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

Michał MAŁAFIEJSKI, Łukasz KUSZNER, Marek KUBALE
Politechnika Gdańska

PODZIELNE SZEREGOWANIE ZADAŃ DWUPROCESOROWYCH NA MASZYNACH DEDYKOWANYCH W CELU MINIMALIZACJI SUMY CZASÓW ZAKOŃCZENIA

Streszczenie. W pracy rozważamy deterministyczne szeregowanie zadań dwuprocesorowych na maszynach dedykowanych, które minimalizuje sumę czasów zakończenia, przy czym dopuszcza się możliwość przerwania wykonywania zadania i ponownego wznowienia obsługi z pomijalnie małym kosztem. W standardowej notacji ten problem zapisujemy jako $P|fix_j = 2, pmtn|\Sigma C_j$.
Wiadomo, że tak postawione zagadnienie jest problemem silnie NP-trudnym. W pracy badamy złożoność obliczeniową problemu, ograniczając liczbę maszyn. Podajemy wielomianowy algorytm dla problemu $P4|fix_j = 2, pmtn|\Sigma C_j$.

PREEMPTIVE SCHEDULING OF BIPROCESSOR TASKS ON DEDICATED MACHINES TO MINIMIZE SUM OF COMPLETION TIMES

Summary. In this paper we consider a problem of preemptive scheduling of biprocessor tasks on dedicated processors in order to minimize the sum of completion times. Using the standard notation this problem is denoted as $P|fix_j = 2, pmtn|\Sigma C_j$.
This problem is strongly NP-hard. We analyze the subproblems obtained by reducing the number of processors. We give an exact polynomial algorithm for open problem $P4|fix_j = 2, pmtn|\Sigma C_j$.

1. Wstęp

Zajmiemy się klasą problemów szeregowania zadań wieloprocesorowych (ang. *multiprocessor task scheduling*). Danych jest n zadań $J_1, \ldots J_n \in \mathcal{J}$ oraz m maszyn (procesorów) $M_1, \ldots M_m \in \mathcal{M}$. Każde zadanie wykonywane jest na pewnym ustalonym (dedykowanym) podzbiorze maszyn. W pracy będziemy się zajmować systemami z zadaniami wykorzystującymi dokładnie dwa dedykowane procesory (ang. *biprocessor tasks*).

Każde zadanie będzie charakteryzowane przez czas wykonywania p_j oraz zbiór procesorów dedykowanych fix_j. Zakładać będziemy, że w legalnych uszeregowaniach żadne dwa zadania współdzielące przynajmniej jeden procesor nie będą wykonywane równocześnie.

O szeregowaniu mówimy, że jest:

- *podzielne* (ang. *preemptive scheduling*), jeśli obsługa każdego z zadań może być przerwana i kontynuowana później z pomijalnie małymi kosztami,
- *niepodzielne* (ang. *nonpreemptive scheduling*) w wypadku, gdy zadań nie wolno dzielić.

Jeśli występują zależności kolejnościowe pomiędzy niektórymi zadanami, to mówimy, że zadania są *zależne* (ang. *dependent*). Ograniczenia mogą być opisane przez digraf acykliczny ze zbiorem wierzchołków $\{J_i\}$ oraz zbiorem łuków postaci $J_i \rightarrow J_j$. Każdy taki łuk oznacza, że w legalnym uszeregowaniu zadanie J_i musi się zakończyć przed rozpoczęciem zadania J_j. Jeśli digraf acykliczny określający dopuszczalną kolejność wykonywania zadań ma taką własność, że dla każdego zadania Z, z wyjątkiem jednego, które ma zostać wykonane na początku, określone jest dokładnie jedno zadanie będące poprzednikiem zadania Z, to mówimy, że ta relacja ma postać *out-tree*.

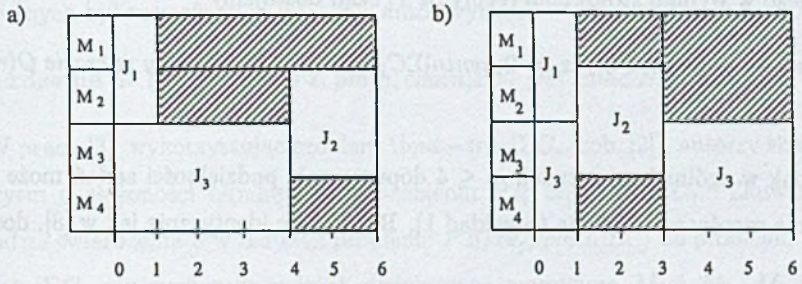
Czas zakończenia dla zadania J_j oznaczymy przez C_j. Kryterium oceny jakości uszeregowania jest minimalizacja sumy wszystkich czasów zakończenia $\sum C_j$ (ang. *total completion time*).

