AUTOMATYKA • 2007 • Tom 11 • Zeszyt 1–2

Wojciech Bożejko*, Józef Grabowski*, Mieczysław Wodecki**

Bloki w problemie przepływowym z minimalizacją sumy kosztów opóźnień

1. Wstęp

W rozpatrywanym problemie przepływowym z minimalizacją sumy kosztów opóźnień (*Total Weighted Tardiness Flow Shop Problem* – w skrócie *TWTFS*), każde z zadań należy kolejno wykonać na wszystkich maszynach, przy czym kolejność wykonywania zadań na każdej maszynie musi być taka sama. W literaturze problem ten jest oznaczany przez $F||\Sigma w_i T_i$. Należy on do klasy problemów silnie NP-trudnych (Lenstra, i in. [13]). W najlepszych algorytmach dla wielu problemów szeregowania, których rozwiązaniami dopuszczalnymi są permutacje, do przeglądu pośredniego i pomijania „gorszych" elementów stosuje się eliminacyjne własności bloków. Dla problemu *TWTFS* nie są obecnie znane żadne takie własności. W dalszej części przedstawiamy pewne uogólnienia klasycznego bloku. Niestety, są one mniej skuteczne niż na przykład te stosowane w algorytmach rozwiązywania problemu przepływowego z kryterium $C_{\max}$. Przeprowadzone eksperymenty obliczeniowe wykazały, że pomimo to, znacznie poprawiły one efektywność algorytmu opartego na metodzie przeszukiwania z zabronieniami. Szczególnie jest to widoczne przy rozwiązywaniu dużych przykładów.

Problemy szeregowania zadań z sumokosztowymi funkcjami celu mają już długą historię. Zdecydowana większość prac dotyczy szeregowania na jednej maszynie. Jedną z pierwszych prac dotyczących problemu przepływowego jest praca Hariri i Pottsa [9]. Przedstawiono w niej algorytm dla problemu $F||\Sigma w_i U$ oparty na metodzie podziału i ograniczeń, którym w rozsądnym czasie można rozwiązywać przykłady do 25 zadań wykonywanych na trzech maszynach. Dla tego problemu, bardzo dobre algorytmy genetyczne są zamieszczone w pracy Bartela i Billauta [3]. Niewiele jest prac poświęconych problemom wielomaszynowym z minimalizacją sumy kosztów opóźnień, tj. funkcji $\Sigma w_i T_i$ oraz jej szczególnemu przypadkowi ΣT_i. O trudności tych problemów świadczy najlepiej fakt, że dla najprostszego przypadku dwóch maszyn i funkcji celu ΣT_i, algorytmy dokładne (Pan i in. [17] oraz Schaller [19]) rozwiązują przykłady z liczbą zadań nie przekraczającą 20. Na temat znacznie trudniejszego problemu przepływowego z kryterium $\Sigma w_i T_i$ opublikowano

* Instytut Informatyki, Automatyki i Robotyki, Politechnika Wrocławska
** Instytut Informatyki, Uniwersytet Wrocławski

jeszcze mniej prac. Algorytmy dokładne przedstawili: Kim [12], Chung i in. [6] oraz Adrabiński, i in. [2] (szczególny przypadek dwumaszynowy jest rozpatrywany przez Wodeckiego [20]). Autorzy podają, że stosując te algorytmy, można rozwiązywać przykłady z co najwyżej kilkunastu zadaniami i kilkoma maszynami. W ostatnich latach ukazują się prace głównie poświęcone algorytmom aproksymacyjnym. Oparte na różnych metodach najnowsze heurystyki i metaheurystyki (głównie ewolucyjne i symulowanego wyżarzania) są przedstawione przez: Kima [11], Rajendrana i Zieglera [18], Hasija i Rajendrana [10], Adenso-Diasa [1] oraz Onwubolu i Davendra [16]. Przegląd metod, algorytmów i publikacji dotyczący wielomaszynowych problemom szeregowania zadań z minimalizacją sumy kosztów opóźnień jest zamieszczony w pracy Vallady i Ruiza [22] oraz Vallady i in. [23].

2. Definicje i oznaczenia

W problemie *TWTFS*, dany jest zbiór n zadań $J = \{1, 2, ..., n\}$, które należy wykonać za pomocą m maszyn ze zbioru $M = \{1, 2, ..., m\}$. Zadanie $i \in J$ należy kolejno wykonać na każdej maszynie, przy czym wykonywanie zadania na maszynie j (operacja $O_{i,j}$) może się rozpocząć dopiero po zakończeniu wykonywania tego zadania na maszynie $j-1 (j = 2, 3, ..., m)$ oraz kolejność wykonywania zadań na każdej z maszyn musi być taka sama. Maszyna w dowolnej chwili może wykonywać, co najwyżej jedno zadanie oraz, wykonywanie zadania nie może być przerwane. Dla zadania $i \in J$, niech $p_{i,j}$ będzie **czasem wykonywania** na maszynie $j \in M$, d_i **żądanym terminem zakończenia**, a w_i **wagą funkcji kosztów**. Dla ustalonej kolejności, zadań przez $C_{i,j}$ oznaczamy **termin zakończenia** zadania $i \in J$ na maszynie $j \in M$. Wówczas, $T_i = \max\{0, C_{i,m} - d_i\}$ jest **opóźnieniem**, a $f_i(C_{i,m}) = w_i T_i$ **kosztem opóźnienia** zadania. Należy wyznaczyć kolejność wykonywania zadań, minimalizującą sumę **kosztów opóźnień**, tj. sumę $\sum_{i=1}^{m} f_i(C_{i,m}) = \sum_{i=1}^{m} w_i T_i$.

