz liczby maszyn m = 2, 3 oraz 4. Dla każdej pary (n, m) wygenerowano po 50 instancji. Wyniki eksperymentu przedstawiono w tablicy 2. Dla każdego algorytmu heurystycznego podano, ile razy dany algorytm znalazł rozwiązanie nie gorsze od rozwiązań znalezionych przez pozostałe algorytmy heurystyczne (A), maksymalne odchylenie od najlepszego rozwiązania (B), oraz odpowiadającą mu (najlepszą znalezioną przez wszystkie algorytmy heurystyczne) wartość L_{max} (C). W tablicy 3 przedstawiono porównanie wyników otrzymanych przy użyciu proponowanych algorytmów z wynikami otrzymanymi przez metaheurystyki [3]. W porównaniu tym w przypadku algorytmów heurystycznych uwzględniono tylko najlepsze rozwiązania znalezione przez którykolwiek z nich.

Tablica 2

Wyniki eksperymentu obliczeniowego – porównanie heurystyk

m		*EDF4DC*	*SPT4DC*	*LPT4DC*	*EDF4DC(1/2)*	*EDF4DC(1/m)*	*EDF4DC(1/n)*	*EDF4DC(1/α)*
2	A	1	17	5	2	3	2	24
	B	49.734	17.107	64.591	29.869	29.869	74.319	20.993
	C	302.625	261.932	396.108	261.932	261.932	377.359	290.028
3	A	1	8	5	3	3	0	31
	B	64.295	45.730	78.478	33.047	45.498	92.492	30.836
	C	211.971	344.125	353.142	381.208	381.208	317.913	266.467
4	A	0	4	8	6	4	1	31
	B	49.792	45.892	46.884	25.134	34.844	67.495	27.761
	C	137.548	274.801	321.381	350.026	343.316	317.163	196.171

Tablica 3

Wyniki eksperymentu obliczeniowego – porównanie z metaheurystykami

m		*TS*	*S.A.*	*GA*	*Heurystyki*
2	A	16	2	7	0
	B	7.317	0.910	0.870	20.909
	C	301.539	310.112	314.492	308.773
3	A	9	0	5	0
	B	5.974	7.323	2.365	25.208
	C	257.193	216.801	268.847	351.674
4	A	7	0	0	0
	B	0	3.249	2.360	14.704
	C	–	348.622	243.037	298.278

Podobnie jak w tablicy 2 podano, ile razy dany algorytm znalazł rozwiązanie nie gorsze od rozwiązań znalezionych przez pozostałe rozważane algorytmy (A), maksymalne odchylenie od najlepszego rozwiązania (B), oraz odpowiadającą mu (najlepszą znalezioną przez wszystkie algorytmy) wartość L_{max} (C). Każda z metaheurystyk odwiedziła 2000 rozwiązań. Ze względu na dużą czasochłonność obliczeń w przypadku algorytmów metaheurystycznych, liczba rozważanych instancji dla poszczególnych rozmiarów problemów była mniejsza niż 50 i wynosiła odpowiednio: 21 dla m = 2, 14 dla m = 3 oraz 7 dla m = 4.

7. Podsumowanie

W pracy zaproponowano algorytmy heurystyczne dla dyskretno-ciągłych problemów szeregowania zadań z kryterium minimalizacji maksymalnego opóźnienia.

Na podstawie analizy teoretycznej stwierdzono, że algorytm dynamiczny przedstawiony w pracy [1] może być w wielu sytuacjach z powodzeniem zastąpiony przez

algorytm EDF4DC oraz jego mutacje. Jednakże są dwa przypadki, w których zalety algorytmu dynamicznego są wyraźne. Pierwszy występuje, gdy liczba zadań jest dużo większa od liczby maszyn i tu głównie zyskamy na czasie obliczeń. Drugi przypadek dotyczy sytuacji, gdy liczba zadań zgłaszających się do systemu nie jest znana a priori, zadania te mają różne czasy gotowości, a informacja o nich nie jest dostępna wcześniej niż w chwili pojawienia się zadania w systemie.

Proponowane algorytmy te porównano między sobą oraz z proponowanymi wcześniej algorytmami metaheurystycznymi na podstawie eksperymentu obliczeniowego.

Z porównania tego wynika, że heurystyka EDF4DC($1/\alpha$) znajduje największą liczbę rozwiązań najlepszych spośród heurystyk. Również jej maksymalne odchylenie od najlepszego znalezionego rozwiązania przyjmuje stosunkowo małe wartości niezależnie od rozmiaru problemu. Oczywiście rozwiązania znajdowane nawet przez najlepszą heurystykę nie są równie dobre, jak rozwiązania znajdowane przez metaheurystyki, ale należy pamiętać, że metaheurystyki wymagają kilkaset razy większych nakładów obliczeniowych.

LITERATURA

1. Janiak A., Przysada J.: Parallel Machine Scheduling with Resource Allocation-Minimization of Maximum Lateness, maszynopis, 1997.
2. Janiak A., Przysada J.: Minimalizacja nieterminowości wykonania zadań na równoległych maszynach. Zeszyty Naukowe Politechniki Śląskiej, seria: Automatyka, z. 117, Gliwice 1996, s. 85-97.
3. Józefowska J., Mika M., Różycki R., Waligóra G., Węglarz J.: Discrete-Continuous Scheduling to Minimize Maximum Lateness. Proc. of 4th International Symposium on Methods and Models in Automation and Robotics, Międzyzdroje, 1997, pp. 947-952.
4. Józefowska J., Węglarz J.: On a Methodology for Discrete-Continuous Scheduling. European Journal of Operational Research, Vol. 107/2, 1998.
5. Józefowska J.: Dyskretno-ciągłe problemy szeregowania zadań. Wydawnictwo Politechniki Poznańskiej, seria: Rozprawy, nr 318, Poznań 1998.
6. Lawrence C., Zhou J.L., Tits A.L.: Users guide for CFSQP Version 2.3 (Released August 1995), available by e-mail: andre@eng.umd.edu.

Recenzent: Prof.dr hab.inż.Ryszard Gessing

Abstract

In this paper we consider discrete-continuous scheduling problems with the objective to minimize the maximum lateness. We assume that jobs are independent and nonpreemptable and there are two types of resources. The discrete resource is represented by a set of identical and parallel machines, the continuous one is renewable and the processing rate of a job at time t depends of the amount of the continuous resource allotted to this job at time t. Simple heuristic algorithms are proposed and compared on a basis of a computational experiment. The algorithms are also compared with other heuristics proposed in the literature. Their main advantage is short computational time. They are outperformed by metaheuristic algorithms (Tabu Search, Simulated Annealing and Genetic Algorithms), however at a cost of much larger computational effort.