ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 2008
Seria: AUTOMATYKA z. 150 Nr kol. 1796

Mariusz MAKUCHOWSKI
Politechnika Wrocławska

PROBLEM GNIAZDOWY Z OGRANICZENIEM BEZ CZEKANIA. RÓWNOLEGŁE ALGORYTMY TABU

Streszczenie. W pracy opisane są równoległe algorytmy poszukiwania z zabronieniami, dedykowane gniazdowemu problemowi z ograniczeniem bez czekania. Proponowane algorytmy zbudowane są z nadrzędnego algorytmu bazującego na wspomnianej technice oraz sterowanego algorytmu konstrukcyjnego. Poszukiwania ograniczone są tylko do rozwiązań możliwych do wygenerowania przez wspomniany algorytm konstrukcyjny. W pracy przedstawia się analizę porównawczą zaproponowanych algorytmów.

THE NO–WAIT JOB SHOP PROBLEM. PARALLEL TABOO ALGORITHMS

Summary. This paper deals with parallel tabu search algorithms for a job shop problem with a no–wait constraint and a makespan criterion. The proposed algorithms consist of a master algorithm based on the mentioned technique and slave constructive algorithm. This approach reduces the number of solutions to check only to solutions that can be generated by means of the constructive algorithm. In this paper a comparative analysis of the proposed algorithms is presented.

1. Wprowadzenie

Można zauważyć, iż w obecnych czasach postęp technologiczny maszyn liczących zmienia kierunek swojego rozwoju. Jeszcze parę lat temu z roku na rok powstawały procesory taktowane coraz szybszymi zegarami. Obecnie zwiększanie mocy obliczeniowej procesorów uzyskuje się głównie przez zrównoleglenie jednostek liczących, w postaci produkcji procesorów nie jedno-, lecz dwu– i czterordzeniowych. Ponadto wszystkie superkomputery uzyskują ogromne moce obliczeniowe, właśnie dzięki rozłożeniu obliczeń na tysiącach, połączonych ze sobą bardzo szybką siecią, procesorach. Obecnie najszybszy superkomputer BlueGene/L, mający moc obliczeniową rzędu 478 TFlops, posiada 212.992 rdzeni [1]. Oczywiście, aby wykorzystać potencjał nowych technologii, należy stosować specjalne dedykowane tym systemom algorytmy równoległe.

Procedury optymalizacji dyskretnej, a w szczególności algorytmy popraw (ang. improvement algorithm), dobrze nadają się do zrównoleglania. Najłatwiejsze zrównoleglenie tych algorytmów może polegać na uruchomieniu kilku jednakowych niezależnych kopii (wątków programu) z zadbaniem o to, by każda z nich przeglądała inną

część przestrzeni rozwiązań. Ponieważ w algorytmach popraw użytkownik sam ustala kompromis pomiędzy jakością uzyskiwanego rozwiązania a czasem pracy algorytmu, zrównoleglanie obliczeń można wykorzystać do znajdowania lepszych rozwiązań w tym samym czasie, lub rozwiązań o podobnej jakości w krótszym czasie, niż robi to analogiczny jednowątkowy algorytm sekwencyjny.

2. Sformułowanie problemu

Problem gniazdowy z ograniczeniem bez czekania z kryterium będącym długością uszeregowania oznaczany jest w notacji Grahama [2] przez $J|no-wait|C_{\max}$. Jego model matematyczny został przedstawiony w pierwszej części publikacji [3]. W bieżącej części pracy obowiązują wszystkie byty zdefiniowane w części wcześniejszej [3].

3. Opis algorytmów

W pracy prezentuje się jeden sekwencyjny oraz dwa równoległe algorytmy dedykowane problemowi gniazdowemu z ograniczeniem bez czekania z kryterium będącym długością uszeregowania. Prezentowane programy oparte są na metodzie poszukiwań z zabronieniami. Pierwszy z prezentowanych algorytmów *HTS1* jest sekwencyjny i względem niego będzie oceniana jakość pozostałych dwóch. Kryterium stopu we wszystkich prezentowanych algorytmach to ustalona liczba przeglądanych rozwiązań.

