Dariusz DOROTA
Politechnika Krakowska

## SZEREGOWANIE ONLINE ZADAŃ WIELOPROCESOROWYCH

**Streszczenie.** Zdefiniowano problem szeregowania on-line zadań wieloprocesorowych, tzn. wymagających synchronicznego równoczesnego użycia wielu procesorów do realizacji zadania, w warunkach niepewności co do repertuaru i liczby pojawiających się zadań. Zaproponowano nowe algorytmy szeregowania działające w czasie rzeczywistym oraz udowodniono pewne ich własności. Rozważono przypadki zadań przerywalnych, nieprzerywalnych, zależnych i niezależnych. Problem ma zastosowanie w systemach wbudowanych o zwiększonej niezawodności działania.

## ONLINE SCHEDUING OF MULTIPROCESSORS TASKS

**Summary.** There is defined problem of on-line scheduling multiprocessors tasks, i.e. requiring the simultaneous use of a few processors, taking into account some uncertainty of providing a variety of incoming tasks. There are proposed some real-time algorithms and proved some their properties. There have been considered cases of preemptive tasks, non-preemptive tasks, depending and independent tasks. Problem has application in the embedded systems with the increased dependability.

## 1. Wstęp

Zagadnienia szeregowania modelują rzeczywiste systemy dedykowane dla różnych zastosowań. W systemach wbudowanych IoT (ang. *Internet of Things*), IoV (ang. *Internet of Vehicles*) o zwiększonym poziomie bezpieczeństwa występuje szczególna klasa zagadnień tego typu nazywana szeregowaniem zadań wieloprocesorowych. Biorąc pod uwagę tendencje w zakresie uniformizacji i modularyzacji sprzętu, zwiększoną niezawodność osiąga się poprzez wprowadzenie redundancji sprzętowej, pozwalającej także na redundancję programową. Redundancja ma na celu automatyczne podejmowanie właściwych decyzji drogą "głosowania" powielonych, identycznych urządzeń, wykonujących te same funkcje i realizujących takie same programy dla identycznych danych. Systemy takiego typu są już powszechnie stosowane między innymi w lotnictwie, pojazdach kosmicznych, samochodach, urządzeniach wojskowych, środkach transportu materiałów niebezpiecznych, instalacjach jądrowych, instalacjach chemicznych, górnictwie, dronach. Łatwość realizacji powoduje rozszerzane takich rozwiązań na kolejne obszary działalności człowieka.

## 2. Sformułowanie problemu

Oznaczmy przez $T = \{T_1, T_2, T_3, ..., T_n\}$ zbiór zadań do wykonania na identycznych maszynach (procesorach) wykorzystywanych równolegle, zdefiniowanych jako $M = \{M_1, M_2, M_3, ..., M_m\}$. Zadanie obliczeniowe $T_i \in T$ trwa $p_i$ jednostek czasu i wymaga do wykonywania synchronicznego (jednoczesnego) dostępu do $a_i \geqslant 1$ procesorów. W zbiorze $T$ jest określona relacja poprzedzania zadań $R \subset T \times T$. Mówimy, że $(T_i, T_j) \in R$ jeśli zadanie $T_i$ musi się zakończyć przed rozpoczęciem zadania $T_j$. Graf $G = (T, R)$ jest acykliczny. W niektórych przypadkach $G$ może mieć specyficzną postać, np. drzewa. Harmonogram realizacji zadań jest opisany poprzez parę wektorów $(S, A)$, gdzie $S = (S_1, \ldots, S_n)$, $S_i$ jest terminem rozpoczęcia zadania $T_i$; $A = (A_1, \ldots, A_n)$, $A_i \subseteq M$, $|A_i| = a_i$, $A_i$ jest zbiorem procesorów alokowanych równocześnie do realizacji zadania $T_i$. Należy wyznaczyć harmonogram pracy procesorów, tak by spełnione były powyższe ograniczenia, oraz minimalizowany był wskaźnik jakości, przyjęty tutaj jako długość uszeregowania (ang. *makespan*) $C_{\max} = \max_{1 \leqslant i \leqslant n}(S_i + p_i)$. Zauważmy, że powyższy opis w konwencji specyficznego problemu szeregowania zadań może mieć równoważny opis jako pewien przypadek deterministyczny RCPS (ang. *Resource Constrained Project Scheduling*). Wynika to z faktu, że procesory mogą być traktowane również jako zasób dyskretny, podzielny z dokładnością do jednostki, żądany przez różne zadania w różnej ilości. Problem ten ma także związek z zagadnieniem optymalnego cięcia taśmy materiału (ang. *strip cutting problem*) [16]. Znane są także inne specyficzne metody modelowania i rozwiązywania zgadnień tego typu jednak, odmiennie od omawianego przypadku, dotyczą one problemów deterministycznych, patrz np. praca [21].

