ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1998
Seria: AUTOMATYKA z. 123 Nr kol. 1389

Jerzy R. NAWROCKI, Adam CZAJKA
Instytut Informatyki, Politechnika Poznańska

SZEREGOWANIE ZADAŃ W SYSTEMACH SILNIE UWARUNKOWANYCH CZASOWO METODĄ CYKLICZNEGO OBCIĄŻENIA

Streszczenie. W artykule przedstawione są wyniki badań dotyczących zastosowania metody minimalizacji obciążeń cyklicznych do szeregowania zadań silnie uwarunkowanych czasowo o okresach tworzących postęp binarny. Pokazano, że problem taki jest silnie NP-trudny i w związku z tym algorytm o złożoności wielomianowej nie istnieje. W tej sytuacji zaproponowana została prosta heurystyka zachłanna, która - jak wynika z przeprowadzonych eksperymentów obliczeniowych - zachowuje się lepiej od innych tego typu znanych heurystyk. Zastosowana do szeregowania zadań w samolocie F-16 dała w bardzo krótkim czasie optymalne rozwiązanie.

PERIODIC LOADING APPROACH TO SCHEDULING HARD REAL-TIME TASKS

Summary. In the paper scheduling hard real-time tasks by minimising periodic loading is studied. It is assumed that task periods belong to a binary geometrical progression 2^0p_0, 2^1p_0, 2^2p_0, .. 2^kp_0. It is shown that such a problem is strongly NP-hard. Thus, a simple greedy heuristics is proposed, which - due to some computational experiments - is pretty effective and better than other greedy heuristics for the periodic loading problem. Applied to scheduling tasks for F-16 the heuristics returned an optimal solution in a very short time.

1. Wstęp

W wielu systemach uwarunkowanych czasowo, takich jak system sterowania samolotem, elektrownią itp. występują zadania (czynności), które z natury mają charakter cykliczny: muszą być one powtarzane z pewną częstotliwością albo inaczej co pewien okres czasu, będący odwrotnością częstotliwości. Niektóre z tych zadań są krytyczne ze względu na bezpieczeństwo i jeśli zostaną zakończone zbyt późno, to może to doprowadzić do poważnej awarii, a nawet utraty życia. Takie zadania nazywane są zadaniami silnie uwarunkowanymi czasowo i muszą one być odpowiednio szeregowane.

Jednym z możliwych podejść do szeregowania zadań silnie uwarunkowanych czasowo jest spojrzenie na problem szeregowania jak na problem minimalizacji maksymalnego obciąże-

nia cyklicznymi zadaniami. Podejście to zostało zaproponowane przez Schweitzera, Drora i Trudeau [9]. Czas procesora jest dzielony na spójne przedziały o jednakowym rozmiarze, zwane ramami (ang. *frames*). Załóżmy, że każda rama ma długość τ jednostek czasowych. Jeśli zadanie T_i ma okres π_i ($\pi_i \geq \tau$), to należy je wykonywać co p_i ram, gdzie $p_i = \lfloor \pi_i/\tau \rfloor$. Niech f_i oznacza fazę zadania T_i, tzn. T_i jest wykonywane w ramach o numerach $f_i + kp_i$, gdzie $k \in \{0, 1, 2, ...\}$. W tej sytuacji szeregowanie zadań można przedstawić jako problem znalezienia takich wartości f_i dla wszystkich zadań T_i, że łączny czas wykonania zadań przypisanych do każdej ramy jest nie większy niż τ. Aby uprościć problem, można przyjąć spotykane w literaturze założenie [2,4-9], że okresy tworzą postęp binarny, czyli postęp geometryczny o ilorazie 2. W artykule przedstawione są wyniki badań dotyczących zastosowania metody minimalizacji obciążeń cyklicznych do szeregowania zadań silnie uwarunkowanych czasowo o okresach tworzących postęp binarny. Pokazano, że problem jest silnie NP-trudny i w związku z tym trudno mieć nadzieję na znalezienie wielomianowego czy nawet pseudo-wielomianowego algorytmu. W tej sytuacji zaproponowano prostą heurystykę zachłanną, która - jak wynika z przeprowadzonych eksperymentów obliczeniowych - zachowuje się lepiej od innych znanych heurystyk zachłannych.