Każde rozwiązanie problemu *TWTFS* może być reprezentowane przez permutację $\pi = (\pi(1), \pi(2), ..., \pi(n))$ zadań ze zbioru J. Przez Π oznaczamy zbiór wszystkich permutacji elementów z J. Dla dowolnej permutacji $\pi \in \Pi$ terminy rozpoczęcia zadań $S_{i,j}$ ($i \in J, j \in M$) na poszczególnych maszynach można wyznaczyć z następujących zależności rekurencyjnych:

$$S_{\pi(1),1} = 0, \quad S_{\pi(i),1} = S_{\pi(i-1),1} + p_{\pi(i-1),1}, \quad i = 2, 3, ..., n;$$

$$S_{\pi(1),j} = S_{\pi(1),j-1} + p_{\pi(1),j-1}, \quad j = 2, 3, ..., m;$$

$$S_{\pi(i),j} = \max\{S_{\pi(i),j-1} + p_{\pi(i),j-1}, S_{\pi(i-1),j} + p_{\pi(i-1),j}\}, \quad i = 2, 3, ..., n, \quad j = 2, 3, ..., m.$$

W tym przypadku terminy zakończenia zadań są równe:

$$C_{\pi(i),j} = S_{\pi(i),j} + p_{\pi(i),j}, \quad i = 1, 2, ..., n, \quad j = 1, 2, ..., m.$$

Przy tych oznaczeniach $C_{\pi(i),m}$ jest terminem zakończenia zadania $\pi(i)$ na ostatniej, m-tej maszynie, a opóźnienie $T_{\pi(i)} = \max\{0, C_{\pi(i),\,m} - d_{\pi(i)}\}$. Jeżeli $T_{\pi(i)} > 0$ to zadanie jest **spóźnione**, a w przeciwnym przypadku **terminowe**. Niech

$$WT(\pi) = \sum_{i=1}^{n} w_{\pi(i)} T_{\pi(i)},$$

będzie **kosztem permutacji** π. Problem *TWTFS* sprowadza się więc do wyznaczenia permutacji **optymalnej** (o minimalnym koszcie) w zborze Π.

Jeżeli $1 \le a \le b \le n$, to ciąg elementów

$$\pi^B = (\pi(a), \pi(a+1), ..., \pi(b)),$$

nazywamy ***subpermutacją*** (podciągiem) permutacji $\pi \in \Pi$.

Długość subpermutacji π^B, $L(\pi^B) = C_{\pi(b),\,m}$, a ***koszt*** $WT(\pi^B) = \sum_{i=a}^{b} w_{\pi(i)} T_{\pi(i)}$.

Niech $\Pi(\pi^B) \subset \Pi$ będzie zbiorem permutacji, które można wygenerować z π przez zmianę kolejności elementów w subpermutacji π^B. Wobec tego, jeżeli $\beta \in \Pi(\pi^B)$, to

$$\pi(i) = \beta(i), \quad i = 1, 2, ..., a-1, b+1, ..., n$$

oraz

$$\{\pi(a), \pi(a+1), ..., \pi(b)\} = \{\beta(a), \beta(a+1), ..., \beta(b)\}.$$

Jeżeli permutacja $\beta \in \Pi(\pi^B)$, to subpermutacja $\beta^B = (\beta(a), \beta(a+1), ..., \beta(b))$.

Dalej, wprowadzamy zmienne:

$$L^{mi}(\pi^B) \le \min\{C_{\delta(b),m} :\ \delta \in \Pi(\pi^B)\},$$

$$L^{mx}(\pi^B) \ge \max\{C_{\delta(b),m} :\ \delta^B \in \Pi(\pi)\},$$

$$WT^{mi}(\pi^B) \le \min\{\sum_{i=a}^{b} w_{\delta(i)} \cdot T_{\delta(i)} :\ \delta \in \Pi(\pi^B)\}$$

oraz

$$WT^{mx}(\pi^B) \ge \max\{\sum_{i=a}^{b} w_{\delta(i)} \cdot T_{\delta(i)} :\ \delta \in \Pi(\pi^B)\}.$$

Jeżeli permutacja $\gamma \in \Pi(\pi^B)$, to wartości $L^{mi}(\pi^B)$ i $L^{mx}(\pi^B)$ są dolnym i górnym ograniczeniem dla $L(\gamma^B)$, czyli $L(\gamma^B)$ należy do przedziału $[L^{mi}(\pi^B), L^{mx}(\pi^B)]$. Podobnie jest dla funkcji kosztu *WT*.

Oznaczmy przez F jedną z wyżej zdefiniowanych funkcji, tj. L (długość) lub WT (koszt). Wartości $F^{mi}(\pi^B)$ i $F^{mx}(\pi^B)$ będą stosowane do oceny, jak dalece subpermutacja π^B różni się od optymalnego uszeregowania zadań ze zbioru $\{\pi^B(a), \pi^B(a+1), ..., \pi^B(b)\}$ w permutacji π.

