AUTOMATYKA • 2007 • Tom 11 • Zeszyt 1–2

Wojciech Bożejko*, Mieczysław Wodecki**

Równoległy algorytm *scatter search* dla problemu przepływowego z kryterium C_{sum}

1. Wstęp

W permutacyjnym problemie przepływowym (*Permutation Flow Shop*) 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, która minimalizuje sumaryczny czas ich zakończenia. W literaturze problem ten jest oznaczany przez $F||C_{\text{sum}}$ i należy on do klasy problemów silnie NP-trudnych.

Jak do tej pory opublikowano niewiele algorytmów rozwiązywani 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" [4, 7]. Dla problemu z kryterium C_{sum} własności tych niestety nie można zastosować, co znacznie zwiększa czas obliczeń algorytmów jego rozwiązywania (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 [12] oraz *NSPD* w [6]) rozwiązywania problemu przepływowego z kryterium C_{sum} cechuje niestety bardzo słaba efektywność. Reeves i Yamada [9] przedstawili hybrydowy algorytm genetyczny posiadający elementy algorytmu przeszukiwania z zabronieniami (*tabu serach*) i symulowanego wyżarzania oraz technikę ścieżek łączących (*path relinking*). Wykonując bardzo dużą liczbę iteracji, wyznaczyli oni najlepsze znane obecnie rozwiązania dla referencyjnych przykładów.

Praca ta stanowi kontynuację badań autorów nad konstrukcjami efektywnych wieloprocesorowych algorytmów rozwiązywania bardzo trudnych problemów kombinatorycznych (np. [1, 2, 3, 11]). W dalszej części przedstawimy algorytm równoległy, oparty na metodzie algorytmu genetycznego, którego wersja równoległa nie tylko przyspiesza jego działanie, ale także poprawia wartości wyznaczanych rozwiązań.

* Instytut Informatyki, Automatyki i Robotyki, PolitechnikaWrocławska
** Instytut Informatyki, Uniwersytet Wrocławski

W kolejnych rozdziałach pracy przedstawiamy: opis rozważanego problemu szeregowania, metodę algorytmu genetycznego, algorytm oraz jego wersję równoległą, a na koniec wyniki obliczeniowe oraz podsumowanie.

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 $\pi = (\pi(1), \pi(1), \ldots, \pi(n))$ będzie permutacją zadań $\{1, 2, \ldots, n\}$, 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, $i = 1, 2, ..., n$).

3. Metoda *scatter search*

Główne idee tej metody są przedstawione między innymi w pracy James i in. [5]. Bazuje ona na ewaluacji zbioru tzw. **rozwiązań startowych**. W wersji klasycznej używano kombinacji liniowej do wyznaczania rozwiązań pośrednich pomiędzy rozwiązaniami startowymi. W przypadku reprezentacji rozwiązania w postaci permutacji, użycie kombinacji liniowej permutacji traktowanej jako wektor zwykle doprowadzi do powstania rozwiązania, który nie jest permutacją. Dlatego też w niniejszej pracy użyto procedury tworzącej ścieżkę łączącą dwa wylosowane rozwiązania ze zbioru zawierającego elementy startowe. Następnie, w procesie stochastycznie sterowanego poszukiwania wybieramy ze ścieżki najlepszy element.

Algorytm 1. ***Scatter search***

for i := 1..*liczba_iteracji* **do**

Krok 1.

Wygenerować zbiór S ($|S|=n$), różnych rozwiązań startowych;

Krok 2.

Dla wylosowanych $n/2$ par rozwiązań startowych z S zastosować procedurę ścieżek łączących (*path relinking*) do wygenerowania zbioru S' zawierającego $n/2$ rozwiązań leżących na ścieżkach;

Krok 3.

Zastosować algorytm lokalnych poszukiwać w celu poprawienia wartości znalezionych rozwiązań ze zbioru S';

Krok 4.

