Uniwersytet im. Adama Mickiewicza w Poznaniu
Wydział Matematyki i Informatyki

Marek Dębczyński

Jednomaszynowe problemy szeregowania zadań zależnych z mieszanymi czasami wykonywania

Rozprawa doktorska
napisana pod kierunkiem
dra hab. Stanisława Gawiejnowicza

Poznań 2013

Składam serdeczne podziękowania mojemu promotorowi
dr. hab. Stanisławowi Gawiejnowiczowi
za wskazanie tematu, wyrozumiałość, poświęcony czas
i życzliwą opiekę w trakcie pisania rozprawy.

Spis treści

Rozdział 1. Wstęp 3

1.1. Tematyka rozprawy 4
1.2. Cel rozprawy 5
1.3. Układ rozprawy 6

Rozdział 2. Preliminaria 8

2.1. Algorytmika 8
2.2. Teoria grafów 10
2.3. Teoria szeregowania zadań 11

Rozdział 3. Przegląd wybranej literatury 17

3.1. Zadania ze stałymi czasami wykonywania 17
3.2. Zadania ze zmiennymi czasami wykonywania 18

Rozdział 4. Rodzina problemów postaci
$1|p_{j,r}(t) \in \{p_1, n-k; p_2, k\}, prec|f_{\max}$ 25

4.1. Sformułowanie problemów rodziny 25
4.2. Zadania ze zmiennymi czasami wykonywania z tej samej rodziny . . 26
4.3. Zadania ze zmiennymi mieszanymi czasami wykonywania 31

Rozdział 5. Rodzina problemów postaci
$1|p_{j,r}(t) \in \left\{p^1_{j,r}(t), n_1; \ldots; p^k_{j,r}(t), n_k\right\}, k\text{–}part|f_{\max}$ 54

5.1. Sformułowanie problemów rodziny 54
5.2. Algorytm dla k–dzielnych ograniczeń kolejnościowych 55

Rozdział 6. Rodzina problemów postaci
$1|p_{j,r}(t) \in \{a_j, n_1; b_j * t, n_2; a_j + b_j * t, n_3\}, prec|f_{\max}$ 63

6.1. Sformułowanie problemów rodziny 63
6.2. Algorytmy dokładne 64
6.3. Algorytmy heurystyczne 70
6.4. Eksperyment numeryczny 88

Rozdział 7. Podsumowanie 90

Bibliografia 93

Spis rysunków 96

Spis tabel 97

Rozdział 1

Wstęp

Teoria szeregowania zadań to dział informatyki, w którym rozważa się problemy polegające na wyznaczeniu przydziałów wykonywanych zadań do maszyn (procesorów) w czasie. Każdy taki przydział zadań, dopuszczalny w sensie przyjętych założeń, jest nazywany uszeregowaniem. Jakość uszeregowania jest oceniana za pomocą różnych kryteriów optymalności.

Dowolny problem szeregowania zadań można sformułować podając parametry opisujące zbiór zadań, zbiór maszyn (procesorów) oraz postać kryterium optymalności. Podstawowymi parametrami opisującymi dowolny zbiór zadań są czasy wykonywania zadań, wagi, czasy gotowości oraz pożądane terminy zakończenia zadań. Ponadto między zadaniami mogą występować ograniczenia kolejnościowe określające częściowy porządek na zbiorze zadań, opisane za pomocą pewnego acyklicznego grafu skierowanego.

W klasycznej teorii szeregowania zadań ([4], [19]) rozważano jedynie takie problemy, w których czasy wykonywania zadań były stałe. Dopiero od kilkunastu lat zaczęto zajmować się problemami ze zmiennymi czasami wykonywania zadań ([2], [10]). Najbardziej popularnymi modelami w tej grupie są modele czasowo-zależne i modele, w których rozpatruje się efekt uczenia się bądź starzenia się zadań. W modelach czasowo-zależnych zakładamy, że czas wykonywania zadania jest opisany przez funkcję, różną od stałej, zależną od czasu rozpoczęcia tego zadania [10]. To założenie pozwala na rozpatrywanie problemów silnie uwarunkowanych czasowo, w których zadania wykonywane później mają dłuższe lub krótsze czasy wykonywania niż te wykonywane wcześniej, np. w zagadnieniach organizacji prac konserwacyjnych, pewnych zagadnieniach militarnych, bądź finansowych. W modelach

szeregowania uwzględniających efekt uczenia się bądź starzenia się zadań zakładamy, że czas wykonywania zadania jest opisany przez funkcję różną od stałej, zależną od pozycji tego zadania w uszeregowaniu [2]. To założenie pozwala na rozpatrywanie problemów, w których czas wykonywania zadania może odpowiednio maleć w zależności od tempa uczenia się pracownika lub rosnąć w efekcie zużywania się narzędzi ([1], [18]).

Problemy tego typu często występują w praktyce. Rozważmy np. problem realizacji dużego projektu programistycznego. Firma informatyczna realizuje nowy projekt złożony z etapów, pomiędzy którymi występują ograniczenia kolejnościowe. Każdy z etapów składa się z zadań różnego typu. Źródłem finansowania prac nad projektem będzie kredyt bankowy wypłacany w ratach odpowiadających fakturom wystawionym po realizacji każdego z zadań. Bank wyrazi zgodę na kredytowanie projektu jeżeli rata kredytu nie będzie za wysoka. Kierownik projektu ma za zadanie skonstruować takie uszeregowanie zadań i kosztów, które zostanie zaakceptowane przez bank. Musi on zatem wybrać takie uszeregowanie zadań aby zminimalizować maksymalny jednostkowy koszt zadania.

1.1. Tematyka rozprawy

W rozprawie będą rozważane problemy szeregowania zadań, w których zbiór zadań zawiera zadania z różnymi postaciami czasów wykonywania, między zadaniami występują niepuste ograniczenia kolejnościowe oraz jakość uszeregowań oceniana jest za pomocą kryterium maksymalnego kosztu $f_{\max}$. W dotychczasowych pracach nad problemami szeregowania zadań ze zmiennymi czasami wykonywania brak jest prac opisujących problemy, w których występują jednocześnie ww. założenia, a autorzy rozważali jedynie następujące szczególne przypadki takich problemów.

Po pierwsze w dotychczasowych badaniach dotyczących szeregowania zadań ze zmiennymi czasami wykonywania przyjmowano, że w zbiorze zadań dopuszczalne są jedynie zadania z czasami wykonywania tylko jednej postaci. To założenie jest silnie ograniczające, ponieważ istnieje wiele problemów,

w których czasy wykonywania zadań są opisane funkcjami różnych typów. Jednym z modeli, w których rozważano zmienne czasy wykonywania zadań różnych postaci, jest model z mieszanymi czasami wykonywania zadań [12].

Po drugie w badaniach problemów szeregowania ze zmiennymi czasami wykonywania przyjmowano zazwyczaj założenie, iż nie występują ograniczenia kolejnościowe. Jedynie kilku autorów rozważało różne modele z niepustymi ograniczeniami kolejnościowymi ([10, Rozdział 13], [11], [21], [22]). Dodanie niepustych ograniczeń kolejnościowych jest istotne ponieważ zwiększa obszar stosowalności omawianych modeli. Ponadto ograniczenia kolejnościowe są często spotykane w praktyce ze względu na wymagania technologiczne bądź montażowe.

Po trzecie w analizowanych problemach rozpatrywano kryteria będące szczególnymi przypadkami kryterium maksymalnego kosztu $f_{\max}$, tj. $C_{\max}$ oraz $L_{\max}$ oznaczające odpowiednio czas zakończenia ostatniego zadania oraz maksymalną nieterminowość ([10, Rozdział 13], [11], [21]).

1.2. Cel rozprawy

Celem rozprawy jest opracowanie wielomianowych algorytmów konstruujących optymalne bądź suboptymalne uszeregowania dla trzech następujących rodzin jednomaszynowych problemów szeregowania zadań.

Pierwsza rodzina dotyczy problemów szeregowania zadań ze zmiennymi mieszanymi czasami wykonywania, zależnymi od czasu rozpoczęcia wykonywania zadania lub jego pozycji w uszeregowaniu, dowolnymi ograniczeniami kolejnościowymi między zadaniami oraz kryterium maksymalnego kosztu $f_{\max}$. W tym przypadku zbiór zadań złożony jest z zadań, których czasy wykonywania są opisane funkcjami z dwóch różnych rodzin.

Druga rodzina dotyczy problemów szeregowania zadań ze zmiennymi mieszanymi czasami wykonywania, zależnymi od czasu rozpoczęcia wykonywania zadania lub jego pozycji w uszeregowaniu, k–dzielnymi ograniczeniami kolejnościowymi między zadaniami oraz kryterium maksymalnego kosztu $f_{\max}$. Przy takich ograniczeniach kolejnościowych zbiór zadań ze zmiennymi mie-

szanymi czasami wykonywania może zawierać zadania z więcej niż tylko dwoma różnymi postaciami czasów wykonywania.

Trzecia rodzina dotyczy z kolei problemów szeregowania zadań ze zmiennymi mieszanymi czasami wykonywania, zależnymi od czasu rozpoczęcia wykonywania zadania, dowolnymi ograniczeniami kolejnościowymi i kryterium maksymalnego kosztu $f_{\max}$. W tym przypadku zbiór zadań złożony jest z zadań, których czasy wykonywania są opisane funkcjami stałymi, proporcjonalnymi oraz liniowymi.

1.3. Układ rozprawy

Rozprawa składa się z 7 rozdziałów o następującej zawartości:

Rozdział 1 zawiera wstęp do rozprawy, omówienie jej tematyki, cel rozprawy oraz jej układ.

Rozdział 2 zawiera opis podstawowych konwencji stosowanych w rozprawie, podstawowych definicji związanych z algorytmiką i teorią grafów oraz podstawowych definicji, oznaczeń i lematów pomocniczych związanych z teorią szeregowania zadań.

Rozdział 3 zawiera przegląd wybranej literatury tematycznie związanej z problemami szeregowania zadań występującymi w rozprawie.

Rozdział 4 zawiera opis wielomianowych algorytmów znajdujących optymalne uszeregowania dla jednomaszynowych problemów szeregowania zadań ze zmiennymi mieszanymi czasami wykonywania, zależnymi od czasu rozpoczęcia wykonywania zadania lub jego pozycji w uszeregowaniu, z dowolnymi ograniczeniami kolejnościowymi oraz kryterium maksymalnego kosztu $f_{\max}$.

Rozdział 5 zawiera opis wielomianowego algorytmu znajdującego optymalne uszeregowania dla jednomaszynowych problemów szeregowania zadań ze zmiennymi mieszanymi czasami wykonywania zależnymi od czasu rozpoczęcia wykonywania zadania lub jego pozycji w uszeregowaniu, z k–dzielnymi ograniczeniami kolejnościowymi oraz kryterium maksymalnego kosztu $f_{\max}$.

Rozdział 6 zawiera omówienie jednomaszynowych problemów szeregowania zadań ze zmiennymi mieszanymi czasami wykonywania zależnymi od

czasu rozpoczęcia wykonywania zadania, z dowolnymi ograniczeniami kolejnościowymi oraz kryterium maksymalnego kosztu $f_{\max}$. Ze względu na NP-trudność tego problemu, w rozdziale zaprezentowane zostaną dwie wersje algorytmów dokładnych, kilkanaście konstrukcyjnych algorytmów heurystycznych i hybrydowy algorytm heurystyczny.

Rozdział 7 zawiera podsumowanie uzyskanych wyników oraz wskazanie potencjalnych dalszych kierunków badań.

Rozdział 2

Preliminaria

W tym rozdziale zaprezentowane zostaną podstawowe definicje i oznaczania używane w całej rozprawie. W podrozdziale 2.1 zaprezentowane zostaną podstawowe definicje związane z algorytmiką. W podrozdziale 2.2 zaprezentowane zostaną podstawowe definicje związane z teorią grafów. W podrozdziale 2.3 zaprezentowane zostaną podstawowe definicje, oznaczenia i lematy związane z teorią szeregowania zadań.

W rozprawie stosowane będą następujące konwencje: ze względu na to, iż maszyny odnoszą się do szerszej grupy zagadnień niż procesory, w rozprawie omawiane będą problemy maszynowe. W każdym rozdziale będą oddzielnie numerowane twierdzenia, przykłady, tabele i rysunki. Przykładowo: Twierdzenie 3.5 oraz Przykład 4.1 oznaczają odpowiednio piąte twierdzenie w rozdziale trzecim oraz pierwszy przykład w rozdziale czwartym. Koniec dowodu będzie oznaczany symbolem □. Zakończenie przykładu będzie oznaczone przez poziomą linię oddzielającą przykład od tekstu w nowym akapicie. Pozostałe oznaczenia będą definiowane w miejscu ich pierwszego wystąpienia.

