ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 2002
Seria: AUTOMATYKA z. 134 Nr kol. 1554

Aleksander BACHMAN, Adam JANIAK, Andrzej KOZIK, Marcin WINCZASZEK
Politechnika Wrocławska

PRZYBLIŻONE ALGORYTMY ROZWIĄZYWANIA JEDNOMASZYNOWEGO PROBLEMU SZEREGOWANIA ZADAŃ O ZMIENNYCH WARTOŚCIACH

Streszczenie. W niniejszej pracy rozpatrzono jednomaszynowy problem szeregowania zadań przy kryterium maksymalizacji sumy wartości zadań. Wartość zadania jest opisana malejącą funkcją potęgową zależną od jego czasu zakończenia wykonywania. Dla badanego problemu skonstruowano i porównano eksperymentalnie szereg algorytmów heurystycznych typu konstrukcyjnego i typu „popraw".

HEURISTIC ALGORITHMS FOR A SINGLE MACHINE SCHEDULING PROBLEM WITH CHANGEABLE JOB VALUES

Summary. The paper deals with a single machine scheduling problem where the sum of job values should be maximized. A job value is given as a exponentially decreasing function dependent on its completion time. We experimentally compared some heuristic algorithms constructed to solve the problem under consideration.

1. Wprowadzenie

Problem rozpatrywany w niniejszej pracy posiada szerokie zastosowania praktyczne wszędzie tam, gdzie mamy do czynienia ze spadkiem pewnej charakterystycznej wartości zadania wraz z upływem czasu. Przykładowo, problem ten pojawia się przy odzysku części ze starych komputerów, samochodów lub innych produktów zaawansowanych technologicznie. Wspomniane powyżej zastosowanie można dokładniej scharakteryzować następująco. Załóżmy, że mamy do dyspozycji pewną partię wysłużonego sprzętu komputerowego. Komputery te nie nadają się do pracy jako całość ze względu na swój wiek (zużycie, awaryjność, nieefektywność), ale niektóre, jeszcze sprawne i w miarę nowoczesne ich podzespoły (np. dyski twarde, karty sieciowe) mogą zostać użyte w komputerach nowszej

generacji bezpośrednio, po przetworzeniu lub jako części zamienne. Pojawia się więc tutaj problem odzyskania sprawnych części z zestawów komputerowych, tzn. rozmontowanie komputerów na części. Każda odzyskana część posiada pewną wartość rynkową, którą można określić w chwili, kiedy jest ona dostępna do dalszego wykorzystania, tj. w momencie całkowitego jej wymontowania z komputera. Należy znaleźć taką kolejność rozmontowywania komputerów, dla której suma wartości odzyskanych części jest maksymalna.

Formalna definicja problemu opisanego powyżej jest następująca. Dana jest pojedyncza maszyna oraz zbiór $J = \{1,\ldots,n\}$ zawierający n niezależnych i niepodzielnych zadań dostępnych do realizacji w chwili $t = 1$. Każde zadanie i jest scharakteryzowane przez czas jego wykonania p_i oraz funkcję zmiany jego wartości $v_i = \omega_i C_i^{a_i}$, gdzie $\omega_i > 0$ oznacza wartość początkową zadania w chwili $t = 1$, natomiast $a_i < 0$ jest współczynnikiem spadku wartości zadania. Należy znaleźć takie uszeregowanie π, dla którego suma wartości zadań liczona w momentach ich zakończenia wykonywania jest maksymalna, tzn.

$$\sum \omega_{\pi(i)} C_{\pi(i)}^{a_{\pi(i)}} \to \max, \tag{1}$$

gdzie $C_{\pi(i)}$ oznacza czas zakończenia wykonywania zadania umieszczonego na i-tej pozycji w uszeregowaniu π.

Pozostała część pracy została zorganizowana następująco. Rozdział 2 zawiera opis algorytmów, które zostały skonstruowane w celu rozwiązania rozpatrywanego problemu. W rozdziale 3 przedstawiono wyniki analizy eksperymentalnej rozwiązań dostarczonych przez zaproponowane algorytmy. Rozdział 4 zawiera krótkie podsumowanie.