Subpermutacja π^B jest θ-***optymalna*** (ze względu na funkcję F), jeżeli

$$F(\pi^B) \in [F^{mi}(\pi^B),\ F^{mi}(\pi^B) + \theta \cdot (F^{mx}(\pi^B) - F^{mi}(\pi^B))] \tag{1}$$

Do rozwiązywania problemu przepływowego z minimalizacją sumy kosztów opóźnień przedstawimy algorytm oparty na metodzie przeszukiwania z zabronieniami. W konstrukcji otoczenia wykorzystamy własności eliminacyjne bloków definiowanych w oparciu o pojęcie θ-optymalności.

3. Algorytm przeszukiwania z zabronieniami

Główne idee metody przeszukiwania z zabronieniami zostały przedstawione przez Glovera w latach 90. XX w. wieku. Jest ona obecnie jedną z najlepszych i najczęściej stosowanych metod aproksymacyjnych. W procesie przeszukiwania otoczenia dopuszcza się w niej możliwość zwiększania wartości funkcji celu (przy wyznaczaniu nowego rozwiązania startowego), aby w ten sposób nie nastąpiło zbyt szybkie zakończenia działania algorytmu, po osiągnięciu pewnego minimum lokalnego. Takie ruchy „w górę" należy jednak kontrolować, ponieważ w przeciwnym przypadku, po osiągnięciu minimum lokalnego, mógłby szybko nastąpić do niego powrót. Dla uniknięcia wpadnięcia w cykl, wprowadza się tzw. listę tabu (rozwiązań lub ruchów zabronionych). Wykonując ruch (wyznaczający rozwiązanie startowe w następnej iteracji), zapamiętuje się jego atrybuty na liście. Generując następnie nowe otoczenie, pomijamy rozwiązania, których atrybuty znajdują się na liście, chyba że spełniają tzw. kryterium aspiracji (tj. są „wyjątkowo" korzystne). Najważniejszymi elementami metody są: ruch, otoczenie i lista ruchów zakazanych.

Niech $M(\pi)$ będzie zbiorem ruchów generujących pewne otoczenie permutacji $\pi \in \Pi$. Ruch $m \in M(\pi)$ nazywamy ***wewnętrznym***, dla subpermutacji π^B, jeżeli zmienia on kolejność elementów jedynie w π^B. Dokładniej, jeżeli ruch m generuje z π permutację $m(\pi) = \beta$, wówczas

$$\beta(i) = \pi(i),\ i = 1, 2, ..., a-1, b+1, ..., n$$

oraz

$$\{\beta(a), \beta(a+1), ..., \beta(b)\} = \{\pi(a), \pi(a+1), ..., \pi(b)\}.$$

Przez $M^{in}(\pi^B)$ oznaczamy zbiór **ruchów wewnętrznych** dla subpermutacji π^B.

Subpermutatcję π^B nazywamy ***blokiem***, jeżeli

$$\forall m \in M^{in}(\pi^B),\quad F(m(\pi)) \geq F(\pi).$$

Z definicji tej wynika, że każdy ruch wewnętrzny bloku generuje permutację, dla której wartość funkcji celu nie jest mniejsza niż $F(\pi)$. Wobec tego, w procedurach przeszukiwania otoczenia można pominąć elementy generowane przez ruchy wewnętrzne bloków. Dzięki temu zmniejsza się rozmiar otoczenia, a więc nastąpi przyśpieszenie procesu jego przeszukiwania. Bloki są od wielu lat z powodzeniem stosowane w algorytmach rozwiązywania problemów szeregowania głównie z kryterium $C_{\max}$.

Do najważniejszych zastosowań należą:

- Permutacyjny problem przepływowy z minimalizacją czasu zakończenia $F||C_{\max}$. Blokami są podzbiory zadań z drogi krytycznej wykonywane na tej samej maszynie (Nowicki i Smuticki [15], Grabowski i Wodecki [7]).
- Problem gniazdowy z minimalizacją czasu zakończenia $J||C_{\max}$. Blokami są podzbiory operacji z drogi krytycznej wykonywane na tej samej maszynie (Nowicki i Smutnicki [16], Grabowski i Wodecki [8]).
- Jednomaszynowy problem minimalizacji sumy kosztów opóźnień zadań $1||\mathrm{S}w_iT_i$ z dwoma typami bloków zawierających odpowiednio zadania terminowe lub spóźnione (Bożejko, Grabowski i Wodecki [5], Wodecki [21]).

Obecnie wprowadzimy pewne uogólnienie klasycznego pojęcia bloku. W „nowym" bloku może się tak zdarzyć, że pewna zmiana kolejności elementów wygeneruje jednak permutację o „niewiele" mniejszej wartości funkcji celu. Zdefiniujemy parametr (próg) pozwalający na określenie i kontrolowanie wielkości „niewiele". Jest on związany z wprowadzonym pojęciem θ-optymalności. Reasumując, własności eliminacyjne „nowych" bloków będziemy stosowali do przeglądu pośredniego elementów otoczenia mając świadomość, że przy okazji możemy odrzucić rozwiązania lepsze od bieżącego, na szczęście jedynie o „niewiele". Aby nie komplikować terminologii, dla tego uogólnienia zachowujemy nazwę bloku bowiem będziemy go stosowali tak jak „prawdziwy" blok. Przeprowadzone eksperymenty obliczeniowe wykazały, że pomimo tej wady zastosowanie bloków w algorytmie przeszukiwania z zabronieniami dla problemu *TWTFS* powoduje skrócenie czasu obliczeń i poprawę wartości wyznaczanych rozwiązań.

