RÓWNOLEGŁY ALGORYTM PRZESZUKIWANIA Z ZABRONIENIAMI DLA CYKLICZNEGO PROBLEMU GNIAZDOWEGO

Wojciech BOŻEJKO, Andrzej GNATOWSKI, Mieczysław WODECKI

Streszczenie: W pracy rozpatrujemy cykliczny problemem gniazdowy, polegający na wytwarzaniu w ustalonych odstępach czasu pewnego zestawu elementów. Optymalizacja procesu sprowadza się do minimalizacji czasu cyklu, tj. czasu, po którym może nastąpić wykonywanie następnej partii tych samych elementów. Ponieważ problem jest silnie NP-trudny, więc do jego rozwiązywania będziemy stosowali algorytm przybliżony zaprojektowany do pracy w środowisku wieloprocesorowym.

Słowa kluczowe: szeregowanie zadań, algorytm równoległy, poszukiwanie z tabu

1. Wstęp

Problemy cykliczne stanowią unikalną, stosunkowo mało zbadaną podklasę problemów szeregowania zadań. Jednak ostatnio cieszą się one znacznie większym zainteresowaniem zarówno praktyków jak i teoretyków, głównie ze względu na ich duże znaczenie praktyczne oraz trudności z konstruowaniem odpowiednio efektywnych algorytmów rozwiązywania. Silna NP-trudność już wielu najprostszych wersji badanego problemu ogranicza zakres stosowania algorytmów dokładnych wyłącznie do instancji o niewielkim rozmiarze.

W rozpatrywanym wielomaszynowym systemie produkcji cyklicznej dowolny element z ustalonej partii (mieszanki) przechodzi w zadanej kolejności technologicznej przez maszyny. Problem polega na minimalizacji czasu cyklu, tj. czasu po którym może nastąpić wykonywanie następnej partii tych samych elementów. Formalnie można go sformułować następująco: rozważany jest system wytwórczy złożony z m stanowisk (maszyn o jednostkowej przepustowości) oznaczonych $M = \{1,2,\ldots,m\}$. W systemie należy wykonywać cyklicznie (w sposób powtarzalny) n zadań danych zbiorem $N = \{1,2,\ldots,n\}$. Zadanie j-te wymaga sekwencji n_j operacji indeksowanych kolejnymi liczbami naturalnymi $(l_{j-1}+1,\ldots,l_{j-1}+n_j)$, wykonywanych w podanej kolejności, gdzie $l_j = \sum_{i=1}^{j} n_i$ jest całkowitą liczbą operacji pierwszych j zadań, $j=1,2,\ldots,n$, $l_0 = 0$, oraz $\sum_{i=1}^{n} n_i = o$. Operacja $i \in O = \{1,2,\ldots,o\}$ ma być wykonywana na maszynie $v_i \in M$ w nieprzerywalnym czasie $p_i > 0$. Należy wyznaczyć *cykliczny* harmonogram pracy systemu przy następujących ograniczeniach: (1) każda maszyna może wykonywać co najwyżej jedną operację w każdej jednostce czasu, (2) każda operacja może być wykonywany na co najwyżej jednej maszynie w każdej jednostce czasu, (3) wykonywanie operacji nie może być przerwane.

Zbiór zadań wykonywanych w pojedynczym cyklu nazywany jest MPS-em (*ang. Minimal Part Set*). MPS-y są przetwarzane jeden po drugim w sposób cykliczny, dostarczając partię zadań (mieszankę produktów) w ilościach odpowiedniej dla każdego cyklu. Problem sprowadza się więc do ustalenia momentów rozpoczęcia wykonywania zadań na maszynach, aby czas cyklu (czas po którym zadanie jest wykonywane w kolejnym MPS-ie) był minimalny.

Przegląd stanu wiedzy dotyczącego cyklicznych zagadnień szeregowania zadań znaleźć można w pracy Levner i in. [1], a także w rozprawie doktorskiej Kampmeyera [2]. Przegląd zagadnień produkcji cyklicznej z zadaną liczbą MPS znaleźć można w pracach Sawika [3, 4]. Nowe własności harmonogramów cyklicznych zaproponował Smutnicki [5-7] oraz Pempera i Smutnicki [8] i Dąbrowski, Pempera, Smutnicki [9].

Wstępne wyniki badań autorów nad problemami cyklicznymi opublikowane zostały w pracach Bożejko, Gniewkowski, Pempera, Wodecki [10], Bożejko, Kacprzak, Wodecki [11] oraz Bożejko i Wodecki [12].