2.1. Algorytmika

Problemy szeregowania zadań są rozwiązywane za pomocą algorytmów szeregowania zadań. Algorytm szeregowania zadań składa się z wejścia, sekwencji kroków i wyjścia. Jeżeli na wejściu algorytmu w miejsce parametrów podstawimy wartości liczbowe, to otrzymamy pewną instancję problemu. Poszczególne kroki algorytmu prowadzą do znalezienia uszeregowania dla da-

nej instancji problemu. Każdy krok z kolei może zawierać skończoną liczbę pewnych operacji arytmetycznych, logicznych, przypisań wartości do zmiennych, pętli, instrukcji warunkowych bądź funkcji, które są złożone z pewnych ciągów elementarnych operacji. Algorytm szeregowania zadań dla dowolnej instancji badanego problemu po skończonej liczbie kroków generuje na wyjściu uszeregowanie. Dla danego problemu może istnieć wiele algorytmów. Ze względu na to, że algorytmy te różnią się liczbą operacji prowadzących do znalezienia uszeregowania, wyróżnia się następujące podstawowe klasy algorytmów.

Algorytm nazywamy wielomianowym, jeżeli liczba jego operacji dominujących dla dowolnej instancji jest rzędu $O\left(p\left(k\right)\right)$, gdzie p jest pewnym wielomianem a k rozmiarem instancji. Algorytmy, które nie są wielomianowe nazywamy wykładniczymi. Pomiędzy tymi klasami algorytmów znajdują się algorytmy pseudowielomianowe. Algorytm nazywamy pseudowielomianowym, jeżeli liczba operacji dominujących dla dowolnej instancji jest rzędu $O\left(p\left(k, M\right)\right)$, gdzie p jest pewnym wielomianem zależnym od dwóch zmiennych k oraz M oznaczających odpowiednio rozmiar instancji i największą liczbę w niej występującą.

Uszeregowania generowane przez algorytmy szeregowania zadań mogą, lecz nie muszą być optymalne. Algorytmy szeregowania zadań, które zwracają uszeregowanie optymalne nazywamy algorytmami dokładnymi. Algorytmy dokładne mogą być zarówno wielomianowe jak i wykładnicze. Czasami znalezienie optymalnego uszeregowania jest możliwe tylko i wyłącznie po sprawdzeniu wszystkich możliwych wariantów. Jest to zazwyczaj proces bardzo kosztowny i czasochłonny, przez co takie podejście często jest nie do przyjęcia. W takich przypadkach możliwe jest skonstruowanie algorytmów heurystycznych, które nie dają gwarancji uzyskania optymalnego uszeregowania, lecz generują dopuszczalne uszeregowania w wielomianowym czasie. Algorytm heurystyczny będziemy nazywać hybrydowym, jeżeli zwracane przez niego uszeregowanie jest najlepszym uszeregowaniem spośród uszeregowań generowanych przez co najmniej dwa algorytmy heurystyczne.

Klasyfikacji problemów szeregowania zadań można dokonać dzięki dwóm

klasom złożoności obliczeniowej. Wszystkie problemy szeregowania zadań, które są rozwiązywalne w wielomianowym czasie przez deterministyczną maszynę Turinga należą do klasy P. Wszystkie problemy szeregowania zadań, które są rozwiązywalne w co najwyżej wielomianowym czasie przez niedeterministyczną maszynę Turinga, należą do klasy NP. Rozwiązania problemów szeregowania zadań należących do klasy NP są weryfikowalne w czasie wielomianowym, lecz nie istnieją dla takich problemów wielomianowe algorytmy, o ile $P \neq NP$. Wykorzystywane w rozprawie pojęcia i podstawowe fakty związane z teorią NP-zupełności będą stosowane w oparciu o konwencje przedstawione w książkach ([9], [10]).

2.2. Teoria grafów

W rozprawie będziemy zakładali, że między zadaniami występują niepuste ograniczenia kolejnościowe opisane za pomocą pewnych acyklicznych grafów skierowanych, których definicję przypominamy poniżej.

Grafem ograniczeń kolejnościowych nazywać będziemy acykliczny graf skierowany $G = (V(G), A(G))$, w którym $V(G)$ oraz $A(G)$ oznaczają odpowiednio zbiór wierzchołków oraz zbiór łuków. Poniżej zaprezentowane zostały definicje pełnych acyklicznych grafów skierowanych, które będą występowały w rozdziale 5.

Definicja 2.1. *(Acykliczny k–dzielny graf skierowany)*
Acykliczny graf skierowany G nazywamy k-dzielnym, jeżeli $V(G) = \bigcup_{i=1}^{k} V_i$, $V_l \cap V_m = \emptyset$ i $A(G) \subseteq \bigcup_{i=1}^{k-1} \bigcup_{j=i+1}^{k} V_i \times V_j$, gdzie $1 \leqslant l \neq m \leqslant k$.

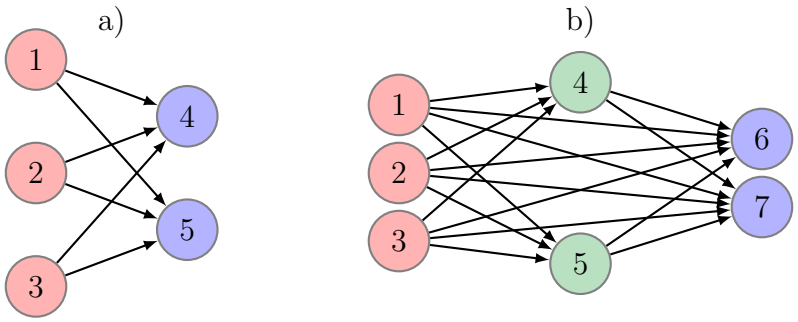
Definicja 2.2. *(Pełny acykliczny k–dzielny graf skierowany)*
Acykliczny k–dzielny graf skierowany G nazywamy pełnym, jeżeli zbiór łuków $A(G) = \bigcup_{i=1}^{k-1} \bigcup_{j=i+1}^{k} V_i \times V_j$.

Szczególnym przypadkiem acyklicznego k-dzielnego grafu skierowanego, dla $k = 2$, jest acykliczny dwudzielny graf skierowany zdefiniowany w następujący sposób:

Definicja 2.3. *(Acykliczny dwudzielny graf skierowany)*
Acykliczny graf skierowany G nazywamy dwudzielnym, jeżeli $V(G) = V_1 \cup V_2$, $V_1 \cap V_2 = \emptyset$ i $A(G) \subseteq V_1 \times V_2$.

Definicja 2.4. *(Pełny acykliczny dwudzielny graf skierowany)*
Acykliczny dwudzielny graf skierowany G nazywamy pełnym, jeżeli zbiór łuków $A(G) = V_1 \times V_2$.

Na Rysunku 2.1 zaprezentowane zostały przykłady pełnego acyklicznego dwudzielnego grafu skierowanego oraz pełnego acyklicznego k-dzielnego grafu skierowanego. Tymi samymi kolorami oznaczono zadania, w których czasy wykonywania zadań opisane są funkcjami z tej samej rodziny.



Rysunek 2.1. Przykłady pełnych acyklicznych grafów skierowanych: a) dwudzielny b) k-dzielny

2.3. Teoria szeregowania zadań

Opisując rozważane w rozprawie problemy szeregowania zadań będziemy posługiwać się następującą notacją:

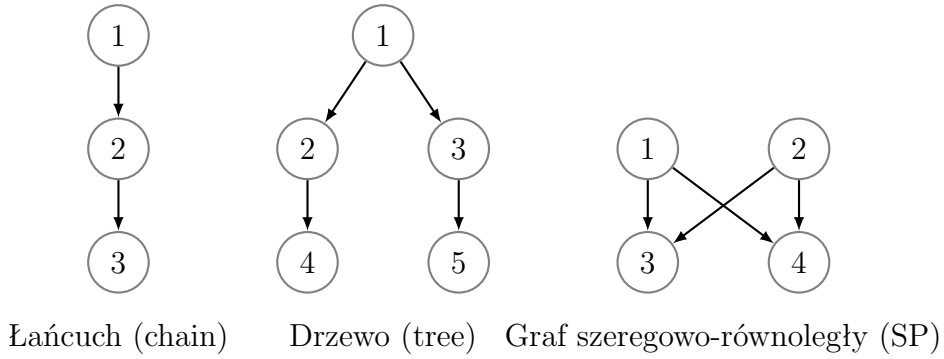
Zbiór zadań oznaczamy $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$. Z każdym zadaniem J_j, dla $1 \leqslant j \leqslant n$ powiązany jest czas wykonywania zadania oznaczany przez $p_{j,r}(t)$, gdzie $r \in \{1, 2, \ldots, n\}$ i $t \geqslant 0$ oznaczają odpowiednio pozycję zadania w uszeregowaniu i czas rozpoczęcia zadania.

Zbiór maszyn, dzięki którym możliwe jest zrealizowanie zadań, oznaczamy $\mathcal{M} = \{M_1, M_2, \ldots, M_m\}$. W rozprawie rozpatrujemy jednomaszynowe problemy szeregowania zadań, stąd $m = 1$. Oznacza to, że w rozprawie ana-

lizowane będą takie problemy szeregowania zadań, w których każde zadanie ze zbioru $\mathcal{J}$ składa się z jednej operacji. Ponadto zakładamy, że zadania są niepodzielne, co oznacza, że rozpoczęte zadanie nie może zostać przerwane.

Zbiór zadań może być opisany ponadto przy użyciu takich parametrów jak: r_j, w_j oraz d_j, oznaczających odpowiednio czas gotowości, wagę oraz pożądany termin zakończenia zadania J_j. Jeżeli w sformułowaniu problemu nie podano wartości tych parametrów, oznacza to, że $r_j = 0$, $w_j = 1$ oraz $d_j = +\infty$ dla dowolnego zadania $J_j \in \mathcal{J}$.

W rozprawie przyjmujemy, że między zadaniami mogą występować ograniczenia kolejnościowe, opisujące częściowy porządek na zbiorze zadań. Ograniczenia kolejnościowe opisane są przy pomocy pewnego acyklicznego grafu skierowanego $G = (V, A)$, gdzie V oraz A oznaczają odpowiednio zbiór wierzchołków oraz zbiór łuków. Przykłady ograniczeń kolejnościowych pokazane zostały na Rysunku 2.2.



Rysunek 2.2. Przykładowe postacie ograniczeń kolejnościowych

W rozdziałach 4 oraz 6 będą występować dowolne ograniczenia kolejnościowe między zadaniami. Z kolei w rozdziale 5 między zadaniami będą występować k-dzielne ograniczenia kolejnościowe.

Rozwiązując dowolny problem szeregowania zadań poszukujemy takiego przydziału zadań do maszyny w czasie, aby zminimalizować przyjęte kryterium optymalności. Każdy taki przydział nazywamy uszeregowaniem.

Definicja 2.5 (Uszeregowanie, por. [10])**.** *Uszeregowanie σ jest przyporządkowaniem zadań do maszyny w czasie tak, by spełnione były następujące warunki:*

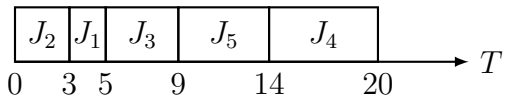
a) w każdej chwili czasu, maszyna wykonuje co najwyżej jedno zadanie oraz każde zadanie jest wykonywane dokładnie raz,
b) zadanie J_j, $1 \leqslant j \leqslant n$ jest wykonywane w przedziale czasu $\langle r_j\,;+\infty)$,
c) wszystkie zadania są wykonane,
d) jeżeli występują ograniczenia kolejnościowe między zadaniami, to zadania wykonywane są w kolejności zgodnej z tymi ograniczeniami,
e) jeżeli występują ograniczenia związane z dodatkowymi zasobami, to są one spełnione.

Uszeregowanie nazywamy dopuszczalnym, jeżeli spełnia wszystkie warunki z Definicji 2.5 oraz inne dodatkowe warunki wynikające ze specyfikacji rozważanego problemu. Dopuszczalne uszeregowanie minimalizujące przyjęte kryterium będziemy nazywać uszeregowaniem optymalnym i oznaczać symbolem $\sigma^\star$. Uszeregowania optymalne będziemy prezentowali za pomocą diagramu Gantta.

Każde zadanie J_j w uszeregowaniu σ opisują następujące parametry: czas rozpoczęcia $S_j(\sigma)$ oraz czas zakończenia $C_j(\sigma) = S_j(\sigma) + p_{j,r}(t)$.

Zilustrujemy wprowadzoną notację przykładem.

Przykład 2.1. Danych jest $n = 5$ zadań o stałych czasach wykonywania postaci $p_1 = 2$, $p_2 = 3$, $p_3 = 5$, $p_4 = 4$ oraz $p_5 = 6$, gotowych do wykonania na jednej maszynie, począwszy od czasu $t_0 = 0$.