2. Problem cyklicznego obciążenia

Problem cyklicznego obciążenia może być sformułowany w następujący sposób:

Dane: skończony zbiór zadań $I = \{T_1, T_2, .., T_n\}$. Każde zadanie T_i z tego zbioru scharakteryzowane jest poprzez okres p_i oraz czas trwania d_i (ang. *duration*) (p_i, $d_i \geq 0$). Zadanie T_i jest wykonywane co p_i ram, zaś każde wykonanie tego zadania trwa dokładnie d_i jednostek czasowych (np. sekund). Ponadto zakładamy, że zadania mają tzw. okresy binarne, czyli dowolny okres $p_i = 2^j$ dla pewnego naturalnego j (dla j = 0, okres p_i wynosi 1, czyli zadanie o takim okresie musi być wykonane w każdej ramie). Okresy binarne pozwalają na rozpatrywanie zadań w horyzoncie czasowym $P = \max\{p_1, p_2, ..., p_n\}$.

Problem optymalizacyjny. Ponumerujmy ramy liczbami 0, 1, 2, ..., P-1. Niech liczba całkowita f_i oznacza fazę zadania T_i ($0 \leq f_i \leq p_i - 1$). Jeżeli zadanie T_i posiada fazę f_i, to musi być ono wykonane w każdej ramie o numerze $f_i + k \cdot p_i$ gdzie k = 0, 1, 2, ..., $P/p_i - 1$. Niech y_j oznacza sumę czasów trwania zadań wykonywanych w ramie o numerze j, czyli y_j równe jest sumie tych d_j, dla których $f_i + k \cdot p_i = j$ dla $k \in \{0, 1, ..., P/p_i - 1\}$. Należy znaleźć dla każdego

zadania T_i taką fazę f_i, aby wartość z oznaczająca maksymalne obciążenie ramy była minimalna ($z = \max\{y_0, y_1, y_2, ..., y_{P-1}\}$).

Wersja decyzyjna powyższego problemu zakłada, iż dane są wartości p_i, d_i oraz pewna liczba całkowita τ ($\tau>0$). Pytanie, czy istnieje funkcja f_i taka, że maksymalne obciążenie ramy $z \leq \tau$. W celu określenia złożoności obliczeniowej powyższego problemu wystarczy rozpatrywać jego wersję decyzyjną.

Twierdzenie 1. Wersja decyzyjna problemu cyklicznego obciążenia jest silnie NP-trudna.

Dowód. Ponieważ zadania są cykliczne, pojawienie się zadania o okresie 2^j implikuje konieczność rozważenia 2^j ram. Zatem nie jest jasne, czy ten problem należy do klasy NP. Udowodnione zostanie więc tylko, że problem jest silnie NP-trudny. W tym celu pokazana zostanie transformacja z problemu trójpodziału [1, 3]. Instancja problemu trójpodziału jest zdefiniowana poprzez liczbę naturalną B oraz zbiór A zawierający $3m$ elementów. Niech każdy element $a \in A$ posiada rozmiar s(a) (rozmiar elementu jest liczbą naturalną) spełniający warunek: $B/4 < s(a) < B/2$ oraz niech suma wszystkich elementów należących do A wynosi $m \cdot B$. Pytanie brzmi, czy zbiór A może być podzielony na m rozłącznych podzbiorów $A_1, A_2, ..., A_m$ takich, że suma rozmiarów elementów należących do każdego z tych podzbiorów wynosi dokładnie B. Powyższe sformułowanie implikuje, iż każdy podzbiór musi zawierać dokładnie 3 elementy. W przypadku gdy $m=1$, problem trójpodziału jest trywialny. Załóżmy zatem, iż $2^{i-1} < m \leq 2^i$ oraz niech $m'=2^i$. Instancja problemu cyklicznego obciążenia odpowiadająca danej instancji problemu trójpodziału może być zbudowana w sposób następujący:

- Każdemu elementowi $a \in A$ przypisujemy zadanie o okresie m' i czasie trwania s(a).
- Ponadto wprowadzamy $m - m'$ zadań o okresie m' i czasie trwania równym B. Zadania te redukują liczbę ram reprezentujących podzbiory A_i z m' do m.
- Wprowadzamy też m' zadań o okresie równym 1 oraz jednostkowym czasie wykonania .
- Niech $\tau = B+1$.

Pytanie, czy istnieje taka funkcja f(i), że $z \leq \tau$.

