ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1992
Seria: AUTOMATYKA z. 109 Nr kol. 1175

Jacek Błażewicz, Maciej Drozdowski
Jan Węglarz

Politechnika Poznańska

SZEREGOWANIE ZADAŃ WIELOPROCESOROWYCH PRZED LINIAMI KRYTYCZNYMI

SCHEDULING MULTIPROCESSOR TASKS TO MINIMIZE MAXIMUM LATENESS

СОСТАВЛЕНИЕ РАСПИСАНИЙ МНОГОПРОЦЕССОРНЫХ ЗАДАЧ ПЕРЕД КРИТИЧЕСКИМИ СРОКАМИ

Streszczenie: W pracy przedstawiono problem szeregowania w systemie czasu rzeczywistego zadań wieloprocesorowych, to znaczy żądających do swego wykonania wielu procesorów jednocześnie. Rozpatrzono minimalizację maksymalnego opóźnienia dla zadań podzielnych i niezależnych.

Summary: In this paper we analyse the problem of scheduling multiprocessor tasks, i.e. tasks requiring more than one processor at a time, in the real time environment. The minimization of the maximum lateness for nonpreemptive and independent tasks is considered.

Резюме: В работе представлен анализ проблемы составления расписаний многопроцессорных задач в системе реального времени, т.е. задач, требующих для выполнения одновременно больше чем одного процессора. В работе рассмотрена минимизация времени опоздания для делимых и независимых задач.

1. WSTĘP

W klasycznej teorii szeregowania zakłada się, że każde zadanie jest wykonywane na jednym procesorze w danej chwili czasu. Wraz z szybkim rozwojem wieloprocesorowych systemów komputerowych oraz elastycznych systemów produkcyjnych założenie to nie jest już tak oczywiste. W pewnych sytuacjach zadanie wieloprocesorowe może żądać do swego wykonania więcej niż jednego procesora jednocześnie. Z takim problemem mamy do czynienia na przykład w samosprawdzających się systemach wieloprocesorowych, w których jeden procesor testuje inne [7, 11]. Takie podejście może również stać się szczególnie ważne wraz z rozwojem wieloprocesorowych systemów komputerowych i związanych z nimi systemów operacyjnych [9, 14].

Powyższe zagadnienia były już rozważane z punktu widzenia teorii szeregowania zadań. W [2] rozważano problem szeregowania zadań dla

kryterium długości uązeregowania i procesorów równoległych. Dla procesorów jednorodnych problem minimalizacji długości uszeregowania był analizowany w [3], a dla procesorów dedykowanych w [5, 12 ,1]. Bardziej ogólną wersję problemu, w której dopuszcza się, aby czas wykonania zadania mógł zależeć od liczby przydzielonych procesorów, rozważano w [8]. W pracy [6] rozważano szeregowanie zadań podzielnych w systemie identycznych procesorów, których połączenia komunikacyjne tworzą wielowymiarową kostkę. Praktyczną konsekwencją dla szeregowania zadań w komputerze o takiej architekturze jest fakt, że zadania będą żądały liczby procesorów, która jest potęgą dwójki. Wyniki tej pracy zostały wykorzystane w [13] do analizy problemu szeregowania zadań w komputerze o takiej architekturze, w środowisku czasu rzeczywistego (tzn. znajdowane jest, jeżeli istnieje, uszeregowanie przed liniami krytycznymi). W niniejszej publikacji rozważany jest problem, w którym zadania żądają dowolnej liczby procesorów jednocześnie, kryterium jest maksymalne opóźnienie.

Problem szeregowania zadań wieloprocesorowych przed liniami krytycznymi definiujemy następująco. Dany jest zbiór zadań $\mathcal{T}=\{T^1,T^k\}$, gdzie $|T^i|=n_i$, $(i=1,k)$ i $n_1+n_k=n$. Każde zadanie $T_i^j\in T^j$ wymaga do swego wykonania dokładnie j procesorów jednocześnie przez okres czasu t_i^j. Zadanie T_i^j jest obecne w systemie w przedziale czasu $[r_i^j,d_i^j]$. Wszystkie zadania są niezależne. Uszeregowanie jest podzielne, to znaczy, każde z zadań może zostać przerwane, a potem wznowione bez dodatkowych kosztów. Zadania szeregujemy na zbiorze $\mathcal{P}=\{P_1, P_2,\ldots P_m\}$ (m nie musi być wielokrotnością k jak w [13]) identycznych procesorów. Kryterium optymalności uszeregowania jest maksymalne opóźnienie $L_{max}=\max_{1\le j\le n_i}\{0, C_j^i-d_j^i\}$, gdzie C_j^i jest momentem zakończenia zadania T_i^j.

