ZESZYTY NAUKOWE POLITECHNIKI ŚLASKIEJ 2006
Seria: AUTOMATYKA z. 143 Nr kol. 1726

Wojciech BOŻEJKO, Mieczysław WODECKI
Politechnika Wrocławska, Uniwersytet Wrocławski

SZEREGOWANIE ZADAŃ Z NAJWCZEŚNIEJSZYMI I NAJPÓŹNIEJSZYMI TERMINAMI ZAKOŃCZENIA

Streszczenie. W pracy przedstawiamy algorytm przybliżony oparty na metodzie przeszukiwania z tabu dla rozwiązywania problemu szeregowania na jednej maszynie zadań, z najwcześniejszymi i najpóźniejszymi terminami zakończenia. W procedurze przeglądania sąsiedztwa (ograniczonego przez eliminację „złych" rozwiazań) stosujemy, jako kryterium wyboru, górne ograniczenie wartości funkcji celu (rozwiązując problem „bez przestojów maszyny").

SCHEDULING WITH EARLINESS AND TARDINESS PENALTIES

Summary. In the paper we present an algorithm which is based on the tabu method to solving single machine scheduling problem with earliness and tardiness penalties. We apply an upper bound as the criterion in the neighborhood searching (solving "no idle" problem).

1. Problemy szeregowania z terminami zakończenia zadań

W systemach wytwarzania dokładnie na czas (*Just In Time*) koszty powoduje nie tylko zbyt późne (tardiness), ale także zbyt wczesne (earliness) wykonanie zadania. Tego typu zagadnienia (w skrócie E/T) są inspiracją do sformułowania wielu problemów szeregowania z funkcjami celu, w których występuje kara zarówno za spóźnienie, jak i przyśpieszenie. Niestety, poza nielicznymi wyjątkami, należą one do klasy problemów silnie NP-trudnych. Stąd w praktyce do ich rozwiązywania są stosowane wyłącznie algorytmy przybliżone. Najlepsze z nich oparte są na metodzie lokalnych przeszukiwań.

Niech $I = \{1, 2,..., n\}$ będzie zbiorem zadań do wykonywania na jednej maszynie (bez przerywania), która w dowolnej chwili wykonuje co najwyżej jedno zadanie. Przez p_i oznaczamy czas wykonywania, a przez e_i oraz d_i odpowiednio żądany *najwcześniejszy* i *najpóźniejszy* termin zakończenia wykonywania zadania $i \in I$. Jeżeli ustalona jest kolejność wykonywania zadań oraz C_i jest *terminem zakończenia* zadania i, to wielkość $E_i = \max\{0, e_i - C_i\}$ nazywamy *przyśpieszeniem*, a $T_i = \max\{0, C - d_i\}$ *spóźnieniem* zadania. Jeżeli $E_i = 0$ oraz $T_i = 0$, to zadanie jest nazywane *terminowym*, a w przeciwnym przypadku odpowiednio *przyśpieszonym* lub

spóźnionym. Wyrażenie $u_iE_i + w_iT_i$ jest *kosztem wykonania* zadania, gdzie u_i oraz w_i ($i \in I$) są nieujemnymi współczynnikami funkcji kosztu.

Problem minimalizacji sumy kosztów wykonania zadań nieterminowych na jednej maszynie (w skrócie ***JET***), polega na minimalizacji funkcji:

$$\sum_{i=1}^{n}(u_iE_i + w_iT_i).$$

W literaturze oznaczany jest przez $1||\Sigma(u_iE_i+w_iT_i)$ i należy do klasy problemów silnie NP-trudnych. Niech Φ_n będzie zbiorem wszystkich permutacji elementów z I. Dowolne rozwiązanie problemu *JET* może być reprezentowane przez parę (π, C), gdzie $\pi \in \Phi_n$ jest permutacją, a $C=(C_{\pi(1)}, C_{\pi(2)}, \ldots, C_{\pi(n)})$ n elementowym ciągiem, w którym $C_{\pi(i)} - p_{\pi(i)}$ jest terminem rozpoczęcia wykonywania $\pi(i) \in I$. Łatwo jest zauważyć, że w optymalnym rozwiązaniu mogą występować przestoje maszyny, tj. $C_{\pi(i+1)} - p_{\pi(i+1)} > C_{\pi(i)}$, $i \in I$. W pracy [4] przedstawiono algorytm wyznaczający, dla dowolnej permutacji zadań, optymalne momenty rozpoczęcia ich wykonywania (optymalne „rozsunięcie" zadań – *optimal timing*). Algorytmy dokładne dla problemu *JET* pozwalają na rozwiązywanie przykładów, w których liczba zadań nie jest większa niż 20. Stąd w praktyce stosuje się głównie algorytmy metaheurystyczne. Bogaty przegląd badań nad problemami *E/T* jest zamieszczony w monografii [3].

