# RÓWNOLEGŁY ALGORYTM POPULACYJNY DLA PROBLEMU GNIAZDOWEGO Z RÓWNOLEGŁYMI MASZYNAMI

**Wojciech BOŻEJKO, Mariusz UCHROŃSKI, Mieczysław WODECKI**

**Streszczenie:** W pracy rozpatrywany jest ogólny problem kolejnościowy z równoległymi maszynami (ang. *flexible job shop problem*). W problemie tym dany jest zbiór zadań oraz zbiór maszyn. Maszyny tego samego typu, tj. o tych samych własnościach funkcjonalnych, które jednak mogą mieć różne parametry techniczne takie jak na przykład wydajność, tworzą gniazdo. Każde zadanie należy wykonać na jednej z maszyn ustalonego typu (gniazda). Dla rozpatrywanego problemu w pracy zaproponowano równoległy algorytm populacyjny.[1]

**Słowa kluczowe:** meta heurystyka, algorytm równoległy, problem gniazdowy, równoległe maszyny.

## 1. Wstęp

Ogólny problem kolejnościowy z równoległymi maszynami polega na przydzieleniu maszyn zadaniom i wyznaczeniu kolejności wykonywania zadań na każdej maszynie, aby zminimalizować czas wykonywania wszystkich zadań ($C_{max}$). Spełnione muszą być przy tym następujące ograniczenia:

- każde zadanie może być wykonywane jednocześnie tylko na jednej, odpowiedniego typu, maszynie,
- żadna maszyna nie może wykonywać jednocześnie więcej niż jedno zadanie,
- wykonywanie zadania nie może być przerwane,
- zachowany musi być porządek technologiczny wykonywania zadań.

Rozpatrywany problem należy on do klasy problemów silnie NP-trudnych, bowiem jest uogólnieniem klasycznego problemu gniazdowego (*job shop*, [3]). Algorytmy dokładne rozwiązywania rozpatrywanego zagadnienia zostały przedstawione w pracach [1] i [7], ale pozwalają one rozwiązać zadania o nie więcej niż 20 zadaniach i 10 maszynach. W literaturze proponuje się także sporo meta heurystyk. Hurink [4] oraz Mastrolilli i Gambardella [5] zaproponowali metodę poszukiwania z zabronieniami (*tabu serach*) dla rozważanego problemu. Gao i in. [2] proponują metody algorytmu genetycznego oraz poszukiwania za zmiennym otoczeniem (*variable neighborhood search*).

W niniejszej pracy proponujemy zastosowanie podejścia dwupoziomowego do rozwiązywania problemu gniazdowego z równoległymi maszynami. Wyższy poziom odpowiedzialny jest za przydział operacji do maszyn, a wykonywany za pomocą algorytmu populacyjnego. Niższy poziom to meta heurystyka znajdująca dobre rozwiązanie problemu gniazdowego powstałego po przypisaniu operacjom maszyn. Na tym poziomie zastosowano wydajny algorytm TSAB Nowickiego i Smutnickiego [6].

[1] *Praca częściowo finansowana z projektu badawczego MNiSW Nr N N514 232237.*

## 2. Sformułowanie problemu

Ogólny problem kolejnościowy z równoległymi maszynami, oznaczany w literaturze przez $FJ_m||C_{\max}$, można sformułować następująco: dany jest zbiór zadań $J = \{1,2,\ldots,n\}$, które należy wykonać na maszynach ze zbioru $M = \{1, 2,\ldots, m\}$. Istnieje rozbicie zbioru maszyn na typy, tj. podzbiory maszyn o tych samych własnościach funkcjonalnych. Zadanie jest ciągiem pewnych operacji. Każdą operację należy wykonać na odpowiedniego typu maszynie w ustalonym czasie. Problem polega na przydzieleniu zadań do maszyn z odpowiedniego typu oraz na wyznaczeniu kolejności wykonywania operacji na maszynach, aby zminimalizować czas wykonania wszystkich zadań. Muszą być przy tym spełnione ograniczenia (*i*-*iv*).

Niech $O = \{1,2,\ldots,o\}$ będzie zbiorem wszystkich operacji. Zbiór ten można rozbić na ciągi odpowiadające zadaniom, przy czym zadanie $j \in J$ jest ciągiem $o_j$ operacji, które będą kolejno wykonywane na odpowiednich maszynach (tj. w ciągu technologicznym). Operacje te *są* indeksowane liczbami $(l_{j-1}+1,\ldots, l_{j-1}+o_j)$, gdzie $l_j$ jest liczbą operacji pierwszych $j$ zadań, $j = 1, 2,\ldots, n$, przy czym $l_0 = 0$.