Niech π^B będzie pewną subpermutacją permutacji $\pi \in \Pi$. Załóżmy, że $\pi(a)$ jest pierwszym, a $\pi(b)$ ostatnim elementem w $\pi^B (1 \le a \le b \le n)$. Wartość funkcji koszt dla permutacji π

$$WT(\pi) = \sum_{i=1}^{n} w_{\pi(i)}T_{\pi(i)} = \sum_{i=1}^{a-1} w_{\pi(i)}T_{\pi(i)} + \sum_{i=a}^{b} w_{\pi(i)}T_{\pi(i)} + \sum_{i=b+1}^{n} w_{\pi(i)}T_{\pi(i)}.$$

Zmiana kolejności elementów w subpermutacji π^B może wygenerować rozwiązanie o mniejszym koszcie, jeżeli w wyniku tego:

- zmniejszy się koszt wykonywania elementów z π^B, tj. suma $\sum_{i=a}^{b} w_{\pi(i)}T_{\pi(i)}$;
- zmniejszą się momenty zakończenia wykonywania pewnych zadań z π^B, co w konsekwencji spowoduje zmniejszenie się także i wartość sumy $\sum_{i=b+1}^{n} w_{\pi(i)}T_{\pi(i)}$.

Elementy subpermutacji π^B będą tworzyły blok, jeżeli ich kolejność ze względu na koszt jak i długość jest „bliska" uszeregowaniu optymalnemu. Nie zawsze spełniają więc one w pełni ograniczenia: **dowolna zmiana kolejności elementów w bloku nie generuje permutacji o mniejszej wartości funkcji celu**. Może się bowiem tak zdarzyć, że dla każdego z tych kryteriów (długości i kosztu) są różne optymalne uszeregowania. Jeżeli nie uda się wyznaczyć jednego „dobrego" ze względu na oba kryteria, wówczas rozpatrywana subpermutacja nie tworzy bloku.

Będziemy wyróżniali dwa typy bloków:

1) *T*-bloki, zawierające zadania terminowe,
2) *D*-bloki, zawierające zadania spóźnione.

Zakładamy, że bloki tworzą subpermutacje zawierające co najmniej trzy zadania. Jak wykazały eksperymenty obliczeniowe, bloki poprawiają znacznie efektywność algorytmu opartego na metodzie przeszukiwania z zabronieniami z otoczeniami generowanymi przez ruchy typu wstaw (*insret*).

3.1. Bloki zadań terminowych

Subpermutacja π^T jest ***T-blokiem*** w permutacji π jeżeli:

(**a**) każde zadanie z π^T jest terminowe w permutacji π, czyli $\forall j\ (a \le j \le b), C_{\pi(j)} \le d_{\pi(j)}$;

(**b**) długość $L(\pi^T)$ jest θ-optymalna.

Ponieważ każde zadanie w *T*-bloku jest terminowe, więc koszt jego wykonywania wynosi 0. Wobec tego, π^T jest ze względu na koszt uszeregowaniem optymalnym. Punkt (**a**) powyższej definicji jest prosty do sprawdzenia. W dalszej części przedstawimy kryteria pozwalające na stwierdzenie, czy π^T jest θ-optymalna ze względu na długość uszeregowania, tj. spełnia punkt (**b**). Należy w tym celu wyznaczyć dolne $L^{mi}(\pi^T)$ i górne $L^{mx}(\pi^T)$ ograniczenia długości.

Wyznaczanie dolnego ograniczenia długości

Niech $p_j^{\min} = \min\{p_{i,j} : i \in \pi^T\}$ będzie najkrótszym czasem wykonywania pewnego zadania z subpermutacji π^T na maszynie $j \in M$. Wówczas, $C_{\pi(a-1),1} + \sum_{j=1}^{m} p_j^{\min}$, jest dolnym ograniczeniem terminu zakończenia zadania, które byłoby wykonywane jako *a*-te (pierwsze z π^T), a czasy jego wykonywania na poszczególnych maszynach byłyby równe: $p_1^{\min}, p_2^{\min}, ..., p_m^{\min}$. Oczywiście, jeżeli jako pierwsze weźmiemy dowolne zadanie z π^T, to termin jego zakończenia (na ostatniej maszynie) będzie większy. Za dolne ograniczenia długości subpermutacji przyjmujemy więc

$$L^{mi}(\pi^T) = C_{\pi(a-1),1} + \sum_{j=1}^{m} p_j^{\min} + \sum_{i=a}^{b} p_{\pi(i),m} - p_m^{\min} \tag{2}$$

Łatwo zauważyć, że $C_{\pi(a-1),1} + \sum_{j=1}^{m} p_j^{\min}$ jest dolnym ograniczeniem terminu zakończenia dowolnego zadania z π^T wykonywanego jako pierwsze spośród wszystkich zadań z tej subpermutacji. Druga suma jest czasem wykonywania, bezpośrednio jedno po drugim, zadań na ostatniej maszynie. Jest ona pomniejszone o $p_m^{\min}$, bowiem element ten już występuje w pierwszej sumie. Z powyższego wynika, że (2) jest dolnym ograniczeniem zakończenia wykonywania, w dowolnej kolejności, zadań z π^T. Złożoność obliczeniowa wyznaczania tego ograniczenia wynosi $O(\max\{n, m\})$.