Wykorzystując notację $\alpha|\beta|\gamma$ z pracy [5] zapiszemy $P|fix_j = 2, pmtn|\Sigma C_j$, co oznacza problem szeregowania podzielnych zadań dwuprocesorowych na dedykowanych maszynach z kryterium minimalizacji sumy czasów zakończenia. W przypadku ustalonej liczby procesorów równej m symbol P zastąpiony będzie przez Pm.

Przykładem zastosowań tego modelu może być transfer plików w sieci komputerowej angażujący dwie maszyny jednocześnie [6] lub system wieloprocesorowy, w którym jednostki testują się wzajemnie wykonując parami testy diagnostyczne [7].

Przykład 1. Dla ilustracji weźmy pod uwagę system złożony z czterech maszyn M_1, M_2, M_3, M_4 oraz trzech zadań J_1, J_2, J_3. Zadanie J_1 ma zostać wykonane na maszynach M_1 i M_2, zadanie J_2 ma zostać wykonane na maszynach M_2 i M_3, zadanie J_3 ma zostać

wykonane na maszynach M_3 i M_4, co zapisujemy $fix_1 = \{M_1, M_2\}$, $fix_2 = \{M_2, M_3\}$, $fix_3 = \{M_3, M_4\}$. Czasy wykonywania wynoszą odpowiednio: $p_1 = 1$, $p_2 = 2$, $p_3 = 4$. Łatwo sprawdzić, że suma czasów zakończenia dla optymalnego uszeregowania wyniesie $1 + 4 + 6 = 11$ dla zadań niepodzielnych lub $1 + 3 + 6 = 10$, jeśli dopuścimy dzielenie zadań. □



Rys. 1. Optymalne uszeregowanie: a) niepodzielne ($\Sigma C_j = 11$), b) podzielne ($\Sigma C_j = 10$)
Fig. 1. The optimal schedule: a) nonpreemptive, b) preemptive

W dalszej części pracy zajmiemy się systemami z ograniczoną liczbą maszyn ($m \leq 4$). Przeanalizujemy złożoność wybranych problemów szeregowania oraz podamy algorytm wielomianowy.

2. Analiza złożoności problemu

Problem ogólny był rozważany w [2, 5, 8].

Twierdzenie 1 [8]. *Problem $P|fix_j, pmtn|\Sigma C_j$ jest silnie NP-trudny, gdy dopuścimy przerwania jedynie w całkowitych momentach czasu.* □

Wynik ten można wzmocnić, gdyż problem pozostaje silnie NP-trudny przy dowolnych przerwaniach zadań. W systemach z co najwyżej trzema procesorami można w danym momencie wykonywać co najwyżej jedno zadanie dwuprocesorowe. Mamy więc problem równoważny zwykłemu szeregowaniu zadań na jednym procesorze. Minimalizację ΣC_j

zapewnia stosowanie znanej reguły (zob. [2]) „najkrótsze najpierw" (SPT – *shortest processing time*), czyli do rozwiązania problemu wystarczy posortować zadania niemalejąco pod względem czasów wykonywania. Mamy więc

Twierdzenie 2. *Problem* $P3|fix_j = 2|\Sigma C_j$ *może być rozwiązany w czasie* $O(n \log n)$. □

Łatwo zauważyć, że jeśli dopuścimy podzielność zadań, nie uda nam się poprawić czasu uzyskanego w wyniku stosowania reguły SPT, skąd dostajemy

Wniosek 1. *Problem* $P3|fix_j = 2, pmtn|\Sigma C_j$ *może być rozwiązany w czasie* $O(n \log n)$. □

Jednak w ogólniejszej sytuacji $m \leq 4$ dopuszczenie podzielności zadań może zmniejszyć sumę czasów zakończenia (przykład 1). Rozumując identycznie jak w [8], dostajemy

