PROBLEM PRZEPŁYWOWY Z PRZEZBROJENIAMI ORAZ CIĄGŁĄ PRACĄ MASZYN

Wojciech BOŻEJKO, Radosław IDZIKOWSKI, Mieczysław WODECKI

Streszczenie: W pracy rozpatrujemy problem przepływowy z przezbrojeniami maszyn pomiędzy kolejno wykonywanymi operacjami oraz ciągłą pracą maszyn (ang. *no idle*). Przyjęte ograniczenia, dotyczące klasycznego problemu przepływowego, są fundamentalne przy budowie modeli do harmonogramowania przedsięwzięć budowlanych realizowanych w systemie potokowym. Przedstawiamy model matematyczny oraz grafowy zagadnienia oraz dowodzimy pewnych jego własności, które zastosowano w konstrukcji algorytmu heurystycznego opartego na metodzie przeszukiwania z tabu

Słowa kluczowe: szeregowanie zadań, przezbrojenia, metaheurystyka

1. Wprowadzenie

W procesie planowania obiektów budowlanych pojawia się problem harmonizacji robót z uwzględnieniem szeregu ograniczeń technologicznych i organizacyjnych. Jednym z najistotniejszych jest konieczność zapewnienia ciągłości wykonywania pewnych prac. W przypadku znacznego rozproszenia realizowanych obiektów koniecznym jest także uwzględnienie czasów przemieszczania sprzętu pracowników oraz ich narzędzi. Jeżeli przedsięwzięcie jest realizowana w systemie potokowym Bożejko i in. [1], [2], to jego odpowiednikiem w przemyśle jest system produkcji przepływowej (ang. *flow*). Już szczególny przypadek rozpatrywanego problemu, tj. klasyczny problem przepływowy ($F \,||\, C_{\max}$) należy do klasy najtrudniejszych, silnie *NP*-trudnych, problemów optymalizacji kombinatorycznej. Ogranicza to zakres stosowania algorytmów dokładnych do instancji o niewielkich rozmiarach. Z tego powodu, do wyznaczania satysfakcjonujących rozwiązań rozpatrywanego problemu, stosujemy algorytm metaheurystyczny oparty na metodzie przeszukiwania z tabu.

2. Sformułowanie problemu

Problem przepływowy z pracą ciągłą pewnych maszyn oraz czasami przezbrojeń maszyn pomiędzy operacjami można sformułować następująco:
Problem: Zadania ze zbioru

$$J = \{J_1, J_2 \ldots, J_n\},$$

należy wykonać na maszynach ze zbioru

$$M = \{M_1, M_2 \ldots, M_m\}.$$

Każdy zadanie $J_i \in \mathsf{J}$ jest ciągiem m operacji

$$J_i = [O_{i,1}, O_{i,2}, \ldots, O_{i,m}],$$

przy czym operacja $O_{i,j}$ $(i=1,2,\ldots,n,\ j=1,2,\ldots,m)$ z zadania J_i jest wykonywana przez maszynę M_j w czasie $p_{i,j}$. Niech *O* będzie zbiorem wszystkich operacji.

Po zakończeniu pewnej, a przed rozpoczęciem następnej operacji należy dokonać przezbrojenie maszyny. Niech $s_{i,j}^{k}$ $(k \in M, i \neq j\ i, j \in \mathrm{J})$ będzie czasem przezbrojenia k - tej maszyny pomiędzy operacjami $O_{k,i}$ oraz $O_{k,j}$. Zakładamy, że każda maszyna ze zbioru M^{ni} $(M^{ni} \subseteq M)$ pracuje w sposób ciągły, tj. bezpośrednio po wykonaniu dowolnej operacji następuje przezbrojenie maszyny i natychmiast rozpoczęcie następnej operacji. Operacje zadania $J_i \in J$ należy wykonać w zadany *porządku technologicznym*, tzn. dowolna operacja $O_{i,j}$ ma być wykonywana po zakończeniu $O_{i,j-1}$, a przed rozpoczęciem $O_{i,j+1}$ $(2 \le j \le m-1)$. Muszą być przy tym spełnione następujące ograniczenia:

(a) każda operacja może być wykonywana tylko przez jedną, określoną przez porządek technologiczny, maszynę,
(b) żadna maszyna nie może wykonywać jednocześnie więcej niż jedną operację,
(c) w ramach każdego zadania musi być zachowany porządek technologiczny,
(d) wykonywanie żadnej operacji nie może być przerwane przed jej zakończeniem,
(e) wyróżnione maszyny muszą pracować w systemie ciągłym.

Operacje każdego zadania są wykonywane w takiej samej kolejności, wobec tego każde rozwiązanie może być reprezentowane przez permutację zadań. Oznaczmy przez Φ zbiór wszystkich takich permutacji ($|\Phi| = n!$). Rozpatrywany w pracy problem sprowadza się do wyznaczenia permutacji zadań (tj. momentów rozpoczęcia wykonywania poszczególnych operacji) spełniających ograniczenia (a)-(e), aby moment zakończenia wszystkich zadań był minimalny. W skrócie problem ten będziemy oznaczali przez **NSFS**.