W przypadku dopuszczenia przerywania zadań zakłada się, że zadanie $T_i \in T$ składa się z sekwencji pewnej liczby $x_i$ operacji $O_i = (o_{i,1}, o_{i,2}, o_{i,3}, ..., o_{i,x_i})$, wykonywanych w tej kolejności, z których każda wymaga synchronicznego użycia $a_i$ procesorów. Oczywiście, musi zachodzić $\sum_{k=1}^{x_i} |o_{i,k}| = p_i$, gdzie $|o_{i,k}|$ oznacza czas trwania operacji $o_{i,k}$. Wówczas podział zdania na operacje nie jest ustalony i podlega wyborowi.

Wychodząc od podstawowej taksonomii Grahama i in. [14] oraz powszechnie używanego zapisu $\alpha|\beta|\gamma$, w książce [24] zaproponowano pewne rozszerzenie znaczenia symbolu $\alpha$ w celu uwzględnienia obsługi równoległej zadań dla różnych problemów szeregowania. Niestety, nawet ta propozycja nie pozwala na symboliczne zapisanie problemu omawianego w tej pracy. Dlatego też, dla potrzeb klasyfikacyjnych rozszerzono zakres znaczeń symbolu $\beta$ zamiast $\alpha$. Mianowicie w $\beta$, $a_i = k$ oznacza, że wszystkie zadania są $k$-procesorowe, $a_i \leqslant k$ oznacza, że zadania są "wymieszane" $1, 2, \ldots, k$-procesorowe, zaś samo $a_i$ oznacza, że zadania są dowolnie-procesorowe. Propozycja ta jest pewną zmianą w stosunku do koncepcji przedstawionej w pracach [24, 3, 26, 16, 18, 13], bowiem wprowadza model wieloprocesorowy niepewny poprzez ograniczenia problemu, a nie poprzez strukturę systemu. Krzystając z podanej tak rozszerzonej taksonomii, problemy rozptrywane w tej pracy są klasy $P|\beta|C_{\max}$, gdzie $\beta \in \{\circ, a_i = k, a_i \leqslant k, a_i, pmtn, prec, fuzzy(p), random(p)\}$. Zaważmy, że sformułowany problem jest istotnie odmienny od klasycznego $P, R, Q||C_{\max}$ ze względu na $a_i$ oraz ze względu na niepewność danych $p_i$ i niepewność zbioru zadań $T$.

## 3. Szeregowanie niepewne

Generalnie rozważamy tylko problemy z danymi niepewnymi, Niepewność w problemach szeregowania może dotyczyć różnych danych i założeń problemu [2]. Mogą to być odpowiednio: (A) nieokreślone czasy pojawienia się zadania; (B) nieokreślone czasy zakończenia lub nieokreślone czasy trwania zadania; (C) nieznane czasy transferu danych (transmisji) pomiędzy zadaniami; (D) nieznane czasy przepływu; (E) nieznane czasy wystąpienia awarii maszyn; (F) nieznane czasy przestoju maszyn, (G) nieznana relacja poprzedzania zadań, (H) nieznane żądania zasobowe, (I) nieznana liczba i zbiór zadań do wykonywania. Nieznajomość parametrów liczbowych (A)...(F) może być modelowana i rozwiązywana w systemach rozmytych lub za pomocą zmiennych losowych. Choć podejścia takie występują w szeregowaniu, to jednak dotychczas nie formułowano ich i nie rozważano ich w kontekście zadań wieloprocesorowych. Tematyka tych badań pozostaje wciąż otwarta. Warianty (G) i (H) nie występowały dotychczas w literaturze. Z kolei wariant (I) wymaga sprecyzowania założeń dotyczących napływających zadań i ich obsługi. Nieskończony napływ zgłoszeń jest założeniem trudnym do przyjęcia w zamkniętych systemach IoT, IoV, choć ma sensowny charakter. Możemy bowiem założyć, że powtarzalne zadania obliczeniowe są unikalne ze względu na dane, napływają losowo, w strumieniu będącym superpozycją kilku różnych strumieni. Alternatywnym założeniem jest realizacja skończonego, lecz nieznanego z góry podzbioru zadań, wybranego ze znanego większego zbioru zadań. Założenie pierwsze pozwala nam modelować i rozwiązywać problem z pomocą kolejek otwartych lub zamkniętych. Założenie drugie – za pomocą obsługi online. W niniejszej pracy, podejście z pracy własnej [9], zostało zmodyfikowane oraz dostosowane do szeregowania online zadań wieloprocesorowych.