Dodać rozwiązania ze zbioru S' do zbioru S. Pozostawić w zbiorze S co najwyżej n rozwiązań, usuwając z niego rozwiązania najgorsze lub powtarzające się;

Krok 5.

if $|S|<n$ **then**

Dolosować nowe rozwiązania do zbioru S tak, aby elementy w S były różne oraz $|S|=n$

end for.

Procedura generowania ścieżki łączącej (*path relinking*)

Podstawą działania procedury generującej ścieżkę łączącą dwa wybrane rozwiązania ze zbioru startowego S jest schemat wielokrokowej fuzji *MSXF* (*Multi Step Crossover Fusion*), operatora zaproponowanego w pracy Reeves i Yamada [9]. Jako miarę odległości służącą do kierowania procesem tworzenia ścieżki pomiędzy $\pi(i)$ oraz $\sigma(i)$ przyjęto miarę Hamminga

$$H(\pi, \sigma) = \text{liczba } i \text{ takich, że } \pi(i) \neq \sigma(i).$$

Złożoność obliczeniowa wyznaczenia wartości odległości jest liniowa $O(n)$, gdzie n jest liczbą elementów w permutacji.

Algorytm 2. Procedura generowania ścieżki łączącej

π_1, π_2 – rozwiązania ze zbioru S. Niech $x = q = \pi_1$;
$N(\pi)$ – otoczenie rozwiązania π wygenerowane za pomocą ruchów typu wstaw (*insert*);
T – parametr;

repeat

Dla każdego $y_i \in N(\pi)$ wyznaczyć odległość $H(y_i, \pi_2)$;
Posortować $y_i \in N(\pi)$ w porządku rosnącym względem miary $H(y_i, \pi_2)$;
repeat
Wybrać element y_i ze zbioru $N(\pi)$ z prawdopodobieństwem odwrotnie proporcjonalnym do indeksu i;
Wyznaczyć $C_{sum}(y_i)$;
Zaakceptować y_i, jeśli $C_{sum}(y_i) \leq C_{sum}(x)$ lub w przeciwnym przypadku z prawdopodobieństwem $P_T(y_i) = exp((C_{sum}(x) - C_{sum}(y_i))/T)$;
Zmienić indeksy y_i, od i do n oraz indeksy y_k, $k = i+1, ..., n$ z k na $k-1$;

until y_i jest akceptowane;
$x \leftarrow y_i$;
if $C_{sum}(x) < C_{sum}(q)$ **then** $q \leftarrow x$;
until *Warunek Zakończenia*;
return q; {*q jest najlepszym elementem leżącym na ścieżce*}

Algorytm generowania ścieżki łączącej kończy działanie po osiągnięciu rozwiązania π_2 lub tez po wykonaniu z góry ustalonej liczbie iteracji.

3.1. Równoległy algorytm *scatter search*

Algorytm równoległy został zaprojektowany z myślą o uruchomieniu na klastrze 152 dwurdzeniowych procesorów Intel Xeon 2,4 GHz połączonych siecią Gigabit Ethernet z przełącznikami 3Com SuperStack 3870), zainstalowanym we Wrocławskim Centrum Sieciowo-Superkomputerowym (WCSS). Superkomputer ten dysponuje pamięcią rozproszoną pomiędzy procesorami. Każdy procesor posiada pamięć lokalną o wielkości 4 GB. Biorąc pod uwagę tę architekturę, zdecydowano się na zaprojektowanie algorytmu opartego na metodzie *scatter search* bazującej na modelu klient-serwer. Obliczenia polegające na wykonaniu procedur tworzących ścieżki łączące będą wykonywane przez procesory na danych lokalnych. Ponadto komunikacja będzie się odbywała względnie rzadko, jedynie w celu wyznaczenie wspólnego zbioru nowych rozwiązań startowych. Proces komunikacji i ewaluacji zbioru rozwiązań startowych S będzie kontrolowany przez procesor o numerze 0. Do implementacji wykorzystano język C++ i bibliotekę MPI.

