ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1996
Seria: AUTOMATYKA z.117 Nr kol. 1337

Stanisław GAWIEJNOWICZ
Uniwersytet im. A. Mickiewicza, Poznań

SZEREGOWANIE ZADAŃ NA PROCESORZE O ZMIENNEJ PRĘDKOŚCI

Streszczenie. W pracy rozważany jest problem minimalizacji długości uszeregowania na jednym procesorze o zmiennej prędkości, opisanej funkcją od ilości już wykonanych zadań. Podano kilka wielomianowych algorytmów znajdujących optymalne uszeregowania w zależności od własności tej funkcji.

MAKESPAN SCHEDULING ON A SINGLE PROCESSOR WITH VARYING SPEED

Summary. In the paper the makespan scheduling problem on a single processor is considered. The speed of a processor depends on the number of already processed jobs and is described by a function. Several polynomial-time algorithms for finding an optimal schedule in dependency of features of the function are given.

1. Wstęp

W wielu zagadnieniach praktycznych często spotykamy się z sytuacją, kiedy ten sam zbiór zadań (prac, czynności itp.) wykonywany w identycznych (lub bardzo podobnych) warunkach różni się, niekiedy dość znacznie, czasami wykonywania poszczególnych zadań.

Zjawisko to, nazywane w literaturze wydłużaniem (ang. *deterioration*) czasu wykonywania zadania, jest ostatnio przedmiotem intensywnych badań. W literaturze reprezentowane są różne podejścia do opisanego problemu, z których dwa są dominujące: szeregowanie zadań o zmiennych czasach wykonywania oraz szeregowanie zadań na procesorach o zmiennej prędkości.

W pierwszym przypadku czas wykonywania zadania jest opisany pewną funkcją zależną od czasu rozpoczęcia danego zadania. Problemy tego typu były rozważane w pracach Browna i Yechialiego [2], Gawiejnowicza i Pankowskiej [4], Gupty i Gupty [5], Ho i innych [7], Kubiaka i van de Veldego [8], Mosheiova [9].

Lista zagadnień, w których znajdują wykorzystanie modele opisujące wydłużanie czasu

wykonywania zadań, obejmuje problematykę prac medycznych (rozdział zasobów w czasie epidemii, szeregowanie żądań obsługi zespołów szybkiej pomocy medycznej), prac porządkowych i pielęgnacyjnych, zagadnień naprawy urządzeń, walki z ogniem, procesów uczenia się itp. Jednym z przykładów ściśle informatycznych jest model obsługi cyklicznej kolejki przez pojedynczy serwer (ang. *single server-cyclic queue*).

Opisując zbiór procesorów przyjmuje się zazwyczaj, że używane procesory należą do jednego z trzech rodzajów: P (maszyny identyczne), Q (maszyny jednorodne) lub R (maszyny dowolne) oraz że prędkość procesora jest zawsze *stała* (por. np. Błażewicz i inni [1]). W pewnych zastosowaniach naturalne jednakże wydaje się przyjęcie założenia, że prędkość procesora (maszyny) może się *zmieniać* w czasie. Kilku autorów rozważało modele, w których zadania są przetwarzane ze zmienną prędkością. Przykładami są prace Drora i innych [3] oraz Węglarza [10], gdzie prędkość przetwarzania (ang. *processing rate*) zależy od ilości aktualnie wykonywanych zadań (Dror i inni) lub podzielnego w sposób ciągły zasobu (Węglarz).

Idee tego typu legły u podstaw drugiego podejścia do problemu wydłużania czasu wykonywania zadania, które opiera się na spostrzeżeniu, że zmienność czasu wykonywania zadania może być skutkiem zmienności prędkości procesora.

Modele szeregowania na procesorach o zmiennej prędkości znajdują zastosowanie w zagadnieniach dotyczących pracy zespołów urządzeń o wspólnym źródle zasilania, maszyn z własnymi, odnawialnymi źródłami zasilania (wózki akumulatorowe, urządzenia o napędzie elektrycznym) czy w analizie pracy zespołów ludzkich (grupy robotników, pielęgniarek) itp.