3.1. Algorytm sekwencyjny – HTS1

Przedstawiany algorytm składa się z dwóch członów: sterującego i wykonawczego. Człon sterujący, oparty na technice poszukiwania z zabronieniami [4], dobiera parametry sterujące członem wykonawczym, nazywanym procedurą wykonawczą H [3].

Dla dalszego opisu algorytmu nadrzędnego ważne jest tylko, iż parametrem sterującym procedurą H jest permutacja zadań; $\pi = \pi(1), \pi(2), \ldots, \pi(r)$, $\pi(i) \in J$, $1 \leqslant i \leqslant r$, zwana permutacją ładującą. Nadrzędne sterowanie jest więc algorytmem, w którym zmienną decyzyjną jest jedna permutacja, zaś funkcją kryterialną jest wartość funkcji celu rozwiązania otrzymanego procedurą H. Takie podejście uniemożliwiło autorowi znalezienie praktycznych zależności pomiędzy zmianą w permutacji ładującej a zmianą długości otrzymywanego harmonogramu. Do pokazania zadziwiających własności tak analizowanego problemu może posłużyć przykład, w którym wyjęcie zadania z permutacji ładującej będzie skutkowało generacją częściowego harmonogramu, dłuższego niż wcześniejszy kompletny harmonogram! Istnienie powyższego zjawiska oznacza, że w wyniku dodania (lub zabrania) zadania do częściowej permutacji ładującej otrzymywany harmonogram może się zarówno wydłużyć, jak i skrócić! Ponieważ ogólny opis algorytmu poszukiwania z zabronieniami można znaleźć między innym w pracy [4], w poniższych podrozdziałach przedstawione zostaną szczegółowo najważniejsze składowe elementy tego algorytmu.

3.1.1. Sąsiedztwo

Sąsiedztwo permutacji wykorzystywane w algorytmie *HTS1* bazuje na ruchach typu wstaw. Ruch $v = (a, b)$, $1 \leqslant a \leqslant r$, $a \neq b$, $a \neq b - 1$, typu wstaw wykonany na permutacji $\pi \in \Pi$ polega na przesunięciu elementu $\pi(a)$ z pozycji a na pozycję b; w wyniku tworzy się nowa permutacja oznaczana $\pi_{(v)} \in \Pi$. Liczba ruchów, a tym samym

rozmiar sąsiedztwa dla permutacji r–elementowej, wynosi $(r-1)^2$. W prezentowanym algorytmie postanowiono ograniczyć wielkość analizowanego sąsiedztwa tylko do rozwiązań otrzymanych ruchami o ograniczonym rozmiarze. Rozmiar ruchu $v = (a, b)$ oblicza się jako liczbę pozycji, o jaką zostaje przestawiony element, $|a - b|$. Ostatecznie analizowane sąsiedztwo powstaje z ruchów $v = (a, b)$, $1 \leqslant a \leqslant r$, $a \neq b$, $a \neq b - 1$, $moveSizeMin \leqslant |a - b| \leqslant moveSizeMax$. Jeśli kilka ruchów generuje rozwiązanie jednakowo dobre, wybierany zostaje ruch o większym rozmiarze. W prezentowanym algorytmie *HTS1* ustalono następujące wartości parametrów $moveSizeMin = 1$, $moveSizeMax = 10$.

3.1.2. Lista tabu

Elementami listy tabu T, zgodnie z ideą zawartą w pracy [4], są pewne atrybuty permutacji i ruchu $AV(\pi, v)$ ostatnich iteracji algorytmu. Niech $(x \rightarrow y)$ oznacza relację kolejnościową zadań x i y w permutacji. Po wykonaniu ruchu $v = (a, b)$, przekształcającego permutację $\pi \in \Pi$ w permutację $\pi_{(v)} \in \Pi$, na listę tabu zapisywana jest pewna para (x, y), dla której w permutacji zachodziła relacja $(x \rightarrow y)$, a w permutacji zamieniła się w relację przeciwną $(y \rightarrow x)$. Oczywiste jest, iż dla danego ruchu może istnieć wiele takich par. W algorytmie zastosowano metodę wyboru pary zadań zabraniających możliwie największej liczby ruchów:

$$AV_{HARD}(\pi, v) = \begin{cases} (\pi(a), \pi(b)) & dla \quad a < b \\ (\pi(b), \pi(a)) & dla \quad a > b \end{cases}. \tag{1}$$