## 4. Szeregowanie online

W szeregowaniu *online* proces przypisywania priorytetów odbywa się na bieżąco, w trakcie funkcjonowania systemu przy założeniu, że napłyną zadania ze zbioru $T$ w pewnych nieznanych apriori momentach czasu, w nieznanej kolejności. Podejście to jest bardziej elastyczne niż offline. W literaturze [24, 12, 5, 8, 1] wszystkie modele szeregowania online można opisać w następujący zunifikowany sposób: (A) lista online (ang. *online list*); (B) czas online (ang. *online time*); (C) aktywność maszyn (ang. *machine activity*). Zasadnicza różnica pomiędzy szeregowaniem online i offline to dostępność informacji o danych: (a) offline – pełna przed rozpoczęciem procesu szeregowania; (b) online – udostępniana stopniowo w trakcie realizacji procesu szeregowania. Dodatkowo przesłankami skłaniającymi do wyboru podejścia (b) mogą też być niekompletność informacji o danych, niepewność czy też indeterminizm zdarzeń czasowych [19, 7]. Model *online list* zakłada, że zadania pojawiają się przypadkowo, pojedynczo, przy czym każde zadanie określa swoją charakterystykę (np. czas trwania, żądania zasobowe). Dla każdego przychodzącego zadania dokonywane jest natychmiastowe zaplanowanie jego wykonania, i decyzja ta nie może zostać zmieniona w przyszłości. Model *online time* zakłada, że zadania pojawiają się przypadkowo, dopuszcza się równoczesność występowania zgłoszeń, niektóre dane z charakterystyki zadania moga być znane, a inne nie. Decyzje uwzględniają wszystkie zadania oczekujace w kolejce, są dynamicz-

ne, zmienne w czasie, szeregowanie jest zwykle przerywalne. Algorytmy szeregujące mogą podejmować decyzje deterministycznie (*algorytmy deterministyczne*) lub losowo (*algorytmy zrandomizowane*). W literaturze wymienia się również inną klasyfikację algorytmów szeregowania online [25, 17]: (A) *przewidujące* (ang. *clairvoyant*), w których jest znany czas obsługi zadań, (B) *nieprzewidujące* (ang. *non-clairvoyant*), w których czas obsługi zadań jest nieznany. Miarą oceny jakości algorymów online jest *współczynnik konkurencyjności*. Porównuje on wartość funkcji celu $f()$ dostarczoną algorytmem online $A$ w odniesieniu do optymalnej wartości funkcji celu $OPT$ jaką można uzyskać a posteriori algorytmem offline, dla każdego przypadku danych $I$. Ponieważ formalne definicje współczynnika konkurencyjności są różne w zależności od klasy *algorytmów* (deterministyczne, zrandomizowane) oraz klasy *przeciwników* (ang. *adaptive adversary*, *strong adaptive adversary*), nie będziemy szerzej dyskutować tego tematu poprzestając na przytoczeniu niezbędnej definicji współczynnika $r$ dla algorytmu deterministycznego

$$f(A, I) \leqslant r \cdot f(OPT, I) + \alpha, \tag{1}$$

gdzie $\alpha$ jest pewną stałą.

## 5. Zadania wieloprocesorowe

Biorąc pod uwagę koncepcję niezawodności, skupiono się w tej pracy głównie na wykorzystaniu zadań jedno- dwu- oraz trzy-procesorowych, tzn. $a_i = 1, 2, 3$. Użycie zadań dwuprocesorowych polega na wzajemnym testowaniu wyniku otrzymanego przez dwa niezależne procesory przed podjęciem finalnej decyzji [9]. Jeżeli odpowiedzi są identyczne wtedy zadanie jest uważane za prawidłowo przetworzone. Zadanie trzyprocesorowe stosuje zasadę głosowania większością, niemożliwą do spełnienia w systemie dwu-procesorowym. Trzeci procesor może być postrzegany jako arbiter, który w razie konieczności przesądza o ostatecznym wyniku w przypadku gdy odpowiedzi dwóch zadań nie są jednoznaczne. Pozwala to na zwiększenie stopnia niezawodności systemu, co zostało wykazane w [10]. W praktyce, aby nie zwiększać niepotrzebnie złożoności całego systemu, redundancja zadań jest realizowana tylko dla niektórych zadań, których realizacja jest krytyczna dla funkcjonowania systemu. Stąd, jest sens rozwiązywania problemów zarówno z $a_i = k$, jak i $a_i \leqslant k$.

