Wojciech BOŻEJKO
Politechnika Wrocławska
Mariusz UCHROŃSKI
Wrocławskie Centrum Sieciowo-Superkomputerowe
Mieczysław WODECKI
Uniwersytet Wrocławski

RÓWNOLEGŁE WYZNACZANIE WZORCÓW W ROZWIĄZYWANIU CYKLICZNEGO PROBLEMU PRZEPŁYWOWEGO Z PRZEZBROJENIAMI

Streszczenie. Tematem pracy jest nowa koncepcja bloków dla cyklicznego problemu przepływowego z przezbrojeniami, wykorzystująca wielokrotne wzorce o różnych rozmiarach wyznaczone dla każdej maszyny będące optymalną kolejności odwiedzania miast w pewnym problemie komiwojażera. Proponujemy wykorzystanie współbieżnego środowiska obliczeń Intel Xeon Phi do szybkiego wyznaczania bloków w oparciu o wzorce, w efekcie znacząco poprawiając jakość otrzymywanych rozwiązań.

PARALLEL PATTERNS DETERMINATION IN SOLVING CYCLIC FLOW SHOP PROBLEM WITH SETUPS

Summary. The subject of the work i the new idea of blocks for cyclic flow shop problem with setup times, using multiply patterns with different sizes determined for each machine being optimal schedule of cities in a traveling salesman problem. We propose to take advantage of Intel Xeon Phi parallel computing environment to fast blocks determination basing on patterns, in effect significantly improving quality of obtained results.

1. Wstep

W ostatnich latach wzrasta zainteresowanie cyklicznymi problemami szeregowania zadań zarówno w środowisku teoretyków zajmujących się zagadnieniami optymalizacji dyskretnej, jak i w środowisku praktyków z przemysłu. Wytwarzanie cykliczne jest bowiem bardzo efektywnym sposobem produkcji dla współczesnych elastycznych systemów produkcyjnych. W literaturze jest wiele opracowań dotyczących różnych aspektów sterowania cyklicznego w przedsiębiorstwach wytwarzających produkty na masową skalę. Spotkać można przykłady zastosowania szeregowania cyklicznego w różnych sferach przemysłu, transportu i logistyki (e.g. Pinto et al. [13], Pinedo [12], Mendez et al. [10], Gertsbakh and Serafini [7], Kats and Levner [9]). Niestety, istniejące modele i narzędzia obliczeniowe pozwalają na wyznaczenie optymalnego (minimalizującego czas cyklu) sterowania dla systemów produkcyjnych wykonujących jedynie niewielką liczbę zadań.

W pracy rozpatrywany jest cykliczny problem przepływowy z czasami przezbrojeń. Silna NP-trudność już wielu najprostszych wersji problemu cyklicznego szeregowania (Smutnicki [6]), a rozpatrywanego problemu w szczególności, ogranicza zakres stosowania algorytmów dokładnych do instancji o małej liczbie zadań, choć w kontekście minimalizacji czasu cyklu konstrukcja i zastosowanie algorytmów dokładnych wydaje się całkowicie uzasadnione (Brucker et al. [5]). Niemniej, ze względu na NP-trudność, do wyznaczania satysfakcjonujących rozwiązań, stosuje się powszechnie szybkie algorytmy przybliżone oparte na technikach przeszukiwań lokalnych, np. symulowane wyżarzanie (w wersji równoległej Bożejko et al. [2]) czy tabu search (Bożejko et al. [3]). Metody tego typu opierają się zazwyczaj na dwupoziomowej dekompozycji problemu: wyznaczenie optymalnej kolejności zadań (poziom górny) oraz wielokrotne wyznaczenie minimalnej wartości kryterium dla danej kolejności zadań (poziom dolny). O ile dla klasycznych, nie-cyklicznych problemów szeregowania rozwiązanie problemu dolnego poziomu można otrzymać w sposób efektywny czasowo przez analizę specyficznego grafu, to w przypadku postawionego problemu rozwiązanie zagadnienia dolnego poziomu jest stosunkowo czasochłonne bowiem, w ogólności, wymaga rozwiązania pewnego zagadnienia programowania liniowego. Stąd wszelkie własności szczególne, w tym pozwalające na bardziej efektywne wyliczanie czasu cyklu, poszukiwanie harmonogramu oraz ograniczenie liczności lokalnie przeglądanego sąsiedztwa lub przyspieszenie szybkości jego przeglądania są bardzo pożądane.