Ruch uważany jest za zabroniony, jeżeli na liście tabu znajduje się przynajmniej jedna para zadań (x, y), dla której występuje relacja $(x \rightarrow y)$ w powstającej permutacji $\pi_{(v)}$. W pracy [4] pokazany jest efektywny sposób wyznaczania statusu danego ruchu.

3.1.3. Rozwiązanie startowe

Pamiętając, iż zmienną decyzyjną w algorytmie *HTS1* jest kolejność zadań przekazywana do procedury H, jako rozwiązanie startowe przyjmuje się permutację ładującą otrzymaną algorytmem $HNEH$ opisanym w pierwszej części [3].

3.2. Algorytmy równoległe – HTS2, HTS3

Algorytm *HTS2* jest algorytmem równolegle uruchamiającym dwa algorytmy *HTS1* z różnymi parametrami sterującymi. Różne wysterowanie algorytmów ma na celu wymuszenie różnych trajektorii poszukiwań. W badanym algorytmie *HTS2* drugi wątek uruchamiany jest z listą tabu dłuższą o 1 niż długość listy tabu wątku pierwszego.

Dla każdej instancji można wygenerować instancję „lustrzaną", w której zamieniono na przeciwne kolejnościowe relacje technologiczne. Praktycznie jest to taka sama instancja, jednakże algorytm generujący rozwiązanie układa zadania w inny sposób niż dla instancji oryginalnej. Uzyskane rozwiązanie instancji lustrzanej jest, oczywiście, lustrzanym odbiciem pewnego rozwiązania instancji oryginalnej. Oznacza to, iż pewne „złośliwości" instancji, powodujące między innymi brak możliwości wygenerowania rozwiązania optymalnego, mogą nie istnieć dla instancji lustrzanej. Co więcej, jeśli algorytm nadrzędny dla pewnych danych ma trudności ze znalezieniem dobrego rozwiązania, to może się okazać, iż w przypadku instancji lustrzanej dana trudność nie występuje. Trudność dla algorytmów poprawy może być tutaj rozumiana jako problemy z opuszczeniem zbyt głębokiego bądź zbyt szerokiego minimum lokalnego lub problemy z poruszaniem się po płaskiej (w sensie wartości funkcji celu) przestrzeni rozwiązań.

Tabela 1

Wartości funkcji celu, błąd δ[%] względem rozwiązań optymalnych i czas t [s]