Celem niniejszej pracy jest zbadanie modelu szeregowania zadań, w którym prędkość procesora jest opisana pewną funkcją v. Bez straty ogólności możemy przyjąć, że prędkość ta zmienia się w przedziale $[0, 1]$. Startując od pewnej wartości początkowej, zmienia się po zakończeniu każdego zadania i staje się równa pewnej wartości końcowej po wykonaniu pewnej liczby zadań. Wówczas następuje przerwa (procesor wtedy nie pracuje), po czym znowu v zmienia się pomiędzy 0 a 1 itd. Specyficzne własności funkcji v zależą od postawionego problemu.

Powyższy model można formalnie opisać następująco: niech dany będzie zbiór n niezależnych, niepodzielnych zadań, z których każde składa się tylko z jednej operacji.

W celu uproszczenia zapisu zdefiniujemy dodatkowo dwa zadania, o numerach 0 i $n+1$, z zerowymi czasami wykonywania. Niech $J = \{1, 2, \ldots, n\}$ oznacza zbiór numerów zadań, a J^* będzie jednym ze zbiorów $\bar{J}$, J lub $\underline{J}$, gdzie $\underline{J} = \{0\} \cup J$, $\bar{J} = J \cup \{n+1\}$. Niech $v : J^* \to [0,1]$ będzie funkcją, która określa prędkość procesora, w chwili rozpoczęcia j-ego zadania, w zależności od liczby już wykonanych zadań. Przyjmujemy, że prędkość ta nie zmienia się w trakcie wykonywania danego zadania i jest opisana przez warunki podane oddzielnie dla każdego konkretnego przypadku.

Niech k oznacza liczbę zadań, które mogą być wykonane bez przerwania pracy procesora, tj. do chwili, kiedy jego prędkość osiągnie wartość końcową (tzn. 0 lub 1).

Niech p_j, $p_{(j)}$ i $p_{[j]}$ oznaczają, odpowiednio, *początkowy* czas wykonywania j-ego zadania, j-ty największy czas wykonywania oraz j-ty w uszeregowaniu czas wykonywania, $j \in J$. Niech b będzie długością przerwy w pracy procesora. W zagadnieniach praktycznych omawianego typu przerwy są stałej długości; stąd, bez straty ogólności, możemy założyć, że $b = 0$ (w przeciwnym przypadku powinniśmy do ostatecznej długości uszeregowania dodać stałą b tyle razy, ile było przerw w pracy).

Przy powyższych założeniach *względny* czas wykonywania $\tilde{p}_{[j]}$ j-ego zadania w uszeregowaniu jest równy

$$\tilde{p}_{[j]} = \frac{p_{[j]}}{v(j)},$$

gdzie $j \in J$; w dalszym toku będziemy pisać v_j zamiast $v(j)$.

Interesować nas będzie kolejność zadań minimalizująca długość uszeregowania

$$C_{max} = \sum_{j=1}^{n} \tilde{p}_{[j]}.$$

Przykładami problemów praktycznych, w których może znaleźć zastosowanie prezentowany model są np. ciężka praca fizyczna ludzi, kiedy wykonanie każdego kolejnego zadania powoduje zmęczenie pracujących, tak że następne zadanie jest wykonywane z mniejszą prędkością, co w efekcie powoduje wydłużenie czasu wykonywania tego zadania (przypadek, kiedy v startuje od 1 i jest malejąca) lub przetwarzanie na maszynie, której temperatura rośnie w trakcie pracy aż do momentu krytycznego, po którym dalszy wzrost temperatury mógłby ją zniszczyć (przypadek, kiedy v startuje od 0 i jest rosnąca). W obu przypadkach po pewnym czasie wymagana jest przerwa.

2. Algorytmy wielomianowe

W dalszym toku podamy kilka algorytmów o wielomianowym czasie wykonania, oddzielnie dla dwu możliwych przypadków w zależności od relacji pomiędzy n i k.

2.1. *Przypadek $k \geq n$*

Oczywiste jest, że w tym przypadku wystarczy rozważyć sytuację, kiedy $n = k$.

Lemat ([6]). *Niech dane będą dwa ciągi liczb (a_j) oraz (b_j). Suma*

$$\sum_j a_j b_j$$

iloczynów odpowiadających sobie elementów obu ciągów jest najmniejsza wtedy, kiedy ciągi (a_j) i (b_j) są monotoniczne w przeciwnym sensie.