Twierdzenie 3. $P4|fix_j = 2|\Sigma C_j$ *jest silnie NP-trudny.*

Dowód: W pracy [3] pokazano, że problem $P2|fix_j|\Sigma C_j$ jest silnie NP-trudny. Mając zatem instancję problemu $P2|fix_j|\Sigma C_j$, konstruujemy instancję problemu $P4|fix_j = 2|\Sigma C_j$ w następujący sposób: zadaniom dedykowanym dla M_1 przyporządkujemy zadania o takim samym czasie wykonywania dedykowane maszynom M_1 i M_2, zadaniom dedykowanym dla M_1 i M_2 przyporządkujemy zadania o takim samym czasie wykonywania dedykowane maszynom M_2 i M_3, zadaniom dedykowanym dla M_2 przyporządkujemy zadania o takim samym czasie wykonywania dedykowane maszynom M_3 i M_4. Zauważmy, że dowolnemu uszeregowaniu zadań skonstruowanej instancji problemu $P4|fix_j = 2|\Sigma C_j$ wzajemnie jednoznacznie odpowiada uszeregowanie instancji problemu $P2|fix_j|\Sigma C_j$ o takiej samej sumie czasów zakończenia, skąd wobec wielomianowości transformacji problemów dostajemy tezę. □

Analizując postać konfliktów pomiędzy zadaniami dwuprocesorowymi w pracy [4] częściowo poprawiono wynik podany w [3] wykazując

Twierdzenie 4 [4]. $P2|fix_j|\Sigma C_j$ *jest NP-trudny nawet wtedy, gdy ograniczymy instancje problemu do takich, że występuje tylko jedno zadanie do wykonania na maszynie* M_1, *jedno zadanie do wykonania na maszynach* M_1 *i* M_2, *a reszta zadań ma zostać wykonana na maszynie* M_2. □

Stąd, analogicznie jak w twierdzeniu 3, otrzymujemy

Wniosek 2. *$P4|fix_j = 2|\Sigma C_j$ jest NP-trudny nawet wtedy, gdy ograniczymy instancje problemu do takich, że występuje tylko jedno zadanie do wykonania na maszynach M_1 i M_2, jedno zadanie do wykonania na maszynach M_2 i M_3, a reszta zadań ma zostać wykonana na maszynach M_3 i M_4.* □

Problem szeregowania z ograniczeniami kolejnościowymi dla czterech maszyn i zadań podzielnych był rozważany w [8], gdzie autor wykazał

Twierdzenie 5. [8]. *$P4|fix_j = 2, pmtn, chain|\Sigma C_j$ jest silnie NP-trudny.* □

W pracy [3], wykorzystując problem $1|out-tree|\Sigma C_j$ (zob. [2]), autorzy skonstruowali algorytm o złożoności $O(n \log n)$ dla problemu $P2|fix_j, pmtn|\Sigma C_j$. Zauważmy, że w dowodzie twierdzenia 3 w redukcji problemu $P2|fix_j, pmtn|\Sigma C_j$ do problemu $P4|fix_j = 2, pmtn|\Sigma C_j$ nie wystąpiły zadania dedykowane maszynom M_1 i M_3, M_1 i M_4 oraz M_2 i M_4, a zatem problem $P4|fix_j = 2, pmtn|\Sigma C_j$, jako ogólniejszy, pozostawał wciąż nierozstrzygnięty.

W dalszej części artykułu uogólnimy wyniki z [3] wykorzystując redukcję problemu $P4|fix_j = 2, pmtn|\Sigma C_j$ do problemu $1|out-tree|\Sigma C_j$, dla którego znany jest algorytm dokładny o złożoności $O(n \log n)$ [2, 3].

3. Algorytm dla problemu $P4|fix_j = 2, pmtn|\Sigma C_j$

Oznaczmy maszyny przez M_1, M_2, M_3, M_4 oraz podzielmy zbiór zadań $\mathcal{J}$ na rozłączne klasy w zależności od dedykowanych par procesorów. Niech $\mathcal{J} = A_1 \cup A_2 \cup A_3 \cup A_4 \cup A_5 \cup A_6$, przy czym zadania z klasy A_1 mają zostać wykonane na maszynach M_1 i M_2, A_2 na M_3 i M_4, A_3 na M_1 i M_3, A_4 na M_2 i M_4, A_5 na M_1 i M_4 oraz A_6 na M_2 i M_3. Powiemy, że klasa A_i jest *stowarzyszona* z klasą A_j, o ile zadania z obu klas mogą być wykonywane równocześnie. Zauważmy, że są dokładnie trzy pary klas stowarzyszonych: A_1 i A_2, A_3 i A_4 oraz A_5 i A_6.

