Uniwersytet im. Adama Mickiewicza w Poznaniu
Wydział Matematyki i Informatyki

Podzielne szeregowanie zadań z pozycyjno-zależnymi czasami wykonywania na dwóch równoległych identycznych maszynach

Marcin Żurowski

Rozprawa doktorska z nauk matematycznych
w dyscyplinie informatyka
napisana pod kierunkiem
prof. UAM dra hab. Stanisława Gawiejnowicza

Poznań, 2019

Streszczenie

W rozprawie rozważamy podzielne szeregowanie zadań z pozycyjno-zależnymi czasami wykonywania na dwóch równoległych identycznych maszynach z długością uszeregowania jako kryterium optymalności. Przedstawiamy przegląd literatury na temat modeli podzielności zadań. Proponujemy nowy model podzielności, w którym tylko jedno pozycyjno-zależne zadanie może być przerywane. Przedstawiamy kilka własności badanego problemu. Prezentujemy dwa algorytmy dokładne i dwa algorytmy heurystyczne rozwiązujące badany problem. Omawiamy wyniki eksperymentów numerycznych, przeprowadzonych w celu porównania zaproponowanych algorytmów.

Abstract

In the thesis, we consider preemptive scheduling of jobs with position-dependent processing times on two parallel identical machines with the maximum completion time criterion. We present a review of the literature of the main models of job preemption. We propose a new model of position-dependent job preemption in which jobs may be preempted but only restricted preemption of a single job is allowed. We present several properties of this problem. We present two exact algorithms and two heuristic algorithm for its solution. We discuss the results of numerical experiments conducted to compare the proposed algorithms.

Spis treści

Rozdział 1

Wstęp

W tym rozdziale przedstawiamy tematykę rozprawy, jej cel i motywację do badań, układ pracy oraz stosowane konwencje.

1.1 Tematyka rozprawy

Niniejsza rozprawa dotyczy pewnych problemów teorii szeregowania zadań. Teoria ta jest działem informatyki teoretycznej, zajmującym się modelowaniem, analizą oraz rozwiązywaniem problemów następującego typu. Mamy dany pewien zbiór zadań, zbiór pewnych zasobów oraz zbiór maszyn, na których te zadania mogą być wykonane z użyciem wyżej wymienionych zasobów. Wykorzystując własności zadań i zasobów oraz uwzględniając nałożone na nie ograniczenia należy skonstruować algorytm znajdujący taki przydział tych zadań do maszyn, który minimalizuje zadane kryterium optymalności (Conway i inni (1967); Coffman jr (1980); Leung (2004); Pinedo (2016)).

Ze względu na to iż pojęcia *zadanie*, *zasób* oraz *maszyna* są rozumiane bardzo szeroko teoria szeregowania zadań ma liczne zastosowania praktyczne (Morton i Pentico (1993); Zweben i Fox (1994); Pinedo (2016)).

Problemy szeregowania zadań rozważa się zarówno w środowisku jednomaszynowym, jak i wielomaszynowym. Istnieje kilka rodzajów środowisk wielomaszynowych (Błażewicz i inni (1996); Grabowski i inni (2003); Kubiak i inni (2002)), jednym z nich jest środowisko maszyn równoległych. W tym przypadku dowolne zadanie można wykonać na dowolnej z maszyn, które mogą pracować równolegle. Jeśli pracują one z taką samą szybkością to mamy do czynienia z maszynami *identycznymi*, w przeciwnym przypadku - z maszynami *jednorodnymi* lub *dowolnymi*.

W teorii szeregowania zadań przyjmuje się zwykle, że czasy wykonywania zadań są stałe - najczęściej są one liczbami całkowitymi nieujemnymi. W ostatnich kilkunastu latach rozważa się w teorii szeregowania zadań problemy ze zmiennymi czasami, zależnymi od czasu rozpoczęcia wykonywania zadania (Gawiejnowicz (2008)), ilości przydzielonego zasobu (Błażewicz i inni (1996))

bądź numeru pozycji zadania w uszeregowaniu (Gawiejnowicz (1996); Agnetis i inni (2014); Azzouz i inni (2018). Te ostatnie nazywane są w literaturze *pozycyjno-zależnymi czasami wykonywania zadań* (ang. position-dependent job processing times).

Większość problemów rozważanych w teorii szeregowania zadań dotyczy szeregowania niepodzielnego, w którym raz rozpoczęte zadanie musi być wykonywane bez przerw do momentu jego zakończenia. Odrębną grupę problemów szeregowania zadań stanowią problemy szeregowania podzielnego, w którym dowolne zadanie może zostać przerwane w trakcie wykonywania, a następnie kontynuowane na tej samej bądź innej maszynie.

W teorii szeregowania zadań znanych jest wiele *kryteriów optymalności*, które są funkcjami czasów zakończenia zadań. Najczęściej spotykanym kryterium jest kryterium minimalizacji długości uszeregowania, $C_{\max}$. Kryterium to jest definiowane następująco:

$$C_{\max} = \max\{C_j\}, \tag{1.1}$$

gdzie C_j oznacza czas zakończenia j-tego zadania, a j przebiega zbiór indeksów zadań.

W rozprawie są rozważane problemy podzielnego szeregowania zadań z pozycyjno-zależnymi czasami wykonywania na dwu równoległych identycznych maszynach oraz z kryterium optymalności $C_{\max}$.

1.2 Cel i motywacja

Ogólnym celem rozprawy jest zdefiniowanie modelu podzielności dla zadań pozycyjno-zależnych i zbadanie jego własności. Cele szczegółowe to zbadanie podstawowych własności uszeregowań podzielnych dla proponowanego modelu podzielności oraz skonstruowanie dla niego algorytmów znajdujących optymalne oraz przybliżone uszeregowania dla badanego problemu.

Motywacja do podjęcia tematyki rozprawy jest zarówno praktyczna, jak i teoretyczna.

Po pierwsze, problemy szeregowania zadań ze zmiennymi czasami lepiej odpowiadają potrzebom praktyki niż problemy z czasami stałymi. Przykładowo, lekarz przyjmujący w gabinecie pacjentów, z każdym kolejnym pacjentem przeprowadza badania wolniej ze względu na zmęczenie. Stąd czas badania pacjenta, który możemy utożsamić z czasem wykonywania pewnego zadania, jest zmienny i się wydłuża. Z drugiej strony pracownik wykonujący pewne detale wraz z upływem czasu nabiera wprawy i wykonuje swoją prace szybciej. W tym przypadku czas wykonywania poszczególnych detali także jest zmienny, lecz się skraca. Ze względów praktycznych (na przykład ze względu na regulaminowe przerwy w pracy) pożądane jest rozszerzenie modeli wyżej wymienionych przypadków o możliwość przerwania wykonywanego zadania. To stanowi praktyczną motywację do badań.

Po drugie, modele szeregowania zadań ze zmiennymi czasami, ze względu na zastosowania, powinny dopuszczać pewien rodzaj podzielności, ponieważ ma ona zwykle korzystny wpływ na długość uszeregowania. W literaturze brak jednak

definicji podzielności zadań ze zmiennymi czasami wykonywania zależnymi od numerów pozycji zadań w uszeregowaniu. Stanowi to teoretyczną motywację do badań.

1.3 Układ rozprawy

Rozprawa składa się z ośmiu rozdziałów o następującej zawartości.

Rozdział 1 zawiera wstęp do rozprawy, omówienie jej tematyki, opis celów rozprawy oraz jej układu. Rozdział 2 zawiera podstawowe definicje teorii algorytmów i teorii szeregowania zadań oraz oznaczenia użyte w rozprawie. Rozdział 3 zawiera opis głównych modeli podzielności zadań ze stałymi czasami wykonywania i przegląd literatury związanej z tymi modelami. Rozdział 4 zawiera opis nowego modelu podzielności zadań pozycyjno-zależnych. Rozdział 5 zawiera formalne sformułowanie badanego problemu, sformułowania własności oraz ich dowody oraz kilka uwag dotyczących złożoności badanego problemu. Rozdział 6 zawiera opis dwóch algorytmów dokładnych, algorytmu wyliczeniowego i algorytmu podziału i ograniczeń, oraz wyniki eksperymentów obliczeniowych dla tych algorytmów. Rozdział 7 zawiera opis dwóch algorytmów heurystycznych oraz wyniki eksperymentów obliczeniowych dla tych algorytmów. Rozdział 8 zawiera podsumowanie uzyskanych wyników oraz wskazuje potencjalne dalsze kierunki badań.

Konwencje przyjęte w rozprawie W rozprawie stosowane będą następujące konwencje: Pierwsze użycie wprowadzonego pojęcia będzie oznaczone *kursywą*. W każdym rozdziale będą oddzielnie numerowane: definicje, twierdzenia, własności, przykłady, uwagi, algorytmy, rysunki, równania i tabele. Na przykład Twierdzenie 3.5 oznacza piąte twierdzenie w rozdziale trzecim. Dowody, będą rozpoczynały się słowem *Dowód* i kończyły znakiem □.

Podziękowania Składam serdeczne podziękowania mojemu promotorowi prof. UAM dr hab. Stanisławowi Gawiejnowiczowi, za wskazanie tematyki rozprawy, pomoc przy redagowaniu artykułów oraz czas poświęcony w trakcie pisania rozprawy.

Składam też podziękowania dr Cezaremu Suwalskiemu za pomoc przy algorytmie podziału i ograniczeń.

Dziękuje dr Joannie Berlińskiej, i mgr Bartłomiejowi Przybylskiemu za uwagi na temat badanego problemu, dzięki którym ustrzegłem się błędów i zaoszczędziłem wiele cennych godzin podczas implementacji algorytmów.

Na koniec dziękuje mojej żonie Agnieszce za wspieranie mnie podczas całych studiów doktoranckich.

Rozdział 2

Preliminaria

W tym rozdziale omówimy użyte w rozprawie definicje i oznaczenia, dotyczące teorii algorytmów i teorii szeregowania zadań, oraz przedstawimy wprowadzoną notację. Rozdział 2 składa się z trzech podrozdziałów. W podrozdziale 2.1 przedstawimy definicje i oznaczenia teorii algorytmów. W podrozdziale 2.2 przedstawimy definicje i oznaczenia teorii szeregowania zadań. Na koniec, w podrozdziale 2.3 przedstawimy oznaczenia wprowadzone w tej rozprawie

2.1 Teoria algorytmów

Problemem algorytmicznym (w skrócie: *problemem*) nazywamy zbiór parametrów oraz zdanie określające własności jakie powinno spełniać rozwiązanie tego problemu. Jednym z rodzajów problemów są *problemy optymalizacyjne*. Są to takie problemy dla których szukamy rozwiązania optymalizującego wartości *funkcji kryterialnej*.

Jeżeli nadamy parametrom problemu pewne wartości liczbowe, to mamy do czynienia z *instancją* tego problemu. *Rozmiarem instancji* nazywamy długość łańcucha kodującego tę instancje w pewnym *schemacie kodowania*, przy czym schemat ten nie może powodować wykładniczego wzrostu rozmiaru instancji.

Algorytm rozwiązujący pewien problem jest to pewna sekwencją kroków, która dla każdej instancji danego problemu zwraca pewne rozwiązanie w skończonej liczbie kroków. Dla danego problemu może istnieć wiele algorytmów. Możemy algorytmy te podzielić, ze względu na jakość rozwiązania:

- *Algorytmem dokładnym* nazywamy algorytm, który dla każdej instancji problemu generuje *rozwiązanie optymalne* (funkcja kryterialna osiąga ekstremum globalne);
- *Algorytmem heurystycznym* nazywamy algorytm, który dla pewnych instancji algorytm może generować rozwiązanie nieoptymalne i nie jest znana dla niego ocena najgorszego przypadku;

- *Algorytmem aproksymacyjnym* nazywamy algorytm, który dla pewnych instancji algorytm może generować rozwiązanie nieoptymalne, ale znana jest dla niego ocena najgorszego przypadku.

Algorytmy rozwiązujące dany problem możemy też podzielić ze względu na liczbę wykonanych operacji. Rząd wielkości liczby tych operacji jest opisany za pomocą notacji asymptotycznej Jednym z symboli tej notacji jest symbol O, którego definicję możemy znaleźć w (Cormen i inni (2017)).

Algorytmy rozwiązujące dany problem dzielimy na wielomianowe i wykładnicze:

- *Algorytmy wielomianowe* to algorytmy w których liczba operacji dominujących jest równa $O(p(k))$, gdzie p jest pewnym wielomianem, a k jest rozmiarem instancji;
- *Algorytmy wykładnicze* to algorytmy, które nie są wielomianowe

Istnieją problemy, które są nazywane problemami trudnymi obliczeniowo (ściśle: *NP-trudnymi*), dla których nie są znane wielomianowe algorytmy oraz dla których udowodniono, iż takie algorytmy dla nich nie istnieją o ile klasa P jest różna od klasy NP. Problem NP-trudny to taki problem, do którego można przekształcić (ściśle: zredukować) w wielomianowym czasie dowolny problem z klasy NP (por. Błażewicz (1988); Cormen i inni (2017))

2.2 Teoria szeregowania zadań

Zanim zdefiniujemy badany problem szeregowania zadań wprowadzimy następującą notację.

Zbiór maszyn oznaczymy $\mathcal{M} = \{M_1, M_2, \ldots, M_m\}$. W rozprawie rozpatrujemy problem dwumaszynowy, stąd $m = 2$, ale w rozdziale 3 jest przykład dla problemu, gdzie $m = 3$.

Zbiór zadań oznaczamy $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$. Z każdym zadaniem J_j powiązany jest *podstawowy czas wykonywania* zadania oznaczany przez p_j. W klasycznej teorii szeregowania zadań podstawowe czasy wykonywania zadań są opisane za pomocą liczb. Dla czasów pozycyjnie-zależnych będziemy używać następujących terminów. *Aktualny czas wykonywania* zadania J_j uszeregowanego bez podziału na r-tej pozycji w uszeregowaniu jest równy $p_{j,r} = p_j * r^a$, gdzie p_j to *podstawowy czas wykonywania* zadania J_j, r to pozycja tego zadania w uszeregowaniu, a $a < 0$ jest *indeksem uczenia się*.

Uszeregowanie (por. Gawiejnowicz (2008)) możemy zdefiniować w następujący sposób.

Definicja 2.1. *Uszeregowanie σ nazywamy przyporządkowanie zadań do maszyn w czasie, tak aby były spełnione następujące warunki:*

1. *w każdej chwili czasu wykonuje się co najwyżej jedno zadanie;*
2. *wszystkie zadania są wykonane;*

3. jeżeli istnieją ograniczenia związane z zadaniami, to są one spełnione.

Z każdym uszeregowaniem powiązany jest czas rozpoczęcia i czas zakończenia każdego zadania J_j oznaczone odpowiednio $S_j(\sigma)$ i $C_j(\sigma) = S_j(\sigma) + p_{k,r}$. Jeżeli będziemy rozważać kilka uszeregowań to będziemy je oznaczać σ, σ' i σ''. Jeśli będziemy rozpatrywać jedno uszeregowanie, to czas rozpoczęcia i czas zakończenia zadania J_j zapiszemy odpowiednio S_j i C_j, zamiast $S_j(\sigma)$ i $C_j(\sigma)$.

Zadaniem niepodzielnym nazwiemy zadanie, które po rozpoczęciu nie może zostać przerwane aż do zakończenia jego wykonywania. *Zadanie podzielne* to takie zadanie, które nie jest niepodzielne. *Uszeregowaniem bez przestoju* nazwiemy takie uszeregowanie w którym do czasu zakończenia wykonywania wszystkich zadań, nie ma chwili, w której któraś z maszyn nie przetwarza zadania. *Porządkiem SPT* nazwiemy takie przyporządkowanie zadań do maszyny, w którym zadania są wykonywane w porządku niemalejącym względem podstawowego czasu wykonywania zadania.

Kryterium optymalności nazwiemy pewną funkcję od wektora czasów zakończenia zadań. W niniejszej rozprawie, jako kryterium optymalności rozważamy długość uszeregowania (patrz 1.1).

Uszeregowaniem dopuszczalnym nazwiemy uszeregowanie, które spełnia zadane warunki wynikające ze specyfikacji problemu. *Uszeregowaniem optymalnym*, $\sigma^\star$, nazwiemy takie uszeregowanie dopuszczalne, które minimalizuje przyjęte kryterium optymalności. *Problemem szeregowania zadań* nazwiemy problem w którym dla zbioru zadań i maszyn należy znaleźć uszeregowanie optymalne. *Algorytmem szeregowania zadań* nazwiemy algorytm, który rozwiązuje problem szeregowania zadań.

2.3 Notacja

W rozprawie będziemy stosować następujące oznaczenia. Symbolem J^i oznaczamy sekwencję niepodzielnych zadań uszeregowanych na i-tej maszynie, $i = 1, 2$, w porządku SPT. Symbolem $|K|$ oznaczamy liczbę elementów w sekwencji K. Symbolem $J_{[j]}(K)$ oznaczamy j-te zadanie w sekwencji K. Symbolem $L_{[q]}(K) = \sum_{j=1}^{|K|} J_{[j]}(K)(j+q-1)^a$ oznaczamy sumę czasów wykonywania wszystkich zadań z sekwencji K uszeregowanych na maszynie od q-tej pozycji. Pozostałe oznaczenia będą definiowane w miejscu ich pierwszego wystąpienia.

Rozdział 3

Główne klasyczne modele podzielności

W teorii szeregowania zadań znanych jest kilka różnych modeli podzielności zadań o stałych czasach wykonywania. W tym rozdziale omówimy najważniejsze z nich. Rozdział 3 składa się z czterech podrozdziałów. W podrozdziale 3.1 przedstawimy model McNaughtona, który jako pierwszy pojawił się w literaturze. W podrozdziale 3.2 przedstawimy kilka modeli z narzuconymi pewnymi ograniczeniami. W podrozdziale 3.3 przedstawimy model w którym przerwanie zadania generuje pewien koszt. Na koniec w podrozdziale 3.4 przedstawimy jeszcze jeden model, nie należący do żadnej z poprzednio wymienionych grup modeli.

3.1 Klasyczny model podzielności

Pierwszy model podzielności zadań, który będziemy nazywać, *klasycznym modelem podzielności*, został wprowadzony w pracy McNaughton (1959). W tym modelu zadanie może zostać podzielone dowolną liczbę razy, przerwanie zadania nie generuje żadnego kosztu, ale to samo zadanie nie może być wykonywane na dwóch różnych maszynach w tym samym czasie. Model klasyczny jest powszechnie stosowany w wielu problemach podzielnego szeregowania ze stałymi czasami wykonywania zadań, począwszy od problemu szeregowania podzielnych zadań o stałych czasach na $m \geqslant 2$ równoległych identycznych maszynach z kryterium optymalności $C_{\max}$, który można rozwiązać algorytmem McNaughtona (patrz algorytm 3.1) w czasie $O(n)$, nie przerywając więcej niż $m-1$ zadań.

Pseudokod algorytmu McNaghtona można sformułować następująco (patrz algorytm 3.1).

Jednym z głównych wyników pracy McNaughton (1959) jest następujące twierdzenie.

Twierdzenie 3.1. *Algorytm 3.1 rozwiązuje problem podzielnego szeregowania n niezależnych zadań podzielnych w sensie McNaughtona na wielu równoległych*

Algorytm 3.1: Algorytm McNaughtona

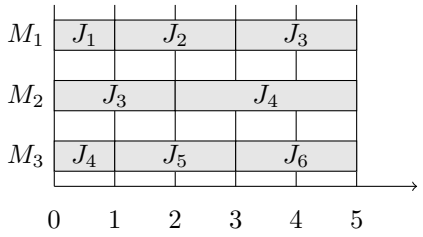
1 Wyliczamy optymalną długość uszeregowania $C^{\star}_{\max} = \max\{\max_{j=1}^{n}\{p_j\}, \frac{1}{m}\sum_{j=1}^{n} p_j\}$

2 Szeregujemy kolejno zadania na maszynie M_1. Jeżeli czas wykonania pewnego zadania na maszynie M_i przekroczy $C^{\star}_{\max}$ to przerywamy to zadanie w chwili $C^{\star}_{\max}$ i kontynuujemy na maszynie M_{i+1} od chwili 0.

identycznych maszynach w czasie $O(n)$.

Dowód powyższego twierdzenia możemy znaleźć w (McNaughton (1959)). Na koniec tego podrozdziału przedstawimy dwa przykłady pokazujące działanie algorytmu McNaughtona.

Przykład 3.1. Załóżmy, że mamy dane $n = 6$ zadań z podstawowymi czasami wykonywania $p_1 = 1$, $p_2 = 2$, $p_3 = 4$, $p_4 = 4$, $p_5 = 2$ i $p_6 = 2$. Jedno z 720 optymalnych uszeregowań przedstawia rysunek 3.1.