Udowodnimy teraz następujące

Twierdzenie 2..1. *Niech $k = n$ oraz $v : \bar{J} \to [0, 1]$ będzie malejącą funkcją taką, że*

$$v(1) = 1, \quad v(k+1) = 0. \tag{1}$$

Wtedy uszeregowanie o minimalnej długości jest otrzymywane przez uporządkowanie zadań w nierosnącej kolejności początkowych czasów wykonywania $p_{[j]}$.

Dowód. Jako ciąg (a_j) weźmy ciąg odwrotności prędkości procesora $(\frac{1}{v_j})$, a jako ciąg (b_j) – ciąg czasów wykonywania $(p_{[j]})$. Ponieważ w tym przypadku

$$C_{max} = \sum_{j=1}^{n} \tilde{p}_{[j]} = \sum_{j=1}^{k} p_{[j]} \cdot \frac{1}{v_j}$$

oraz ciąg $(\frac{1}{v_j})$ jest niemalejący, więc C_{max} będzie najmniejsze (na mocy lematu) wtedy, gdy ciąg $(p_{[j]})$ będzie nierosnący. □

Z powyższego wynika, że ten przypadek jest rozwiązywany przez algorytm (równoważny powszechnie znanemu LPT) o czasowej złożoności $O(n \log n)$.

Twierdzenie 2. 2. *Niech $k = n$ oraz $v : \bar{J} \to [0, 1]$ będzie rosnącą funkcją taką, że*

$$v(0) = 0, v(k) = 1. \tag{2}$$

Wtedy uszeregowanie o minimalnej długości jest otrzymywane przez uporządkowanie zadań wg nierosnącej kolejności początkowych czasów wykonywania $p_{[j]}$.

Dowód jest analogiczny jak w poprzednim przypadku. Zauważmy, że algorytm ten jest równoważny algorytmowi SPT, także o czasowej złożoności $O(n\log n)$.

2.2. Przypadek $k < n$

Załóżmy, że ciąg (p_j) jest dany w postaci listy wejściowej L oraz że v jest monotoniczna. Ponieważ istnieje dokładnie $m = (n \mod k) + 1$ momentów czasu, kiedy prędkość procesora jest równa 1, możemy podzielić listę L na m podlist, $L_1, L_2, \ldots, L_m$, w których zadania są uporządkowane odpowiednio do prędkości procesora.

Algorytm A oparty na tej idei można sformułować następująco:

```
begin
  sort(L);
  m := div(n,k)+1;
  for j := 1 to m do
        Lj := ∅;
  while L ≠ ∅ do begin
        for j := 1 to m do
          if L ≠ ∅ then begin
              weź jako p pierwszy element listy L;
              Lj := Lj ∪ { p };
              L := L \ { p };
            end
          else exit;
        end;
  for j := 1 to m do
        L := L ∪ Lj;
end
```

gdzie $sort(L)$ oznacza sortowanie nierosnąco (niemalejąco) elementów listy L, jeżeli v jest malejąca (rosnąca), $\emptyset$ – zbiór pusty oraz `div(n,k)` – iloraz całkowity z dzielenia n przez k. Na wyjściu algorytmu lista L zawiera początkowe czasy wykonywania zadań we właściwym (optymalnym) porządku. Agorytm ten, tak jak poprzednie, ma złożoność czasową rzędu $O(n\log n)$. Oznaczmy przez A_1 (A_2) wersję algorytmu A z sortowaniem nierosnącym (niemalejącym). Zachodzą wówczas następujące

Twierdzenie 2. 3. *Niech $k < n$ oraz $v : J \to (0,1]$ będzie kawałkami stałą, malejącą funkcją taką, że v jest malejąca w każdym przedziale między rozpoczęciem (m_1k+1)-ego zadania a zakończeniem $(m_1+1)k$-ego zadania oraz*

$$v(m_1k+1) = 1, \tag{3}$$

$$0 < v((m_1+1)k) \ll 1, \tag{4}$$

gdzie $m_1 = 0,1,2,\ldots,(n \bmod k)$. Wtedy algorytm A_1 znajduje uszeregowanie o minimalnej długości.

Twierdzenie 2. 4. *Niech $k < n$ oraz $v : J \to (0,1]$ będzie kawałkami stałą, rosnącą funkcją taką, że v jest rosnąca w każdym przedziale między rozpoczęciem (m_1k+1)-ego zadania a zakończeniem $(m_1+1)k$-ego zadania oraz*

$$0 < v(m_1k+1) \ll 1, \tag{5}$$

$$v((m_1+1)k) = 1, \tag{6}$$

gdzie $m_1 = 0,1,2,\ldots,(n \bmod k)$. Wtedy algorytm A_2 znajduje uszeregowanie o minimalnej długości.