Własność 1. *Jeśli w pewnym przedziale czasowym $[a, b]$ wykonywane jest zadanie z klasy A_i, to równolegle może być wykonywane tylko zadanie z klasy A_j, stowarzyszonej z A_i.* □

Każde dwa zadania należące do tej samej klasy A_i są ze sobą w konflikcie. Jest więc sens mówić o kolejności wykonywania zadań w klasie A_i w pewnym legalnym harmonogramie uszeregowania zadań. Niech więc $J_{Ai(j)}$ oznacza zadanie, które w klasie A_i zakończy się jako j-te z kolei w tym harmonogramie.

Własność 2. *W każdym harmonogramie optymalnym, jeśli dwa procesory są bezczynne w pewnym przedziale czasu, to oznacza, że w klasie zadań dedykowanej tym procesorom nie pozostało już więcej zadań do wykonania.*

Dowód: Przypuśćmy przeciwnie, że dwa procesory są bezczynne w pewnym przedziale czasu oraz w odpowiadającej im klasie pozostały zadania do wykonania. Założyliśmy, że jest to harmonogram optymalny, jeśli więc pozostałoby jeszcze jakieś zadanie, to likwidując przestój i wykonując je wcześniej otrzymalibyśmy nowy harmonogram z sumą czasów mniejszą co najmniej o czas zlikwidowanego przestoju, sprzeczność z optymalnością harmonogramu. □

Załóżmy, że w przedziale czasu $[a, b]$ wykonywane jest bez przerw zadanie J. Powiemy wtedy, że w przedziale czasu $[a, b]$ wykonywany jest *fragment* zadania J, przy czym za *długość fragmentu* będziemy przyjmować wartość $b-a$. Jeżeli ponadto bezpośrednio przed momentem a oraz po momencie b nie jest wykonywane zadanie J, to taki fragment zadania J nazywać będziemy *blokiem*.

Własność 3. *W obrębie tej samej klasy zadań możemy zamienić kolejność wykonywania fragmentów o jednakowej długości dowolnych dwóch zadań, nie tracąc legalności uszeregowania, ani nie ingerując w czasy wykonywania pozostałych zadań.* □

Własność 4. *W każdym uszeregowaniu optymalnym zadanie $J_{Ai(j)}$ zaczyna się po zakończeniu zadania $J_{Ai(j-1)}$, dla $j > 1$.*

Dowód: Załóżmy przeciwnie, że zadanie $J_{Ai(j)}$ zaczyna się przed zakończeniem zadania $J_{Ai(j-1)}$. Z własności 3 wynika, że po zamianie kolejności wykonywania odpowiednich fragmentów obu zadań otrzymalibyśmy harmonogram z mniejszą sumą czasów zakończenia, czyli sprzeczność. □

Własność 5. *W każdym harmonogramie optymalnym zadania w każdej klasie muszą być uszeregowane w porządku niemalejących czasów wykonywania, zgodnie z regułą SPT.*

Dowód: Gdyby tak nie było, to moglibyśmy zamienić zadanie krótsze wykonywane później z zadaniem dłuższym wykonywanym wcześniej, nie zmieniając czasów wykonywania pozostałych zadań. W oczywisty sposób dostaniemy wówczas harmonogram z mniejszą sumą czasów zakończenia. □

Uporządkujmy zatem zadania w klasie A_i względem niemalejących czasów wykonywania. Przez $J_{Ai(j)}$ oznaczymy zadanie z klasy A_i o j-tym czasie wykonywania.

Własność 6. *W każdym uszeregowaniu optymalnym chwila rozpoczęcia wykonywania każdego bloku zadania jest chwilą zakończenia wykonywania innego zadania lub początkiem harmonogramu.*

Dowód: Załóżmy przeciwnie, że istnieje taki optymalny harmonogram, który nie spełnia tezy. Spośród wszystkich takich harmonogramów rozważmy taki harmonogram, w którym liczba bloków zadań, które nie rozpoczynają się w chwili zakończenia wykonywania pewnego innego zadania, jest najmniejsza z możliwych. Niech t_p oznacza najwcześniejszy moment rozpoczęcia bloku zadania, nie będący czasem zakończenia pewnego innego zadania ani początkiem harmonogramu. Dla ustalenia uwagi przyjmijmy, że w chwili t_p rozpoczyna się blok zadania $J_{A1(i)}$ z klasy A_1.