2. Algorytmy heurystyczne

Złożoność obliczeniowa problemu sformułowanego w poprzednim rozdziale jest sprawą otwartą, jednakże przeprowadzone dotychczas badania nad tym problemem pozwalają stwierdzić z dużym prawdopodobieństwem, że jest on NP-trudny. Zatem znalezienie optymalnego algorytmu, który rozwiązuje badany problem w wielomianowym czasie, jest mało prawdopodobne. Wobec tego skonstruowano szereg algorytmów heurystycznych, które znajdują rozwiązanie przybliżone dla badanego problemu. Prezentowane algorytmy zostały podzielone na algorytmy typu „popraw" i algorytmy typu konstrukcyjnego.

2.1. Algorytmy typu popraw

Prezentowane w tym podrozdziale algorytmy służą do poprawy rozwiązania początkowego, które można uzyskać np. przy pomocy algorytmów opisanych w podrozdziale 2.2. Zastosowanie algorytmów typu „popraw" prezentowanych poniżej nigdy nie powoduje zmniejszenia wartości funkcji celu.

Własność opisana poniżej została wykorzystana przy konstrukcji algorytmów opierających się na wymianie sąsiednich zadań.

Własność 1. Jeżeli dla pewnego zadania $i\,(i=1,\ldots,n-1)$ zachodzi $\omega_{\pi(i)}C_{\pi(i)}^{\ \alpha_{\pi(i)}}+\omega_{\pi(i+1)}C_{\pi(i+1)}^{\ \alpha_{\pi(i+1)}}<\omega_{\pi(i+1)}\left(C_{\pi(i-1)}+p_{\pi(i+1)}\right)^{\alpha_{\pi(i+1)}}+\omega_{\pi(i)}C_{\pi(i+1)}^{\ \alpha_{\pi(i)}}$, wtedy zamiana kolejności wykonywania zadań z pozycji i-tej oraz i+1-szej poprawia wartość funkcji celu.

Dowód. Powyższy rezultat można uzyskać poprzez zamianę sąsiednich zadań, jednakże ze względu na ograniczenia objętościowe niniejszej pracy pominięto dowód tej własności.

Algorytm ZamianaP. Działanie algorytmu polega na sprawdzeniu wartości funkcji celu przed i po zamianie dwóch sąsiednich zadań. Jeżeli wartość funkcji celu zostaje zwiększona, wtedy zadania zostają zamienione miejscami, w przeciwnym przypadku kolejność wykonywania zadań pozostaje bez zmian. Zamianę sąsiednich zadań wykonuje się dla zadań z pozycji i-tej oraz i+1-szej dla $i=1,\ldots,n-1$. Złożoność algorytmu wynosi $O(n)$.

Algorytm ZamianaT. Zasada działania tego algorytmu jest podobna do działania Algorytmu SwapForward z tą różnicą, że zamianę sąsiednich zadań wykonuje się dla zadań z pozycji i-1-szej oraz i-tej dla $i=n,\ldots,2$. Złożoność algorytmu wynosi $O(n)$.

Algorytm ZamianaPT. Działanie tego algorytmu polega na wykonaniu w pierwszej kolejności algorytmu ZamianaP, a następnie algorytmu ZamianaT. Złożoność algorytmu wynosi $O(n)$.

Algorytm N-Wstaw. W prezentowanym algorytmie wykorzystano otoczenie typu „wstaw", które dla pewnego rozwiązania bazowego jest tworzone przez wstawienie zadania z pozycji i-tej $(i=1,\ldots,n)$ na pozycję inną niż i-ta przy zachowaniu kolejności wykonywania pozostałych zadań. W otoczeniu tym znajdowane jest najlepsze rozwiązanie, które staje się

rozwiązaniem bazowym dla kolejnego kroku algorytmu, jeżeli poprawia wartość funkcji celu. Operacja opisana powyżej jest wykonywana dopóty, dopóki w wygenerowanym otoczeniu istnieje rozwiązanie lepsze niż rozwiązanie bazowe (pierwszy warunek stopu). Drugim warunkiem stopu jest wykonanie pewnej liczby iteracji algorytmu. Dzięki wykorzystaniu *Szybkiej metody przeglądu otoczenia typu „wstaw"* złożoność obliczeniowa opisywanego algorytmu wynosi $O(n^2)$.

