AUTOMATYKA • 2009 • Tom 13 • Zeszyt 2

Wojciech Bożejko*, Michał Czapiński**, Mieczysław Wodecki**

Równoległy algorytm hybrydowy dla problemu przepływowego z kryterium C_{sum}

1. Wstęp

W permutacyjnym problemie przepływowym (*permutation flow shop*) dany jest zbiór zadań oraz zbiór maszyn. Każde z zadań należy wykonać kolejno na wszystkich maszynach, przy czym kolejność wykonywania zadań na każdej maszynie musi być taka sama. Optymalizacja polega na wyznaczeniu kolejności wykonywania zadań na maszynach, która minimalizuje sumaryczny czas ich zakończenia. W literaturze problem ten jest oznaczany prze $F\|C_{sum}$ (w skrócie SFS) i należy on do klasy problemów silnie NP-trudnych.

Jak do tej pory opublikowano niewiele algorytmów rozwiązywania omawianego problemu. W zdecydowanej większość prac dotyczących problemu przepływowego, jako kryterium przyjmuje się minimalizację terminu zakończenia wszystkich zadań $C_{\max}$. Problem ten jest powszechnie uznawany za prostszy w rozwiązywaniu z uwagi na pewne szczególne własności, na przykład tzw. „własności blokowe" (Nowicki i Smutnicki [8], Grabowski i Wodecki [5]). Dla problemu z kryterium C_{sum} własności tych niestety nie można zastosować, co znacznie zwiększa czas obliczeń algorytmów (także i tych aproksymacyjnych). Stąd rosnące zainteresowanie metodami programowania równoległego cechującymi się o wiele większymi możliwościami w tej dziedzinie.

Algorytmy konstrukcyjne (*LIT* i *SPD* zamieszczone w pracy Wang i in. [12] oraz *NSPD* z pracy Liu [7]) rozwiązywania problemu SFS cechuje bardzo słaba efektywność. Reeves i Yamada [10] przedstawili hybrydowy algorytm genetyczny posiadający elementy algorytmu przeszukiwania z zabronieniami (*tabu serach*) i symulowanego wyżarzania (*simulated annealing*) oraz technikę ścieżek łączących (*path relinking*). Wykonując bardzo dużą liczbę iteracji, algorytmem tym wyznaczono najlepsze rozwiązania dla referencyjnych przykładów. Główną wadą algorytmu jest czasochłonność – obliczenia dla zestawów o rozmiarze 50×5 trwały 45 minut, a dla zestawów o rozmiarze 50×10 aż 90 minut. W pracach Bożejko i Pempera [1] oraz Bożejko i Wodecki [2], przedstawiono szybkie i efektywne

* Instytut Informatyki, Automatyki i Robotyki Politechniki Wrocławskiej
** Instytut Informatyki, Uniwersytet Wrocławski

algorytmy równoległe oparte na metodzie przeszukiwania z tabu oraz poszukiwań rozproszonych (*scatter search*) z łączeniem ścieżek. Wyznaczono wiele lepszych od dotychczasowych rozwiązań.

Zastosowanie metod programowania wieloprocesorowego pozwala przyśpieszyć obliczenia. Efektywność procesu znacznie wzrasta, gdy istnieje możliwość komunikowania się procesorów. Wymiana informacji pozwala na wspólną strategię prowadzenia obliczeń i w związku z tym, znaczne ich skrócenie. Jednak w przypadku obliczeń rozproszonych, komunikacja może nawet przekroczyć efektywny czas pracy procesorów. Stąd w algorytmie zastosowaliśmy sposób realizacji współpracy (komunikacji), w którym procesory są częściowo niezależne (*semi-independent*), tj. komunikacja występuje rzadko co określoną liczbę iteracji.

Metoda symulowanego wyżarzania (w skrócie SW) jest prosta w implementacji i szczególnie w swej klasycznej postaci (z funkcją akceptacji Boltzmanna) nie wymaga częstej komunikacji. Jest więc idealna do realizacji w środowisku obliczeń rozproszonych. Stosując odpowiednie otoczenia oraz zmiany temperatury, w sposób naturalny można także intensyfikować i dywersyfikować obliczenie.