Możliwe są wyłącznie następujące przypadki:

1. Bezpośrednio przed chwilą t_p było wykonywane zadanie z klasy A_1 lub A_2.

   Korzystając z własności 2 i wiedząc, że w chwili t_p pozostało jeszcze nie zakończone zadanie $J_{A1(i)}$, stwierdzamy, że bezpośrednio przed czasem t_p było wykonywane zadanie z klasy A_1.

   Z założenia t_p jest chwilą rozpoczęcia wykonywania bloku zadania $J_{A1(i)}$, zatem korzystając z własności 4 i założenia o optymalności harmonogramu wnioskujemy, że w chwili t_p zadanie $J_{A1(i-1)}$ musiało się zakończyć, sprzeczność.

2. Bezpośrednio przed chwilą t_p było wykonywane zadanie należące do jednej z klas A_3, A_4, A_5 lub A_6.

   Niech krótszy z bloków zadań wykonywanych bezpośrednio przed momentem t_p rozpoczyna się w chwili $t_p - a$. Dla ustalenia uwagi przyjmijmy, że blok ten należy do klasy A_3.

Ponadto niech b oznacza czas, jaki upłynął od chwili t_p do pierwszego momentu zakończenia jakiegokolwiek zadania lub do momentu rozpoczęcia wykonywania kolejnego bloku zadania z A_3.

Załóżmy wpierw, że w chwili $t_p + b$ kończy się wykonywanie pewnego zadania. Ponieważ w czasie od t_p do $t_p + b$ nie jest wykonywane żadne zadanie z klasy A_3, zatem możemy zmodyfikować harmonogram zmniejszając o a czasy wykonywania wszystkich zadań wykonywanych w czasie od t_p do $t_p + b$ oraz odpowiednio zwiększając o b czasy wykonywania wszystkich zadań wykonywanych w czasie od $t_p - a$ do t_p. Ponieważ suma czasów zakończenia zmniejszy się o co najmniej a, otrzymamy sprzeczność z optymalnością początkowego uszeregowania.

Jeśli natomiast w chwili $t_p + b$ nie kończy się żadne zadanie, wtedy musi się zacząć fragment zadania z A_3. Zamieniając kolejność wykonywania zadań w taki sposób jak poprzednio, otrzymamy uszeregowanie z mniejszą liczbą bloków nie zaczynających się w chwili zakończenia innego zadania, sprzeczność.

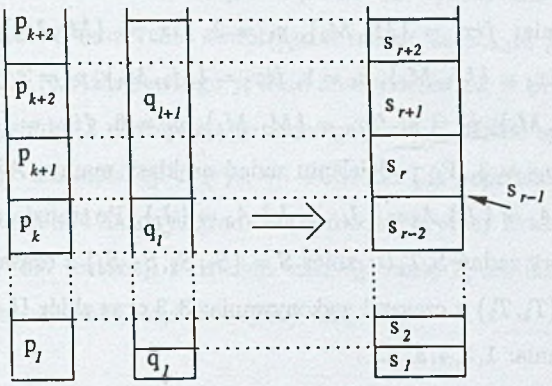
Dowód własności 6 został zakończony. □

Własność 7. *W każdym optymalnym uszeregowaniu przerwanie wykonania niezakończonego zadania $J_{Ai(k)}$ z klasy A_i następuje tylko w chwili zakończenia zadania z klasy A_j z nią stowarzyszonej.*

Dowód: Załóżmy, że w pewnym optymalnym harmonogramie zadanie $J_{Ai(k)}$ zostało przerwane. Przerwanie musiało być spowodowane rozpoczęciem wykonywania fragmentu pewnego zadania J będącego w konflikcie z $J_{Ai(k)}$, czyli pochodzącego z jednej z klas różnych od A_i i A_j na mocy własności 1. Z własności 6 wynika, że rozpoczęcie wykonywania fragmentu zadania J następuje bezpośrednio po zakończeniu pewnego innego zadania, a zatem mogło to być tylko zadanie z klasy A_j. □

Weźmy teraz pod uwagę instancję I problemu $P4|fix_j = 2, pmtn|\Sigma C_j$ z n zadaniami. Skonstruujemy instancję problemu $1|out-tree|\Sigma C_j$ w następujący sposób. Każdej parze klas stowarzyszonych odpowiadać będzie łańcuch zadań utworzony zgodnie z rysunkiem 2.