Szybka metoda przeglądu otoczenia typu „wstaw". Podzielmy ruchy generujące otoczenie typu „wstaw" na ruchy przemieszczające zadania w przód oraz w tył. Rozważmy ruchy w przód (dla ruchów w tył postępowanie jest analogiczne). Otoczenie generowane przez te ruchy powstaje przez wstawienie zadania z pozycji *i*-tej (z permutacji bazowej) na kolejne pozycje w permutacji, tj. przez wykonanie *n-i* zamian sąsiednich zadań. Zamiana miejscami dwóch sąsiednich zadań (z pozycji *i*-tej oraz *i*+1-szej) nie zmienia czasu zakończenia wykonywania zadania na pozycji *i*-1-szej oraz czasu rozpoczęcia wykonywania zadania na pozycji *i*+2-giej. W związku z tym, jeżeli w permutacji bazowej znamy wartość funkcji celu V_{przed} dla zadań z pozycji [1; *i*-1], wartość funkcji celu V_{po} dla zadań z pozycji [*i*+2; n] oraz czas zakończenia wykonywania T_0 zadania z pozycji *i*-1-szej, to możemy w czasie O(1) obliczyć wartość funkcji celu dla permutacji po zamianie zadań według wzoru $V = V_{przed} + \omega_{\pi(i+1)}\left(T_0 + p_{\pi(i+1)}\right)^{\alpha_{\pi(i+1)}} + \omega_{\pi(i)}\left(T_0 + p_{\pi(i+1)} + p_{\pi(i)}\right)^{\alpha_{\pi(i)}} + V_{po}$.

Powyższą metodę wykorzystano w algorytmie N-Wstaw oraz w algorytmie NEH, którego opis znajduje się w podrozdziale 2.2.

2.2. Algorytmy konstrukcyjne

Podstawową techniką zastosowaną w algorytmach prezentowanych w tym podrozdziale jest sortowanie zadań według wartości pewnego wyrażenia.

Algorytm Sort f(p)/p. Działanie algorytmu polega na uszeregowaniu zadań według nierosnących wartości wyrażenia $\left(\omega_i \cdot p_i^{\alpha_i}\right)/p_i$. Złożoność algorytmu wynosi $O(n \log n)$.

W kolejnych dwóch algorytmach wykorzystywana jest procedura **SORT_V/T(b,e)**, która sortuje zadania znajdujące się na pozycjach [b,..,e] według nierosnących wartości wyrażenia $\left(\omega_i \cdot \left(C_{\pi(b-1)} + p_i\right)^{\gamma_i}\right)/p_i$.

Algorytm Sort2 f(p)/p

Krok 1. Wykonaj SORT_V/T(1,n), a następnie ZamianaPT.

Krok 2. Podstaw k:=2.

Krok 3. Podstaw i:=1.

Krok 4. Wykonaj SORT_V/T($\lceil (i\cdot n)/k \rceil, \lceil (i+1)\cdot n/k \rceil$). Podstaw i:=i+2.

Krok 5. Jeżeli i<k, to idź do Kroku 4. W przeciwnym wypadku wykonaj ZamianaPT.

Krok 6. Podstaw k:= $k\cdot 2$. Jeżeli k<n/2, to idź do Kroku 3.

Krok 7. Koniec.

Złożoność obliczeniowa powyższego algorytmu wynosi $O(n\log^2 n)$.

Poniższy algorytm stanowi modyfikację poprzedniego algorytmu.

Algorytm Sort3 f(p)/p

Krok 1. Wykonaj SORT_V/T(1,n), a następnie ZamianaPT.

Krok 2. Podstaw k:=2.

Krok 3. Podstaw i:=1.

Krok 4. Wykonaj SORT_V/T($\lceil (i\cdot n)/k \rceil, \lceil (i+1)\cdot n/k \rceil$). Podstaw i:=i+2.

Krok 5. Jeżeli i<k, to idź do Kroku 4. W przeciwnym wypadku wykonaj ZamianaPT.

Krok 6. Podstaw i:=1.

Krok 7. Wykonaj SORT_V/T($\lceil (i\cdot n)/k - n/(2\cdot k) \rceil, \lceil (i\cdot n)/k + n/(2\cdot k) \rceil$). Podstaw i:=i+2.