Przykład	$r \times m$	C^{OPT}	C^{HTS1}	C^{HTS2}	C^{HTS3}	δ^{HTS1}	δ^{HTS2}	δ^{HTS3}	t^{HTS1}	t^{HTS2}	t^{HTS3}
ft06	6× 6	73	73	73	73	0.0	0.0	0.0	0.0	0.0	0.0
ft10	10×10	1607	1607	1607	1607	0.0	0.0	0.0	0.4	0.2	0.2
la01	10× 5	971	975	975	971	0.4	0.4	0.0	0.1	0.1	0.1
la02	10× 5	937	961	961	937	2.6	2.6	0.0	0.2	0.1	0.1
la03	10× 5	820	820	820	820	0.0	0.0	0.0	0.1	0.1	0.1
la04	10× 5	887	888	888	887	0.1	0.1	0.0	0.2	0.1	0.1
la05	10× 5	777	781	781	777	0.5	0.5	0.0	0.1	0.1	0.1
la16	10×10	1575	1575	1575	1575	0.0	0.0	0.0	0.4	0.2	0.2
la17	10×10	1371	1384	1384	1384	0.9	0.9	0.9	0.4	0.2	0.2
la18	10×10	1417	1507	1507	1507	6.4	6.4	6.4	0.4	0.2	0.2
la19	10×10	1482	1512	1512	1491	2.0	2.0	0.6	0.4	0.2	0.2
la20	10×10	1526	1526	1526	1526	0.0	0.0	0.0	0.4	0.2	0.2
orb01	10×10	1615	1615	1615	1615	0.0	0.0	0.0	0.4	0.2	0.2
orb02	10×10	1485	1485	1485	1485	0.0	0.0	0.0	0.4	0.2	0.2
orb03	10×10	1599	1599	1599	1599	0.0	0.0	0.0	0.4	0.2	0.2
orb04	10×10	1653	1738	1653	1653	5.1	0.0	0.0	0.4	0.2	0.2
orb05	10×10	1363	1370	1370	1367	0.5	0.5	0.3	0.4	0.2	0.2
orb06	10×10	1555	1555	1555	1555	0.0	0.0	0.0	0.3	0.2	0.2
orb07	10×10	689	705	705	701	2.3	2.3	1.7	0.4	0.2	0.2
orb08	10×10	1319	1319	1319	1319	0.0	0.0	0.0	0.4	0.2	0.2
orb09	10×10	1445	1445	1445	1445	0.0	0.0	0.0	0.4	0.2	0.2
orb10	10×10	1557	1575	1575	1571	1.2	1.2	0.9	0.4	0.2	0.2
Średnia						**1.0**	**0.8**	**0.5**	0.3	0.2	0.2

Algorytm *HTS3* uruchamia dwie kopie algorytmu *HTS1*: jedną dla instancji oryginalnej, a drugą dla instancji lustrzanej.

4. Badania numeryczne

Badania testowe zostały podzielone na dwa etapy. W pierwszym z nich zbadano zachowanie się prezentowanych trzech algorytmów na przykładach stosunkowo łatwych. Wszystkie przykłady tej grupy udało się rozwiązać optymalnie algorytmem dokładnym opartym na metodzie podziału i ograniczeń [5]. Dla każdego z przykładów wyznaczono rozwiązanie C^{alg} kolejnymi algorytmami $alg \in \{HTS1, HTS2, HTS3,\}$ w czasie t^{alg}. Dodatkowo dla każdego rozwiązania C^{alg} policzono względny błąd

$$\delta^{alg} = \frac{C^{alg} - C^{REF}}{C^{REF}} \cdot 100\%, \quad alg \in \{HTS1, HTS2, HTS3\} \tag{2}$$

w stosunku do wartości funkcji celu rozwiązań optymalnych (znanych z literatury [6]), $C^{REF} = C^{OPT}$. Wszystkie omawiane wyniki zamieszczono w tabeli 1. Tabela zakończona jest wersem podającym średnią wartość błędów otrzymanych rozwiązań oraz średnią wartość czasów pracy poszczególnych algorytmów.

Etap drugi badań przeprowadzony został w analogiczny sposób jak etap pierwszy. Tym razem omawiane algorytmy testowane były na przykładach na tyle „trudnych" (dużych), iż nieznana jest po dzień dzisiejszy optymalna wartość funkcji celu. W tym przypadku otrzymane wartości funkcji celu uzyskanych uszeregowań po-

Tabela 2

Wartości funkcji celu, błąd δ[%] względem rozwiązań referencyjnych i czas t [s]