Rysunek 3.1: Uszeregowanie optymalne bez przestoju wygenerowane przez algorytm McNaughtona

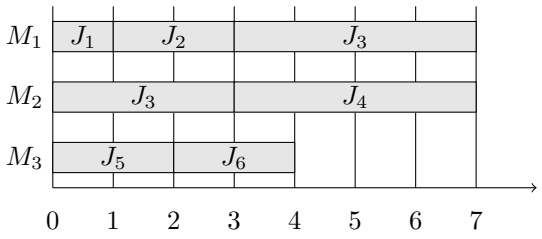
W tym przypadku wszystkie maszyny kończą prace w tym samym czasie, ponieważ $C^{\star}_{\max}$ jest równe średniemu obciążeniu maszyn.

Przykład 3.2. Załóżmy, że mamy dane $n = 6$ zadań z podstawowymi czasami wykonywania $p_1 = 1$, $p_2 = 2$, $p_3 = 7$, $p_4 = 4$, $p_5 = 2$ i $p_6 = 2$. Jedno z 720 optymalnych uszeregowań przedstawia rysunek 3.2.

W tym przypadku na ostatniej maszynie występuje przestój, ponieważ $C^{\star}_{\max}$ jest równe maksimum po wszystkich czasach wykonywania zadań.

3.2 Modele ograniczonej podzielności

Następne dwa modele podzielności zadań zaproponowane w pracy (Ecker i Hirschberg (1993)), to *ograniczona k podzielność* (ang. k-restricted preemption) i *dokładnie ograniczona k podzielność* (ang. exact-k-restricted preemption). W przypadku ograniczonej k podzielności zadanie może zostać podzielone dopiero



Rysunek 3.2: Uszeregowanie optymalne z przestojem wygenerowane przez algorytm McNaughtona

po wykonaniu k jednostek czasu. W konsekwencji możemy podzielić zadanie na kilka części, w których podstawowe czasy wykonywania wszystkich części z wyjątkiem ostatniego, nie są mniejsze niż k jednostek czasu. W przypadku dokładnie ograniczonej k podzielności, wszystkie części podzielonego zadania, z wyjątkiem ostatniej, są dokładnie równe k jednostek czasu.

Głównym wynikiem tej pracy był dowód, że dla równoległych identycznych maszyn dla $m = 2$ możliwe jest skonstruowanie optymalnego k-podzielnego uszeregowania w czasie liniowym, pod warunkiem, że zadania są niezależne, a $k \in \mathbb{N}$ jest stałe. Dla dowolnego k problem jest NP-trudny dla obu modeli podzielności.

Niektóre najnowsze wyniki dotyczące modelu ograniczonej k podzielności, czytelnik może znaleźć w pracach (Barański (2011); Pieńkosz i Prus (2015)).

Inny model podzielności zadań, nazwanym *ograniczoną podzielnością* (ang. limited preemption), został wprowadzony w pracy (Coffman jr i Even (1997)). W tym modelu przerwane zadanie nie może zostać przeniesione na inną maszynę.

Głównym wynikiem tej pracy był dowód, że stosunek maksymalnego czasu zakończenia uszeregowania bez podziału i maksymalnego czasu zakończenia uszeregowania z ograniczoną podzielnością jest ograniczony z góry przez $\frac{4}{3}$.

W oparciu o powyższe modele można konstruować nowe modele podzielności. Na przykład, ustalając $k = 1$ w modelu z ograniczoną k podzielnością, uzyskujemy model *podzielność w całkowitych momentach czasu* (ang. integer preemption) zaproponowany w pracy (Baptiste i inni (2012)).

Głównym wynikiem tej pracy było zastosowanie tego modelu do szeregowania zadań z ograniczeniami kolejnościowymi w postaci łańcucha, czasami gotowości i regularnym kryterium optymalności dla równoległych identycznych maszyn. Pokazano, że dla każdego problemu z klasyczną podzielnością można skonstruować uszeregowanie z podzielnością w całkowitych momentach czasu.

Niektórzy autorzy rozważali problemy szeregowania z ograniczeniami liczby podzielonych zadań do jednego (ang. single preemption). Na przykład Soper i Strusevich (2018a,b) rozważali szeregowanie zadań na równoległych, identycznych i jednolitych maszynach z podzielnością co najwyżej jednego zadania i maksymalnym czasem zakończenia jako kryterium optymalności.

Głównym wynikiem tej pracy był dowód, że w przypadku dwóch równoległych identycznych i jednolitych maszyn, problem z podzielnością co najwyżej jednego zadania można rozwiązać w czasie wielomianowym, podczas gdy dla trzech równoległych identycznych maszyn problem jest trudny obliczeniowo (NP-trudny). Autorzy przeprowadzili również parametryczną analizę jakości uszeregowań z podzielnością co najwyżej jednego zadania.

Niektóre najnowsze wyniki dotyczące modelu z ograniczeniami liczby podzielności, czytelnik może znaleźć w pracy (Jiang i inni (2012)).

3.3 Kosztowe modele podzielności

W przedstawionych do tej pory modelach podzielności zadań czas przetwarzania zadania przed podziałem jest równy sumie czasów przetwarzania części podzielonego zadania. W artykule Julien i inni (1997) autorzy wprowadzają dwa modele podzielności, w których każde przerwanie wydłuża czas przetwarzania zadania ponieważ dzielone zadanie potrzebuje czasu na rozruch przed wznowieniem. W pierwszym modelu nazywanym *podzielność z rozruchem* (ang. preemption setup model), czas rozpoczęcia każdej części zadania jest opóźniony o czas potrzebny na rozruch. W drugim modelu nazwanym *podzielność z wznowieniem* (ang. preemption startup model), tylko wznawiane zadanie jest opóźnione o czas potrzebny na rozruch.

Dla tych modeli, autorzy przedstawiają optymalny algorytm dla problemu jedno-maszynowego typu online oraz maksymalnego ważonego czasu przepływu i maksymalnego spóźnienia jako kryteriów optymalności. Liu i Cheng (2002) wprowadzili wariant modelu "podzielność z wznowieniem", nazywany *podzielnością z powtarzalnym wznowieniem* (ang. preemtion-repeat), w którym każde zadanie jest zdefiniowane przez czas gotowości, czas przetwarzania, czas dostarczenia i czas konfiguracji który jest potrzebny kiedykolwiek zadanie się rozpoczyna, czy to przy inicjalizacji czy po przerwaniu, pod warunkiem że czas rozruchu musi być całkowicie powtórzony.

Głównym wynikiem tej pracy był dowód, że problem z minimalizacją maksymalnego czasu dostawy jest trudny obliczeniowo (silnie NP-trudny) nawet jeśli czas wznowienia jest jednostkowy dla każdego zadania. Autorzy zaproponowali schemat aproksymacyjny dla tego problemu, to znaczy rodzinę algorytmów aproksymacyjnych o złożoności zależnej od rozmiaru danych wejściowych i wielkości błędu.

3.4 Inne modele podzielności

Głównym wynikiem w pracy Xing i Zhang (2000) był zaprezentowany wielomianowy algorytm dla wielo-maszynowego problemów ze specjalnym przypadkiem klasycznej podzielności, nazwanej *rozdzielna podzielność* (ang. splitting preemption), w którym części podzielonego zadania mogą być wykonywane na kilku maszynach jednocześnie. Jiang i inni (2013) rozważają wariant tego modelu,

w którym przed przetworzeniem zadania konieczna jest konfiguracja wykonana przez serwer, a zarówno praca, jak i konfiguracja zadania mogą zostać przerwane. Niektóre najnowsze wyniki dotyczące modelu rozdzielnej podzielności, czytelnik może znaleźć w Liu i inni (2018).

Rozdział 4

Nowy model podzielności

Podzielne szeregowanie zadań z pozycyjno-zależnym czasem wykonywania nie było rozważane wcześniej. W abstraktach Żurowski (2018a), Żurowski (2018b) i Żurowski i Gawiejnowicz (2018), przedstawiono sformułowania wstępnych wyników dla tego typu problemu.

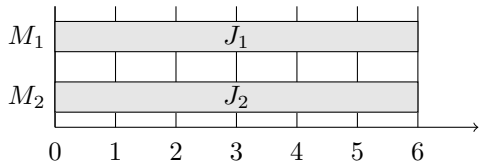
W tym rozdziale, przedstawimy definicję podzielności zadań pozycyjno-zależnych i zilustrujemy ją odpowiednimi przykładami. Rozdział 4 składa się z trzech podrozdziałów. W podrozdziale 4.1 przedstawimy na przykładzie wpływ podzielności zadań na uszeregowania z zadaniami pozycyjno-zależnymi. W podrozdziale 4.2 zaprezentujemy nową definicję podzielności oraz pokażemy jej działanie na przykładzie. Na koniec w podrozdziale 4.3 przedstawimy definicję współczynnika podziału zadania.

4.1 Konsekwencje podzielności

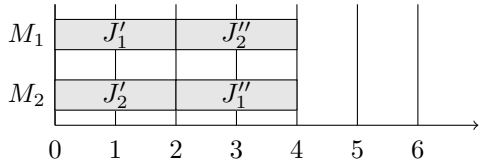
Zanim wprowadzimy definicję podzielności zadań pozycyjno-zależnych, zilustrujemy wpływ tego rodzaju podzielności na długość uszeregowania następującym przykładem.

Przykład 4.1. Rozważmy różne uszeregowania dla $n = 2$ zadań z podstawowym czasem wykonywania $p_1 = p_2 = 6$, przedstawione na rysunku 4.1. (Symbol P na rysunku 4.1b–4.1c oznacza czas w którym przerywamy zadanie.)

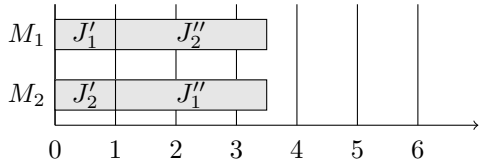
Ponieważ czasy wykonywania zadań zależnych od pozycji w uszeregowaniu są nierosnącymi funkcjami pozycji zadania w uszeregowaniu, suma czasu wykonywania obu części podzielonego zadania może nie być równa czasowi wykonywania zadania bez podziału (por. Żurowski (2018a)). Co więcej, sekwencja podzielności tego samego zadania wielokrotnie skraca jego czas wykonywania, w przeciwieństwie do szeregowania zadań ze stałymi czasami wykonywania, gdzie suma czasu wykonywania obu części podzielonego zadania jest równa czasowi wykonywania zadania bez podziału (por. McNaughton (1959)). Jest to przedstawione na rysunku 4.1d, gdzie symbol P_k oznacza czas, w którym zostało przerwane, a $k = 1, 2$ to liczba przerwań.



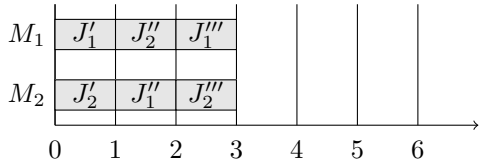
(a) Uszeregowanie bez podzielności



(b) Uszeregowanie z podzielnością w czasie $P = 2$



(c) Uszeregowanie z podzielnością w czasie $P = 1$



(d) Uszeregowanie z podzielnością w czasie $P_1 = 1$ i $P_2 = 2$

Rysunek 4.1: Podzielność zadań a długość uszeregowania

Ten przykład pokazuje, że podzielne szeregowanie zadań pozycyjno-zależnych ma inny charakter niż szeregowanie zadań o stałych czasach wykonywania.

4.2 Nowa definicja podzielności

W związku z przykładem 4.1 w pracy będzie stosowany następujący wariant definicji podzielności zadań wprowadzonej w pracy (Żurowski (2018a)).

Definicja 4.1. *Zadanie J_k rozpoczynające się w chwili S_k będzie nazywane* zadaniem podzielonym *jeżeli spełnione są następujące warunki:*

1. *zadanie J_k zostało przerwane w chwili P i było kontynuowane na innej maszynie w czasie T, gdzie $S_k < P < C_k$ i $T > P$;*
2. *części podzielonego zadania J_k to, odpowiednio, J'_k i J''_k;*

3. *zadanie J'_k rozpoczęło się w czasie $S'_k = T$ i zakończyło się w czasie C'_k, natomiast zadanie J''_k rozpoczęło się w chwili $S''_k = S_k$ i zakończyło się w chwili $C''_k = P$;*

4. *zadań J'_k i J''_k nie można wykonywać w tym samym czasie i nie mogą one być przerywane;*

5. *podstawowy czas wykonywania zadań J'_k i J''_k jest równy $p_k = p'_k + p''_k$;*

6. *aktualne czasy wykonywania zadań J'_k i J''_k są zdefiniowane funkcją podobnej postaci co aktualny czas wykonywania zadania J_k.*

W całej rozprawie, tak jak w pracy Żurowski (2018a), stosujemy definicję 4.1 zakładając, że tylko pierwsze zadanie uszeregowane na maszynie M_2 może zostać podzielone, i wznowione jako ostatnie zadanie na maszynie M_1.

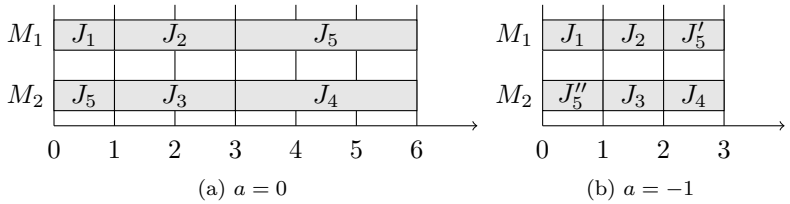
Uwaga 4.1. *Dla ujednolicenia zapisu zadanie J'_k będzie zawsze uszeregowane na maszynie M_1, natomiast zadanie J''_k będzie zawsze uszeregowane na maszynie M_2.*

Uwaga 4.2. *Jeżeli będziemy rozważać dwa uszeregowania σ i σ' w których to samo zadanie jest podzielone w innym czasie, to podstawowe czasy wykonywania zadań, będziemy oznaczać $p'_k(\sigma)$ i $p'_k(\sigma')$ oraz $p''_k(\sigma)$ i $p''_k(\sigma')$.*

Następny przykład dotyczy liczby dopuszczalnych uszeregowań w zależności od przyjętej definicji podzielności.

Przykład 4.2. Mamy dane $n = 5$ zadań z podstawowymi czasami wykonywania $p_1 = 1$, $p_2 = 2$, $p_3 = 2$, $p_4 = 3$ i $p_5 = 4$.

Na rysunku 4.2 przedstawiamy dwa podzielne uszeregowania dla powyższej instancji. Kiedy zadania mają stały czas wykonywania ($a = 0$), optymalne uszeregowanie możemy uzyskać używając algorytmu McNaughtona (McNaughton (1959)). Jedno ze stu dwudziestu optymalnych uszeregowań dla tej instancji przedstawia rysunek 4.2a.

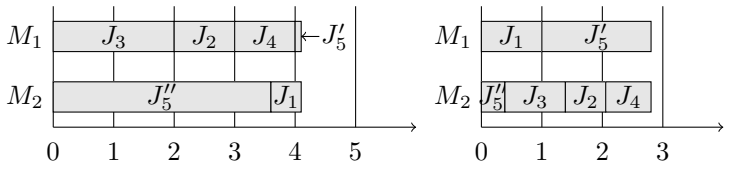


(a) $a = 0$ (b) $a = -1$

Rysunek 4.2: Klasyczna podzielność a podzielność pozycyjno-zależna

Jednakże, jeśli rzeczywisty czas wykonywania zadań jest pozycyjno-zależny, np. kiedy $a = -1$, to znalezienie podzielnego optymalnego uszeregowania staje się bardziej złożonym problemem. Przykładowe uszeregowanie σ dla tego przypadku jest przedstawione na rysunku 4.2b, gdzie $C_{\max}(\sigma) = 3$ ale uszeregowanie σ nie jest optymalne.

Główną przyczyną braku optymalności dla uszeregowania przedstawionego na rysunku 4.2b jest fakt, że równość całkowitych obciążeń obu maszyn w uszeregowaniu nie gwarantuje, że uszeregowanie jest optymalne, ponieważ może istnieć krótsze uszeregowanie z równymi obciążeniami obu maszyn. Ta sytuacja jest przedstawiona na rysunku 4.3. Rysunek 4.3a przedstawia uszeregowanie σ' z $C_{\max}(\sigma') = 4\frac{1}{10}$. Zauważmy, że maszyny M_1 i M_2 kończą przetwarzanie zadań w tym samym czasie. To uszeregowanie jest dłuższe niż uszeregowanie σ. Jednak, istnieje krótsze uszeregowanie, $\sigma^\star$, w którym zadania są w innej kolejności na maszynach M_1 i M_2 (patrz rysunek 4.3b). Obie maszyny kończą przetwarzanie zdań w tym samym czasie, ale uszeregowanie $\sigma^\star$ jest krótsze niż σ, tj. $C_{\max}(\sigma^\star) = 2\frac{29}{36} < C_{\max}(\sigma)$. Zwróćmy uwagę, że $\sigma^\star$ jest jedynym optymalnym uszeregowaniem dla tej instancji.



(a) Uszeregowanie σ' (b) Optymalne uszeregowanie $\sigma^\star$

Rysunek 4.3: Dwa różne uszeregowania z jednym podzielonym zadaniem

Przykład 4.2 pokazuje, że rozważany problem nie jest bezpośrednim uogólnieniem problemu szeregowania podzielnych zadań o stałych czasach na dwóch równoległych identycznych maszynach z kryterium optymalności $C_{\max}$.

4.3 Współczynnik podziału zadania

Ponieważ podział zadania J_k generuje dwa nowe zadania J'_k i J''_k takie, że $p'_k + p''_k = p_k$, należy zdefiniować wartości p'_k i p''_k. Aby to zrobić musimy wiedzieć ile jednostek zadania J_k zostało wykonane w czasie wykonywania zadań J'_k i J''_k. Dlatego wprowadzimy pojęcie *współczynnik podziału* zadania J_k (por. Żurowski (2018a)).

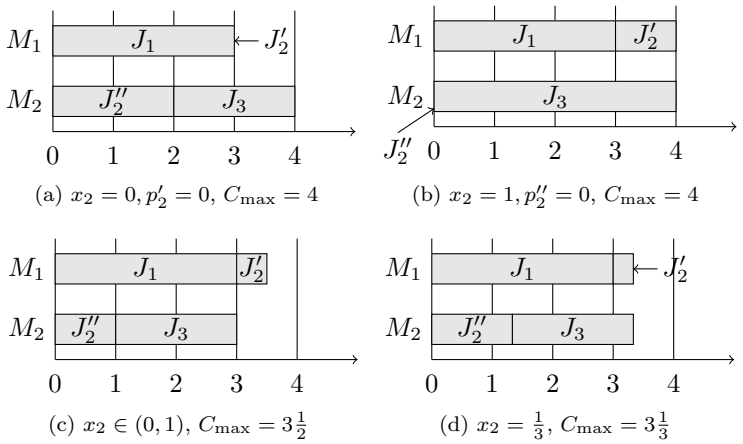
Definicja 4.2. *Niech będzie dane uszeregowanie σ w którym zadanie J_k będzie podzielone zgodnie z definicją 4.1. Wyrażenie $x_k(\sigma) = \frac{p'_k}{p_k}$ nazywamy współczynnikiem podziału zadania J_k.*

Zauważmy, że współczynnik podziału zadania może przyjmować wartości z zakresu $[0, 1]$.

Uwaga 4.3. *Kiedy będziemy rozważać to samo uszeregowanie, wtedy współczynnik podziału zadania będziemy oznaczać x_k zamiast $x_k(\sigma)$.*

Zilustruję definicję 4.2 następującym przykładem.

Przykład 4.3. Załóżmy, że mamy $n = 3$ zadań z podstawowymi czasami wykonywania $p_1 = 3$, $p_2 = 2$, $p_3 = 4$ i z podzielonym zadaniem J_2 w chwili $0 \leqslant C_2'' \leqslant p_2$. Różne wartości współczynnika podziału zadania x_2 prowadzą do różnych typów uszeregowań, jak pokazuje rysunek 4.4. Rysunek 4.4d przedstawia najkrótsze uszeregowanie dla danej sekwencji zadań z podzielonym zadaniem J_2.



(a) $x_2 = 0, p_2' = 0$, $C_{\max} = 4$ (b) $x_2 = 1, p_2'' = 0$, $C_{\max} = 4$

(c) $x_2 \in (0,1)$, $C_{\max} = 3\frac{1}{2}$ (d) $x_2 = \frac{1}{3}$, $C_{\max} = 3\frac{1}{3}$

Rysunek 4.4: Przykład współczynnika podziału zadania

Przykład 4.3 pokazuje, że wartości graniczne współczynnika podziału zadania x_2 wskazują dwa specjalne przypadki. Na przykład, jeżeli $x_2 = 0$, (patrz rysunek 4.4a) wtedy podstawowy czas wykonywania zadania J_2' równa się $p_2' = 0$, natomiast jeśli $x_2 = 1$, (patrz rysunek 4.4b) to $p_2' = p_2$.