Zauważmy, iż całkowity czas trwania wszystkich zadań wynosi $m' \cdot 1 + m \cdot B + (m'-m) \cdot B = m'(B+1)$. Zatem średni czas trwania zadań przypisanych do pojedynczej ramy wynosi dokładnie $B+1 = \tau$. Jeśli więc odpowiedź na powyższe pytanie brzmi TAK, to każda rama będzie zawierać zadania, których całkowity czas trwania wynosi B + 1. W takim przypadku

- każda rama zawiera dokładnie jedno zadanie o okresie 1 i jednostkowym czasie trwania;
- $m' - m$ ram zawiera duże zadania o czasie trwania równym B.

Zatem, zbiór A może być rozdzielony pomiędzy m ram (rozłącznych podzbiorów), a każda rama (podzbiór) zawiera zadania o całkowitym rozmiarze równym B. W takim razie odpowiedź na problem trójpodziału także brzmi TAK.

Teraz załóżmy przypadek odwrotny, tj. odpowiedź na problem trójpodziału brzmi TAK. W takim razie można dokonać następującej alokacji:

- zadania o okresie 1 i jednostkowym czasie trwania przypisujemy do każdej ramy;
- zadania o czasie trwania B przypisujemy do $m' - m$ losowo wybranych ram;
- zadania odpowiadające podzbiorom $A_1, A_2, ..., A_m$ przypisujemy do pozostałych m ram.

Uwzględniając powyższą alokację, odpowiedź dla wersji decyzyjnej problemu cyklicznego obciążenia również brzmi TAK.

Przedstawiona transformacja ma złożoność wielomianową, bowiem $m' \leq 2m$.

EQD

3. Heurystyki zachłanne

W związku z tym, iż problem cyklicznego obciążenia jest silnie NP-trudny, sensowne wydaje się opracowanie heurystyki, która dawałaby rozsądne rozwiązania, nie leżące zbyt daleko od optimum. Heurystyka przedstawiona w tym punkcie (algorytm CABT) zwraca szczególną uwagę na alokację zadań o największych czasach trwania. Aby zrozumieć ideę działania tejże heurystyki, rozpatrzmy prosty przykład. Niech dany będzie zbiór $I = \{T_1, T_2, T_3, T_4\}$. Zadania z tego zbioru zdefiniowane są następująco: $d_1=50$, $p_1=2$; $d_2=48$, $p_2=2$; $d_3=90$, $p_3=4$; $d_4=80$, $p_4=4$. W celach porównawczych wykorzystano dwie z ośmiu heurystyk Drora [9]:

- **Heurystyka NDP** (ang. *non-decreasing period*). Posortuj zadania według niemalejącej wartości ich okresu p_i (w przypadku jednakowych wartości p_i sortuj według nierosnącej wartości czasów trwania d_i). Kolejno przydzielaj posortowane zadania do ram, wybierając za każdym razem najmniej obciążoną ramę.
- **Heurystyka NID** (ang. *non-increasing duration*). Posortuj zadania według nierosnącej wartości ich czasów trwania d_i (w przypadku jednakowych wartości d_i sortuj według niemalejącej wartości okresów p_i). Kolejno przydzielaj posortowane zadania do ram, wybierając za każdym razem najmniej obciążoną ramę.

Niestety zarówno strategia NDP, jak i NID dają rozwiązania, które są dalekie od optimum (wartość z wynosi 138 – patrz rys. 1).

Ramy dla NDP			
0	1	2	3
50		50	
	48		48
	90		
			80

Ramy dla NID			
0	1	2	3
90			
	80		
	50		50
48		48	

Ramy dla CABT			
0	1	2	3
90			
		80	
	50		50
	48		48

Rys. 1. Przyporządkowanie zadań do ram dla strategii NDP, NID oraz CABT
Fig. 1. Assignment of tasks to frames for NDP, NID and CABT strategies