W niniejszej pracy proponujemy wykorzystanie nowych własności eliminacyjnych, tzw. wzorców, do zmniejszenie liczby rozwiązań przeglądanych podczas generowania otoczenia przez algorytmy poszukiwań lokalnych, takie jak przeszukiwanie z zabronieniami czy symulowane wyżarzanie. Wyznaczanie wzorców może być wykonana zarówno sekwencyjne, jak i równolegle, z wykorzystaniem wieloprocesorowego środowiska obliczeń. Odpowiednie własności sformułowane zostały z wykorzystaniem modelu maszyny PRAM, będącej standardem teoretycznej weryfikacji złożoności obliczeniowej algorytmów równoległych.

2. Opis problemu

Rozpatrywany w pracy problem cyklicznego wytwarzania można sformułować następująco: dany jest zbiór n zadań $\mathcal{J} = \{1, 2, \ldots, n\}$, które należy wykonać cyklicznie (w sposób powtarzalny) na maszynach ze zbioru $\mathcal{M} = \{1, 2, \ldots, m\}$. Dowolne zadanie należy wykonać kolejno, na każdej z m maszyn $1, 2, \ldots, m$ (porządek technologiczny). Zadanie $j \in \mathcal{J}$ jest ciągiem m operacji $O_{1,j}, O_{2,j}, \ldots, O_{m,j}$. Operacja $O_{k,j}$ odpowiada czynności wykonywania zadania j na maszynie k, w czasie $p_{k,j}$ ($k = 1, 2, \ldots, m$, $j = 1, 2, \ldots, n$). Po zakończeniu pewnej, a przed rozpoczęciem następnej operacji należy wykonać przezbrojenie maszyny. Niech $s^k_{i,j}$ ($k \in \mathcal{M},\ i \neq j\ i, j \in \mathcal{J}$) będzie czasem przezbrojenia pomiędzy operacją $O_{k,i}$ oraz $O_{k,j}$.

Zbiór zadań wykonywanych w pojedynczym cyklu nazywany jest MPS-em (*ang. minimal part set*). MPS-y są przetwarzane cyklicznie, jeden po drugim.

Należy wyznaczyć kolejność wykonywania zadań (taką samą na każdej maszynie), która minimalizuje czas cyklu, tj. termin rozpoczęcia wykonywania zadań ze zbioru $\mathcal{J}$ w następnym cyklu. Muszą być przy tym spełnione następujące ograniczenia:

(a) każda operacja może być wykonywana tylko przez jedną maszynę,

(b) żadna maszyna nie może wykonywać jednocześnie więcej niż jedną operację,

(c) zachowany musi być porządek technologiczny wykonywania operacji,

(d) wykonywanie żadnej operacji nie może być przerwane przed jej zakończeniem.

(e) każda maszyna, pomiędzy kolejno wykonywanymi operacjami, wymaga przezbrojenia.

(f) każda operacja jest kolejno wykonywana (w następujących po sobie MPS-ach) po upływie czasu cyklu.

Rozpatrywany problem sprowadza się ustalenia momentów rozpoczęcia wykonywania zadań na maszynach spełniających ograniczenia (a)-(f), aby czas cyklu (czas po którym zadanie jest wykonywane w kolejnym MPS-ie) był minimalny.

Zakładamy, że w każdym z MPS-ów, na każdej maszynie, zadania sa wykonywane w takiej samej kolejności. Wobec tego, w cyklicznym harmonogramie, kolejność wykonywania zadań na maszynach może być reprezentowana przez permutację zadań w pierwszym MPS-ie. Bowiem na jej podstawie możemy wyznaczyć momenty rozpoczęcia wykonywania zadań na maszynach w pierwszym MPS-ie. Zwiększając je o wielokrotność czasu cyklu, otrzymamy momenty rozpoczęcia wykonywania zadań w dowolnym z MPS-ów (moment rozpoczęcia wykonywania dowolnej operacji w kolejnym z MPS-ów należy zwiększyć o czas cyklu). Niech Φ będzie zbiorem wszystkich permutacji elementów ze zbioru zadań $\mathcal{J}$. Wobec tego, rozpatrywany w pracy problem sprowadza się do wyznaczenia permutacji zadań (elementu zbioru Φ) minimalizującej długość czasu cyklu. W skrócie problem ten będziemy oznaczali przez CFS (ang. *Cyclic Flow Shop*).