Uwaga 4.4. *Dla znanej wartości współczynnika podziału zadania x_k czasy wykonywania zadań J_k' i J_k'' są równe odpowiednio*

$$p_k' = x_k p_k s^a$$

$$p_k'' = (1 - x_k) p_k.$$

W związku z przykładem 4.3 ograniczam moje dalsze rozważania tylko do uszeregowań *bez przestoju*, ponieważ przez wstawienie czasu przestoju zwiększamy czasy zakończenia zadań uszeregowanych po tym czasie, co może zwiększyć wartość $C_{\max}$.

Rozdział 5

Problem PSLE

W rozdziale tym przedstawimy formalną definicję badanego problemu oraz jego własności. Rozdział 5 składa się z trzech podrozdziałów. W podrozdziale 5.1 sformułujemy badany problem. W podrozdziale 5.2 przedstawimy jego własności. Na koniec, w podrozdziale 5.3 powiemy kilka słów o złożoności tego problemu.

5.1 Sformułowanie problemu

Problem, który rozważamy w rozprawie można sformułować jako następujący problem szeregowania zadań pozycyjno-zależnych.

Mamy dane $n \geqslant 3$ zadań $J_1, J_2, \ldots, J_n$ i dwie równoległe identyczne maszyny M_1, M_2, dostępne od chwili 0. Szeregujemy zadania na obu maszynach bez przestoju. Wszystkie zadania są niezależne, a czas wykonywania każdego zadania zależy od pozycji zadania w uszeregowaniu. Dokładniej, *aktualny czas wykonywania* zadania J_j uszeregowanego bez podziału na r-tej pozycji w uszeregowaniu jest równy $p_{j,r} = p_j * r^a$, gdzie p_j to *podstawowy czas wykonywania* zadania J_j, r to pozycja tego zadania w uszeregowaniu, a $a < 0$ jest *indeksem uczenia się*. Wybraliśmy taką formę efektu uczenia się ponieważ jest to najpopularniejsza forma w literaturze (patrz np. ostatnia praca przeglądowa autorstwa Azzouz i inni (2018)).

Czas wykonywania podzielonego zadania J_j jest zdefiniowany podobnie, pod warunkiem, że podstawowy czas wykonywania zadania J_j jest sumą podstawowych czasów wykonywania dwóch części zadania: jednej ukończonej przed przerwaniem i drugiej po przerwaniu.

Obie części podzielonego zadania są traktowane jak dwa nowe niezależne i niepodzielne zadania, tj. nie mogą być podzielone i nie mogą być wykonywane na obu maszynach jednocześnie lub na tej samej maszynie w różnych przedziałach czasu.

Kryterium optymalności uszeregowania jest długość uszeregowania $C_{\max}$.

W całej rozprawie nazywam ten problem *PSLE* (ang. *Preemptive Scheduling with a Learning Effect*).

5.2 Własności

W tym podrozdziale, przedstawimy podstawowe własności problemu PSLE. Podrozdział ten składa się z czterech sekcji. W sekcji 5.2.1 przedstawimy własności dotyczące współczynnika podziału zadania. W sekcji 5.2.2 podamy definicję zadania dominującego oraz opiszemy pewne konsekwencje jego istnienia. W sekcji 5.2.3 pokażemy własności dotyczące uszeregowań optymalnych. Na koniec, w sekcji 5.2.4 przedstawimy własności związane z transformacjami uszeregowań.

5.2.1 Własności współczynnika podziału zadania

Problem PSLE dotyczy podzielnego szeregowania, czyli takiego w którym chociaż jedno zadanie zostało podzielone. Ponieważ do podzielności zadania dochodzi wtedy, kiedy jego dwie części mają niezerowy czas wykonywania, to należało by formalnie pokazać jak wartość współczynnika podziału zadania, ma się do tego faktu. Pierwsza z właściwości pokazuje jaką wartość może mieć współczynnik podziału zadania, kiedy zadanie jest podzielone.

Własność 5.1. *Zadania J'_k i J''_k w uszeregowaniu σ mają niezerowy czas wykonania zadania wtedy i tylko wtedy gdy współczynnik podziału $x_k(\sigma)$ zadania J_k należy do przedziału $(0, 1)$.*

Dowód. ($\Rightarrow$) Rozważmy dowolne podzielne uszeregowanie dla problemu PSLE. Niech zadanie J_k będzie podzielone w czasie t_k na dwa nowe zadania J'_k i J''_k, z odpowiednio niezerowym czasem wykonywania p'_k i p''_k. Ponieważ $p_k > 0$ i $p'_k > 0$, to $\frac{p'_k}{p_k} = x_k(\sigma) > 0$. Ponieważ $p'_k > 0$ i $p''_k > 0$, to na mocy definicji 4.1 $p'_k < p_k$. A stąd wynika, że $x_k(\sigma) = \frac{p'_k}{p_k} < 1$.

($\Leftarrow$) Rozważmy dowolne podzielne uszeregowanie dla problemu PSLE i niech współczynnik podziału zadania $x_k(\sigma) \in (0, 1)$. Ponieważ $x_k(\sigma) = \frac{p'_k}{p_k} > 0$ i $p_k > 0$, to $p'_k > 0$. Podobnie, ponieważ $x_k(\sigma) < 1$ i $p_k > 0$, to $p'_k < p_k$, i $p_k - p'_k = p''_k > 0$. □

Powyższa własność została zilustrowana przykładem 4.3.

Ponieważ w badanym problemie zawsze przerywamy zadanie J_k uszeregowane na pierwszej pozycji na maszynie M_2, to moment w którym to robimy ma wpływ na długość zadania J'_k uszeregowanego jako ostatnie na maszynie M_1. Co więcej zmieniając ten moment zmieniamy czas wykonania zadania J'_k. Kolejna własność pokazuje zależność między czasem przerwania zadania J_k a czasami zakończenia zadań J'_k i J''_k.

Własność 5.2. *Niech będzie dane uszeregowanie σ w którym zadania J'_k i J''_k mają niezerowy czas wykonania zadania i s jest pozycją zadania J'_k. Wtedy jeżeli przekształcimy uszeregowanie σ na σ' w taki sposób, że czas zakończenia zadania J''_k zmieni się następująco $C''_k(\sigma') = C''_k(\sigma) + t$, gdzie $t \in (-C''_k, p_k - C''_k)$, to nowy czas zakończenia zadania J'_k będzie równy $C'_k(\sigma') = C'_k(\sigma) - ts^a$.*

Dowód. Aby udowodnić powyższą własność musimy pokazać jaka jest wartość współczynnik podziału zadania $x_k(\sigma')$ po przekształceniu, a następnie za jego pomocą pokazać jaka jest wartość $C'_k(\sigma')$ czasu zakończenia zadania J'_k.

Rozważmy następujące fakty:

- $x_k(\sigma') = \frac{p'_k(\sigma')}{p_k}$ (patrz definicja 4.2)
- $p'_k = p_k - p''_k$ (patrz definicja 4.1)
- ponieważ zadanie J''_k jest uszeregowane na pierwszej pozycji na maszynie M_2 to $p''_k(\sigma') = C''_k(\sigma')$

Korzystając z tych faktów mamy

$$x_k(\sigma') = \frac{p_k - C''_k(\sigma')}{p_k}.$$

Następnie korzystając z założenia, że $C'''_k(\sigma') = C'''_k(\sigma) + t$, $C'''_k(\sigma) = p''_k(\sigma)$ oraz z uwagi 4.4 mamy

$$x_k(\sigma') = \frac{p_k - ((1 - x_k(\sigma))p_k + t)}{p_k} = \frac{p_k - p_k + x_k(\sigma)p_k - t}{p_k}.$$

Przekształcając powyższe równanie otrzymujemy

$$x_k(\sigma') = x_k(\sigma) - \frac{t}{p_k}. \tag{5.1}$$

Teraz pokażemy wartość $C'_k(\sigma')$ czasu zakończenia zadania J'_k. Zauważmy, że zadanie J'_k rozpoczyna się w tym samym czasie w obu uszeregowaniach σ i σ', czyli $S'_k(\sigma) = S'_k(\sigma')$. Korzystając z uwagi 4.4 i faktu, że $C'_k(\sigma') = S'_k(\sigma') + p'_k(\sigma')$ mamy

$$C'_k(\sigma') = S'_k(\sigma) + x_k(\sigma')p_k s^a.$$

Podstawiamy do powyższego równania współczynnik podziału zadania (5.1) otrzymamy

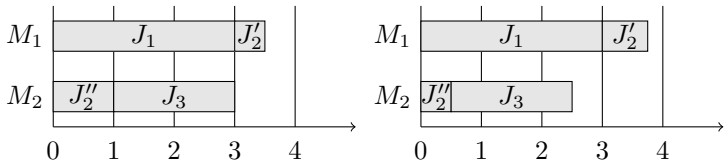
$$C'_k(\sigma') = S'_k(\sigma) + (x_k(\sigma) - \frac{t}{p_k})p_k s^a = S'_k(\sigma) + x_k(\sigma)p_k s^a - ts^a.$$

Ponieważ $S'_k(\sigma) + x_k(\sigma)p_k s^a = C'_k(\sigma)$ to

$$C'_k(\sigma') = C'_k(\sigma) - ts^a$$

co kończy dowód. □

Własność 5.2 zilustrujemy następującym przykładem pokazującym wpływ współczynnika podziału zadnia na długość uszeregowania.



(a) Uszeregowanie σ (b) Uszeregowanie σ'

Rysunek 5.1: Wpływ zmiany momentu podziału zadania na uszeregowanie

Przykład 5.1. Weźmy $n = 3$ zadania o podstawowych czasach wykonywania równych $p_1 = 3$, $p_2 = 2$, $p_3 = 4$ oraz indeksie uczenia się $a = -1$. Zadanie J_1 przydzielimy do pierwszej maszyny, zadanie J_3 do drugiej maszyny, a zadanie J_2 podzielimy w chwili 1 otrzymując uszeregowanie σ (patrz rysunek 5.1a), wtedy zadanie J_2' będzie wykonywane przez $\frac{1}{2}$ jednostki czasu i zakończy się w chwili $C_2' = 3\frac{1}{2}$. Zauważmy, że kiedy podzielimy zadanie J_2 w chwili $\frac{1}{2}$ otrzymując uszeregowanie σ' (patrz rysunek 5.1b), to zadanie J_2' będzie wykonywane przez $\frac{3}{4}$ jednostki czasu i zakończy się w chwili $C_2' = 3\frac{3}{4}$.

5.2.2 Własności zadania dominującego

Kolejne pojęcie, jakie wprowadzamy, wiąże się z faktem, że niektóre zadania mogą mieć tak duży podstawowe czasy wykonywania, że wykonanie ich będzie trwało dłużej niż wykonanie pozostałych zadań, jak ma to miejsce w Algorytmie McNaughtona, kiedy $C^{\star}_{\max} = \max_j\{p_j\}$ (patrz algorytm 3.1 i przykład 3.2). Każde takie zadanie będzie nazywane *zadaniem dominującym.*

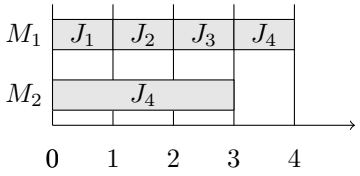
Definicja 5.1. *Niech $\bar{J} = J \setminus \{J_d\}$, $J^1 \subseteq \bar{J}$, $J^2 = \bar{J} \setminus J^1$ i $\bar{J^1} = J^1 \setminus \{J_k\}$, gdzie J_k jest podzielonym zadaniem. To zadanie J_d nazwiemy* zadaniem dominującym, *jeśli $d = \arg\max_i\{p_i\}$ i dla wszystkich podzbiorów zbioru zadań $\bar{J}$ spełniony jest następujący warunek*

$$\max_k\{L_{[1]}(\bar{J^1}) + p_k(|\bar{J^1}| + 1)^a\} \leqslant L_{[2]}(J^2) + p_d(|J^2| + 2)^a. \tag{5.2}$$

Każde zadanie, które nie spełnia definicji 5.1, nazwiemy *zadaniem nie-dominującym.* Uszeregowanie z zadaniem dominującym (bez zadania dominującego) będziemy nazywać *uszeregowaniem zdominowanym (niezdominowanym).* Zilustrujemy definicję 5.1 następującymi przykładami.

Pierwszy przykład przedstawia zadanie dominujące w uszeregowaniu zadań o stałych czasach wykonywania.

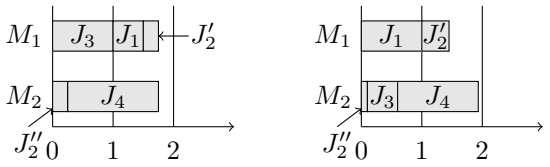
Przykład 5.2. Weźmy $n = 4$ zadania o podstawowych czasach wykonywania równych $p_1 = 1$, $p_2 = 1$, $p_3 = 1$ i $p_4 = 4$. Jedno z dwudziestu czterech optymalnych uszeregowań zostało przedstawione na rysunku 5.2. Zauważmy, że podstawowy czas wykonywania zadania dominującego J_4 jest równy długości uszeregowania $C_{\max} = p_4$, niezależnie od tego czy zadanie dominujące będzie przerwane czy nie.



Rysunek 5.2: Zadanie dominujące w klasycznym przypadku

Drugi przykład przedstawia zadanie dominujące dla uszeregowania zadań pozycyjno-zależnych.

Przykład 5.3. Mamy dane $n = 4$ zadań z podstawowymi czasami wykonywania $p_1 = p_2 = p_3 = 1$ i $p_4 = 3$, i niech indeks uczenia się będzie równy $a = -1$. Zadanie J_4 jest zadaniem nie-dominującym (patrz rysunek 5.3a). Jednakże, jeśli zwiększymy p_4 do 4, to zadanie J_4 będzie zadaniem dominującym (patrz rysunek 5.3b).



(a) Uszeregowanie bez zadania dominującego

(b) Uszeregowanie z zadaniem dominującym J_4

Rysunek 5.3: Uszeregowanie zdominowane a uszeregowanie niezdominowane

Kolejna własność mówi o wpływie istnienia zadania dominującego na uszeregowanie, ale zanim je przedstawimy musimy wprowadzić następującą definicję.

Definicja 5.2. *Całkowite obciążenie $TL^i(\sigma)$ maszyny M_i w uszeregowaniu σ jest równe sumie rzeczywistych czasów przetwarzania wszystkich zadań uszeregowanych na tej maszynie.*

Własność 5.3. *Jeżeli w instancji istnieje zadanie dominujące, to nie istnieje uszeregowanie optymalne dla tej instancji.*

Dowód. Załóżmy, że tak nie jest, i mamy uszeregowanie optymalne $\sigma^\star$ dla instancji w której istnieje zadanie dominujące J_d. Zgodnie z nierównością (5.2) $TL^1(\sigma^\star) < TL^2(\sigma^\star)$ (w przypadku kiedy w nierówności (5.2) lewa strona jest równa prawej, to aby zadania na drugiej maszynie zaczynały się od drugiej pozycji część zadania J_k musi zostać przeniesiona na pierwszą pozycję na maszynę M_2, co spowoduje nierówność obciążeń obu maszyn). To oznacza, że

$$C_{\max}(\sigma^\star) = TL^2(\sigma^\star) = (1 - x_k(\sigma^\star))p_k + L_{[2]}(J^2 \setminus \{J_d\}) + p_d(|J^2| + 2)^a. \quad (5.3)$$

Ponieważ zadanie J_k jest podzielone na dwie niezerowe części to zgodnie z własnością 5.1, $x_k(\sigma^\star) \in (0, 1)$. Skonstruujmy uszeregowanie σ w taki sposób, że

w uszeregowaniu $\sigma^\star$ zastąpimy współczynnik podziału $x_k(\sigma^\star)$ współczynnikiem $x_k(\sigma)$ takim, że $x_k(\sigma^\star) < x_k(\sigma) < 1$.
Ponieważ $x_k(\sigma) < 1$ więc $TL^1(\sigma) < TL^2(\sigma)$, czyli

$$C_{\max}(\sigma) = TL^2(\sigma) = (1 - x_k(\sigma))p_k + L_{[2]}(J^2 \setminus \{J_d\}) + p_d(|J^2| + 2)^a. \quad (5.4)$$

Odejmując (5.4) od (5.3) otrzymujemy

$$\begin{aligned} C_{\max}(\sigma^\star) - C_{\max}(\sigma) &= (1 - x_k(\sigma^\star))p_k - (1 - x_k(\sigma))p_k \\ &= ((1 - x_k(\sigma^\star)) - (1 - x_k(\sigma)))p_k. \end{aligned}$$

Ponieważ $x_k(\sigma^\star) < x_k(\sigma)$ to $(1 - x_k(\sigma^\star)) - (1 - x_k(\sigma)) = x_k(\sigma) - x_k(\sigma^\star) > 0$, co oznacza, że $C_{\max}(\sigma^\star) > C_{\max}(\sigma)$, co przeczy temu, że uszeregowanie $\sigma^\star$ jest optymalne. □

Idee powyższego dowodu przedstawimy na przykładzie 5.3. Na rysunku 5.3b mamy uszeregowanie z zadaniem dominującym w którym czas wykonywania zadania J_k'' jest równy p_k'', a $C_{\max} = p_k'' + L_{[2]}(J^2)$. Ponieważ dla dowolnego $p_k'' > 0$ istnieje liczba dodatnia mniejsza od p_k'', więc zgodnie z własnością 5.2 zawsze możemy skonstruować krótsze uszeregowanie, dlatego nie możemy wyznaczyć uszeregowania optymalnego.

Ponieważ na mocy własności 5.3 nie istnieje uszeregowanie optymalne dla instancji z zadaniem dominującym to w dalszych rozważaniach rozpatrujemy instancje problemu PSLE bez zadań dominujących.

5.2.3 Własności uszeregowania optymalnego

Teraz przedstawimy kilka właściwości, które charakteryzują optymalne uszeregowanie dla problemu PSLE.

Własność 5.4 (Pierwszy warunek konieczny uszeregowania optymalnego)**.** *Jeżeli uszeregowanie σ jest optymalne, to pewne zdanie musi być podzielone.*

Dowód. Niech będzie dane optymalne uszeregowanie $\sigma^\star$ takie, że $TL^1(\sigma^\star) \geqslant TL^2(\sigma^\star)$ i żadne zadanie nie zostało przerwane, gdzie

$$\begin{aligned} TL^1(\sigma^\star) &= L_{[1]}(J^1 \setminus \{J_k\}) + p_k s^a \\ TL^2(\sigma^\star) &= L_{[1]}(J^2) \end{aligned}$$

i J_k jest ostatnim zadaniem uszeregowanym na maszynie M_1 na s-tej pozycji. (Jeżeli $TL^1(\sigma^\star) \leqslant TL^2(\sigma^\star)$, to należy przenumerować maszyny M_1 i M_2.)

Przenieśmy zadanie J_k na pierwszą pozycję na maszynę M_2 i przerwijmy je w chwili

$$0 < t < \min\{p_k, L_{[1]}(J^2) - L_{[2]}(J^2)\}.$$

Przez σ oznaczmy nowe uszeregowanie w którym zadanie J_k będzie przerwane w powyższy sposób. Pokażemy że $C_{\max}(\sigma) < C_{\max}(\sigma^\star)$.

Rozważmy obciążenie maszyny M_1. Zauważmy,

$$TL^1(\sigma) = L_{[1]}(J^1 \setminus \{J_k\}) + x_k(\sigma)p_k s^a$$
$$TL^1(\sigma^\star) = L_{[1]}(J^1 \setminus \{J_k\}) + p_k s^a,$$

a ponieważ $x_k(\sigma) \in (0,1)$ zgodnie z własnością 5.1, więc

$$TL^1(\sigma) < TL^1(\sigma^\star).$$

Rozważmy obciążenie maszyny M_2. Ponieważ $r^a > (r+1)^a$ dla $a < 0$, to $L_{[1]}(J^2) > L_{[2]}(J^2)$. Zatem

$$TL^2(\sigma) = L_{[2]}(J^2) + t$$
$$TL^2(\sigma^\star) = L_{[1]}(J^2)$$

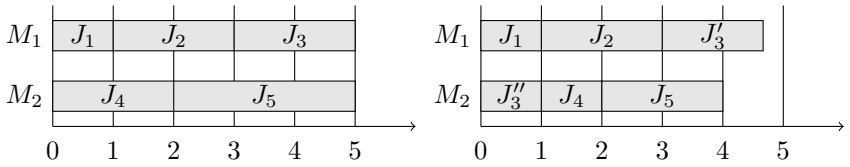
a ponieważ $t < L_{[1]}(J^2) - L_{[2]}(J^2)$, to

$$TL^2(\sigma) < TL^2(\sigma^\star).$$

Ponieważ obciążenia maszyn w uszeregowaniu σ są mniejsze od obciążeń maszyn w uszeregowaniu $\sigma^\star$, to $C_{\max}(\sigma) < C_{\max}(\sigma^\star)$. Sprzeczność. □

Powyższą własność zilustrujemy następującym przykładem.