Strategia NDP faworyzuje zadania o krótkich okresach, nawet jeśli posiadają one niewielkie czasy wykonania. To powoduje, iż w momencie, gdy należy przydzielić większe zadania o dłuższych okresach, to nie ma już wtedy wolnych ram i zadania te muszą współdzielić ramy z zadaniami rozlokowanymi wcześniej. Prowadzi to do niezadowalających rozwiązań. Wydawałoby się, iż strategia NID, która rozpatruje najpierw duże zadania, powinna dawać lepsze wyniki. Niestety przydziela ona duże zadania o długich okresach do pierwszych wolnych, następujących po sobie, ram. Wówczas zadania o krótszych okresach muszą zostać przydzielone do ram, w których znajdują się już zadania o okresach dłuższych. Zatem także i ta strategia prowadzi do rozwiązań dalekich od optimum. Rozsądne wydawałoby się zatem alokowanie najpierw dużych zadań, ale należałoby unikać przydzielania ich do ram następujących po sobie (należy je w miarę możliwości przydzielać do nie zajętych ram, które są maksymalnie od siebie oddalone). Idea polega na przeglądaniu ram dla zadania o okresie $p(i) \geq 4$ w specyficznej kolejności:

$0,$

$\frac{1}{2}p_i,$

$\frac{1}{4}p_i, \quad \frac{3}{4}p_i,$

$\frac{1}{8}p_i, \quad \frac{3}{8}p_i, \quad \frac{5}{8}p_i, \quad \frac{7}{8}p_i,$

$\ldots,$

$\frac{1}{2^j}p_i, \quad \frac{3}{2^j}p_i, \quad \frac{5}{2^j}p_i, \quad \ldots, \quad \frac{2k+1}{2^j}p_i, \quad \ldots, \quad \frac{2^j-1}{2^j}p_i,$

$\ldots,$

$2, \quad 6, \quad 10, \quad 14, \quad \ldots, \quad p_i - 2,$

1

Ramy, które nie będą sprawdzane podczas takiego przeszukiwania, posiadają numery: 3, 5, 7, 9 itd. Jeżeli proces przeszukiwania zakończy się na ramie o numerze 1, oznacza to, iż tylko ramy o numerach 3, 5, 7, ... mogą być wolne. W tym momencie należy kontynuować alokację

zadań do ram używając zwykłej reguły wyboru najmniej obciążonej ramy. Powyższą sekwencję przeszukiwania można opisać za pomocą następujących wzorów rekurencyjnych:

$$
\begin{aligned}
n_0 &= 0 \\
n_1 &= p_i / 2 \\
n_{2j} &= n_j - \frac{p_i}{2 \cdot Power2(2j)}, \quad dla \quad j \geq 1 \\
n_{2j+1} &= n_j + \frac{p_i}{2 \cdot Power2(2j)}, \quad dla \quad j \geq 1
\end{aligned}
\tag{3.1}
$$

Taka strategia przeszukiwania została zaadaptowana w algorytmie CABT przedstawionym poniżej (rozwiązanie znalezione przez ten algorytm zostało zaprezentowane na rys. 1 – w tym przypadku $z = 98$). Drugą istotną cechą algorytmu CABT jest branie pod uwagę całego horyzontu czasowego. W odróżnieniu od algorytmów zaproponowanych w [9], algorytm CABT przy wyborze najmniej obciążonej ramy dla danego zadania T_i o okresie p_i rozpatruje nie tylko ramy o numerach 0, 1, 2, ..., p_i-1, ale także ramy p_i, p_i+1, ..., P-1 (patrz krok 5 algorytmu CABT). Aby pokazać, jak istotne jest rozpatrywanie wszystkich ram w horyzoncie czasowym P, rozpatrzmy prosty przykład. Niech dane będą 4 ramy o numerach 0, 1, 2 oraz 3, których obciążenie wynosi odpowiednio: 60, 70, 90 i 70. Spróbujmy uszeregować zadanie o okresie p_i=2 i czasie trwania d_i=20. W przypadku rozpatrywania ram tylko z okresu zadania (czyli ram o numerach 0 i 1) zadanie takie należałoby umieścić w ramie o numerze 0 (jest ona mniej obciążona niż rama 1). Otrzymana wartość z wynosi wówczas 110 (jest to obciążenie ramy 2). Z globalnego punktu widzenia (rozpatrując ramy o numerach od 0 do 3) znacznie korzystniej jest przydzielić takie zadanie do ramy 1 – wówczas otrzymana wartość z wynosi 90.

Algorytm CABT (ang. *Carefully Assigned Big Tasks*).