3. Model matematyczny

Niech $[S^k]_{m\times n}$ będzie macierzą terminów rozpoczęcia wykonywania zadań k-tego MPS-a (dla ustalonej kolejności $\pi \in \Phi$), gdzie $S^k_{i,j}$ oznacza termin rozpoczęcia wykonywania zadania j na maszynie i. Zakładamy, że zadania, w kolejnych MPS-ach wykonywane są cyklicznie. Oznacza to, że istnieje stała $T(\pi)$ (okres) taka, że

$$S^{k+1}_{i,\pi(j)} = S^k_{i,\pi(j)} + T(\pi), \;\; i = 1, ..., m, \; j = 1, ..., n, \; k = 1, 2, ... \qquad (1)$$

Okres $T(\pi)$ zależy oczywiście od permutacji π i jest nazywany *czasem cyklu* systemu. Minimalną wartość $T(\pi)$, dla ustalonej π, będziemy nazywać *minimalnym czasem cyklu* i oznaczać przez $T^*(\pi)$. Ponieważ kolejność wykonywania zadań w ramach dowolnego MPS-a jest taka sama, więc wystarczy wyznaczyć kolejność zadań π dla jednego (*pierwszego*) MPS-a i dokonać jego przesunięcia o wielkość $k \cdot T(\pi)$, k=1,2,... na osi czasu. Dla ustalonej kolejności wykonywania zadań $\pi \in \Phi$, optymalną wartość czasu cyklu $T^*(\pi)$ można wyznaczyć rozwiązując odpowoiednie zadanie programowania liniowego (zobacz Bożejko i in. [1]).

Dla dowolnej kolejności wykonywania zadań w ramach pierwszego MPS-a, rozwiązując powyższe zadanie programowania liniowego, można w czasie wielomianowym wyznaczyć minimalną wartość czasu cyklu. W przypadku algorytmu dokładnego (przegląd zupełny) rozwiązania problemu CFS należy więc to wykonać dla każdej z $n!$ permutacji - elementu zbioru Φ. W następnym rozdziale przedstawiamy metodę przybliżoną rozwiązania rozpatrywanego w tej pracy problemu.

4. Metoda rozwiązania

W wielu algorytmach heurystycznych rozwiązywania problemów NP-trudnych są przeglądane otoczenia, tj. podzbiory przestrzeni rozwiązań. W przypadku, gdy rozwiązaniami problemu są permutacje, zazwyczaj otoczenia są generowane przez ruchy typu insert lub swap oraz ich złożenia [4]. Polegają one na zmianie pozycji elementów w permutacji. Liczba elementów takiego otoczenia wynosi co najmniej $n(n-1)/2$, gdzie n jest rozmiarem danych. W praktycznych zastosowaniach (przy dużych n), przeglądanie otoczenia jest najbardziej czasochłonnym elementem algorytmu. Z opisu, zamieszczonych w literaturze eksperymentów obliczeniowych wynika, że liczba iteracji algorytmu ma bezpośredni wpływ na jakość wyznaczanych rozwiązań. Stąd, poszukiwanie metod przyspieszających działanie pojedynczej iteracji algorytmu. Jedną z nich jest zmniejszenie liczby elementów otoczenia oraz równoległe ich generowanie i przeglądanie. W przypadku problemów szeregowania zadań na wielu maszynach z minimalizacją czasu wykonywania zadań (C_{max}) są w tym celu z powodzeniem stosowane "własności eliminacyjne bloków" [8]. Podobne własności zastosujemy w algorytmie rozwiązywania problemu wyznaczania minimalnego czasy cyklu, a dokładniej - minimalnego czasu pracy pojedynczej maszyny. Umożliwiają one eliminację elementów z otoczenia, które bezpośrednio nie dają poprawy najlepszego do tej pory znalezionego rozwiązania.

