ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1998
Seria: AUTOMATYKA z. 125 Nr kol. 1391

Józef GRABOWSKI, Jarosław PEMPERA
Politechnika Wrocławska

ALGORYTMY SZEREGOWANIA ZADAŃ Z KRYTERIUM MINIMALNO-KOSZTOWYM

Streszczenie. W niniejszej pracy przedstawia się szereg algorytmów heurystycznych dla zagadnienia szeregowania zadań na jednej maszynie z kryterium optymalizacji sumy kosztów zadań wykonywanych nieterminowo. Przedstawiono wyniki obliczeniowe oraz analizę porównawczą

ALGORITHMS OF SEQUENCING OF JOBS FOR TOTAL WEIGHTED TARDINESS PROBLEM

Summary. In the paper, one - machine sequencing problem is considered under condition that the total weighted tardiness cost is minimized. Some approximation algorithms, computation results and discussion of the performance of algorithms are presented.

1. Wstęp

W niniejszej pracy przedmiotem rozważań jest zagadnienie szeregowania zadań na jednej maszynie, w którym kryterium optymalizacji jest sumą kosztów zadań wykonywanych nieterminowo. Tego rodzaju zagadnienia występują w wielu sytuacjach praktycznych, zwłaszcza w komputerowo zintegrowanym wytwarzaniu (CIM) oraz w systemach wytwarzania na żądanie (JIT). Ponadto zagadnienie to może pojawić się przy optymalizacji gniazd krytycznych, gdzie tylko jedna z maszyn ma ograniczoną przepustowość ("wąskie gardło"), natomiast pozostałe maszyny, ze względu na dużą moc produkcyjną, posiadają nieograniczoną przepustowość ("bez ograniczeń"), zatem nie musimy zakładać tutaj konieczności szeregowania zadań.

W pracy przedstawiono: sformułowanie zagadnienia, krótki przegląd literatury, algorytmy heurystyczne, wyniki obliczeniowe oraz analizę porównawczą.

2. Sformułowanie problemu

Dany jest zbiór n zadań $J=\{J_1,J_2,...,J_n\}$, które mają być wykonane za pomocą jednej maszyny. Zakładamy dalej, że maszyna ta może wykonywać w danej chwili tylko jedno zadanie

oraz, że wykonywanie zadania nie może być przerywane. Dla każdego zadania J_i, $(i=1,2,...,n)$ określony jest czas trwania wykonania p_i oraz żądany termin zakończenia realizacji d_i. Przekroczenie terminu d_i związane jest z poniesieniem dodatkowych kosztów, które zależą od "wagi" zadania w_i oraz długości spóźnienia: $T_i=max(0,C_i-d_i)$, gdzie C_i jest terminem zakończenia realizacji zadania J_i. Zagadnienie optymalizacji polega na określeniu takiej kolejności wykonywania zadań na maszynie, aby suma ważonych kosztów spóźnień $\sum_{i=1}^{n} w_i T_i$ była minimalna. Zagadnienie to (NP-trudne) jest jednym z najtrudniejszych zagadnień kolejnościowych. Zastosowanie metod dokładnych ze względu na czas obliczeń staje się niemożliwe już dla zagadnień o niewielkich rozmiarach [1],[4],[7]. Stąd też, aktualnie, w literaturze badania są ukierunkowane na budowę algorytmów heurystycznych. Ogólnie algorytmy heurystyczne dzielą się na dwie klasy: algorytmy typu konstrukcyjnego oraz algorytmy lokalnej optymalizacji, opartej na zamianie kolejności zadań wewnątrz danego uszeregowania. Algorytmy tego rodzaju nazywane są niekiedy algorytmami popraw.

3. Algorytmy konstrukcyjne

W niniejszym rozdziale przedstawimy krótko kilka algorytmów konstrukcyjnych najczęściej spotykanych w literaturze. Najprostszymi algorytmami tego typu są dwa algorytmy: SWPT (Shortest Weighted Proccesing Time) [8] i (EDD) Earliest Due Date [2]. Złożoność obliczeniowa tych algorytmów jest $O(n\ln n)$. Algorytm SWPT szereguje zadania zgodnie z niemalejącymi wartościami p_i/w_i, natomiast algorytm EDD szereguje zgodnie z niemalejącymi wartościami d_i. W pracy [8] pokazano, że jeżeli żadne z zadań nie może być wykonane w terminie (czyli jeżeli $p_i>d_i$ dla wszystkich $i=1,2,...,n$), to algorytm SWPT znajduje rozwiązanie optymalne.