2.1. Model matematyczny

Jeżeli zadania są wykonywane w kolejności $\pi \in \Phi$ oraz $C_{\pi(i),j}$ jest momentem zakończenia operacji $O_{\pi(i),j}$, to wyznaczenie momentu zakończenia wykonywania wszystkich zadań sprowadza się do wyznaczenia:

$$C_{\max}(\pi) = C_{m,\pi(n)} \tag{1}$$

przy ograniczeniach:

$$C_{i,\pi(j)} + p_{i+1,\pi(j)} \le C_{i+1,\pi(j)},\ i=1,...,m-1,\ j=1,...,n, \tag{2}$$

$$C_{i,\pi(j)} + p_{i,\pi(j+1)} + s^{i}_{\pi(j),\pi(j+1)} \le C_{i,\pi(j+1)},\ i=1,...,m,\ j=1,...,n-1, \tag{3}$$

$$C_{i,\pi(j)} = C_{i,\pi(j-1)} + p_{i,\pi(j)} + s^{i}_{\pi(j-1),\pi(j)},\ j=2,3,...,n, i \in M^{ni}, \tag{4}$$

W tym przypadku momenty rozpoczęcia operacji

$$S_{i,\pi(j)} = C_{i,\pi(j)} - p_{i,\pi(j)}, i=1,2,\ldots,m,\ j=1,2,\ldots,n. \tag{5}$$

Bez straty ogólności możemy przyjąć, że termin rozpoczęcia wykonywania pierwszej operacji przez pierwszą maszynę $S_{1,\pi(1)} = 0$.

Można łatwo sprawdzić, że określone przez (1)-(4) momenty rozpoczęcia i zakończenia operacji spełniają ograniczenia (a)-(e), więc są rozwiązaniem dopuszczalnym problemu **NSFS**. Wobec tego, rozwiązanie rozpatrywanego w pracy problemu sprowadza się do wyznaczenia permutacji optymalnej $\pi^* \in \Phi$ takiej, że

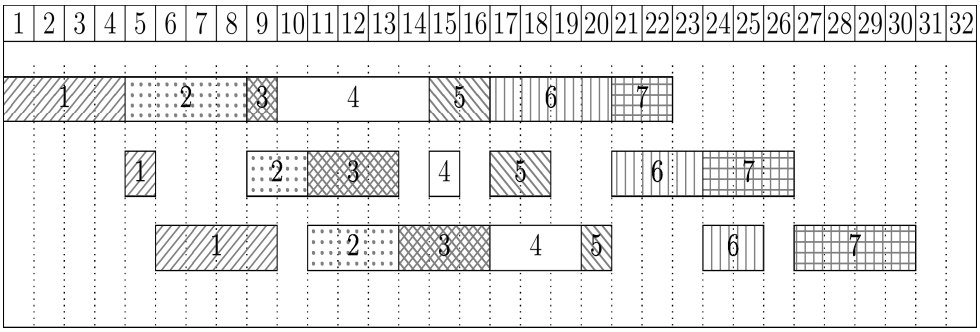
$$C_{\max}(\pi^*) = \min\{C_{\max}(\pi) : \pi \in \Phi\}.$$

Rozpatrujemy przykład problemu szeregowania siedmiu zadań $(n = 7)$ na trzech maszynach $(m = 3)$. Czasy wykonywania zadań na poszczególnych maszynach (operacji) są zamieszczone w tabeli 1.

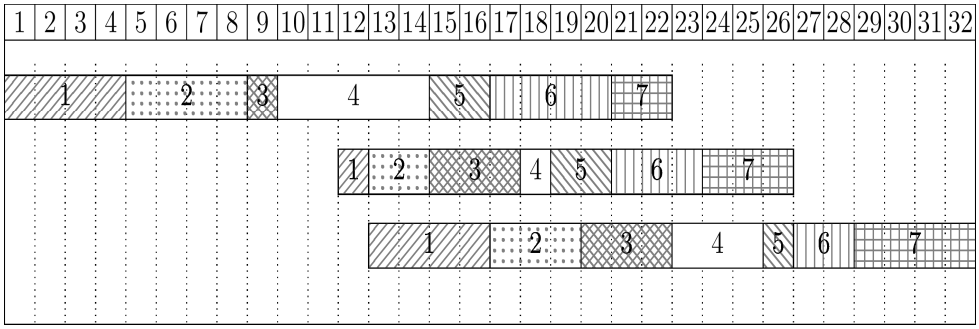
Tabela 1. Czasy wykonywania zadań na maszynach.