2. Model matematyczny

Zbiór operacji O w pojedynczym MPS może być w naturalny sposób zdekomponowany na podzbiory $O_k = \{j \in O : v_j = k\}$, z których każdy odpowiada operacjom wykonywanych na maszynie k; niech $m_k = |O_k|$, $k \in M$. *Kolejność wykonywania* operacji na maszynach jest definiowana jako m-krotka $\pi = (\pi_1,...,\pi_m)$, gdzie $\pi_k = (\pi_k(1),\ldots,\pi_k(m_k))$ jest permutacją elementów zbioru O_k, $k \in M$; $\pi_k(i)$ oznacza element O_k, który jest na pozycji i w π_k. Niech Π_k będzie zbiorem wszystkich permutacji elementów z O_k. Stąd kolejność wykonywania $\pi \in \Pi_1 \times \Pi_2 \times ... \times \Pi_m$.

Harmonogram dla kolejności wykonywania x-tego MPS (według kolejności wykonywania π) opisujemy zasadniczo przez dwa wektory zdarzeń $S^x = (S_1^x, \ldots, S_o^x)$, $C^x = (C_1^x, \ldots, C_o^x)$, gdzie S_j^x *i* C_j^x oznaczają odpowiednio termin rozpoczęcia i zakończenia operacji j w x-tym MPS. Zdarzenia te muszą spełniać następujące ograniczenia:

$$C_i^x \le S_{i+1}^x,\ i = l_{j-1}+1,\ldots,l_j-1,\ j=1,\ldots,n, \tag{1}$$

$$C_{\pi_k(i)}^x \le S_{\pi_k(i+1)}^x,\ i=1,\ldots,m_k-1,\ k=1,\ldots,m, \tag{2}$$

$$C_i^x = S_i^x + p_i,\ i=1,\ldots,o, \tag{3}$$

$$S_i^x \ge 0,\ i=1,\ldots,n, \tag{4}$$

gdzie x=1,2,… .

Ograniczenie (1) wynika z porządku technologicznego wykonywania operacji wewnątrz zadań, podczas gdy (2) z sekwencyjnego charakteru pracy maszyn. Równania (3) oraz (4) są oczywiste. Mówimy, że kolejność wykonywania π jest dopuszczalna, jeśli istnieje co najmniej jeden harmonogram S^x, C^x, x=1,2,… , spełniający ograniczenia (1)–(4). Zauważmy, że używając (3) możemy reprezentować harmonogram dla x-tego MPS przez jeden wektor S^x. Wtedy ograniczenia (1)–(4) przyjmą następującą postać:

$$S_i^x + p_i \le S_{i+1}^x,\ i = l_{j-1}+1,\ldots,l_j-1,\ j=1,\ldots,n, \tag{5}$$

$$S_{\pi_k(i)}^x + p_{\pi_k(i)} \le S_{\pi_k(i+1)}^x,\ i=1,\ldots,m_k-1,\ k=1,\ldots,m, \tag{6}$$

$$S_i^x \ge 0,\ i=1,\ldots,n. \tag{7}$$

W proponowanym modelu możliwe są „przeploty" zadań z kolejnych MPS, tj. wykonywanie operacji z kolejnego, x+1-go MPS przed zakończeniem wykonywania wszystkich operacji z x-tego MPS na danej maszynie. Przeploty takie mogą zostać

zabronionie (z różnych względów, m.in. technologicznych lub organizacyjnych) poprzez dodanie do modelu ograniczenia:

$$C^{x}_{\pi_k(m_k)} \le S^{x+1}_{\pi_k(1)}, k = 1,\ldots,m, \tag{8}$$

lub równoważnego:

$$S^{x}_{\pi_k(m_k)} + p_{\pi_k(m_k)} \le S^{x+1}_{\pi_k(1)}, k = 1,\ldots,m. \tag{9}$$

Definicja 1 *Czas cyklu T w fazie ustalonej harmonogramu, tj. gdy różnica momentów rozpoczęcia operacji w kolejnych ich powtórzeniach jest stała, wynosi* $T = S_j^{x+1} - S_j^x$, x=1,2,….

Czas cyklu T jest liczbą rzeczywistą i zależy od π, a oczywiście nie każda jego wartość jest dopuszczalna, bowiem momenty rozpoczęcia wykonywania zadań S_j^x, j=1,2,…,o, x=1,2,…, muszą także spełniać ograniczenia (5)–(7). Przez $T(\pi)$ będziemy oznaczać minimalną wartość czasu cyklu, spełniającą ograniczenia (5)–(7).