Jeżeli przyjmiemy dodatkowo, że zadania są wykonywane bez przestojów maszyny, to otrzymamy problem oznaczany w literaturze przez $1|no\ idle|\Sigma(u_iE_i+w_iT_i)$ (w skrócie $JET^{n}\text{-}^{i}$), należący także do klasy problemów NP-trudnych.

2. Własności problemu $JET^{n}\text{-}^{i}$

Dowolna permutacja, $\pi \in \Phi_n$ jest rozwiązaniem dopuszczalnym dla problemu $JET^{n}\text{-}^{i}$, bowiem termin zakończenia zadania $\pi(i) \in I$, $C_{\pi(i)} = \sum_{j=1}^{i} p_{\pi(j)}$. Zadanie $\pi(i)$ jest ***przyśpieszone***, jeżeli jego wykonywanie kończy się przed najwcześniejszym terminem ($C_{\pi(i)} < e_{\pi(i)}$), ***terminowe***, jeśli $e_{\pi(i)} \le C_{\pi(i)} \le d_{\pi(i)}$ oraz ***spóźnione***, jeśli wykonywanie kończy się za najpóźniejszym terminem, tj. $C_{\pi(i)} > d_{\pi(i)}$.

Permutację $\pi \in \Phi_n$ można rozbić na subpermutacje (ciągiem bezpośrednio po sobie następujących elementów zwanych ***blokami***)

$$\Omega = [B^1, B^2, \ldots, B^v],$$

z których każda jest ciągiem zadań spełniających ograniczenia:

1. $B^i = (\pi(a^i), \pi(a^{i+1}), \ldots, \pi(b^{i-1}), \pi(b^i)), \quad a^i = b^{i+1}, 1 \le i \le v, b^0 = 0, b^v = n.$
2. Każde zadanie $j \in B^i$ spełnia dokładnie jeden z warunków:

$$e_j > C_{\pi(b^i)}, \qquad \text{lub} \qquad \text{(C1)}$$

$$e_j \le C_{\pi(b^{i-1})} + p_j \wedge d_j \ge C_{\pi(b^i)}, \qquad \text{lub} \qquad \text{(C2)}$$

$$d_j < C_{\pi(b^{i-1})} + p_j, \qquad \text{(C3)}$$

3. B^i jest maksymalną subpermutacją w π spełniającą dokładnie jeden z warunków C1 lub C2 lub C3.

Jest oczywiste, że warunek C1 spełniają zadania przyśpieszone, C2 zadania terminowe, a C3 zadania spóźnione. Stąd wynika, że istnieją odpowiednio trzy typy bloków: *zadań przyśpieszonych* (*E*-bloki), *terminowych* (*T*-bloki) lub *spóźnionych* (*D*-bloki). Korzystając z powyższych warunków, każdą permutację $\pi \in \Phi_n$ można rozbić na bloki. Dla dowolnego bloku B rozbicia Ω permutacji $\pi \in \Phi_n$ niech

$$F_B^{n-i} = \sum_{\pi(j) \in B} (u_{\pi(j)} E_{\pi(j)} + w_{\pi(j)} T_{\pi(j)}), \qquad (1)$$

wówczas wartość funkcji celu dla permutacji π wynosi:

$$F^{n-i}(\pi) = \sum_{j=1}^{n} (u_{\pi(j)} E_{\pi(j)} + w_{\pi(j)} T_{\pi(j)}) = \sum_{B \in \Omega} F_B .$$

Niech δ będzie subpermutacją permutacji $\pi \in \Phi_n$. Zadania w δ występują zgodnie z regułą *WSPT* (Weighted Shortest Processing Time), jeżeli są uszeregowane według nierosnących wartości ilorazów w_j / p_j. Podobnie, elementy w δ występują zgodnie z regułą *WLPT* (Weighted Longest Processing Time), jeżeli są uszeregowane według nieniemalejących wartości ilorazów u_j / p_j.

