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

Józef GRABOWSKI, Jarosław PEMPERA
Politechnika Wrocławska

NOWE WYKORZYSTANIE METODY BLOKOWEJ W KONSTRUKCJI ALGORYTMÓW HEURYSTYCZNYCH DLA OGÓLNEGO PROBLEMU PRZEPŁYWOWEGO

Streszczenie. Własności blokowe są z powodzeniem stosowane do usuwania ruchów nierokujących poprawy dla wielu otoczeń stosowanych w algorytmach popraw dla problemów szeregowania. W pracy, dla ogólnego problemu przepływowego, zaproponowano nowy sposób przeglądania otoczenia, który pozwala na eliminację znacznie większego zbioru ruchów. W celu sprawdzenia efektywności metody przeprowadzono eksperyment komputerowy na instancjach Taillarda.

NEW CONCEPT OF USING BLOCK METHOD IN THE CONSTRUCTION LOCAL SEARCH HEURISTICS FOR THE GENERAL FLOWSHOP PROBLEM

Summary. The block properties are successfully applied to a priory eliminate non promising moves from the neighborhood for many local search algorithm of solving scheduling problems. In this paper, for the general flowshop problem, it presents new method of searching the neighborhood, which gives rise to eliminate highly greater set of moves. To validate efficiency of the proposed method, computational experiment have been executed on a well-known Taillard's benchmarks.

1. Wstęp

W przepływowym systemie produkcyjnym każde zadanie należy wykonać kolejno na wszystkich maszynach. Zadanie optymalizacji polega na wyznaczeniu dopuszczalnego harmonogramu minimalizującego moment zakończenia realizacji wszystkich zadań. Ze względu na bardzo dużą liczbę rzeczywistych systemów produkcyjnych o charakterze przepływowym, problem ten od przeszło pięćdziesięciu lat jest przedmiotem zainteresowania badaczy oraz praktyków [1]. Niestety, tylko dla dwumaszynowego problemu przepływowego istnieje dokładny algorytm wielomianowy [2]. Dla większej liczby maszyn problem ten jest NP-zupełny [3]. Znane algorytmy dokładne wyznaczają rozwiązania optymalne [4] w czasie akceptowalnym dla praktyków jedynie dla niewielkiej liczby zadań, dlatego praktycy poszukują efektywnych algorytmów heurystycznych, w szczególności algorytmów opartych na metodach przeszukiwań lokalnych. W ogólnym problemie przepływowym FS

(niepermutacyjnym) dopuszcza się dowolną kolejność wykonywania zadań na maszynach, natomiast w permutacyjnym problemie przepływowym kolejność wykonywania zadań na wszystkich maszynach jest identyczna. Dla permutacyjnego FS opracowano algorytmy oparte na niemalże wszystkich najnowszych metodach konstrukcji algorytmów heurystycznych dla problemów kombinatorycznych. Najbardziej efektywne z nich bazują na szeroko rozumianych metodach przeszukiwania lokalnego, tj. przeszukiwania genetycznego [5], symulowanego wyżarzania [6], przeszukiwania z zabronieniami [7–9], przeszukiwania neuronowego [10], przeszukiwania mrówkowego [11]. Niestety, dla ogólnego FS, ze względu na znacznie większą złożoność oraz trudności w konstruowaniu efektywnych otoczeń dla algorytmów poprawy, nie opracowano tak wielu algorytmów, pomimo ewidentnego zmniejszenia czasu wykonywania zadań w przypadku zrezygnowania z wymogu identycznych kolejności wykonywania zadań na wszystkich maszynach [12].

