AUTOMATYKA • 2009 • Tom 13 • Zeszyt 2

Wojciech Bożejko*, Mariusz Uchroński*, Mieczysław Wodecki**

Równoległa meta^{2}heurystyka dla problemu gniazdowego z równoległymi maszynami

1. Wprowadzenie

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 tum następujące ograniczenia:

(i) każde zadanie może być wykonywane jednocześnie tylko na jednej maszynie określonego dla niego typu,
(ii) żadna maszyna nie może wykonywać jednocześnie więcej niż jedno zadanie,
(iii) wykonywanie zadania nie może być przerwane,
(iv) zachowany musi być porządek technologiczny wykonywania zadań.

Rozpatrywany problem należy do klasy problemów silnie NP-trudnych (jest rozszerzeniem klasycznego problem gniazdowego). Algorytmy dokładne bazujące na dysjunktywnej reprezentacji grafowej rozwiązania były proponowane w literaturze (Adrabiński i Wodecki [1], Pinedo [14]), ale są one efektywne dla instancji problemu z więcej niż 20 zadaniami i 10 maszynami. Dlatego też zaproponowano wiele algorytmów przybliżonych rozwiązywania rozpatrywanego problemu szeregowania zadań, głównie metaheurystyk. Hurink [7] proponował algorytm poszukiwania z zabronieniami (*tabu search*). Dauzère-Pérès i Pauli [4] rozszerzyli klasyczny model dysjunktywny o możliwość reprezentacji przyporządkowania zadań do maszyn określonego typu. Mastrolilli i Gambardella [10] rozpatrywali algorytm poszukiwania z zabronieniami z efektywną metodą przeglądania otoczeń dla problemu gniazdowego z równoległymi maszynami. Wielu autorów proponuje metodę przyporządkowania zadań do maszyn, a następnie wyznaczania kolejności wykonywania operacji na poszczególnych maszynach. Taki podejście zastosowali Brandimarte [3] oraz Pauli [12]. Rozwiązują oni rozpatrywany problem jako problem przydziału opera-

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

cji do maszyn, a następnie używają metaheurystyki do rozwiązania klasycznego problemu gniazdowego. Także podejście genetyczne zaadoptowane zostało do rozwiązywania problemu gniazdowego z równoległymi maszynami. Ostatnie prace to Jia i inni [8], Ho i Tay [6], Kacem i inni [9], Pazzella i inni [13]. Gao i inni [5] proponują hybrydowy algorytm genetyczny z lokalnym poszukiwaniem opartym na metodzie zmiennych otoczeń (*variable neighborhood search,* VNS).

W dalszej części przedstawiono formalną definicję problemu oraz zaproponowano równoległy dwupoziomowy algorytm metaheurystyczny, zwany dalej meta^{2}heurystyką. Wykonano obliczenia na referencyjnych przykładach z literatury. Otrzymane wyniki wskazują jednoznacznie, że opisane algorytmy w krótkim czasie wyznaczają bardzo dobre rozwiązania.

2. Sformułowanie problemu

Problem gniazdowy z równoległymi maszynami (*flexible job shop problem*) może być sformułowany następująco. Dany jest zbiór zadań $J = \{1, 2, ..., n\}$, które należy wykonać na maszynach ze zbioru $M = \{1,2, ..., m\}$. Istnieje rozbicie zbioru maszyn na typy, tj. podzbiory maszyn o tych samych własnościach funkcjonalnych. Zadanie jest ciągiem pewnych operacji. Każdą operację należy wykonać na maszynie odpowiedniego typu w ustalonym czasie. Problem polega przydzieleniu zadań do maszyn z odpowiedniego typu oraz na wyznaczeniu kolejności wykonywania operacji na każdej maszynie, aby zminimalizować czas wykonania wszystkich zadań $\underline{C}_{\max}$. Muszą być przy tym spełnione ograniczenia (i)–(iv).

Niech $O = \{1, 2, ..., o\}$ 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 (tzw. ciąg technologiczny). Operacje te są indeksowane liczbami $(l_{j-1}+1, l_{j-1} + 2, ..., l_{j-1} + o_j)$, gdzie $l_j = \sum_{i=1}^{j} o_i$ jest liczbą operacji pierwszych j zadań, $j = 1, 2, ..., n$, przy czym $l_0 = 0$, a $o = \sum_{i=1}^{n} o_i$.