1. Niech m = |I|. Przygotuj P = max{p_1, p_2, ..., p_m} pustych ram ponumerowanych liczbami 0, 1, ..., P - 1.
2. Przyporządkuj do każdej z ram wszystkie zadania o okresie p_i = 1 (jeżeli p_i = 1 to f_i = 0). Jeśli dana rama zawiera tylko i wyłącznie zadania o okresie 1, to ramę taką nazywamy quasi-pustą.
3. Posortuj wszystkie zadania o okresach $p_i \geq 2$ według nierosnących wartości ich czasów trwania d_i i umieść je na liście L. W przypadku zadań o tych samych wartościach d_i, uporządkuj je według niemalejących wartości ich okresów p_i.
4. Powtórz następujące kroki:

(a) Niech T_i oznacza pierwsze zadanie na liście L.

(b) Znajdź najmniejsze takie j, że rama n_j jest pusta (patrz wzór 3.1).

(c) Przyjmij, że $f_i = n_j$, usuń zadanie T_i z listy L oraz uaktualnij wszystkie ramy o numerach $f(i) + k \cdot p_i$ (gdzie $k = 0, 1, ..., P/p_i - 1$) alokując w nich zadanie T_i.

(d) Powtarzaj kroki od (a) do (c), jak długo $n_j \neq 1$ i lista L nie jest pusta.

5. Przydziel pozostałe zadania z listy L do ram wybierając za każdym razem najmniej obciążoną ramę. Dokładniej mówiąc, niech $\sigma(t)$ oznacza całkowity rozmiar zadań przydzielonych do ramy o numerze t ($t \in \{0, 1, ..., P-1\}$). Poczynając od pierwszego zadania na liście L, dla każdego z kolejnych zadań T_i należy znaleźć takie $\varphi \in \{0, 1, ..., p_i - 1\}$, że $\sigma(\varphi + k \cdot p_i)$ jest minimalne dla wszystkich wartości k ze zbioru $\{0, 1, ..., P/p_i - 1\}$. Następnie przyjmij $f_i = \varphi$ i uaktualnij wszystkie ramy o numerach $f_i + k \cdot p_i$.

4. Eksperymenty obliczeniowe

W celu przetestowania zachowania się algorytmu CABT został on porównany z algorytmami NDP oraz NID na losowo generowanych instancjach testowych. Zostały przeprowadzone dwa rodzaje testów:

- Pierwszy z nich bazował na teście opisanym w [9]. W naszym przypadku instancja testowa składała się z 50 zadań. Czas trwania d_i pojedynczego zadania był liczbą całkowitą losowaną z przedziału [10, X], gdzie $X = 5k+10$, $k \in N$ (przeprowadzono testy dla wartości $k = 0, 1, ..., 50$). Okresy $p_i = 2^j$ wszystkich zadań były także losowane dla $j \in \{2, 3, 4, 5\}$. Ponadto, w celu uzyskania dokładniejszych wyników zdecydowaliśmy się generować po 100 instancji dla każdej z przyjętych wartości k, a następnie obliczyć wartość średnią (w [9] każdy eksperyment składał się tylko z 25 losowych instancji).
- Drugi test próbował uwzględnić rzeczywistą sytuację, gdy zadania o krótszych okresach są mniejsze (czas trwania jest krótszy). Takie zadania są zwykle krytyczne dla pracy całego systemu silnie uwarunkowanego czasowo i muszą być wykonane często i szybko. W tym teście czas trwania d_i pojedynczego zadania był liczbą całkowitą losowaną z przedziału $(\alpha k_i, k_i]$, gdzie $\alpha \in (0,5; 1]$ i $k_i = 10 \cdot p_i$. Podobnie jak w poprzednim teście pojedyncza instancja składała się z 50 zadań o okresach $p_i = 2^j$ dla wartości j losowanych ze zbioru $\{2, 3, 4, 5\}$. Także tutaj dla każdej wartości α zostało wygenerowanych 100 instancji a następnie