2. Model grafowy

Niech grafy $H(\pi)$ i $G(\pi)$ będą zdefiniowane w następujący sposób [13]:

$$G(\pi) = (\mathcal{O}, \mathcal{R} \cup \mathcal{E}(\pi)), \tag{10}$$

$$H(\pi) = (\mathcal{O}, \mathcal{R} \cup \mathcal{E}(\pi) \cup \mathcal{E}^*(\pi)), \tag{11}$$

gdzie $\mathcal{O}$ jest zbiorem wierzchołków, a $\mathcal{R} \cup \mathcal{E}(\pi) \cup \mathcal{E}^*(\pi)$ to zbiory łuków:

$$\mathcal{R} = \bigcup_{j=1}^{n} \bigcup_{i=l_{j-1}+1}^{l_j-1} \{(i, i+1)\}, \tag{12}$$

$$\mathcal{E}(\pi) = \bigcup_{k=1}^{m} \bigcup_{i=1}^{m_k-1} \{(\pi_k(i), \pi_k(i+1))\}, \tag{13}$$

$$\mathcal{E}^*(\pi) = \bigcup_{k=1}^{m} \{(\pi_k(m_k), \pi_k(1))\}. \tag{14}$$

Łuki kolejności technologicznej oznaczmy przez $\mathcal{R}$, łuki kolejności maszynowej przez $\mathcal{E}(\pi)$, a łuki cykliczne przez $\mathcal{E}^*(\pi)$. Niech wierzchołek $j \in \mathcal{O}$ reprezentuje j-tą operację i posiada wagę 0. Łuki ze zbioru $\mathcal{R}$ reprezentują kolejność technologiczną wykonywania operacji w zadaniu; łuki ze zbioru $\mathcal{E}(\pi)$ kolejność wykonywania operacji na maszynach, a łuki ze zbioru $\mathcal{E}^*(\pi)$ ograniczenie poprzedzania w kolejnych cyklach:

$$\left(S_{\pi_k(m_k)} + p_{\pi_k(m_k)}\right) - T \le S_{\pi_k(1)}. \tag{15}$$

Każdy łuk $(i,j) \in \mathcal{R} \cup \mathcal{E}(\pi)$ obciążony jest wagą p_i, łuki $(i,j) \in \mathcal{E}^*(\pi)$ posiadają wagę $p_i - T$.

Własność 1 [15] *Kolejność wykonywania* π *jest dopuszczalna wtedy i tylko wtedy, gdy graf* G(π) *nie zawiera cyklu oraz* T *zostało wybrane tak, że* H(π) *nie zawiera cyklu o dodatniej długości.*

3. Przeszukiwanie z zabronieniami

Metoda poszukiwania z zabronieniami (*tabu search, TS*) została pierwotnie zaproponowana przez Glovera [14]. Jest ona modyfikacją metody przeszukiwania lokalnego. Dopuszcza się możliwość zwiększania wartości funkcji celu (przy wyznaczaniu nowego rozwiązania generującego otoczenie), aby w ten sposób zwiększyć szansę na osiągnięcie minimum globalnego. W celu uniknięcia cyklicznego odwiedzania rozwiązań, skierowania poszukiwań w obiecujące regiony przestrzeni oraz umożliwienie wyjścia z ekstremum lokalnego wprowadza się tzw. mechanizm zabronień. Wykonując ruchy zapamiętuje się rozwiązania, atrybuty rozwiązań lub ruchów na liście tabu (LT). Generując otoczenie nie rozpatrujemy rozwiązań znajdujących się na tej liście chyba, że spełniają kryterium aspiracji, to jest warunki, przy których ograniczenia tabu można pominąć. Na Rysunku 1 przedstawione zostały kolejne kroki równoległego algorytmu przeszukiwania z zabronieniami.

Algorytm 1. Przeszukiwanie z zabronieniami
π^* – najlepsze znalezione dotychczas rozwiązanie,
$LT := \emptyset$;
Krok 0: Wyznacz rozwiązanie startowe π^0; $\pi := \pi^0$;
Krok 1: Wyznacz sąsiedztwo $N(\pi)$ bieżącego rozwiązania π;
Usuń z sąsiedztwa $N(\pi)$ elementy zabronione przez listę LT;
Krok 2: Podziel $N(\pi)$ na $k = \left\lceil \frac{N(\pi)}{p} \right\rceil$ grup;
Każda grupa zawiera co najwyżej p elementów;
Krok 3: Dla każdej grupy $s=1,2,...,k$ znajdź (używając p procesorów)
uszeregowania π^s o najmniejszej wartości minimalnego czasu cyklu T
Krok 4: Znajdź uszeregowanie $z \in N(\pi)$ dla którego
$z=\arg\min\{T(\pi^s): s=1,2,...,k\}$
Krok 5: Jeśli $T(z) < T(\pi^*)$ **to** $\pi^* := z$;
Umieść atrybuty ruchu prowadzącego z π do z na liście LT;
Krok 6: Jeśli (*warunek stopu* jest spełniony) **to** Stop;
w przeciwnym wypadku idź do kroku 1.;

Rys. 1. Pseudokod dla algorytmu przeszukiwania z zabronieniami

W drugim kroku algorytmu sąsiedztwo $N(\pi)$ jest dzielone na rozłączne zbiory. Dla każdej grupy k wartość funkcji celu jest wyznaczana przy użyciu p procesorów. Liczba procesorów użytych w kroku trzecim może być stałą lub uzależniona od rozmiaru sąsiedztwa.