Wyznaczanie górnego ograniczenia długości

Każdemu rozwiązaniu (permutacji zadań) problemu przepływowego z kryterium $C_{\max}$ można przyporządkować graf skierowany (zobacz np. [7]), w którym wierzchołkami są operacje. Wagą wierzchołka jest czas wykonywania odpowiadającej mu operacji. Łuki poziome reprezentują kolejność wykonywania zadań na każdej maszynie, a pionowe – kolejność wykonywania zadania przez maszyny (porządek technologiczny). Najdłuższa droga w tym grafie jest nazywana **drogą krytyczną**. Jej długość (suma wag wierzchołków) jest równa czasowi wykonywania zadań (w kolejności występowania) w π na maszynach, tj. jest równa $C_{\pi(n),\,m}$.

Niech $W(\pi)$ będzie zbiorem wierzchołków (operacji) drogi krytycznej w grafie odpowiadającym rozwiązaniu $\pi \in \Pi$.

Łatwo wykazać, że zbiór $W(\pi)$ spełnia następujące warunki:

– zwiera co najmniej jedną operację z każdej maszyny (**W1**)
– liczba jego elementów wynosi $n + m - 1$ (**W2**)

Są to pewne warunki konieczne, które musi spełnić dowolny podzbiór zbioru operacji, aby jego elementy tworzyły drogę krytyczną. Łatwo zauważyć, że nie każdy podzbiór spełniający (**W1**) i (**W2**) jest drogą krytyczną. Nie są to więc warunki wystarczające. Poniżej przedstawimy algorytm wyznaczania pewnego podzbioru zbioru operacji, spełniającego ograniczenia (**W1**) i (**W2**). Na jego podstawie obliczymy wartość górnego ograniczenia długości subpermutacji ze zbioru $L^{mx}(\pi^T)$.

Niech O^T będzie zbiorem wszystkich operacji z π^T.

Algorytm *UBT* {Wyznaczanie górnego ograniczenia długości subpermutacji}

Krok 0.

$A \leftarrow \varnothing;\ L^{mx}(\pi^T) := C_{\pi(a-1),1};$

Krok 1.

for j:=1 **to** m **do**

wyznaczyć najdłuższy czas wykonywania operacji na maszynie

$p_{u,j} = \max\{p_{i,j} :\ i = 1,2,...,n\}$ **and** $L^{mx}(\pi^T) := L^{mx}(\pi^T) + p_{u,j}$

and $A \leftarrow A \cup \{O_{u,j}\};$

Krok 2.

while $|A| < b - a + m$ **do**

wyznaczyć czas wykonywania najdłuższej operacji

$p^*_{u,v} = \max\{p_{i,j} :\ O_{i,j} \in O^T \setminus A\}$, a następnie $L^{mx}(\pi^T) := L^{mx}(\pi^T) + p^*_{u,v}$ **and**

$A \leftarrow A \cup \{O_{u,v}\};$

end.

Złożoność obliczeniowa algorytmu wynosi $O(n^2m)$.

W kroku 1 są sumowane najdłuższe czasy wykonywania operacji na każdej maszynie. W każdej iteracji kroku 2, do $L^{mx}(\pi^T)$ jest dodawany najdłuższy czas wykonywania operacji spośród wszystkich jeszcze nie wybranych. Jeżeli w pewnej iteracji (kroku 1 lub 2) czas wykonywania operacji $O_{i,j} \in O^T$ dodaje się do bieżącej wartości zmiennej $L^{mx}(\pi^T)$, to w następnych iteracjach operacja ta nie jest rozpatrywana. Należy ona bowiem już do zbioru A. Łatwo zauważyć, że po zakończeniu algorytmu zmienna $L^{mx}(\pi^T)$ jest sumą $(b - a + 1) + m - 1$ składników. Wobec tego zbiór A spełnia ograniczenia (**W1**) i (**W2**) dla zadań z π^T.

3.2. Bloki zadań spóźnionych

Wyznaczanie D-bloku, dla rozpatrywanego problemu jest znacznie trudniejsze od wyznaczania T-bloku. Generalnie jest to subpermutacja zawierająca zadania spóźnione, także po dowolnej zmianie ich kolejności. Ponadto, po lokalnej optymalizacji ze względu na koszt jest ona θ-optymalna.

Subpermutacja zadań

$$\pi^D = (\pi(a), \pi(a+1), ..., \pi(b)), \quad 1 \le a < b \le n,$$

jest *D-**blokiem*** w permutacji $\pi \in \Pi$, jeżeli:

(**a'**) zadanie $\pi(i), i = a, a+1, ..., b$ przestawione na pozycję pierwszą w π^D jest spóźnione,

(**b'**) po wykonaniu lokalnej optymalizacji koszt $WT(\pi^D)$ jest θ-optymalny.

Z ograniczenia (**a'**) wynika, że każde zadanie jest w D-bloku zawsze spóźnione. Lokalną optymalizację występującą w punkcie (**b'**) będziemy wykonywali podobnie, jak optymalizację w D-blokach dla jednomaszynowego problemu $1\|\Sigma w_i T_i$ (zobacz [5]). W przypadku problemu przepływowego wyznaczamy jedynie pewne uszeregowanie suboptymalne. Będzie ono podstawą do szacowania terminów zakończenia zadań. Aby stwierdzić, czy subpermutacja jest θ-optymalna, należy obliczyć dolne $WT^{mi}(\pi^D)$ i górne $WT^{mx}(\pi^D)$ ograniczenie funkcji kosztu.