Zbiór maszyn $M = \{1, 2, ..., m\}$ można rozbić na q podzbiorów maszyn tego samego typu (gniazd), przy czym i-ty ($i = 1, 2, ..., q$) typ M^i zawiera m_i maszyn, które są indeksowane liczbami $(t_{i-1} + 1, t_{i-1} + 2, ..., l_{i-1} + m_i)$, gdzie $t_i = \sum_{i=1}^{i} m_i$ jest liczbą maszyn w pierwszych i typach, $i = 1, 2, ..., q$, 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 ($k = 1, 2, ... , q$) gnieździe. Ciąg zbiorów operacji

$$Q_k = \left[Q^k_{t_{k-1}+1}, Q^k_{t_{k-1}+2}, ..., Q^k_{t_{k-1}+m_k} \right],$$

takich że

$$O^k = \bigcup_{i=t_{k-1}+1}^{t_{k-1}+m_k} Q_i^k \text{ i } Q_i^k \cap Q_j^k = \varnothing, i \neq j, i, j = t_{k-1}+1, t_{k-1}+2, ..., t_{k-1}+m_k$$

nazywamy 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.

Ciąg

$$Q = [Q^1, Q^2, ..., Q^q],$$

gdzie Q^i (i = 1, 2, ..., q) jest przydziałem w *i*-tym gnieździe nazywamy przydziałem operacji zbioru O do maszyn ze zbioru M.

Jeżeli dokonano przydziału operacji do maszyn, wówczas wyznaczenie optymalnego terminu wykonywania operacji (w tym i kolejności wykonywania operacji na maszynach) sprowadza się do rozwiązania klasycznego problemu gniazdowego.

Niech $K = (K_1, K_2, ..., K_m)$, będzie ciągiem zbiorów, gdzie $K_i \in 2^{o^i}$, i = 1, 2, ..., m. W szczególności elementy tego ciągu mogą być zbiorami pustymi. Przez K oznaczamy zbiór wszystkich takich ciągów. Moc zbioru K wynosi $2^{|o^1|} \cdot 2^{|o^2|} \cdot ... \cdot 2^{|o^q|}$.

Jeżeli Q jest dowolnym przydziałem operacji do maszyn, to $Q \in K$ (oczywiście, zbiór K zawiera także ciągi, które są niedopuszczalne, tj. nie są przydziałem operacji do maszyn).

Dla dowolnego ciągu zbiorów $\kappa = (\kappa_1, \kappa_2, ..., \kappa_m)$ ($\kappa \in K$) przez $\Pi_i(\kappa)$ oznaczamy zbiór wszystkich permutacji elementów z κ_i. Dalej, niech $\pi(\kappa) = (\pi_1(\kappa), \pi_2(\kappa), ..., \pi_m(\kappa))$ będzie konkatenacją m ciągów (permutacji), gdzie $\pi_i(\kappa) \in \Pi_i(\kappa)$. Tak więc $\pi(\kappa) \in \Pi(\kappa) = \Pi_1(\kappa) \times \Pi_2(\kappa) \times ... \times \Pi_m(\kappa)$. Łatwo zauważyć, że jeżeli $\kappa = (\kappa_1, \kappa_2, ..., \kappa_m)$ jest pewnym przydziałem operacji do maszyn, to zbiór $\pi_i(\kappa)$ (i = 1, 2, ..., m) zawiera wszystkie permutacje (możliwe kolejności wykonywania) operacji ze zbioru κ_i na maszynie i. Dalej, niech

$$\Phi = \{(\kappa, \pi(\kappa)) : \kappa \in K \wedge \pi(\kappa) \in \Pi(\kappa)\}.$$

Dowolne rozwiązanie dopuszczalne rozpatrywanego problemu jest parą $(Q, \pi(Q)) \in \Phi$, gdzie Q jest przydziałem operacji do maszyn, a $\pi(Q)$ konkatenacją permutacji wyznaczających kolejność wykonywania operacji przydzielonych każdej z maszyn spełniającą ograniczenia (*i*)–(*iv*).

3. Meta^{2}heurystyka

Algorytm zaproponowany w pracy składa się z dwóch modułów:

1) modułu przydziału operacji do maszyn,
2) modułu rozwiązującego klasyczny problem gniazdowy.

Każdy z modułów oparty jest na pewnym algorytmie metaheursytycznym, dlatego też cały dwupoziomowy algorytm nazywać będziemy meta^{2}heurystyką (meta-kwadrat-heurystyką).

Moduł przydziału operacji do maszyn. Moduł ten jest oparty na metodzie tabu search i jest wykonywany sekwencyjnie. Pozwala on na wybór jednej z równoległych maszyn, na której ma zostać wykonana dana operacja.