Kolejnymi algorytmami konstrukcyjnymi są algorytmy COVERT (Cost Over Time) oraz AU (Apparent Urgency) [5]. W algorytmach tych do częściowego uszeregowania zadań dołączane jest kolejne zadanie wybrane zgodnie z pewną dynamiczną regułą priorytetu. I tak, w algorytmie CONVERT wybierane jest zadanie J_i o największej wartości $I_i w_i/p_i$, gdzie :

$$I_j = \begin{cases} 1 & \text{jeżeli } d_j \le P(S) + p_j \\ [P(S \cup E) - d_j) \,/\, (P(E) - p_j] & \text{jeżeli } P(S) + p_j < d_j < P(S \cup E \\ 0 & \text{jeżeli } P(S \cup E) + p_j \le d_j \end{cases}$$

$P(Q) = \sum_{j \in Q} p_j$, S - zbiór zadań znajdujących się w częściowym uszeregowaniu,

E - zbiór zadań, które nie są jeszcze uszeregowane $(E \cap S=\emptyset)$.

Natomiast w algorytmie AU wybierane jest zadanie o największej wartości AU_i, gdzie :

$$AU_j = \frac{w_j}{p_j} \exp(- \max(0, d_j - P(S) - p_j) \,/\, (k\overline{p}) \quad ,$$

k - jest parametrem algorytmu związanym z rozpiętością wartości d_j zagadnienia,
$\overline{p}$ - jest średnim czasem wykonywania zadań.

Złożoność obliczeniowa algorytmów COVERT oraz AU jest $O(n^2)$.

4. Algorytmy lokalnej optymalizacji

Zwykle, w tego rodzaju metodach, algorytm rozpoczyna działanie od pewnego dopuszczalnego rozwiązania początkowego. Dla tego rozwiązania określa się zbiór rozwiązań "sąsiednich " oraz kolejność przeglądania tego zbioru. Podstawowy krok algorytmu polega na znalezieniu w zbiorze rozwiązań sąsiednich rozwiązania lepszego (lub najlepszego), które staje się rozwiązaniem początkowym w następnym kroku.

Dla naszego zagadnienia algorytmy tego rodzaju były stosowane w niewielkim zakresie [3],[5]. Stąd też tutaj zamierzamy zaprezentować szereg takich algorytmów wraz z wynikami obliczeniowymi.

Niech permutacja π określa kolejność wykonywania zadań $J_1,J_2,...,J_n$ dla naszego zagadnienia. Dalej, niech $v=(x,y)$ będzie parą liczb całkowitych określających wyróżnione pozycje x oraz y w permutacji π. Ogólnie rzecz ujmując, zadania znajdujące się na tych pozycjach będą usuwane i umieszczane na innych pozycjach w π, tworząc w ten sposób nową permutację π^v. Stąd też para $v=(x,y)$ jest nazywana "ruchem". Niech $V=\{v=(x,y)\}$ będzie pewnym zbiorem ruchów oraz niech $\prod^V$ będzie zbiorem rozwiązań sąsiednich (permutacji π^v), generowanych poprzez ruchy ze zbioru V. Podstawowym elementem w algorytmach lokalnej optymalizacji jest określenie zasad konstrukcji zbioru rozwiązań sąsiednich oraz kolejności ich przeglądania (generowania). Dalej niech $\overline{V}$ oznacza uporządkowany (wg. kolejności użycia) ciąg ruchów ze zbioru V. Ciąg taki będziemy nazywać schematem. Niekiedy schemat $\overline{V}$ zawiera tylko pewien podzbiór ruchów ze zbioru V. W pracy badane były dwie niżej opisane zasady konstrukcji zbioru V oraz schematu $\overline{V}$ [9].