2. Opis problemu

W ogólnym problemie przepływowym dany jest zbiór zadań $J=\{1,2,...,n\}$, który należy wykonać przy użyciu m maszyn ze zbioru $M=\{1,2,...,m\}$. Zadanie $j\in J$ należy kolejno wykonywać na każdej maszynie. Rozpoczęcie wykonywania zadania na maszynie k, $k\in\{2,\ldots,m\}$ może nastąpić dopiero po zakończeniu obróbki na maszynie k-1. Ze względu na miejsce wykonywania zadania, zadanie $j\in J$ można podzielić na m operacji $O_{j1},O_{j2},...,O_{jm}$. Operacja O_{jk} odpowiada wykonywaniu tego zadania, bez przerywania, na k-tej maszynie przez czas $p_{jk}>0$. Maszyna może w dowolnej chwili wykonywać co najwyżej jedno zadanie. Należy znaleźć harmonogram wykonywania operacji w systemie spełniający www. ograniczenia oraz minimalizujący moment zakończenia realizacji wszystkich zadań.

Kolejność wykonywania zadań na maszynach można opisać za pomocą zestawu składającego z m permutacji $\pi=(\pi_1,\ldots,\pi_m)$. Każda z permutacji π_k , $k=1,\ldots,m$, określona jest na zbiorze $\{1,\ldots,n\}$ i oznacza kolejność wykonywania zadań na maszynie k. Zestaw π będziemy również nazywali permutacją lub kolejnością.

Dla ustalonej kolejności π dopuszczalny harmonogram wykonywania zadań na maszynach określony przez terminy rozpoczęcia (zakończenia) wykonywania zadań S_{jk} (C_{jk}), $j=1,\ldots,n$, $k=1,\ldots,m$ musi spełniać następujące ograniczenia:

$$S_{j1}\geq 0,\quad j=1,\ldots,n, \tag{1}$$

$$C_{jk}=S_{jk}+p_{jk},\quad j=1,\ldots,n,\ k=1,\ldots,m, \tag{2}$$

$$S_{jk}\geq C_{j,k-1},\quad j=1,\ldots,n,\ k=2,\ldots,m, \tag{3}$$

$$S_{\pi k(j),k}\geq C_{\pi k(j-1),k},\quad j=2,\ldots,n,\ k=1,\ldots,m. \tag{4}$$

Nierówność (1) i równość (2) są oczywiste. Nierówność (3) modeluje ograniczenie technologiczne i oznacza, że moment rozpoczęcia wykonywania danego zadania na danej maszynie nie może być wcześniejszy od momentu zakończenia realizacji tego zadania na maszynie poprzedniej. Nierówność (4) modeluje ograniczenie wynikające z jednostkowej przepustowości maszyn.

3. Model grafowy problemu

Problem wyznaczenia dopuszczalnego harmonogramu wykonywania zadań dla ustalonej kolejności ich wykonywania π można sprowadzić do problemu wyznaczenia długości najdłuższych dróg w pewnym grafie skierowanym. Dla zadanej kolejności π definiujemy graf $G(\pi)=(N,T\cup F(\pi))$ ze zbiorem węzłów N i zbiorem nieobciążonych łuków $T\cup F(\pi)$, gdzie :

$$N=\{1,...,n\}\times\{1,...,m\}, \tag{5}$$

węzeł $(j,k)\in N$ reprezentuje k-tą operację zadania $\pi(j)$ i obciążony jest wagą $p_{\pi(j)k}$,

$$T=\bigcup_{j=1}^{n}\bigcup_{k=1}^{m-1}\{((j,k),(j,k+1))\}, \tag{6}$$

zbiór T zawiera modelujące ograniczenia technologiczne (3)

$$F(\pi)=\bigcup_{k=1}^{m}\bigcup_{j=1}^{n-1}\{((\pi_k(j),k),(\pi_k(j+1),k))\} \tag{7}$$

Każdy łuk $((\pi_k(j),k),(\pi_k(j+1),k))\in F(\pi)$ modeluje ograniczenia maszynowe (4).

Wyznaczenie najwcześniejszego terminu zakończenia wykonywania operacji O_{jk} równoważne jest wyznaczeniu długości najdłuższej drogi dochodzącej do węzła (j,k) (z obciążeniem tego węzła) w grafie $G(\pi)$.