Problem rozwiązano z pomocą dwóch różnych algorytmów. Pierwszy z nich został oparty na generacji wszystkich dopuszczalnych zbiorów zadań, które można wykonywać jednocześnie na danym zbiorze procesorów. Drugi z algorytmów jest algorytmem przybliżonym. Wykorzystuje on poszukiwanie w kierunku największego spadku wartości L_{max} oraz metodę tabu. Omówione zostały wyniki eksperymentalnego porównania obu metod.

2. ROZWIĄZANIE PROBLEMU

2.1. METODA DOPUSZCZALNYCH ZBIORÓW ZADAŃ

Uporządkujmy wszystkie zdarzenia w systemie zadań (momenty przybycia i linie krytyczne) w rosnącej kolejności $e_0<e_1<\ldots<e_r$. $e_0=0$ jest momentem pojawienia się pierwszego zadania, e_r jest ostatnią

linią krytyczną, $\tau_\ell = e_\ell - e_{\ell-1}$ jest długością odpowiedniego przedziału czasu między dwoma kolejnymi zdarzeniami.

Przez dopuszczalny zbiór zadań będziemy rozumieć zbiór zadań, które mogą być wykonywane jednoczesnie. Niech M_ℓ będzie liczbą różnych dopuszczalnych zbiorów zadań w ℓ-tym przedziale czasu, o długości wykonywania $x_{\ell i}$ dla i-tego dopuszczalnego zbioru. Niech $Q^1_{\ell j}$, $Q^k_{\ell j}$ będą zbiorami indeksów dopuszczalnych zbiorów zadań w przedziale ℓ zawierających, odpowiednio, zadania T^1_j, T^k_j. Teraz nasz problem może zostać sformułowany jako zadanie programowania liniowego (LP):

$$\min L_{max} \tag{1}$$

przy ograniczeniach

$$\sum_{i=1}^{M_\ell} x_{\ell i} \le \tau_\ell = \begin{cases} e_1 + L_{max} & \ell = 1 \\ e_\ell - e_{\ell-1} & \ell = 2,\ldots,r \end{cases} \tag{2}$$

$$\sum_{\ell=1}^{r} \sum_{i \in Q^1_{\ell j}} x_{\ell i} = t^1_j \qquad j = 1 \ldots n_1 \tag{3}$$

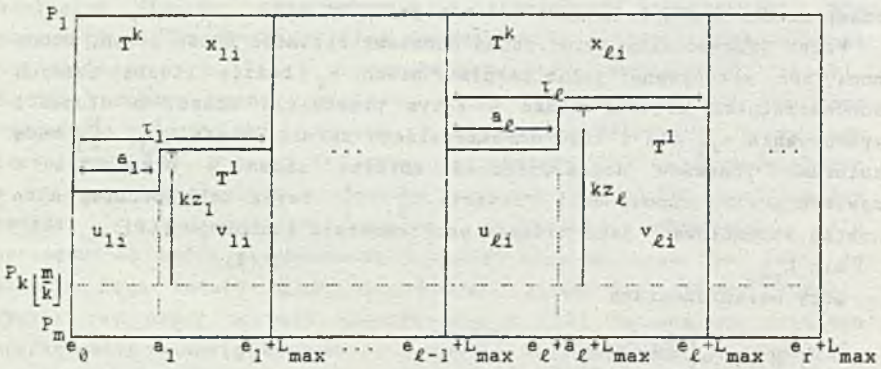
$$\sum_{\ell=1}^{r} \sum_{i \in Q^k_{\ell j}} x_{\ell i} = t^k_j \qquad j = 1 \ldots n_k \tag{4}$$