Rysunek 2.3. Diagram Gantta optymalnego uszeregowania w Przykładzie 2.1

Wówczas mamy, że $S_2(\sigma) = 0$, $S_1(\sigma) = C_2(\sigma) = 3$, $S_3(\sigma) = C_1(\sigma) = 5$, $S_5(\sigma) = C_3(\sigma) = 9$, $S_4(\sigma) = C_5(\sigma) = 14$ oraz $C_4(\sigma) = 20$.

Jakość uzyskiwanych uszeregowań możemy oceniać dzięki ustalonemu kryterium optymalności. Brucker [3] rozróżnia dwa główne typy kryterium optymalności:

- $f_{\max} = \max\limits_{j \in \mathcal{J}} \{f_j(C_j)\}$ – funkcja maksymalnego kosztu
- $\sum f_j = \sum\limits_{j \in \mathcal{J}} f_j(C_j)$ – całkowity koszt

Szczególnymi postaciami kryterium $f_{\max}$ są następujące kryteria:

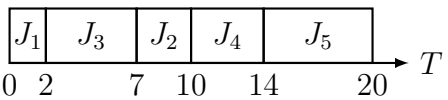
- $C_{\max} = \max\limits_{j \in \mathcal{J}} \{C_j\}$ – maksymalny czas zakończenia ostatniego zadania
- $L_{\max} = \max\limits_{j \in \mathcal{J}} \{L_j\} = \max\limits_{j \in \mathcal{J}} \{C_j - d_j\}$ – maksymalna nieterminowość

Szczególnymi postaciami kryterium $\sum f_j$ są następujące kryteria:

- $\sum C_j = \sum\limits_{j \in \mathcal{J}} C_j$ – suma czasów zakończenia zadań
- $\sum w_j C_j = \sum\limits_{j \in \mathcal{J}} w_j C_j$ – suma ważonych czasów zakończenia zadań

Dla problemu szeregowania zadań z kryterium $L_{\max}$, z którym spotkamy się jeszcze w rozdziale 6, podamy postać optymalnego uszeregowania, które otrzymujemy zgodnie z niemalejącym porządkiem względem d_j (zobacz diagram Gantta na Rysunku 2.4).

Przykład 2.2. Danych jest $n = 5$ zadań gotowych do wykonania na jednej maszynie, począwszy od czasu $t_0 = 0$, z czasami wykonywania jak w Przykładzie 2.1 oraz dodatkowo pożądanymi terminami zakończenia $d_1 = 4$, $d_2 = 7$, $d_3 = 5$, $d_4 = 15$ oraz $d_5 = 22$.



$$L_{\max} = \max\{-2, 2, 3, -1, -2\} = 3$$

Rysunek 2.4. Diagram Gantta optymalnego uszeregowania w Przykładzie 2.2

W klasycznej teorii szeregowania zadań czasy wykonywania zadań opisane są za pomocą liczb. W rozprawie będziemy rozpatrywali problemy ze zmien-

nymi czasami wykonywania, opisanymi za pomocą funkcji różnych od stałych i zależnych od czasu rozpoczęcia lub pozycji zadania w uszeregowaniu.

Do opisu problemów wykorzystamy rozszerzoną wersję trójpolowej notacji $\alpha|\beta|\gamma$ zaprezentowaną w pracy [12]. Omówimy poszczególne pola występujące w tej notacji na podstawie następującego problemu:

$$\underbrace{1}_{\alpha} \,|\, \underbrace{p_{j,r}\left(t\right) = b_j * t, tree}_{\beta} \,|\, \underbrace{C_{\max}}_{\gamma}$$

Pole α służy do opisu zbioru maszyn, dla jednomaszynowych problemów szeregowania zadań w tym polu mamy 1. Pole β z kolei opisuje zbiór zadań. Zawiera ono w szczególności informacje na temat czasów wykonywania zadań oraz postaci ograniczeń kolejnościowych między zadaniami. W naszym przypadku zbiór zadań zawiera zadania z czasami wykonywania opisanymi za pomocą funkcji proporcjonalnych $p_{j,r}\left(t\right) = b_j * t$. Ograniczenia kolejnościowe opisane są za pomocą drzewa (*tree*). Ostatnim polem jest γ, które zawiera informacje dotyczące postaci kryterium optymalności. W naszym problemie rozważamy kryterium optymalności jest kryterium minimalizacji maksymalnego czasu zakończenia zadania $C_{\max}$.

Postaci czasów wykonywania zadań rozważane w rozprawie

W rozprawie, o ile nie stwierdzono inaczej będziemy zakładać, że czasy wykonywania zadań $p_{j,r}\left(t\right)$ ze zbioru $\mathcal{J}$, mogą przyjąć postać:

$$p_{j,r}\left(t\right) = p_j, \tag{1}$$
$$p_{j,r}\left(t\right) = p_j * \left(A + B * t\right), \tag{2}$$
$$p_{j,r}\left(t\right) = p_j + B * r^a, \tag{3}$$
$$p_{j,r}\left(t\right) = r^a * \left(A + B * t\right). \tag{4}$$

gdzie $1 \leqslant j \leqslant n$, $1 \leqslant i \leqslant k$, $p_j > 0$, $A \geqslant 0$, $B > 0, a \neq 0$ i $r \geqslant 1$. Przez p_j i r oznaczamy odpowiednio podstawowe czasy wykonywania i pozycję zadania w uszeregowaniu. Parametr a oznacza albo efekt uczenia się (jeżeli $a < 0$) albo starzenia się (jeżeli $a > 0$).

Lematy pomocnicze

W dalszej części rozprawy będziemy często wykorzystywać następujące lematy pomocnicze:

Lemat 2.1. *Uszeregowanie zawierające przestoje zwraca niemniejszą wartość funkcji maksymalnego kosztu, niż uszeregowanie bez przestojów.*

Dowód. Rozważmy uszeregowanie, w którym występują przestoje. Załóżmy, że pomijamy pierwszy z nich, wtedy czasy rozpoczęcia i zakończenia występujących po nim zadań będą miały mniejsze wartości. Usuwając pozostałe przestoje dostaniemy uszeregowanie bez przestojów, w którym czasy zakończenia zadań są mniejsze, niż w przypadku uszeregowania z tymi przestojami. Funkcja kosztu jest niemalejąca ze względu na czas zakończenia zadania, zatem w uszeregowaniu nie zawierającym przestojów maksymalna wartość funkcji kosztu jest niewiększa od uszeregowania z przestojami. □

Lemat 2.2. *Jeżeli zbiór $\mathcal{J}$ zawiera tylko zadania z czasami wykonywania jednej z postaci* (1)–(4)*, to $C_{\max}$ nie zależy od uszeregowania.*

Dowód. Problemy z czasami wykonywania postaci (1) i (2) były rozważane wcześniej odpowiednio w [4] i [10]. Pokażemy, że wartość $C_{\max}$ nie zależy od uszeregowania dla czasów wykonywania (3) oraz (4).

Niech σ będzie dowolnym uszeregowaniem takim, że zadanie J_i poprzedza zadanie J_j i niech J_j będzie uszeregowane na pozycji k, gdzie $1 < k \leqslant n$. Niech σ' będzie uszeregowaniem, w którym J_i i J_j zostały zamienione miejscami.

Zakładając, że t_i jest czasem rozpoczęcia zadania J_i w σ dla czasów postaci (3) mamy, że: $C_i(\sigma) = C_j\left(\sigma'\right) = t_i + p_i + b * k^a$ oraz $C_j(\sigma) = C_i\left(\sigma'\right) = t_i + p_i + b * k^a + p_j + b * (k+1)^a$. Stąd $C_{\max}(\sigma) = C_{\max}\left(\sigma'\right)$.

Zakładając, że t_i jest czasem rozpoczęcia zadania J_i w σ dla czasów postaci (4) mamy, że: $C_i(\sigma) = C_j\left(\sigma'\right) = t_i + k^a * (A + B * t_i)$ oraz $C_j(\sigma) = C_i\left(\sigma'\right) = t_i + k^a * (A + B * t_i) + (k+1)^a * (A + B * (t_i + k^a * (A + B * t_i)))$. Stąd $C_{\max}(\sigma) = C_{\max}\left(\sigma'\right)$. □

Rozdział 3

Przegląd wybranej literatury

W tym rozdziale omówione zostaną główne wyniki badań dotyczące jednomaszynowych problemów szeregowania zadań tematycznie związanych z problemami rozważanymi w rozprawie. W podrozdziale 3.1 zaprezentowane zostaną wyniki dla problemów szeregowania zadań ze stałymi czasami wykonywania. W podrozdziale 3.2 omówione zostaną wyniki dla problemów szeregowania zadań ze zmiennymi czasami.

3.1. Zadania ze stałymi czasami wykonywania

W teorii szeregowania zadań rozważano do tej pory głównie problemy ze stałymi czasami wykonywania zadań. W pracy przeglądowej [14] zaprezentowano szereg wyników uzyskanych dla jednomaszynowych problemów szeregowania zadań ze stałymi czasami wykonywania. Najogólniejszy z nich dotyczy minimalizacji maksymalnego kosztu $f_{\max}$ przy dowolnych ograniczeniach kolejnościowych między zadaniami. Problem ten można sformułować następująco.

Dany jest zbiór zadań $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$, których czasy wykonywania opisane są funkcjami stałymi $p_{j,r}(t) = p_j$, gdzie $j \in \{1, 2, \ldots, n\}$. Między zadaniami występują dowolne ograniczenia kolejnościowe opisane przy pomocy acyklicznego grafu skierowanego $G = (V, A)$. Naszym celem jest znalezienie optymalnego uszeregowania $\sigma^\star$ dla kryterium $f_{\max}$.

W pracy [17] zaproponowano dla tego problemu następujący wielomianowy algorytm. W pseudokodzie tego algorytmu (patrz Algorytm 3.1) symbo-

le $NS(G), V(G)$ i $A(G)$ oznaczają odpowiednio zbiór zadań bez następników, zbiór wierzchołków i łuków w G. Dodatkowo symbole $(\circ)$, $|$ oraz $\deg^+(j)$ oznaczają odpowiednio początkowe uszeregowanie, operator konkatenacji oraz stopień wychodzący wierzchołka j.

Algorytm 3.1 dla problemu $1|p_{j,r}(t) = p_j, prec|f_{\max}$

$\sigma^\star \leftarrow (\circ)$
$T \leftarrow t_0 + \sum_{j=1}^{n}(p_j)$
while $V(G) \neq \emptyset$ **do**
    $NS(G) \leftarrow \left\{j \in V(G) : \deg^+(j) = 0\right\}$
    Znajdź $k \in NS(G)$ takie, że $f_k(T) = \min\{f_j(T) : j \in NS(G)\}$
    $\sigma^\star \leftarrow (k|\sigma^\star)$
    $T \leftarrow T - (p_k)$
    $V(G) \leftarrow V(G) \setminus \{k\}$
    $A(G) \leftarrow A(G) \setminus \{(i,k) : i \in V(G)\}$
end while
return $\sigma^\star$

Twierdzenie 3.1. *([17]) Algorytm 3.1 znajduje optymalne uszeregowanie dla problemu postaci* $1|p_{j,r}(t) = p_j, prec|f_{\max}$ *w czasie* $O(n^2)$.

W dalszej części rozprawy będziemy prezentowali algorytmy będące modyfikacjami Algorytmu 3.1.

3.2. Zadania ze zmiennymi czasami wykonywania

W pewnym momencie rozwoju teorii szeregowania zadań modele szeregowania zadań ze stałymi czasami wykonywania stały się niewystarczające ([10], [2]). Coraz częściej obserwowano, że czasy wykonywania zadań nie zawsze są stałe, lecz mogą zależeć od wielu czynników. W taki sposób narodziła się rodzina problemów szeregowania zadań ze zmiennymi czasami wykonywania. Zauważono, że niektóre zadania są silnie uwarunkowane czasowo. Dla przykładu możemy podać gaszenie pożaru przez straż pożarną, gdzie im później strażacy rozpoczną gaszenie, tym większa część budynku może być objęta ogniem, co wydłuży z kolei czas wykonania całego zadania. Druga grupa problemów dotyczy zadań, których czas wykonywania zależy od poziomu wiedzy pracownika lub stanu technicznego narzędzi, które są używane podczas ich

wykonywania. I tak – im bardziej doświadczony pracownik, tym szybciej wykonuje powierzone mu zadanie. Z drugiej strony w wyniku efektu zużywania się narzędzi czas pracy może się wydłużać, poprzez przestoje i wymuszone naprawy.

Do najpopularniejszych modeli szeregowania zadań ze zmiennymi czasami wykonywania zaliczamy: modele czasowo-zależne oraz modele uwzględniające efekt uczenia się bądź starzenia się zadań, zwane również modelami pozycyjno-zależnymi.

Problemy szeregowania zadań czasowo-zależnych

W problemach szeregowania czasowo-zależnych czas wykonywania zadania jest opisany funkcją różną od stałej, zależną od czasu rozpoczęcia tego zadania. Szeregowanie zadań czasowo-zależnych jest stosunkowo nowym nurtem w teorii szeregowania zadań. Pomimo, że pierwsza praca z tej tematyki ukazała się pod koniec lat siedemdziesiątych XX w., to znaczna część prac pochodzi z ostatnich 15 lat. Najobszerniejszą dyskusję na temat dotychczas rozważanych problemów szeregowania zadań czasowo-zależnych odnajdziemy w monografii [10].