Mechanizm zabronień realizowany jest za pomocą listy tabu stanowiącej krótkoterminową historię przeszukiwań. Dodawane są do niej odwiedzone przez algorytm rozwiązania. Gdy długość listy osiągnie maksymalną ustaloną wartość L_{max} przed dodaniem kolejnego rozwiązania usuwane jest najstarsze. W przypadku gdy w danym kroku wszyscy sąsiedzi znajdują się na liście tabu, usuwane są z niej kolejne rozwiązania, począwszy od najstarszego, tak długo, aż co najmniej jeden z sąsiadów nie jest zabroniony. Dodatkowo wprowadzono kryterium aspiracji: każdy sąsiad o wartości funkcji celu mniejszej od dotychczasowego najlepszego rozwiązania nie jest zakazany, nawet jeśli znajduje się na liście tabu.

Sąsiedztwo. Atrybuty ruchu prowadzącego do rozwiązania sąsiedniego opisywane są nieuporządkowaną parą zadań $\{i, j\}, M(i) = M(j) = m$, których kolejność wykonywania na

maszynie *m* jest zamieniana. W algorytmie przyjęto strategię wyboru sąsiada o najmniejszej wartości funkcji celu. Sąsiedztwo tworzone jest w oparciu o teorię bloków: blokiem B dla ścieżki μ w grafie $H(\pi)$ nazywa się taką podścieżkę o maksymalnej długości, która zawiera tylko łuki kolejności maszynowej $\mathcal{R}$. Jak pokazano w [16], czas cyklu może ulec poprawie tylko jeśli modyfikuje się pierwszy lub ostatni łuk bloku *B*. W badaniach eksperymentalnych w [16] opisane sąsiedztwo okazało się być najlepsze z rozważanych, stąd zdecydowano się na jego zastosowanie.

Procedura startowa. Do rozpoczęcia działania algorytmu *TS* potrzebne jest dopuszczalne rozwiązanie początkowe. Tworzone jest ono za pomocą prostego algorytmu ustalającego taką kolejność wykonywania operacji na maszynach, w której wszystkie operacje z zadania i są wykonywane przed operacjami zadania j, jeśli $i < j$.

3.1. Funkcja celu

Funkcja celu $f(\pi)$ dla zadanej kolejności wykonywania operacji π zwraca minimalny czas cyklu $T(\pi)$, $f(\pi) = T(\pi)$. Do tej pory do jej wyznaczania w literaturze stosowana była metoda Howarda [17] (używano jej między innymi w [16]). W niniejszej pracy użyto rozwiązania własnego opartego o metodę bisekcji oraz algorytm Bellmana-Forda.

Znane są sposoby oszacowania wartości długości czasu cyklu celu z góry ($T^*(\pi)$) i z dołu ($T_*(\pi)$). Oszacowaniem górnym może być $T^*(\pi) = C_{max}(\pi)$, a oszacowaniem dolnym maksimum sum czasów wykonywania operacji na maszynach w pojedynczym MPSie:

$$T_*(\pi) = \max_{1 \le k \le m} \left\{ \sum_{i=1}^{m_k} p_{\pi_k}(i) \right\}. \tag{16}$$

Ponieważ suma $\sum_{i=1}^{m_k} p_{\pi_k}(i)$ nie zależy od kolejności π, również tak zdefiniowane dolne oszacowanie nie zależy od π, czyli $T_* = T_*(\pi)$.

3.2. Zmodyfikowany algorytm Bellmana-Forda (B-F)

Za pomocą zmodyfikowanego algorytmu Bellmana-Forda możliwe jest wyznaczenie cyklu o największej wadze w grafie $H(\pi)$ przechodzącego przez zadany wierzchołek reprezentujący operację i. Algorytm Bellmana-Forda [18] umożliwia wyznaczenie najkrótszej ścieżki pomiędzy dwoma wierzchołkami w grafie ważonym. W grafie nie może występować cykl o ujemnej wadze osiągalny ze źródła.

W zastosowaniu do rozważanego problemu należy zmodyfikować algorytm Bellmana-Forda tak, aby znajdował najdłuższą ścieżkę w grafie $H(\pi)$ pomiędzy wierzchołkami i i j. Warunek nie występowania cyklu o łącznej ujemnej wadze zostaje zastąpiony warunkiem nie występowania cyklu o łącznej dodatniej wadze osiągalnej ze źródła. W grafie $H(\pi)$ każdy z wierzchołków ma co najwyżej dwa łuki wychodzące. Stąd najdłuższy cykl przechodzący przez wierzchołek i ma wagę:

$$L_i(T, \pi) = \max\{\big(BF(i, j_1) + \delta(j_1, i)\big), \big(BF(i, j_2) + \delta(j_2, i)\big)\}, \tag{17}$$

gdzie $BF(i, j_1)$ oznacza najdłuższą ścieżkę między wierzchołkami j_1 i i, a $\delta(j_1, i)$ wagę łuku (j_1, i). Jeśli graf $H(\pi)$ posiada dodatni cykl osiągalny z wierzchołka i, algorytm B-F znaleźć może ścieżkę o największej wadze i długości rzędu $O(|V[H(\pi)]| \cdot |E[H(\pi)]|)$. Ścieżkę taką oznaczono dalej jako $L_{BF_i}(\pi, T)$.