W pracy [3] przedstawiono metodę rozwiązania problemu CFS oraz algorytm sekwencyjny przeszukiwania z zabronieniami. W dalszej części rozdziału przedstawimy w skrócie główne elementy metody.

Dla ustalonej permutacji $\pi \in \Phi$ oraz maszyny $k \in \Phi$

$$T_k(\pi) = \sum_{i=1}^{n-1} (p_{k,\pi(i)} + s^k_{\pi(i),\pi(i+1)}) + p_{k,\pi(n)} + s^k_{\pi(n),\pi(1)} \tag{2}$$

jest czasem wykonywania zadań w kolejności π, wraz z przezbrojeniem pomiędzy zadaniem $\pi(n)$, a $\pi(1)$ (tj. zadaniem ostatnim w danym MPS-ie, a pierwszym w następnym). Można łatwo udowodnić, że minimalny czas cyklu

$$T^*(\pi) = \min\{T_i(\pi) : \ i = 1, 2, \ldots, m\}. \tag{3}$$

Własność 1. *([3]). Warunkiem koniecznym zmniejszenia wartości minimalnego czasu cyklu $T^*(\beta)$ jest skrócenie czasu pracy k-tej maszyny, tj. zmniejszenie $T_k(\beta)$, gdzie $T^*(\beta) = T_k(\beta)$.*

Wyznaczenie minimalnego czasu pracy k-tej maszyny, tj. wartości $\min\{T_k(\delta) : \delta \in \Phi\}$ można sprowadzić do następującego problemu komiwojażera.

Niech $H_k = (\mathcal{V}, \mathcal{E}; p, s)$ będzie grafem pełnym, gdzie

- zbiór wierzchołków: $\mathcal{V} = \mathcal{J}$,
- zbiór krawędzi: $\mathcal{E} = \{(v, u) : \ v \neq u, \ v, u \in \mathcal{V}\}$,
- wagi wierzchołków: $p(v) = p_{k,v}, \ v \in \mathcal{V}$,
- wagi krawędzi: $s(e) = s^k_e, \ e \in \mathcal{E}$.

Własność 2. *([3]) Czas pracy k-tej maszyny $T_k(\pi)$ jest równy długości (tj. sumie wag wierzchołków i krawędzi) cyklu Hamiltona $(\pi(1), \pi(2), \ldots, \pi(n), \pi(1))$ w grafie H_k.*

Własność 3. *([3]) Minimalny czas pracy k-tej maszyny jest równy długości drogi komiwojażera w grafie H_k, tj. minimalnego (ze względu na długość) cyklu Hamiltona.*

Niech π_k^* będzie optymalną drogą komiwojażera w grafie H_k $(k = 1, 2, \ldots, m)$. Jest to optymalna (tj. minimalna ze względu na czas wykonywania) kolejność zadań ze zbioru $\mathcal{J}$ na k-tej maszynie. Permutację tą będziemy nazywali *wzorcem* dla *k*-tej maszyny.

Aby więc zmniejszyć czas pracy k-tej maszyny $T_k(\pi)$ będziemy generowali z π permutacje uwzględniając występowanie poszczególnych elementów we wzorcu. Wzorce umożliwiają także eliminowanie elementów z otoczenia, które nie dają poprawy bieżącej wartości czasu cyklu w algorytmach lokalnych poszukiwań.

4.1. Bloki zadań

Niech

$$B = (\pi(a), \pi(a+1), \ldots, \pi(b)), \tag{4}$$

będzie ciągiem bezpośrednio występujących po sobie zadań w permutacji $\pi \in \Phi$, π_k^* *wzorcem* dla *k*-tej maszyny oraz u, v $(u \neq v,\ 1 \leqslant u, v \leqslant n)$ parą liczb takich, że:

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

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

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

Jeżeli ciąg zadań (4) spełnia warunki **W1** i **W3** lub **W2** i **W3**, to nazywamy go *blokiem* na *k*-tej maszynie ($k \in \mathcal{M}$).