Zadania w problemach szeregowania czasowo-zależnych posiadają najczęściej następujące postaci czasów wykonywania:

- $p_{j,r}(t) = b_j * t$ – proporcjonalne
- $p_{j,r}(t) = p_j * (A + B * t)$ – proporcjonalno-liniowe
- $p_{j,r}(t) = a_j + b_j * t$ – liniowe

Problemy szeregowania zadań pozycyjno-zależnych

W problemach szeregowania zadań z efektem uczenia się bądź starzenia się zadań, zakładamy z kolei, że czas wykonywania zadania jest opisany przez funkcję różną od stałej, która zależy od pozycji zadania w uszeregowaniu. To założenie pozwala nam rozpatrywać problemy, w których czas wykonywania zadania może maleć lub wzrastać w powiązaniu z tempem uczenia się pracownika, który wykonuje zadanie lub starzeniem się narzędzi, które są wy-

korzystywane podczas prac nad tym zadaniem. Problematyka szeregowania zadań pozycyjno-zależnych opisana została w pracy przeglądowej [2].

Zadania w problemach szeregowania pozycyjno-zależnych posiadają najczęściej następujące postaci czasów wykonywania:

- $p_{j,r}(t) = p_j * r^a$
- $p_{j,r}(t) = p_j + B * r^a$

Omawiane dalej problemy szeregowania zadań ze zmiennymi czasami wykonywania zostały podzielone na trzy grupy:

1. Problemy z kryterium $f_{\max}$ i dowolnymi ograniczeniami kolejnościowymi
2. Problemy z niepustymi ograniczeniami kolejnościowymi
3. Problemy z mieszanymi postaciami czasów wykonywania

Problemy szeregowania zadań z kryterium $f_{\max}$

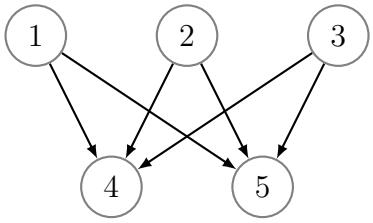
Pierwszy z omawianych problemów ma postać $1|p_{j,r}(t) = p_j * t, prec|f_{\max}$. W monografii [10, Rozdział 13] przedstawiono dla tego problemu wielomianowy algorytm, który oznaczamy jako Algorytm 3.2, będący adaptacją Algorytmu 3.1 na przypadek proporcjonalnych czasów wykonywania, w którym linie 2 oraz 8 mają teraz odpowiednio postać '$T \leftarrow t_0 * \prod_{j=1}^{n}(1+p_j)$' oraz '$T \leftarrow (T)/(1+p_k)$'.

Wymieniony wyżej Algorytm 3.2 jest szczególnym przypadkiem innego algorytmu przedstawionego w monografii [10], dotyczącego problemu szeregowania zadań z liniowo-proporcjonalnymi czasami wykonywania. Dla problemu postaci $1|p_{j,r}(t) = p_j * (A + B * t), prec|f_{\max}$ zaproponowano algorytm oznaczany przez nas jako Algorytm 3.3, który znajduje optymalne uszeregowanie w wielomianowym czasie. Jest on modyfikacją Algorytmu 3.1, w którym linie 2 oraz 8 zastąpiono odpowiednio przez '$T \leftarrow (t_0 + A/B) * \prod_{j=1}^{n}(1 + B * p_j)$' oraz '$T \leftarrow (T - A * p_k)/(1 + B * p_k)$'.

Twierdzenie 3.2. *([10]) Algorytm 3.3 znajduje optymalne uszeregowanie dla problemu postaci* $1|p_{j,r}(t) = p_j * (A + B * t), prec|f_{\max}$ *w czasie* $O(n^2)$.

Zilustrujemy działanie Algorytmu 3.3 na poniższym przykładzie.

Przykład 3.1. Danych jest $n = 5$ zadań z czasami wykonywania postaci $p_1 = t, p_2 = 3 * t, p_3 = 2 * t, p_4 = t, p_5 = 4 * t$, gdzie $A = 0$, $B = 1$. Czas rozpoczęcia $t_0 = 1$. Funkcje kosztów mają następującą postać $f_1 = 2 \times C_1 + 1$, $f_2 = C_2 + 15$, $f_3 = C_3 + 2$, $f_4 = C_4 + 3$, $f_5 = 2 \times C_5 + 3$. Ograniczenia kolejnościowe między zadaniami przedstawiono na Rysunku 3.1.



Rysunek 3.1. Ograniczenia kolejnościowe między zadaniami w Przykładzie 3.1

Opiszemy teraz kolejne iteracje Algorytmu 3.3 potrzebne do znalezienia optymalnego uszeregowania.

W pierwszej iteracji naszego algorytmu obliczamy czas zakończenia ostatniego zadania $T = 240$. Następnie konstruujemy zbiór zadań bez następników $NS(G) = \{4, 5\}$. Spośród zadań z tego zbioru wybieramy zadanie o najmniejszej wartości funkcji kosztu i umieszczamy je na ostatniej pozycji w uszeregowaniu. Tym zadaniem jest J_4, ponieważ $f_4(240) = 243 < f_5(240) = 483$. Zatem $\sigma^\star = (4)$. Aktualizujemy zbiory $V(G)$, $A(G)$ oraz $NS(G)$.

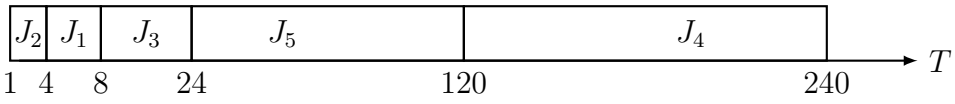
Następnie, obliczamy $T = \frac{240}{2} = 120$ oraz konstruujemy $NS(G) = \{5\}$. Zadanie J_5 dodajemy do uszeregowania $\sigma^\star = (5, 4)$ i aktualizujemy zbiory $V(G)$, $A(G)$ oraz $NS(G)$.

Ponownie obliczamy wartość $T = \frac{120}{5} = 24$ oraz konstruujemy zbiór $NS(G) = \{1, 2, 3\}$. Ponieważ $f_3(24) = 26 < f_2(24) = 39 < f_1(24) = 49$ zadanie J_3 dodajemy do uszeregowania $\sigma^\star = (3, 5, 4)$.

W czwartej iteracji, $T = \frac{24}{3} = 8$ oraz $NS(G) = \{1, 2\}$. Ponieważ $f_1(8) = 17 < f_2(8) = 23$ zadanie J_1 dodajemy do uszeregowania $\sigma^\star = (1, 3, 5, 4)$.

W ostatniej iteracji obliczamy wartość $T = \frac{8}{2} = 4$ oraz konstruujemy zbiór $NS(G) = \{2\}$. Stąd wybieramy J_2 i umieszczamy je w uszeregowaniu.

Uzyskujemy zatem optymalne uszeregowania $\sigma^\star = (2, 1, 3, 5, 4)$, dla którego wartość funkcji kryterialnej $f_{\max} = 243$.



Rysunek 3.2. Optymalne uszeregowanie w Przykładzie 3.1

Problemy szeregowania zadań z niepustymi ograniczeniami kolejnościowymi

Jednomaszynowe problemy szeregowania zadań z niepustymi ograniczeniami kolejnościowymi między zadaniami można podzielić według postaci czasów wykonywania zadań.

W pracy [13] zaprezentowano jednomaszynowe problemy szeregowania zadań z czasami wykonywania zależnymi od pozycji zadania w uszeregowaniu:

- $1|p_{j,r} = p_j * r, SP|C_{\max}$ – złożoność $O(n \log n)$
- $1|p_{j,r} = p_j \gamma^{r-1}, SP|C_{\max}$ – złożoność $O(n \log n)$ dla $\gamma > 0, \gamma \neq 1$

W pracy [8] zaprezentowano jednomaszynowy problem szeregowania zadań z czasami wykonywania zależnymi od pozycji zadania w uszeregowaniu, który dla pewnych ograniczeń kolejnościowych jest rozwiązywalny w czasie wielomianowym.

Trzecia praca [23] dotyczy problemów postaci $1|p_{i,j} = a_{i,j} + b_{i,j}r, \delta|C_{\max}$, gdzie $\delta \in \{weak\ chain, strong\ chain, SP\}$ oznacza ograniczenia kolejnościowe między zadaniami w postaci słabego łańcucha, silnego łańcucha oraz grafów szeregowo-równoległych. W przypadku ograniczeń kolejnościowych w postaci silnego łańcucha, zadania występujące w tym łańcuchu muszą być wykonane sekwencyjnie i nie jest możliwe w tym czasie wstawienie do uszeregowania zadania z innego łańcucha. W przypadku słabego łańcucha takie wstawienie zadania jest możliwe.

Kolejna grupa problemów dotyczy jednomaszynowych problemów szeregowania zadań czasowo-zależnych. Jedynie kilku autorów rozważało tego typu problemy z niepustymi ograniczeniami kolejnościowymi między zadaniami.

W monografii [10, Rozdział 13] rozważano problemy następującej postaci $1|p_j = a_j + b_j * t, \delta|C_{\max}$, gdzie $\delta \in \{chains, tree, forest, SP\}$ oznacza ograniczenia kolejnościowe odpowiednio w postaci łańcuchów, drzewa, lasu i grafu szeregowo-równoległego oraz $a_j > 0$, $b_j > 0$ dla $1 \leqslant j \leqslant n$. Wyniki uzyskane w tej monografii przedstawiono w Tabeli 3.1.

Problem	Złożoność
$1\|p_j = a_j + b_j * t, chains\|C_{\max}$	$O(n \log n)$
$1\|p_j = a_j + b_j * t, tree\|C_{\max}$	$O(n \log n)$
$1\|p_j = a_j + b_j * t, forest\|C_{\max}$	$O(n \log n)$
$1\|p_j = a_j + b_j * t, SP\|C_{\max}$	$O(n \log n)$

Tabela 3.1. Wyniki uzyskane w monografii [10, Rozdział 13]

W pracy [11] rozpatrywano problemy postaci $1|p_j = a_j - b_j * t, \delta|C_{\max}$, zakładając, że $a_j > 0$, $1 < b_j < 0$ oraz $b_j \left(\sum_{i=1}^{n} a_i - a_j\right) < a_j$ dla $1 \leqslant j \leqslant n$. Złożoność problemów jest podobna do tych zaprezentowanych w Tabeli 3.1.

W monografii [10] oraz pracy [21] zaprezentowano wielomianowe algorytmy dla problemu $1|p_j = p_j (A + B * t), prec|C_{\max}$. Natomiast w pracy [13] dla problemu postaci $1|p_j = p_j (1 \pm a * t), prec|C_{\max}$ skonstruowano liniowy algorytm.

Problem	Złożoność
$1\|p_j = p_j (A + B * t), prec\|C_{\max}$	$O(n \log n)$
$1\|p_j = p_j (1 \pm a * t), prec\|C_{\max}$	$O(n)$

Tabela 3.2. Wyniki zaprezentowane w monografii [10] oraz w pracach [21] i [13]

Problemy szeregowania zadań z mieszanymi czasami wykonywania zadań

Ostatnia grupa wyników dotyczy problemów szeregowania zadań z mieszanymi czasami wykonywania zadań.

W 2010r. Gawiejnowicz i Lin w pracy [12] zaproponowali nowy model szeregowania zadań zależnych z mieszanych czasami wykonywania. Dla tego modelu zbiór zadań $\mathcal{J}$ może zawierać jednocześnie zadania z czasami wykonywania opisanymi funkcjami różnych typów. Autorzy rozszerzyli trójpolową

notację [14] poprzez dodanie do parametru β liczby zadań z określonego typu. Rozważmy dla przykładu zbiór n zadań, gdzie n_1 oraz n_2 zadań ma czasy wykonywania postaci odpowiednio $p_j * t$ oraz $t * r^a$. Otrzymujemy wtedy następujący problem $1|p_{j,r}(t) \in \{p_j * t, n_1; t * r^a, n_2\}|C_{\max}$, gdzie $n_1 + n_2 = n$.

W powyższej pracy autorzy zaprezentowali szereg własności dotyczących jednomaszynowego problemu szeregowania zadań mieszanych z kryterium $\sum C_j$ oraz zaproponowali algorytm rozwiązujący problem następującej postaci: $1|p_{j,r}(t) \in \{a_j, n_1; b_j * t, n_2; a_j + b_j * t, n_3\}|C_{\max}$ w czasie $O(n \log n)$. Idea tego algorytmu jest następująca: na początku szeregujemy w dowolnym porządku zadania z czasami proporcjonalnymi. Następnie na kolejnych pozycjach umieszczamy zadania z czasami liniowymi w nierosnącym porządku $\frac{b_j}{a_j}$. Jako ostatnie szeregujemy zadania z czasami stałymi w dowolnej kolejności i otrzymujemy optymalne uszeregowanie.