Przykład 5.4. Mamy dane $n = 5$ zadań z podstawowymi czasami wykonywania $p_1 = 1$, $p_2 = 4$, $p_3 = 6$, $p_4 = 2$ i $p_5 = 6$, i indeks uczenia się $a = -1$. Na



(a) Uszeregowanie bez podziału $C_{\max} = 5$

(b) Uszeregowanie z podziałem $C_{\max} = 4\frac{2}{3}$

Rysunek 5.4: Uszeregowanie bez podziału, a uszeregowanie z podzielonym zadaniem

rysunku 5.4a mamy zadania uszeregowane bez podziału w taki sposób, aby obciążenia obu maszyn były sobie równe, $C_{\max} = 5$. Jednakże, jeśli przeniesiemy zadanie J_3 na pierwszą pozycję maszyny M_2 i przerwiemy je w chwili 1 (patrz rysunek 5.4b), to otrzymamy krótsze uszeregowanie w którym $C_{\max} = 4\frac{2}{3}$.

Skoro wiemy, że w uszeregowaniu optymalnym musi być podzielone jedno zadanie, więc manipulując wartością jego współczynnika podziału zadania moglibyśmy z danego uszeregowania spróbować skonstruować krótsze uszeregowanie, o czym mówi następująca własność.

Własność 5.5. *Niech będzie dane uszeregowanie σ z $C_{max}(\sigma) = \max\{TL^1(\sigma), TL^2(\sigma)\}$. Jeśli $TL^1(\sigma) \neq TL^2(\sigma)$, to istnieje uszeregowanie σ' takie, że $C_{max}(\sigma') < C_{max}(\sigma)$.*

Dowód. Niech σ będzie danym uszeregowaniem z $C_{\max}(\sigma) = \max\{TL^1(\sigma), TL^2(\sigma)\}$ i takim że $TL^1(\sigma) \neq TL^2(\sigma)$.
Rozważmy dwa przypadki: przypadek 1, kiedy w uszeregowaniu σ nie ma podzielonego zadania, i przypadek 2, kiedy w uszeregowaniu σ jest jedno podzielone zadanie.

W przypadku 1, możemy skonstruować z uszeregowania σ inne uszeregowanie, σ', z jednym podzielonym zadaniem. Wtedy, zgodnie z własnością 5.4, mamy $C_{\max}(\sigma') < C_{\max}(\sigma)$.

Rozważmy przypadek 2. Niech σ będzie uszeregowaniem, w którym $TL^1(\sigma) > TL^2(\sigma)$ i niech zadanie J_k będzie podzielone w taki sposób, że J'_k jest uszeregowane na ostatniej pozycji na maszynie M_1, podczas gdy zadanie J''_k jest uszeregowane na pierwszej pozycji na maszynie M_2. Wtedy, zadanie J''_k rozpoczyna się w chwili 0 i kończy się w chwili $C''_k(\sigma)$.

Skonstruujemy nowe uszeregowanie, σ', w którym zadanie J_k jest podzielone w czasie $C''_k(\sigma) < t \leqslant b$, gdzie $b = \frac{TL^1(\sigma) - TL^2(\sigma)}{s^a + 1}$ ($TL^1 + b \leqslant TL^2 - bs^a$) a s jest numerem pozycji zadania J'_k. Na mocy własności 5.2, otrzymujemy $TL^2(\sigma') > TL^2(\sigma)$, ale ponieważ $t \leqslant b$, to $TL^2(\sigma') \leqslant TL^1(\sigma')$. Konstruując uszeregowania σ' z uszeregowania σ skracamy podstawowy czas wykonywania p'_k zadania J'_k. Stąd $TL^1(\sigma') < TL^1(\sigma)$. Ponieważ $TL^2(\sigma') \leqslant TL^1(\sigma')$ i $TL^1(\sigma') < TL^1(\sigma)$, to $C_{\max}(\sigma) > C_{\max}(\sigma')$. Co należało pokazać.

Przypadek kiedy $TL^1(\sigma) < TL^2(\sigma)$ możemy udowodnić w podobny sposób. □

Z własność 5.5 wynika następująca własność, która jest warunkiem koniecznym optymalności uszeregowania dla problemu PSLE.

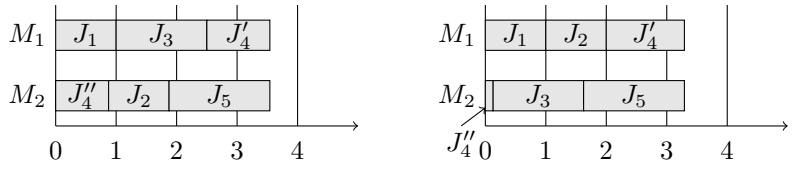
Własność 5.6 (Drugi warunek konieczny uszeregowania optymalnego)**.**
Jeżeli uszeregowanie $\sigma^\star$ jest optymalne, to $TL^1(\sigma^\star) = TL^2(\sigma^\star)$.

Dowód. Załóżmy, że uszeregowanie $\sigma^\star$ jest optymalne i $TL^1(\sigma^\star) \neq TL^2(\sigma^\star)$. To, na mocy własności 5.5, możemy skonstruować z uszeregowania $\sigma^\star$ inne uszeregowanie, σ, takie że $TL^1(\sigma) = TL^2(\sigma)$ i $C_{\max}(\sigma) < C_{\max}(\sigma^\star)$. Sprzeczność. □

Jednak, jak pokazuje następny przykład, równość całkowitych obciążeń obu maszyn w uszeregowaniu nie gwarantuje, że uszeregowanie jest optymalne, ponieważ może istnieć krótsze uszeregowanie z równymi obciążeniami obu maszyn. Zobrazuje nam to następujący przykład.

Przykład 5.5. Załóżmy, że mamy $n = 5$ zadań z podstawowymi czasami wykonywania $p_1 = 1$, $p_2 = 2$, $p_3 = 3$, $p_4 = 4$, $p_5 = 5$ i indeksem uczenia się $a = -1$. Rysunek 5.5a przedstawia uszeregowanie σ z $C_{\max}(\sigma) = 3\frac{13}{24}$, w którym maszyny M_1 i M_2 kończą wykonywanie zadań w tym samym czasie, a współczynnik podziału zadania jest równy $x_4(\sigma) = \frac{25}{32}$. Jednakże, jak pokazano na rysunku 5.5b, jeśli zmienimy sekwencję zadań na maszynach M_1 i M_2, i przerwiemy J_2 ze

współczynnikiem podziału zadania $x_4(\sigma^\star) = \frac{31}{32}$, to w nowym uszeregowaniu, $\sigma^\star$, obie maszyny kończą wykonywanie zadań ponownie w tym samym czasie ale uszeregowanie $\sigma^\star$ jest krótsze niż σ, tj. $C_{\max}(\sigma^\star) = 3\frac{7}{24} < C_{\max}(\sigma)$. Zwróćmy również uwagę na to że $\sigma^\star$ jest unikalnym optymalnym uszeregowaniem dla tej instancji.



(a) Uszeregowanie σ, $C_{\max}(\sigma) = 3\frac{13}{24}$

(b) Optymalne uszeregowanie $\sigma^\star$, $C_{\max}(\sigma^\star) = 3\frac{7}{24}$

Rysunek 5.5: Dwa różne uszeregowania z pojedynczym podzielonym zadaniem

Wpływ efektu uczenia się powoduje, że niektóre zmiany sekwencji niepodzielonych zadań przypisanych w danym uszeregowaniu do tej samej maszyny mogą prowadzić do krótszych uszeregowań niż dane uszeregowanie wyjściowe. Szukając uszeregowania optymalnego musimy więc wykonać następujące czynności:

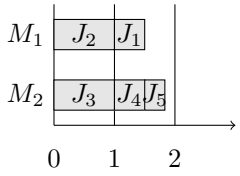
- wyznaczyć dzielone zadanie J_k,
- wyznaczyć sekwencje zadań niepodzielonych $\underline{J} = J \setminus \{J_k\}$,
- z powstałej sekwencji zadań niedzielonych $\underline{J}$ wygenerować pewną podsekwencję zadań, którą przydzielimy do pierwszej maszyny J^1,
- dopełnienia powstałych sekwencji przydzielić do drugiej maszyny $J^2 = \underline{J} \setminus J^1$.

Postępując według powyższych kroków otrzymujemy dwie rozłączne sekwencję zadań J^1 i J^2 oraz dzielone zadanie J_k. Czyli $J = J^1 \cup J^2 \cup \{J_k\}$. Aby znaleźć uszeregowanie optymalne musimy więc sprawdzić wszystkie takie sekwencje. Jednak podczas tej czynności natrafimy na dwa problemy.

Po pierwsze, nietrudno zauważyć, że niektóre z tych podziałów nie będą spełniały własności 5.6. Rozpatrzmy następujący przykład.

Przykład 5.6. Załóżmy, że mamy dane $n = 5$ zadań z podstawowymi czasami wykonywania $p_i = 1$ dla $1 \leqslant i \leqslant 5$ oraz $a = -1$. Próbujemy podzielić zadanie J_1, do pierwszej maszyny przydzielamy zadanie J_2, a pozostałe zadania szeregujemy na drugiej maszynie. Powstałe uszeregowanie przedstawia rysunek 5.6. Zauważmy, że obciążenia obu maszyn nie są równe i dodatkowo zadanie J_1 nie zostało podzielone.

Powyższy przykład pokazuje sytuację, która będzie występowała często, to znaczy generując wszystkie podziały sekwencji $\underline{J}$ otrzymamy takie sekwencje zadań którym będą odpowiadały uszeregowania nie spełniające własności 5.6.



Rysunek 5.6: Uszeregowanie nie spełniające własności 5.6

Po drugie, jeśli własność 5.6 będzie spełniona, to powinniśmy znać wartość współczynnika podziału dzielonego zadania.

W rozwiązaniu powyższych problemów pomoże nam następująca własność.

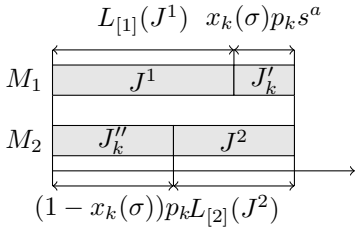
Własność 5.7. *Niech σ będzie uszeregowaniem takim, że $TL^1(\sigma) = TL^2(\sigma)$, zadanie J_k jest uszeregowane jako ostatnie na maszynie M_1 i s jest pozycją tego zadania w uszeregowaniu. Wtedy*

$$0 \leqslant \frac{L_{[2]}(J^2) - L_{[1]}(J^1) + p_k}{p_k(s^a + 1)} < 1. \tag{5.5}$$

Dowód. Załóżmy, że $TL^1(\sigma) = TL^2(\sigma)$ i niech zadanie J_k będzie uszeregowane na maszynie M_1 na pozycji s jako ostatnie zadanie uszeregowane na tej maszynie. Równość $TL^1(\sigma) = TL^2(\sigma)$ jest równoważna równości

$$L_{[1]}(J^1) + x_k(\sigma)p_k s^a = L_{[2]}(J^2) + (1 - x_k(\sigma))p_k \tag{5.6}$$

jak pokazano na rysunku 5.7.



Rysunek 5.7: Przykład uszeregowania z $x_k(\sigma) \in [0, 1)$

Obliczając z (5.6) współczynnik podziału zadania $x_k(\sigma)$, otrzymujemy

$$x_k(\sigma) = \frac{L_{[2]}(J^2) - L_{[1]}(J^1) + p_k}{p_k(s^a + 1)}.$$

Są dwa możliwe przypadki: przypadek 1, kiedy zadanie J_k jest podzielone i przypadek 2, kiedy zadanie J_k nie jest podzielone.

W przypadku 1, na mocy własności 5.1 mamy, że $x_k(\sigma) \in (0, 1)$.

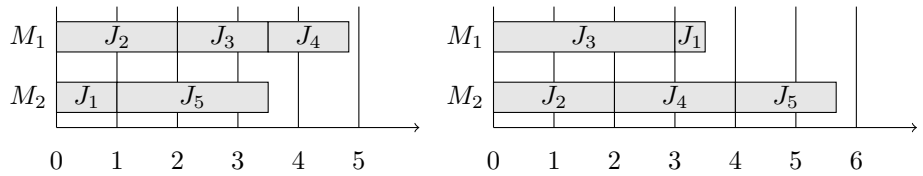
W przypadku 2, zadanie J_k jest uszeregowane w całości na pierwszej pozycji na maszynie M_2, a to oznacza, że $L_{[1]}(J^1) = L_{[2]}(J^2) + p_k$, stąd wynika że licznik

ilorazu w (5.5) równa się 0, czyli iloraz jest równy 0. Stąd, wartość $x_k(\sigma)$ zawiera się w przedziale $[0, 1)$. □

Mając dany odpowiedni podział sekwencji zadań i własność 5.7 jesteśmy w stanie określić, czy podział ten spełnia własność 5.6. Dodatkowo zauważmy, że jeżeli $TL^1(\sigma) = TL^2(\sigma)$, to (5.5) jest alternatywną definicją współczynnika podziału zadania $x_k(\sigma)$. Jednak, jeśli $TL^1(\sigma) \neq TL^2(\sigma)$, to (5.5) może mieć wartość spoza przedziału $[0, 1)$. Zilustrujemy ten fakt poniższym przykładem.

Przykład 5.7. Załóżmy, że mamy dane $n = 5$ zadań z podstawowymi czasami wykonywania $p_1 = 1$, $p_2 = 2$, $p_3 = 3$, $p_4 = 4$, $p_5 = 5$ i chcemy podzielić zadanie J_1. Rozważymy dwa uszeregowania, σ i σ'.

W uszeregowaniu σ, szeregujemy zadania J_2, J_3, J_4 na maszynie M_1 a zadanie J_5 na maszynie M_2. Wtedy, wartości $L_{[1]}(J^1)$ i $L_{[2]}(J^2)$ są odpowiednio równe $4\frac{5}{6}$ i $2\frac{1}{2}$. W tym przypadku wartość (5.5) wynosi $-1\frac{1}{15} < 0$ (patrz rysunek 5.8a).



(a) Uszeregowanie σ (b) Uszeregowanie σ'

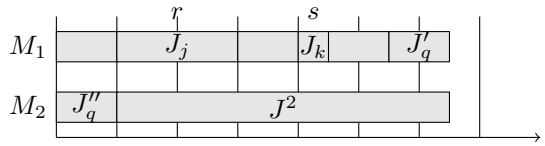
Rysunek 5.8: Współczynnik podziału zadania a podzielność zadań

W uszeregowaniu σ', szeregujemy zadanie J_3 na maszynie M_1 i zadania J_2, J_4, J_5 na maszynie M_2. Wtedy, wartości $L_{[1]}(J^1)$ i $L_{[2]}(J^2)$ są odpowiednio równe 3 i $3\frac{7}{12}$. W tym przypadku, wartość (5.5) wynosi $1\frac{1}{18} > 1$ (patrz rysunek 5.8b).

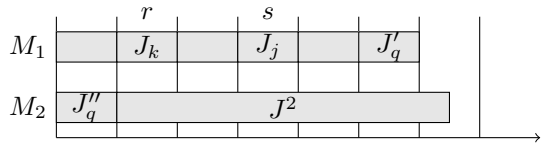
Następnie powinniśmy zastanowić się czy kolejność niepodzielonych zadań przydzielonych do jednej maszyny ma znaczenie. Z faktu iż algorytm SPT jest optymalny dla jednomaszynowego problemu z niepodzielnymi zadaniami pozycyjno-zależnymi oraz kryterium $C_{\max}$ (zobacz Mosheiov (2001)), wynika następująca własność (por. Żurowski (2018a)).

Własność 5.8. *W optymalnym uszeregowaniu wszystkie niepodzielone zadania na maszynie M_i $i = 1, 2$, są uszeregowane w porządku SPT względem podstawowych czasów wykonywania zadań.*

Dowód. Załóżmy że w optymalnym uszeregowaniu $\sigma^\star$ niepodzielone zadania nie są w porządku SPT. To znaczy, że istnieje zadanie J_j uszeregowane na r-tej pozycji i zdanie J_k uszeregowane na s-tej pozycji, $r < s$, takie że $p_k < p_j$ (patrz rysunek 5.9a). Skonstruujemy z uszeregowania $\sigma^\star$ nowe uszeregowanie, σ, takie że zadanie J_j jest uszeregowane na s-tej pozycji a zadanie J_k jest uszeregowane



(a) Uszeregowanie $\sigma^\star$



(b) Uszeregowanie σ

Rysunek 5.9: Porządek SPT w optymalnym uszeregowaniu

na r-tej pozycji (patrz rysunek 5.9b). Ponieważ zadania uszeregowane po s-tej pozycji są wykonywane w obu uszeregowaniach na tych samych pozycjach, wystarczy pokazać, że $C_k(\sigma^\star) > C_j(\sigma)$.

Niech $S_k(\sigma^\star) = S_j(\sigma) + (p_j - p_k)r^a$, gdzie $(p_j - p_k)r^a$ jest różnicą między rzeczywistymi czasami wykonywania zadań uszeregowanych na r-tej pozycji w obu uszeregowaniach, i niech $r^a > s^a$ dla $a < 0$. Wówczas

$$\begin{aligned} C_k(\sigma^\star) &= S_k(\sigma^\star) + p_{k,s} \\ &= S_j(\sigma) + (p_j - p_k)r^a + p_k s^a \\ &> S_j(\sigma) + (p_j - p_k)s^a + p_k s^a \\ &= S_j(\sigma) + p_j s^a = C_j(\sigma). \end{aligned}$$

Ponieważ $C_k(\sigma^\star) > C_j(\sigma)$, to $TL^1(\sigma) < TL^1(\sigma^\star)$. Stąd na mocy własności 5.5, możemy skonstruować z σ uszeregowanie, σ', takie że $C_{\max}(\sigma') < C_{\max}(\sigma) = C_{\max}(\sigma^\star)$. Sprzeczność. □

Następna własność pokazuje o ile mogą się różnić sumy $L_{[2]}(J^2)$ i $L_{[1]}(J^1)$ dla uszeregowania σ, pod warunkiem, że $TL^1(\sigma) = TL^2(\sigma)$.

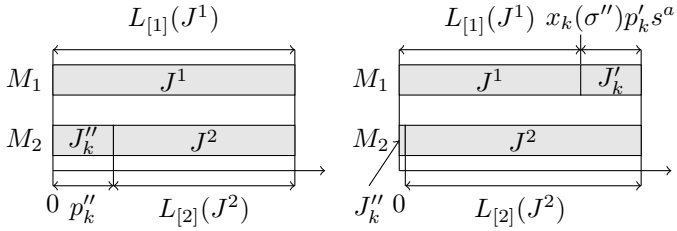
Własność 5.9. *Niech zadanie J_k będzie uszeregowane jako ostatnie, niech s będzie pozycją zadania J_k w uszeregowaniu σ. Jeżeli $TL^1(\sigma) = TL^2(\sigma)$, to $D + p_k \geqslant 0$ i $D < p_k s^a$.*

Dowód. Niech $D = L_{[2]}(J^2) - L_{[1]}(J^1)$ i $TL^1(\sigma) = TL^2(\sigma)$. Musimy rozważyć dwa przypadki: kiedy D jest najmniejsze (uszeregowanie σ') i kiedy D jest największe (uszeregowanie σ'').

Różnica D jest najmniejsza, kiedy $L_{[1]}(J^1) > L_{[2]}(J^2)$ i $L_{[1]}(J^1)$ jest największe. Wtedy $x_k(\sigma') = 0$ i całe dzielone zadanie jest uszeregowane na pierwszej pozycji na maszynie M_2 (patrz rysunek 5.10a). Wtedy $L_{[1]}(J^1) = p''_k + L_{[2]}(J^2)$ i po

przekształceniu i podstawieniu $D = L_{[2]}(J^2) - L_{[1]}(J^1)$, otrzymujemy $D + p''_k = 0$. Ponieważ $p_k \geqslant p''_k$, mamy $D + p_k \geqslant 0$.

Różnica D jest największa, kiedy $L_{[1]}(J^1) < L_{[2]}(J^2)$ i $L_{[1]}(J^1)$ jest najmniejsze. Wtedy $x_k(\sigma'')$ dąży do 1, ale nie jest równy 1 ponieważ dla spełnienia równości $TL^1(\sigma'') = TL^2(\sigma'')$ zadania z sekwencji J^2 muszą być uszeregowane na maszynie M_2 począwszy od drugiej pozycji. Wówczas prawie całe dzielone zadanie jest uszeregowane na ostatniej pozycji na maszynie M_1 (patrz rysunek 5.10b). Stąd $D < p_k s^a$. □



(a) Uszeregowanie σ', $D + p_k \geqslant 0$ (b) Uszeregowanie σ'', $D < p_k s^a$

Rysunek 5.10: Dolne i górne ograniczenie na D dla równych obciążeń maszyn

5.2.4 Własności związane z modyfikacją uszeregowania

Dwie ostatnie własności opisują wpływ usunięcia (wstawienia) zadania z (do) uszeregowania.