Zbiór maszyn $M = \{1, 2,\ldots, m\}$ można rozbić na $q$ podzbiorów maszyn tego samego typu (*gniazd*), przy czym $i$-ty ($i = 1, 2,\ldots, q$) typ $M_i$ zawiera $m_i$ maszyn, które *są* indeksowane liczbami $(t_{i-1}+1,\ldots,t_{i-1} + m_i)$, gdzie $t_i$ jest liczbą maszyn w pierwszych $i$ typach, $i = 1,2,\ldots,q$, przy czym $t_0 = 0$.

Operację $v \in O$ należy wykonać w gnieździe $\mu(v)$, tj. na jednej z maszyn zbioru $M^{\mu(v)}$ w czasie $p_{v,j}$, gdzie $j \in M^{\mu(v)}$.

Niech

$$O^k = \{ v \in O : \mu(v) = k \}$$

będzie zbiorem operacji wykonywanych w $k$-tym ($k = 1, 2,\ldots, q$) gnieździe. Ciąg zbiorów operacji

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

takich, że dla każdego $k = 1,2,\ldots,q$

$$O^k = \bigcup_{i=t_{k-1}+1}^{t_{k-1}+m_k} Q^i \quad \text{oraz} \quad Q^i \cap Q^j = \emptyset,\ i \neq j,\ i,j = 1,2,\ldots,m,$$

nazywamy *przydziałem operacji zbioru O do maszyn ze zbioru M* (w skrócie **przydziałem operacji do maszyn**).

Ciąg $[Q^{t_{k-1}+1}, Q^{t_{k-1}+2}, \ldots, Q^{t_{k-1}+m_k}]$ jest **przydziałem** maszynom operacji w $i$-tym gnieździe (w skrócie *przydziałem* w $i$-tym gnieździe). W szczególnym przypadku maszyna może nie wykonywać żadnej operacji i wówczas w przydziale operacji w gnieździe zbiór operacji do wykonywania przez tę maszynę jest pusty.

## 2. Metoda rozwiązania

Ze względu na liczbę operacji, możliwych przydziałów operacji do maszyn jest wykładnicza ilość. Każdy dopuszczalny przydział generuje pewien NP-trudny problem gniazdowy (*job shop*), którego rozwiązanie sprowadza się do wyznaczenia optymalnych kolejności wykonywania operacji na maszynach. Wobec tego optymalne rozwiązanie problemu gniazdowego z równoległymi maszynami wymaga rozwiązania wykładniczej liczby NP-trudnych problemów. Dlatego też będziemy stosowali algorytm przybliżony sprowadzający się do iteracyjnego wykonywania następujących dwóch kroków:

**Krok 1:** Wyznaczenie przydziału operacji do maszyn.
**Krok 2:** Rozwiązanie problemu gniazdowego dla przydziału wyznaczonego w kroku 1.

W Kroku 1 będzie stosowany algorytm oparty na idei doskonalenia populacji. Każdy osobnik reprezentować będzie przydział operacji do maszyn, a jego funkcją przystosowania będzie wartość rozwiązania problemu gniazdowego generowanego przez dany przydział operacji do maszyn, a rozwiązany w Kroku 2 za pomocą algorytmu poszukiwania z zabronieniami TSAB [6].

## 3. Równoległy algorytm populacyjny

Zaproponowany w pracy algorytm rozwiązujący problem gniazdowy z równoległymi maszynami rozpoczyna swoje działanie od wygenerowania populacji startowej przydziałów operacji do maszyn. Populacja ta generowana jest losowo. Następnie dla każdego elementu w populacji zostaje uruchomiony algorytm lokalnych poszukiwań, który dostarcza zbioru minimów lokalnych. W kolejnym kroku ustala się pozycje, na których elementy występujące na tych samych pozycjach w wielu minimach lokalnych, tworząc zbiór elementów i pozycji ustalonych. Nowa populacja (zbiór przydziałów operacji do maszyn) jest generowana w taki sposób, że elementy na ustalonych pozycjach pozostają bez zmian. Aby zapobiec zakończenia działania algorytmu w sytuacji gdy wszystkie pozycje są ustalone i niemożliwe jest losowe wygenerowanie nowej populacji, w każdej iteracji „najstarszy" ustalony element i pozycja zostają usunięte ze zbioru elementów i pozycji ustalonych. Poniżej przedstawiony został ogólny schemat algorytmu.