Weźmy dla przykładu klasy A_1 i A_2. Uporządkujmy zadania z klasy A_1 według niemalejących czasów wykonywania $p_1 \leq p_2 \leq \ldots \leq p_a$, podobnie dla klasy A_2 otrzymujemy $q_1 \leq q_2 \leq \ldots \leq q_b$. Zgodnie ze schematem przedstawionym na rysunku 2 tworzymy



Rys. 2. Transformacja problemu szeregowania zadań podzielnych na niepodzielne
Fig. 2. Transformation of preemptive scheduling tasks into nonpreemptive

zadania $S_1, S_2, \ldots, S_{a+b}$ o czasach wykonywania $s_1, s_2, \ldots, s_{a+b}$. W wypadku gdy $\Sigma_{i=1}^{k} p_i = \Sigma_{i=1}^{l} q_i$, tworzymy zadanie o czasie wykonywania 0. Dodatkowo wprowadzamy łańcuch zależności: $S_r \rightarrow S_{r+1}$ dla $r \in \{1, \ldots, a+b-1\}$.

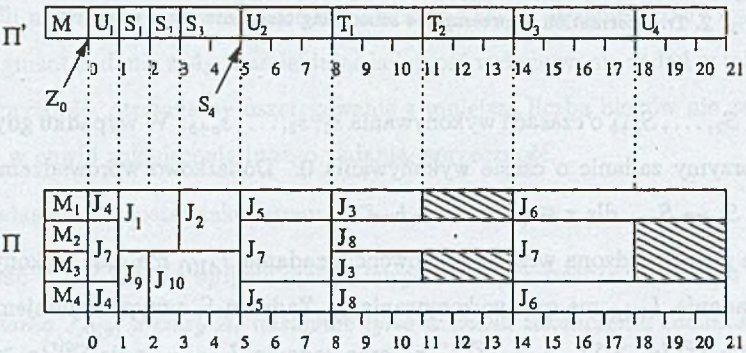
Zgodnie z wprowadzoną wcześniej konwencją zadanie $J_{A1(i)}$ ma czas wykonywania p_i, podobnie zadanie $J_{A2(j)}$ ma czas wykonywania q_j. Zadaniu S_r przyporządkujemy zadanie $J_{A1(p(r))}$, o ile $\Sigma_{i=1}^{p(r)} p_i \geq \Sigma_{i=1}^{r} s_i > \Sigma_{i=1}^{p(r)-1} p_i$, oraz zadanie $J_{A2(q(r))}$, o ile $\Sigma_{i=1}^{q(r)} q_i \geq \Sigma_{i=1}^{r} s_i > \Sigma_{i=1}^{q(r)-1} q_i$, na przykład na rysunku 2 mamy $p(r) = k+1, q(r) = l+1, p(r+1) = k+2$.

Analogicznie tworzymy zadania T_r z klas A_3 i A_4 oraz U_r z klas A_5 i A_6, o czasach wykonywania odpowiednio t_r i u_r, wraz z odpowiednimi łańcuchami zależności. Wprowadzamy dodatkowo zadanie Z_0 z czasem wykonywania 0, które ma być wykonane przed zadaniami S_1, T_1, U_1. Otrzymaliśmy zatem instancję I' problemu $1|out-tree|\Sigma C_j$ z $n+1$ zadaniami.

Możemy znaleźć optymalne uszeregowanie Π' w czasie $O(n \log n)$ [1]. Na podstawie uszeregowania Π' dla instancji I' konstruujemy uszeregowanie Π dla instancji I problemu $P4|fix_j = 2, pmtn|\Sigma C_j$ w następujący sposób: jeśli w chwili t uszeregowania Π' wykonywane jest zadanie S_i, to w uszeregowaniu Π wykonywane są zadania $J_{A1(p(i))}$ oraz $J_{A2(q(i))}$. Analogicznie postępujemy dla zadań T_r, U_r. Pokażemy teraz przykład takiej konstrukcji.