Własność 5.10. *Niech σ będzie sekwencją s zadań uszeregowanych na maszynie M_i w porządku SPT, $i = 1, 2$. Jeżeli usuniemy zadanie $J^i_{[r]}$ z sekwencji σ, obciążenie $TL^i(\sigma)$ skróci się jeśli*

$$p^i_{[s]}(s)^a > \sum_{j=r}^{s-1} (p^i_{[j+1]} - p^i_{[j]}) j^a. \tag{5.7}$$

Dowód. Niech σ i σ' oznacza sekwencję s zadań uszeregowanych na maszynie M_i, w porządku SPT i sekwencja σ' nie zawiera zadania $J^i_{[r]}$. Aby udowodnić (5.7), wystarczy sprawdzić kiedy $TL^i(\sigma) - TL^i(\sigma') > 0$. Zauważmy, że

$$TL^i(\sigma) = \sum_{j=1}^{r-1} p^i_{[j]} j^a + p^i_{[r]} r^a + \sum_{j=r+1}^{s} p^i_{[j]} j^a, \tag{5.8}$$

natomiast

$$TL^i(\sigma') = \sum_{j=1}^{r-1} p^i_{[j]} j^a + \sum_{j=r+1}^{s} p^i_{[j]} (j-1)^a. \tag{5.9}$$

Odejmując prawe strony (5.8) i (5.9) otrzymujemy

$$\begin{aligned} TL^i(\sigma) - TL^i(\sigma') &= p^i_{[r]} r^a + \sum_{j=r+1}^{s} p^i_{[j]} j^a - \sum_{j=r+1}^{s} p^i_{[j]} (j-1)^a = \\ &= p^i_{[r]} r^a + \cdots + p^i_{[s]} (s)^a - (p^i_{[r+1]} r^a + \cdots + p^i_{[s]} (s-1)^a) = \\ &= \sum_{j=r}^{s-1} (p^i_{[j]} - p^i_{[j+1]}) j^a + p^i_{[s]} (s)^a. \end{aligned}$$

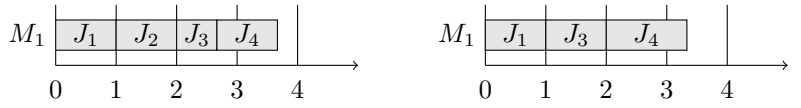
Ponieważ $TL^i(\sigma) - TL^i(\sigma') > 0$, to

$$\sum_{j=r}^{s-1} (p^i_{[j]} - p^i_{[j+1]}) j^a + p^i_{[s]} (s)^a > 0.$$

Po przekształceniu tej nierówności otrzymujemy (5.7). □

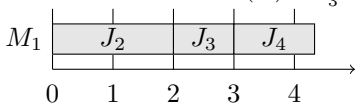
Zilustrujemy własność 5.10 następującym przykładem.

Przykład 5.8. Mamy $n = 4$ zadań z podstawowymi czasami wykonywania $p_1 = 1$, $p_2 = 2$, $p_3 = 2$, $p_4 = 4$ i $a = -1$. Wszystkie zadania są uszeregowane na maszynie M_1. Obciążenie maszyny $TL^1(\sigma)$ dla uszeregowania $\sigma = (1, 2, 3, 4)$ jest przedstawione na rysunku 5.11a. Po usunięciu zadania J_2 otrzymujemy uszeregowanie $\sigma' = (1, 3, 4)$ z mniejszym obciążeniem maszyny (patrz rysunek 5.11b) ale usunięcie zadania J_1 powoduje, że $TL^1(\sigma'')$ dla $\sigma'' = (2, 3, 4)$ wzrasta (patrz rysunek 5.11c).



(a) Uszeregowanie $\sigma = (1, 2, 3, 4)$ z $TL^1(\sigma) = 3\frac{2}{3}$

(b) Uszeregowanie $\sigma' = (1, 3, 4)$ z $TL^1(\sigma') = 3\frac{1}{3}$



(c) Uszeregowanie $\sigma'' = (2, 3, 4)$ z $TL^1(\sigma'') = 4\frac{1}{3}$

Rysunek 5.11: Wpływ usunięcia zadania na obciążenie maszyny

W podobny sposób możemy udowodnić następującą własność.

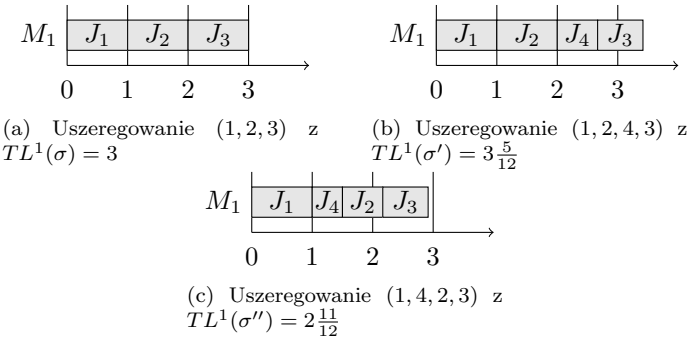
Własność 5.11. *Niech σ będzie sekwencją s zadań uszeregowanych na maszynie M_i w porządku SPT, $i = 1, 2$. Jeżeli wstawimy zadanie $J^i_{[r]}$ do sekwencji σ w taki sposób, że zostanie zachowany porządek SPT, obciążenie $TL^i(\sigma)$ skróci się*

jeśli

$$\sum_{j=r}^{s-1}(p^i_{[j+1]} - p^i_{[j]})j^a > p^i_{[s]}(s)^a. \tag{5.10}$$

Zilustrujemy własność 5.11 następującym przykładem.

Przykład 5.9. Mamy $n = 3$ zadań z podstawowymi czasami wykonywania $p_1 = 1$, $p_2 = 2$, $p_3 = 3$ i $a = -1$. Jak w przykładzie 5.8, wszystkie zadania są uszeregowane na maszynie M_1. Obciążenie maszyny $TL^1(\sigma)$ dla uszeregowania $\sigma = (1, 2, 3)$ jest przedstawione na rysunku 5.12a. Po wstawieniu do σ nowego zadania, J_4, z podstawowym czasem wykonywania $p_4 = 2$ otrzymujemy nowe uszeregowanie $\sigma' = (1, 2, 4, 3)$ z większym obciążeniem maszyny (patrz rysunek 5.12b), chociaż zachowany jest porządek SPT. Wstawienie w podobny sposób zadania J_4 z $p_4 = 1$ skutkuje uszeregowaniem $\sigma'' = (1, 4, 2, 3)$ z mniejszym obciążeniem maszyny (patrz rysunek 5.12c).



(a) Uszeregowanie $(1, 2, 3)$ z $TL^1(\sigma) = 3$

(b) Uszeregowanie $(1, 2, 4, 3)$ z $TL^1(\sigma') = 3\frac{5}{12}$

(c) Uszeregowanie $(1, 4, 2, 3)$ z $TL^1(\sigma'') = 2\frac{11}{12}$

Rysunek 5.12: Wpływ dodania zadania na obciążenie maszyny

Uwaga 5.1. *Zauważmy, że jeśli do pewnej sekwencji dodajemy zadanie o podstawowym czasie wykonywania większym niż mają zadania w tej sekwencji, tak aby zachować porządek SPT, to takie zadanie trafi zawsze na ostatnią pozycję tej sekwencji, a co więcej nie będzie spełniona nierówność 5.10, czyli sekwencja taka zawsze się wydłuży.*

Powyższa uwaga będzie wykorzystana do konstrukcji pewnego algorytmu w rozdziale 6.

5.3 Uwagi na temat złożoności

Problem PSLE pozostaje problemem otwartym. Rozpatrując wiele instancji tego problemu zauważyliśmy, że na długość uszeregowania diametralnie większy wpływ ma sposób przydziału zadań do maszyn niż wybór dzielonego zadania. Ponieważ przydzielając zadania do maszyn musimy sprawdzić wszystkie możliwości, których jest $O(n2^{n-1})$, więc możemy postawić hipotezę, że problem PSLE

jest problemem NP-trudnym. Dodatkowo brak rozwiązania optymalnego dla instancji z zadaniem dominującym sprawia, że problem PSLE nie jest podobny do żadnych innych znanych nam problemów szeregowania zadań.

Rozważając zmianę definicji podzielności na zmodyfikowaną definicję ograniczonej k podzielności (patrz Ecker i Hirschberg (1993)), uzyskalibyśmy możliwość znalezienia rozwiązania optymalnego dla instancji z zadaniem dominującym, jednak takie podejście wygenerowało by uszeregowanie z przestojem, podobnie jak ma to miejsce w algorytmie McNaughton'a, kiedy $C^{\star}_{\max} = \max_j\{p_j\}$ (patrz algorytm 3.1).

Szczególne instancje problemu PSLE, na przykład takie w której wszystkie podstawowe czasy wykonywania zadań są sobie równe możemy rozwiązać w czasie wielomianowym podobnie jak ma to miejsce dla problemu szeregowania podzielnych zadań o stałych czasach na dwóch równoległych identycznych maszynach z kryterium optymalności $C_{\max}$. Nie badaliśmy innych bardziej skomplikowanych instancji tego problemu. Możemy więc sformułować następujące twierdzenie.

Twierdzenie 5.1. *Jeżeli mamy instancję I problemu PSLE w której wszystkie zadania mają podstawowy czas wykonywania $p_j = c$ dla $1 \leqslant j \leqslant n$ i $c > 0$, to rozwiązanie optymalne dla tej instancji można znaleźć w czasie $O(n^2)$.*

Dowód. Musimy pokazać, że instancja I nie jest zdominowana, i pokazać algorytm o złożoności $O(n^2)$.

Aby pokazać, że instancja I nie jest zdominowana wystarczy pokazać uszeregowanie dla którego (5.2) nie jest spełnione. Takim uszeregowaniem może być uszeregowanie σ w którym wszystkie zadania z wyjątkiem jednego (kandydata na zadanie dominujące) są uszeregowane na maszynie M_1. Już dla dwóch zadań mamy $TL^1(\sigma) = c * 1^a > c * 2^a = TL^2(\sigma)$, a instancja I ma co najmniej 3 zadania.

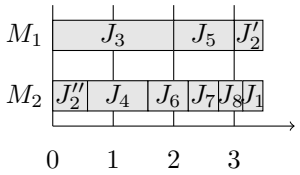
Wielomianowy algorytm znajdujący uszeregowanie optymalne dla instancji I w czasie $O(n^2)$ przedstawia pseudokod 5.1. Powyższy algorytm przemieści z maszyny M_1 na maszynę M_2 $n-1$ zadań. W każdej iteracji najbardziej czasochłonną operacją jest obliczenie obciążenia obu maszyn, które wykonuje się w czasie $O(n)$. Cały algorytm działa więc w czasie $O(n^2)$. □

Na koniec tego rozdziału pokażemy uszeregowanie optymalne dla instancji w której wszystkie zadania mają równy podstawowy czas wykonywania zadania.

Przykład 5.10. Mamy dane $n = 8$ zadań z podstawowymi czasami wykonywania $p_j = 2$ dla $1 \leqslant j \leqslant n$. Optymalne uszeregowanie dla tej instancji przedstawia rysunek 5.13, $C_{\max} = 3\frac{19}{40}$.

Algorytm 5.1: Wielomianowy algorytm znajdujący rozwiązanie optymalne dla instancji I

1 wszystkie zadania z wyjątkiem zadania J_1 (które jest zadaniem dzielonym) szeregujemy na maszynie M_1;
2 zadanie J_1 szeregujemy na maszynie M_2;
3 liczymy długość uszeregowania i zapamiętujemy ją w zmiennej $C_{\max}$, a uszeregowanie w zmiennej Opt;
4 jeżeli na maszynie M_1 nie ma żadnych zadań przechodzimy do punktu 9;
5 bierzemy ostatnie zadanie z maszyny M_1 i szeregujemy je na ostatniej pozycji na maszynie M_2;
6 obliczamy współczynnik podziału zadania i zależnie od jego wartości dzielimy zadanie J_1, albo szeregujemy je w całości na maszynie o mniejszym obciążeniu;
7 liczymy długość uszeregowania jeżeli jest mniejsza od zmiennej $C_{\max}$ to aktualizujemy zmienne $C_{\max}$ i Opt;
8 przechodzimy do punktu 4;
9 koniec algorytmu.



Rysunek 5.13: Uszeregowanie optymalne dla 8 zadań o równym podstawowym czasie wykonywania

Rozdział 6

Algorytmy dokładne

W rozdziale tym przedstawimy dwa algorytmy dokładne dla problemu PSLE opisane w rozdziale 5 oraz porównamy ich czas wykonywania dla losowych instancji. Rozdział 6 składa się z trzech podrozdziałów. W podrozdziale 6.1 przedstawimy algorytm wyliczeniowy dla problemu PSLE. W podrozdziale 6.2 zaprezentujemy algorytm podziału i ograniczeń dla badanego problemu. Na koniec w podrozdziale 6.3 omówimy wyniki eksperymentów z tymi algorytmami.

6.1 Algorytm wyliczeniowy

W tym podrozdziale przedstawimy pierwszy dokładny algorytm dla problemu PSLE (patrz algorytm 6.1), który jest algorytmem wyliczeniowym. Algorytm ten najpierw wybiera dzielone zadanie, a następnie generuje wszystkie możliwe uszeregowania zawierające podzielone zadanie, na koniec wybierając najkrótsze uszeregowanie.

W pseudokodzie algorytmu wyliczeniowego użyliśmy następujących funkcji:

- Funkcja `isDominantJob` przyjmuje jako argument sekwencję zadań J i używając nierówności (5.2), sprawdza czy w tej sekwencji istnieje zadanie dominujące, a jeżeli tak jest zwraca wartość logiczną `true`. W przeciwnym przypadku funkcja zwraca wartość logiczną `false`. Funkcja `isDominantJob` działa w czasie $O(2^n)$, gdzie n jest liczbą zadań.

- Funkcja `sortSPT` przyjmuje jako argument sekwencję zadań J i sortuje je w porządku niemalejącym względem podstawowych czasów wykonywania zadania. Funkcja `sortSPT` działa w czasie $O(n \lg n)$, gdzie n jest liczbą zadań.

- Funkcja `Schedule` przyjmuje trzy argumenty J^1, J^2 i J_k, odpowiednio sekwencję zadań uszeregowanych w całości na pierwszej i drugiej maszynie oraz dzielone zadanie. Na wyjściu funkcja zwraca strukturę `R` (reprezentującą uszeregowanie) o następujących polach:

- `n` liczba zadań
- `k` numer podzielonego zadania
- `x_k` współczynnik podziału zadania
- `M[]` wektor $n+1$-elementowy przechowujący numer maszyny dla każdego zadania
- `S[]` wektor $n+1$-elementowy przechowujący czas rozpoczęcia każdego zadania w uszeregowaniu
- `C[]` wektor $n+1$-elementowy przechowujący czas zakończenia każdego zadania w uszeregowaniu

Funkcja `Schedule` działa w czasie $O(n)$, gdzie n jest liczbą zadań.

- Funkcja `allSubSequences` przyjmuje na wejściu sekwencję zadań J i zwraca wszystkie podsekwencje sekwencji J. Funkcja `allSubSequences` działa w czasie $O(2^n)$, gdzie n jest liczbą zadań.
- Funkcja `getCmax` przyjmuje na wejściu strukturę `R` reprezentującą uszeregowanie, i zwraca długość uszeregowania. Funkcja `getCmax` działa w czasie $O(n)$, gdzie n jest liczbą zadań.

Algorytm 6.1, który dla podanej na wejściu początkowej sekwencji zadań, J, generuje uszeregowanie optymalne, działa następująco.

Algorytm 6.1: Algorytm wyliczeniowy dla problemu PSLE

```
1  if isDominantJob(J) then
2  |  return
3  end
4  J̄ = sortSPT(J)
5  Opt = Schedule(J̄,(),NIL)
6  for i = 1 to n do
7  |  J_ = J̄ \ {J_i}
8  |  forall S ∈ allSubSequences(J_) do
9  |  |  R = Schedule(S, J_ \ S, J_i)
10 |  |  if getCmax(R)<getCmax(Opt) then
11 |  |  |  Opt = R
12 |  |  end
13 |  end
14 end
15 return Opt
```

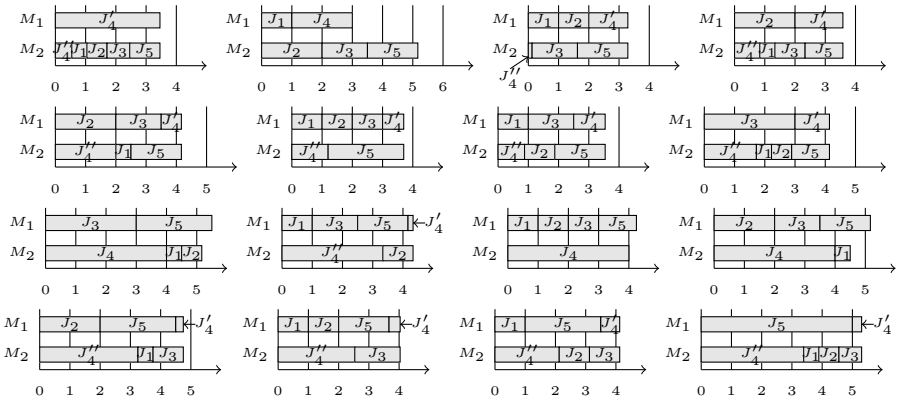
W linii 1, funkcja `isDominantJob` weryfikuje, czy w danej instancji istnieje zadanie dominujące. Jeśli istnieje, w linii 2 algorytm zatrzymuje się. W linii 4, funkcja `sortSPT` sortuje wszystkie zadania w porządku SPT. W linii 5, funkcja `Schedule` tworzy uszeregowanie bez podzielonego zadania, gdzie wszystkie

zadania są uszeregowane na maszynie M_1. Tak skonstruowane uszeregowanie jest aktualnie najlepszym uszeregowaniem. Znając aktualnie najlepsze uszeregowanie, algorytm 6.1 wybiera w linii 7 sekwencję $\underline{J}$ bez podzielonego zadania J_i. W linii 8, pętla `forall` iteruje przez wszystkie sekwencje S zadań które zostaną uszeregowane na maszynie M_1. W linii 9, funkcja `Schedule` generuje uszeregowanie z podzielonym zadaniem J_i, gdzie zadania z sekwencji S są uszeregowane na maszynie M_1 a pozostałe zadania są uszeregowane na maszynie M_2. W liniach 10–11, algorytm sprawdza, czy długość nowego uszeregowania, R, jest krótsza od aktualnie najlepszego uszeregowania Opt; jeśli jest mniejsza, Opt jest aktualizowane. Optymalne uszeregowanie, Opt, jest zwracane w linii 15.

Twierdzenie 6.1. *Algorytm 6.1 rozwiązuje problem PSLE w czasie* $O(n^2 2^n)$.

Dowód. W liniach 1–5 funkcją o najdłuższym czasie wykonywania jest funkcja `isDominantJob` działająca w czasie $O(2^n)$. Pozostałe funkcje `sortSPT` i `Schedule` wykonują się z czasami odpowiednio $O(n \lg n)$ i $O(n)$. Pętla `for` w linii 6 wykonuje się n razy, a pętla `forall` w linii 8 wykonuje się 2^n razy. Funkcje i operacje w liniach 7 oraz 9–11 wykonują się w czasie $O(n)$. Algorytm 6.1 działa więc w czasie $O(n^2 2^n)$. □

Przedstawimy sposób działania algorytmu 6.1 na następującym przykładzie.



Rysunek 6.1: Wszystkie uszeregowania z podzielonym zadaniem J_4

Przykład 6.1. Mamy $n = 5$ zadań z podstawowymi czasami wykonywania $p_1 = 1$, $p_2 = 2$, $p_3 = 3$, $p_4 = 4$, $p_5 = 5$. Algorytm 6.1 generuje 80 uszeregowań. Na rysunku 6.1 prezentujemy wszystkie 12 uszeregowań w których zadanie J_4 jest podzielone. Algorytm 6.1 generuje jedno optymalne uszeregowanie, $\sigma^\star$, dane na rysunku 5.5b, gdzie zadanie J_4 jest podzielone, współczynnik podziału zadnia $x_4(3\frac{7}{24}) = \frac{31}{32}$ i $C_{\max}(\sigma^\star) = 3\frac{7}{24}$.