**Algorytm 1. Równoległy algorytm populacyjny PM$^2$h (*parallel population-based meta-square-heuristics*)**

**Krok 0.** Wygeneruj losową populację startową $P_0$ przydziałów operacji do maszyn;

**Krok 1.** Podziel $P_i$ na $k = \left\lceil \frac{|P_i|}{p} \right\rceil$ grup;

Każda grupa zawiera co najwyżej *p* elementów;

**Krok 2.** Dla każdej grupy $k$ (wykorzystując p procesorów) znajdź uszeregowanie zadań odpowiadające przydziałowi operacji do maszyn $Y \in P_i$ oraz wartość $C_{\max}(Y, \pi(Y))$;

**Krok 3.** Znajdź przydział operacji do maszyn $Z \in P_i$ dla którego $C_{\max}(Z, \pi(Z)) = \min\{C_{\max}(Y, \pi(Y)) : Y \in P_i\}$;

**Krok 4.** **Jeśli** $C_{\max}(Z, \pi(Z)) < C_{\max}(Q^*, \pi(Q^*))$ **to**

$\pi(Q^*) = \pi(Z)$;

$Q^* = Z$;

**Krok 5.** Dla każdej pozycji w populacji $P_i$ znajdź liczbę przydziałów operacji do maszyn $v$ w których maszyna m jest na pozycji $l$;

**Krok 6.** Ustal pozycje w populacji $P_i$ dla których $\frac{v}{|P_i|} > \alpha$;

**Krok 7.** Umieść losowo wybrane elementy (maszyny) na wolnych pozycjach;
$P_{i+1} \leftarrow P_i$

**Krok 8.** Jeśli spełniony jest warunek zakończenia to STOP;
W przeciwnym wypadku idź do Kroku 1;

W proponowanym algorytmie populacyjnym startowa populacja jest generowana losowo. Rozmiar generowanej populacji jest równy 100. W pierwszym kroku przedstawionego wyżej algorytmu populacja jest dzielona na zbiory rozłączne.

$$\bigcup_{i=1}^{k} P_i = P, \quad \bigcap_{i=1}^{k} P_i = \emptyset.$$

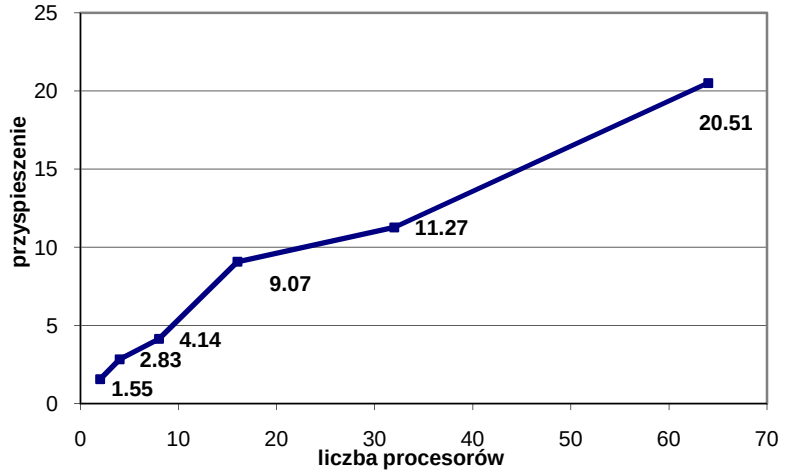
Liczba grup, na które dzielona jest populacja zależy od liczby użytych procesorów. Każdy element populacji stanowi przydział operacji do maszyn, dla którego uruchamiany jest algorytm poszukiwań lokalnych. Do wyznaczenia zbioru minimów lokalnych został wykorzystany algorytm TSAB [6] rozwiązujący klasyczny problem gniazdowy.

## 4. Eksperymenty obliczeniowe

Równoległy algorytm populacyjny dla problemu gniazdowego z równoległymi maszynami został zaimplementowany w języku C++ z wykorzystaniem biblioteki MPI i uruchomiony na klastrze obliczeniowym Nova w Wrocławskim Centrum Sieciowo Superkomputerowym. Algorytm był uruchamiany pod kontrolą systemu kolejkowego OpenPBS. Obliczenia zostały wykonane dla różnej liczby procesorów (2, 4, 8 16, 32, 64). Na podstawie uzyskanych wyników, wyznaczone zostało przyspieszenie względne algorytmu (*orthodox speedup*) z następującej zależności $s = t_s / t_p$, gdzie $t_s$ oznacza czas obliczeń dla algorytmu sekwencyjnego, a $t_p$ - czas obliczeń dla algorytmu równoległego. Dla przyspieszenia wyznaczanego w taki sposób porównane zostały czasy działania algorytmu sekwencyjnego i równoległego uruchamianych na tym samym sprzęcie. Na Rysunku 1 została przedstawiona zależność przyspieszenia algorytmu równoległego od liczby procesorów używanych przez ten algorytm. Algorytm został przetestowany dla 15 instancji testowych zaproponowanych w pracy [5]. W Tabeli 1 zostały zamieszczone wyniki eksperymentów obliczeniowych.