Wyznaczanie wartości dolnego ograniczenia kosztu

Najpierw wyznaczamy pewne suboptymalne (ze względu na długość) uszeregowanie zadań z π^D. Dla każdego zadania obliczamy całkowity czas wykonywania $P_i = \sum_{j=1}^{m} p_{\pi(i),j}$, a następnie sortujemy zadania zgodnie z nierosnącymi wartościami ilorazów w_i/P_i, $i = a, a+1, ..., b$. Niech $\vec{\pi}^D$ będzie w ten sposób wyznaczonym ciągiem elementów z subpermutacji π^D. Usuwamy przestoje maszyn (dosuwając operacje do lewej strony) rezygnując z ograniczenia: **wykonywanie zadania może się rozpocząć, po jego zakończeniu na poprzedniej maszynie** oraz obliczamy termin zakończenia C_j^i każdego zadania i z $\vec{\pi}^D$ na maszynie $j \in M$. Na podstawie tych terminów wyznaczamy koszt wykonywania zadań z $\vec{\pi}^D$, czyli dolne ograniczenia kosztu zadań z $\Pi(\pi^D)$.

Algorytm *LBD* {Dolne ograniczenie kosztu zadań ze zbioru $\Pi(\pi^D)$}

Wyznaczyć suboptymalne uszeregowanie zadań $\bar{\pi}^D$:

for j:=1 **to** m **do** $C_j^{\bar{\pi}^D}(a) := C_{\bar{\pi}(a),j}$;

for i:=a+1 **to** b **do** $C_j^{\bar{\pi}^D}(i) := C_j^{\bar{\pi}^D}(i-1) + p_{\bar{\pi}(i),j}$;

$$WT^{mi}(\pi^D) = \sum_{i=1}^{a-1} w_{\pi(i)} T_{\pi(i)} + \sum_{i=b+1}^{n} w_{\pi(i)} T_{\pi(i)};$$

for i:=a **to** b **do**

$$WT^{mi}(\pi^D) := WT^{mi}(\pi^D) + w_{\bar{\pi}^D(i)} \cdot \max\{(C_m^{\bar{\pi}^D}(i) - d_{\bar{\pi}^D(i)}), 0\}$$

end.

Złożoność obliczeniowa algorytmu wyznaczania $WT^{mi}(\pi^D)$ wynosi $O(n \cdot m)$.

Wyznaczanie wartości górnego ograniczenia kosztu

Przy wyznaczaniu wartości górnego graniczenia kosztów permutacji ze zbioru $\Pi(\pi^D)$ będziemy korzystali z pewnych idei zastosowanych przy wyznaczaniu podobnego ograniczenia dla *T*-bloków oraz dolnego ograniczenia dla *D*-bloków. Algorytm wyznaczania $WT^{mx}(\pi^D)$ składa się z czterech kroków. W trzech pierwszych obliczamy przybliżenia terminów zakończenia zadań, a w ostatnim koszt ich wykonania.

Algorytm *UBW* {Górne ograniczenie kosztu permutacji zadań z π^D}

Krok 1.

stosując algorytm *UTB* (zobacz: *Bloki zadań terminowych*), obliczyć $L^{mx}(\pi^D)$ termin zakończenia ostatniego zadania na maszynie m;

Krok 2.

wyznaczyć „najgorszą„ permutację β zadań ze zbioru $\{\pi(a), \pi(a+1), ..., \pi(b)\}$, tj. taką, że

$$w_{\beta(1)} / p_{\beta(1),m} \le w_{\beta(2)} / p_{\beta(2),m} \le ... \le w_{\beta(b-a+1)} / p_{\beta(b-a+1),m};$$

Krok 3.

$$C_m^{\beta(1)} := L^{mx}(\pi^D); \quad WT^{mx}(\pi^D) := 0;$$

for j:=b–1 **downto** a **do** $C_m^{\beta(j)} := C_m^{\beta(j+1)} - p_{\beta(j+1),m}$;

Krok 4.

for j:=a **to** b **do** $WT^{mx}(\pi^D) := WT^{mx}(\pi^D) + w_{\beta(j)} \cdot \max\{(C_m^{\beta(j)} - d_{\beta(j)}), 0\}$

end.

Złożoność obliczeniowa algorytmu wynosi $O(\max\{n \ln n, m\})$.

3.3. Wyznaczanie bloków w permutacji

W dowolnej permutacji $\pi \in \Pi$ bloki wyznaczamy podobnie jak dla problemu $1||\Sigma wT_i$. Dokładnie jest to opisane w pracy [5]. Ogólna idea jest następująca:

Rozpatrujemy kolejno zadania $\pi(1), \pi(2), ..., \pi(n)$.

Przypadek 1. Załóżmy, że $\pi(1)$ jest w permutacji π terminowe. Wówczas jest ono kandydatem na pierwszy element T-bloku. Jeżeli zadania $\pi(2)$ i $\pi(3)$ są także terminowe, to sprawdzamy, czy subpermutacja $\pi^T = (\pi(1), \pi(2), \pi(3))$ spełnia ograniczenie (**b**) w definicji T-bloku. Jeżeli tak, to mamy pierwszy 3-elementowy blok, który staramy się powiększyć. Jeżeli dodając zadanie, nie można już powiększyć bieżącego bloku, to zadanie to jest kandydatem na pierwsze zadanie następnego bloku. Z kolei, gdy zadanie $\pi(2)$ lub $\pi(3)$ nie jest terminowe lub zadania terminowe $\pi(1)$, $\pi(2)$, $\pi(3)$ nie spełnia ograniczenie (**b**) w definicji T-bloku, wówczas konstrukcję nowego bloku rozpoczynamy od zadania $\pi(2)$. Jest ono kandydatem na pierwsze zadanie w bloku.