Zadanie	1	2	3	4	5	6	7
Maszyna 1	4	4	1	5	2	4	2
Maszyna 2	1	2	3	1	5	3	3
Maszyna 3	4	3	3	3	1	2	4

Rozpatrujemy uszeregowanie zadań, permutację naturalną $\pi = (1,2,3,4,5,6,7)$. Harmonogram Gantta, dla klasycznego problemu szreregowania, tj. harmonogramu spełniającego ograniczenia (2)-(4), jest przedstawiony na rysunku 1. Termin ukończenia wszystkich zadań $C_{\max}(\pi) = 30$. Na rysunku 2 przedstawiono diagram Gantta dla problemu z ciągłą pracą drugiej maszyny (dodatkowo z ograniczeniem (5)), tj. $M^{ni} = \{Maszyna\ 2\}$. W tm przypadku, termin ukończenia wszystkich zadań $C_{\max}(\pi) = 32$.



Rys. 1. Klasyczny problem przepływowy



Rys. 2. Problem przepływowy z ciągłą pracą maszyn

Łatwo udowodnić, że wartość optymalnego rozwiązania problemu przepływowego jest dolnym ograniczenie wartości rozwiązania problemu **NSFS**.

3. Metoda rozwiązania

Rozpatrywany w pracy problem wyznaczenia kolejności wykonywania zadań jest *NP-trudny*. Do jego rozwiązania będziemy stosowali algorytm heurystyczny oparty na na metodzie przeszukiwania z tabu (Glover [6], [7]), w skrócie TS. Jest to obecnie jedna z najbardziej efektywnych metod konstruowania algorytmów przybliżonych, a przy tym deterministyczna. Zasadniczym elementem tej metody jest otoczenie - sposób jego generowania oraz przeszukiwania ma decydujący wpływ na czas obliczeń oraz jakość wyznaczanych rozwiązań. Aby zmniejszyć czas wykonywania pojedynczej iteracji, będziemy korzystali z subotoczeń generowanych z wykorzystaniem tzw. „własności eliminacyjnych bloków" stosowanych w najlepszych algorytmach rozwiązywania problemu przepływowego, opublikowanych przez Nowickiego i Smutnickiego [10] oraz Grabowskiego i Wodeckiego [8].

3.1. Otoczenia

Zarówno w klasycznych już dziś algorytmach przybliżonych rozwiązywania problemów optymalizacji dyskretnej (np. przeszukiwanie z tabu, symulowane wyżarzanie, algorytmy memetyczne, itd.), jak i obecnie coraz bardziej popularnych algorytmach bazujących na metodach sztucznej inteligencji (np. sieci neuronowe, algorytmy stadne, immunologiczne, mrówkowe, itd.) stosuje się przeszukiwanie otoczeń. Jego celem jest wyznaczenie lokalnego minimum z pewnego podzbioru przestrzeni rozwiązań. W przypadku, gdy rozwiązania dopuszczalne problemu optymalizacyjnego są reprezentowane przez permutacje, do generowania otoczeń są stosowane ruchy - funkcje zamieniające pozycjami elementy w permutacji. Do najczęściej stosowanych należą ruchy typu zamień (ang. *swap*) oraz wstaw (ang. *insert*), a także ich złożenia (tzw. multiruchy, Bożejko i Wodecki [4]). Otoczenia generowane przez pojedyncze ruchy mają zazwyczaj liczbę elementów rzędu $O(n^2)$ (gdzie n jest liczbą elementów permutacji). W przypadku multiruchów może być ona wykładnicza Congram i in. [5] oraz [4]. W literaturze przedstawiono wiele własności przyspieszających proces przeszukiwania otoczeń. Wśród nich można wyróżnić tzw. „własności eliminacyjne bloków", które z powodzeniem zastosowano w algorytmach rozwiązywania trudnych problemów jednomaszynowych

(Wodecki [12]) oraz wielomaszynowych (Nowicki i Smutnicki [10]). W dalszej części pracy przedstawimy metodę generowania otoczeń oraz subotoczeń zastosowaną w algorytmie TS rozwiązywania problemu **NSFS**. Udowodnimy własności, umożliwiające pominięcie wielu ruchów (elementów otoczenia), ponieważ generowane przez te ruchy rozwiązania nie dają poprawy wartości aktualnie najlepszego rozwiązania.

Niech $\pi \in \Phi$ będzie pewną *n*-elementową permutacją - rozwiązaniem dopuszczalnym rozpatrywanego problemu. *Ruchem* nazywamy przekształcenie generujące z π nową permutację, która jest rozwiązaniem dopuszczalnym problemu, tj. należy do zbioru Φ. Ruch polega więc na zmianie kolejności elementów w permutacji π. Jeżeli *I* jest pewnym ustalonym zbiorem ruchów, wówczas

$$N_I(\pi) = \{\beta : \beta = r(\pi), r \in I\} \tag{6}$$

jest *otoczeniem* permutacji π. Oczywiście, zbiór $N_I(\pi) \subseteq \Phi$.