Podsumowanie

Problemy szeregowania zadań ze zmiennymi czasami wykonywania i niepustymi ograniczeniami kolejnościowymi są rzadko rozpatrywane. Najczęściej rozważa się problemy, w których czasy wykonywania zadań są opisane tylko funkcjami jednego typu. W większości prac rozważa się również jedynie szczególne przypadki kryterium maksymalnego kosztu $f_{\max}$. Ponadto problemy szeregowania zadań z mieszanymi czasami wykonywania zadań wystąpiły tylko w jednej pracy.

Jak wynika z zaprezentowanego przeglądu literatury, problemy szeregowania zadań ze zmiennymi czasami wykonywania, niepustymi ograniczeniami kolejnościowymi i kryterium $f_{\max}$ nie są jeszcze wystarczająco zbadane. W rozprawie będą rozważane zatem problemy szeregowania zadań ze zmiennymi czasami wykonywania, w których występują jednocześnie mieszane czasy wykonywania, niepuste ograniczenia kolejnościowe oraz kryterium maksymalnego kosztu $f_{\max}$, będące uogólnieniem kryteriów $C_{\max}$ oraz $L_{\max}$.

Rozdział 4

Rodzina problemów postaci

$1|p_{j,r}(t) \in \{p_1, n-k; p_2, k\}, prec|f_{\max}$

W tym rozdziale zaprezentowana zostanie pierwsza z rozważanych w rozprawie rodzin jednomaszynowych problemów szeregowania zadań zależnych z mieszanymi czasami wykonywania, dowolnymi ograniczeniami kolejnościowymi między zadaniami oraz kryterium maksymalnego kosztu $f_{\max}$. Podrozdział 4.1 zawiera sformułowanie omawianych problemów. W podrozdziale 4.2 zaprezentowane zostaną wielomianowe algorytmy rozwiązujące problemy szeregowania zadań ze zmiennymi czasami wykonywania zadań opisanymi funkcjami z tej samej rodziny. W podrozdziale 4.3 zaprezentowane zostaną wielomianowe algorytmy rozwiązujące problemy szeregowania zadań ze zmiennymi mieszanymi czasami wykonywania zadań opisanymi funkcjami z dwóch różnych rodzin.

4.1. Sformułowanie problemów rodziny

Niech dany będzie zbiór $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ zadań. Czas wykonania zadania J_j oznaczamy przez $p_{j,r}(t)$, gdzie $1 \leqslant j \leqslant n$. Pierwsze zadanie zaczyna się w chwili początkowej $t_0 \geqslant 0$. Pomiędzy zadaniami występują dowolne ograniczenia kolejnościowe określające częściowy porządek na zbiorze zadań. Ograniczenia kolejnościowe opisane są za pomocą acyklicznego grafu skierowanego $G = (V, A)$, gdzie V i A oznaczają odpowiednio zbiór wierzchołków i zbiór łuków w grafie G.

Czasy wykonywania zadań są mieszane, tzn. mogą mieć różną postać w zależności od zadania i są opisane przez funkcje, które jednocześnie zale-

żą od pozycji, jak i czasu rozpoczęcia zadania. Precyzując – aktualny czas wykonywania $p_{j,r}(t)$ zadania $J_j \in \mathcal{J}$ uszeregowanego na pozycji r w czasie $t \geqslant t_0$ może mieć jedną z następujących postaci: $p_{j,r}(t) = p_j + b * r^a$, $p_{j,r}(t) = t * r^a, p_{j,r}(t) = p_j * t$ lub $p_{j,r}(t) = p_j$, gdzie podstawowe czasy wykonywania $p_j > 0$ dla $1 \leqslant j \leqslant n, b > 0, a \neq 0$ i $r \geqslant 1$ oznacza pozycję zadania w uszeregowaniu. Dodatkowo dla każdego zadania $J_j \in \mathcal{J}$ znana jest niemalejąca funkcja kosztu $f_j(t)$, która mierzy koszt zakończenia zadania J_j w chwili t. Zakładamy, że jej wartość możemy oszacować w stałym czasie. Naszym celem jest znalezienie uszeregowania σ o najmniejszej wartości $f_{\max}(\sigma) = \max\limits_{1 \leqslant j \leqslant n} \{f_j(C_j(\sigma))\}$ spośród wszystkich uszeregowań zgodnych z ograniczeniami kolejnościowymi, gdzie $C_j(\sigma)$ oznacza czas zakończenia zadania $J_j \in \mathcal{J}$ w uszeregowaniu σ.

Wykorzystując trójpolową notację wprowadzoną w pracy [12], problemy z tej rodziny przyjmują postać $1|p_{j,r}(t) \in \{p_1, n-k; p_2, k\}, prec|f_{\max}$, gdzie zbiór zadań $\mathcal{J}$ podzielony został na $n-k$ i k zadań z czasami wykonywania odpowiednio postaci p_1 i p_2, a $p_1, p_2 \in \{p_j, p_j * t, p_j + b * r^a, t * r^a\}$.

4.2. Zadania ze zmiennymi czasami wykonywania z tej samej rodziny

Algorytmy zaprezentowane w tym podrozdziale są adaptacją Algorytmu 3.1 na przypadek zadań zależnych od czasu rozpoczęcia zadania lub pozycji zadania w uszeregowaniu. Na wejściu algorytmów mamy dane czas rozpoczęcia t_0, graf ograniczeń kolejnościowych między zadaniami $G = (V, A)$, funkcje kosztów f_1, f_2,..., f_n, podstawowe czasy wykonywania p_1, p_2, ..., p_n, indeks efektu uczenia się (lub starzenia się) zadań a i stałą b związaną z efektem uczenia się (lub starzenia się) zadań. Na wyjściu algorytmów dostajemy optymalne uszeregowanie $\sigma^\star$. Idea tych algorytmów jest następująca. Dla danego grafu G, w każdym kroku wybieramy spośród zbioru zadań bez następników to zadanie, które ma najmniejszą wartość funkcji kosztu na danej pozycji i umieszczamy je na początku dotychczas uszeregowanych zadań. W podobny sposób postępujemy z wszystkimi pozostałymi do uszeregowania zadaniami.

Pierwszy z algorytmów dotyczy problemu $1|p_{j,r}(t) = p_j + b*r^a, prec|f_{\max}$, w którym czasy wykonywania zadań zależą od pozycji r zadania w uszeregowaniu. Pseudokod algorytmu jest dany poniżej (patrz Algorytm 4.1).

Algorytm 4.1 dla problemu $1|p_{j,r}(t) = p_j + b*r^a, prec|f_{\max}$

1: $\sigma^\star \leftarrow (\circ)$
2: $T \leftarrow t_0 + \sum_{r=1}^{n}(p_r + b*r^a)$
3: $r \leftarrow n$
4: **while** $V(G) \neq \emptyset$ **do**
5: $\quad NS(G) \leftarrow \left\{j \in V(G) : \deg^+(j) = 0\right\}$
6: $\quad$ Znajdź $k \in NS(G)$ taki, że $f_k(T) = \min\{f_j(T) : j \in NS(G)\}$
7: $\quad \sigma^\star \leftarrow (k|\sigma^\star)$
8: $\quad T \leftarrow T - (p_k + b*r^a)$
9: $\quad V(G) \leftarrow V(G) \setminus \{k\}$
10: $\quad A(G) \leftarrow A(G) \setminus \{(i,k) : i \in V(G)\}$
11: $\quad r \leftarrow r - 1$
12: **end while**
13: **return** $\sigma^\star$

Twierdzenie 4.1. *Algorytm 4.1 konstruuje optymalne uszeregowanie dla problemu postaci* $1|p_{j,r}(t) = p_j + b*r^a, prec|f_{\max}$ *w czasie* $O(n^2)$.

Dowód. Zauważmy, że Algorytm 4.1 generuje jedynie dopuszczalne uszeregowania, które nie naruszają ograniczeń kolejnościowych między zadaniami. Wynika to z faktu, iż w pętli '**while**' (linie 4–12) rozważane są jedynie te zadania, które nie posiadają następników. Zauważmy, że wystarczy rozważyć uszeregowania posiadające dwie następujące własności.

Po pierwsze zgodnie z Lematem 2.1 wystarczy, że rozważymy jedynie uszeregowania nie zawierające przestojów.

Po drugie zgodnie z Lematem 2.2 czas zakończenia ostatniego zadania, dla problemu $1|p_{j,r}(t) = p_j + b*r^a, prec|f_{\max}$, nie zależy od uszeregowania.

Teraz pokażemy, że uszeregowanie generowane przez Algorytm 4.1 ma minimalny koszt. Niech $NS(\mathcal{J})$ oraz $f^\star_{\max}(J_k)$ oznaczają odpowiednio zbiór zadań bez następników i maksymalny koszt w optymalnym uszeregowaniu dla każdego $J_k \in NS(\mathcal{J})$. Załóżmy, że T jest czasem zakończenia ostatniego zadania i niech J_k będzie zadaniem bez następnika takim, że:

$$f_k(T) = \min_{J_j \in NS(\mathcal{J})}\{f_j(T)\}. \tag{4.1}$$

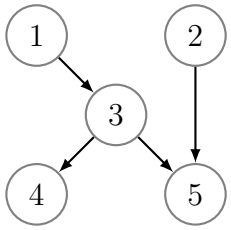
Wtedy, koszt uszeregowania zadania J_k na ostatniej pozycji jest równy $f^{\star}_{\max}(\mathcal{J}) = \max\{f_k(T), f^{\star}_{\max}(\mathcal{J}\setminus\{J_k\})\}$. Ten koszt jest optymalny ponieważ $f^{\star}_{\max}(\mathcal{J}) \geqslant f^{\star}_{\max}(\mathcal{J}\setminus\{J_k\})$ i $f^{\star}_{\max}(\mathcal{J}) \geqslant \min\limits_{J_j \in NS(\mathcal{J})}\{f_j(T)\}$.

Powtarzając procedurę konstruowania zbioru $NS(\mathcal{J})$ i wybierania zadania spełniającego (4.1) dla wszystkich nieuszeregowanych zadań z tego zbioru, otrzymamy optymalne uszeregowanie.

Złożoność obliczeniowa Algorytmu 4.1 może zostać oszacowana w następujący sposób. Linię 2 możemy wykonać w czasie $O(n)$. Pętla '**while**' wykona się w czasie $O(n)$. Na skonstruowanie zbioru $NS(G)$ w linii 5 potrzebujemy czasu $O(n)$. Złożoność obliczeniowa Algorytmu 4.1 wynosi zatem $O(n^2)$. □

Zilustrujemy działanie Algorytmu 4.1 na poniższym przykładzie.

Przykład 4.1. Danych jest $n = 5$ zadań z następującymi podstawowymi czasami wykonywania $p_1 = 1$, $p_2 = 2$, $p_3 = 3$, $p_4 = 4$, $p_5 = 5$. Indeks starzenia się zadań $a = 0.5$, czynnik starzenia się zadań $b = 2$ oraz czas rozpoczęcia zadań $t_0 = 0$. Funkcje kosztów mają następującą postać $f_1 = C_1 + 2$, $f_2 = C_2^2 + 1$, $f_3 = 3 \times C_3$, $f_4 = C_4 + 14$, $f_5 = 2 \times C_5 + 5$. Ograniczenia kolejnościowe między zadaniami przedstawiono na Rysunku 4.1.



Rysunek 4.1. Ograniczenia kolejnościowe w Przykładzie 4.1

Poniższe iteracje Algorytmu 4.1 konstruują optymalne uszeregowanie.

Iteracja 1 Obliczamy czas zakończenia ostatniego zadania $T = 31.76$ oraz konstruujemy zbiór zadań bez następników $NS(G) = \{4, 5\}$. Obliczamy wartości funkcji kosztów dla zadań z tego zbioru. Ponieważ $f_4(31.76) = 45.76 < f_5(31.76) = 68.53$ zatem zadanie J_4 umieszczamy w uszeregowa-

niu $\sigma^\star = (4)$. Aktualizujemy wartość $T = 31.76 - 4 - 4.47 = 23.29$ oraz zbiór wierzchołków $V(G)$ i łuków $A(G)$.

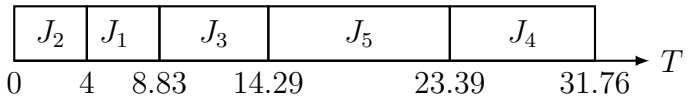
Iteracja 2 Konstruujemy zbiór zadań $NS(G) = \{5\}$ i obliczamy wartość funkcji $f_5(23.29) = 51.59$ i umieszczamy zadanie J_5 w $\sigma^\star = (5, 4)$. Aktualizujemy wartość $T = 23.29 - 5 - 4 = 14.29$ oraz zbiory $V(G)$ i $A(G)$.