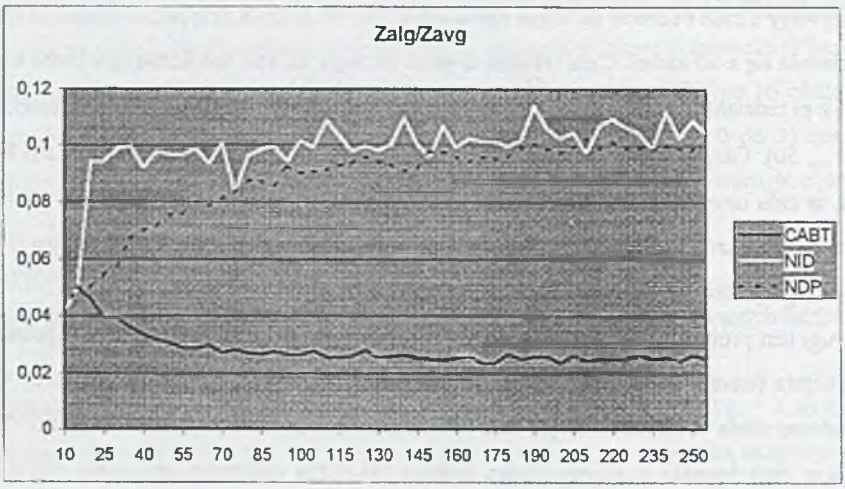
na podstawie tak otrzymanych wyników obliczono odpowiednie wartości średnie: z_{CABT}, z_{NDP} oraz z_{NID}.

Wyniki uzyskane w trakcie testów prezentują wykresy 1 i 2. Dla algorytmu przybliżonego *alg* można zdefiniować błąd przybliżenia e_{alg} zdefiniowany jako $e_{alg} = (z_{alg} - z_{opt})/z_{opt}$. Niestety, znalezienie wartości z_{opt} jest bardzo czasochłonne (problem jest silnie NP-trudny). Zatem uwzględniając fakt, że: $z_{opt} \geq g_{av}$ można estymować błąd przybliżenia e_{alg} za pomocą błędu E_{alg} zdefiniowanego jako:

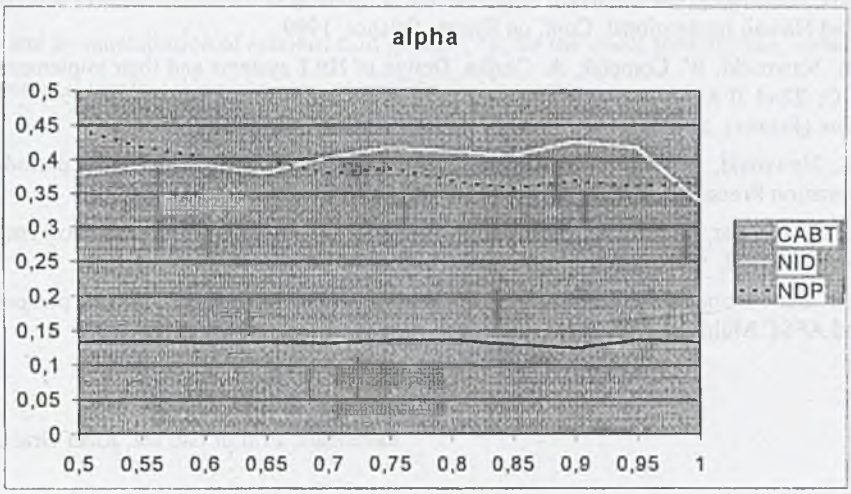
$$E_{alg} = (z_{alg} - g_{av}) / g_{av}$$

gdzie g_{av} jest to średnie obciążenie pojedynczej ramy zadaniami (suma wszystkich instancji zadań w uszeregowaniu podzielona przez liczbę ram). Oczywiście, prawdą jest, że $E_{alg} \geq e_{alg}$.

Obserwując wykresy 1 i 2 łatwo zauważyć, iż algorytm CABT jest znacznie lepszy od algorytmów NDP oraz NID (błąd dla algorytmu CABT nie przekracza 5% w pierwszym i 14% w drugim teście). W przypadku algorytmów NDP oraz NID błędy te są znacznie większe i wynoszą odpowiednio: około 10% dla pierwszego i 40% dla drugiego testu. Ponadto zauważmy, że algorytmy NID oraz NDP zachowują się o wiele lepiej (dają lepsze wyniki) dla instancji testowych, w których zadania mają zbliżone rozmiary. Algorytm CABT z kolei działa sprawniej, gdy rozmiary zadań w takiej instancji są zróżnicowane.