To podejście można rozszerzyć także na przypadek, w którym $T = \{T^1, T^2, \ldots T^k\}$. Główną wadą tej metody jest używanie dużej liczby dopuszczalnych zbiorów zadań. Dla m ustalonego liczba dopuszczalnych zbiorów zadań jest ograniczona wielomianowo i jest $O(n^m)$. Ponieważ problem programowania liniowego można rozwiązać w czasie ograniczonym od góry przez wielomian zależny od liczby zmiennych i ograniczeń, zatem powyższy problem szeregowania można także rozwiązać w czasie wielomianowym. W praktyce jednak czas obliczeń może być duży ze względu na dużą liczbę dopuszczalnych zbiorów zadań.

2.2. ALGORYTM PRZYBLIŻONY

Wady metody dokładnej doprowadziły do zaproponowania szybszego algorytmu przybliżonego.

Z [2] wiadomo, że każde dopuszczalne uszeregowanie w przedziale $[e_{\ell-1}, e_\ell]$ można przekształcić w tzw. A-uszeregowanie. A-uszeregowanie, jest uszeregowaniem, w którym zadania typu T^k są wykonywane na procesorach $P_1, \ldots P_{k\lfloor m/k \rfloor - kz_\ell}$ $z_\ell \in Z^+$ w całym przedziale czasu $[e_{\ell-1}, e_\ell]$ oraz przez $P_{k\lfloor m/k \rfloor - kz_\ell + 1}, \ldots P_{k\lfloor m/k \rfloor - kz_\ell + k}$ w pewnym przedziale $[e_{\ell-1}, e_{\ell-1} + a_\ell]$ $(a_\ell \le \tau_\ell)$. Uszeregowanie wygląda wtedy tak jak na rysunku 1.



Rys.1.A-uszeregowanie
Fig.1.An A-schedule

Nasz problem szeregowania można uważać za problem poszukiwania najlepszego rozwiązania w dyskretnej przestrzeni $\bar{z}=z_1 \times z_2 \times \ldots \times z_r$. Będziemy poszukiwać takich wartości $z_1, z_2, \ldots, z_r$, dla których L_{max} jest minimalne.

Oznaczmy przez $x_{\ell i}$ sumaryczny czas wykonywania zadania T_i^k w całym przedziale ℓ na procesorach $P_1 \ldots P_{k\lfloor m/k\rfloor - kz_\ell}$ i na procesorach $P_{k\lfloor m/k\rfloor - kz_\ell+1} \ldots P_{k\lfloor m/k\rfloor - kz_\ell+k}$ w przedziale $[e_{\ell-1}, e_{\ell-1}+a_\ell]$. $u_{\ell i}$ - niech oznacza czas wykonania zadania T_i^1 w przedziale czasu $[e_{\ell-1}, e_{\ell-1}+a_\ell]$ na procesorach $P_{k\lfloor m/k\rfloor - kz_\ell+k} \ldots P_m$, a $v_{\ell i}$ - czas wykonywania T_i^1 w przedziale $[e_{\ell-1}+a_\ell, e_\ell]$ na $P_{k\lfloor m/k\rfloor - kz_\ell} \ldots P_m$. Nasz problem może zostać sformułowany następująco:

$$\min L_{max} \tag{5}$$

przy ograniczeniach

$$\sum_{i=1}^{n_k} x_{\ell i} \le a_\ell + (\lfloor m/k \rfloor - z_\ell)\tau_\ell \qquad \ell=1 \ldots r \tag{6}$$

$$\sum_{i=1}^{n_1} u_{\ell i} \le a_\ell ((z_\ell - 1)k + m - k\lfloor m/k \rfloor) \qquad \ell=1 \ldots r \tag{7}$$

$$\sum_{i=1}^{n_1} v_{\ell i} \le (\tau_\ell - a_\ell)(z_\ell k + m - k\lfloor m/k \rfloor) \qquad \ell=1 \ldots r \tag{8}$$

$$0 \le u_{\ell i} \le a_\ell \qquad \ell=1 \ldots r \tag{9}$$

$$0 \le v_{\ell i} \le \tau_\ell - a_\ell \qquad \ell=1 \ldots r \tag{10}$$

$$0 \le a_\ell, x_{\ell i} \le \tau_\ell = \begin{cases} e_1 + L_{max} & \ell = 1 \\ e_\ell - e_{\ell-1} & \ell = 2,\ldots,r \end{cases} \qquad (11)$$