Jeżeli B jest blokiem zadań spóźnionych (*D*-blokiem), w którym zadania występują zgodnie z regułą *WSPT,* to jest to optymalne uszeregowanie zadań tego bloku (funkcja (1) osiąga wartość minimalną – Smith [2]). Podobnie jeżeli B jest blokiem zadań przyśpieszonych (*E*-blokiem), to uszeregowanie elementów zgodnie z zasadą *WLPT* jest optymalne. Mówimy, że permutacja π jest ***uporządkowana***, ze względu na rozbicie Ω, jeżeli zadania w *D*-blokach występują zgodnie z regułą *WSPT,* a w *E*-blokach – zgodnie z regułą *WLPT.*

Obecnie przedstawimy twierdzenie (eliminacyjne własności bloków), które jest podstawą konstrukcji subotoczeń w algorytmach lokalnych przeszukiwań.

Twierdzenie 1. *Niech Ω będzie rozbiciem uporządkowanej permutacji $\pi \in \Phi_n$ na bloki. Jeżeli permutacja $\beta \in \Phi_n$ powstała z π przez zmianę kolejności elementów oraz*

$$F^{n-i}(\beta) < F^{n-i}(\pi),$$

to w permutacji β przynajmniej jeden element z pewnego bloku rozbicia Ω jest przed pierwszym lub za ostatnim elementem tego bloku. ■

Z powyższego twierdzenia wynika, że warunkiem koniecznym aby wygenerować z π permutację o mniejszej wartości funkcji celu, jest przestawienie przynajmniej jednego elementu przed pierwszy lub za ostatni element pewnego bloku.

3. Algorytm przeszukiwania z tabu

Algorytmy oparte na metodzie przeszukiwania z tabu (*tabu serach, TS*) są z powodzeniem stosowane do wyznaczania bardzo dobrych rozwiązań przybliżonych dla NP-trudnych problemów optymalizacji kombinatorycznej. Generalnie metoda ta polega na iteracyjnym polepszaniu bieżącego rozwiązania poprzez lokalne przeszukiwanie. Rozpoczyna się od pewnego rozwiązania startowego x^0. W każdej iteracji, dla bieżącego rozwiązania x^i, wyznacza się sąsiedztwo $N(x^i)$ – podzbiór zbioru rozwiązań dopuszczalnych. Sąsiedztwo jest generowane przez ruchy, tj. pewne

przekształcenia rozwiązania x^i. Następnie z sąsiedztwa wyznaczamy najlepszy element x^{i+1}, który przyjmuje się za bieżące rozwiązanie w następnej iteracji. Aby zapobiec generowaniu, w nowych iteracjach, rozwiązań niedawno rozpatrywanych (powstawaniu cykli), zapamiętuje się ruchy (ich atrybuty) na liście rozwiązań zakazanych, tzw. liście tabu. Poniżej opiszemy główne elementy metody.

Ruch i sąsiedztwo

W każdej iteracji algorytmu *TS* jest wyznaczane sąsiedztwo – podzbiór zbioru rozwiązań dopuszczalnych. Jest ono generowany przez ruchy – przekształcenia określone na zbiorze wszystkich permutacji.

Niech k i l ($k < l$) będą parą pozycji w permutacji:

$$\pi = (\pi(1), \pi(2), \ldots, \pi(k-1), \pi(k), \pi(k+1), \ldots, \pi(l-1), \pi(l), \pi(l+1), \ldots, \pi(n)).$$

Ruch typu wstaw (insert) oznaczany przez i_l^k (*i-ruch*), polega na przestawieniu zadania $\pi(k)$ z pozycji k na pozycję l. Generuje on permutację $i_l^k(\pi) = \pi_l^k \in \Phi_n$, gdzie:

$$\pi_l^k(i) = \begin{cases} \pi(k), & i = l, \\ \pi(i), & l < i < k, \\ \pi(i+1), & i = k, k+1, \ldots, l-1. \end{cases}$$

Ruch i_l^k, w przypadku, gdy $k > l$, definiujemy podobnie. Przez $M(\pi)$ oznaczamy zbiór wszystkich takich ruchów. Moc tego zbioru wynosi $n(n-1)$.