Tab. 1. Wyniki przeprowadzonych eksperymentów.

| problem | O | (LB,UB) | MG [5] | $PM^2h$ |
|---|---|---|---|---|
| la01 | 50 | (570,574) | 571 | 572 |
| la02 | 50 | (529,532) | 530 | 530 |
| la03 | 50 | (477,479) | 478 | 479 |
| la04 | 50 | (502,504) | 502 | 504 |
| la05 | 50 | (457,458) | 457 | 459 |
| la06 | 75 | (799,800) | 799 | 799 |
| la07 | 75 | (749,750) | 750 | 750 |
| la08 | 75 | (765,767) | 765 | 765 |
| la09 | 75 | (853,854) | 853 | 854 |
| la10 | 75 | (804,805) | 804 | 805 |
| la11 | 100 | (1071,107) | 1071 | 1071 |
| la12 | 100 | 936 | 936 | 936 |
| la13 | 100 | 1038 | 1038 | 1038 |
| la14 | 100 | 1070 | 1070 | 1070 |
| la15 | 100 | (1089,1090) | 1090 | 1090 |




Rys. 1. Zależność przyspieszenia algorytmu od liczby użytych procesorów

Dla porównania jakości rozwiązań dostarczonych przez równoległy algorytm populacyjny ($PM^2h$) w Tabeli 1 zostały umieszczone rozwiązania uzyskane przez algorytm MG zaproponowany w pracy [5]. Otrzymane wyniki są porównywalnej jakość. Należy podkreślić, że celem niniejszej pracy nie było stworzenie nowego lepszego algorytmu sekwencyjnego, ale jak najefektywniejsze jego zrównoleglenie (uzyskanie znaczącego przyspieszenia obliczeń). Zależność uzyskanego przyspieszenia od liczby użytych

procesorów prezentuje Rysunek 1. Dla 64 procesorów uzyskano ponad 20-krotne skrócenie czasu obliczeń, przy liniowo rosnącym trendzie uzyskiwanych przyspieszeń.

**Wnioski**

W pracy został zaproponowany równoległy algorytm populacyjny dla problemu gniazdowego z równoległymi maszynami. Z przeprowadzonych eksperymentów wynika, że zastosowanie algorytmu równoległego pozwala na 20 – krotne przyspieszenie obliczeń w stosunku do algorytmu sekwencyjnego.

**Literatura**


1. Adrabiński A., Wodecki M., An algorithm for solving the machine sequencing problem with parallel machines, Zastosowania Matematyki XVI, 3, 1979, 513-541.
2. Gao J., Sun L., Gen M., A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems, Computers & Operations Research 35, 2008, 2892-2907.
3. Garey M.R., Johnson D.S., Sethi R., The complexity of the flowshop and jobshop scheduling, Math. Oper. Res. 1 (2), 1974, 117-128.
4. Hurink E., Jurisch B., Thole M., Tabu search for the job shop scheduling problem with multi-purpose machine, Operations Research Spektrum 15, 1994, 205-215.
5. Mastrolilli M., Gambardella L.M., Effective neighborhood functions for the flexible job shop problem, Journal of Scheduling 3(1), 2000, 3-20.
6. Nowicki E., Smutnicki C., A fast tabu search algorithm for the permutation flow-shop problem, European Journal of Operational Research 91, 1996, 160-175.
7. Pinedo M., Scheduling: theory, algorithms and systems, Englewood cliffs, NJ: Prentice-Hall, 2002.



Dr Wojciech BOŻEJKO
Mgr inż. Mariusz UCHROŃSKI
Instytut Informatyki, Automatyki i Robotyki
Politechnika Wrocławska
e-mail: wojciech.bozejko@pwr.wroc.pl
mariusz.uchronski@pwr.wroc.pl

Dr Mieczysław WODECKI
Instytut Informatyki
Uniwersytet Wrocławski
email: mwd@ii.uni.wroc.pl