Poniżej przedstawiamy algorytm sekwencyjny wyznaczania wszystkich bloków w permutacji.

Algorytm *SeqBlock*

$\pi = (\pi(1), \pi(1), \ldots, \pi(n))$ - permutacja;
$\pi^* = (\pi^*(1), \pi^*(1), \ldots, \pi^*(n))$ - wzorzec permutacji π;
t - liczba bloków;
$(b_1, b_2, \ldots, b_t)$ - wektor pozycji początkowych bloków w π;
$t \leftarrow 1$; $i \leftarrow 1$;
while $(i \leqslant n)$ **do**
  $b_t \leftarrow i$; $q \leftarrow (\pi^*)^{-1}(\pi(i))$;
  while $(\pi(i) = \pi^*(i))$ **do**
    $q \leftarrow q + 1$; $i \leftarrow i + 1$;
    $i \leftarrow i + 1$;

Złożoność obliczeniowa algorytmu wynosi $O(n)$.

Wyznaczanie wzorca (optymalnej drogi komiwojażera w grafie H_k) jest problemem NP-trudnym. Dlatego będziemy stosowali algorytmy przybliżone, np. 2-opt. Dla każdej maszyny wzorzec należy wyznaczyć raz, przed uruchomieniem właściwego algorytmu.

4.2. Równoległe wyznaczanie bloków

Aby przyspieszyć działanie algorytmu wyznaczania minimalnego czasu cyklu przedstawiamy metodę zrównoleglenia najbardziej czasochłonnej, wykonywanej w każdej iteracji, procedury wyznaczania bloków.

Własność 4. *Wyznaczenie bloków dla cyklicznego problemu przepływowego z przezbrojeniami można wykonać w czasie* $O(\log n)$ *na* mn*-procesorowej maszynie CREW PRAM.*

Na Rys. 1 przedstawiono algorytm `pblocks` wyznaczania bloków za pomocą n procesorów z wykorzystaniem biblioteki OpenMP. Algorytm ten został następnie uruchomiony na koprocesorze Intel Xeon Phi. Danymi wejściowymi są: rozmiar permutacji w której wyznaczane są bloki `n`, permutacja `pi` oraz permutacja – wzorzec `pi_ptr`. Tablice `b_b` oraz `b_e` zawierają po zakończeniu procedury pozycje początków i końców kolejnych bloków.

```
int pblocks(int n, int *pi, int *pi_ptr,
    int *b_b, int *b_e) {
  int *pi_ptr_ = new int[n+2];
  int *b = new int[n+1];
  int *b_ = new int[n+1];
  int *bk = new int[n+1];
  int *p = new int[n+2];
  int *p_ = new int[n+2];
  pi_ptr[0] = pi_ptr[n+1] = -1;
  zeros_pi(n, pi_ptr_);
  pi_ptr_[0] = pi_ptr_[n+1] = -1;
  #pragma omp parallel for
  for(int i=1; i<=n; ++i)
    pi_ptr_[pi_ptr[i]] = i;
  #pragma omp parallel for
  for(int i=1; i<=n; ++i)
  {
    if(pi[i-1]==pi_ptr[pi_ptr_[pi[i]] - 1])
      b[i] = 1; else b[i] = 0;
    if(pi[i+1]==pi_ptr[pi_ptr_[pi[i]] + 1])
      bk[i] = 1; else bk[i] = 0;
    p[i] = p_[i] = 0;
  }
  p[1] = b[1]; p[0] = 0;
  #pragma omp parallel for
  for(int i=1; i<=n; ++i)
    if(b[i] == 0 and b[i+1] == 1)
      b[i] = 1; else b[i] = 0;
  #pragma omp parallel for
  for(int i=n; i>=1; --i)
    if(bk[i] == 0 and bk[i-1] == 1)
      bk[i] = 1; else bk[i] = 0;
  for(int i=2; i<=n; ++i)
    p[i] = p[i-1] + b[i];
  int nblocks = 0;
  #pragma omp parallel for
  for(int i=1; i<=n; ++i)
    if(b[i] == 1)
      b_b[p[i]] = i;
    if(bk[i] == 1)
      b_e[p[i]] = i;
  #pragma omp parallel for
  for(int i=1; i<=n; ++i)
  {
    #pragma omp critical
    if(b_b[i] != 0 or b_e[i] != 0)
      nblocks++;
  }
  #pragma omp parallel for
  for(int i=1; i<=nblocks; ++i)
    if(b_b[i] == 0 and b_e[i] != 0)
      b_b[i] = 1;
    if(b_e[i] == 0 and b_b[i] != 0)
      b_e[i] = n;

  delete[] pi_ptr_; delete[] b; delete[] b_;
  delete[] p; delete[] p_;
  return nblocks;
}
```