Przykład 2. Dla ilustracji weźmy pod uwagę system złożony z czterech maszyn M_1,

M_2, M_3, M_4 oraz dziesięciu zadań $J_1, J_2, \ldots, J_{10}$, o zbiorach procesorów dedykowanych i czasach wykonywania: $fix_1 = \{M_1, M_2\}$, $p_1 = 2$, $fix_2 = \{M_1, M_2\}$, $p_2 = 2$, $fix_3 = \{M_1, M_3\}$, $p_3 = 3$, $fix_4 = \{M_1, M_4\}$, $p_4 = 1$, $fix_5 = \{M_1, M_4\}$, $p_5 = 3$, $fix_6 = \{M_1, M_4\}$, $p_6 = 7$, $fix_7 = \{M_2, M_3\}$, $p_7 = 8$, $fix_8 = \{M_2, M_4\}$, $p_8 = 6$, $fix_9 = \{M_3, M_4\}$, $p_9 = 1$, $fix_{10} = \{M_3, M_4\}$, $p_{10} = 3$. Po podzieleniu zadań na klasy mamy: $A_1 = \{J_1, J_2\}$ $A_2 = \{J_9, J_{10}\}$ $A_3 = \{J_3\}$ $A_4 = \{J_8\}$ $A_5 = \{J_4, J_5, J_6\}$ $A_6 = \{J_7\}$. Po transformacji na instancję I' otrzymujemy zbiory zadań S, T, U: zbiór $S = \{S_1, S_2, S_3, S_4\}$ z czasami wykonywania: $1, 1, 2, 0$, zbiór $T = \{T_1, T_2\}$ z czasami wykonywania: $3, 3$ oraz zbiór $U = \{U_1, U_2, U_3, U_4\}$ z czasami wykonywania: $1, 3, 4, 3$. □



Rys. 3. Optymalne uszeregowanie Π' oraz odpowiadające mu uszeregowanie Π
Fig. 3. Optimal schedule Π' and the corresponding schedule Π

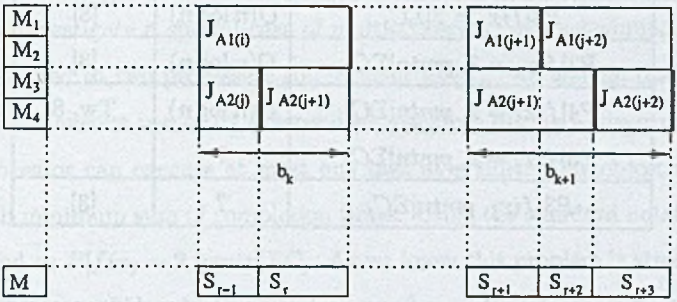
Ponieważ zadanie Z_0 trwa 0 jednostek czasu, więc nie wpływa na sumę czasów zakończenia. Pozostałe zadania z uszeregowania Π' kończą się w tej samej chwili, co odpowiadające im zadania w uszeregowaniu Π, zatem dostajemy

Lemat 6. *Dla dowolnego legalnego uszeregowania Π' dla instancji I' istnieje uszeregowanie Π dla instancji I o takiej samej sumie czasów zakończenia.* □

Lemat 7. *Uszeregowanie optymalne dla instancji I' ma taką samą sumę czasów zakończenia jak optymalne uszeregowanie dla instancji I.*

Dowód: Rozważmy uszeregowanie optymalne $\hat{\Pi}$ dla instancji I. Przez b_k oznaczymy czas wykonywania k-tej maksymalnej części harmonogramu $\hat{\Pi}$ (rys. 4), w której wykonywane

są wyłącznie zadania z klas A_1 i A_2, bez przerw na wykonywanie zadań z innych klas. Na mocy własności 5 oraz 7 dla dowolnego k mamy, że $\Sigma_{r=1}^{k} b_k$ jest równa $\Sigma_{r=1}^{i} p_r$, dla pewnego i lub $\Sigma_{r=1}^{j} q_r$, dla pewnego j, skąd analogicznie jak w przypadku transformacji pokazanej na rysunku 3 dostajemy, że uszeregowanie $\widehat{\Pi}$ można sprowadzić do pewnego uszeregowania $\widehat{\Pi}'$ dla instancji I' (rys. 4). Podobnie jak poprzednio zauważmy, że sumy czasów zakończenia obu harmonogramów są równe. Na mocy lematu 6 dla uszeregowania optymalnego Π' dla instancji I' istnieje uszeregowanie Π dla instancji I o identycznej sumie czasów zakończenia, skąd dostajemy tezę. □



Rys. 4. Ilustracja redukcji uszeregowania $\widehat{\Pi}$ do uszeregowania dla instancji I'
Fig. 4. Illustration for reduction of the schedule $\widehat{\Pi}$ into the schedule for instance I'

Podsumowując, dla instancji I problemu $P4|fix_j = 2, pmtn|\Sigma C$ możemy w czasie $O(n \log n)$ skonstruować instancję I' problemu $1|out - tree|\Sigma C_j$ (rys. 2), następnie w czasie $O(n \log n)$ potrafimy znaleźć uszeregowanie optymalne Π' dla instancji I' [1, 2]. Na mocy lematu 7 skonstruowane uszeregowanie Π dla instancji I jest optymalne, skąd ostatecznie mamy

Twierdzenie 8. *Istnieje algorytm dokładny o złożoności $O(n \log n)$ rozwiązujący problem $P4|fix_j = 2, pmtn|\Sigma C_j$.* □

4. Podsumowanie

Wyniki prezentowane w pracy zostały zawarte w tablicy 1.