Iteracja 3 Konstruujemy zbiór $NS(G) = \{2, 3\}$ i obliczamy wartości funkcji kosztów. Ponieważ $f_3(14.29) = 42.88 < f_2(14.29) = 205.28$ to zadanie J_3 zostaje umieszczone w uszeregowaniu $\sigma^\star = (3, 5, 4)$. Aktualizujemy wartość $T = 14.29 - 3 - 3.46 = 8.83$ oraz zbiory $V(G)$ i $A(G)$.

Iteracja 4 Konstruujemy zbiór $NS(G) = \{1, 2\}$ i obliczamy wartości funkcji kosztów. Ponieważ $f_1(8.83) = 9.83 < f_2(8.83) = 62.28$ zatem zadanie J_1 umieszczamy w uszeregowaniu $\sigma^\star = (1, 3, 5, 4)$. Aktualizujemy wartość $T = 8.83 - 1 - 2.83 = 4$ oraz zbiory $V(G)$ i $A(G)$.

Iteracja 5 Konstruujemy zbiór $NS(G) = \{2\}$. Następnie obliczamy wartość $f_2(4) = 17$ i umieszczamy J_2 w uszeregowaniu $\sigma^\star = (2, 1, 3, 5, 4)$.

Iteracja 6 Optymalne uszeregowanie to $\sigma^\star = (2, 1, 3, 5, 4)$ z $f_{\max} = 51.59$



Rysunek 4.2. Optymalne uszeregowanie w Przykładzie 4.1

Następny algorytm dotyczy problemu szeregowania czasowo–zależnego postaci $1|p_{j,r}(t) = t * r^a, prec|f_{\max}$, w którym czasy wykonywania zadań zależą jednocześnie od czasu rozpoczęcia t oraz pozycji r zadania w uszeregowaniu. W tym przypadku musimy założyć, że czas rozpoczęcia $t_0 > 0$, gdyż w przeciwnym wypadku czasy wykonywania zadań byłyby zerowe.

Dla tego problemu również czas zakończenia ostatniego zadania nie zależy od uszeregowania (Lemat 2.2) oraz optymalne uszeregowanie nie zawiera przestojów (Lemat 2.1). Modyfikując w Algorytmie 4.1 linie 2 oraz 8 odpowiednio na postać '$T \leftarrow t_0 * \prod_{r=1}^{n}(1 + r^a)$' oraz '$T \leftarrow T/(1 + r^a)$', otrzymu-

jemy Algorytm 4.2 rozwiązujący nasz problem. Pseudokod zmodyfikowanego algorytmu jest następujący:

Algorytm 4.2 dla problemu $1|p_{j,r}(t) = t * r^a, prec|f_{\max}$

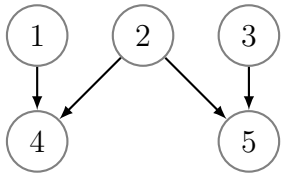
$\sigma^\star \leftarrow (\circ)$
$T \leftarrow t_0 * \prod_{r=1}^{n} (1 + r^a)$
$r \leftarrow n$
while $V(G) \neq \emptyset$ **do**
    $NS(G) \leftarrow \left\{ j \in V(G) : \deg^+(j) = 0 \right\}$
    Znajdź $k \in NS(G)$ taki, że $f_k(T) = \min\{f_j(T) : j \in NS(G)\}$
    $\sigma^\star \leftarrow (k|\sigma^\star)$
    $T \leftarrow T/(1 + r^a)$
    $V(G) \leftarrow V(G) \setminus \{k\}$
    $A(G) \leftarrow A(G) \setminus \{(i,k) : i \in V(G)\}$
    $r \leftarrow r - 1$
end while
return $\sigma^\star$

Używając podobnego rozumowania jak w dowodzie Twierdzenia 4.1, uzyskujemy następujące twierdzenie.

Twierdzenie 4.2. *Algorytm 4.2 konstruuje optymalne uszeregowanie dla problemu postaci* $1|p_{j,r}(t) = t * r^a, prec|f_{\max}$ *w czasie* $O(n^2)$.

Zilustrujemy działanie Algorytmu 4.2 na poniższym przykładzie.

Przykład 4.2. Danych jest $n = 5$ zadań z czasami wykonywania postaci $p_{j,r}(t) = t * r^a$, gdzie $1 \leqslant j \leqslant 5$ z indeksem uczenia się $a = -0.25$ i czasem rozpoczęcia zadań $t_0 = 1$. Funkcje kosztów są tej samej postaci jak w Przykładzie 4.1. Ograniczenia kolejnościowe przedstawiono na Rysunku 4.3.



Rysunek 4.3. Ograniczenia kolejnościowe w Przykładzie 4.2

Poniższe iteracje algorytmu konstruują optymalne uszeregowanie.

Iteracja 1 Obliczamy czas zakończenia ostatniego zadania $T = 18.46$ oraz konstruujemy zbiór zadań bez następników $NS(G) = \{4, 5\}$. Następnie obliczamy wartości funkcji kosztu dla zadań ze zbioru $NS(G)$. Ponieważ $f_4(18.46) = 32.46 < f_5(18.46) = 41.92$ zadanie J_4 umieszczamy w uszeregowaniu $\sigma^\star = (4)$. Aktualizujemy wartość $T = 18.46/(1 + 5^{-0.25}) = 11.06$ oraz zbiór wierzchołków $V(G)$ i łuków $A(G)$.

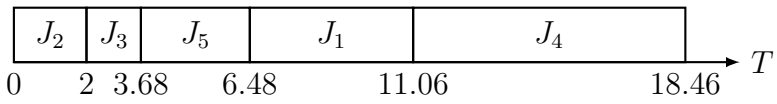
Iteracja 2 Konstruujemy zbiór $NS(G) = \{1, 5\}$, obliczamy wartości funkcji $f_1(11.06) = 13.06 < f_5(11.06) = 27.12$ i umieszczamy zadanie J_1 na przedostatniej pozycji w uszeregowaniu $\sigma^\star = (1, 4)$. Aktualizujemy wartość $T = 11.06/(1 + 4^{-0.25}) = 6.48$ oraz zbiory $V(G)$ i $A(G)$.

Iteracja 3 Konstruujemy zbiór $NS(G) = \{5\}$, obliczamy wartość funkcji kosztu $f_5(6.48) = 17.96$ i umieszczamy J_5 w $\sigma^\star = (5, 1, 4)$. Aktualizujemy wartość $T = 6.48/(1 + 3^{-0.25}) = 3.68$ oraz zbiory $V(G)$ i $A(G)$.

Iteracja 4 Konstruujemy zbiór $NS(G) = \{2, 3\}$, obliczamy wartości funkcji kosztów $f_3(3.68) = 11.05 < f_2(3.68) = 14.56$ i umieszczamy J_3 w uszeregowaniu $\sigma^\star = (3, 5, 1, 4)$. Aktualizujemy wartość $T = 3.68(1 + 2^{-0.25}) = 2$ oraz zbiory $V(G)$ i $A(G)$.

Iteracja 5 Konstruujemy zbiór $NS(G) = \{2\}$. Następnie obliczamy wartość $f_2(2) = 5$ i umieszczamy J_2 w uszeregowaniu $\sigma^\star = (2, 3, 5, 1, 4)$.

Iteracja 6 Optymalne uszeregowanie to $\sigma^\star = (2, 3, 5, 1, 4)$ z $f_{\max} = 32.46$



Rysunek 4.4. Optymalne uszeregowanie w Przykładzie 4.2

4.3. Zadania ze zmiennymi mieszanymi czasami wykonywania

Uogólnimy teraz wyniki uzyskane w podrozdziale 4.2 na przypadek zadań z mieszanymi czasami wykonywania wprowadzając dwa typy czasów: stałe i zmienne. Stałe czasy wykonywania zadań opisane są przez liczby, podczas gdy zmienne czasy wykonywania zadań opisane są przez funkcje.

W przeciwieństwie do problemów z zadaniami tego samego typu, własność dotycząca niezmienności długości uszeregowania, opisana w Lemacie 2.2, nie zachodzi dla zadań z mieszanymi czasami wykonywania. W tym przypadku dla różnych uszeregowań możemy uzyskiwać inne czasy zakończenia ostatniego zadania T. Istnieje jednak możliwość, aby pogrupować wszystkie uszeregowania w taki sposób, że w każdej takiej grupie czasy zakończenia ostatniego zadania będą takie same. Następnie dla każdej z tych grup, stosując algorytmy przedstawione wcześniej, konstruujemy uszeregowania z minimalnym kosztem. Wśród znalezionych uszeregowań wybieramy to optymalne.

Aby skorzystać z tej idei, zdefiniujemy pojęcie *szablonu*. Szablonem H będziemy nazywać wektor zero-jedynkowy taki, że $H[i] = 0$ ($H[i] = 1$) jeżeli czas wykonywania zadania J_i jest stały (zmienny). Szablon grupuje uszeregowania z tym samym czasem zakończenia ostatniego zadania T. Zauważmy, że mamy $\binom{n}{k} = O\left(n^k\right)$ unikalnych szablonów dla k stałych zadań.

Pokażemy, jak wykorzystać szablony na przykładzie algorytmu dla problemu $1|p_{j,r}(t) \in \{p_j + b * r^a, n-k; p_j, k\}, prec|f_{\max}$. Na wejściu algorytmu, poza danymi używanymi przez Algorytmy 4.1 i 4.2, mamy dane k zadań stałych, podczas gdy na wyjściu dostajemy optymalne uszeregowanie $\sigma^\star$.

Zanim sformułujemy algorytm, musimy zdefiniować cztery pomocnicze funkcje: FirstTemplate, NextTemplate, Reverse i CalcT.

Funkcja FirstTemplate dla danej instancji P funkcji opisujących czasy wykonywania zadań, generuje w czasie $O(n)$ pierwszy szablon, mający taki sam porządek zadań stałych i zmiennych jak w instancji początkowej P.

```
1: function FirstTemplate(P)
2:     for r ← 1 to n do
3:         if P[r] jest stałe then H[r] ← 0
4:         else H[r] ← 1
5:         end if
6:     end for
7:     return H
```

Funkcja Reverse dla danego szablonu H i indeksów i i k, odwraca w czasie $O(n)$ porządek elementów w $H[i], H[i+1], \ldots, H[k]$. W pseudokodzie

tej funkcji przez symbol $\leftrightarrow$ oznaczamy operator zamiany miejscami dwóch elementów.

```
1: function Reverse(var H, i, k)
2:     while i < k do
3:         H[i] ↔ H[k]
4:         i ← i + 1
5:         k ← k − 1
6:     end while
7:     return H
```

Funkcja NextTemplate dla danego szablonu H i szablonu początkowego H_P uzyskanego przez funkcję FirstTemplate, tworzy następny szablon do H, o ile nowy szablon różni się od H_P; w przeciwnym wypadku generuje zerowy szablon $H_0 := [0, 0, \ldots, 0]$. Funkcja wykonuje się w czasie $O(n)$, ponieważ obie pętle '**while**' w liniach 3–4 i 7–8 oraz funkcja Reverse w linii 12 potrzebują czasu co najwyżej $O(n)$.

```
1: function NextTemplate(var H, H_P)
2:     i ← n
3:     while i > 1 and H[i − 1] ⩾ H[i] do i ← i − 1
4:     if i ≠ 1 then
5:         j ← i
6:         while j ⩽ n and H[j] ⩾ H[i − 1] do j ← j + 1
7:         j ← j − 1
8:         H[i − 1] ↔ H[j]
9:     end if
10:    H ← Reverse(H, i, n)
11:    if H = H_P then return H_0
12:    else return H
13:    end if
```

Funkcja CalcT dla danego szablonu H i czasu początkowego t_0, oblicza wartość T w czasie $O(n)$.

```
1: function CalcT(H, t_0)
2:     T ← t_0
3:     for r ← 1 to n do
4:         if H[r] = 1 then T ← T + (p_r + b ∗ r^a)
5:         else T ← T + p_r
6:     end for
7:     return T
```

Działanie wymienionych funkcji ilustruje następujący przykład.

Przykład 4.3. Danych jest $n = 5$ zadań. W tym $k = 2$ zadań J_1 i J_5 ze stałymi czasami wykonywania odpowiednio postaci $p_1 = 1$ i $p_5 = 5$. Zadania J_2, J_3 oraz J_4 są zmienne i mają odpowiednio czasy wykonywania postaci $p_2 = 2 + 3 * r^2$, $p_3 = 3 + 3 * r^2$ oraz $p_4 = 4 + 3 * r^2$.

Opiszemy teraz działanie poszczególnych funkcji.

FirstTemplate Dla danej instancji $P = [p_1, p_2, p_3, p_4, p_5]$ przy użyciu funkcji FirstTemplate otrzymujemy pierwszy szablon $H = [0, 1, 1, 1, 0]$.