## 6. Algorytmy

Dla niepewnego szeregowania online zaprojektowano nowe algorytmy z zakresem $a_i \leqslant k < m$ będące rozwinięciem odpowiednich deterministycznych algorytmów offline z $a_i = 1$, patrz np. [13]. Punktem wyjścia do rozważań oraz otrzymania referencyjnego rozwiązania offline są algorytmy dla zadań jednoprocesorowych, mianowicie algorytm Mutza-Coffmana [23] dla jednoprocesorowych zależnych zadań przerywalnych oraz LPT (Longest Processing Time [15]) dla jednoprocesorowych nieprzerywalnych zadań niezależnych. Zauważmy, że nie wszystkie zagadnienia offline są wielomianowe. Nowe algorytmy dla przypadku zadań wieloprocesorowych z $a_i > 1$ występują dalej w tej pracy pod nazwami: algorytm MC (Algorytm 1+2) oraz m-LPT (Algorytm 3), odpowiednio. Ponieważ oceny współczynnika konkurencyjności zawarte w kolejnych rozdziałach dokonano dla przypadku zadań niezależnych, nie formułowano wersji Algorytmu 3 dla zadań zależnych.

Zastosowane oznaczenia dla problemu szeregowania online: $C_i$ - termin zakończenia zadania $T_i$, $w_i = p_i \cdot a_i$ priorytet zadania $T_i$, $d_i$ - najpóźniejszy możliwy czas rozpoczęcia zadania $T_i$, $D_i = d_i + p_i$ - najpóźniejszy możliwy termin zakończenia zadania $T_i$ (dwa ostatnie oznaczenia są opcjonalnie określane w specyfikacji). Algorytm 1 oraz 2 dla szeregowania on-line zadań wieloprocesorowych przedstawiono w formie pseudokodu. Kolejność wyznaczania i analizowania priorytetów zależy od zadań i grafu $G$ oraz jego struktury. Zależność zadań reprezentowana grafem $G$ jest oznaczona w pseudokodzie jako TG. Opis znaczenia kroków algorytmu podano w komentarzach.

**Algorytm 1.** *Szeregowanie zadań wieloprocesorowych, wersja online-owa:*

1. *t = 0;* $TG_{struct} = read(TG)$*;*
2. $TG_{path} = find\_paths(TG_{struct})$*;* /* $TG_{path}$ jest zbiorem ścieżek, jeżeli $TG_{path} = \emptyset$ to $TG_{path} = TG_{struct}$ */
3. **for** $T_i \in T$ **do** $TG_{level} = calculate\_levels(TG_{path})$; */* oblicznie priorytetów zadań */*
4. $MP = TG_{level}.w$*;* /* MP jest zbiorem priorytetów */
5. $MMP = \max\{w_i : T_i \in MP\}$*;* /* wyznaczenie maksymalnych priorytetów w zbiorze MP */
6. $HP = \{T_j \in T : \ w_j = MMP\}$ /* znajdź wszystkie zadania ze zbioru MMP */
7. **for** $T_i \in HP$ **do**
8. **if** $(w_i > 1)$ /* analizuj zadania z niezerowymi priorytetami */
9. **if** $(|HP| > 1)$ /* Jeżeli $H_i$ jest identyczne dla więcej niż jednego zadania */
10. $LHP = \max(TG_{level}.high)$; /* LHP jest zbiorem zadań o takich samych $H_i$ ze zbioru HP */
11. **if** $(|LHP| > 1)$
12. $w_i = \max\{w_j : \ T_j \in LHP\}$ */** wybierz z LHP zadanie o najwyższym indeksie **/*
13. **else** $w_i = \max\{w_j : T_j \in HP\}$
14. **else** $w_i = MMP$*;*
15. **end for;**
16. **for** $T_i \in T$ **do**
17. **if** $((p_i - [p_i]) == 0)$
18. $x = 1$*;*
19. *usuń* $w_i$ *ze zbioru* $MP$*;*
20. **else** $x = p_i - [p_i]$*;*
21. **end for***;*
22. **for** $T_i \in TG^F_{struct}$ **do**
23. *uszereguj* $T_i$ *dla jednostki czasowej* $x$ *wg algorytmu McNaughton'a [22];*
24. **if** $|P(t)| > 0$
25. **go to** *step 8* /* Funkcja $P(t)$ sprawdza czy istnieje niezajęty procesor/niezajęte procesory w chwili czasowej $t$ */
26. **else**