W pracy przedstawiamy ideę konstrukcji wielowątkowego algorytmu symulowanego wyżarzania dla problemów optymalizacji dyskretnej, których rozwiązaniami dopuszczalnymi są permutacje. Idea algorytmu rozproszonego jest zbliżona do wyspowego algorytmu genetycznego. Realizuje się niezależne wątki – algorytmy SW z różnymi parametrami, które podobnie jak w algorytmie genetycznym są zmieniane co pewną liczbę iteracji. Działanie algorytmu wielowątkowego ilustrujemy na przykładzie permutacyjnego problemu przepływowego z kryterium C_{sum}. Otrzymane wyniki porównujemy z najlepszymi znanymi w literaturze. Algorytm wieloprocesorowy nie tylko przyśpiesza obliczenia, ale także poprawia wartości rozwiązań (przy takiej samej liczbie iteracji, jak algorytm sekwencyjny) oraz wykazuje znacznie szybszą zbieżność.

2. Definicje i oznaczenia

Permutacyjny problem przepływowy można sformułować następująco. Dany jest zbiór n zadań $J = \{1, 2, \ldots, n\}$ oraz zbiór m maszyn $M = \{1, 2, \ldots, m\}$. Zadanie $j \in J$ jest ciągiem m operacji $O_{j1}, O_{j2}, \ldots, O_{jm}$. Operację O_{jk} należy wykonać, bez przerywania, na maszynie k w czasie p_{jk}. Wykonywanie zadania na maszynie k ($k = 2, ..., m$) może się rozpocząć dopiero po zakończeniu wykonywania tego zadania na maszynie $k - 1$. Należy wyznaczyć kolejność, minimalizującą sumę czasów zakńczenia wykonania zadań.

Niech $\pi = (\pi(1), \pi(1), ..., \pi(n))$ będzie permutacją zadań $\{1, 2, ..., n\}$, a Π zbiorem wszystkich takich permutacji. Każda permutacja $\pi \in \Pi$ wyznacza jednoznacznie kolejność wykonywania zadań na maszynach (na każdej maszynie taką samą). W omawianym problemie należy wyznaczyć permutację $\pi^* \in \Pi$ taką, że:

$$C_{sum}(\pi^*) = \min_{\pi \in \Pi} C_{sum}(\pi),$$

gdzie

$$C_{sum}(\pi) = \sum_{j=1}^{m} C_{n,j}(\pi),$$

a $C_{i,j}(\pi)$ jest czasem zakończenia wykonywania zadania i na maszynie j, gdy są one wykonywane w kolejności π (tj. zadanie $\pi(i)$ jest wykonywane jako i-te w kolejności, $i = 1, 2, ..., n$).

Czasy wykonywania zadań mogą być wyznaczane z następującej zależności rekurencyjnej:

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

z warunkami poczatkowymi:

$$C_{\pi(0),j} = 0, \quad j = 1, 2, ..., m \quad \text{oraz} \quad C_{\pi(i),0} = 0, \quad i = 1, 2, ..., n.$$

3. Metoda symulowanego wyżarzania

Ze względu na prostotę implementacji, a jednocześnie jej uniwersalizm, metoda symulowanego wyżarzania jest z powodzeniem stosowana do rozwiązywania wielu problemów optymalizacji dyskretnej. Po raz pierwszy została zastosowana przez Kirkpatricka [6] oraz Černy'ego [3]. Jej idea pochodzi z termodynamiki. Polega ona na iteracyjnym polepszaniu bieżącego rozwiązania poprzez lokalne przeszukiwanie. Rozpoczyna się od pewnego rozwiązania startowego. Następnie generuje się jego otoczenie (sąsiedztwo) oraz wyznacza losowo pewne rozwiązanie z tego otoczenia, które przyjmuje się (z pewnym prawdopodobieństwem) za rozwiązanie startowe w następnej iteracji. Podstawowymi elementami algorytmu opartego na metodzie symulowanego wyżarzania są:

1) *otoczenie* – podzbiór zbioru rozwiązań dopuszczalnych, generowany przez ruchy (przekształcenia) z rozwiązania startowego,
2) *funkcja akceptacji* – określa prawdopodobieństwo, z jakim są akceptowane (za rozwiązania startowe) elementy otoczenia,
3) *schemat chłodzenia* – funkcja generująca parametr zwany temperaturą, mający bezpośredni wpływ na funkcję akceptacji.

Niech $\pi \in \Pi$ będzie dowolną permutacją startową, $\mathbb{N}_\pi$ jej otoczeniem, a π^* najlepszym do tej pory znalezionym rozwiązaniem (na początek przyjmujemy za π^* permutację π). Przez $\varphi(t)$ oznaczmy schemat chłodzenia (t – temperatura) oraz przez $\Psi_t(\pi, \beta)$ ($\beta \in \mathbb{N}_\pi$) funkcję akceptacji.