$$z_\ell \in Z^+, \quad z_\ell \le \lfloor m/k \rfloor \qquad \ell = 1\ldots r \qquad (12)$$

$$\sum_{\ell=1}^{r} x_{\ell i} = t_i^k \qquad i = 1\ldots n_k \qquad (13)$$

$$\sum_{\ell=1}^{r} (u_{\ell i} + v_{\ell i}) = t_i^1 \qquad i = 1\ldots n_1 \qquad (14)$$

Ograniczenia (6) oznaczają, że suma czasów wykonywania części zadań T^k w przedziale $[e_\ell, e_{\ell-1}]$ nie może być większa niż suma czasów pracy wykonujących je procesorów. Ograniczenia (7) i (8) wymuszają to, że sumy T^1 w przedziałach $[e_{\ell-1}, e_{\ell-1}+a_\ell]$ i $[e_{\ell-1}+a_\ell, e_\ell]$ nie są większe niż czasy pracy odpowiednich procesorów. Ograniczenia (9), (10), (11) wyrażają to, że żadna z części pojedynczego zadania nie może być wykonywana dłużej niż odpowiadający jej przedział czasu. Ograniczenie (12) jest rezultatem sformułowania problemu. Równania (13) i (14) gwarantują wykonanie wszystkich zadań.

Powyższy problem jest zagadnieniem nieliniowego mieszanego programowania matematycznego. Jednak dla ustalonego z_ℓ $\ell = 1\ldots\lfloor m/k \rfloor$ problem redukuje się do programowania liniowego. To spostrzeżenie prowadzi do sformułowania $O(\lfloor m/k \rfloor^n)$ różnych zadań programowania liniowego. Ponieważ ta liczba w praktyce może być duża, posłużymy się dodatkowymi informacjami, by zredukować liczbę analizowanych przypadków. Poniższe twierdzenia upraszczają proces poszukiwania rozwiązania [4].

Twierdzenie 1. $L_{max}(\bar{z})$ jest funkcją unimodalną dla z_1=const.

Twierdzenie 2. $L_{max}(z_1)$ dla z_ℓ=const, $\ell = 1\ldots\lfloor m/k \rfloor$, ma co najwyżej dwa minima.

Powyższe dwa twierdzenia zapewniają, że liczba lokalnych minimów jest ograniczona i nie wzrasta wraz z rozmiarem problemu. Będziemy poszukiwać rozwiązania w przestrzeni $\bar{z}$ w następujący sposób. Wartość $L_{max}(\bar{z})$ będzie ulepszana tylko z z_ℓ zmieniającym się, podczas gdy $z_1, \ldots, z_{\ell-1}, z_{\ell+1}, \ldots z_r$ będą stałe. Po znalezieniu najlepszej wartości ze zmiennym z_ℓ, "uwalniamy" $z_{\ell+1}$ (pozostałe składniki $\bar{z}$ są stałe) i powtarzamy procedurę. To podejście może jednak spowodować zatrzymanie się procesu poszukiwań, gdy rozwiązanie lepsze od bieżącego różni się

w więcej niż jednej składowej wektora $\bar{z}$. Aby uniknąć takich "pułapek", zastosujemy metaheurystykę Tabu Search (TS) [10, 15]. Sformułowanie metaheurystyki TS jest następujące.