Niech

$$\Omega = [B^1, B^2, \ldots, B^v],$$

będzie rozbiciem uporządkowanej permutacji $\pi \in \Phi_n$ na bloki. Przez

$$W(\pi) = \{ i^k : \exists B^j, \pi(k), \pi(l) \in B^j, \ B^j \in \Omega \}.$$

oznaczamy zbiór *ruchów wewnętrznych* (*i*-ruchów, zmieniających kolejność elementów tylko wewnątrz bloków). Z twierdzenia 1. wynika, że dowolny ruch ze zbioru W) nie generuje permutacji o mniejszej wartości funkcji celu niż $F^{n-i}(\pi)$. Stąd, za *sąsiedztwo* permutacji $\in \Phi_n$ przyjmujemy zbiór:

$$N(\pi) = \{ i_l^k(\pi) \in \Phi_n : \ i_l^k \in M(\pi) \},$$

gdzie:

$$M(\pi) = \{ i_l^k : \ i_l^k \notin W(\pi) \ \ oraz \ \ l \neq k, \ \ l, k \in I \} \qquad (2)$$

jest zbiorem ruchów *zewnętrznych* (tj. ruchów, które dla problemu JET^{n-i} mogą przynieść poprawę wartości funkcji celu).

Lista ruchów zakazanych

Aby zapobiec powstawaniu cykli (powrotowi do tej samej permutacji po pewnej liczbie iteracji algorytmu), ustalone atrybuty każdego ruchu zapamiętuje się na liście ruchów zakazanych. Jest ona realizowana na zasadzie kolejki FIFO. Wykonując ruch i_j^r (tj. generując z π permutację π_j^r), na liście tabu zapisujemy atrybuty tego ruchu, trójkę: $(\pi(r), j, F^{n-i}(\pi_j^r))$. Załóżmy, że rozpatrujemy ruch $i_l^k \in M(\beta)$, generujący permutację β_l^k. Jeżeli na liście tabu jest trójka (r, j, Ψ) taka, że $\beta(k) = r$, $l = j$ oraz $F^{n-i}(\beta_l^k) \geq \Psi$, to ruch taki eliminujemy (usuwamy) ze zbioru $M(\beta)$.

Poniżej zamieszczamy algorytm rozwiązywania problemu *JET* oparty na metodzie przeszukiwania z tabu. W każdej iteracji jest generowane sąsiedztwo dla problemu JET^{n-i} (**bez przestojów maszyny**), z którego wybieramy najlepszy element β (kryterium wyboru jest funkcja celu F^{n-i}). Następnie, stosując algorytm optymalnego „rozsunięcia" (zamieszczony w [4]), obliczamy czasy zakończenia zadań w permutacji β oraz porównujemy otrzymane rozwiązanie z najlepszym do tej pory znalezionym (π^*, C^*). Permutacja β jest rozwiązaniem startowym w następnej iteracji algorytmu. Warunkiem zatrzymania jest z góry ustalona liczba iteracji.

W opisie algorytmu *TSL* jest listą tabu, zmienna *Maxiter* – liczbą iteracji algorytmu (kryterium zatrzymania), a (π^*, C^*) – najlepszym wyznaczonym rozwiązaniem.

Algorytm *TS_JET*

Input: rozwiązanie startowe dla problemu *JET*, (π^0, C^0);

Output: suboptymalne uszeregowanie (π^*, C^*);

Krok 0: {*Rozpoczęcie obliczeń*}

$(\pi^*, C^*) \leftarrow (\pi^0, C^0)$; $\pi \leftarrow \pi^0$; $TSL \leftarrow \emptyset$; iter:=0;

Krok 1: {*Sąsiedztwo*}

Wyznaczyć zbiór ruchów zewnętrznych $M(\pi)$, zgodnie z (2);

if $M(\pi) = \emptyset$ **then EXIT**;

Krok 2: {*Przeglądanie sąsiedztwa*}

Wyznaczyć ruch $m_l^k \in M(\pi)$ taki, że

$$F^{n-i}(m_l^k(\pi)) = \min\{F^{n-i}(i_s^t(\pi)) : i_s^t \in M(\pi)\};$$

if $F(\pi^*) > F(\pi_l^k)$ **then**

begin

Stosując algorytm z pracy [4] wykonać optymalne „rozsunięcie" permutacji π_l^k, tj. obliczyć optymalne terminy zakończenia zadań

$C=(C_{\pi(1)}, C_{\pi(2)}, \ldots, C_{\pi(n)})$ **and** $(\pi^*, C^*) \leftarrow (\pi_l^k, C)$

end;

Atrybuty ruchu m_l^k umieścić na liście *TSL*; *iter*:=*iter*+1;

Krok 3: {*Kryterium zatrzymania*}

if *iter* > *Maxiter* **then EXIT else goto Krok 1.**

Warunkiem zakończenia algorytmu jest ustalona liczba iteracji *Maxiter*.