Przykład	$r \times m$	C^{REF}	C^{HTS1}	C^{HTS2}	C^{HTS3}	δ^{HTS1}	δ^{HTS2}	δ^{HTS3}	t^{HTS1}	t^{HTS2}	t^{HTS3}
la11	20× 5	1714	1705	1696	1710	-0.5	-1.1	-0.2	1.9	1.0	1.0
la12	20× 5	1507	1602	1558	1543	6.3	3.4	2.4	2.0	1.0	1.0
la13	20× 5	1665	1724	1715	1724	3.5	3.0	3.5	1.9	1.0	1.0
la14	20× 5	1757	1767	1722	1767	0.6	-2.0	0.6	1.9	1.0	1.0
la15	20× 5	1771	1779	1779	1779	0.5	0.5	0.5	1.8	0.9	1.1
la21	15×10	2092	2059	2163	2144	-1.6	3.4	2.5	2.0	1.1	1.1
la22	15×10	1928	1955	1955	1955	1.4	1.4	1.4	1.8	0.9	1.0
la23	15×10	2038	2213	2134	2213	8.6	4.7	8.6	1.8	0.9	1.0
la24	15×10	2061	2072	2044	2108	0.5	-0.8	2.3	2.0	1.0	1.0
la25	15×10	2034	2033	1995	2033	-0.0	-1.9	-0.0	1.8	0.9	1.0
la26	20×10	2664	2996	2788	2981	12.5	4.7	11.9	5.7	2.8	2.8
la27	20×10	2933	2881	2881	2881	-1.8	-1.8	-1.8	5.1	2.7	3.0
la28	20×10	2789	2833	2833	2833	1.6	1.6	1.6	5.2	2.8	2.7
la29	20×10	2565	2606	2606	2606	1.6	1.6	1.6	5.0	2.5	2.7
la30	20×10	2835	2808	2931	2779	-1.0	3.4	-2.0	4.9	2.6	2.5
la31	30×10	3822	3993	3989	3993	4.5	4.4	4.5	18.9	9.7	10.7
la32	30×10	4186	4201	4236	4339	0.4	1.2	3.7	21.3	10.7	10.7
la33	30×10	3754	3976	4055	3933	5.9	8.0	4.8	20.7	10.4	10.4
la34	30×10	3938	4064	4064	4064	3.2	3.2	3.2	20.8	10.8	10.7
la35	30×10	3908	4118	4118	3824	5.4	5.4	-2.1	21.4	10.7	10.6
la36	15×15	2810	2890	2890	2918	2.8	2.8	3.8	3.2	1.6	1.8
la37	15×15	3044	2977	3086	3127	-2.2	1.4	2.7	3.5	1.7	1.8
la38	15×15	2726	2691	2691	2691	-1.3	-1.3	-1.3	3.2	1.7	1.7
la39	15×15	2752	2938	2883	2852	6.8	4.8	3.6	3.4	1.7	1.7
la40	15×15	2838	2594	2889	2804	-8.6	1.8	-1.2	3.3	1.7	1.8
swv01	20×10	2396	2358	2358	2358	-1.6	-1.6	-1.6	5.4	2.7	2.7
swv02	20×10	2485	2613	2418	2479	5.2	-2.7	-0.2	4.9	2.5	2.6
swv03	20×10	2489	2545	2519	2493	2.2	1.2	0.2	5.3	2.7	2.7
swv04	20×10	2506	2575	2546	2575	2.8	1.6	2.8	5.2	2.7	2.7
swv05	20×10	2482	2561	2474	2466	3.2	-0.3	-0.6	5.5	2.8	2.8
swv06	20×15	3291	3602	3602	3587	9.5	9.5	9.0	8.4	4.2	4.5
swv07	20×15	3331	3432	3432	3418	3.0	3.0	2.6	9.7	4.9	4.9
swv08	20×15	3611	3742	3608	3742	3.6	-0.1	3.6	10.3	5.1	5.1
swv09	20×15	3378	3390	3400	3400	0.4	0.7	0.7	9.6	4.9	4.8
swv10	20×15	3552	3715	3568	3610	4.6	0.5	1.6	9.7	4.9	5.2
Średnia						**2.3**	**1.8**	**2.1**	6.8	3.5	3.5

równywane były z najlepszymi znanymi do tej pory wartościami z literatury [6, 7]. Uzyskane rezultaty zamieszczone zostały w tabeli 2. Ujemne wartości błędów δ^{alg}, $alg \in \{HTS1, HTS2, HTS3\}$ oznaczają, iż odpowiedni algorytm znalazł rozwiązanie lepsze niż najlepsze rezultaty opublikowane do tej pory.