0. Poszukiwanie w kierunku najmniejszego spadku.
   nb:=0; {Znajduje: s - rozwiązanie bieżące, s^* - najlepsze dotychczas znalezione , s=s^* ; nb - licznik iteracji}
1. Wygeneruj zbiór dopuszczalnych rozwiązań sąsiadów bieżącego rozwiązania s; { zmieniając w górę i w dół wartości jednej składowej $\bar{z}$}
2. Wybierz najlepsze rozwiązanie s' ze zbioru utworzonego w 1.
3. If s'∈lista_TABU then go to 5.
4. If $A(L_{max}(s))>L_{max}(s')$ then zignoruj status TABU s' i go to 5
   else go to 7; {A(L) jest tzw. funkcją aspiracji}
5. s:=s';
   dodaj s' do listy TABU i jeżeli trzeba, usuń najstarszy element z listy TABU.
6. If $L_{max}(s')<L_{max}(s^*)$ then s^*:=s' and nb:=0; {jeżeli jest poprawa, wyzeruj licznik iteracji}
   else if odległość s' do s^* jest większa niż to jest pożądane {np. 2 w dowolnym składowej} then rozpocznij poszykiwania z s^*;
7. Zmień rozważaną składową $\bar{z}$; nb:=nb+1;
8. if nb≤NBmax then go to 1
9. stop {Zatrzymaj się, jeżeli NBmax+1 iteracji bez poprawy}

Potencjalną zaletą metody TS nad przeszukiwaniem wszystkich możliwych wartości $\bar{z}$ jest to, że nie każde możliwe rozwiązanie jest sprawdzane. Ponieważ $L_{max}(\bar{z})$ jest wypukłe, dlatego dla każdej składowej z_ℓ cały jej zakres wartości jest przemierzany co najwyżej dwa razy. Pierwszy raz - w poszukiwaniu minimum w kierunku największego spadku. Drugi raz - podążając za dnem doliny wartości L_{max} stosując metodę TS. W otoczeniu optymalnego rozwiązania metoda TS analizuje tylko skończoną liczbę końcowych kroków, by znaleźć całkowitoliczbowe optymalne $\bar{z}$. Tak więc redukuje to średnio liczbę sprawdzanych wartości wektora $\bar{z}$ z $O(\lfloor m/k \rfloor^n)$ do O(mr). Każde zadanie programowania liniowego jest rozwiązywalne w wielomianowym czasie, gdyż mamy $O(n^2)$ zmiennych i O(n) ograniczeń. Co więcej, podejście to daje algorytm heurystyczny o wielomianowej średniej złożoności, także gdy m nie jest ustalone.

3. WYNIKI EKSPERYMENTÓW OBLICZENIOWYCH

W tym rozdziale przedstawione zostaną wyniki eksperymentów obliczeniowych przeprowadzonych nad algorytmami przedstawionymi w rozdziałach 2.1 i 2.2.

Programy symulatorów zostały napisane w języku Turbo Pascal (wersja 5.0). Symulacje przeprowadzono pod nadzorem systemu operacyjnego MS-DOS 3.30 na komputerze IBM 386 (AT). Oba symulatory używały tego samego generatora danych i procedury rozwiązującej zadania programowania liniowego. W symulacjach zwrócono głównie uwagę na czas wykonania programów, obszar wykorzystywanej pamięci operacyjnej i jakość generowanych rozwiązań. Rozwiązano ponad 20.000 przykładowych instancji problemu. Osiągnięte wyniki przedstawiono na rysunkach 2-4. Skrótem DZZ (dopuszczalne zbiory zadań) oznaczono metodę pierwszą, skrótem TABU - drugą.

3.1. POROWNANIE CZASU WYKONANIA

Rezultaty przykładowych eksperymentów przedstawiono na rysunku 2. Można zauważyć, że czas wykonania algorytmu przybliżonego rośnie znacznie wolniej niż metody wykorzystującej dopuszczalne zbiory zadań. Dla 10 zadań metoda przybliżona dominuje. Stwierdzono ponadto, że czas wykonania dla obu metod silnie zależy od liczby zadań i przedziałów.

3.2. POROWNANIE WYKORZYSTANIA PAMIĘCI

Rezultaty przedstawiono na rysunku 3. Można stwierdzić, że metoda przybliżona wykorzystuje mniej pamięci, i przy ok 10 zadaniach jest lepsza od metody wykorzystującej dopuszczalne zbiory zadań. Rezultatem mniejszego zużycia pamięci jest to, że metoda przybliżona rozwiązuje większe instancje niż metoda dokładna przy takiej samej zajętości pamięci operacyjnej. Rysunek 4 przedstawia mapę instancji rozwiązywalnych przez obie metody.