W konstrukcjach wielu algorytmów rozwiązywania problemów optymalizacji dyskretnej są z powodzeniem stosowane zarówno otoczenia o wielomianowej, jak i wykładniczej liczbie elementów. Im otoczenie jest większe, tym dłużej trawa jego przeszukiwanie. Są to zazwyczaj najbardziej czasochłonne obliczenia w wielu algorytmach metaheurystycznych. W dalszej części pracy udowodnimy, że pewne ruchy, które nie generują rozwiązań lepszych niż π, można pominąć w procesie generowania otoczenia.

3.2. Reprezentacja grafowa rozwiązania

Dla rozwiązania $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$, tj. kolejności wykonywania zadań przez każdą z maszyn konstruujemy graf skierowany

$$G(\pi) = (V, A(\pi); p, s, r),$$

z obciążonymi wierzchołkami i łukami, gdzie:

(a) zbiór wierzchołków $V = O$,

(b) zbiór łuków $A(\pi) = R(\pi) \cup E^+(\pi) \cup E^-(\pi)$ zawiera:

- *łuki pionowe*: $R(\pi) = \bigcup_{i=1}^{n} \bigcup_{k=1}^{m-1} \{O_{k,\pi(i)}, O_{k+1,\pi(i)}\}$. Łuki tego zbioru łączą kolejne operacje tego samego zadania (reprezentują ciąg technologiczny).
- *łuki poziome*: $E^+(\pi) = \bigcup_{k=1}^{m} \bigcup_{i=1}^{n-1} \{O_{k,\pi(i)}, O_{k,\pi(i+1)}\}$, reprezentujące permutację π,
- *łuki powrotne*: $E^-(\pi) = \bigcup_{k \in B^{ni}} \bigcup_{m=1}^{n-1} \{O_{k,\pi(i+1)}, O_{k,\pi(i)}\}$, zapewniające ciągłość pracy maszyny.

(c) wagi wierzchołków $p : V \to \Re$, $p(O_{i,j}) = p_{i,j}$, $O_{i,j} \in J_i$, $J_i \in J, i \in M$,

(d) wagi łuków:

- pionowych, jest równa zero,
- poziomych $s : E^+(\pi) \to \Re$, $s(O_{k,\pi(i)}, O_{k,\pi(i+1)}) = s^k_{\pi(i),\pi(i+1)}$, $k \in M$, $i = 1,2,\ldots,n-1$,

- powrotnych $r: E^{+}(\pi) \to \Re,\ r(O_{k,\pi(i+1)}, O_{k,\pi(i)}) = p_{k,\pi(i)} - s^{k}_{\pi(i),\pi(i+1)}$.

Ciąg wierzchołków $W(v_1, v_w) = (v_1, v_2, \ldots, v_w)$ grafu $G(\pi)$ takich, że $(v_i, v_{i+1}) \in A(\pi)$, $i = 1,2,..., w-1$ nazywamy *ścieżką* z wierzchołka v_1 do v_w. Jej długość

$$L(v_1, v_w) = \sum_{i=1}^{w} p(v_i) + \sum_{i=1}^{w-1} s_{v_i, v_{i+1}}$$

jest równa sumie wag wierzchołków i łuków (włącznie z wagą pierwszego i ostatniego wierzchołka). Przez $W^{*}(O_{1,\pi(1)}, O_{m,\pi(n)})$ oznaczamy *ścieżkę krytyczną* w grafie $G(\pi)$, tj. najdłuższą ścieżkę w grafie z wierzchołka $O_{1,\pi(1)}$ do $O_{m,\pi(n)}$, przez $L^{*}(O_{1,\pi(1)}, O_{m,\pi(n)})$ jej długość.

Wniosek 1 *Dla permutacji zadań* $\pi \in \Phi$ *czas zakończenia wykonywania wszystkich operacji przez maszyny jest równy długości ścieżki krytycznej* $W^{*}(O_{1,\pi(1)}, O_{m,\pi(n)})$ *w grafie* $G(\pi)$, *tj.* $C_{\max}(\pi) = L^{*}(O_{1,\pi(1)}, O_{m,\pi(n)})$.

Dla ustalonej permutacji $\pi \in \Phi$, niech $P^{*}(O_{1,\pi(1)}, O_{m,\pi(n)})$ będzie ciągiem wierzchołków ścieżki krytycznej w grafie $G(\pi)$. Subpermutację zadań (tj. ciąg bezpośrednio występujących po sobie zadań w permutacji)

$$K^{k} = (\pi(a^{k}), \pi(a^{k}+1), \ldots, \pi(b^{k}))\ k \in M$$

nazywamy *k-tą składową* ścieżki krytycznej, jeżeli:

(i) operacje z K^{k} są wykonywane przez tą samą *k*-tą maszynę,

(ii) K^{k} jest maksymalną (ze względu na zawieranie) subpermutacją spełniającą ograniczenie (*i*).