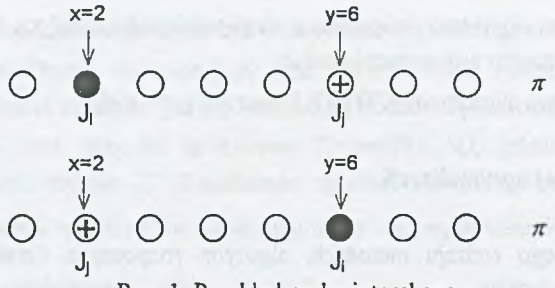
Konstrukcja A (typu "interchange")

W tej konstrukcji ruch $v=(x,y)$ będzie oznaczał wymianę zadań J_i oraz J_j, znajdujących się, odpowiednio, na pozycjach x oraz y w π (rys.1). W tym przypadku zbiór ruchów jest określony następująco:

$$V_A=\{v=(x,y)|(y>x)\wedge(x,y\in\{1,2,...,n\})\}$$

W konstrukcji A przyjęliśmy następujący schemat (uporządkowany zbiór ruchów):

$$\overline{V_A}=((1,2),(1,3),(1,4),...,(1,n),(2,3),...,(n-1,n))$$

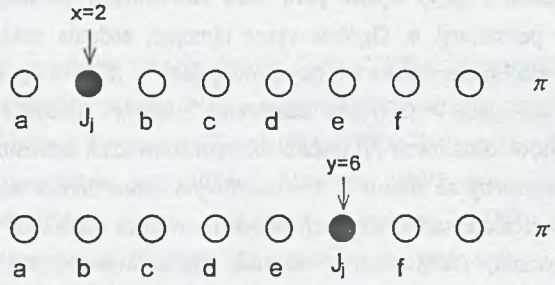


Rys. 1. Przykład ruchu interchange
Fig. 1. An example of the interchange move

Konstrukcja B (typu "insert")

W tej konstrukcji ruch $v=(x,y)$ będzie oznaczał pobranie zadania J_j znajdującego się na pozycji x oraz wstawienie go na pozycję y w π (rys.2). Innymi słowy, zadanie J_j z pozycji x jest przenoszone na pozycję y. W tym przypadku zbiór ruchów jest określony następująco:

$$V_B=\{v=(x,y)|(y\neq x)\wedge(y\neq x-1)\wedge(x,y\in\{1,2,...,n\})\}$$



Rys. 2. Przykład ruchu insert
Fig. 2. An example of the insert move

W konstrukcji B przyjęliśmy następujący schemat :

$$\overline{V_B}=((1,2),(1,3),(1,4),...,(1,n),(2,1),(2,3)...,(n,n-1))$$

Kolejnym elementem w algorytmach lokalnej optymalizacji jest określenie strategii wykorzystania wartości funkcji celu rozwiązań sąsiednich generowanych przy użyciu ruchów ze schematu $\overline{V}$. W pracy badane były następujące strategie:

Strategia 1

W tej strategii algorytm startuje z pewnego rozwiązania początkowego π; generuje wszystkie rozwiązania sąsiednie przy użyciu ruchów v ze schematu $\overline{V}$ dla π. Dla każdego rozwiązania π^v jest obliczana wartość funkcji celu $f(\pi^v)$. Jeżeli po wygenerowaniu wszystkich rozwiązań znaleziono permutację β taką, że $f(\beta)$ jest najmniejszą wartością i taką, że $f(\beta)<f(\pi)$, to β staje się rozwiązaniem początkowym i algorytm kontynuuje obliczenia od

początku schematu dla nowego rozwiązania początkowego β. Jeżeli nie znaleziono rozwiązania lepszego niż π, to algorytm kończy obliczenia.

W tej strategii kolejność generowania rozwiązań sąsiednich (kolejność użycia ruchów) może być dowolna.

Strategia 2

Algorytm generuje rozwiązania sąsiednie wg schematu $\overline{V}$ dla π, jeżeli znajdzie β taką, że $f(\beta)<f(\pi)$, to w tym momencie β staje się rozwiązaniem początkowym i algorytm kontynuuje obliczenia od początku schematu dla nowego rozwiązania początkowego β (generowanie rozwiązań sąsiednich dla π jest przerywane).

Strategia 3

Tutaj (podobnie jak dla strategii 2) algorytm znajdując β taką, że $f(\beta)<f(\pi)$, określa także numer porządkowy N ruchu v (który wygenerował β) w schemacie $\overline{V}$. Wtedy β staje się rozwiązaniem początkowym i algorytm kontynuuje obliczenia rozpoczynając od $N+1$ ruchu schematu. Algorytm kończy obliczenia po przeglądnięciu określonej (przez użytkownika) liczby *ITER* pełnych schematów.