Wyniki obliczeniowe

W literaturze brak jest danych referencyjnych dla problemu *JET*. Na stronie internetowej (http://www.ms.ic.ac.uk/jeb/orlib/schinfo.html) są zamieszczone przykłady dla problemu, w którym $e_i = d_i = const$, $i \in I$ (common due date). Uzupełniliśmy je, generując losowo (rozkład jednostajny) najpóźniejsze terminy zakończenia d_i – na przedziale $[P(1 - TF - RDD/2), P(1 - TF + RDD/2)]$, gdzie $P = \sum_{i=1}^{n} p_i$, a parametry $RDD, TF \in \{0.2, 0.4, 0.6, 0.8, 1.0\}$. Natomiast, najwcześniejszy termin zakończenia był losowany z przedziału $[0, d_i]$. Dla każdej z 25 par wartości parametrów *RDD* i *TF* generowano 5 przykładów. Za rozwiązanie startowe przyjęto (π^0, C^0), gdzie π^0 jest

permutacja losową, a C^0 wektorem optymalnych czasów zakończenia zadań (dla permutacji π^0) wyznaczonym przez algorytm z [4]. Otrzymane wartości rozwiązań porównano z wielkościami wyznaczonymi następująco. Dla każdego przykładu wyznaczano permutacje zadań, stosując algorytm symulowanego wyżarzania (dla problemu „bez przestojów maszyny") z [1]. Następnie obliczano optymalne czasy zakończenia zadań (dla wyznaczonej permutacji – algorytm z [4]) oraz, na ich podstawie, wartość funkcji celu. W tabeli 1 przedstawiono średnie i minimalne względne odchylenie (poprawę) dla każdej grupy przykładów. Wyniki algorytmu *TS_JET* są znacznie lepsze od zmodyfikowanych rozwiązań (referencyjnych) algorytmu symulowanego wyżarzania. Czas obliczeń jednego przykładu nie przekracza kilkunastu sekund (na komputerze z procesorem 1.8GHz).

Tabela 1

Średnie ($PRD_{śr}$) oraz minimalne (PRD_{min}) względne odchylenie

n	$Maxiter = n^2$		$Maxiter = 2n^2$	
	$PRD_{śr}$	PRD_{min}	$PRD_{śr}$	PRD_{min}
40	-8.32	-0.32	-12.16	-3.71
50	-11.87	-2.16	-14.26	-9.85
100	-18.56	-9.29	-23.87	-14.33
średnio	-12.91	-3.92	-16.76	-9.30

LITERATURA

1. Bożejko W., Wodecki M.: Task realizations optimization with earliness and tardiness penalties In distributed computation systems, LNCS, No. 3528, Springer Verlag 2005, p. 69-75.
2. Smith W.E.: Various Optimizers for Single-Stage Production. Naval Research Logist Quartely, 3, 1956, p. 59-66.
3. T'kindt V., Billaut J-C.: Multicriteria scheduling: theory, models and algorithms. Springer, Berlin 2002.
4. Wan G., Yen B.P.-C.: Tabu serach for single machine scheduling with distinct due windows and weighted earliness/tardiness penalties. European Journal of Operational Research, 142, 2002, p. 271-281.

Recenzent: Prof. dr hab. inż. Joanna Józefowska

Abstract

In Just In Time (JIT) systems costs are implied by earliness and tariness of the job's executing. In this paper we consider a single machine scheduling problem with earliness and tardiness penalties. It is represented by $1||\Sigma(u_iE_i+w_iT_i)$ in the literature and it belongs to the strongly NP-hard class. We present an algorithm which is based on the tabu search method. Applying eliminated properties of blocks, we determine restricted neighborhood. We apply an upper bound as the criterion in the neighborhood searching (solving "no idle" problem). Computational experiments show, that the algorithm works fast and the results of its work are very good.