3.3. POROWNANIE JAKOSCI ROZWIĄZAŃ

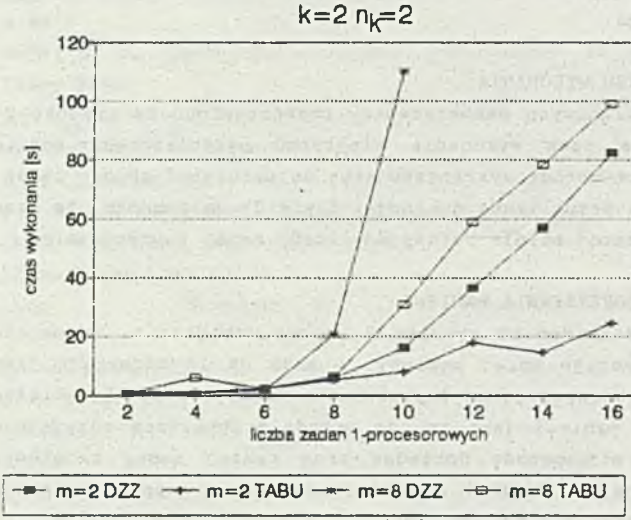
Analizie poddano losową próbkę 1046 instancji z liczbą procesorów wynoszącą 2...16 i liczbą zadań 2...16 typu T^1 i T^2. Jedynie dla 8 (0.76%) przykładów metoda przybliżona dała gorsze rozwiązanie niż metoda dokładna.

Wyniki badań symulacyjnych można podsumować stwierdzeniem, że metoda przybliżona wykorzystująca poszukiwanie w kierunku spadku L_{max} i metaheurystykę Tabu Search daje dobre wyniki w porównaniu z dokładną

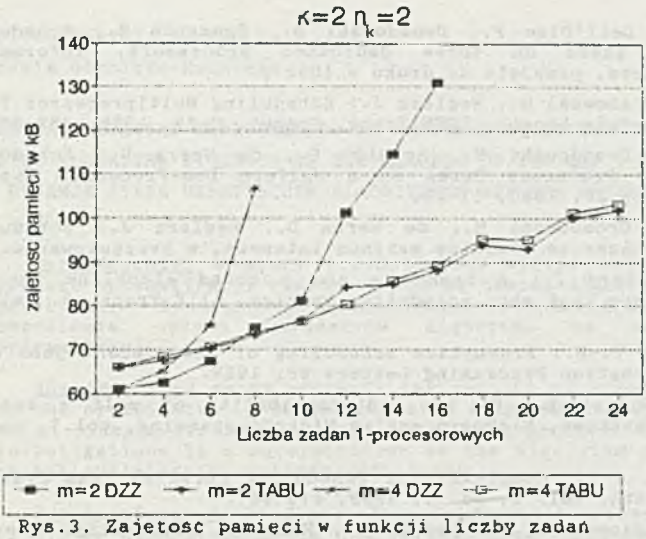
metodą wykorzystującą dopuszczalne zbiory zadań. Z punktu widzenia zajętości pamięci operacyjnej i czasu wykonania algorytm przybliżony jest lepszy dla większych instancji problemu.

4. ZAKOŃCZENIE

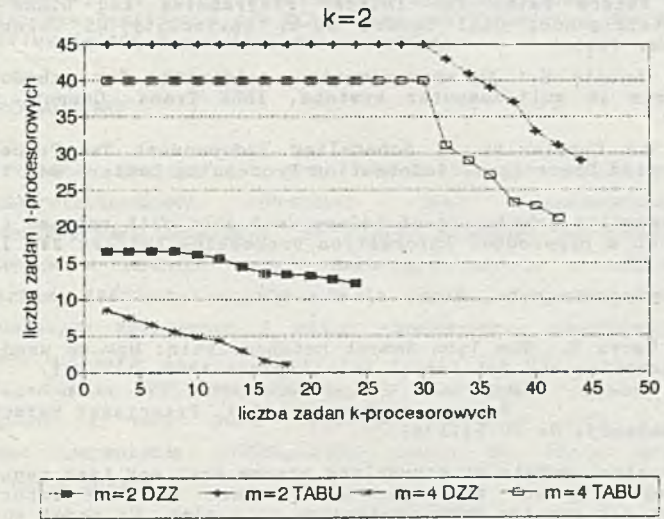
W pracy przedstawiono dwie metody rozwiązania problemu szeregowania zadań wieloprocesorowych w środowisku czasu rzeczywistego. Pierwsza metoda daje rozwiązanie dokładne, druga - przybliżone. Przeprowadzone eksperymenty pokazują, że algorytm przybliżony nie ustępuje dokładnemu nie tylko z punktu widzenia szybkości i zajętości pamięci, ale także pod względem jakości generowanych rozwiązań.