Algorytm 3. Równoległy algorytm scatter search dla modelu SIMD bez pamięci współdzielonej

parfor p := 1..*liczba_procesorów* **do**
for i := 1..*liczba_iteracji* **do**
Krok 1. **if** (p=0) **then** {*tylko procesor o numerze* 0}
Wygenerować zbiór niepowtarzających się rozwiązań startowych S, przy czym $|S|=n$;.
Wysłać zbiór S do pozostałych procesorów;
else {*pozostałe procesory*}
Odebrać od procesora 0 zbiór rozwiązań startowych S;.
end if;
Krok 2. Dla wylosowanych $n/2$ par rozwiązań startowych z S zastosować procedurę ścieżek łączących (*path relinking*) do wygenerowania S' – zbioru zawierajacego $n/2$ rozwiązań leżących na ścieżkach;
Krok 3. Zastosować algorytm lokalnych poszukiwać w celu poprawienia wartości znalezionych rozwiązań ze zbioru S';
Krok 4. **if** (p<>0) **then**
Wysłać rozwiązania ze zbioru S' do procesora 0;

else {*tylko procesor o numerze* 0}
Odebrać zbiory *S*' od pozostałych procesorów i dodać je do zioru *S*;
Krok 5. Pozostawić w zbiorze *S* co najwyżej *n* rozwiązań usuwając najgorsze oraz te, które się powtarzają;
if $|S|<n$ **then**
Dolosować nowe rozwiązania do zbioru S tak, aby elementy w *S* się nie powtarzały oraz $|S|=n$.
end if;
end if;
end for;
end parfor.

4. Eksperymenty obliczeniowe

Przedstawiony w poprzednim rozdziale algorytm był testowany na 50 przykładach o pięciu rozmiarach z liczbą operacji od 100 do 500 ($n \times m$ = 20×5, 20×10, 20×20, 50×5, 50×10). Dane te zostały wygenerowanych przez Taillarda [10] i wraz z najlepszymi znanymi rozwiązaniami zamieszczonymi w pracy Reevesa, Yamady [9], znajdują się w zbiorze na stronie OR-Library [8].

Dla każdego uruchomienia algorytmu równoległego wyznaczano następujące wielkości:

1. *ARPD – Average Percentage Relative Deviation*, średni procentowy błąd względem rozwiązania referencyjnego z pracy [9];
2. t_{total} (w sekundach) – czas wykonania wszystkich 50 przykładów;
3. t_{cpu} (w sekundach) – sumaryczny czas pracy wszystkich procesorów;
4. *speedup* – przyspieszenie algorytmu.

Obliczenia przeprowadzono dla liczby iteracji 10 (tab. 1 i 2) oraz 100 (tab. 3 i 4).

Tablela 1
Średnie procentowe błędne *ARPD* (liczba_iteracji=10)

$n \times m$	Liczba procesorów			
	1	2	4	8
20×5	0,91	0,85	0,63	0,61
20×10	0,78	0,46	0,50	0,49
20×20	0,67	0,43	0,43	0,30
50×5	3,08	2,55	2,92	2,41
50×10	3,03	2,85	2,83	2,82
Średnia	**1,69**	**1,43**	**1,46**	**1,33**

Tablela 2
Czasy obliczeń (sumarycznie dla 50 przykładów, liczba_iteracji=10).
Wyodrębniono udział komunikacji w całkowitym czasie obliczeń

Liczba procesorów	Klaster procesorów Xeon 3000 2.4 GHz		
	t_{total} (min:s)	t_{cpu} (min:s)	komunikacja
1	2:53	2:56	–
2	3:04	5:29	6,4%
4	3:36	8:01	25,9%
8	4:29	20:57	55,5%