Rys. 1. Algorytm równoległego wyznaczania bloków.

5. Eksperymenty obliczeniowe

Równoległa procedura wyznaczania bloków została zaimplementowana w języku C++ z wykorzystaniem biblioteki OpenMP. Dane na potrzeby eksperymentów obliczeniowych (permutacje zadań) zostały wygenerowane losowo. Liczba elementów permutacji zmieniała się w zakresie od 10^3, aż do 10^7. Eksperymenty obliczeniowe zostały

przeprowadzone środowisku obliczeniowym z pamięcią współdzieloną – koprocesorze Intel Xeon Phi 3120A (6GB, 1.1 GHz) pozwalającym na wykorzystanie 228 rdzeni.

Pierwsza faza eksperymentów miała na celu wykazanie efektywności działania (sekwencyjnego) mechanizmu bloków opartych na wzorcach. W tym celu procedurę wyznaczania bloków umieszczono w klasycznym algorytmie przeszukiwania z zabronieniami (*tabu search*, TS), z listą tabu o długości 7. Uruchomiono dwie wersje algorytmu - z blokami oraz bez, dla ustalonej liczby 1000 iteracji. Badano procentowy błąd względny (*Percentage Relative Deviation*, PRD) do rozwiązania referencyjnego otrzymanego przy pomocy algorytmu NEH (Nawaz i in. [11]) dla danych testowych pochodzących z pracy [14]. Wyniki zamieszczone w Tabeli 1 wskazują, że przy prawie dwukrotnie krótszym czasie działania algorytmu otrzymano wyniki znacząco lepsze (32,8% poprawy względem NEH) w porównaniu do wersji algorytmu bez mechanizmu bloków (29,0% poprawy względem NEH).

Tabela 1
Porównanie PRD do NEH dla algorytmu TS z blokami oraz bez - 1000 iteracji.

$n \times m$	$t[s]$	$t_B[s]$	PRD	PRD_B
20×5	2,2	0,8	-30,0	-32,7
20×10	3,0	0,9	-29,5	-31,6
20×20	4,7	1,2	-29,8	-30,7
50×5	35,7	17,7	-32,5	-34,1
50×10	48,4	22,0	-29,6	-34,5
50×20	73,9	28,3	-28,3	-31,8
100×5	292,1	181,8	-30,7	-36,7
100×10	391,9	203,8	-28,1	-33,6
100×20	604,7	266,4	-27,9	-31,6
200×10	3212,2	1791,1	-26,7	-32,9
200×20	5255,5	2497,7	-26,2	-31,1
Średnio	902,2	455,6	-29,0	-32,8

Tabela 2
Przyspieszenie procedury `pblocks`.

$\lambda^{(1)}$	$p = 16$	$p = 32$	$p = 64$	$p = 128$
1	0,001	0,001	0,001	0,001
2	0,003	0,002	0,002	0,001
5	0,007	0,006	0,006	0,004
10	0,015	0,013	0,012	0,008
20	0,033	0,029	0,023	0,019
50	0,129	0,129	0,100	0,073
100	0,400	0,347	0,261	0,195
200	0,988	0,884	0,697	0,507
500	2,234	2,222	1,799	1,285
1000	3,360	3,552	2,967	2,278
2000	4,679	5,307	4,961	3,853
5000	6,306	8,270	8,154	7,043
10000	7,091	9,766	10,775	9,679