Przez $K = [K^{1}, K^{2}, \ldots, K^{m}]$ będziemy oznaczali ciąg kolejnych składowych permutacji π. Łatwo zauważyć, że każde zadanie należy do przynajmniej jednej ze składowych oraz jeżeli $i < j, i, j \in M$, to

1. $K^{i} \cap K^{j} = \varnothing$, gdy $i+1 < j$ lub
2. $K^{i} \cap K^{j} = \{\pi(b^{i}) = \pi(a^{j})\}$, gdy $i+1 = j$.

Elementami wspólnymi kolejnych składowych są więc jedynie ich pierwsze oraz ostatnie elementy.

3.3. Bloki zadań

Ciąg wierzchołków $(v_1, v_2, \ldots v_t)$ $(v_i \in V, i = 1,2,\ldots,t)$ grafu $G(\pi)$ nazywamy *ścieżką komiwojażera* z v_1 do v_t jeżeli jest to najkrótsza ścieżka pomiędzy tymi wierzchołkami zawierająca wszystkie elementy zbioru $\{v_2, v_3, \ldots, v_{t-1}\}$. Łatwo zauważyć, że dowolna zmiana kolejności wierzchołków tej ścieżki nie zmniejsza odległości pomiędzy v_1 do v_t w grafie $G(\pi)$.

Niech

$$K^k = (\pi(a^k), \pi(a^k+1), \ldots, \pi(b^k))$$

będzie *k*-tą składową. Permutację $\pi_k^* = (\pi_k^*(a^k), \pi_k^*(a^k+1)\ldots, \pi^*(b^k)$ nazywamy *wzorcem* dla *k*-tej składowej, jeżeli $\pi(a^k) = \pi^*(a^k)$, $\pi(b^k) = \pi^*(b^k)$ oraz $(\pi^*(a^k), \ldots, \pi^*(b^k))$ jest ścieżką komiwojażera zawierającą dokładnie wszystkie wierzchołki składowej K^k. Wobec tego, π_k^* jest optymalną kolejnością wykonywania zadań składowej K^k przez *k*-tą maszynę.

Niech

$$J^l = (\pi(a), \pi(a+1), \ldots, \pi(b)), \tag{7}$$

będzie ciągiem bezpośrednio występujących po sobie zadań w składowej K^k, π_k^* jej wzorcem oraz u, v $(u \neq v, a^k \leq u, v \leq b^k)$ parą pozycji w permutacji π takich, że:

W1: $\pi(a) = \pi_k^*(u), \pi(a+1) = \pi_k^*(u+1), \ldots, \pi(b-1) = \pi_k^*(v-1), \pi(b) = \pi_k^*(v)$,

lub

W2: $\pi(b) = \pi_k^*(u), \pi(b-1) = \pi_k^*(u+1), \ldots, \pi(a+1) = \pi_k^*(v-1), \pi(a) = \pi_k^*(v)$,

W3: J^l jest maksymalnym podciągiem ze względu na zawieranie (tj. nie można go powiększyć ani o element $\pi(a-1)$, ani o $\pi(b+1)$), spełniającym ograniczenie **W1** lub **W2**.

Jeżeli ciąg zadań (7) w permutacji π spełnia warunki **W1** i **W3** lub **W2** i **W3**, to nazywamy go *blokiem* dla *k*-tej maszyny ($k \in M$). Blok bez pierwszego i ostatniego elementu (tj. gdy $b - a \geq 2$) nazywamy *blokiem wewnętrznym*.

Niech $J^k = [J_1^k, J_2^k, \ldots, J_{n_k}^k]$ będzie ciągiem bloków dla k-tej składowej. Łatwo wykazać, że każdy element składowej K^k należy do dokładnie jednego bloku. Algorytm rozbicia składowej na bloki (*n* elementowej permutacji) ma złożoność obliczeniową $O(n)$, zobacz [3].

Twierdzenie 1 *Jeżeli permutacja* β *została wygenerowana z* $\pi \in \Phi$ *przez zmianę kolejności elementów w pewnym bloku wewnętrznym, to*

$$C_{\max}(\beta) \geq C_{\max}(\pi).$$

Dowód. Niech β będzie permutacją wygenerowaną z $\pi \in \Phi$ przez zamianę kolejności elementów w pewnym bloku wewnętrznym permutacji π. Zakładamy **nie wprost**, że $C_{\max}(\beta) < C_{\max}(\pi)$. Idea dalszej części bazuje na dowodzie Twierdzenia 3 zamieszczonego w pracy [8].

Z Twierdzenia 1 wynika, że generując otoczenie można pominąć rozwiązania wyznaczone przez zmianę kolejności elementów bloku wewnętrznego. Nie dają bowiem one bezpośredniej poprawy wartości bieżącego rozwiązania.