Zmiana temperatury (parametru t, mającego wpływ na funkcję akceptacji $\Psi_t(\pi, \beta)$) następuje po wykonaniu pewnej liczby iteracji zwanych *pętlą*. W opisie algorytmu jest to parametr R, który zazwyczaj przyjmuje wartość równą liczbie elementów otoczenia.

Warunkiem zatrzymania jest w praktyce maksymalna liczba iteracji lub czas obliczeń. Poszczególne elementy algorytmu mogą być realizowane na wiele sposobów. W konstrukcji algorytmu przyjęto:

- *otoczenie* $\mathbb{N}_\pi$ – zbiór permutacji generowany z π przez ruchy typu wstaw (*insert*), zobacz Grabowski i Wodecki [5],
- *funkcję akceptacji* Boltzmanna, postaci:

$$\Psi_t(\pi,\beta) = \exp[-(F(\beta) - F(\pi))/t],$$

- *geometryczny schemat chłodzenia,* postaci:

$$t_{k+1} = \alpha * t_k,$$

gdzie parametr $0 < \alpha < 1$ jest wyznaczany eksperymentalnie.

Schemat algorytmu symulowanego wyżarzania jest zamieszczony poniżej.

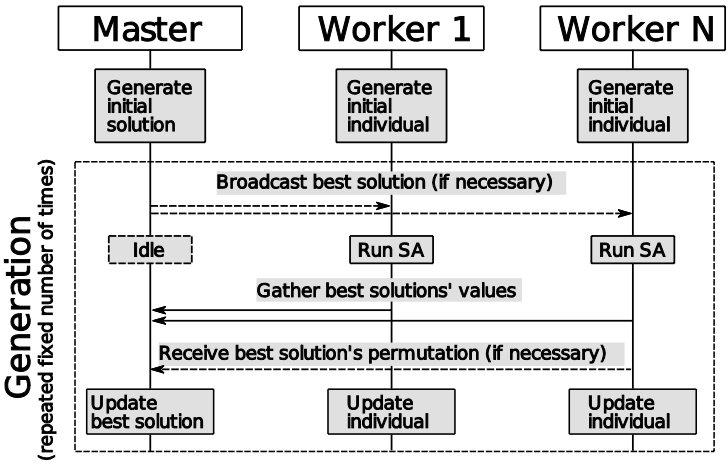
Standardowy algorytm symulowanego wyżarzania.
repeat
$i \leftarrow 0$;
while $i \leq R$ **do**
begin
$i \leftarrow i+1$;
Wyznaczyć losowo z otoczenia $\mathbb{N}_\pi$ element β;
if $F(\beta) < F(\pi^*)$ **then** $\pi^* \leftarrow \beta$;
if $F(\beta) < F(\pi)$ **then** $\pi \leftarrow \beta$
else
if $\Psi_t(\pi, \beta) > random[0,1)$ **then** $\pi \leftarrow \beta$
end;{i}
zmodyfikować parametr kontrolny t, zgodnie ze schematem chłodzenia $\varphi(t)$;
until Warunek zatrzymania;

4. Równoległy algorytm hybrydowy

Równoległy algorytm hybrydowy (w skrócie) HSA bazuje na standardowym algorytmie symulowanego wyżarzania. Zawiera mechanizm wyznaczania parametrów algorytmu SA oparty na idei algorytmu genetycznego. Na konfigurację algorytmu SA składają się następujące parametry:

- temperatura maksymalna (początkowa),
- temperaturę minimalna (końcowa),
- współczynnik chłodzenia α (schematu geometrycznego),
- tempo chłodzenia, tj. co ile iteracji algorytmu następuje obniżenie temperatury.

W algorytmie HSA, kompletna informacja o konfiguracji algorytmu SA będzie nazywana *osobnikiem*. Dodatkowo, każdy osobnik ma przypisany *czas życia* (nieujemną liczbę całkowitą). Jeśli czas ten osiągnie wartość zero, wówczas podczas ewolucji taki osobnik jest usuwany. W równoległym algorytmemie HSA, jeden wyróżniony procesor nazywany *zarządcą* organizuje pracę pozostałych procesów – *robotników*. Schemat działania algorytmu HSA jest przedstawiony na rysunku 1.