*27.* $p_i = p_i - x$; $t = t + x$;
*28.* **end for;**

**Algorytm 2.** *Funkcja* $calculate\_levels(TG_{path})$ *oblicza priorytety zadań.*

*1.* **if** $TG_{path} = \emptyset$
*2.* **for** $T_j \in TG_{path}$ do
*3.* $w_j = p_j \cdot a_j$;
*4.* $TG_{level}.add(w_j, p_j)$;
*5.* **end for;**
*6.* **else**
*7.* **for** $A_j \in TG_{path}$ **do** /* $A_j$ jest ścieżką z ostatnim zadaniem $T_j$ */
*8.* $w_j = \sum_{k \in A_j} p_k \cdot a_k$; $time_j = \sum_{k \in A_j} p_k$; $high_j = \sum_{k \in A_j} E_k$;
*9.* $TG_{level}.add(w_j, p_j)$;
*10.* *oznacz* $A_j$ *jako obliczone;*
*11.* **end for;**
*12.* **for** $w_i \in TG_{level}$ **do**
*13.* **if** $\exists(\max(TG_{level}.time) \leqslant D_i)$
*14.* $w_i = \max(TG_{level}(w_j))$
*15.* **else**
*16.* *Uszeregowanie nie istnieje*
*17.* **end for;**

Wyznaczanie ścieżki dla zadań powiązanych grafem TG jest realizowane przy użyciu algorytmu $A^*$ [20] przez procedurę $designate\_path$. Ponieważ w wersji online-owej opisanego algorytmu mogą zostać wzięte pod uwagę zarówno zadanie zależne jak i niezależne to, aby zapewnić uniwersalność algorytmu, muszą być obsłużone obie sytuacje.

$designate\_path(TG_{struct})$

1. **if** $(\forall T_i \; \forall \; T_j, \exists(\text{path } T_i \rightarrow T_j))$
   $TG_{path} = calculate\_path(TG_{struct})$ /* użycie algorytmu $A^*$ */
2. **else** $TG_{path} = NULL$;

Omówimy krótko zaproponowane algorytmy 1+2. Jako specyfikacja systemu przyjmowany jest graf zadań, gdzie wprowadzono oznaczenia dla zadań wieloprocesorowych (wybierane w sposób losowy, co do liczby żądanych zasobów oraz numeru zadania). Jeżeli istnieją zależności pomiędzy zadaniami to z wykorzystaniem algorytmu A* [20] są wyszukiwane ścieżki w grafie określające zależności pomiędzy zadaniami. Odwołanie się do funkcji $calculate\_levels$, przenosi obsługę do algorytmu 2, który to opisuje sposób postępowania przy wyznaczaniu priorytetów. Zgodnie z zaproponowanym podejściem przy wyznaczaniu priorytetów dużą rolę odgrywa przede wszystkim atrybut wieloprocesorowości zadania, a także ewentualna zależność zadań. Jeżeli jej nie ma, to wyznaczanie priorytetu opiera się o parametr $a_i$. W przeciwnym przypadku, dodatkowo brana jest pod uwagę sumaryczna wartość wszystkich priorytetów zadań znajdujących się na ścieżce, począwszy od zadania aktualnie rozpatrywanego, a na ostatnim zadaniu w rozpatrywanej ścieżce skończywszy. Ze zbioru priorytetów wybierane są zadania

o najwyższych wartościach uwzględniając położenie każdego z zadań na ścieżce (o ile takie występują). Wybierane są zadania z określonymi powyżej priorytetami (które są gotowe do wykonania w rozpatrywanej chwili czasowej) i następnie zadania te zostają alokowane na wybranych procesorach (zgodnie z algorytmem Mc'Naughton'a), tak aby (o ile to możliwe) nie występowały niewykorzystywane (będące bezczynnymi) procesory w danej jednostce czasu.

W pracy [15] przedstawiono zagadnienia szeregowania online jednoprocesorowych, niepodzielnych zadań, na maszynach równoległych, ze zbiorem zadań $T$. Przyjęto tam, że pojawiające się zadanie jest przypisywane do pierwszego wolnego (ang. *idle*) procesora, algorytm LIST. Wychodząc z tego modelu zaproponowany nowy algorytm onlinowy m-LIST dla przypadku dowolnego $a_i$. Alokowanie kolejnego zadania z listy następuje, jeśli $a_i$ procesorów jest wolnych. Dla $a_i = 1$ otrzymujemy algorytm LIST.