3.4. Generowanie subotoczeń

W literaturze opisano wiele ruchów bazujących na zamianie kolejności elementów w permutacji. Na podstawie Twierdzenia 1 można wysnuć przypuszczenie, że korzystnym będzie stosowanie ruchów typu „wstaw" (*i-ruch*) powiększających bloki. Generalnie, taki ruch sprowadza się do przestawienia elementu z jego pozycji w permutacji przed lub za inny element. Dokładniej, dla różnych pozycji t,l w permutacji $\pi \in \Phi$ ruch typu wstaw i_l^t generuje nową permutację $\pi_l^t = i_l^t(\pi)$ przez przestawienie elementu $\pi(t)$ z pozycji t na pozycję l w π.

Otoczenie (6) generowane z zastosowaniem tzw. „*własności eliminacyjnych bloków*" (Twierdzenie 1) będziemy nazywali *subotoczeniem*. Jego stosowanie znacznie przyspiesza działanie algorytmu TS. Im większe są bloki tym mniejsze są subotoczenia, a więc i krótszy jest czas ich przeszukiwania.

Jeżeli $I(\pi)$ jest zbiorem wszystkich *i-ruchów*, to **otoczeniem** permutacji $\pi \in \Phi$, jest zbiór

$$N_I(\pi) = \{i_l^t(\pi) = \pi_l^t : i_l^t \in I(\pi)\}.$$

Liczba elementów otoczenia (moc zbioru $N_I(\pi)$ generowanego przez i-ruchy wynosi $(n-1)^2$.

Niech

$$K = [K^1, K^2, \ldots, K^m]$$

będzie ciągiem składowych permutacji $\pi \in \Phi$. Zakładamy, że dla *k*-tej składowej

$$K^k = (\pi(a^k), \ldots, \pi(b^k),$$

permutacja

$$\pi_k^* = (\pi^*(a^k), \ldots, \pi^*(b^k)$$

jest wzorcem, a $J_l^k \;\; l = 1,2,\ldots,n_k$ *l*-tym blokiem. Dla ustalenia uwagi załóżmy, że

$$J_l^k = (\pi(a), \pi(a+1), \ldots, \pi(b)).$$

Jeżeli $\pi^*(v) = \pi(a) \;\; (a^k \le v \le b^k)$, to przez i_{a-1}^v oznaczamy *i*-rucha polegający na przestawieniu elementu z pozycji v na pozycję $a-1$ (bezpośredni przez blok). Podobie, przez przez i_{b+1}^v oznaczamy ruch polegający na przestawieniu elementu z pozycji v na pozycję $b+1$ (bezpośrednio za blok).

Dla bloku $J_l^k, l = 1,2,\ldots,n_k,$ definiujemy zbiory zadań *kandydatów* do przestawienia w permutacji π

$$I_l^k = \{i_v^u) : u = a^k, a^k+1, \ldots, a-2, b+2, b+3, \ldots, b^k, v \in \{a-1, b+1\}\},$$

Zbiory te zawierają elementy, które będą przestawiane „za" lub „przed" odpowiedni blok, tj. za ostatni lub przed pierwszy element bloku.

Oczywiście ruchy ze zbiorów I_l^k **mogą** (lecz nie muszą) przynieść poprawę bieżącej wartości rozwiązania $C_{\max}(\pi)$.

Na podstawie przedstawionych rozważań, ostatecznie generując z π nową permutację

będziemy stosowali ruchy ze zbioru

$$I(\pi) = \bigcup_{k=1}^{m} \bigcup_{l=1}^{n_k} I_l^k(\pi), \tag{8}$$

Rozmiar tego otoczenia zależy bezpośrednio od liczby bloków, a nie bezpośrednio od liczby zadań *n* oraz maszyn *m*.

Przejdziemy obecnie do opisu metody wyznaczania elementu w permutacji, który zostanie przestawione za ostatni (lub przed pierwszy) element bloku. Szukamy *i*-ruchu generującego permutację (graf) o możliwie najmniejszej długości ścieżki krytycznej.

Dla uproszczenia zapisu przyjmujemy następujące założenia:

1. bieżąca permutacja $\pi(i) = (1,2,3,\ldots,n)$,
2. rozpatrujemy *r*-ty blok $B = (a, a+1,\ldots,b-1,b)$,
3. pomijamy indeksy maszyn (np. przy oznaczeniach czasów przezbrojeń).

Dla dowolnego ruchu $i_v^l \in I(\pi), v = a-1$ (tj. przestawiającego element l bezpośrednio **przed** blok B) wprowadzamy oznaczenie

$$\Delta_{af}(v,l) = s_{l-1,+1} - s_{l-1,l} - s_{l,l+1} + s_{v,a} + s_{v,l} - s_{l,a}. \tag{9}$$

Podobnie dla ruch $i_v^l \in \mathrm{I}_b(\pi), v = b+1$ (tj. przestawiającego element l bezpośrednio **za** blok B)

$$\Delta_{bf}(v,l) = s_{l-1,l+1} - s_{l-1,l} - s_{l,l+1} + s_{b,l} + s_{l,b+1} - s_{b,b+1}. \tag{10}$$

Złożoność obliczeniowa wyznaczenia wartości wyrażenia $\Delta_\lambda(v,l)$, $\lambda \in \{a,b\}$ jest $O(1)$.