Tablela 3
Średnie procentowe błędne *ARPD* (liczba_iteracji=100)

$n \times m$	Liczba procesorów			
	1	2	4	8
20×5	0,28	0,24	0,17	0,21
20×10	0,10	0,06	0,12	0,08
20×20	0,12	0,06	0,07	0,03
50×5	2,25	1,97	2,01	1,97
50×10	2,22	2,07	1,91	2,01
Średnia	**0,99**	**0,88**	**0,86**	**0,86**

Tablela 4
Czasy obliczeń (sumarycznie dla 50 przykładów, liczba_iteracji=100).
Wyodrębniono udział komunikacji w całkowitym czasie obliczeń

Liczba procesorów	Klaster procesorów Xeon 3000 2.4 GHz		
	t_{total} (min:s)	t_{cpu} (min:s)	komunikacja
1	27:04	27:03	–
2	28:34	52:24	5,5%
4	34:00	104:17	25,6%
8	42:03	209:14	55,4%

Średnia względna poprawa *ARPD* dla wersji 8-procesorowej w stosunku do 1-procesorowej wyniosła 21,3% dla 10 itreracja oraz 13,1% dla 100 iteracji. Klaster, na którym przeprowadzono obliczenia, posiada dwie jednostki obliczeniowe w każdym węźle. Komunikacja pomiędzy procesorami w tym samym węźle jest znacznie szybsza niż komunikacja

pomiędzy węzłami. Można to zaobserwować na podstawie wyników zamieszczonych w tabeli 2. Wynika z nich bowiem, że najmniejszy narzut związany z komunikacja, spośród algorytmów wieloprocesorowych, ma wersja z dwoma procesorami.

5. Podsumowanie

W pracy przedstawono konstrukcję algorytmu rozwiązywania permutacyjnego problemu przepływowego z kryterium C_{sum} opartą na równoległym algorytmie *scatter search*, modelu klient-serwer. Przy implementacji algorytmu zastosowano ideę ścieżek łączących (*path relinking*). Wyniki obliczeniowe wskazują, że algorytm równoległy jest znacznie efektywniejszy od sekwencyjnego. Otrzymane rozwiązania (po niewielkiej liczbie iteracji) tylko nieznacznie różnią się od najlepszych obecnie znanych w literaturze.

Literatura

[1] Bożejko W., Wodecki M.: *Solving the flow shop problem by parallel tabu search*. IEEE Computer Society, 2002, 189–194

[2] Bożejko W., Wodecki M.: *Parallel algorithm for some single machine scheduling problems*. Automatyka, 2002, z. 134, 81–90

[3] Bożejko W., Wodecki M.: *Permutacyjny problem przepływowy. Algorytmy równoległe symulowanego wyżarzania*. Automatyka, 2002, z. 134, 90–101

[4] Grabowski J., Pempera J.: *New block properties for the permutation flow-shop problem with application in TS*. Journal of Operational Research Society, 2001, 52, 210–220

[5] James T., Rego C., Glover F.: *Sequential and Parallel Path-Relinking Algorithms for the Quadratic Assignment Problem*. IEEE Intelligent Systems, 2005, 58–65

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

[7] Nowicki E., Smutnicki C.: *A fast tabu search algorithm for the permutation flow-shop problem*. European Journal of Operational Research, 1996, 91, 160–175

[8] OR-Library: http://people.brunel.ac.uk/~mastjjb/jeb/info.html

[9] 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

[10] Taillard E.: *Benchmarks for basic scheduling problems*. European Journal of Operational Research, 1993, 64, 278–285

[11] Wodecki M., Bożejko W.: *Solving the flow shop problem by parallel simulated annealing*. Lecture Notes in Computer Science, 2002, 2328, Springer Verlag, 236–247

[12] Wang C., Chu C., Proth J.: *Heuristic approaches for n/m/F/*SC_i *scheduling problems*. European Journal of Operational Research, 1997, 636–644