Długość najdłuższej drogi do węzła $(\pi_m(n),m)$ określa najwcześniejszy moment zakończenia wykonywania wszystkich zadań. Najdłuższa droga w $G(\pi)$ nazywana jest ścieżką krytyczną w $G(\pi)$. Ścieżka krytyczna, z oczywistych względów rozpoczynająca się w węźle $(\pi_1(1),1)$, może być reprezentowana przez sekwencję węzłów. Niech $u=(u_1,u_2,\ldots,u_w)$ oznacza jedną dowolnie wybraną ścieżkę krytyczną w $G(\pi)$, gdzie $u_i=(j_i,k_i)\in N$, $1\le i\le w$, w jest liczbą węzłów w tej ścieżce.

W ścieżce u można wyróżnić maksymalny podciąg $(u_g,\ldots,u_h)=((j_g,k_g),\ldots,(j_h,k_h))$ ścieżki u taki, że $k_g=\ldots=k_h$ oraz $((j_i,k_i),(j_{i+1},k_{i+1}))\in F(\pi)$ dla $i=g,\ldots,h\text{-}1$, $g<h$. Sekwencję zadań $B_{gh}=(j_g,j_{g+1},\ldots,j_{h-1},j_h)$ będziemy nazywali *blokiem zadań*. W ścieżce krytycznej znajduje się co najwyżej m bloków. Dla bloku $B_{gh}=(j_g,j_{g+1},\ldots,j_{h-1},j_h)$ definiujemy: *pierwsze* zadanie w bloku jako j_g, *ostatnie* zadanie w bloku jako j_h oraz *blok wewnętrzny* jako podsekwencję $B^{*}_{gh}=(j_{g+1},\ldots,j_{h-1})$.

Niech π_v będzie kolejnością wykonywania zadań otrzymaną przez modyfikację kolejności π. Z własności blokowych [4] wynika, że $C_{\max}(\pi_v)\ge C_{\max}(\pi)$, jeżeli modyfikacja π polega na:

- zmianie kolejności wykonywania operacji wewnętrznych bloków,
- zmianie kolejności operacji nienależących do bloków,
- wstawieniu do bloków operacji nienależących do ścieżki krytycznej.

4. Proponowana metoda przeglądania sąsiedztwa dla ogólnego FS

Czas działania algorytmów opartych na metodach popraw oraz jakość dostarczanych przez nie rozwiązań istotnie zależy od sposobu generowania rozwiązań sąsiednich oraz ich liczby. Ze względu na liczbę operacji zamieniających swoją pozycję definicje zbiorów ruchów możemy podzielić na: zorientowane na przemieszczanie jednej operacji zadania, np. dla problemu gniazdowego lub

problemów z maszynami równoległymi, oraz zorientowane na przemieszczanie wszystkich operacji zadania (całego zadania), np. permutacyjny FS.

Jednym z najskuteczniejszy typów ruchów dla problemów szeregowania zadań jest ruch typu wstaw. Dla rozważanego problemu, w przypadku definicji zorientowanej na przesuwanie wszystkich operacji pojedynczych zadań, ruch typu wstaw możemy opisać za pomocą m+1-tki $v=(j,y_1,\ldots,y_m)$, który polega na usunięciu wszystkich operacji zadania j z π oraz ich ponownym wstawieniu do π, przy czym operacja wykonywana na maszynie k wstawiana jest na pozycję y_k w π_k, $k=1,\ldots,m$. Zbiór wszystkich ruchów wykonywanych operacjami zadania j będziemy oznaczali symbolem V_j. Oczywiście, liczba elementów V_j wynosi n^m. Czas obliczeń można znacząco zmniejszyć, eliminując ruchy, o których apriorycznie wiemy, że nie przyniosą poprawy rozwiązania π (własności blokowe).

Dotychczas w literaturze wyznaczanie zbioru wyeliminowanych ruchów następowało na podstawie analizy rozwiązania bazowego. W dalszej części pracy prezentujemy szkic metody pozwalającej na usunięcie dodatkowych ruchów na podstawie analizy generowanych w odpowiedniej kolejności rozwiązań sąsiednich.