Moduł rozwiązujący klasyczny problem gniazdowy. Moduł ten jest wykorzystywany do wyznaczenia uszeregowania odpowiadającego przyporządkowaniu operacji do maszyn. Rozwiązuje on klasyczny problem gniazdowy powstały po przyporządkowaniu operacji do maszyn. Moduł ten wykorzystuje konstrukcyjny algorytm INSA [11] lub TSAB [11].

Krok 0. Wyznacz początkowy przydział operacji do maszyn Q^0 oraz odpowiadające mu uszeregowanie zadań $\pi(Q^0)$.

Krok 1. Wyznacz sąsiedztwo $N(Q)$ bieżącego przydziału operacji do maszyn, usuń z sąsiedztwa elementy zabronione przez listę T.

Krok 2. Podziel $N(Q)$ na $k = \left\lceil \frac{|N(Q)|}{p} \right\rceil$ grup.
Każda grupa zawiera co najwyżej p elementów (p – liczba procesorów).

Krok 3. Dla każdej grupy k znajdź uszeregowanie zadań $\pi(Y)$ odpowiadające przydziałowi operacji do maszyn $Y \in N(Q)$ oraz wartość $C_{\max}(Y, \pi(Y))$.

Krok 4. Znajdź przydział operacji do maszyn $Z \in N(Q)$, dla którego $C_{\max}(Z, \pi(Z)) = \min\{C_{\max}(Y, \pi(Y)) : Y \in N(Q)\}$.

Krok 5. **Jeśli** $C_{\max}(Z, \pi(Z)) < C_{\max}(Q^*, \pi(Q^*))$, **to**

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

$$Q^* = Z.$$

Umieść Z na liście T;

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

$$Q = Z.$$

Krok 6. Jeśli spełniony jest warunek zakończenia, to STOP.
W przeciwnym wypadku idź do kroku 1.

Sąsiedztwo dla rozwiązania $N(Q)$ generowane jako przyporządkowania zadań do maszyn, w których każdemu zadaniu przydzielane są maszyny o wyższym oraz niższym nu-

merze spośród maszyn tego samego typu. W drugim kroku przedstawionego wyżej algorytmu sąsiedztwo $N(Q)$ jest dzielone na zbiory rozłączne

$$\bigcup_{i=1}^{k} N_i(Q) = N(Q), \quad \bigcap_{i=1}^{k} N_i(Q) = 0.$$

Dla każdej grupy k wartość $C_{\max}$ jest obliczana przez p procesorów GPU. Liczba procesorów użytych w kroku trzecim zależy od rozmiaru sąsiedztwa. W kroku trzecim wartość $C_{\max}$ odpowiadająca przydziałowi operacji do maszyn jest wyznaczana przy użyciu algorytmu INSA [11] lub TSAB [11]. Lista T przechowuje pary (u, v), gdzie u oznacza pozycję w wektorze przydziału operacji do maszyn, a v jest maszyną do, której u jest przydzielone przed wykonaniem ruchu. Początkowe przyporządkowanie operacji do maszyn jest wyznaczone przez poszukiwanie globalnego minimum w tabeli czasów wykonania opisanego w pracy Pezzella [13].

4. Eksperymenty obliczeniowe

Równoległy algorytm meta^{2}heurystyczny dla problemu gniazdowego z równoległymi maszynami został zaimplementowany w języku C (CUDA) dla GPU i uruchomiony na 128-procesorowej karcie graficznej Tesla C870 dysponującej mocą obliczeniową 512 GFLOPS. Karta graficzna została zainstalowana na serwerze Hewlett-Packard z procesorem 2 Dual-Core AMD 1 GHz Operon z 1MB pamięci cache, 8 GB pamięci RAM z zainstalowanym 64-bitowym systemem operacyjnym Linux Debian 5.0. Algorytm został przetestowany dla grup 10 przykładów testowych zaproponowanych przez Brandimarte [3].

Tabela 1
Wyniki czasów obliczeń oraz przyspieszenia dla przykładów testowych Brandimarte [3]

Problem	$n \times m$	Flex.	O	t_p [ms]	t_s [ms]	*speedup s*
Mk01	10 × 6	2.09	55	3.93	35.82	9.11
Mk02	10 × 6	4.10	58	3.98	57.77	14.52
Mk03	15 × 8	3.01	150	34.52	1638.94	47.48
Mk04	15 × 8	1.91	90	8.72	157.87	18.10
Mk05	15 × 4	1.71	106	20.11	91.92	4.57
Mk06	10 × 15	3.27	150	40.09	2115.36	52.76
Mk07	20 × 5	2.83	100	15.84	494.17	31.20
Mk08	20 × 10	1.43	225	162.05	2275.69	14.04
Mk09	20 × 10	2.53	240	164.18	6248.89	38.06
Mk10	20 × 15	2.98	240	152.43	8190.89	53.74