**Algorytm 3.** *Niech $C_k$, $k \in M$, będzie terminem, począwszy od którego procesor $k$ jest stale wolny (ang. idle). Rozpocznij od $C_k = 0$, $k \in M$. Z listy zadań oczekujących pobierz kolejne zadanie i nazwij je $T_i$. Wyznacz najwcześniejszy moment czasowy $S_i$ taki, że $a_i$ procesorów jest w stanie idle począwszy od $S_i$. Zaplanuj i zaktualizuj odpowiednie $C_k$. Powtórz proces dla kolejnych zadań z listy.*

## 7. Analiza – zadania niepodzielne

Poniższe Twierdzenie 1 jest rezultatem znanym z literatury. Algorytm LIST dla $a_i = 1$ wymienia się wśród najlepszych algorytmów klasy online-list [6, 15].

**Twierdzenie 1.** *Algorytm LIST dla problemu $P|online - list, a_i = 1|C_{\max}$ ma współczynnik konkurencyjności $r = 2 - \frac{1}{m}$.*

W celu sformułowaniu kolejnych własnych rezultatów dotyczacych algorytmu m-LIST, zdefiniujemy pewną aproksymację. Załóżmy, że posiadamy jedynie zadania wieloprocesorowe, dla których $a_i = const$ i jest takie samo dla wszystkich zadań. Wówczas możemy pogrupować procesory w sposób następujący: (A) dla zadań 2-procesorowych, grupowanie po 2 maszyny, (B) dla zadań 3-procesorowych, grupowanie po 3 maszyny, (C) dla zadań k-procesorowych, grupowanie po k-maszyn. Następnie stosujemy Algorytm 3 dla odpowiednio pogrupowanych procesorów i sztucznie zdefiniowanych zadań jednoprocesorowych. Wychodząc od poprzedniego twierdzenia, otrzymujemy:

**Twierdzenie 2.** *Algorytm m-LIST dla problemu $P|online-list, a_i = 2|C_{\max}$ ma współczynnik konkurencyjności*

$$r = \begin{cases} 2 - \frac{2}{m} & m \text{ parzyste} \\ \\ 2 - \frac{2}{m-1} & m \text{ nieparzyste} \end{cases} \tag{2}$$

*Dowód.* Ponieważ zadanie wykorzystuje procesory w sposób synchroniczny, istnieje co najwyżej $[m/2]$ par procesorów równocześnie dostępnych do realizacji zadania. Bez straty ogólności rozważań załóżmy, że sa to pary $(1, 2)$, $(2, 3)$, ...,$(m - 1, m)$ jeśli $m$ jest parzyste oraz $(1, 2)$, $(2, 3)$, ...,$(m - 2, m - 1)$ jeśli $m$ jest nieparzyste. Zauważmy, że w przypadku nieparzystego $m$ nie ma możliwości użycia procesora

$m$, ponieważ nie ma on pary. Dalej transformujemy problem $P|online-list, a_i = 2|C_{max}$ z danymi $n$, $m$, $(p_j,\ j = 1, 2, \ldots n)$ do pomocniczego problemu $P|online-list, a_i = 1|C_{max}$ z danymi $n'$, $m'$, $(p'_j,\ j = 1, 2, \ldots, n)$ w następujący sposób: $n' = n$, $m' = [m/2]$, $(p'_j = p_j,\ j = 1, 2, \ldots, n)$. Zgodnie z twierdzeniem 1 zachodzi

$$r = 2 - \frac{1}{m'}. \tag{3}$$

Ponieważ $m' = [m/2] = m/2$ dla $m$ parzystego oraz $m' = [m/2] = (m-1)/2$ dla $m$ nieparzystego, podstawiając $m'$ do (3) otrzymujemy (2). □

Korzystając z idei powyższego dowodu można udowodnić natychmiast przez analogię następujące twierdzenie (dowód jako oczywisty pominięto). Niech $[x]$ oznacza część całkowitą liczby $x$.

**Twierdzenie 3.** *Algorytm m-LIST dla problemu $P|online-list, a_i = k|C_{max}$ ma współczynnik konkurencyjności $r = 2 - \frac{1}{[\frac{m}{k}]}$.*

Twierdzenie 3 zastosowane dla $a_i = 2$ dostarcza wynik w pełni zgodny z twierdzeniem 2. Funkcja $r = 2 - \frac{1}{[\frac{m}{k}]}$ jest malejąca względem $k$, zaś wartość maksymalną osiąga dla $k = 1$. Dla przypadku zadań o wymieszanych $a_i$, $1 \leqslant a_i \leqslant k$, współczynnik $r$ powienien szacować każdą instancję $I$, w szczególności taką, że $a_i = 1$, $T_i \in T$. Stąd wynika, że współczynnik konkurencyjności dla m-LIST z dowolnymi różnymi $a_i$ jest $r = 2 - \frac{1}{m}$.