Rys 2. Zależność błędu względnego E od ilorazu max{ T_i}/min{T_i}
Fig. 2. Relative error E vs. the ratio max{T_i}/min{T_i}



Rys. 3. Zależność błędu względnego E od współczynnika α
Fig. 3. Relative error E vs. factor α

5. Zakończenie

Wobec silniej NP-trudności rozpatrywanego problemu w artykule zaproponowano prostą heurystykę zachłanną, która starannie rozmieszcza zadania o długim czasie trwania i rozpatrujezawsze cały horyzont czasowy, a nie tylko pierwszy okres zadania. Z przeprowadzonych eksperymentów wynika, że jest to ciekawa heurystyka, dająca często lepsze wyniki niż znane do tej pory heurystyki zachłanne dla tego problemu. Co więcej, okazało się, że zastosowana do praktycznego problemu szeregowania zadań w samolocie F-16 [10] dała optymalne rozwiązanie w bardzo krótkim czasie.

LITERATURA

1. J. Błażewicz, K.H. Ecker, G. Schmidt, J. Węglarz, Scheduling Computer and Manufacturing Processes, Springer-Verlag, Berlin, 1996.
2. Y. Cai, M.C. Kong, Nonpreemptive scheduling of periodic tasks in uni- and multiprocessor systems, Algorithmica, vol. 15, 572-599, 1996.
3. M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman and Company, San Francisco, 1979.
4. M.J. Gonzalez Jr., W. Soh, Periodic job scheduling in a distributed processor system, IEEE Trans. Aerospace and Electronic Systems, vol. AES-12, no. 5 (1976), 530-535.
5. R.M. Kieckhafer, C.J. Walter, A.M. Finn, P.M. Thambidurai, The MAFT architecture for distributed fault-tolerance, IEEE Trans. Computers, vol. C-37 (1988), no. 4, 398-405.

6. R.M. Kieckhafer, Fault-tolerant real-time task scheduling in the MAFT distributed system, 22nd Hawaii International Conf. on System Science, 1989.
7. J.R. Nawrocki, W. Complak, A. Czajka, Design of HRT systems and their implementation in C, 22nd IFAC/IFIP Workshop on Real Time Programming, September 15-17, 1997, Lyon (France), 57-62.
8. J.R. Nawrocki, A. Czajka, W. Complak, Scheduling cyclic tasks with binary periods, Information Processing Letter, vol. 65 (1998) 173-178.
9. P.J. Schweitzer, M. Dror, P. Trudeau, The periodic loading problem: formulation and heuristics, INFOR, vol. 26, no. 1 (1988), 40-62.
10. D.E. Sundstrom, W.B. Anderson, S.A. Alford Jr., F-16 multiplex: a system perspective, 2nd AFSC Multiplex Data Bus Conference, 10-12 October 1978.

Recenzent: Prof.dr hab.inż. Józef Grabowski

Abstract

In many real time systems such as air craft controller, power plan controller etc. there are tasks (activities) which are periodic by nature: they have to be repeated with a given frequency or with a given period which is the reciprocal of a frequency. Some of these tasks are mission critical: they have to be completed on time or otherwise some expensive equipment or even human live will be in danger. Such tasks are called hard real-time tasks and they have to be scheduled accordingly.

One of possible approaches to scheduling hard real-time tasks is minimisation of periodic loading. That approach has been proposed by Schweitzer, Dror and Trudeau [9]. Processor time is split into a number of equal size intervals called frames. Assume each frame is τ time units wide. If a task T_i has a period π_i ($\pi_i \geq \tau$), it has to be performed every p_i frames, where $p_i = \lfloor \pi_i/\tau \rfloor$. Let f_i denote a phase of a task T_i, i.e. T_i is performed in every time frame of a number $f_i + kp_i$, where k=0, 1, 2, ... Now one can schedule a set of tasks by finding for each task T_i such a phase f_i that for every time frame, total execution time of all the tasks assigned to that frame is not greater than τ. To make the problem simpler one can assume that periods of tasks form a geometrical progression of a quotient 2 (such an assumption has been taken e.g. in [2, 4-9]).

In the paper the problem of scheduling hard real-time tasks by minimisation of periodic loading is investigated. It is shown that the problem is strongly NP-hard. As a consequence, practically there is no chance of finding polynomial or pseudo-polynomial algorithm for the problem. Thus, a greedy heuristic algorithm (CABT) is proposed, which is simple enough and

effective. It improves the non-decreasing duration strategy [9] by careful assignment of big tasks and by minimisation of maximal load globally, i.e. for the whole time horizon, instead of minimising it locally taking into account only the first p_i frames.