Ręczne obliczanie czasów wykonywania zadań było dla nas niezwykle żmudnym i niełatwym zajęciem. Algorytm wyliczeniowy powstał w celu szybkiego

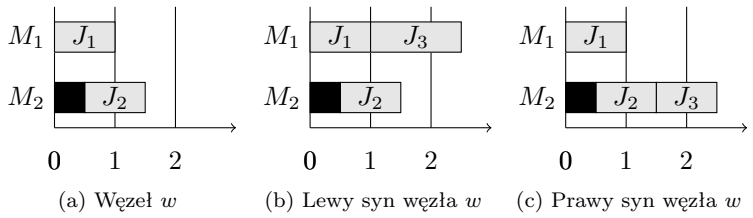
wyznaczania uszeregowań optymalnych dla instancji o niewielkiej liczbie zadań. Postanowiliśmy spróbować przyśpieszyć ten proces stosując algorytm opisany w następnym podrozdziale.

6.2 Algorytm podziału i ograniczeń

W tym podrozdziale przedstawimy drugi dokładny algorytm dla problemu PSLE (patrz algorytm 6.2), który jest algorytmem podziału i ograniczeń (Woeginger (2003)). Chociaż algorytmy tego typu są szeroko stosowane dla rozwiązywania problemów z zadaniami pozycyjno-zależnymi (praca przeglądowa Azzouz i inni (2018)) to zazwyczaj są one stosowane dla problemów jednomaszynowych. Tylko kilku autorów proponuje algorytm podziału i ograniczeń dla problemów szeregowania na maszynach równoległych zadań z pozycyjno-zależnymi czasami wykonywania (patrz, np., Shim i Kim (2008); Okołowski i Gawiejnowicz (2010)). Co więcej, zgodnie z moją najlepszą wiedzą, nie zaproponowano żadnych algorytmów tego typu dla problemów szeregowania zadań na równoległych maszynach z podzielnymi zadaniami pozycyjno-zależnymi.

Algorytm podziału i ograniczeń, podobnie jak inne algorytmy tego typu, rozwiązuje problem PSLE używając dwóch operacji: *podziału* (ang. branching) i *ograniczeń* (ang. bounding).

Operacja podziału służy do budowania częściowych uszeregowań. W przypadku naszego problemu wybieramy zadanie, które ma być podzielone, J_i, a następnie szeregujemy pozostałe zadania według poniższego schematu. Tworzymy drzewo binarne, z pustym uszeregowaniem w korzeniu i kompletnymi uszeregowaniami w liściach drzewa. Każdy wewnętrzny węzeł drzewa odpowiada częściowemu uszeregowaniu, a przejście o jeden poziom w górę (w dół) odpowiada dodaniu (odjęciu) pojedynczego zadania. Dla przykładu, w wewnętrznym węźle w (patrz rysunek 6.2a), mamy częściowe uszeregowanie i dwa węzły potomne węzła są tworzone przez dodanie pojedynczego zadania: jeżeli przydzielimy zadanie J_3 do maszyny M_1, to utworzymy lewego syna węzła w (patrz rysunek 6.2b) a jeśli przydzielimy zadanie J_3 do maszyny M_2, to utworzymy prawego syna węzła w (patrz rysunek 6.2c).



(a) Węzeł w (b) Lewy syn węzła w (c) Prawy syn węzła w

Rysunek 6.2: Lewy i prawy syn węzła w

Operacja ograniczania eliminuje te węzły drzewa, które mają wartość kryterium mniejszą od najlepszego aktualnego dolnego ograniczenia wartości te-

go kryterium. W przypadku naszego problemu, operacja ograniczania korzysta z faktu, że tworząc dzieci węzła w dodajemy zadanie niekrótsze (biorąc pod uwagę podstawowy czas wykonywania zadania) od zadań które występują w uszeregowaniu reprezentowanym przez węzeł w. Zadanie w zależności od węzła potomnego zostaje dodane do pierwszej lub drugiej maszyny, a korzystając z uwagi 5.1 wiemy, że obciążenie maszyny do której zostaje dodane zadanie zawsze się zwiększy. W związku z powyższym, zależnie od tego do której maszyny dodamy zadanie, długość uszeregowania zwiększy się, albo pozostanie bez zmian. Wyjaśnimy ten proces na przykładzie. Mamy dwie maszyny M_1 i M_2 takie że $TL^1(\sigma) < TL^2(\sigma)$. Jeśli dodamy nowe zadanie J_r do maszyny M_1 tworząc uszeregowanie σ' to mamy dwa przypadki: przypadek 1, gdy $TL^1(\sigma') < TL^2(\sigma')$ albo przypadek 2, gdy $TL^1(\sigma') > TL^2(\sigma')$ zależnie od długości zadania J_r i różnicy między obciążeniami obu maszyn. W przypadku 1 długość uszeregowania nie zmieni się. W przypadku 2 długość uszeregowania zwiększy się podobnie jak w sytuacji gdy zadanie J_r zostanie dodane do maszyny M_2. Podsumowując, po dodaniu zadanie długość uszeregowania z pewnością nie zmniejszy się. Dlatego operacja ograniczanie eliminuje poddrzewo odpowiadające uszeregowaniom niekrótszym niż obecnie najkrótsze znane uszeregowanie.

W pseudokodzie algorytmu podziału i ograniczeń użyliśmy funkcji przedstawionych w poprzednim podrozdziale oraz następujących nowych funkcji:

- Funkcja `Heuristic` przyjmuje jako argument sekwencję zadań J i zwraca strukturę danych H przechowującą ścieżkę w drzewie binarnym od korzenia do liścia będącego rozwiązanie przybliżonym. Heurystyka użyta w badaniach działa w taki sposób, że kolejne zadania przydziela na przemian do pierwszej i drugiej maszyny. Funkcja `Heuristic` działa w czasie $O(n)$, gdzie n jest liczbą zadań (funkcja ta, jako algorytm $H1$, jest badana w rozdziale 7).

- Funkcja `getNodes` przyjmuje jako argument strukturę danych H, i zwraca zbiór węzłów na ścieżce od korzenia do liścia, który jest rozwiązaniem przybliżonym wygenerowanym przez funkcję `Heuristic`. Funkcja `getNodes` działa w czasie $O(n)$, gdzie n jest liczbą zadań.

- Struktura danych `Stack` jest strukturą danych typu FILO (ang. First In Last Out) (patrz Cormen i inni (2017)) na której można wykonywać następujące operacje:

  - Procedura `PushAll` przyjmuje jako argumenty stos s oraz kolekcję węzłów h, i umieszcza węzły z kolekcji h na stosie s. Procedura `PushAll` działa w czasie $O(n)$, gdzie n jest liczbą zadań.
  - Funkcja `isEmpty` przyjmuje jako argument stos s i jeżeli stos jest pusty zwraca wartość logiczną `true`. W przeciwnym wypadku funkcja zwraca wartość logiczną `false`. Funkcja `isEmpty` działa w czasie $O(1)$.
  - Funkcja `Pop` przyjmuje jako argument stos s i zwraca węzeł będący na szczycie stosu s usuwając go ze stosu. Funkcja `Pop` działa w czasie

$O(1)$.

- Funkcja `isLeaf` przyjmuje jako argument węzeł R i jeżeli węzeł jest liściem zwraca wartość logiczną `true`. W przeciwnym przypadku funkcja zwraca wartość logiczną `false`. Funkcja `isLeaf` działa w czasie $O(1)$.
- Funkcja `Rules` przyjmuje jako argumenty węzeł Opt i węzeł R, sprawdza czy długość uszeregowania reprezentowanego przez węzeł R jest większa od aktualnie najlepszego uszeregowani Opt. Jeśli tak jest zwraca wartość logiczną `true`. W przeciwnym przypadku funkcja zwraca wartość logiczną `false`. Funkcja `Rules` działa w czasie $O(1)$.
- Funkcja `getChildren` przyjmuje jako argumenty węzeł R oraz numer dzielonego zadania, i zwraca dwa węzły będące potomkami węzła R. Funkcja `getChildren` działa w czasie $O(1)$.

Algorytm 6.2: Algorytm typu branch-and-bound dla problemu PSLE

```
1  if isDominantJob(J) then
2  |  return
3  end
4  J̄ = sortSPT(J)
5  Opt = Schedule(J̄,(),NIL)
6  for i = 1 to n do
7  |  J = J̄ \ {J_i}
8  |  H = Heuristic(J)
9  |  h = getNodes(H)
10 |  Stack s
11 |  PushAll(s,h)
12 |  while not isEmpty(s) do
13 |  |  R = Pop(s)
14 |  |  if isLeaf(R) and getCmax(R) < getCmax(Opt) then
15 |  |  |  Opt = R
16 |  |  end
17 |  |  if not Rules(Opt,R) then
18 |  |  |  PushAll(s, getChildren(R,i))
19 |  |  end
20 |  end
21 end
22 return opt
```

Dla podanej na wejściu początkowej sekwencji zadań, J, algorytm 6.2 generuje na wyjściu uszeregowanie optymalne. W linii 1, funkcja `isDominantJob` weryfikuje, używając (5.2), czy w danej instancji problemu PSLE istnieje zadanie dominujące. Jeśli istnieje, w linii 2 algorytm zatrzymuje się. W linii 4,

funkcja `sortSPT` sortuje wszystkie zadania w porządku SPT. W linii 5, funkcja `Schedule` tworzy uszeregowanie bez podzielonego zadania, gdzie wszystkie zadania są uszeregowane na maszynie M_1. Tak skonstruowane uszeregowanie jest aktualnie najlepszym uszeregowaniem. Znając aktualnie najlepsze uszeregowanie, algorytm 6.2 wybiera w linii 7 sekwencję $\underline{J}$ bez podzielonego zadania J_i. W linii 8, funkcja `Heuristic` tworzy rozwiązanie przybliżone H. W linii 9, funkcia `getNodes` generuje z H wszystkie węzły na ścieżce od korzenia do liścia. W linii 10, algorytm 6.2 inicjalizuje stos s, i dodaje do niego wszystkie węzły z h w linii 11. Pętla `while` w linii 12 wykonuje linie 13–19, tak długo jak $s \neq \phi$. W linii 13, węzeł R jest usuwany ze stosu s. Jeżeli węzeł R jest liściem i długość uszeregowania w tym węźle jest krótsza od aktualnie najlepszego uszeregowania, to aktualnie najlepsze uszeregowanie jest aktualizowane (linia 15). W linii 17, funkcja `Rules` weryfikuje czy węzeł R spełnia pewne warunki. Jeśli nie, w linii 18 są generowane węzły potomne węzła R i dodane do stosu s. Ostatecznie optymalne uszeregowanie, *Opt*, jest zwracane w linii 15.

Twierdzenie 6.2. *Algorytm 6.2 rozwiązuje problem PSLE w czasie* $O(n^2 2^{n-1})$.

Dowód. W liniach 1–5 funkcją o najdłuższym czasie wykonywania jest funkcja `isDominantJob` działająca w czasie $O(2^n)$. Pozostałe funkcje `sortSPT` i `Schedule` wykonują się z czasami odpowiednio $O(n \lg n)$ i $O(n)$. Pętla `for` w linii 6 wykonuje się n razy, a pętla `while` w linii 12 wykonuje się 2^{n-1} razy. Funkcje i operacje w liniach 7–9 oraz funkcja `getCmax` wykonują się w czasie $O(n)$. Funkcje w liniach 13–18 z wyjątkiem funkcji `getCmax` wykonują się w czasie $O(1)$. Algorytm 6.2 działa więc w czasie $O(n^2 2^{n-1})$. □

Zilustrujemy algorytm 6.2 następującym przykładem.

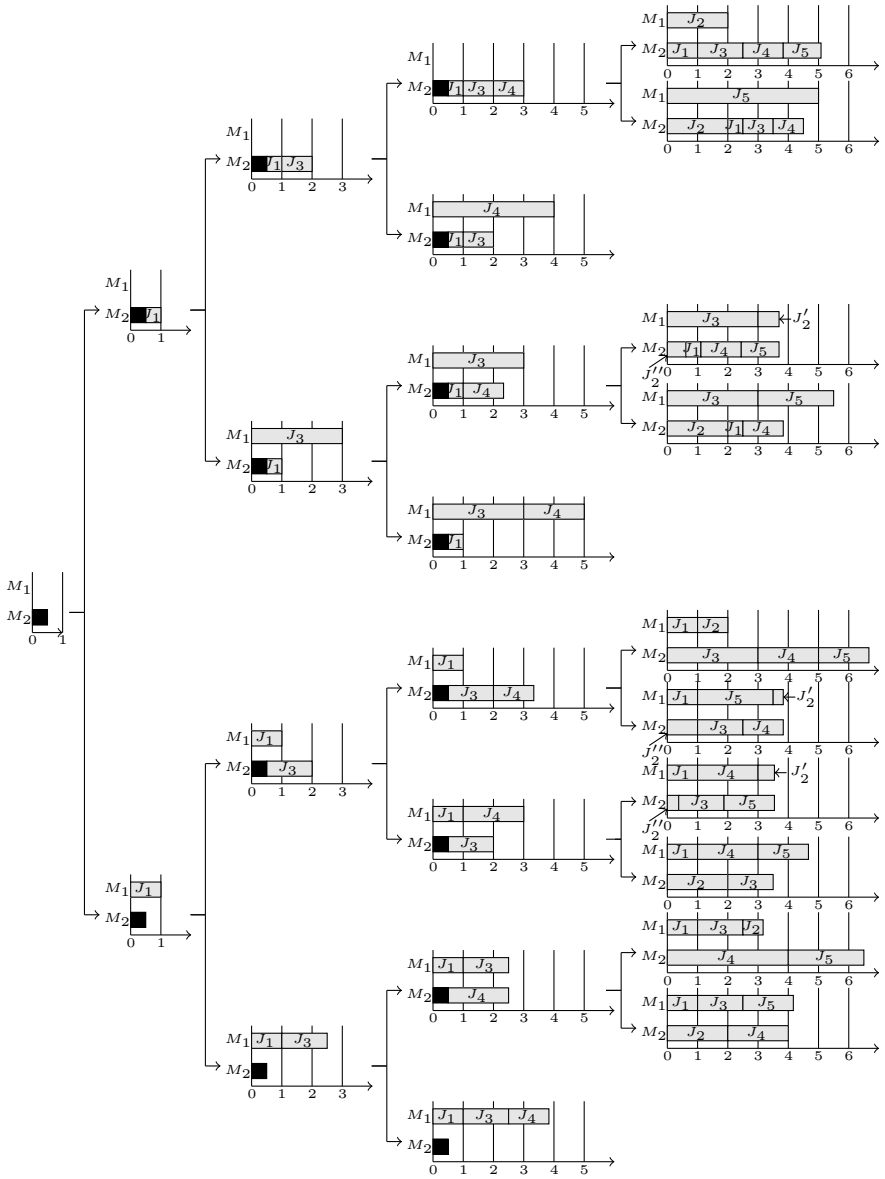
Przykład 6.2. Mamy $n = 5$ zadań z podstawowymi czasami wykonywania $p_1 = 1$, $p_2 = 2$, $p_3 = 3$, $p_4 = 4$, $p_5 = 5$. Algorytm 6.2 generuje pięć drzew T_i, $1 \leqslant i \leqslant 5$, po jednym dla każdego przerwanego zadania J_i. Na rysunku 6.3 przedstawiamy drzewo T_2.

Optymalne uszeregowanie $\sigma^\star$ dla danej instancji prezentuje rysunek 5.5b, gdzie podzielone jest zadanie J_4, współczynnik podziału zadania $x_4(3\frac{7}{24}) = \frac{31}{32}$ i $C_{\max}(\sigma^\star) = 3\frac{7}{24}$.

Z twierdzenia 6.2 wynika iż algorytm podziału i ograniczeń winien znajdować optymalne uszeregowanie szybciej niż algorytm wyliczeniowy. W celu weryfikacji tej hipotezy przeprowadzono eksperymenty numeryczne, w których porównano czasy działania obu omawianych algorytmów dokładnych dla tego samego zbioru instancji. Wyniki tych eksperymentów przedstawiono w następnym rozdziale.

6.3 Wyniki eksperymentów obliczeniowych

W niniejszym podrozdziale przedstawimy wyniki eksperymentów obliczeniowych zestawione w taki sposób, oby można było porównać algorytm wyliczeniowy z algorytmem podziału i ograniczeń. Dla tego drugiego dokonaliśmy też analizy liczby pominiętych węzłów.



Rysunek 6.3: Drzewo T_2

Zaimplementowaliśmy nasze algorytmy w języku C++, stosując 64-bitowy kompilator GCC w wersji 4.9.2. Nasze implementacje zostały przetestowane na laptopie z procesorem Intel Core i5-6200U 2.40 GHz, pod kontrolą systemu Windows 10.

Aby zweryfikować maksymalny rozmiar instancji, który można rozwiązać

w rozsądnym czasie, przeprowadziliśmy dwa eksperymenty obliczeniowe. Dla każdego n, wygenerowaliśmy 20 losowych instancji, w których podstawowe czasy wykonywania zadań są liczbami całkowitymi wylosowanymi z przedziału $[1, 10]$. Wyniki tych eksperymentów podano w tabelach 6.1–6.2.

Tabela 6.1: Czasy wykonywania algorytmu 6.1

n	$t_{\min}$	t_{avg}	$t_{\max}$
3	0.0000	0.0000	0.0000
4	0.0000	0.0000	0.0000
5	0.0000	0.0000	0.0000
6	0.0000	0.0002	0.0010
7	0.0000	0.0006	0.0010
8	0.0000	0.0008	0.0010
9	0.0020	0.0023	0.0030
10	0.0040	0.0054	0.0070
11	0.0130	0.0143	0.0220
12	0.0310	0.0435	0.1570
13	0.0720	0.0801	0.1290
14	0.1690	0.1796	0.2130
15	0.3890	0.4012	0.4210
16	0.8940	0.9687	1.1810
17	2.0210	2.0725	2.3060
18	4.5620	4.7886	5.6870
19	10.2640	10.7635	12.4330
20	22.7060	24.9457	27.2200
21	57.0340	57.9904	59.5520
22	126.2660	141.0591	165.8200
23	288.8460	334.8256	371.3000
24	522.2080	533.9919	548.8070
25	1 131.7000	1 205.5655	1 549.6800

W tabeli 6.1, podano minimalne, średnie i maksymalne czasy obliczeń dla algorytmu wyliczeniowego (patrz algorytm 6.1). W tabeli, symbole n, $t_{\min}$, t_{avg} i $t_{\max}$ oznaczają odpowiedni rozmiar instancji, minimalny, średni i maksymalny czas obliczeń.

W tabeli 6.2, podano minimalne, średnie i maksymalne czasy obliczeń dla algorytmu podziału i ograniczeń (patrz algorytm 6.2). W tabeli, symbole n, $t_{\min}$, t_{avg} i $t_{\max}$ oznaczają odpowiedni rozmiar instancji, minimalny, średni i maksymalny czas obliczeń.

W tabeli 6.3, przedstawiono minimalną, średnią i maksymalną liczbę pominiętych węzłów przez algorytm podziału i ograniczeń (patrz algorytm 6.2). W tabeli, symbole n, $d_{\min}$, d_{avg} i $d_{\max}$ oznaczają odpowiedni rozmiar instancji, minimalną, średnią i maksymalną liczbę pominiętych węzłów.

Rysunek 6.4 przedstawia wartości procentowe minimalnej, średniej i maksymalnej liczby pominiętych węzłów dla algorytmu podziału i ograniczeń (patrz

Tabela 6.2: Czasy wykonywania algorytmu 6.2

n	$t_{\min}$	t_{avg}	$t_{\max}$
3	0.0000	0.0000	0.0000
4	0.0000	0.0000	0.0000
5	0.0000	0.0000	0.0000
6	0.0000	0.0000	0.0000
7	0.0000	0.0000	0.0000
8	0.0000	0.0000	0.0000
9	0.0000	0.0000	0.0000
10	0.0000	0.0000	0.0000
11	0.0050	0.0075	0.0130
12	0.0130	0.0189	0.0260
13	0.0310	0.0540	0.1820
14	0.0680	0.0890	0.1350
15	0.1260	0.1784	0.2100
16	0.2130	0.3734	0.4670
17	0.5840	0.8012	1.0190
18	1.3270	1.6917	2.0750
19	2.2860	3.8172	5.1800
20	5.2870	7.2792	8.9010
21	10.6370	15.3653	19.1120
22	23.4730	30.3815	38.5110
23	51.0270	65.1982	82.9030
24	81.6890	124.9764	150.6090
25	178.9860	223.0283	279.8000

algorytm 6.2).