W calu wyznaczenia przyspieszenia obliczeń, jakie można uzyskać, stosując algorytm równoległy wykorzystujący p procesorów GPU, została zaimplementowana sekwencyjna wersja algorytmu wykorzystująca jeden procesor GPU. Przyspieszenie (*speedup*) wyznaczane w taki sposób jest określane w literaturze jako przyspieszenie względne (*orthodox speedup,* Alba [2]) i porównuje czasy działania algorytmu równoległego i sekwencyjnego uruchamianych na tym samym sprzęcie. W tabeli 1 zostały przedstawione wyniki przeprowadzonych eksperymentów w postaci czasów trwania obliczeń dla algorytmu sekwencyjnego i równoległego oraz przyspieszenia obliczeń. Przyspieszenie względne s zostało wyznaczone z zależności $s = t_s/t_p$, gdzie t_s oznacza czas obliczeń dla algorytmu sekwencyjnego, a t_p czas obliczeń dla algorytmu równoległego. W kolumnie oznaczonej Flex. została zamieszczona średnia liczba maszyn przypadająca na każdą operację. Można zauważyć, że większe wartości przyspieszenia zostały uzyskane dla instancji testowych z dużą liczbą zadań n i dużą liczbą operacji o.

Przeprowadzone eksperymenty obliczeniowe pozwalają stwierdzić, że w środowisku GPU, czyli przy wykorzystaniu do obliczeń procesorów kart graficznych, możliwe jest nawet ponad 50-krotne skrócenie czasu obliczeń, w porównaniu z obliczeniami jednoprocesorowymi, przy zachowaniu tej samej jakości otrzymywanych rozwiązań.

5. Wnioski

W pracy została zaproponowana dwupoziomowa równoległa metaheurystyka, w której każde rozwiązanie z wyższego poziomu, tzn. przydział operacji do maszyn, definiuje NP-trudny problem gniazdowy, który jest rozwiązywany przez drugą metaheurystykę (konstrukcyjny algorytm INSA lub oparty na metodzie tabu search algorytm TSAB). W module dokonującym przydziału operacji do maszyn (poziom wyższy) została wykorzystana metaheurystyka oparta na metodzie populacyjnej. W module rozwiązującym problem szeregowania zadań (poziom niższy) został wykorzystany algorytm tabu search dla klasycznego problemu gniazdowego. Wykorzystanie algorytmów dokładnych (np. algorytm podziału i ograniczeń) na obu poziomach daje możliwość uzyskania rozwiązania optymalnego dla rozpatrywanego problemu.

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] Alba E., *Parallel Metaheuristics. A New Class of Algorithms*. Wiley & Sons Inc., 2005.

[3] Brandimarte P., *Routing and scheduling in a flexible job shop by tabu search.* Annals of Operations Research, 41, 1993, 157–183.

[4] Dauzčre-Pérčs S., Pauli J., *An integrated approach for modeling and solvingthe general multiprocessor job shop scheduling problem using tabu search.* Annals of Operations Research, 70, 3, 1997, 281–306.

[5] 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.

[6] Ho N.B., Tay J.C., *GENACE: an efficient cultural algorithm for solving the Flexible Job-Shop Problem*. IEEE International Conference on Robotics and Automation, 2004, 1759–1766.
[7] 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.
[8] Jia H.Z., Nee A.Y.C., Fuh J.Y.H., Zhang Y.F., *A modified genetic algorithm for distributed scheduling problems*. International Journal of Intelligent Manufacturing, 14, 2003, 351–362.
[9] Kacem I., Hammadi S., Borne P., *Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems*. IEEE Transactions on Systems, Man, and Cybernetics, Part C 32(1), 2002, 1–13.
[10] Mastrolilli M., Gambardella L.M., *Effective neighborhood functions for the flexible job shop problem.* Journal of Scheduling, 3(1), 2000, 3–20.
[11] Nowicki E., Smutnicki C., *A fast tabu search algorithm for the permutation flow-shop problem*. European Journal of Operational Research, 91, 1996, 160–175.
[12] Pauli J., *A hierarchical approach for the FMS schduling problem.* European Journal of Operational Research, 86(1), 1995, 32–42.
[13] Pezzella F., Morganti G., Ciaschetti G., *A genetic algorithm for the Flexible Job-schop Scheduling Problem.* Computers & Operations Research 35, 2008, 3202–3212.
[14] Pinedo M., *Scheduling: theory, algorithms and systems.* Englewood Cliffs, NJ: Prentice-Hall, 2002.