Niech π będzie rozwiązaniem bazowym, natomiast π_v będzie nowym rozwiązaniem sąsiednim wygenerowanym przez wykonanie ruchu $v=(j,y_1,\ldots,y_m)$ w π. Rozpatrzmy przypadek, w którym ścieżka krytyczna u w $G(\pi_v)$ przechodzi przez operacje zadania j. Łatwo można sprawdzić, że co najmniej jedna operacja tego zadania wchodząca w skład ścieżki krytycznej nie będzie ostatnią operacją odpowiedniego bloku (z wyjątkiem przypadku, w którym ostatnia operacja tego zadania wykonywana jest jako ostatnia na ostatniej maszynie). Przyjmijmy dodatkowo, że operacja ta jest jedną tego typu operacją, wówczas z własności blokowych wnioskujemy, że dowolne przesunięcie pozostałych operacji nie przyniesie poprawy (ruchy te mogą być wyeliminowane). Zatem jednym ruchem rokującym poprawę wartości funkcji celu jest ruch tą operacją w prawo za ostatnią w bloku, do którego należy ta operacja. Z podobnych rozważań przeprowadzonych dla operacji, która nie jest pierwszą w bloku, wynika, że należy przesunąć tę operację przed pierwszą operację w bloku. Jeżeli w ścieżce u znajduje się więcej niż jedna operacja zadania j, której przesunięcie w prawo rokuje poprawę rozwiązania, wówczas należy przesunąć operację wykonywaną na maszynie o najniższym numerze.

Algorytm wyznaczenia najlepszego ruchu w V_j

Krok 1. Wykonaj ruch $v=(j,1,\ldots,1)$ w π.

Krok 2. Wyznacz $C_{\max}(\pi)$, ścieżkę krytyczną w $G(\pi)$. Uaktualnij najlepszy ruch.

Krok 3. Wyznacz operację do przesunięcia w prawo.

Krok 4. Jeżeli istnieje, przesuń o jedną pozycję w prawo, idź do Kroku 2, w przeciwnym wypadku STOP.

Niech C_{jk} (Q_{jk}) będzie długością najdłuższej drogi w $G(\pi_v)$ dochodzącej do (odchodzącej od) węzła reprezentującego operację zdania j wykonywaną na maszynie k. Można pokazać, że $O(m)$ czasu zajmuje: (*i*) wyznaczenie $C_{\max}(\pi_v)$, (*ii*) wyznaczenie węzłów, dla których należy uaktualnić wartości C_{jk} oraz Q_{jk} po wykonaniu ruchu pojedynczą operacją w prawo.

Tabela 1

Wyniki badań testowych

Grupa	ave *D*	max *DIV*	ave *DIV*	*CPU*
20×5	5,4	0,80	0,27	2
20×10	5,7	2,12	0,60	8
20×20	5,9	0,18	-1,17	32
50×5	12,8	0,18	0,06	16
50×10	8,0	1,63	0,83	51
50×20	8,4	2,03	1,35	205

5. Algorytmy heurystyczne

W celu oceny zaproponowanego otoczenia zaprojektowano i zaimplementowano algorytm TS oparty na metodzie przeszukiwania z zabronieniami. Został on zaprogramowany w środowisku Visual C++ 2008 i testowany na pierwszych sześciu grupach instancji problemu przepływowego, które zostały zaproponowane przez Taillarda [8]. W zestawie tym dla każdej pary $n \times m$: 20×5, 20×10, 20×20, 50×5, 50×10, 50×20 znajduje się 10 przykładów testujących o znanych optymalnych wartościach funkcji celu dla przypadku permutacyjnego. Rozwiązanie początkowe dla algorytmu TS zostało wygenerowane algorytmem NEH [9]. Dla każdej instancji problemu algorytm wykonał 1000 iteracji na komputerze z procesorem Intel Core 2 Duo 2.66 GHz.

Dla każdego przykładu wyznaczono następujące wielkości:

- π^{TS} – najlepsze rozwiązanie wygenerowane algorytmem TS,
- $DIV(\pi^{TS})$ – 100% $(C_{\max}(\pi^{TS})-C^{opt})/C^{opt}$ względna różnica wartości funkcji celu rozwiązania wygenerowanego algorytmem TS i wartości optymalnej wyznaczonej dla problemu permutacyjnego,
- *CPU* – czas obliczeń algorytmu TS,
- ave *D* – średnia liczba aktualizowanych wartości C_{jk} i Q_{jk} przypadająca na jeden ruch.