Twierdzenie 2 *Jeżeli permutacja* π_v^l *została wygenerowana z* $\pi \in \Phi$ *przez wykonanie ruchu* $i_v^l \in I(\pi)$, *to termin zakończenia wszystkich zadań*

$$C_{\max}(\pi_v^l) \geq C_{\max}(\pi) + \Delta_{af}(v,l),$$

gdzie $\lambda \in \{a,b\}$.

Dowód. Dowód sprowadza się do wykazania, że w grafie $G(\pi_v^l)$ istnieje pewna droga z wierzchołka $O_{1,\pi_v^l(1)}$ do wierzchołka $O_{m,\pi_v^l(n)}$ o długości $C_{\max}(\pi) + \Delta_{af}(v,l)$. Ponieważ $C_{\max}(\pi_v^l)$ jest najdłuższą taką drogą, stąd wynika nierówność w tezie twierdzenia.

Wartość wyrażenia $C_{\max}(\pi) + \Delta_{af}(v,l)$ jest więc dolnym oszacowaniem rozwiązania $C_{\max}(\pi_v^l)$. Wobec tego, $\Delta_\lambda(x)\ \lambda \in \{af, bf\}$ mogą być stosowane jako kryterium wyboru najlepszego ruchu - elementu z subotoczenia $N_I(\pi)$.

Stosując ruchy typu zamień (*swap*), w podobny sposób możemy generować oraz przeszukiwać subotoczenia w algorytmie TS.

4. Wyznaczanie wzorców

Do wyznaczania bloków dla k-tej maszyny konieczny jest wzorzec - ścieżka komiwojażera pomiędzy parą wierzchołków w grafie. Jej wyznaczenie jest problemem *NP-trudnym*. Dla praktycznych problemów z dużą liczbą zadań, czas obliczeń algorytmu optymalnego jest w praktyce nieakceptowalny. W tym przypadku stosować będziemy algorytm przybliżony *2-opt*. Ma on niewielką złożoność $O(n^2)$, i jak wynika z opisanych w literaturze eksperymentów obliczeniowych, wyznacza rozwiązania różniące się o kilka procent od optymalnych. Zastosowanie algorytmu przybliżonego powoduje, że mogą nie być spełnione założenia dotyczące definicji bloku (wzorzec może nie być ścieżką komiwojażera). W wyniku tego, korzystając z Twierdzenia 1, przy generowaniu subotoczeń mogą być ewentualnie wyeliminowane „dobre" (tj. dające bezpośrednią poprawę) rozwiązania.

5. Eksperymenty obliczeniowe

Przeprowadzone zostały eksperymenty numeryczne których celem było zweryfikowanie efektywności badanego algorytmu przeszukiwania z tabu. Algorytm zaimplementowany został w języku C++ w środowisku Microsoft Visual Studio 2010 i przetestowany na komputerze Samsung NP730U3E-S02PL wyposażonym w procesor Intel Core i7-3537U 2,0 GHz pracujacym pod kontrolą systemu operacyjnego Windows 10.

Obliczenia zostały przeprowadzone na zbiorze 110 przykładów przepływowego problemu szeregowania z rpzezbrojeniami zaczerpniętych z pracy Ruiz i Stützle [11]. Dane pogrupowano w 11 grup, każda o tej samej liczbie zadań oraz maszyn. Jako miarę jakości algorytmu przyjęto średnie procentowe odchylenie (*Percentage Relative Deviation, PRD*) najlepszego otrzymanego rozwiązania π względem rozwiązania referencyjnego π^{ref} :

$$PRD(\pi) = 100\%(C_{max}(\pi) - C_{max}(\pi^{ref}))/C_{max}(\pi^{ref}).$$

Algorytm został zaimplementowany w dwóch wersjach: z otoczeniem generowanym ruchami typu wstaw (*insert*) oraz zamień (*swap*). Obliczenia przeprowadzono dla liczby iteracji wynoszącej 1000. Dla każdej instancji problemu jako referencyjne przyjęte zostało rozwiązanie otrzymane za pomocą konstrukcyjnego algorytmu NEH (Nawaz, Enscore, Ham [9]). Otrzymane wyniki zaprezentowane są w tabeli 2. Poszczególne kolumny oznaczają odpowiednio:

- t_N – czas działania algorytmu NEH,
- t_I – czas działania algorytmu TS z otoczeniem typu wstaw (*insert*),
- t_S – czas działania algorytmu TS z otoczeniem typu zamień (*swap*),
- PRD_I – procentowy błąd względny algorytmu z ruchami typu wstaw
- PRD_S – procentowy błąd względny algorytmu z ruchami typu zamień.