Krok 8. Jeżeli i<k-1, to idź do Kroku 7. W przeciwnym wypadku wykonaj ZamianaPT.

Krok 9. Podstaw k:= $k\cdot 2$. Jeżeli k<n/2, to idź do Kroku 3.

Krok 10. Koniec.

Złożoność obliczeniowa algorytmu opisanego powyżej wynosi $O(n\log^2 n)$.

Algorytm MinStrata. Ze zbioru zadań nie uszeregowanych wybierz zadanie, dla którego wyrażenie $\left(a_j\cdot \varpi_j\cdot t^{a_j-1}\right)/p_j$ osiąga najmniejszą wartość i wstaw je na pierwszą wolną pozycję w permutacji. Powtarzaj powyższą czynność, dopóki wszystkie zadania nie zostaną uszeregowane. Złożoność algorytmu wynosi $O(n^2)$.

Algorytm !SWAP opisany poniżej konstruuje rozwiązanie w oparciu o własność niewymienialności zadania.

Definicja. Zadanie i nazywamy niewymienialnym, jeżeli zachodzi

$$\forall_{i<j} \omega_{\pi(i)} C_{\pi(i)}^{\alpha_{\pi(i)}} + \omega_{\pi(j)} \left(C_{\pi(i)} + p_{\pi(j)}\right)^{\alpha_{\pi(i)}} > \omega_{\pi(j)} C_{\pi(j)}^{\alpha_{\pi(j)}} + \omega_{\pi(i)} \left(C_{\pi(j)} + p_{\pi(i)}\right)^{\alpha_{\pi(i)}} \quad (2)$$

tzn. wymiana zadania i z dowolnym innym zadaniem j uszeregowanym w permutacji za zadaniem i ($i < j$) jest nieopłacalna.

Istnieje co najwyżej jedno zadanie niewymienialne, a to oznacza, że jeżeli dla zadania n zachodzi (2) (zadanie n jest niewymienialne), to dla dowolnego innego zadania $k<n$ nierówność (2) nie jest spełniona.

Algorytm !SWAP (Zamiana)

Krok 1. Spośród nie uszeregowanych zadań wybierz zadanie niewymienialne. Jeżeli takie zadanie nie istnieje, to wybierz dowolne zadanie (np. za pomocą algorytmu MinStrata).

Krok 2. Wybrane zadanie wstaw na koniec permutacji.

Krok 3. Jeżeli istnieją nie uszeregowane zadania, to idź do Kroku 1.

Złożoność algorytmu wynosi $O(n^3)$.

Algorytm NEH (Nawaz, Enscore, Ham [3]). Utwórz listę zadań oczekujących na uszeregowanie. Kolejność tych zadań na liście może być dowolna, np. uzyskana przez dowolny algorytm konstrukcyjny. W prezentowanej tutaj implementacji kolejność zadań na liście jest opisana przez naturalną permutację zadań lub przez rozwiązania uzyskane przy pomocy algorytmów Sort f(p)/p oraz MinStrata. Dla zadań kolejno pobieranych z listy należy, w tworzonej permutacji, znaleźć taką pozycję, dla której wartość funkcji celu dla tej permutacji jest maksymalna. Wykorzystując *Szybką metodę przeglądu otoczenia typu „wstaw"*, wybór takiej pozycji można zrealizować w O(n) krokach. Złożoność algorytmu wynosi $O(n^2)$.

3. Eksperyment obliczeniowy

Celem przeprowadzonego eksperymentu obliczeniowego było porównanie jakości rozwiązań generowanych przez prezentowane algorytmy. Jako miarę jakości (*MJ*) przyjęto średnie odchylenie wartości funkcji celu rozwiązania otrzymanego przez badany algorytm od

najlepszej z wartości funkcji celu uzyskanej przez wszystkie algorytmy dla danej instancji problemu, czyli:

$$MJ = \frac{1}{N} \cdot \sum_{j=1}^{N} \frac{F_j^* - F_j}{F_j^*} \cdot 100\%, \qquad (3)$$

gdzie N - liczba przebadanych instancji problemu, F_j^* - wartość najlepszego uzyskanego rozwiązania dla instancji j, natomiast F_j - wartość rozwiązania danego algorytmu. Eksperyment przeprowadzono dla trzech rozmiarów problemu $n = \{30, 250, 500\}$.