Strategia 4

Taka sama jak strategia 3, z tym że zamiast ostrej nierówności $f(\beta)<f(\pi)$ mamy tutaj $f(\beta)\leq f(\pi)$.

Łącząc odpowiednio konstrukcje A oraz B ze strategiami 1,2,3 oraz 4, opracowaliśmy następujące algorytmy bazowe: A1,B1,A2,B2,A3,B3,A4,B4. Przykładowo, algorytm A3 oznacza połączenie konstrukcji A (która posiada schemat generowania rozwiązań sąsiednich $\overline{V_A}$) ze strategią 3 wykorzystywania wartości funkcji celu.

Dysponując algorytmami bazowymi możemy skonstruować algorytmy o bardziej złożonej strukturze. W niniejszej pracy badaliśmy kilka algorytmów o strukturze szeregowej i równoległej. Dla przykładu, struktura szeregowa A1B1 oznacza, że najpierw rozwiązanie początkowe jest przekształcane (poprawiane) przez A1, po czym rozwiązanie wynikowe tego algorytmu staje się początkowe dla B1. Następnie, po przekształceniu przez B1, rozwiązanie wynikowe staje się początkowe dla A1, itd.

Natomiast struktura równoległa A1/B1 oznacza, że rozwiązanie początkowe jest przekształcane równolegle (równocześnie) przez A1 oraz B1, następnie rozwiązania wynikowe obu algorytmów są porównywane i rozwiązanie lepsze staje się początkowe dla A1/B1, itd. Oczywiście, łatwo sobie wyobrazić zasady funkcjonowania struktur zawierających więcej niż dwa algorytmów bazowych.

Zestaw badanych algorytmów złożonych przedstawiono w następnym punkcie pracy.

5. Wyniki badań testujących

Algorytmy lokalnej optymalizacji (LO) przedstawione w poprzednim rozdziale porównano z pewnymi heurystykami spotykanymi w literaturze. Heurystykami tymi były META oraz SAPW[4]. Ta ostatnia huerystyka (jedna z najlepszych) jest oparta na metodzie symulowanego wyżarzania: Algorytm META natomiast jest kompozycją algorytmów konstrukcyjnych SWPT, EDD, COVERT i AU, opisanych w trzecim rozdziale. Kompozycja ta polegała na tym, że każdy przykład był rozwiązywany za pomocą tych czterech algorytmów, a najlepsze rozwiązanie było uznane za wynik działania algorytmu META.

Ogólna zasada działania algorytmu SAPW wygląda następująco. W algorytmie tym rozwiązania sąsiednie generowane są wg schematu $\overline{V_A}$ dla pewnego rozwiązania początkowego π. Dla każdego wygenerowanego rozwiązania sąsiedniego π^v obliczana jest wartość $f(\pi^v)$, która porównywana jest z wartością $f(\pi)$. Jeżeli $f(\pi^v) \le f(\pi)$, wówczas π^v staje się rozwiązaniem początkowym. W przeciwnym przypadku rozwiązanie π^v może stać się rozwiązaniem początkowym z prawdopodobieństwem p. Prawdobieństwo p zależy od wartości różnicy $\Delta = f(\pi^v) - f(\pi)$ oraz od wartości funkcji celu $f^* = f(\pi^*)$ dla najlepszego dotychczas znalezionego rozwiązania π^*, a także od wartości parametru K_i, zmniejszającego się liniowo wraz ze wzrostem numeru iteracji algorytmu (jedna iteracja oznacza wykonanie pełnego schematu ruchów). Prawdopodobieństwo p wyznacza się z zależności $p = \exp\left(\frac{-\Delta}{-0.01 f^*} \ln(K_i)\right)$, gdzie: $K_{i+1} = aK_i$, $a = \left(K_f / K_0\right)^{\frac{1}{L-1}}$, natomiast K_f, K_0, L są parametrami algorytmu. W przypadku zaakceptowania nowego rozwiązania, podobnie jak w strategiach 3 i 4, zapamiętywany jest numer porządkowy N ruchu v, który wygenerował rozwiązanie π^v i algorytm kontynuuje obliczenia dla nowego rozwiązania początkowego od $N+1$ ruchu w schemacie. Algorytm kończy działanie, gdy $K_i = K_f$.