Dowód. Niech v będzie kawałkami stałą, malejącą funkcją określoną jak w twierdzeniu 2..3. oraz $n = mk + r$. Przedstawmy C_{max} jako sumę $m+1$ sum w następujący sposób:

$$C_{max} = \sum_{j=1}^{k} \tilde{p}_{[j]} + \sum_{j=k+1}^{2k} \tilde{p}_{[j]} + \ldots + \sum_{j=mk+1}^{mk+r} \tilde{p}_{[j]} = S_{1,k} + S_{k+1,2k} + \ldots + S_{mk+1,mk+r},$$

gdzie S_{i_1,i_2} oznacza sumę $\sum_{j=i_1}^{i_2} \tilde{p}_{[j]}$. Ponieważ $\tilde{p}_{[j]} = p_{[j]} \cdot \frac{1}{v_j}$ oraz ciąg $(\frac{1}{v_j})$ jest kawałkami rosnący, zatem C_{max} będzie najmniejsza wtedy, gdy w sumach S_{i_1,i_2} (dla $i_1 = 1, k+1,\ldots,$mk+1 oraz $i_2 = k, 2k, \ldots, mk+r$) m największych wartości spośród $p_{[j]}$ będzie pomnożonych przez $v_j = 1$; kolejnych m (co do wielkości) wartości $p_{[j]}$ – przez następne, najbliższe 1, wartości prędkości itd. Kontynuując to postępowanie otrzymamy $m+1$ ciągów uporządkowanych odpowiednio do wartości v_j w taki sposób, że łączna suma wszystkich $p_{[j]}$ (równa C_{max}) będzie minimalna. Dowód dla przypadku, kiedy v jest kawałkami stała i rosnąca, jest podobny. □

Rozważmy teraz przypadek, kiedy $v, v : J \to (0,1]$ jest *dowolną* funkcją. Możemy otrzymać optymalne uszeregowanie poprzez posortowanie początkowych czasów wykonywania zadań, a następnie poprzestawianie ich odpowiednio do wartości v w taki sposób,

aby suma $\sum_{j=1}^{n} p_{[j]} \cdot \frac{1}{v_j}$ była minimalna. Algorytm B oparty na tej idei można sformułować następująco (-- oznacza komentarz):

begin

posortuj nierosnąco p_j; -- utworzenie listy $p_{(j)}$

for $j := 1$ **to** n **do begin**

$s_{1,j} := v_j$;

$s_{2,j} := j$

end;

posortuj nierosnąco $s_{1,j}$;

przenumeruj listę s *odpowiednio do wartości* $s_{1,(j)}$;

for $j := 1$ **to** n **do**

$L_j := 0$;

for $j := 1$ **to** n **do**

wstaw $p_{(j)}$ *na* $s_{2,j}$*-te miejsce w liście* L -- utworzenie listy $p_{[j]}$

end

Algorytm B także ma złożoność czasową $O(n \log n)$. Zachodzi następujące

Twierdzenie 2. 5. *Niech* $v : J \to (0, 1]$ *będzie dowolną funkcją. Wtedy algorytm* B*, zdefiniowany jak wyżej, znajduje uszeregowanie o minimalnej długości.*

Dowód wynika bezpośrednio z lematu oraz analizy algorytmu B. Zauważmy, że ponieważ w algorytmie B nie występuje k, to twierdzenie 2..5. zachodzi dla obu, $k \geq n$ oraz $k < n$, przypadków.

3. Podsumowanie

Modele teorii szeregowania zadań, w których pewne parametry zbioru zadań czy też maszyn są opisane zależnościami ciągłymi, odgrywają coraz większą rolę, zwłaszcza w zastosowaniach.

W pracy przedstawiono nowy model szeregowania zadań na procesorze o zmiennej prędkości, opisanej pewną funkcją zależną od liczby już wykonanych zadań. Pokazano, że dla monotonicznych oraz dowolnych funkcji prędkości, jak również dla dowolnej relacji

pomiędzy liczbą zadań a liczbą przerw w pracy procesora istnieją wielomianowe algorytmy optymalne.