NextTemplate i Reverse Na wejściu mamy dane szablon $H = [0, 1, 1, 1, 0]$ i szablon początkowy $H_P = [0, 1, 1, 1, 0]$. Dla zmiennej $i = 5$ w pętli '**while**' analizujemy szablon od końca szukając takiej pozycji i dla której wartość $H[i-1] < H[i]$. W naszym przypadku $i = 2$. Następnie szukamy takiej pozycji j w szablonie H, dla której $H[j] < H[i-1]$. W naszym przypadku $j = 4$. Zamieniamy miejscami w szablonie H wartości na pozycjach $H[1] \leftrightarrow H[4]$ otrzymując szablon $H = [1, 1, 1, 0, 0]$. Następnie przy użyciu funkcji Reverse(H,2,5) otrzymujemy następny w kolejności leksykograficznej szablon $H = [1, 0, 0, 1, 1]$. Ponieważ jest on różny od szablonu H_P funkcja NextTemplate generuje szablon H.

CalcT Dla danego szablonu $H = [0, 1, 1, 1, 0]$, czasu rozpoczęcia $t_0 = 0$ oraz czasów wykonywania zadań funkcja CalcT oblicza czas zakończenia ostatniego zadania $T = 1 + 2 + 3 * 2^2 + 3 + 3 * 3^2 + 4 + 3 * 4^2 + 5 = 102$.

Funkcje te są wykorzystywane w poniższym algorytmie dla problemu $1|p_{j,r}(t) \in \{p_j + b * r^a, n-k; p_j, k\}, prec|f_{\max}$. W pseudokodzie tego algorytmu (patrz Algorytm 4.3), przez G_H, σ_H i F_H oznaczamy odpowiednio kopię acyklicznego grafu skierowanego $G = (V, A)$ opisującego ograniczenia kolejnościowe między zadaniami, początkowe puste uszeregowanie i minimalny koszt odpowiadający szablonowi H. Symbol $F_{\min}$ oznacza minimalny koszt wśród wszystkich badanych szablonów.

Algorytm składa się z dwóch pętli '**while**' i jego główna idea jest następująca. Na początku, przy pomocy funkcji FirstTemplate przypisujemy do szablonu H początkowy szablon H_P i ustawiamy wartość zmiennej $F_{\min}$ (linie 1–3). Zewnętrzna pętla '**while**' działa dla wszystkich możliwych do uzyskania szablonów. W każdej iteracji tej pętli (linie 4–32) aktualizujemy wartości F_H, σ_H i G_H (linie 5–7), obliczamy nową wartość T (linia 8) i wykonujemy wewnętrzną pętle '**while**' (linie 10–26).

Wewnętrzna pętla '**while**' służy do znalezienia uszeregowania o najmniejszej wartości funkcji kosztu, zgodnego z ograniczeniami kolejnościowymi między zadaniami oraz aktualnie rozpatrywanym szablonem. W tym celu w wewnętrznej pętli wybieramy spośród zadań bez następnika to zadanie, które ma najmniejszy koszt na danej pozycji (linia 12) i umieszczamy je na początku już uszeregowanych zadań (linia 17). Musimy pamiętać by typ wybranego zadania był zgodny z typem w danym szablonie H na pozycji, którą rozpatrujemy (linia 13), w przeciwnym razie otrzymane uszeregowanie nie byłoby zgodne z rozpatrywanym szablonem. Postępując podobnie z pozostałymi zadaniami, aż uszeregujemy je wszystkie. Wewnętrzna pętla może zwrócić uszeregowania z najmniejszą wartością funkcji kosztu dla danego szablonu lub może zostać przerwana gdy znajdziemy zadanie, które jest niezgodne z analizowanym szablonem lub wartość funkcji kosztu będzie większa, niż ta, która aktualnie pamiętamy w zmiennej $F_{\min}$.

Przy użyciu funkcji NextTemplate generujemy następny szablon (linia 31). Jeżeli nie zwróci ona szablonu zerowego H_0, to przechodzimy do nowej iteracji zewnętrznej pętli '**while**' i powtarzamy wszystkie omówione kroki. W przeciwnym przypadku oznacza to, że rozpatrzyliśmy wszystkie szablony i algorytm generuje optymalne uszeregowanie $\sigma^\star$ z minimalnym kosztem $F_{\min}$.

Algorytm 4.3 dla problemu $1|p_{j,r}(t) \in \{p_j + b * r^a, n-k; p_j, k\}, prec|f_{\max}$

```
1: H_P ← FirstTemplate(P)
2: H ← H_P
3: F_min ← ∞
4: while H ≠ H_0 do
5:     F_H ← −∞
6:     σ_H ← (∘)
7:     G_H ← G
8:     T ← CalcT(H, t_0)
9:     r ← n
10:    while V(G_H) ≠ ∅ do
11:        NS(G_H) ← {j ∈ V(G_H) : deg⁺(j) = 0}
12:        Znajdź k ∈ NS(G_H) gdzie f_k(T) = min{f_j(T) : j ∈ NS(G_H)}
13:        if zadanie J_k jest typu H[r] and f_k < F_min then
14:            if zadanie J_k jest zmienne then T ← T − (p_k + b * r^a)
15:            else T ← T − p_k
16:            end if
17:            σ_H ← (k|σ_H)
18:            V(G_H) ← V(G_H) \ {k}
19:            A(G_H) ← A(G_H) \ {(i, k) : i ∈ V(G_H)}
20:            r ← r − 1
21:            if F_H < f_k then F_H ← f_k
22:            end if
23:        else
24:            break
25:        end if
26:    end while
27:    if F_min > F_H and r = 0 then
28:        F_min ← F_H
29:        σ* ← σ_H
30:    end if
31:    H ← NextTemplate(H, H_P)
32: end while
33: return σ*
```

Twierdzenie 4.3. *Algorytm 4.3 konstruuje optymalne uszeregowanie dla problemu* $1|p_{j,r}(t) \in \{p_j + b * r^a, n-k; p_j, k\}, prec|f_{\max}$ *w czasie* $O\left(n^{k+2}\right)$.

Dowód. Zauważmy, że dla naszego problemu zachodzi również własność opisana w Lemacie 2.1. Dlatego też w dalszej części zakładamy, że analizujemy tylko uszeregowania bez przestojów.

Następnie zauważmy, że Algorytm 4.3 generuje wszystkie możliwe szablony i każdy z nich jest rozpatrywany dokładnie raz. To wynika z faktu, że używamy w tym celu (linia 31) funkcji NextTemplate, która w sposób

standardowy generuje wszystkie permutacje w porządku leksykograficznym (patrz, n.p., Knuth [15]). Zaczynamy od szablonu H_P skonstruowanego przy pomocy funkcji FirstTemplate dla danej instancji początkowej P. Następnie badamy szablon od jego ostatniego elementu porównując każde dwa sąsiadujące elementy aby znaleźć największy indeks i taki, że $H[i-1] \geqslant H[i]$. Jeżeli taki indeks istnieje, to szukamy największego indeksu j takiego, że $H[j] \geqslant H[i-1]$ i zamieniamy miejscami $H[j]$ z $H[i-1]$. To powoduje, że podciąg $H[i]$,$H[i+1]$,$\ldots$,$H[n]$ jest posortowany nierosnąco. Następny szablon jest generowany na podstawie aktualnie rozpatrywanego szablonu poprzez odwrócenie kolejności elementów w tym podciągu. W podobny sposób postępujemy z kolejnymi szablonami tak długo, o ile warunek w linii 13, który zapobiega dwukrotnemu egzaminowaniu tego samego szablonu, nie jest spełniony. Jeżeli warunek ten jest spełniony, wtedy funkcja NextTemplate tworzy zerowy szablon H_0 i algorytm kończy działanie, generując uszeregowanie $\sigma^\star$. Zatem wszystkie możliwe szablony są wygenerowane i każdy z nich jest sprawdzony jedynie raz.

Zauważmy, że dla danego szablonu H wartość T nie zależy od porządku zadań tego samego typu. Faktycznie niech w szablonie H zadania tego samego typu będą na pozycjach $k_1, k_1+1, \ldots, k_2$. Wtedy $T = t_0 + \sum_{r=k_1}^{k_2}(p_r + b * r^a)$ i wartość tej sumy nie zależy od kolejności jej składników.

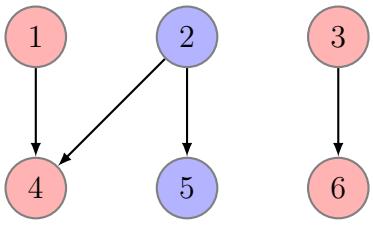
Na koniec zauważmy, że dla każdego szablonu Algorytm 4.3 generuje jedynie dopuszczalne uszeregowania, które nie naruszają ograniczeń kolejnościowych między zadaniami, oraz że linie 8–26 zawierają pseudokod Algorytmu 4.1 z tylko jednym dodatkowym warunkiem w linii 13. Warunek ten ma za zadanie eliminować uszeregowania niezgodne z aktualnie rozpatrywanym szablonem i kończyć działanie bieżącej iteracji, w sytuacji gdy aktualnie największy koszt jest większy, niż poprzednio zapisany w zmiennej $F_{\min}$. Dlatego też, zmodyfikowany Algorytm 4.1 generuje uszeregowanie σ_H o najmniejszej maksymalnej wartości funkcji kosztu i zapisuje ją w zmiennej F_H. Jeżeli $F_H < F_{\min}$, to $F_{\min}$ przyjmuje wartość F_H i $\sigma^\star$ staję się σ_H. Zatem, Algorytm 4.3 generuje optymalne uszeregowanie $\sigma^\star$.

Złożoność obliczeniowa Algorytmu 4.3 może zostać oszacowana w na-

stępujący sposób. Linia 1 potrzebuje czasu $O(n)$. Musimy przeanalizować $\binom{n}{k}$ szablonów dla k stałych zadań, więc pętla '**while**' w linii 4 wykona się w czasie $O\left(n^k\right)$. Zauważmy również, że dodanie linii 13–14 do Algorytmu 4.1 nie zmienia jego czasu pracy $O(n^2)$ oraz, że linia 31 potrzebuje czasu $O(n)$. Złożoność obliczeniowa Algorytmu 4.3 wynosi zatem $O\left(n^{k+2}\right)$. □

Zilustrujemy działanie Algorytmu 4.3 następującym przykładem.

Przykład 4.4. Danych jest $n = 6$ zadań w tym $k = 4$ zadań J_1, J_3, J_4 oraz J_6 ze stałymi czasami wykonywania odpowiednio postaci $p_1 = 2$, $p_3 = 3$, $p_4 = 4$ oraz $p_6 = 6$. Zadania J_2 i J_5 są zmienne i mają odpowiednio podstawowe czasy wykonywania postaci $p_2 = 1$ i $p_5 = 5$ wraz z indeksem starzenia się $a = 2$ i wspólnym czynnikiem starzenia się $b = 1$. Czas rozpoczęcia $t_0 = 0$. Funkcje kosztów są następującej postaci $f_1 = 0.3 \times C_1 + 25$, $f_2 = C_2$, $f_3 = 0.5 \times C_3 + 8$, $f_4 = 0.03 \times C_4^2$, $f_5 = 4\sqrt{C_5} + 6$ oraz $f_6 = 0.75 \times C_6$. Ograniczenia kolejnościowe między zadaniami przedstawione są na Rysunku 4.5, gdzie kolorem czerwonym i niebieskim oznaczono odpowiednio zadania ze stałymi i zmiennymi czasami wykonywania.



Rysunek 4.5. Ograniczenia kolejnościowe w Przykładzie 4.4

Musimy rozważyć $\binom{n}{k} = \binom{6}{4} = 15$ unikalnych szablonów. Używając procedury FirstTemplate generujemy początkowy szablon H. Ten szablon ma postać $H = [0, 1, 0, 0, 1, 0]$ i $F_{\min} = \infty$. W każdej iteracji ustalamy $F_H = -\infty$, $\sigma_H = (\circ)$, G_H jako kopię acyklicznego grafu skierowanego G i $r = 6$.

Opiszemy teraz działanie Algorytmu 4.3 dla każdego szablonu, prezentując kolejne iteracje potrzebne do znalezienia optymalnego uszeregowania.