Algorytmy lokalnej optymalizacji, badane w niniejszej pracy, zostały zakodowane w języku C++ i były testowane na komputerze IBM RISC System/6000, 200 MHz. Badania testowe (podobnie jak w [5]) przeprowadzono na przykładach, dla których wartości p_{jk}, w_i oraz d_i zostały wygenerowane według rozkładu równomiernego, przy czym wartości p_{ik} zostały wygenerowane ze zbioru (1,2,...,100), w_i ze zbioru (1,...,10), natomiast d_i ze zbioru $[P(1-TF-RDD/2), P(1-TF+RDD/2))]$, gdzie $P = \sum_{i=1}^{n} p_i$, RDD i TF są parametrami określającymi zakres wartości terminu zakończenia realizacji oraz średni rozrzut wartości tego terminu. Badania zostały przeprowadzone dla wartości RDD i TF ze zbioru {0.2, 0.4, 0.8, 1.0} oraz n=20, 40, 50 i 100. Dla każdej kombinacji wartości n, RDD i TF wygenerowano 100 przykładów testujących, otrzymując w sumie 10,000 problemów. Rozwiązanie początkowe dla algorytmów LO i SAPW otrzymano przy użyciu algorytmu AU z k=0.5, 0.9, 2.0, 2.0, 2.0 dla TF= 0.2, 0.4, 0.6, 0.8, 1.0.

Dla każdego przykładu wyliczono następujące wielkości :

C^H - wartość funkcji celu otrzymana algorytmem H,

$C^m = min_H(C^H)$ - najmniejsza wartość funkcji celu otrzymana jednym z badanych algorytmów.

Na podstawie tych wielkości obliczono:

$r^H = 100\%(C^H-C^m)/C^m$ - względny błąd rozwiązania otrzymanego algorytmem H względem najlepszego rozwiązania otrzymanego jednym z badanych algorytmów,

p^H - procent przykładów dla których $C^H=C^m$,

$r^H_{śr}$ - średni względny błąd dla algorytmu H.

W tablicy 1 przedstawiono wyniki badań testujących.

6. Wnioski końcowe

Z przeprowadzonych badań testujących wynika, że algorytmy LO, badane przez autorów, są zdecydowanie lepsze od algorytmów konstrukcyjnych reprezentowanych przez algorytm META. Ponadto, większość złożonych algorytmów LO (szeregowych i równoległych) daje lepsze wyniki (zwłaszcza dla większych n) niż heurystyka oparta na metodzie symulowanego wyżarzania SAPW.

Tablica 1

Wyniki badań

Algorytm H	n=20		n=40		n=50		n=100	
	p^H	$r^H_{śr}$	p^H	$r^H_{śr}$	p^H	$r^H_{śr}$	p^H	$r^H_{śr}$
META	13	9,85	6	12,34	6	13,81	5	17,55
SAPW (L=50,K_0=0.6,K_f=10^{-4})	58	0,25	24	0,73	21	0,92	12	2,12
Algorytmy bazowe								
A1	64	0,81	41	0,65	35	0,63	17	0,57
A2	64	0,93	41	0,75	34	0,75	16	0,70
A3	64	0,95	40	0,81	33	0,77	14	0,85
A4 iter=50	78	0,43	57	0,46	49	0,45	24	0,50
B1	81	0,52	58	0,60	48	0,71	21	0,81
B2	74	0,99	48	1,49	39	1,94	15	1,94
B3	74	0,96	48	1,31	40	1,72	16	1,64
B4 iter=50	59	1,98	33	2,06	27	2,16	10	3,00
Algorytmy o strukturze szeregowej								
A1B1	89	0,19	76	0,21	70	0,24	43	0,23
A1B2	88	0,20	75	0,22	68	0,25	42	0,23
A1B3	88	0,20	75	0,22	68	0,25	42	0,23
A2B1	87	0,30	72	0,31	66	0,36	37	0,32
A2B2	86	0,31	71	0,34	64	0,38	35	0,36
A2B3	86	0,31	71	0,34	64	0,38	36	0,35
A3B1	87	0,31	72	0,38	64	0,40	34	0,48