Rys. 1. Algorytm równoległy HSA

Na początku obliczeń zarządca wyznacza rozwiązanie startowe i rozsyła je do pozostałych procesorów. Każdy robotnik generuje losowego osobnika i ustala czas jego życia. Następnie wykonywana jest pewna ustalona liczba głównych pętli programu zwanych dalej *generacjami*. Każda generacja składa się z następujących kroków:

1. Jeśli rozwiązanie bazowe zmieniło się od ostatniej generacji, to jest ono rozsyłane do wszystkich robotników.
2. Każdy robotnik uruchamia algorytm SA na zadaną liczbę iteracji, zaczynając od rozwiązania startowego (każdy robotnik ma potencjalnie inną konfigurację algorytmu SA). Otrzymane najlepsze rozwiązania są odsyłane do zarządcy.
3. Zarządca wybiera najlepsze rozwiązanie spośród nadesłanych. Jeśli jest gorsze od bieżącego, to jest ono ignorowane. W przeciwnym wypadku zastępuje ono rozwiązanie bazowe. Na początku następnej generacji zostanie ono rozesłane, jako nowe rozwiązanie startowe, do wszystkich robotników.
4. Wykonywana jest ewolucja osobników. Każdy robotnik, który po uruchomieniu algorytmu SA otrzymał rozwiązanie lepsze od rozwiązania bazowego, przywraca czas życia swojego osobnika do stanu początkowego. W przeciwnym wypadku czas życia osobnika jest zmniejszany o jeden. Jeśli czas życia osobnika się wyzeruje, to jest on zastępowany nowym, losowym osobnikiem.

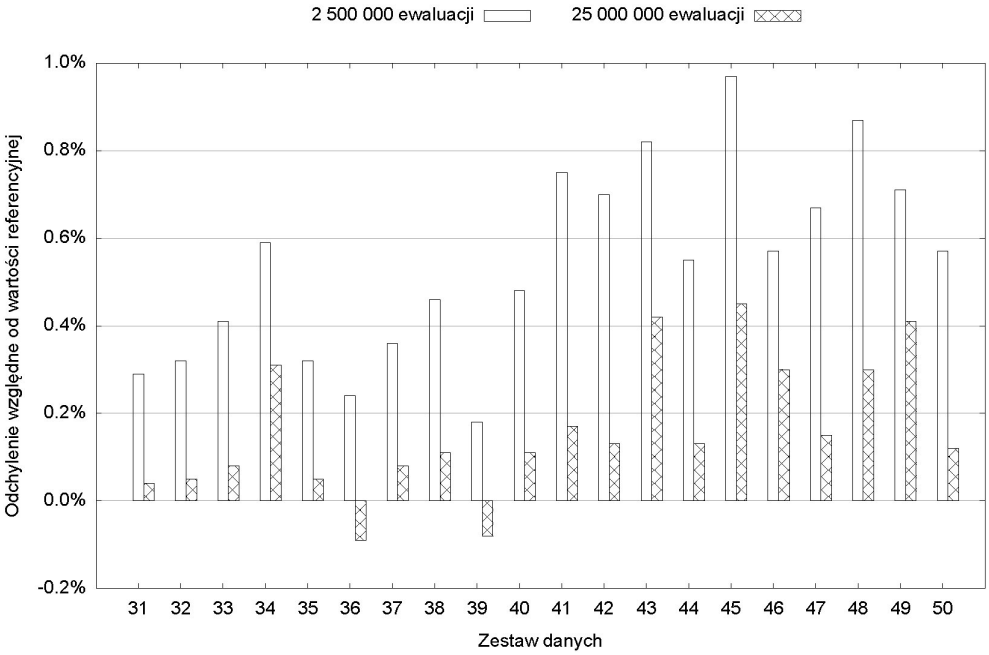
5. Eksperymenty obliczeniowe

Algorytm równoległy HSA zaimplementowano w język C++ z wykorzystaniem biblioteki MPI. Obliczenia wykonano na klastrze Astral posiadającym 856 procesorów Intel Xeon 3 GHz z pamięcią lokalną o wielkości 2 GB, zainstalowanym na Uniwersytecie Cranfield.

Parametrami algorytmu HSA są: *liczba generacji* – liczba głównych przebiegów algorytmu, *rozmiar populacji* – liczba osobników biorących udział w każdej kolejnej generacji, *liczba iteracji* algorytmu SA wykonywanych w każdej generacji, *minimalny czas życia* – „czas" (liczba generacji), po których osobnik jest usuwany, jeżeli nie poprawiono rozwiązania. Wszystkie parametry algorytmu są dokładnie opisane w pracy Czapińskiego [4].