Przypadek 2. Załóżmy, że zadanie $\pi(1)$ jest w permutacji π spóźnione. W tym przypadku jest ono kandydatem na pierwsze zadanie w D-bloku. Dalej, postępujemy jak w Przypadku 1, sprawdzając oczywiście odpowiednie dla D-bloku warunki (**a**') i (**b**').

Postępując jak wyżej, wyznaczamy (jeżeli istnieją) bloki w permutacji $\pi \in \Pi$.

Złożoność obliczeniowa algorytmu wyznaczania bloków w permutacji wynosi $O(n^2m)$. Istnienie bloków oraz liczba elementów w blokach zależy od parametru θ ($0 < \theta < 1$).

4. Eksperymenty obliczeniowe

Do rozwiązywania problemu przepływowego z minimalizacją sumy kosztów opóźnień zaadoptowano algorytm *RMMTS* (dla problemu jednomaszynowego) zamieszczony w pracy [5]. W skrócie będziemy go oznaczać przez *RMMTSF*. Dane testowe wygenerowano losowo. Tak jak dla problemu jednomaszynowego $1||\Sigma w_iT_i$ wyznaczono czasy wykonywania operacji $p_{i,j}$($i \in J$, $j \in M$) oraz wagi funkcji kary w_i, $i \in J$. Niech $P_i = \sum_{j=1}^{m} p_{i.j}$ będzie całkowitym czasem wykonywania zadania $i \in J$ na maszynach. Żądany termin zakończenia $d_j = P_j \cdot (1+3\mu)$, $i \in J$, przy czym parametr μ jest realizacją zmiennej losowej o rozkładzie jednostajnym na odcinku [0, 1]. Generator ten jest opisany w pracy [12]. Podobny sposób generowania danych dla problemu przepływowego jest stosowany między innymi w pracy Hasija i Rajendrana [10]. Dane testowe tworzą 8 grup o rozmiarach nxm: 20×5, 20×10, 50×5, 50×10, 50×20, 100×5, 100×10 i 100×20. Dla każdego rozmiaru wyznaczono 25 przykładów, w sumie więc dane zawierają 200 przykładów. Algorytm *RMMTS* został zaprogramowany w języku C++. Eksperymenty obliczeniowe wykonano na mikrokomputerze IBM z zegarem 1,8 GHz. Do wyznaczania rozwiązania startowego zastosowano algorytm *NEH* [14]. W literaturze brak jest jakichkolwiek danych i wyników porównawczych dla rozpatrywanego problemu. Po wykonaniu wstępnych obliczeń przyjęto $\theta = 0{,}2$ oraz długość listy tabu 11. Zasadnicze testy obliczeniowe wykonano także dla wersji algorytmu *RMMTSF* bez bloków, którą oznaczamy przez $RMMTSF_{bb}$. Każdy z algorytmów wykonywał 2000 iteracji. Otrzymane wyniki porównano z rozwiązaniami algorytmu *NEH*. Średnie błędy względne δ_{aprd} oraz średnie czasy obliczeń t_{aprd} są przedstawione w tabeli 1.

Tablela 1
Średni błąd względny δ_{aprd} oraz czas obliczeń t_{aprd}

$n \times m$	*RMMTSF*		$RMMTSF_{bb}$	
	δ_{apred}	t_{aprd}	δ_{apred}	t_{aprd}
20×5	–3,48	0,9	–3,24	0,8
20×10	–3,82	1,7	–3,87	2,1
50×5	–5,28	2,1	–4,12	2,2
50×10	–5,71	3,1	–5,83	3,9
50×20	–6,12	3,8	–5,34	4,6
100×5	–8,31	8,4	–8,29	9,3
100×10	–8,76	13,7	–7,64	18,6
100×20	–8,12	24,3	–8,76	31,5
Średnio	**6,20**	**7,25**	**4,55**	**9,13**

Na podstawie otrzymanych wyników można stwierdzić, że zastosowanie bloków dało widoczną poprawę. Algorytm z blokami wyznacza lepsze rozwiązania w znacznie krótszym czasie. Średnia poprawa rozwiązania startowego (wyznaczonego przez *NEH*) wynosi 6,20%, podczas gdy dla algorytmu bez bloków wynosi ona 4,55%. Ponadto średni czas działania tego algorytmu jest o około 22% mniejszy. Średnie czasy obliczania jednego przykłady wynoszą odpowiednio 7,25 oraz 9,13 sekundy. Skrócenie czasu obliczeń można uzasadnić następująco. W algorytmie bez bloków przeglądane jest pełne otoczenie. Jego liczba elementów wynosi $O(n^2)$, a czas obliczenia wartości funkcji celu, dla jednego elementu, jest rzędu $O(n \cdot m)$. Z kolei procedura wyznaczania bloków w permutacji ma złożoność obliczeniową $O(n^2 m)$. Eksperymenty obliczeniowe wykazały ponadto, że subotoczenie generowane z wykorzystaniem własności eliminacyjnych bloków jest o około 35% mniejsze od pełnego otoczenia. Tak więc eliminacja przez przegląd pośredni wielu niepersperktywicznych elementów otoczenia daje widoczną poprawę wartości wyznaczanych rozwiązań oraz skraca czas obliczeń