Tabela 2. Czas obliczeń oraz PRD dla ustalonej liczby iteracji.

$n \times m$	$t_N(s)$	$t_I(s)$	$t_S(s)$	$PRD_I(\%)$	$PRD_S(\%)$
20×5	0,0034	0,0186	0,0177	-2,74	-2,19
20×10	0,0193	0,0567	0,0567	-2,64	-2,66
20×20	0,0204	0,1306	0,1285	-3,83	-3,14
50×5	0,0991	0,2092	0,2156	-1,56	-1,69
50×10	0,1630	0,4528	0,4459	-3,25	-3,86
50×20	0,3034	0,9229	0,9302	-3,48	-3,82
100×5	7,7354	1,2632	1,2317	-0,88	-0,99
100×10	1,5440	2,7882	2,7226	-1,26	-1,37
100×20	3,2280	5,7525	5,7632	-2,29	-2,73
200×10	20,766	19,158	19,243	-0,84	-1,02
200×20	43,731	41,346	41,033	-1,50	-1,73
Średnio	**6,4194**	**6,5544**	**6,5265**	**-2,21**	**-2,29**

Na podstawie zamieszczonych wyników można stwierdzić, że wybór ruchu, przy generowaniu otoczenia, praktycznie nie ma wpływu na czas obliczeń oraz wartości wyznaczanych rozwiązań (różnice są niewielkie). Należy podkreślić, że obie wersje algorytmu przeszukiwania z tabu mają niemal identyczny średni czas obliczeń, jak algorytm konstrukcyjny NEH.

6. Wnioski

W pracy przedstawiono algorytm przeszukiwania z tabu dla problemu przepływowego z ciągłą pracą pewnych maszyn oraz przezbrojeniami maszyn pomiędzy kolejno wykonywanymi operacjami. Przy generowaniu otoczeń wykorzystano eliminacyjne własności bloków ze ścieżki krytycznej. Natomiast, w procedurze przeszukiwania subotoczenia, wartości funkcji celu są szacowane z dołu w czasie stałym, niezależnym od liczby zadań oraz liczby maszyn.

Literatura

1. Bożejko W., Hejducki Z., Wodecki M., Applying metaheuristic strategies in construction projects management, Journal of Civil Engineering and Management, Taylor & Francis, 2012, Valume 18(5), 621–630.
2. Bożejko W., Hejducki Z., Uchroński M., Wodecki M., Solving resource-constrained con-struction scheduling problems with overlaps by metaheuristic, Journal of Civil Engineering and Management, Taylor & Francis, 2014, Volume 20(5), 649-659.
3. Bożejko W., Uchroński M., Wodecki M., Block approach to the cyclic flow shop scheduling, Computers & Industrial Engineering, 81, 2015, 158–166.
4. Bożejko W., Wodecki M., On the theoretical properties of swap multimoves, Operations Research Letters, 35(2), 2007, 227–231.
5. Congram, R.K., Potts C.N., Van de Velde S.L., An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem, INFORMS Journal on Computing, vol. 14, no. 1, 2002, 52–67.
6. Glover F., Tabu search. Part I.ORSA Journal on Computing, 1, 1989, 190-206.
7. Glover F., Tabu search. Part II.ORSA Journal on Computing, 2, 1990, 4–32.

8. Grabowski J., Wodecki M., A very fast tabu search algorithm for the permutation flow shop problem with makespan criterion, Computers & Operations Research, 31, 2004, 1891–1909.
9. Nawaz M., Enscore E.E., Ham I., A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem, *OMEGA*, **11**/**1**, 1983, 91–95.
10. Nowicki E., C. Smutnicki, A fast tabu search algorithm for the permutation flow-shop problem, European Journal of Operational Research, 1996, 91, 160–175.
11. Ruiz R., Stützle T., An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives, European Journal of Operational Research 187 (3), 2008, 1143–1159.
12. Wodecki M., A block approach to earliness-tardiness scheduling problems, International Journal on Advanced Manufacturing Technology, 40, 2009, 797–807.

Dr hab. Wojciech BOŻEJKO, prof. nadzw.
Mgr inż. Radosław IDZIKOWSKI
Katedra Automatyki, Mechatroniki i Systemów Sterowania
Wydział Elektroniki
Politechnika Wrocławska,
Wyb. Wyspiańskiego 27,50-370 Wrocław
e-mail: wojciech.bozejko@pwr.edu.pl
radoslaw.idzikowski@pwr.edu.pl

Dr hab. Mieczysław WODECKI, prof. nadzw.
Instytut Informatyki Uniwersytet Wrocławski,
ul.Joliot-Curie 50-383 Wrocław,
e-mail: mieczyslaw.wodecki@uwr.edu.pl