(1) $\lambda = n \cdot 10^3$, koprocesor Intel Xeon Phi 3120A

Druga faza eksperymentów polegała na zmierzeniu przyspieszenia procedury równoległego wyznaczania bloków. Wyniki eksperymentów obliczeniowych dla koprocesora Intel Xeon Phi 3120A zostały zamieszczone w tabeli 2. Dla różnej liczby zadań w permutacji przyspieszenie początkowo dość szybko wzrasta, osiąga maksimum, a następnie powoli zmniejsza się. Liczba procesorów dla którego osiągane jest maksimum przyspieszenia zależy od rozmiaru problemu. Posługując się pojęciem *skalowalności* algorytmów równoległych można powiedzieć, że dla $5000 \cdot 10^3$ zadań w permutacji równoległa metoda wyznaczania bloków charakteryzuje się dla $p = 1 \cdots 32$ silną skalowalnością, ponieważ wraz ze wzrostem liczby procesorów wzrasta przyspieszenie.

6. Wnioski i uwagi

W pracy przedstawiono nową koncepcję bloków dla cyklicznego problemu przepływowego z przezbrojeniami maszyn, wykorzystującą wielokrotne wzorce o różnych rozmiarach. Wzorce reprezentują optymalną kolejności odwiedzania miast w pewnym problemie komiwojażera, zapewniając własności blokowe problemu. Zastosowanie tych własności blokowych pozwala na znaczną redukcję liczby przeglądanych sąsiadów w metaheurystycznych algorytmach opartych na przeglądaniu otoczeń.

LITERATURA

1. Bożejko W., Uchroński M., Wodecki M. (2016). Parallel metaheuristics for the cyclic flow shop scheduling problem. Computers & Industrial Engineering, 95, 156–163.
2. Bożejko, W., Pempera, J., Wodecki, M. (2015). Parallel Simulated Annealing Algorithm for Cyclic Flexible Job Shop Scheduling Problem. Lecture Notes in Artificial Intelligence No. 9120, Springer, 603–612.
3. Bożejko, W., Uchroński, M., Wodecki, M. (2015). Block approach to the cyclic flow shop scheduling. Computers & Industrial Engineering, 81, 158–166.
4. Bożejko, W., Wodecki, M. (2007). On the theoretical properties of swap multimoves. Operations Research Letters, 35(2), 227–231.
5. Brucker, P., Burke, E. K., Groenemeyer, S. (2012). A branch and bound algorithm for the cyclic job-shop problem with transportation. Computers & Operations Research, 39, 12, 3200–3214.
6. Smutnicki, C., New features of the cyclic job shop scheduling problem. In Proceedings of 20th International Conference on Methods and Models in Automation and Robotics MMAR 2015, IEEE Press, 1000–1005.
7. Gertsbakh, I., Serafini, P. (1991). Periodic transportation schedules with flexible departure times. European Journal of Operational Research, 50, 298—309.
8. Grabowski, J., Wodecki, M. (2004). A very fast tabu search algorithm for the permutation flow shop problem with makespan criterion. Computers & Operations Research, 31, 1891–1909.
9. Kats, V., Levner, E. (2010). A fast algorithm for a cyclic scheduling problem with interval data. In Proceedings of the annual operations research society of Israel (ORSIS-2010) conference, February 2010, Nir Etzion, Israel.
10. Mendez, C. A., Cerda, J., Grossmann, I. E., Harjunkoski, I., Fahl, M. (2006). State-ofthe-art review of optimization methods for short-term scheduling of batch processes. Computers and Chemical Engineering, 30, 913—946.
11. Nawaz, M., Enscore, Jr, E.E., Ham, I. (1983). A heuristic algorithm for the m–machine, n–job flow–shop sequencing problem. OMEGA International Journal of Management Science, 11, 91–95.
12. Pinedo, M. (2005). Planning and scheduling in manufacturing and services. New York: Springer.
13. Pinnto, T., Barbosa-Povoa, A. P. F. D., Novais, A. Q. (2005). Optimal design and retrofit of batch plants with a periodic mode of operation. Computers and Chemical Engineering, 29, 1293–1303.
14. Ruiz, R., Stützle, T. (2008). 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), 1143–1159.