5. Podsumowanie

W pracy przedstawiono permutacyjny problem przepływowy z minimalizacją sumy kosztów opóźnień. Jest on powszechnie uważany za znacznie trudniejszy od tego problemu z kryterium $C_{\max}$. Do jego rozwiązania zastosowano algorytm oparty na metodzie przeszukiwania z zabronieniami. Osłabiając ograniczenia klasycznego bloku, otrzymaliśmy pewne jego uogólnienie. Własności eliminacyjne „nowych" bloków wykorzystaliśmy w procedurze wyznaczania subotoczeń (generowanych przez ruchy typu wstaw). Przeprowadzone eksperymenty obliczeniowe wykazały, że stosowanie bloków poprawia wartości wyznaczanych rozwiązań oraz skraca czas obliczeń.

Literatura

[1] Adenso-Dias B.: *Restricted neighborhood in the tabu search for the flowshop problem*. EJOR, 1992, 62, 27–37

[2] Adrabiński A.: Grabowski J., Wodecki M.: *Permutacyjne zagadnienie taśmowe postaci* $F||Sw_iT_i$. Archiwum Automatyki i Robotyki, 1991, Tom XXXVI, Zeszyt 3–4, 459–47

[3] Bertel S., Billaut J.C.: *A genetic algorithm for an industrial multiprocessor flowshop scheduling problem with recirculation*. EJOR, 2004, 159, 651–662

[4] Bożejko W., Wodecki M.: *Solving Flow Shop Problem by Parallel Simulated Annealing*. 2002, LNCS 2328, Springer-Verlag, 236–244

[5] Bożejko W., Grabowski J., Wodecki M.: *Block approach-tabu search algorithm for single machine total weighted tardiness problem*. 2006, Computers & Industrial Engineering, 50, 1/2, 1–14

[6] Chung C., Flynn J., Kirca O.: *A branch and bound algorithm to minimize the total tardiness for n-machine permutation flowshop problem*s. EJOR, 2006,174(1), 1–10

[7] Grabowski J., Wodecki M.: *A very fast tabu search algorithm for the permutation flow shop problem with makespan criterion*. Computers and Operations Research, 2004, 31, 1891–1909

[8] Grabowski J., Wodecki M.: *A very fast tabu search algorithm for the job shop problem*. in: Rego C., Alidaee B., editors. Adaptive memory and evolution; tabu search and scatter search, Dordrecht, Kluwer Academic Publishers, 2005, 117–144

[9] Hariri A.M.A., Potts C.N.: *Branch and Bound Algorithm to Minimize the Number of Late Jobs in a Permutation Flowshop*. EJOR, 1989, 38, 228–237

[10] Hasija S., Rajendran C.: *Scheduling in flowshop to minimize total tardiness of jobs*. International Journal of Production Research, 2004, 42(11), 2289–2301

[11] Kim Y.D.: *Heuristics for flowshop scheduling problemsw minimizing mean tardiness*. Journal of the Operational Research Society, 1993, 44, 19–28

[12] Kim Y.D.: *Minimizingtotal tardiness in permutation flowshop*. EJOR, 1995, 85, 541–555

[13] Lenstra J.K., Rinnoy Kan A.H.G., Brucker P.: *Complexity of machine scheduling problems*. Annals of Discrete Mathematics, 1977, 1, 343–362

[14] Navaz M., Enscore E.E., Ham I.: *A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem*. OMEGA, 1983, 11(1), 91–95

[15] Nowicki E., Smutnicki C.: *A Fast tabu serach algorithm for the job shop problem*. Management Scence, 1996, 42, 797–813

[16] Onwubolu G., Davendra D.: *Scheduling flow shop using differential evolution algorithm*. EJOR, 2006, 171, 674–692

[17] Pan J., Chen J., Chao C.: *Minimizng tardiness ina two-machine flow-shop*. Computers & Operations Research, 2002, 29, 869–885

[18] Rajendran C., Ziegler H.: *Scheduling to minimize the sum of weighted flowtime and weighted tardiness of jobs in a flowshop with sequence-dependent setup times*. EJOR, 2003, 147(3), 513–522

[19] Schaller J.: *Note on minimizing total tardiness in a two-machine flow-shop*. Computers & Operations Research, 30(5), 2005, 3273–3281

[20] Wodecki M.: *Dwumaszynowy problem przepływowy z minimalizacją kosztów opóźnień*. Pod red. R. Knosali: Komputerowo Zintegrowane Zarządzanie, WNT, Warszawa, 1999, 471–480

[21] Wodecki M.: *A branch-and-bound parallel algorithm for single-machine total weighted tardiness problem.* Advanced manufacturing technology, 2007 (accepted)

[22] Valada E., Ruiz R.: *New genetic algorithms with path relinking for the minimum tardiness permutation flowshop problem*, 2006 (submitted)

[23] Valada E., Ruiz R., Millea G.: *Minimizing total tardiness in the m-machine flowshop problem: a review and evaluotion heuristics and me-taheuristics*. Computers & Operations Research, 2006 (in press)