Dodając nowe założenia dotyczące np. wag zadań, czasów gotowości, podzielności itp. można otrzymać model o bogatszych własnościach. Interesujące także wydaje się przeniesienie idei procesora o zmiennej prędkości do zagadnień szeregowania na procesorach dedykowanych.

LITERATURA

1. Błażewicz J., Ecker K., Schmidt G., Węglarz J.: Scheduling in Computer and Manufacturing Systems. Springer-Verlag 1993.

2. Browne S., Yechiali U.: Scheduling deteriorating jobs on a single processor. Operations Research, vol. 38, 1990, pp. 495-498.

3. Dror M., Stern H.J., Lenstra J.K.: Parallel machine scheduling: processing rates dependent on number of jobs in operation. Managment Science, vol. 33, 1987, pp. 1001-1009.

4. Gawiejnowicz S., Pankowska L.: Scheduling jobs with varying processing times. Information Processing Letters, vol. 54, 1995, pp. 175-178.

5. Gupta J.N.D., Gupta S.K.: Single facility scheduling with nonlinear processing times. Computers and Industrial Engineering, vol. 14, 1988, pp. 387-393.

6. Hardy G.H., Littlewood J.E., Polya G.: Inequalities. University Press 1934.

7. Ho K.I.-J., Leung J.Y.-T., Wei W.-D.: Complexity of scheduling tasks with time-dependent execution times. Information Processing Letters, vol. 48, 1993, pp. 315-320.

8. Kubiak W., van de Velde S.L.: Scheduling deteriorating jobs to minimize makespan. Report LPOM-94-12, Labolatory of Production and Operations Managment, Department of Mechanical Engineering, University of Twente 1994.

9. Mosheiov G.: Scheduling jobs under simple linear deterioration. Computers and Operational Research, vol. 21, 1994, pp. 653-659.

10. Węglarz J.: Multiprocessor scheduling with memory allocation - a deterministic approach. IEEE Transactions on Computers, vol. C-29, 1980, pp. 703-710.

Recenzent: Dr hab. inż. Mirosław Zaborowski, prof. Pol.Śl.

Wpłynęło do Redakcji do 30.06.1996 r.

Abstract

In some applications it seems natural to assume that the speed of a processor can change in time. In the literature exist several possible approaches to the problem. The approach presented by us is based on the assumption that the change of the speed of the processor is due to the number of already completed jobs.

In the paper is presented the new model of scheduling in which the speed of a processor is described by a function v . Without loss of generality we can assume that the speed varies in $[0, 1]$ interval. Starting from an initial value it changes after completion of each job and becomes equal to an end value after executing some number of jobs. Then there is a break for some time, when the processor does not work, after that the speed again varies between 0 and 1, etc. Specific properties of the function v depend on the stated problem.

The above model can be precised as follows: we are given n independent, not preemptable jobs, each of which consists of one operation only. Let $J = \{1, 2, \ldots, n\}$ denote a set of job indices and let $v : J \to [0, 1]$ be a function which specifies the speed of the processor (at the moment of the beginning of the j-th job) in the dependence on the number of already executed jobs. We assume that the speed does not change during execution of a job and is described by the set of conditions separately for each particular case. Let k denote a number of jobs which can be executed without breaking the work of the processor, i.e. up to the time when its speed reaches an end value (0 or 1). Let b be the length of each break in the work of the processor. We assume that the length b of the break is constant and thus, without loss of generality, we can suppose that $b = 0$. Let p_j and $p_{[j]}$ denote, respectively, the starting processing time of the j-th job and the processing time of the j-th job in a schedule, $j \in J$.

Under the above assumptions the relative processing time $\bar{p}_{[j]}$ of the j-th job in a schedule is equal to $\bar{p}_{[j]} = \frac{p_{[j]}}{v(j)}$ where $j \in J$. We are interested in an order of the jobs which minimizes the maximum completion time of all jobs $C_{max} = \sum_{j=1}^{n} \bar{p}_{[j]}$.

Using a well-known lemma about minimizing the sum of products of the corresponding elements of two number sequences $(a_j), (b_j)$ we prove theorems concerning the optimal schedule for a single processor of the above type and as for monotonic as for any function v, separately for the case $n \geq k$ and $n < k$. For the each case we give the polynomial-time algorithm for finding the optimal schedule.