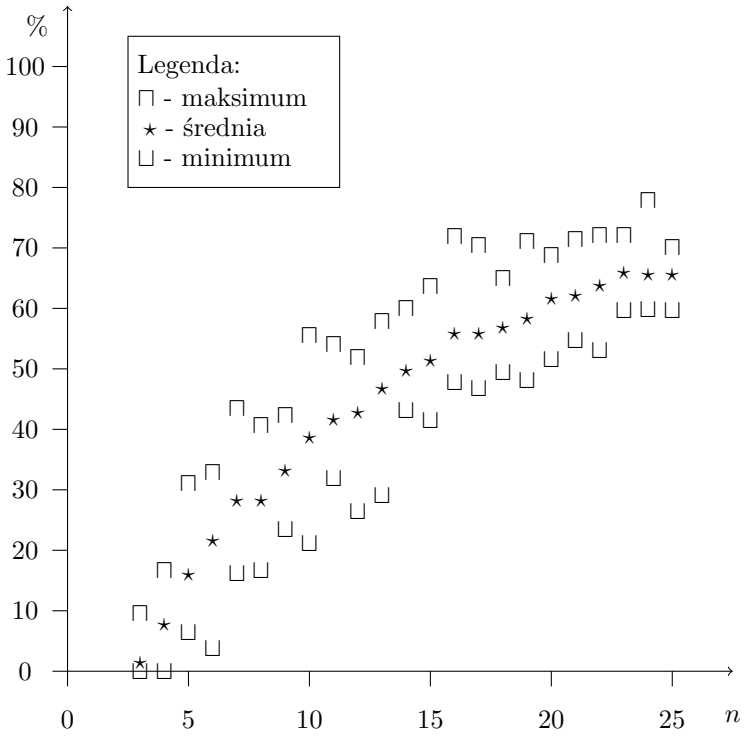
Wyniki tych eksperymentów pokazują, że proponowany algorytm podziału i ograniczeń jest szybszy niż algorytm wyliczeniowy. Nie jest to zaskoczeniem, ponieważ algorytm podziału i ograniczeń rozważa mniejszą liczbę uszeregowań. Zaskakującym wynikiem jest jednak średni procent pominiętych węzłów, który jest dość duży. Dla instancji z liczbą zadań od 3 do 15, średni procent pominiętych węzłów wzrastał od 1.43% do 51.35%, podczas gdy dla instancji większej niż 15 zadań, ten procent był jeszcze większy i wahał się między 51.35% a 65.54%.

Podczas eksperymentów zaobserwowaliśmy że algorytm podziału i ograniczeń pomija różne liczby węzłów dla różnych instancji. Wynik ten był spodziewany, ale staraliśmy się znaleźć główny czynnik wpływający na liczbę pominiętych węzłów. Na przykład, spośród wszystkich instancji z 4 zadaniami które testowaliśmy, największą liczbę pominiętych węzłów, 10, zaobserwowaliśmy dla instancji $(3, 4, 5, 6)$, podczas gdy najmniejszą liczbę pominiętych węzłów, 0, zaobserwowaliśmy dla instancji $(1, 2, 3, 9)$ i $(1, 1, 6, 10)$. Na podstawie wyników tych eksperymentów sformułowaliśmy hipotezę mówiącą o różnicy między podstawowym czasem wykonywania największego i najmniejszego zadania wpływa na liczbę pominiętych węzłów. Aby potwierdzić tę hipotezę wymagane jest wię-

Tabela 6.3: Liczba pominiętych węzłów przez algorytm podziału i ograniczeń

n	liczby		
	$d_{\min}$	d_{avg}	$d_{\max}$
3	0	0.3000	2
4	0	4.6000	10
5	10	24.6000	48
6	14	81.4000	124
7	143	249.8500	386
8	338	574.3000	828
9	1 076	1 521.2500	1 944
10	2 162	3 948.9000	5 674
11	7 184	9 347.1000	12 166
12	12 932	21 020.7000	25 500
13	30 962	49 744.1000	61 478
14	98 800	113 817.7000	137 560
15	204 014	252 385.5500	312 710
16	500 424	584 459.7000	753 138
17	1 040 134	1 245 189.8000	1 569 808
18	2 332 020	2 680 866.3000	3 064 930
19	4 784 910	5 809 914.0500	7 084 570
20	10 817 986	12 901 889.3000	14 414 046
21	24 111 550	27 340 557.9500	31 421 114
22	48 882 442	58 820 082.7000	66 445 488
23	114 973 540	127 083 720.2500	138 992 692
24	240 583 328	263 944 898.2000	313 670 100
25	500 343 524	549 828 018.8500	587 822 986

cej eksperymentów.



Rysunek 6.4: Liczba pominiętych węzłów (w %)

Rozdział 7

Algorytmy heurystyczne

W rozdziale tym przedstawimy dwa algorytmy przybliżone oraz porównamy ich czasy działania i jakość wygenerowanych przez nie rozwiązań. Rozdział 7 składa się z trzech podrozdziałów. W podrozdziałach 7.1 i 7.2 opiszemy działanie obu algorytmów. W podrozdziale 7.3 przedstawimy wyniki dwóch eksperymentów numerycznych w których dokonano porównania działania obu heurystyk.

7.1 Algorytm heurystyczny H1

W tym podrozdziale przedstawimy pierwszy algorytm heurystyczny dla problemu PSLE (patrz algorytm 7.1). W tym algorytmie mając dane dzielone zadanie, zamiast rozpatrywać wszystkie możliwe uszeregowania pozostałych zadań, przydzielamy zadania na przemian do pierwszej i drugiej maszyny.

W pseudokodzie algorytmu heurystycznego użyliśmy funkcji przedstawionych w poprzednim rozdziale oraz następującej notacji. Wyrażenie $J^i \cup \{J_r\}$ oznacza dodanie zadania J_r na koniec sekwencji J^i.

Wielomianowy algorytm heurystyczny przedstawia pseudokod 7.1.

Dla podanej na wejściu początkowej sekwencji zadań, J, algorytm 7.1 generuje uszeregowanie przybliżone. W linii 1, funkcja `sortSPT` sortuje wszystkie zadania w porządku SPT. W linii 2, funkcja `Schedule` tworzy uszeregowanie bez podzielonego zadania, gdzie wszystkie zadania są uszeregowane na maszynie M_1. Tak skonstruowane uszeregowanie jest aktualnie najlepszym uszeregowaniem. Znając aktualnie najlepsze uszeregowanie, algorytm 7.1 wybiera w linii 4 sekwencję $\underline{J}$ bez podzielonego zadania J_i. W liniach 5 i 6 algorytm tworzy dwie puste sekwencje J^1 i J^2. W linii 7, pętla `for` iteruje przez wszystkie zadania z sekwencji $\underline{J}$. W linii 8, w instrukcji warunkowej `if` sprawdzane jest czy licznik pętli j jest liczbą nieparzystą, jeżeli tak jest, to w linii 9 rozpatrywane j-te zadanie z sekwencji $\underline{J}$ jest przydzielane do sekwencji J^1. W przeciwnym razie, w linii 11, kiedy j jest parzyste j-te zadanie z sekwencji $\underline{J}$ jest przydzielane do sekwencji J^2. W linii 14, funkcja `Schedule` generuje uszeregowanie z podzielo-

Algorytm 7.1: Algorytm heurystyczny $H1$ dla problemu PSLE

```
1  J̄ = sortSPT(J)
2  Apr = Schedule(J̄,(),NIL)
3  for i = 1 to n do
4  |  J_ = J̄ \ {J_i}
5  |  J^1 = ()
6  |  J^2 = ()
7  |  for j = 1 to |J_| do
8  |  |  if j mod 2 = 1 then
9  |  |  |  J^1 = J^1 ∪ {J_[j](J_)}
10 |  |  else
11 |  |  |  J^2 = J^2 ∪ {J_[j](J_)}
12 |  |  end
13 |  end
14 |  R =Schedule(J^1,J^2,J_i)
15 |  if getCmax(R)<getCmax(Apr) then
16 |  |  Apr = R
17 |  end
18 end
19 return Apr
```

nym zadaniem J_i, gdzie zadania z sekwencji J^1 są uszeregowane na maszynie M_1 a zadania z sekwencji J^2 są uszeregowane na maszynie M_2. W liniach 15–16, algorytm sprawdza, czy długość nowego uszeregowania, R, jest mniejsza od aktualnie najlepszego uszeregowania Apr; jeśli jest mniejsza, Apr jest aktualizowane. Końcowe uszeregowanie, Apr, jest zwracane w linii 19.

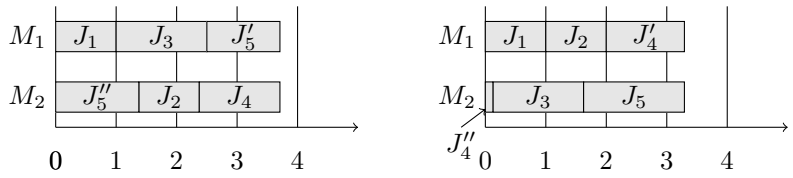
Twierdzenie 7.1. *Algorytm 7.1 rozwiązuje problem PSLE w czasie $O(n^3)$.*

Dowód. Funkcje `sortSPT` i `Schedule` mają czasy wykonywania, odpowiednio $O(n \lg n)$ i $O(n)$. Pętla `for` w linii 3 wykonuje się n razy, a pętla `for` w linii 7 wykonuje się $n-1$ razy. Funkcje i operacje w liniach 4 oraz 14–15 wykonują się w czasie $O(n)$. Pozostałe operacje wykonują się w czasie $O(1)$. Algorytm 7.1 działa więc w czasie $O(n^3)$. □

Przedstawimy sposób działania algorytmu 7.1 na następujących przykładach.

Przykład 7.1. Mamy $n = 5$ zadań z podstawowymi czasami wykonywania $p_1 = 1$, $p_2 = 2$, $p_3 = 3$, $p_4 = 4$, $p_5 = 5$ i indeksem uczenia się $a = -1$. Najkrótsze uszeregowanie σ wygenerowane przez algorytm 7.1 przedstawia rysunek 7.1a, $C_{\max}(\sigma) = 3\frac{17}{24}$. Tymczasem uszeregowanie optymalne $\sigma^\star$ przedstawia rysunek 7.1b, $C_{\max}(\sigma) = 3\frac{7}{24}$.

Przykład 7.2. Mamy dane $n = 5$ zadań z podstawowymi czasami wykonywania $p_1 = 3$, $p_2 = 4$, $p_3 = 6$, $p_4 = 2$ i $p_5 = 6$, i indeks uczenia się $a = -1$.

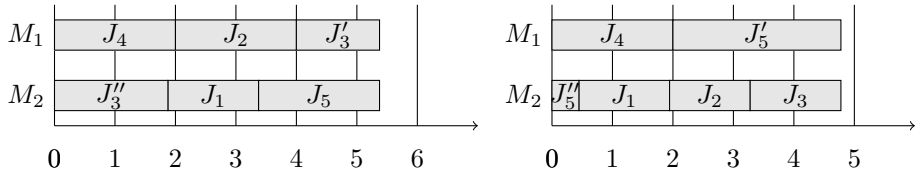


(a) Uszeregowanie σ, $C_{\max}(\sigma) = 3\frac{17}{24}$

(b) Optymalne uszeregowanie $\sigma^\star$, $C_{\max}(\sigma^\star) = 3\frac{7}{24}$

Rysunek 7.1: Algorytm heurystyczny $H1$, a algorytm dokładny

Najkrótsze uszeregowanie σ wygenerowane przez algorytm 7.1 przedstawia rysunek 7.2a, $C_{\max}(\sigma) = 5\frac{3}{8}$. Tymczasem uszeregowanie optymalne $\sigma^\star$ przedsta-



(a) Uszeregowanie σ, $C_{\max}(\sigma) = 5\frac{3}{8}$

(b) Uszeregowanie $\sigma^\star$, $C_{\max}(\sigma) = 4\frac{7}{9}$

Rysunek 7.2: Algorytm heurystyczny $H1$, a algorytm dokładny

wia rysunek 7.2b, $C_{\max}(\sigma) = 4\frac{7}{9}$. Kolejne kroki przydziału zadań do maszyn dla podzielonego zadania J_3 przedstawia rysunek 7.3. Na maszynie M_2 zadania przydzielane są od 2-giej pozycji, co symbolizuje czarny prostokąt.

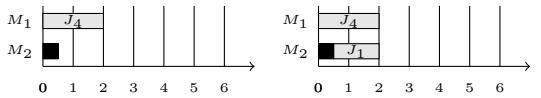
7.2 Algorytm heurystyczny H2

W podrozdziale tym przedstawimy drugi algorytm heurystyczny dla problemu PSLE (patrz algorytm 7.2). W tym algorytmie mając dane dzielone zadanie, zamiast rozpatrywać wszystkie możliwe uszeregowania pozostałych zadań, przydzielamy zadania do maszyny o najmniejszym obciążeniu.

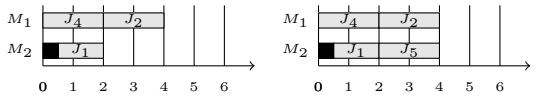
W pseudokodzie algorytmu heurystycznego użyliśmy funkcji przedstawionych w poprzednim rozdziale.

Ten wielomianowy algorytm heurystyczny przedstawia pseudokod 7.2.

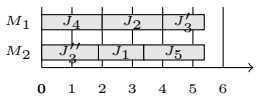
Dla podanej na wejściu początkowej sekwencja zadań, J, algorytm 7.2 generuje uszeregowanie przybliżone. W linii 1, funkcja `sortSPT` sortuje wszystkie zadania w porządku SPT. W linii 2, funkcja `Schedule` tworzy uszeregowanie bez podzielonego zadania, gdzie wszystkie zadania są uszeregowane na maszynie M_1. Tak skonstruowane uszeregowanie jest aktualnie najlepszym uszeregowaniem. Znając aktualnie najlepsze uszeregowanie, algorytm 7.2 wybiera w linii 4



(a) Przydzielone zadanie J_4 (b) Przydzielone zadanie J_1



(c) Przydzielone zadanie J_2 (d) Przydzielone zadanie J_5



(e) Przydzielone zadanie J_3

Rysunek 7.3: Kolejne kroki algorytmu $H1$, dla dzielonego zadania J_3

sekwencję $\underline{J}$ bez podzielonego zadania J_i. W liniach 5 i 6 algorytm tworzy dwie puste sekwencje J^1 i J^2. W linii 7, pętla `for` iteruje przez wszystkie zadania z sekwencji $\underline{J}$. W linii 8, w instrukcji warunkowej `if` sprawdzane jest czy sekwencja zadań $L_{[1]}(J^1)$ jest niewiększa od sekwencji zadań $L_{[2]}(J^2)$, i jeżeli tak jest, to w linii 9 rozpatrywane j-te zadanie z sekwencji $\underline{J}$ jest przydzielane do sekwencji J^1. W przeciwnym razie, w linii 11, j-te zadanie z sekwencji $\underline{J}$ jest przydzielane do sekwencji J^2. W linii 14, funkcja `Schedule` generuje uszeregowanie z podzielonym zadaniem J_i, gdzie zadania z sekwencji J^1 są uszeregowane na maszynie M_1 a zadania z sekwencji J^2 są uszeregowane na maszynie M_2. W liniach 15–16, algorytm sprawdza, czy długość nowego uszeregowania, R, jest mniejsza od aktualnie najlepszego uszeregowania Apr; jeśli jest mniejsza, Apr jest aktualizowane. Końcowe uszeregowanie, Apr, jest zwracane w linii 19.

Twierdzenie 7.2. *Algorytm 7.2 rozwiązuje problem PSLE w czasie $O(n^3)$.*

Dowód powyższego twierdzenia jest analogiczny do dowodu twierdzenia 7.1. Jedyna różnica występuje w linii 8 obu algorytmów. W algorytmie 7.1 sprawdzanie warunku podanego w tej linii wymaga czasu $O(1)$, natomiast w algorytmie 7.2 jest to czas $O(n)$, co jednak nie ma wpływu na rząd złożoności obu algorytmów.

Przedstawimy sposób działania algorytmu 7.2 na następujących przykładach.

Przykład 7.3. Mamy $n = 5$ zadań z podstawowymi czasami wykonywania $p_1 = 1$, $p_2 = 2$, $p_3 = 3$, $p_4 = 4$, $p_5 = 5$ i indeksem uczenia się $a = -1$. Najkrótsze uszeregowanie σ wygenerowane przez algorytm 7.2 przedstawia rysunek 7.4a, $C_{\max}(\sigma) = 3\frac{17}{24}$. Uszeregowanie optymalne $\sigma^\star$ przedstawia rysunek 7.4b, $C_{\max}(\sigma) = 3\frac{7}{24}$.

Algorytm 7.2: Algorytm heurystyczny $H2$ dla problemu PSLE

```
1  J̄ = sortSPT(J)
2  Apr = Schedule(J̄,(),NIL)
3  for i = 1 to n do
4  |  J_ = J̄ \ {J_i}
5  |  J^1 = ()
6  |  J^2 = ()
7  |  for j = 1 to |J_| do
8  |  |  if L_[1](J^1) ⩽ L_[2](J^2) then
9  |  |  |  J^1 = J^1 ∪ {J_[j](J_)}
10 |  |  else
11 |  |  |  J^2 = J^2 ∪ {J_[j](J_)}
12 |  |  end
13 |  end
14 |  R =Schedule(J^1,J^2,J_i)
15 |  if getCmax(R)<getCmax(Apr) then
16 |  |  Apr = R
17 |  end
18 end
19 return Apr
```

Przykład 7.4. Mamy dane $n = 5$ zadań z podstawowymi czasami wykonywania $p_1 = 3$, $p_2 = 4$, $p_3 = 6$, $p_4 = 2$ i $p_5 = 6$, i indeks uczenia się $a = -1$. Najkrótsze uszeregowanie σ wygenerowane przez algorytm 7.2 przedstawia rysunek 7.5a, $C_{\max}(\sigma) = 5\frac{23}{24}$. Uszeregowanie optymalne $\sigma^\star$ przedstawia rysunek 7.5b, $C_{\max}(\sigma) = 4\frac{7}{9}$. Kolejne kroki przydziału zadań do maszyn dla podzielonego zadania J_3 przedstawia rysunek. 7.6. Na maszynie M_2 zadania przydzielane są od 2-giej pozycji, czego nie zaznaczono na rysunku.

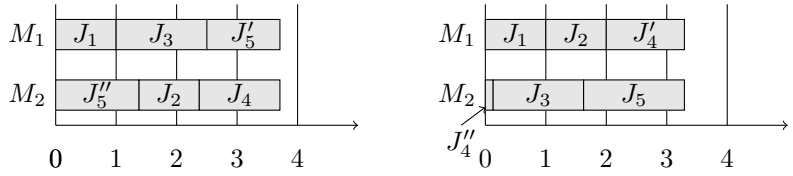
7.3 Wyniki eksperymentów obliczeniowych

W niniejszym podrozdziale przedstawimy wyniki eksperymentów obliczeniowych zestawione w taki sposób, aby można było porównać oba algorytmy heurystyczne, pod względem czasu wykonywania jak i jakości.

Aby porównać algorytmy heurystyczne między sobą oraz z algorytmem dokładnym, przeprowadziliśmy dwa eksperymenty obliczeniowe.

7.3.1 Porównanie średnich czasów działania

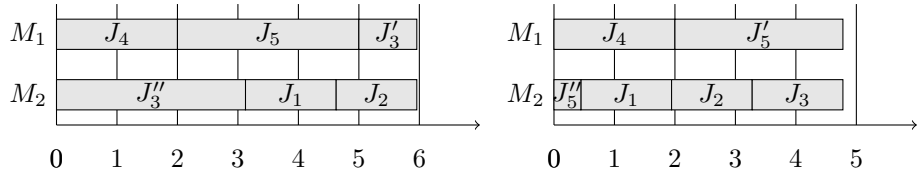
Celem pierwszego z eksperymentu było porównanie czasu działania obu algorytmów heurystycznych. W pierwszym eksperymencie przetestowaliśmy nasze algorytmy na losowych instancjach z liczbą zadań od 1000 do 20000 z krokiem 1000. Dla każdego n, wygenerowaliśmy 20 losowych instancji, w których pod-



(a) Uszeregowanie σ, $C_{\max}(\sigma) = 3\frac{17}{24}$

(b) Optymalne uszeregowanie $\sigma^\star$, $C_{\max}(\sigma^\star) = 3\frac{7}{24}$

Rysunek 7.4: Algorytm heurystyczny $H2$, a algorytm dokładny



(a) Uszeregowanie σ, $C_{\max}(\sigma) = 5\frac{23}{24}$

(b) Uszeregowanie $\sigma^\star$, $C_{\max}(\sigma) = 4\frac{7}{9}$

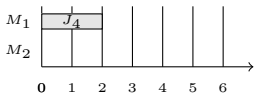
Rysunek 7.5: Algorytm heurystyczny $H2$, a algorytm dokładny

stawowe czasy wykonywania zadań są liczbami całkowitymi wylosowanymi z przedziału $[1, 10]$. W sumie przetestowano 400 instancji.