Tablica 1

Porównanie algorytmów dla parametrów $a_i \in [-1;0)$

	$p_i \in (0;1]$			$p_i \in [1;10]$		
	$n = 30$	$n = 250$	$n = 500$	$n = 30$	$n = 250$	$n = 500$
NEH	0.213%	0.815%	1.026%	0.329%	0.759%	0.812%
Sort f(p)/p + NEH	0.004%	0.044%	0.066%	0.007%	0.049%	0.077%
MinStrata + NEH	0.004%	0.003%	0.004%	0.016%	0.028%	0.028%
Sort f(p)/p + ZamianaPT	0.097%	2.259%	4.177%	0.155%	1.727%	2.789%
Sort2 f(p)/p	0.007%	0.569%	1.577%	0.023%	0.303%	0.809%
Sort3 f(p)/p	0.005%	0.294%	0.871%	0.028%	0.087%	0.317%
Grecki	**13.340%**	**18.360%**	**18.580%**	**8.923%**	**14.520%**	**16.190%**
!SWAP	0.002%	0.003%	0.004%	0.022%	0.029%	0.031%
!SWAP + N-Wstaw	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%
MinStrata	0.288%	0.139%	0.063%	0.161%	0.069%	0.083%
MinStrata + ZamianaPT	0.007%	0.006%	0.007%	0.020%	0.039%	0.038%

Dla $n = 30$ jako F_j^* przyjęto wartość rozwiązania optymalnego, uzyskanego metodą podziału i ograniczeń. Wartości parametrów a_i, ω_i oraz p_i były generowane z rozkładem normalnym. Dla wszystkich wygenerowanych instancji parametr ω_i był generowany z przedziału $(0;6]$. Parametry a_i były generowane z trzech następujących przedziałów $[-1;0)$, $[-5;-1]$ oraz $[-5;0)$, natomiast parametry p_i były generowane z przedziałów $(0;1]$, $[1;10]$ oraz $(0;10]$. Uzyskane rezultaty zostały przedstawione w tablicach 1 – 3. Dodatkowego omówienia wymagają zastosowane nazwy algorytmów. Nazwy typu alg1 + alg2 oznaczają, że

rozwiązanie uzyskane przez algorytm alg1 staje się rozwiązaniem początkowym algorytmu alg2. W algorytmie NEH lista zadań jest określona przez permutację naturalną. Algorytm Grecki został opisany w pracy [4].

Tablica 2

Porównanie algorytmów dla parametrów $a_i \in [-5;-1]$

	$p_i \in (0;1]$			$p_i \in [1;10]$		
	$n = 30$	$n = 250$	$n = 500$	$n = 30$	$n = 250$	$n = 500$
NEH	0.119%	0.066%	0.050%	1.491%	0.136%	3.973%
Sort f(p)/p + NEH	0.010%	0.003%	0.004%	0.053%	0.387%	0.565%
MinStrata + NEH	0.006%	0.000%	0.001%	0.050%	0.453%	1.491%
Sort f(p)/p + ZamianaPT	0.076%	0.039%	0.051%	0.936%	9.615%	13.360%
Sort2 f(p)/p	0.011%	0.012%	0.014%	0.074%	0.771%	1.999%
Sort3 f(p)/p	0.010%	0.004%	0.006%	0.053%	0.293%	0.368%
Grecki	**43.170%**	**80.100%**	**74.380%**	**27.840%**	**45.770%**	**56.910%**
!SWAP	0.062%	0.001%	0.001%	0.050%	0.247%	0.809%
!SWAP + N-Wstaw	0.000%	0.000%	0.000%	0.004%	0.000%	0.095%
MinStrata	0.534%	0.934%	1.223%	2.794%	5.385%	5.140%
MinStrata + ZamianaPT	0.009%	0.002%	0.008%	0.072%	0.519%	1.655%

Eksperymenty wykazały szczególną wrażliwość jakości rozwiązań na generowane wartości parametrów zadań. W szczególności zaobserwowano duże różnice jakości otrzymanych rozwiązań dla parametru a_i generowanego z przedziałów $[-1;0)$ oraz $[-5;-1]$ (tabela 1 oraz 2). W ogólności, wyniki uzyskane dla $a_i \in [-1;0)$ są dużo lepsze od wyników uzyskanych dla $a_i \in [-5;-1]$. Na jakość rozwiązań miał także wpływ zakres wartości parametru p_i, tzn. dla większości badanych algorytmów wyniki uzyskane dla $p_i \in (0;1]$ są lepsze od wyników uzyskanych dla $p_i \in [1;10)$.