Z własności cyklicznego problemu gniazdowego wynika, że każdy cykl grafu $H(\pi)$ musi zawierać łuk cykliczny $(\pi_k(m_k), \pi_k(1))$. Stąd wiadomo, że cykl o największej wadze w grafie zawierać będzie operację:

$$i_c \in O_c = \{\pi_1(1), \pi_2(1), \dots, \pi_m(1)\}. \tag{18}$$

Niech $L(T, \pi)$ oznacza cykl o największej wadze w grafie $H(\pi)$. Uruchamiając zmodyfikowany algorytm Bellmana-Forda dla grafu $H(\pi)$ i operacji ze zbioru O_c można więc wyznaczyć:

$$L(T, \pi) = \max_{i_c \in O_c} \{L_{i_c}(\pi, T)\}. \tag{19}$$

Analogiczne rozumowanie przeprowadzić można gdy graf $H(\pi)$ zawiera cykl o dodatniej wadze, otrzymując $L_{BF}(\pi, T)$.

3.3. Metoda równego podziału (bisekcji)

Dla ustalonego π, $T(\pi)$ szacować można za pomocą metody równego podziału, szukając jak najmniejszego T takiego, że $L(\pi, T) \leq 0$. Jako przedział startowy należy przyjąć $[T_*, T^*(\pi)]$. Opis algorytmy można znaleźć w [19]. Ponieważ:

$$L(T, \pi) = L_{BF}(\pi, T) \text{ dla } T \geq T(\pi) \tag{20}$$

oraz:

$$L(T, \pi) > 0 \wedge L_{BF}(\pi, T) > 0 \text{ dla } T < T(\pi), \tag{21}$$

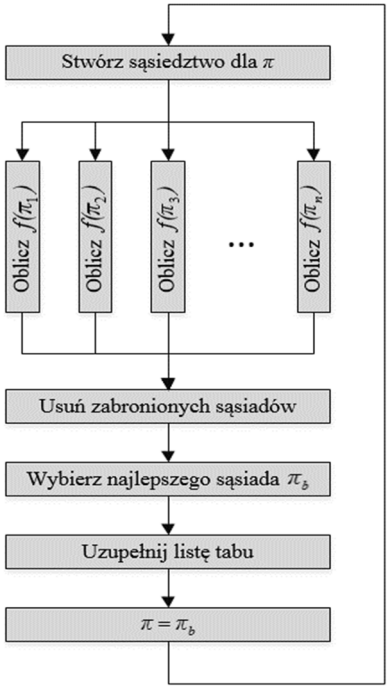
w poszukiwaniu minimalnego czasu cyklu metodą bisekcji funkcje $L_{BF}(\pi, T)$ i $L(T, \pi)$ można stosować zamiennie. Ponieważ funkcja $L_{BF}(T, \pi)$ jest ciągła na przedziale $[T_*, T^*(\pi)]$, oraz:

$$L_{BF}(T_*, \pi) > 0 \wedge L_{BF}(T^*(\pi)_*, \pi) < 0, \tag{22}$$

dozwolone jest użycie metody równego podziału. W przypadku gdy $L_{BF}(T_*, \pi) = 0 \vee L_{BF}(T^*(\pi)_*, \pi) = 0$ stosowanie bisekcji nie jest zasadne, bo znane jest rozwiązanie optymalne. Dodatkowo ścisła monotoniczność funkcji $L_{BF}(T, \pi)$ gwarantuje istnienie jednego pierwiastka równania $L_{BF}(T, \pi) = 0$, dla zadanego π.

3.4. Wersja równoległa algorytmu

Obliczanie wartości funkcji oceniającej zabiera największą część czasu pojedynczej iteracji TS (patrz Rysunek 2), stąd obszar ten zbadano pod kątem możliwości przyspieszenia algorytmu. Zrównoleglić można obliczenia związane z wyznaczaniem funkcji oceniającej, bądź obliczać ją dla wielu permutacji jednocześnie. W pracy tej przyjęto drugie z wymienionych rozwiązań.



Rys. 2. Uproszczony schemat pętli algorytmu TS

W każdej iteracji algorytmu TS generowane jest otoczenie rozwiązania. Następnie obliczana jest wartość funkcji oceniającej dla każdego z sąsiadów. W zrównoleglonym algorytmie obliczania te wykonywane są równolegle. Ze względu na architekturę sprzętu na jakim przeprowadzane były eksperymenty, wybrano technologię implementacji *OpenMP*. Pozwala ona tworzyć wieloplatformowe aplikacje dla systemów wieloprocesorowych ze współdzieloną pamięcią. Do zrównoleglenia użyto klauzuli pozwalającej na wybór liczby wątków obliczeniowych:

#pragma omp parallel for num_threads(LICZBA_WATKOW).