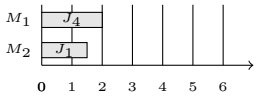
W instancjach wygenerowanych w tym eksperymencie nie występowały zadania dominujące, co można uzasadnić w następujący sposób. W instancji mogą występować zadania z podstawowym czasem wykonywania będącymi liczbami całkowitymi z przedziału $[1, 10]$. Czyli kandydatem na zadanie dominujące jest zadanie J_d, dla którego $p_d = 10$. Weźmy najbardziej pesymistyczną instancję, czyli taką w której istnieje jedno zadanie o podstawowym czasie wykonywania $p_d = 10$, a pozostałe zadania są zadaniami jednostkowymi. Dla takiej instancji wystarczy pokazać jedno uszeregowanie w którym nierówność 5.2 nie jest spełniona. Szukamy uszeregowania w którym wszystkie zadania poza kandydatem na zadanie dominujące są uszeregowane na maszynie M_1, a zadanie J_d szeregujemy na maszynie M_2. Wtedy zadanie J_d wykonuje się przez 5 jednostek (uszeregowane na drugiej pozycji zgodnie z nierównością 5.2), natomiast czas wykonania pozostałych zadań na pierwszej maszynie jest większy od 5, ponieważ zadań tych jest 9999, a dla $a = -1$ i 84 zadań jednostkowych obciążenie maszyny M_2 jest większe od 5 ($\sum_{i=1}^{84} \frac{1}{i} > 5$). Ponieważ pokazaliśmy, że dla najbardziej pesymistycznej instancji, istnieje uszeregowanie w którym nierówność 5.2 nie jest spełniona, to dla innych instancji $n > 85$ i $p_i \in [1, 10]$ również nie istnieje zadanie dominujące.

Wyniki tych eksperymentów podano w tabelach 7.1–7.2.

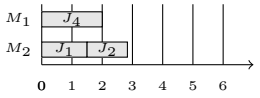
W tabeli 7.1, podano minimalne, średnie i maksymalne czasy obliczeń dla pierwszej heurystyki (patrz algorytm 7.1). W tabeli, symbole n, $t_{\min}$, t_{avg} i $t_{\max}$ oznaczają odpowiedni rozmiar instancji, minimalny, średni i maksymalny czas



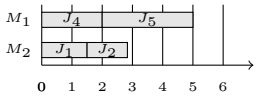
(a) Przydzielone zadanie J_4



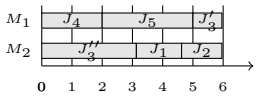
(b) Przydzielone zadanie J_1



(c) Przydzielone zadanie J_2



(d) Przydzielone zadanie J_5



(e) Przydzielone zadanie J_3

Rysunek 7.6: Kolejne kroki algorytmu $H2$, dla dzielonego zadania J_3

obliczeń.

W tabeli 7.2, podano minimalne, średnie i maksymalne czasy obliczeń dla drugiej heurystyki (patrz algorytm 7.2). W tabeli, symbole n, $t_{\min}$, t_{avg} i $t_{\max}$ oznaczają odpowiedni rozmiar instancji, minimalny, średni i maksymalny czas obliczeń.

Rysunek 7.7 przedstawia wykres średniego czasu działania obu heurystyk (patrz algorytm 7.1 i algorytm 7.2). Oś x reprezentuje liczbę zadań w instancji, natomiast oś y przedstawia czas wykonywania algorytmów wyrażony w sekundach.

Wyniki tego eksperymentu pokazują, że heurystyka $H1$ działa średnio szybciej niż heurystyka $H2$, czego należało się spodziewać, ponieważ heurystyka $H2$ wymaga po dodaniu każdego zadania do maszyny policzenia czasu wykonania wszystkich zadań uszeregowanych na tej maszynie, natomiast heurystyka $H1$ oprócz operacji dodawania sprawdza jedynie parzystość licznika pętli.

Tabela 7.1: Czasy wykonywania algorytmu 7.1

n	$t_{\min}$	t_{avg}	$t_{\max}$
1000	0.104	0.111	0.129
2000	0.417	0.439	0.464
3000	0.952	0.992	1.066
4000	1.677	1.714	1.770
5000	2.628	2.662	2.839
6000	3.805	3.885	4.085
7000	5.223	5.340	5.575
8000	6.707	6.793	7.316
9000	8.488	8.611	8.705
10000	10.493	10.563	11.091
11000	12.694	12.747	12.850
12000	15.226	15.383	15.532
13000	17.765	18.100	19.488
14000	20.849	21.624	26.637
15000	23.696	23.865	24.419
16000	27.123	27.512	27.746
17000	30.480	30.879	32.143
18000	34.154	34.829	36.412
19000	38.508	39.089	43.109
20000	42.609	43.295	45.194

7.3.2 Porównanie średniej jakości rozwiązań

Celem drugiego z eksperymentów było zmierzenie jakości rozwiązań wygenerowanych przez algorytmy heurystyczne. Aby tego dokonać, użyliśmy wzoru na błąd bezwzględny $b = |b_{opt} - b_H|$, gdzie b_{opt} jest długością uszeregowania optymalnego, a b_H jest długością uszeregowania wyznaczoną przez algorytm heurystyczny. W eksperymencie użyliśmy instancji z rozdziału 6, ponieważ dla tych instancji w rozdziale 6 zostały obliczone długości uszeregowania optymalnego. Instancje te nie zawierają zadań dominujących, ponieważ zostało to sprawdzone przez algorytm dokładny.

W tabeli 7.3, przedstawiono minimalny, średni i maksymalny błąd bezwzględny popełniony przez pierwszą heurystykę (patrz algorytm 7.1). W tabeli, symbole n, $b_{\min}$, b_{avg} i $b_{\max}$ oznaczają odpowiedni rozmiar instancji, minimalny, średni i maksymalny błąd bezwzględny.

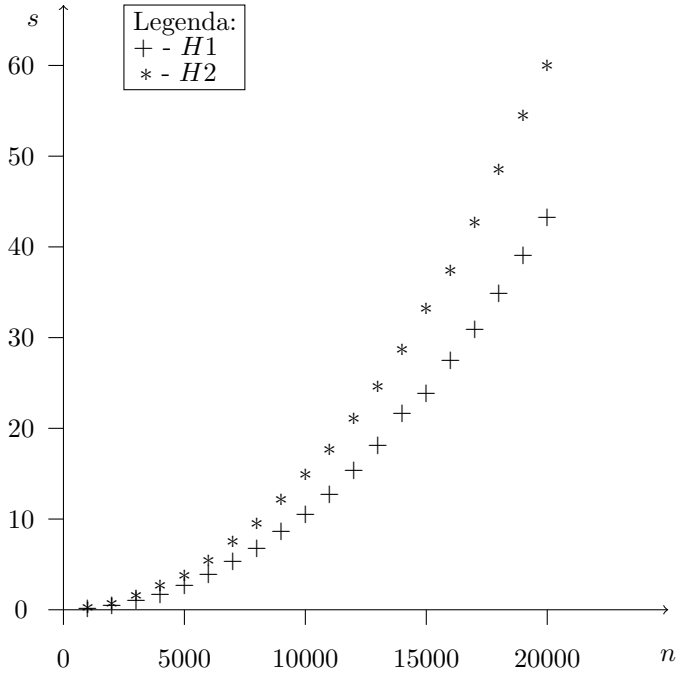
W tabeli 7.4, przedstawiono minimalny, średni i maksymalny błąd bezwzględny popełniony przez drugą heurystykę (patrz algorytm 7.2). W tabeli, symbole n, $b_{\min}$, b_{avg} i $b_{\max}$ oznaczają odpowiedni rozmiar instancji, minimalny, średni i maksymalny błąd bezwzględny.

Na rysunku 7.8, który przedstawia średni błąd bezwzględny obu heurystyk (patrz algorytm 7.1 i algorytm 7.2), oś x reprezentuje liczbę zadań w instancji, natomiast oś y przedstawia błąd bezwzględny wyrażony w jednostkach czasu.

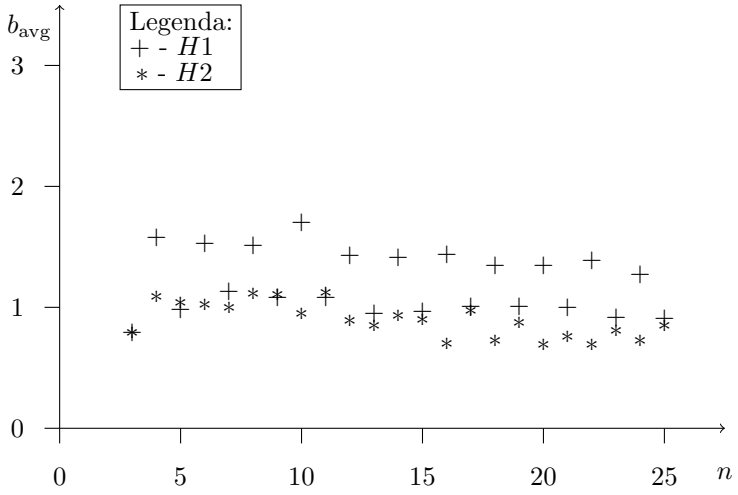
Tabela 7.2: Czasy wykonywania algorytmu 7.2

n	$t_{\min}$	t_{avg}	$t_{\max}$
1 000	0.155	0.173	0.209
2 000	0.640	0.691	0.821
3 000	1.439	1.493	1.585
4 000	2.509	2.627	2.990
5 000	3.643	3.771	3.974
6 000	5.226	5.355	5.499
7 000	7.133	7.469	8.125
8 000	9.316	9.528	9.778
9 000	11.777	12.089	12.796
10 000	14.616	14.895	15.085
11 000	17.558	17.635	17.884
12 000	20.917	21.047	21.417
13 000	24.572	24.602	24.713
14 000	28.493	28.612	29.789
15 000	32.726	33.216	39.475
16 000	37.241	37.310	37.522
17 000	42.037	42.634	44.020
18 000	47.351	48.483	49.507
19 000	52.683	54.411	56.510
20 000	58.687	59.916	60.949

Wyniki tego eksperymentu pokazują, że heurystyka $H2$ generuje wyniki bliższe optymalnym, niż heurystyka $H1$. Zauważmy, że dla heurystyki $H1$ dla instancji o parzystej liczbie zadań średni błąd bezwzględny jest gorszy o około $\frac{1}{2}$ jednostki czasu od średniego błędu bezwzględnego dla instancji o liczbie zadań nieparzystych. Anomalia taka nie występuje w przypadku heurystyki $H2$, co wskazuje czym należy się kierować przy konstrukcji lepszych heurystyk.



Rysunek 7.7: Średni czas działania obu heurystyk



Rysunek 7.8: Średni błąd bezwzględny obu heurystyk

Tabela 7.3: Błąd bezwzględny algorytmu 7.1

n	błąd bezwzględny		
	$b_{\min}$	b_{avg}	$b_{\max}$
3	0.000	0.796	2.333
4	0.542	1.576	2.958
5	0.000	0.979	2.033
6	0.800	1.530	2.283
7	0.156	1.129	2.450
8	0.838	1.509	2.140
9	0.577	1.086	1.805
10	0.986	1.703	3.005
11	0.566	1.077	2.191
12	0.982	1.434	1.991
13	0.434	0.951	1.492
14	1.049	1.410	2.191
15	0.603	0.968	1.790
16	1.031	1.437	2.734
17	0.533	1.015	1.684
18	0.928	1.349	1.874
19	0.608	1.011	1.591
20	0.961	1.347	1.883
21	0.632	0.999	1.358
22	1.001	1.393	1.927
23	0.627	0.916	1.639
24	0.866	1.275	2.054
25	0.555	0.908	1.240

Tabela 7.4: Błąd bezwzględny algorytmu 7.2

n	błąd bezwzględny		
	$b_{\min}$	b_{avg}	$b_{\max}$
3	0.000	0.796	2.333
4	0.000	1.094	1.764
5	0.111	1.036	2.167
6	0.376	1.024	1.933
7	0.156	0.997	1.792
8	0.212	1.110	1.927
9	0.438	1.102	1.791
10	0.344	0.951	1.679
11	0.175	1.121	1.828
12	0.138	0.895	1.438
13	0.294	0.853	1.653
14	0.330	0.926	1.543
15	0.361	0.897	1.453
16	0.196	0.703	1.729
17	0.403	0.973	1.350
18	0.108	0.717	1.509
19	0.155	0.868	1.542
20	0.174	0.689	1.335
21	0.179	0.764	1.335
22	0.364	0.690	1.305
23	0.392	0.806	1.151
24	0.265	0.718	1.259
25	0.257	0.854	1.322

Rozdział 8

Podsumowanie

W niniejszym rozdziale podsumowujemy wyniki badań zawartych w rozprawie oraz przedstawiamy dalsze kierunki badań.

8.1 Główne wyniki rozprawy

W rozprawie otrzymano następujące wyniki:

- sformułowano pewien dwu-maszynowy problem szeregowania zadań z kryterium $C_{\max}$ oraz ze zmiennymi czasami wykonywania zależnymi od numerów pozycji zadań w uszeregowaniu;
- opracowano definicję pewnego rodzaju podzielności zadań dla wyżej wymienionego problemu;
- zbadano podstawowe własności wyżej wymienionego problemu;
- podano przykład wielomianowo rozwiązywalnego szczególnego przypadku badanego problemu;
- opracowano wyliczeniowy algorytm dokładny oraz algorytm dokładny podziału i ograniczeń dla badanego problemu;
- opracowano dwa algorytmy przybliżone dla badanego problemu;
- przeprowadzono wstępne eksperymenty numeryczne z implementacjami wyżej wymienionych algorytmów.

8.2 Sposoby prezentacji wyników rozprawy

Główne wyniki rozprawy prezentowano na następujących międzynarodowych konferencjach:

- The 28th European Conference on Operational Research (EURO 2016, Poznań, lipiec 2016);
- The 20th International Conference on Operations Research (OR 2017, Berlin, wrzesień 2017).
- The Second International Workshop on Dynamic Scheduling Problems (IWDSP 2018, Poznań, czerwiec 2018);
- The 23rd International Conference on Methods and Models in Automation and Robotics (MMAR 2018, Międzyzdroje, sierpień 2018)

Główne wyniki badań zostały opublikowane w następujących pracach:

- (Żurowski (2018a)) „Preemptive scheduling of jobs with a learning effect on two parallel machines". W *Operations Research Proceedings 2017*, strony 587–592. Springer;
- (Żurowski (2018b)) „Some remarks on preemptive scheduling of jobs with a learning effect". W *Extended abstracts of The Second International Workshop on Dynamic Scheduling Problems (IWDSP Poznań, June 26-28, 2018)*, strony 95–98. Polish Mathematical Society;
- (Żurowski i Gawiejnowicz (2018)) „An enumerative algorithm for a two-machine preemptive job scheduling with a learning effect". W *Proceedings of the 23rd International Conference on Methods and Models in Automation and Robotics (MMAR Międzyzdroje, August 26-29, 2018)*, strony 926–931.

Praca „Scheduling preemptable position-dependent jobs on two parallel identical machines", zawierająca główne wyniki z rozdziału 5, została wysłana do druku.

8.3 Kierunki dalszych badań

Wyniki rozprawy mogą stanowić wstęp do dalszych badan, wśród których najistotniejsze wydaja się następujące:

- zbadanie złożoności ogólnego problemu;
- zbadanie problemu pod kątem modyfikacji definicji podzielności;
- ulepszenie algorytmu dokładnego, poprzez znalezienie ograniczenia dolnego na długość uszeregowania;
- ulepszenie algorytmów heurystycznych.

Bibliografia

Agnetis, A., Billaut, J., Gawiejnowicz, S., Pacciarelli, D., i Soukhal, A. (2014). *Multiagent Scheduling: Models and Algorithms*. Springer.

Azzouz, A., Ennigrou, M., i Said, L. B. (2018). Scheduling problems under learning effects: classification and cartography. *International Journal of Production Research*, 56(4):1642–1661.

Błażewicz, J. (1988). *Złożoność obliczeniowa problemów kombinatorycznych*. Wydawnictwa Nukowo-Techniczne.

Błażewicz, J., Ecker, K. H., Pesch, E., Schmidt, G., i Węglarz, J. (1996). *Scheduling Computer and Manufacturing Processes*. Springer.

Baptiste, P., Carlier, J., Kononov, A., Queyranne, M., Sevastyanov, S., i Sviridenko, M. (2012). Integer preemptive scheduling on parallel machines. *Operations Research Letters*, 40:440–444.

Barański, T. (2011). Task scheduling with restricted preemptions. W *Proceedings of the Federated Conference on Computer Science and Information Systems (FedCSIS)*, strony 231–238.

Coffman jr, E. G., edytor (1980). *Teoria szeregowania zadań*. Wydawnictwa Nukowo-Techniczne.

Coffman jr, E. G. i Even, S. (1997). A note on limited preemption. *Parallel Processing Letters*, 8:3–6.

Conway, R. W., Maxwell, W. L., i Miller, L. W. (1967). *Theory of Scheduling*. Addison-Wesley. reprint Dover Publications 2003.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., i Clifford, S. (2017). *Wprowadzenie do algorytmów*. Wydawnictwo Naukowe PWN.

Ecker, K. i Hirschberg, R. (1993). A note on limited preemption. *Task scheduling with restricted preemptions*, 694:464–475.

Gawiejnowicz, S. (1996). A note on scheduling on a single processor with speed dependent on a number of executed jobs. *Information Processing Letters*, 57:297–300.

Gawiejnowicz, S. (2008). *Time-Dependent Scheduling*. Springer.

Grabowski, J., Nowicki, E., i Smutnicki, C. (Warszawa 2003). Metoda blokowa w zagadnieniach szeregowania zadań.

Jiang, Y., Dong, J., i Ji, M. (2013). Preemptive scheduling on two parallel machines with a single server. *Computers & Industrial Engineering*, 66:514–518.

Jiang, Y., Weng, Z., i Hu, J. (2012). Algorithms with limited number of preemptions for scheduling on parallel machines. *Lecture Notes in Computer Science*, 7285:93–104.

Julien, F. M., Magazine, M. J., i Hall, N. G. (1997). Generalized preemption models for single-machine dynamic scheduling problems. *IIE Transactions*, 29:359–372.

Kubiak, W., Błażewicz, J., Formanowicz, P., Breit, J., i G.Schmidt (2002). Two-machine flow shops with limited machine availability. *European Journal of Operational Research*, 136:528–540.

Leung, J. Y.-T., edytor (2004). *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. Chapman & Hall/CRC.

Liu, C., Wang, C., Zhang, Z.-H., i Zheng, L. (2018). Scheduling with job-splitting considering learning and the vital-few law. *Computers & Operations Research*, 90:264–274.

Liu, Z. i Cheng, T. E. (2002). Scheduling with job release dates, delivery times and preemption penalties. *Information Processing Letters*, 82:107–111.

McNaughton, R. (1959). Scheduling with deadlines and loss functions. *Management Science*, 6:1–12.

Morton, T. E. i Pentico, D. W. (1993). *Heuristic Scheduling Systems: With Applications to Production Systems and Project Management*. John Wiley & Sons.

Mosheiov, G. (2001). Scheduling problems with a learning effect. *European Journal of Operational Research*, 132:687–693.

Okołowski, D. i Gawiejnowicz, S. (2010). Exact and heuristic algorithms for parallel-machine scheduling with dejong's learning effect. *Computers & Industrial Engineering*, 59:272–279.

Pieńkosz, K. i Prus, A. (2015). Task scheduling with restricted preemptions on two parallel processors. W *20th International Conference on Methods and Models in Automation and Robotics (MMAR)*, strony 58–61.

Pinedo, M. (2016). *Scheduling: Theory, Algorithms and Systems*. Springer.

Shim, S.-O. i Kim, Y.-D. (2008). A branch and bound algorithm for an identical parallel machine scheduling problem with a job splitting property. *Computers & Operations Research*, 35:863–875.

Soper, A. i Strusevich, V. (2018a). Scheduling with a single preemption on uniform parallel machines. *Discrete Applied Mathematics*.

Soper, A. J. i Strusevich, V. S. (2018b). Parametric analysis of the quality of single preemption schedules on three uniform parallel machines. *Annals of Operations Research*.

Żurowski, M. (2018a). Preemptive scheduling of jobs with a learning effect on two parallel machines. W *Operations Research Proceedings 2017*, strony 587–592. Springer.

Żurowski, M. (2018b). Some remarks on preemptive scheduling of jobs with a learning effect. W *Extended abstracts of The Second International Workshop on Dynamic Scheduling Problems (IWDSP Poznań, czerwiec 26-28, 2018)*, strony 95–98. Polish Mathematical Society.

Żurowski, M. i Gawiejnowicz, S. (2018). An enumerative algorithm for a two-machine preemptive job scheduling with a learning effect. W *Proceedings of the 23rd International Conference on Methods and Models in Automation and Robotics (MMAR Międzyzdroje, sierpień 26-29, 2018)*, strony 926–931.

Woeginger, G. (2003). Exact algorithms for np-hard problems: A survey. *Lecture Notes in Computer Science*, 2570:185–207.

Xing, W. i Zhang, J. (2000). Parallel machine scheduling with splitting jobs. *Discrete Applied Mathematics*, 103:259–269.

Zweben, M. i Fox, M. S., edytorzy (1994). *Intelligent Scheduling*. Morgan Kaufmann Publishers.

Spis rysunków

Spis tabel

Spis algorytmów

Spis symboli