Tablica 3

Porównanie algorytmów dla parametrów $a_i \in [-5;0), p_i \in (0;10]$

	$n = 30$	$n = 250$	$n = 500$
NEH	0.404%	0.233%	0.120%
Sort f(p)/p + NEH	0.045%	0.031%	0.015%
MinStrata + NEH	0.031%	0.011%	0.008%
Sort f(p)/p + ZamianaPT	0.282%	0.387%	0.305%
Sort2 f(p)/p	0.058%	0.054%	0.057%
Sort3 f(p)/p	0.035%	0.016%	0.011%
Grecki	**16.290%**	**45.090%**	**61.250%**
!SWAP	0.031%	0.016%	0.007%
!SWAP + N-Wstaw	0.000%	0.000%	0.000%
MinStrata	0.857%	0.679%	1.851%
MinStrata + ZamianaPT	0.114%	0.027%	0.016%

Średni błąd rozwiązań otrzymanych przez opisane w rozdziale 2 algorytmy wynosi ułamek procenta, co potwierdza ich przydatność i dobrą jakość, tzn. pozwalają one na znalezienie rozwiązania bardzo bliskiego optymalnemu w bardzo krótkim czasie. Szczególnie efektywny jest algorytm !SWAP + N-Wstaw, który w prawie wszystkich przeprowadzonych testach znajdował rozwiązanie optymalne kosztem niewielkiego wzrostu nakładów obliczeniowych w stosunku do pozostałych algorytmów. Algorytm Grecki [4] daje wyniki znacząco gorsze od wyników uzyskanych przez pozostałe algorytmy (błąd rzędu kilkunastu – kilkudziesięciu procent).

4. Podsumowanie

W pracy rozpatrzono jednomaszynowy problem szeregowania zadań o zmiennych wartościach. Wartości zadań były opisane malejącą funkcją potęgową zależną od czasu zakończenia jego wykonywania. Opierając się na przypuszczeniu, że badany problem jest NP-trudny, skonstruowano i eksperymentalnie przebadano 6 algorytmów typu konstrukcyjnego oraz 4 algorytmy typu „popraw". Odniesieniem dla skonstruowanych algorytmów był Algorytm Grecki zaprezentowany w pracy [4], który był pierwszym algorytmem rozwiązania

badanego problemu. Algorytmy zaprezentowane w niniejszej pracy potwierdziły swoją wyższość w stosunku do Algorytmu Greckiego.

LITERATURA

1. Graham R. L., Lawler E. L., Lenstra J. K., Rinnooy Kan A. H. G.: Optimization and approximation in sequencing and scheduling: a survey. Annals of Discrete Mathematics, vol. 5, 1979, s. 287-326.
2. Smith W. E.: Various optimizers for single-stage production, Naval Research Logistics Quarterly, vol. 3, 1956, s. 59-66.
3. Nawaz M., Enscore Jr. E. E., Ham I.: A heuristic algorithm for the m-machine n-job flow-shop sequencing problem, OMEGA International Journal of Management Science, vol. 11, 1983, s. 91-95.
4. Voutsinas T. G., Pappis C. P.: Scheduling jobs with values exponentially deteriorating over time, Raport Techniczny Uniwersytetu w Pireusie, 2000.

Recenzent: Prof. zw. dr hab. inż. Jan Węglarz

Abstract

A single machine problem of scheduling jobs with changeable values was considered in this paper. A job value was given by an exponential function dependent on job completion time. The objective function was the maximization of the total job values calculated at their completion times. Based on the assumption that the considered problem is NP-hard, we constructed and experimentally compared several heuristic algorithms. An extensive experimental analysis showed that in general the algorithms generated solutions just a few percent worse than the optimal one found by a Branch & Bound method. An exception from this statement is the Greek algorithm, which generated solutions more than 10% worse than the optimal one. In general, the solutions obtained by Greek algorithm are even 80% worse than the best solutions found by the algorithms constructed in this paper.