A3B2	86	0,33	70	0,40	63	0,42	33	0,51
A3B3	86	0,33	70	0,40	63	0,42	33	0,51
B1A1	90	0,17	76	0,21	69	0,25	43	0,24
B1A2	90	0,17	75	0,20	69	0,26	42	0,25
B1A3	90	0,17	75	0,20	69	0,26	42	0,24
B2A1	86	0,33	71	0,39	63	0,51	38	0,42
B2A2	86	0,35	71	0,40	62	0,51	37	0,44
B2A3	86	0,34	71	0,39	62	0,52	37	0,44
B3A1	86	0,32	72	0,36	64	0,45	37	0,39
B3A2	86	0,32	72	0,36	64	0,45	37	0,40
B3A3	86	0,32	72	0,35	64	0,44	37	0,41
Algorytmy o strukturze równoległej								
A1/B1	91	0,13	81	0,17	75	0,19	53	0,20

Porównując algorytmy złożone LO należy stwierdzić, że algorytm o strukturze równoległej A1/B1 jest znacznie lepszy od algorytmów o strukturze szeregowej i jest on najlepszy spośród wszystkich badanych algorytmów i to dla wszystkich wartości n. Średni błąd względny tego algorytmu jest mniejszy 0.25% i wynosi 0.13% dla n=20, zwiększa się nieznacznie wraz ze wzrostem n, osiągając poziom 0.2% dla n=100.

Analizując rezultaty badań testujących można stwierdzić, że kolejność występowania algorytmów bazowych w złożonych algorytmach szeregowych nie ma istotnego znaczenia, bowiem wyniki obliczeń są zbliżone. Poza tym można zauważyć, że algorytmy złożone (szeregowe) oparte na strategii 1 dają korzystniejsze rezultaty niż algorytmy zawierające inne strategie.

Dla porządku odnotujemy jeszcze, że wyniki obliczeniowe potwierdzają przewagę algorytmów złożonych nad bazowymi.

Ponadto, zaletami przedstawionych algorytmów LO jest brak parametrów dodatkowych, wymagających wyznaczenia (strojenia) przed rozpoczęciem zasadniczych obliczeń rozwiązujących dany przykład.

LITERATURA

1. Adrabiński A., Grabowski J., Wodecki M. : Algorytm rozwiązania zagadnienia kolejnościowego postaci $n|1|\sum w_i T_i$, Archiwum Automatyki i Telemechamiki., tom 33, z. 4, 1998, pp. 623-636.
2. Backer K.R. : Introduction to sequencing and scheduling, Wiley, New York, 1974.
3. Matsuo H., Suh C.J., Sullivan R.S.: A Controled Search Simulated Annealing Method for the single Machine Weighted Problem, Annals of Operation Research 21, 1989, pp. 85-108.
4. Potts C.N., Van Wassenhove L.N. : A Branch and Bound Algorithm for Total Weighted Tardiness Problem, Operation Research, 33, 1985, pp. 363-377.

5. Potts C.N., Van Wassenhove L.N. : Single Machine Tardiness Sequencing Heuristics, IIE Transactions, 23, 1991, pp. 346-354.
6. Rinnooy Kan A.H.G., Lageweg B.J., Lenstra J.K. : Minimizing Toatal Cost One-Machine Scheduling, Operation Research, 26 1975 pp. 908-927.
7. Shrag L., Baker K.R. : Dynamic Programming solution of Sequencing Problems with Precedence Constrains, Operations Research, 26, 1978, pp. 444-449.
8. Smith W.E. : Various Optimizers for Single-Stage Produktion, Naval Research Logist Quartely, 3, 1956, pp. 59-66.
9. Smutnicki C. : Optimization and Control in Just-Time Manufacturing System, Seria Monografie, Oficyna Wydawnicza PWr, Wrocław 1997.

Recenzent: Prof.dr hab.inż. Tadeusz Sawik

Abstract

This paper deals with heuristics for single machine total weighted tardiness problem. Consider n jobs $J_1,J_2,...,J_n$, to be processed without interruption on a single machine that can handle only one job at a time. Job J_i requires a processing time p_i, has a weight w_i, and has a due date d_i. For a given sequence of the jobs the completion time C_i and the tardiness $T_i=max(0,C_i-d_i)$ of job J_i can be computed. The objective is to find a processing order of the jobs that minimizes $\sum_{i=1}^{n} w_i T_i$. The problem belongs to the class of NP-hard problems what justifies searching for heuristics algorithms. In the paper, we propose several approximation algorithms. Finally, the computation results and discussion of the performance of algorithms are presented.