Umożliwia ona równoczesne wykonywanie przez wiele wątków pętli *for*. Procedura przydziału zadań do wątków jest realizowana automatycznie przez *OpenMP*.

4. Wyniki eksperymentalne

Algorytm został zaimplementowany w języku C++ w środowisku Visual Studio 2013. Testy przeprowadzono na komputerze z 6-cio rdzeniowym (12 wątków) procesorem Intel Core i7-4930K 3.90 GHz, systemie Windows 8.1 Pro z 32GB pamięci RAM.

4.1. Instancje testowe

Użyto przykładów ze zbioru *OR-Library* po raz pierwszy opisanego w [20], do których dostęp można uzyskać na stronie internetowej [21]. Wybrano instancje dla problemu gniazdowego. Zmieniono go w problem cykliczny przyjmując, że zadania z instancji tworzą pojedynczy MPS. Listę problemów testowych zawiera Tabela 1.

Tab. 1. Instancje testowe dla problemu gniazdowego.

Instancja	liczba zadań	liczba maszyn	liczba operacji	instancja	liczba zadań	liczba maszyn	liczba operacji
la01	10	5	50	la23	15	10	150
la02	10	5	50	la24	15	10	150
la03	10	5	50	la25	15	10	150
la04	10	5	50	la26	20	10	200
la05	10	5	50	la27	20	10	200
la06	15	5	75	la28	20	10	200
la07	15	5	75	la29	20	10	200
la08	15	5	75	la30	20	10	200
la09	15	5	75	la31	30	10	300
la10	15	5	75	la32	30	10	300
la11	20	5	100	la33	30	10	300
la12	20	5	100	la34	30	10	300
la13	20	5	100	la35	30	10	300
la14	20	5	100	la36	15	15	225
la15	20	5	100	la37	15	15	225
la16	10	10	100	la38	15	15	225
la17	10	10	100	la39	15	15	225
la18	10	10	100	la40	15	15	225
la19	10	10	100	ft06	6	6	36
la20	10	10	100	ft10	10	10	100
la21	15	10	150	ft20	20	5	100
la22	15	10	150				

4.2. Parametry

Wstępnie sprawdzono dla jakich długości listy tabu uzyskiwano najlepsze wyniki. Wybrano $L_{max} = 7$. Aby porównać czas wykonywania algorytmu z [16], ustalono liczbę iteracji $Iter_{max}$ na 200 i 800. Dodatkowo wykonano symulacje dla $Iter_{max} = \{3200, 6400, 12800, 50000\}$. Wersję zrównolegloną algorytmu uruchamiano dla ilości wątków $TH = \{2,4,6,8,10,12\}$ i $Iter_{max} = 200$ dla wszystkich instancji testowych. Mierzone były:

- średnie odchylenie względne czasu cyklu; odchylenie zdefiniowano jako:

$$\Delta T = \frac{T_H - T_{LB}}{T_{LB}} \cdot 100\% , \tag{23}$$

  gdzie T_H oznacza czas cyklu obliczony przez algorytm heurystyczny TS, a T_{LB} dolne oszacowanie czasu cyklu;
- średni czas wykonywania algorytmu t_{CPU};
- dla wersji równoległej średnie przyspieszenie:

$$s = \frac{t_{CPU_k}}{t_{CPU_1}}, \tag{24}$$

  gdzie t_{CPU_k} oznacza średni czas wykonywania algorytmu przy wykorzystaniu k wątków;

dla różnych liczby iteracji lub wątków.

4.3. Wyniki

Ze względu na różne założenia dotyczące rozłączności MPS, wartości ΔT nie mogą być bezpośrednio porównywane z wynikami pracy [16]. W pracy [16] dla $H_{*o} = 1$ gwarantowana jest rozłączność MPS, jednak wprowadzone jest dodatkowe ograniczenie $\forall_{i \in O}\ C_i^x \leq S_i^{x+1}$. Co więcej, w pracy tej zastosowano inne oszacowania dolne rozwiązań - skorzystano z literaturowych oszacowań dla pokrewnego problemu gniazdowego. Dla $H_{*o} = 2$ dolne oszacowanie budowano w ten sam sposób co w niniejszej pracy, nie jest jednak gwarantowana rozłączność MPS. Aby móc porównać czasy działania algorytmu zaproponowanego w [16], oszacowano różnicę mocy obliczeniowej procesorów użytych w eksperymentach. Na podstawie porównania wydajności rdzeń-na-rdzeń [22] oszacowano $t'_{CPU} \approx t_{CPU} \cdot 0.25$, gdzie t'_{CPU} oznacza przybliżony czas wykonywania algorytmu z [16] na procesorze użytym w niniejszym eksperymencie.