## 8. Analiza - zadania podzielne

Podobnej analizie poddano zadania podzielne i niezależne, $R = \emptyset$. Literaturowy przegląd optymalnych algorytmów online w pracy [5], szacuje współczynnik konkurencyjności dla najlepszego algorytmu dla problemu jednomaszynowego z przerwaniami.

**Twierdzenie 4.** *Algorytm 1 dla problemu szeregowania $P|pmnt, online-list, a_i = 1|C_{\max}$ ma współczynnik konkurencyjności $\frac{1}{1-(1-\frac{1}{m})^m}$.*

*Dowód.* Odwołać się można do znanego rezultatu literaturowego, omówionego między innymi w pracy [6], ponieważ został przedstawiony jako rozwiązanie, które przy liczbie maszyn dążącej do nieskoczonyści dąży do $\frac{e}{e-1} \approx 1,58$, zatem przy przedstawionej specyfikacji również wykazuje taką cechę. □

Przechodząc do szeregowanie zadań wieloprocesorowych otrzymujemy:

**Twierdzenie 5.** *Algorytm MC (Algorytm 1) dla problemu szeregowania $P|pmnt, online-list, a_i = 2|C_{\max}$ ma współczynnik konkurencyjności*

$$r = \frac{1}{1-(1-\frac{1}{[\frac{m}{2}]})^{[\frac{m}{2}]}}. \tag{4}$$

*Dowód.* Ponieważ zadanie wykorzystuje procesory w sposób synchroniczny, istnieje co najwyżej $[m/2]$ par procesorów równocześnie dostępnych do realizacji zadania. Bez straty ogólności rozważań załóżmy, że sa to pary $(1, 2)$, $(2, 3)$, ...,$(m-1, m)$

jeśli $m$ jest parzyste oraz $(1,2)$, $(2,3)$, ...,$(m-2,m-1)$ jeśli $m$ jest nieparzyste. Zauważmy, że w przypadku nieparzystego $m$ nie ma możliwości użycia procesora $m$, ponieważ nie ma on pary. Dalej transformujemy problem $P|pmnt, online - list, a_i = 2|C_{\max}$ z danymi $n$, $m$, $(p_j,\ j = 1,2,\ldots n)$ do pomocniczego problemu $P|pmnt, prec, online - list, a_i = 1|C_{\max}$ z danymi $n'$, $m'$, $(p'_j,\ j = 1,2,\ldots,n)$ w następujący sposób: $n' = n$, $m' = [m/2]$, $(p'_j = p_j,\ j = 1,2,\ldots,n)$. Zgodnie z twierdzeniem 4 zachodzi

$$r = \frac{1}{1-(1-\frac{1}{m'})^{m'}}. \tag{5}$$

Ponieważ $m' = [m/2] = m/2$ dla $m$ parzystego oraz $m' = [m/2] = (m-1)/2$ dla $m$ nieparzystego, podstawiając $m'$ do (5) otrzymujemy (4). □

Kolejne twierdzenie jest dość oczywiste, zatem nie wymaga dowodu.

**Twierdzenie 6.** *Algorytm MC (Algorytm 1) dla problemu szeregowania* $P|pmnt, online - list, a_i = k|C_{\max}$, *gdzie* $k \leqslant m$, *z kryterium* $C_{max}$ *ma współczynnik konkurencyjności*

$$r = \frac{1}{1-(1-\frac{1}{[\frac{m}{k}]})^{[\frac{m}{k}]}} \tag{6}$$

Przez analogię, algorytm MC (Algorytm 1) dla problemu $P|pmtn, online - list, a_i \leqslant k|C_{\max}$ ma współczynnik konkurencyjności taki jak w twierdzeniu 4.

## 9. Podsumowanie

Zaproponowane zostały algorytmy online dla szeregowania zadań wieloprocesorowych w przypadku nieprzerywalnym i przerywalnym, wraz z oceną ich współczynnika konkurencyjności. Te pesymistyczne oceny mają charakter teoretyczny, formułowany jako twierdzenia. Otrzymane wyniki nie wymagają przeprowadzenia eksperymentów numerycznych na losowej próbce instancji, ponieważ zarówno instancja pesymistyczna jak i rozwiązanie pesymistyczne występuje statystycznie rzadko w systemach rzeczywistych. Faktycznie rezultaty badań komputerowych zwykle dostarczą wyników uśrednionych po eksperymentalnej próbce instancji, zatem są bliższe analizie probabilistycznej w sensie średnim niż analizie konkurencyjności. Niemniej, zarówno analiza konkurencyjności jak i analiza eksperymentalna dostarczają uzupełniających się informacji o charakterystyce numerycznej algorytmów.