Rys.2. Czas wykonania w funkcji liczby zadań
Fig.2. Execution time vs number of tasks



Rys.3. Zajetosc pamięci w funkcji liczby zadań
Fig.3. Memory consumption vs number of tasks



Rys.4. Porownanie rozmiarow rozwiazywanych instancji
Fig.4. Comparison of sizes of solvable instances

LITERATURA

[1] Błażewicz J., Dell'Olmo P., Drozdowski M., Speranza M.: Scheduling multiprocessor tasks on three dedicated processors, Information Processing Letters, przyjęte do druku w 1992 r.

[2] Błażewicz J., Drabowski M., Węglarz J.: Scheduling Multiprocessor Tasks to Minimize Schedule Length, IEEE Trans. Comput. C-35, 1986, 389-393.

[3] Błażewicz J., Drozdowski M., Schmidt G., de Werra D.: Scheduling Independent Two Processor Tasks on a Uniform Duo-Processor System, Discr. Appl. Math 28, 1990, 11-20.

[4] Błażewicz J., Drozdowski M., de Werra D., Węglarz J.: Scheduling multiprocessor tasks to minimize maximum lateness, w przygotowaniu.

[5] Bozoki G., Richard J.: A branch - and - bound algorithm for the continuous-process task shop scheduling problem, AIIE Trans. 3, vol. 2, 1970, 246-252.

[6] Chen G.-T., Lai T.-H.: Preemptice scheduling of independent jobs on a hypercube, Information Processing Letters 28, 1988.

[7] Dal Cin M., Dilger E.: On the diagnosibility of self - testing multiprocessor systems, Microprocessing Microprogramming, vol.7, no. 3, 1981, 177-184.

[8] Du J., Leung J. Y-T.: Complexity of Scheduling Parallel Task Systems, Siam J. Disc. Math. Vol. 2, No. 4, 1989, 473-487.

[9] Gehringer E., Siewiorek D., Segall Z., Parallel processing. The Cm* experience, Digital Press, 1987.

[10] Glover F.: Future Paths for Integer Programming and Links to Artificial Intelligence, CAAI Report 85-8, University of Colorado, Boulder, October 1985.

[11] Krawczyk H., Kubale M.: An approximation algorithm for scheduling diagnostic tests in multicomputer systems, IEEE Trans. Comput. C-34 1985, 869-872.

[12] Kubale M.: The Complexity of Scheduling Independent Two-Processor Tasks on Dedicated Processors, Information Processing Letters 24, 1987, 141-147.

[13] Plehn J.: Preemptive scheduling of independent jobs with release times and deadlines on a hypercube, Information Processing Letters 34, 1990, 161-166.

[14] Seitz C.: The cosmis cube, Comm. of the ACM, no. 1, 1985, vol. 28, 22-33.

[15] Hertz A., de Werra D.: The Tabu Search Metaheuristic: How we used it, Annals of Mathematics and Artificial Intelligence 1990, 111-121.

Recenzent: Doc.dr h.in2. Franciszek Marecki
Wpłynęło do Redakcji do 30.04.1992.

Abstract: Classical models of scheduling assume that any task requires for its processing only one processor at a time. In some important applications a task may require more processors at a time. We model such a situation by assuming that each task $T_i^j \in T^j$ (j=1 or k) requires exactly j arbitrary processors (from among m identical processors) simultaneously during t_i^j time units within period $[r_i^j, d_i^j]$. Tasks are assumed to be independent and preemptable. The problem is to find a schedule with the minimum value of maximum lateness. Two algorithms are proposed. The first is based on linear programming the second - approximation one, is based on tabu search. Results of a computational comparison of the two methods, are also reported.