Wciąż nierozstrzygnięta pozostaje złożoność problemu $P5|fix_j = 2, pmtn|\Sigma C_j$ nawet

Tablica 1

Klasyfikacja złożoności

Problem	Złożoność	Referencja
$P\|fix_j, pmtn\|\Sigma C_j$	sNPh	[8]
$P2\|fix_j\|\Sigma C_j$	sNPh	[3]
$P4\|fix_j = 2\|\Sigma C_j$	sNPh	Tw. 3
$P4\|fix_j = 2, p_j = 1, chain\|\Sigma C_j$	NPh	[8]
$P2\|fix_j, pmtn\|\Sigma C_j$	$O(n \log n)$	[3]
$P3\|fix_j = 2\|\Sigma C_j$	$O(n \log n)$	[8]
$P3\|fix_j = 2, pmtn\|\Sigma C_j$	$O(n \log n)$	[8]
$P4\|fix_j = 2, pmtn\|\Sigma C_j$	$O(n \log n)$	Tw. 8
$P5\|fix_j = 2, pmtn\|\Sigma C_j$	?	
$P3\|fix_j, pmtn\|\Sigma C_j$	?	[3]

w przypadku, gdy zadania wykonywane są wyłącznie na parach procesorów $\{M_i, M_i + 1\}$ dla $i = 1, \ldots, 4$. Podobnie otwarty pozostaje problem $P3|fix_j, pmtn|\Sigma C_j$ [3].

LITERATURA

1. Adolphson D., Hu T. C.: Optimal linear ordering, SIAM J. Appl. Math. 25, 1973, pp. 403-423.
2. Brucker P.:Scheduling algorithms, Springer-Verlag, Berlin 1995.
3. Cai X., Lee C. Li C.: Minimizing total completion time in two-processor task systems with prespecified processor allocations, Naval Res. Logist. 45, 1998, pp. 231-242.
4. Giaro K., Kubale M., Małafiejski M., Piwakowski K.: Dedicated scheduling of biprocesor tasks to minimize mean flow time, Lecure Notes in Computer Science 2328 (2002), pp. 87–96.
5. Hoogeveen J.A., van de Velde S.L., Veltman B.: Complexity of scheduling multiprocessor tasks with prespecified processor allocations, Discrete Applied Mathematics 55, 1994, pp. 259-272.
6. Coffman E.G., Garey M.R., Johnson D.S., LaPaugh A.S.: Scheduling file transfers, SIAM Journal on Computing 14, 1985, pp. 744-780.
7. Krawczyk H., Kubale M.: An approximation algorithm for diagnostic test scheduling in multicomputer systems, IEEE Transactions on Computers 34, 1985, pp. 869-872.

8. Kubale M.: Preemptive versus nonpreemptive scheduling of biprocessor tasks on dedicated processors, European Journal of Operational Research 94, 1996, pp. 242-251.

Recenzent: Prof. dr hab. inż. Andrzej Świerniak

Abstract

This paper is devoted to the complexity of a class of multiprocessor scheduling problems. An essential property of many real-life systems is parallel processing, so a classical assumption that each job should be processed by one processor only has no longer been justified. We investigate a special case of multiprocessor tasks scheduling with all tasks having preassigned to two processors (biprocessor tasks). We assume that each task requires the simultaneous use of its prespecified machines during a given processing time but each processor can execute at most one task at a time. Our objective is to find a schedule with minimum sum of completion times. Using the standard notation this problem is denoted as $P|fix_j = 2, pmtn|\Sigma C_j$. As we know this problem is strongly NP-hard. We analyze its subproblems by reducing the number of processors up to a fixed number m. When $m \leq 3$ all tasks become incompatible, therefore this problem is equivalent to scheduling ordinary jobs on one processor. Our main result is a polynomial algorithm for so far unsolved problem $P4|fix_j = 2, pmtn|\Sigma C_j$. The cases $m > 4$ remains open. Comparing preemptive and nonpreemptive models it appears that $P4|fix_j = 2|\Sigma C_j$ is strongly NP-hard.