Następnie dla każdej grupy wyznaczono wartości średnie ww. wielkości oraz wyznaczono maksymalną wartość wielkości *DIV*.

Z analizy wyników eksperymentu komputerowego prezentowanych w tabeli 1 wynika, że harmonogramy niepermutacyjne są krótsze od permutacyjnych nawet o 2%. Przewagę harmonogramów niepermutacyjnych szczególnie widać w przypadku grup o dużej liczby maszyn i zadań. Średnia liczba aktualizacji wartości C_{jk} i Q_{jk} jest stosunkowo niewielka, nieznacznie zmniejsza się wraz ze wzrostem liczby maszyn i zwiększa wraz ze wzrostem liczby zadań. Dla ustalonej liczby zadań n średni czas działania algorytmu jest proporcjonalny do m^2.

BIOBLIOGRAFIA

1. Gupta J.N.D., Stafford S.: Flowshop scheduling research after five decades. European Journal of Operational Research, 169, 2006, s. 699–711.
2. Johnson S.M.: Optimal two– and three–stage production schedules with setup times included. Naval Research Logistics Quarterly, 1, 1954, s. 61–68.

3. Garey M.R., Johnson D.S., Sethi R.: The complexity of flowshop and jobshop scheduling. Mathematics of Operations Research, 1, 1976, s. 117–129.
4. Grabowski J., Nowicki E., Smutnicki C.: Metoda blokowa w zagadnieniach szeregowania zadań. EXIT, Warszawa 2003.
5. Low C., Liang Z.: Determining optimal combination of genetic operators for flow shop scheduling problem. International Journal of Advanced Manufacturing Technology, 30, 2006, 302–308.
6. Low C., Yeh J.Y., Huang K.I.: A robust simulated annealing heuristic for flow shop scheduling problem. International Journal of Advanced Manufacturing Technology, 23, 2004, 762–767.
7. Nowicki E., Smutnicki C.: A fast taboo search algorithm for the permutation flow–shop problem. European Journal of Operational Research, 106, 1998, s. 226–253.
8. Grabowski J., Pempera J.: New block properties for the permutation flow–shop problem with application in TS. Journal of the Operational Research Society, 26, 2001, s. 210–220.
9. Grabowski J., Wodecki M.: A very fast tabu search algorithm for the permutation flow shop problem with makespan criterium. Computers and Operations Research 31, 2004, s. 1891–1909.
10. Solimanpur M., Vrat P., Shankar R.: A neuro-tabu search heuristic for the flow shop scheduling problem. Computers and Operations Research 31, 2004, s. 2151–2164.
11. Ying K.C., Liao C.J. (2004). An ant colony system for permutation flow–shop sequencing. Computers and Operations Research 31, 2004, s. 762–767.
12. Tandon M., Cummings P.T. Levan M.D.: Flowshop sequencing with non-permutation schedules. Computers and Chemical Engineering. 15, 1991, s. 601–607.
13. Taillard E.: Benchmarks for basic scheduling problems, European Journal of Operational Research, 64, 1993, s. 278–285.
14. Nawaz E., Enscore E.E., Ham I.: A heuristic algorithm for the m–machine, n–job flow–shop sequencing problem. Omega International Journal of Management Science, 11, 1993, s. 91-95.

Recenzent: Prof. dr hab. inż. Tadeusz Sawik

Abstract

The block properties are successfully applied to a priory eliminate non promising moves from the neighborhood for many local search algorithm of solving scheduling problems. In this paper, for the general flowshop problem, it presents new method of searching the neighborhood, which gives rise to eliminate highly greater set of moves. To validate efficiency of the proposed method, the tabu search algorithm have been implemented and executed on a well-known Taillard's benchmarks. The results of the experiment demonstrates the high efficiency of the proposed method and that near-optimal non-permutation schedules are evidently shortest than optimal schedules for the permutation flow shop problem.