Postawmy hipotezę, iż uzyskiwane rozwiązania algorytmami *HTS2* i *HTS3* są gorsze niż rozwiązania otrzymane algorytmem *HTS1*. Wykonując statystyczne testy otrzymanych wyników z tabeli 1 i tabeli 2, można stwierdzić, iż na poziomie istotności $\alpha = 0.05$ hipotezę tę należy odrzucić. Ponadto, dla tego samego poziomu istotności $\alpha = 0.05$ należy także odrzucić hipotezę, że algorytm *HTS2* dostarcza średnio rozwiązań z tym samym błędem co algorytm *HTS1*. Oznacza to, iż na podstawie uzyskanej próbki wyników pomiarowych można stwierdzić, że na 95% algorytm *HTS2* generuje rozwiązania o średniej wartości funkcji celu lepszej niż algorytm *HTS1*.

5. Podsumowanie

Pierwszy wniosek z przeprowadzonych badań jest następujący: algorytmy *HTS2* i *HTS3* uruchamiające dwa równoległe wątki działają odpowiednio dwukrotnie krócej, dostarczając średnio rozwiązań o 0.5% lepszych w sensie wartości funkcji celu niż jednowątkowy algorytm *HTS1*.

Drugi wniosek z pracy jest taki, iż zaproponowane algorytmy są bardzo wydajne. Wyniki najlepszych z opublikowanych algorytmów CLM [6], TS [7] pokazują, że uzyskują one rezultaty średnio 0,5% lepsze niż przykładowo *HTS2*, lecz w relatywnie znacznie dłuższym czasie. Pisząc o relatywnie dłuższym czasie pracy algorytmu uwzględniono zarówno różnicę w szybkości komputerów, jak i fakt równoległych obliczeń algorytmu *HTS2*. Dodatkowo każdy z prezentowanych w pracy algorytmów znalazł kilka rozwiązań lepszych niż najlepsze rezultaty opublikowane do tej pory.

BIBLIOGRAFIA

1. http://www.top500.org/
2. Graham R., Lawler E., Lenstra J., Rinnooy Kan A.: Optimization and approximation in deterministic sequencing and scheduling: a survey, Annals of Discrete Mathematics, 1979, 5, 287.
3. Makuchowski M.: Problem gniazdowy z ograniczeniem bez czekania. Algorytmy konstrukcyjne. Zeszyty Politechniki Śląskiej (ten numer).
4. Nowicki E.: Metoda tabu w problemach szeregowania zadań produkcyjnych. Oficyna Wydawnicza Politechniki Wrocławskiej, Wrocław 1999.
5. Mascis A., Pacciarelli D.: Job shop scheduling with blocking and no-wait constraints, European Journal of Operational Research. 142(3), 2002, p. 498.
6. Framinan J., Schuster C.: An enhanced timetabling procedure for the no-wait job shop problem: a complete local search approach, Computers and Operations Research 33 (5), 2006, 1200.
7. Schuster C.: No-wait Job Shop Scheduling: Tabu Search and Complexity of Subproblems, Mathematical Methods of Operations Research 63 (3), 2006, 473.

Recenzent: Prof. dr hab. inż. Marek Kubale

Abstract

In the previous paper [3] some solution constructive procedure for a job shop problem with the no–wait constrain and a makespan criterion has been proposed. That solution constructive procedure, called H procedure, is controlled by some job permutation called loading permutation. The H procedure generates $r!$ (r – is the number of tasks) high quality, in criterion sense, solutions.

The algorithms ($HTS1$, $HTS2$ and $HTS3$) presented, in this paper, consist of two levels. At the high level, there is a certain heuristic called the master algorithm and

at the low level, there is the H procedure. The master algorithms choose a loading permutation for the H procedure. They are based on taboo search method. The algorithm $HTS1$ is sequential, the algorithms $HTS2$ and $HTS3$ are parallel. They run simultaneously two threads of $HTS1$. The threads of $HTS2$ analysis different solutions, since the parameters of the master algorithms are different. In $HTS3$, the threads analysis completely different solutions since the first thread works on an original instance, and the second thread works on the symmetrical one. Computational results show the high efficiency of the proposed algorithms. For well–known from literature benchmarks, each presented algorithm finds a few solutions that are better than the best known from literature solutions.