LITERATURA

1. Albers S.: Online Scheduling. Introduction to scheduling, CRC Press, 2009.
2. Behnamian J.: Survey on fuzzy shop scheduling, Fuzzy Optimization and Decision Making, Springer, 2016.
3. Błażewicz, J. et al.: Handbook on scheduling: from theory to applications, Springer Science & Business Media, 2007.

4. Błażewicz J. and Liu Z.: Scheduling multiprocessor tasks with chain constraints, European Journal of Operational Research, Elsevier, 1996.
5. Bożejko W., Gawlińska E.: Algorytmy szeregowania online. W: Bożejko W., Pempera J. (red.): Optymalizacja dyskretna w informatyce, automatyce i robotyce, Oficyna Wydawnicza Politechniki Wrocławskiej, Wrocław, 2012.
6. Chen B., Van Vliet A. and Woeginger G.J.: An optimal algorithm for preemptive on-line scheduling, Operations Research Letters 18, 1995.
7. Chen X.: Selected problems of online scheduling on parallel machines, Rozprawa doktorska, Politechnika Poznańska, Poznań, 2014.
8. Davis R. and Burns A.: A survey of hard real-time scheduling for multiprocessor systems, ACM Computing Surveys (CSUR), ACM New York, NY, USA, 2011.
9. Dorota D.: Dual-processor tasks scheduling using modified Muntz-Coffman algorithm. In: International Conference on Dependability and Complex Systems. Springer, Cham, 2018.
10. Dorota D.: Scheduling Tasks in a System with a Higher Level of Dependability. In: International Conference on Dependability and Complex Systems. Springer, Cham, 2019.
11. Dorota D.: Scheduling tasks with uncertain times of duration. In: International Conference on Dependability and Complex Systems. Springer, Cham, 2020.
12. Drozdowski M.: Scheduling for parallel processing. Springer, London, 2009.
13. Drozdowski M.: Scheduling multiprocessor tasks - an overview, European Journal of Operational Research, Elsevier, 1996.
14. Graham R.L., Lawler E.L., Lenstra J.K. and Rinnooy Kan A.H.G.: Optimization and approximation in deterministic sequencing and scheduling: a survey, Annals of Discrete Mathematics 5, 1979.
15. Graham R.: Bounds for certain multiprocessing anomalies, Bell System Technical Journal, Wiley Online Library, 1966.
16. Hurink J. and Paulus J.: Online algorithm for parallel job scheduling and strip packing, International Workshop on Approximation and Online Algorithms, Springer, 2007.
17. Im S.: Online scheduling algorithms for average flow time and its variants, University of Illinois at Urbana-Champaign, 2012.
18. Johannes B.: Scheduling parallel jobs to minimize the makespan, Journal of Scheduling, Springer, 2006.
19. Karp R. M.: On-Line Algorithms Versus Off-Line Algorithms: How Much, Algorithms, Software, Architecture: Information Processing 92: Proceedings of the IFIP 12th World Computer Congress, Madrid, Spain, 7-11 September 1992.
20. Klein, D. and Manning C. D.: A* Parsing: Fast Exact Viterbi Parse Selection, In Proceedings of the Human Language Technology Conference and the North American Association for Computational Linguistics, HLT-NAACL, 2003.

21. Makuchowski M.: Problemy gniazdowe z operacjami wielomaszynowymi : Własności i algorytmy. Rozprawa doktorska. Raporty Instytutu Cybernetyki Technicznej PWr. 2004, Ser. PRE; nr 37. 196 s.
22. McNaughton R.: Scheduling with deadlines and loss functions, Management Science, INFORMS, 1959.
23. Muntz R.R. and Cofmann E.G. Jr.: Preemptive scheduling of real-time tasks on multiprocessors systems, Journal of the ACM, 17(2), 1970.
24. Pinedo M. : Scheduling. Springer, New York, NY, 2015.
25. Pruhs, K., Jiri S. and Torng, E.: Online scheduling, 2004.
26. Smutnicki C. : Algorytmy szeregowania zadań. Oficyna Wydawnicza Politechniki Wrocławskiej, 2012.