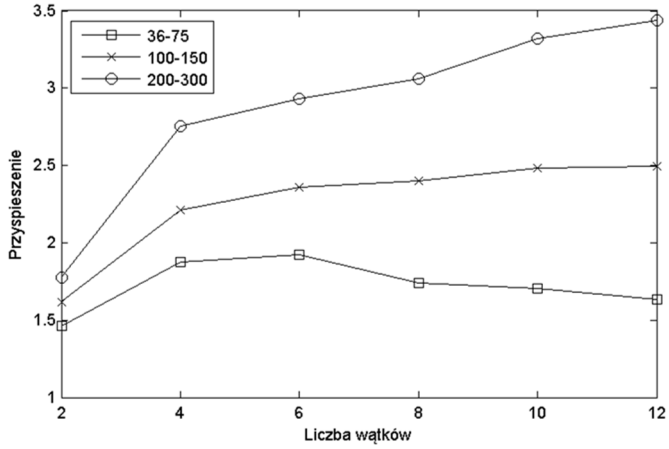
Wersja sekwencyjna algorytmu. Rezultat eksperymentu numerycznego pokazano w Tabeli 2. Algorytm wymagał średnio większej liczby iteracji do uzyskania wyniku zbliżonego do tego z [16]. Dopiero przy $Iter_{max} = 6400$ uzyskano rezultat $\Delta T < 6$. Jednocześnie poszczególne iteracje algorytmu własnego wykonywały się szybciej (również po przeliczeniu czasu na t'_{CPU}). Pozwoliło to na zbadanie zachowania algorytmu dla większych $Iter_{max}$. Zwiększanie $Iter_{max}$ powyżej 12800 powodowało niewielkie poprawienie ΔT. Dla wielu z instancji problemu udało się uzyskać czas cyklu równy dolnemu jego oszacowaniu. W podobnym czasie takie samo zjawisko zaobserwowano w [16].

Tab. 2. Wyniki obliczeń sekwencyjnego algorytmu TS.

$Iter_{max}$	TS		Brucker, Kampmeyer [16], $H_{*o} = 1$			Brucker, Kampmeyer [16], $H_{*o} = 2$		
	ΔT [%]	t_{CPU} [s]	ΔT [%]	t_{CPU} [s]	t'_{CPU} [s]	ΔT [%]	t_{CPU} [s]	t'_{CPU} [s]
200	69.71	6	14.42	317	78.25	12.69	436	109
800	27.57	19	5.83	567	141.75	5.42	623	155.75
3200	12.08	57	-	-	-	-	-	-
6400	5.7	107	-	-	-	-	-	-
12800	4.63	178	-	-	-	-	-	-
50000	4.38	524	-	-	-	-	-	-

Wersja równoległa algorytmu. Dla liczby iteracji $Iter_{max} = 200$ uruchomiono sekwencyjną oraz równoległą wersję algorytmu. Liczbę wątków ograniczono do 12, ze względu na parametry techniczne procesora. Rysunek 3 przedstawia wyniki eksperymentu. Instancje testowe podzielono na 3 grupy według liczby operacji.

Wartość przyspieszenia zależy od liczby sąsiadów w otoczeniu w każdej z iteracji algorytmu TS oraz czasu potrzebnego do wyznaczenia wartości funkcji oceniającej. W większych instancjach testowych otoczenia są przeciętnie większe oraz wyliczanie funkcji oceniającej zabiera więcej czasu. Stąd przyspieszenie dla grupy $(200 - 300)$ jest największe. Spadek przyspieszenia dla większej ilości wątków dla grupy $(36 - 100)$ wynika z narzutu wiążącego się z użyciem równoległej pętli *for* z *OpenMP*.



Rys. 3. Zależność przyspieszenia od liczby wątków dla $\mathrm{Iter}_{\max} = 200$.

5. Wnioski

W pracy przedstawiono równoległy algorytm przeszukiwania z zabronieniami dla cyklicznego problemu gniazdowego. Badania przeprowadzono na literaturowych danych testowych. Zrównoleglona wersja algorytmu TS pozwala na wykonywanie większej liczby iteracji w tym samym czasie, w porównaniu do algorytmu sekwencyjnego, co prowadzi to do uzyskiwania mniejszych czasów cyklu. Czynnikiem ograniczającym przyspieszenie jest wielkość sąsiedztwa. W badanych instancjach problemu, w celu uzyskania większego przyspieszenia należałoby zrównoleglić inną część algorytmu, np. algorytm Bellmana-Forda lub bisekcji. Dla zaproponowanego algorytmu, przy wykorzystaniu większej liczby procesorów, możliwe byłoby zastosowanie innych strategii tworzących większe sąsiedztwa.