Na czas działania algorytmu decydujący wpływ ma liczba ewaluacji funkcji celu, która jest iloczynem liczby generacji, rozmiaru populacji oraz liczby iteracji w generacji.

Testy przeprowadzono na 30 przykładach Taillarda [10] (ta31 – ta50) o rozmiarze $n \times m$ 50×5, 50×10, 50×20 zamieszczonych na stronie [9]. Otrzymane wyniki są przedstawione na rysunku 2.



Rys. 2. Wyniki algorytmu równoległego HSA

Już przy 2,5 milionach ewaluacji funkcji celu maksymalny błąd względny nie przekracza 1%, a błąd średni jest poniżej 0,5%. Dziesięciokrotne zwiększenie liczby ewaluacji daje znaczną poprawę wyników. Średni błąd jest o ponad połowę mniejszy. Dla dwóch przykładów ta36 i ta39 otrzymano lepsze od referencyjnych rozwiązania. Świadczy to o dobrej zbieżności algorytmu.

Niech $T(p)$ będzie czasem działania algorytmu równoległego na p procesorach. Wówczas $S(p) = T(1)/T(p)$ jest *przyśpieszeniem*, a $E(p) = (S(p)/p)\cdot 100\%$ *efektywnością* algorytmu. Przyśpieszenie oraz efektywność algorytmu HSA, dla przykładu problemu przepływowego z $n = 50$ zadaniami i $m = 10$ maszynami, przedstawiono w tabeli 1.

Tabela 1
Przyśpieszenie oraz efektywność algorytmu równoległego

Liczba procesorów p	128k ewaluacji		12.8M ewaluacji	
	Przyśpieszenie $S(p)$	Efektywność $E(p)$	Przyśpieszenie $S(p)$	Efektywność $E(p)$
1	1.00	100.00%	1.00	100.00%
2	2.00	100.00%	2.00	99.78%
4	3.92	98.00%	3.99	99.83%
8	7.84	98.00%	7.87	98.32%
16	14.00	87.00%	15.73	98.32%

6. Podsumowanie

W pracy przedstawono metodę rozwiązywania permutacyjnego problemu przepływowego z kryterium C_{sum} opartą na równoległym algorytmie symulowanego wyżarzania z elementami algorytmu genetycznego. Przeprowadzono eksoerymenty obliczeniowe na reprezentatywnej grupie danych referencyjnych. Otrzymane w krótkim czasie rozwiązania tylko nieznacznie różnią się od najlepszych obecnie znanych w literaturze.

Literatura

[1] Bożejko W., Pempera J., *Parallel Tabu Search Algorithm for the Permutation Flow Shop Problem with Criterion of Minimizing Sum of Job Completion Times*. 2008, Conference on Human System Interaction HSI'08, IEEE Computer Sciety, 283–332.

[2] Bożejko W., Wodecki M., *Parallel path-relinking method for the flow shop scheduling problem*. 2008, International Conference on Computational Science (ICCS 08) LNCS 5101, 264–273.

[3] Černy V., *Thermodynamical approach to travelling salesman problem: An efficient simulation algorithm*. J. Optim. Theory Appl., 45, 1985, 41–51.

[4] Czapiński M., *Parallel simulated annealing with geneticenhancement for flowshop problem with* C_{sum}. 2009 (w redakcji).

[5] 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.

[6] Kirkpatrick S., Gellat C.D., Vecchi M.P., *Optimization by simulated annealing*. Science, 220, 1983, 671–680.

[7] Liu J., *A new heuristic algorithm for csum flowshop scheduling problems*. Personal Communication, 1997.

[8] Nowicki E., Smutnicki C., *A fast tabu search algorithm for the permutation flow-shop problem*. European Journal of Operational Research, 91, 1996, 160–175.
[9] OR-Library: http://people.brunel.ac.uk/~mastjjb/jeb/info.html.
[10] Reeves C.R., Yamada T., *Solving the Csum Permutation Flowshop Scheduling Problem by Genetic Local Search*. IEEE International Conference on Evolutionary Computation, 1998, 230–234.
[11] Taillard E., *Benchmarks for basic scheduling problems*. European Journal of Operational Research, 64, 1993, 278–285.
[12] Wang C., Chu C., Proth J., *Heuristic approaches for n/m/F/*$\sum C_i$ *scheduling problems*. European Journal of Operational Research, 1997, 636–644.