Praca została częściowo sfinansowana ze środków Narodowego Centrum Nauki przyznanych na podstawie decyzji numer DEC-2012/05/B/ST7/00102.

Bibliografia

1. Levner E., Kats V, Lopez A.P., Cheng T.C.E. Complexity of cyclic scheduling problems: A state-of-the-art survey. Computers and Industrial Engineering, 59, 2010, 352-361.
2. Kampmeyer T., Cyclic Scheduling Problems, Ph.D. Dissertation, Universitat Osnabruck, 2006.
3. Sawik T., A mixed integer program for cyclic scheduling of flexible flow lines, Bulletin of the Polish Academy of Sciences, Technical Sciences, Vol. 62, No. 1, 2014, 121–128.
4. Sawik T., Batch vs. cyclic scheduling in flexible flow shops by mixed integer programming, International Journal of Production Research 50(18), 2012, 5017–5034.
5. Smutnicki C.: Scheduling with high variety of customized compound products. Decision Making in Manufacturing and Services, 1, 1/2, 2007, 91-110.
6. Smutnicki C., Smutnicki A.: Nowe własności harmonogramów cyklicznych w systemie przepływowym, Automatyka, t. 11, z. 1/2, 2007, 275-285.

7. Smutnicki C., Smutnicki A.: Harmonogramowanie cykliczne w systemie gniazdowym, Zastosowania teorii systemów, AGH, 2007, 105-115.
8. Pempera J., Smutnicki C., Minimalizacja czasu cyklu wytwarzania na linii: Podejście genetyczne z ekspresją genów. Automatyka 2005, t. 9, z. 1/2, 189-199.
9. Dąbrowski P., Pempera J., Smutnicki C.: Minimizing cycle time of the flow line-genetic approach with gene expression, Lecture Notes in Computer Science, vol. 4431, 2007, 194-201.
10. Bożejko W., Gniewkowski Ł., Pempera J., Wodecki M., Cyclic hybrid flow-shop scheduling problem with machine setups, Procedia Computer Science Vol. 29, 2014, 2127–2136.
11. Bożejko W., Kacprzak Ł., Wodecki M., Cykliczny problem przepływowy z przezbrojeniami maszyn, Innowacje w Zarządzaniu i Inżynierii Produkcji (red. R. Knosala), Oficyna Wydawnicza Polskiego Towarzystwa Zarządzania Produkcją, Opole 2014, 484-493.
12. Bożejko W., Wodecki M., Problemy cykliczne z równoległymi maszynami, w: Automatyzacja procesów dyskretnych, Teoria i zastosowania (red. A. Świerniak, J. Krystek), Gliwice 2014, 27-36. ISBN 978-83-62652-67-9.
13. Smutnicki C., Nowicki E.: A fast taboo search algorithm for the job shop problem. Management science, Issue 42.6, 1996, 797-813.
14. Glover F., Laguna M., Tabu Search, Kluwer, 1997.
15. Smutnicki C.: Opracowanie metod efektywnego wyznaczania czasu cyklu dla zadanej kolejności wykonywania zadań w systemie gniazdowym, Raport PREPRINTY nr 78/2013.
16. Brucker P., Kampmeyer T.: Tabu search algorithms for cyclic machine scheduling problems. Journal of Scheduling, Issue 8.4, 2005, 303-322.
17. Dasdan A., Irani S. S., Gupta R. K.: Efficient algorithms for optimum cycle mean and optimum cost to time ratio problems. Proceedings of the 36th annual ACM/IEEE Design Automation Conference, 1999, 37-42.
18. Cormen T. H., Leiserson C. E., Rivest R. L., Clifford, S.: Introduction To Algorithms. MIT Press, 2001, 588-592.
19. Burden R. L., Faires J. D.: Numerical Analysis (9th ed.), Brooks/Cole, Cengage Learning, 2011, 48-54.
20. Beasley J. E.: OR-Library: distributing test problems by electronic mail. Journal of the Operational Research Society, Issue 41(11), 1990, 1069-1072.
21. OR-LIBRARY, dostęp: http://people.brunel.ac.uk/~mastjjb/jeb/info.html [09 01 2015].
22. Geekbench Browser, dostęp: http://browser.primatelabs.com/ [09 01 2015].

Dr hab. Wojciech BOŻEJKO, prof. nadzw.
Mgr inż. Andrzej GNATOWSKI
Wydział Elektroniki, Politechnika Wrocławska
50-370 Wrocław, ul. Wyb. Wyspiańskiego 27,
e-mail: wojciech.bozejko@pwr.wroc.pl
andrzej.gnatowski@pwr.wroc.pl

Dr hab. Mieczysław WODECKI, prof. nadzw.
Instytut Informatyki, Uniwersytet Wrocławski
50-383 Wrocław, ul. Joliot-Curie
e-mail: mwd@ii.uni.wroc.pl