Szablon nr 1 – $H = (0, 1, 0, 0, 1, 0)$

Iteracja 1 Obliczamy wartość czasu zakończenia ostatniego zadania $T = 50$ i ustawiamy pozycję $r = 6$. Konstruujemy zbiór zadań bez następników $NS(G_H) = \{4, 5, 6\}$. Następnie obliczamy wartości funkcji kosztów dla zadań ze zbioru $NS(G_H)$. Ponieważ $f_5(50) = 34.28 < f_6(50) = 37.5 < f_4(50) = 75$ wybieramy zadanie J_5. Typ zadania nie jest zgodny z $H[6]$. Przerywamy iterację i generujemy następny szablon.

i	H	T	$NS(G_H)$	$F_{\min}$	F_H	r	σ_H
1	$H(0, 1, 0, 0, 1, 0)$	50	$(4, 5, 6)$	∞	$-\infty$	6	$(\circ)$

Tabela 4.1. Podsumowanie przebiegu pierwszej iteracji Algorytmu 4.3

Szablon nr 2 – $H = (0, 1, 0, 1, 0, 0)$

Iteracja 1 Obliczamy długość uszeregowania $T = 41$ oraz konstruujemy zbiór $NS(G_H) = \{4, 5, 6\}$. Dla zadań z tego zbioru wartości funkcji kosztów wynoszą $f_6(41) = 30.75 < f_5(41) = 31.61 < f_4(41) = 50.43$, więc wybieramy zadanie J_6. Typ zadania J_6 jest zgodny z $H[6]$ oraz wartość $f_6(41) < F_{\min}$, więc umieszczamy je w uszeregowaniu $\sigma_H = (6)$. Aktualizujemy wartość $T = 41 - 6 = 35$ oraz zbiór wierzchołków $V(G_H)$ i łuków $A(G_H)$. Ustawiamy pozycję $r = 5$ oraz wartość zmiennej $F_H = 30.75$.

Iteracja 2 Konstruujemy zbiór zadań $NS(G_H) = \{3, 4, 5\}$. Ponownie obliczamy wartości funkcji kosztów dla zadań z tego zbioru. Wybieramy zadanie J_3, ponieważ $f_3(35) = 25.5 < f_5(35) = 29.66 < f_4(35) = 36.75$. Typ zadania J_3 jest zgodny z $H[5]$ oraz wartość $f_3(35) < F_{\min}$, więc umieszczamy je w uszeregowaniu $\sigma_H = (3, 6)$. Aktualizujemy wartość $T = 35 - 3 = 32$, zbiory $V(G_H)$ i $A(G_H)$ oraz ustawiamy pozycję $r = 4$.

Iteracja 3 Konstruujemy zbiór zadań $NS(G_H) = \{4, 5\}$. Wybieramy zadanie J_5, ponieważ $f_5(32) = 28.6 < f_4(32) = 30.72$. Typ zadania J_5 jest zgodny z $H[4]$ oraz wartość $f_5(32) < F_{\min}$, więc umieszczamy je w uszeregowaniu $\sigma_H = (5, 3, 6)$. Aktualizujemy wartość $T = 32 - 5 - 16 = 11$, zbiory $V(G_H)$ i $A(G_H)$ oraz ustawiamy pozycję $r = 3$.

Iteracja 4 Konstruujemy zbiór zadań $NS(G_H) = \{4\}$. Wybieramy zadanie J_4. Ponieważ $f_4(11) = 3.63 < F_{\min}$ oraz typ zadania J_4 jest zgodny z ak-

tualnie analizowanym szablonem, więc umieszczamy je w uszeregowaniu $\sigma_H = (4, 5, 3, 6)$. Aktualizujemy wartość $T = 11 - 4 = 7$, zbiory $V(G_H)$ i $A(G_H)$ oraz ustawiamy pozycję $r = 2$.

Iteracja 5 Konstruujemy zbiór zadań $NS(G_H) = \{1, 2\}$. Wybieramy zadanie J_2, ponieważ $f_2(7) = 7 < f_1(7) = 27.10$. Typ zadania J_2 jest zgodny z $H[2]$ oraz wartość $f_2(7) < F_{\min}$, więc umieszczamy je w uszeregowaniu $\sigma_H = (2, 4, 5, 3, 6)$. Aktualizujemy wartość $T = 7 - 2 - 4 = 1$, zbiory $V(G_H)$ i $A(G_H)$ oraz ustawiamy pozycję $r = 1$.

Iteracja 6 Konstruujemy zbiór zadań $NS(G_H) = \{1\}$. Wybieramy zadanie J_1. Ponieważ $f_1(1) = 25.30 < F_{\min}$ oraz typ zadania J_1 jest zgodny z aktualnie analizowanym szablonem, więc umieszczamy je w uszeregowaniu $\sigma_H = (1, 2, 4, 5, 3, 6)$. Aktualizujemy wartość $T = 0$, zbiory $V(G_H)$ i $A(G_H)$ oraz ustawiamy pozycję $r = 0$.

Poprawny szablon Dla analizowanego szablonu znaleźliśmy uszeregowanie z najmniejszą wartością maksymalnego kosztu $F_H = 30.75$. Ponieważ $F_H = 30.75 < F_{\min} = \infty$, więc $\sigma^\star \leftarrow (1, 2, 4, 5, 3, 6)$ oraz $F_{\min} \leftarrow 30.75$.

i	H	T	$NS(G_H)$	$F_{\min}$	F_H	r	σ_H
1	$H(0,1,0,1,0,0)$	41	$(4,5,6)$	∞	30.75	6	(6)
2	$H(0,1,0,1,0,0)$	35	$(3,4,5)$	∞	30.75	5	$(3,6)$
3	$H(0,1,0,1,0,0)$	32	$(4,5)$	∞	30.75	4	$(5,3,6)$
4	$H(0,1,0,1,0,0)$	11	(4)	∞	30.75	3	$(4,5,3,6)$
5	$H(0,1,0,1,0,0)$	7	$(1,2)$	∞	30.75	2	$(2,4,5,3,6)$
6	$H(0,1,0,1,0,0)$	1	(1)	30.75	30.75	1	$(1,2,4,5,3,6)$

Tabela 4.2. Podsumowanie przebiegu drugiej iteracji Algorytmu 4.3

Szablon nr 3 – $H = (0, 1, 1, 0, 0, 0)$

Iteracja 1 Obliczamy długość uszeregowania $T = 34$ oraz konstruujemy zbiór $NS(G_H) = \{4, 5, 6\}$. Wartości funkcji kosztów dla zadań z tego zbioru wynoszą $f_6(34) = 25.5 < f_5(34) = 29.3 < f_4(34) = 34.7$, więc wybieramy zadanie J_6. Typ zadania J_6 jest zgodny z $H[6]$ oraz wartość $f_6(34) < F_{\min} = 30.75$, więc umieszczamy je w uszeregowaniu $\sigma_H = (6)$. Aktualizujemy wartość $T = 34 - 6 = 28$ oraz zbiory $V(G_H)$ i $A(G_H)$. Ustawiamy pozycję $r = 5$ oraz wartość zmiennej $F_H = 25.5$.

Iteracja 2 Konstruujemy zbiór zadań $NS(G_H) = \{3, 4, 5\}$ i obliczamy wartości funkcji kosztów dla zadań z tego zbioru. Wybieramy zadanie J_3, ponieważ $f_3(28) = 22 < f_4(28) = 23.52 < f_5(28) = 27.17$. Typ zadania J_3 jest zgodny z $H[5]$ oraz wartość $f_3(28) < F_{\min}$, więc umieszczamy je w uszeregowaniu $\sigma_H = (3, 6)$. Aktualizujemy wartość $T = 28 - 3 = 25$, zbiory $V(G_H)$ i $A(G_H)$ oraz ustawiamy pozycję $r = 4$.

Iteracja 3 Konstruujemy zbiór $NS(G_H) = \{4, 5\}$. Wybieramy zadanie J_5, ponieważ $f_4(25) = 18.75 < f_5(25) = 26$. Typ zadania J_4 jest zgodny z aktualnie analizowanym szablonem oraz wartość $f_4(25) < F_{\min}$, więc umieszczamy je w uszeregowaniu $\sigma_H = (4, 3, 6)$. Aktualizujemy wartość $T = 25 - 4 = 21$, zbiory $V(G_H)$ i $A(G_H)$ oraz ustawiamy pozycję $r = 3$.

Iteracja 4 Konstruujemy zbiór zadań $NS(G_H) = \{1, 5\}$. Wybieramy zadanie J_5, ponieważ $f_5(21) = 24.33 < f_1(21) = 31.30$. Typ zadania J_5 jest zgodny z $H[3]$ oraz wartość $f_5(21) < F_{\min}$, więc umieszczamy je w uszeregowaniu $\sigma_H = (5, 4, 3, 6)$. Aktualizujemy wartość $T = 21 - 5 - 9 = 7$, zbiory $V(G_H)$ i $A(G_H)$ oraz ustawiamy pozycję $r = 2$.

Iteracja 5 Konstruujemy zbiór zadań $NS(G_H) = \{1, 2\}$. Wybieramy zadanie J_2, ponieważ $f_2(7) = 7 < f_1(7) = 27.10$. Typ zadania J_2 jest zgodny z $H[2]$ oraz wartość $f_2(7) < F_{\min}$, więc umieszczamy je w uszeregowaniu $\sigma_H = (2, 5, 4, 3, 6)$. Aktualizujemy wartość $T = 7 - 2 - 4 = 1$, zbiory $V(G_H)$ i $A(G_H)$ oraz ustawiamy pozycję $r = 1$.

Iteracja 6 Konstruujemy zbiór zadań $NS(G_H) = \{1\}$. Wybieramy zadanie J_1. Ponieważ $f_1(1) = 25.30 < F_{\min}$ oraz typ zadania J_1 jest zgodny z aktualnie analizowanym szablonem, więc umieszczamy je w uszeregowaniu $\sigma_H = (1, 2, 5, 4, 3, 6)$. Aktualizujemy wartość $T = 0$, zbiory $V(G_H)$ i $A(G_H)$ oraz ustawiamy pozycję $r = 0$.

Poprawny szablon Dla analizowanego szablonu znaleźliśmy uszeregowanie z najmniejszą wartością maksymalnego kosztu $F_H = 25.5$. Ponieważ $F_H = 25.5 < F_{\min} = 30.75$, więc $\sigma^\star \leftarrow (1, 2, 5, 4, 3, 6)$ oraz $F_{\min} \leftarrow 25.5$.

i	H	T	$NS(G_H)$	$F_{\min}$	F_H	r	σ_H
1	$H(0,1,1,0,0,0)$	34	$(4,5,6)$	30.75	25.5	6	(6)
2	$H(0,1,1,0,0,0)$	28	$(3,4,5)$	30.75	25.5	5	$(3,6)$
3	$H(0,1,1,0,0,0)$	25	$(4,5)$	30.75	25.5	4	$(4,3,6)$
4	$H(0,1,1,0,0,0)$	21	$(1,5)$	30.75	25.5	3	$(5,4,3,6)$
5	$H(0,1,1,0,0,0)$	7	$(1,2)$	30.75	25.5	2	$(2,5,4,3,6)$
6	$H(0,1,1,0,0,0)$	1	(1)	25.5	25.5	1	$(1,2,5,4,3,6)$

Tabela 4.3. Podsumowanie przebiegu trzeciej iteracji Algorytmu 4.3

Szablon nr 4 – $H = (1,0,0,0,0,1)$

Iteracja 1 Czas zakończenia ostatniego zadania $T = 58$. Konstruujemy zbiór zadań $NS(G_H) = \{4,5,6\}$, obliczamy wartości funkcji kosztów $f_5(58) = 36.46 < f_6(58) = 43.5 < f_4(58) = 100.92$ i wybieramy zadanie J_5. Typ zadania J_5 jest zgodny z $H[6]$ ale wartość $f_5(58) > F_{\min} = 25.5$, więc przerywamy iterację i generujemy następny szablon.

i	H	T	$NS(G_H)$	$F_{\min}$	F_H	r	σ_H
1	$H(1,0,0,0,0,1)$	58	$(4,5,6)$	25.5	$-\infty$	6	$(\circ)$

Tabela 4.4. Podsumowanie przebiegu czwartej iteracji Algorytmu 4.3

Szablon nr 5 – $H = (1,0,0,0,1,0)$

Iteracja 1 Czas zakończenia ostatniego zadania $T = 58$. Konstruujemy zbiór zadań $NS(G_H) = \{4,5,6\}$, obliczamy wartości funkcji kosztów $f_5(47) = 33.42 < f_6(47) = 35.25 < f_4(47) = 66.27$ i wybieramy zadanie J_5. Typ zadania J_5 jest zgodny z $H[6]$ ale wartość $f_5(47) > F_{\min} = 25.5$. Przerywamy iterację i generujemy następny szablon.

i	H	T	$NS(G_H)$	$F_{\min}$	F_H	r	σ_H
1	$H(1,0,0,0,1,0)$	47	$(4,5,6)$	25.5	$-\infty$	6	$(\circ)$

Tabela 4.5. Podsumowanie przebiegu piątej iteracji Algorytmu 4.3

Szablon nr 6 – $H = (1,0,0,1,0,0)$

Iteracja 1 Czas zakończenia ostatniego zadania $T = 38$. Konstruujemy zbiór $NS(G_H) = \{4,5,6\}$. Wartości funkcji kosztów dla zadań z tego

zbioru wynoszą $f_6(38) = 28.5 < f_5(38) = 30.66 < f_4(38) = 43.32$. Typ zadania J_6 jest zgodny z $H[6]$, ale wartość $f_6(38) > F_{\min} = 25.5$, więc przerywamy iterację i generujemy następny szablon.

i	H	T	$NS(G_H)$	$F_{\min}$	F_H	r	σ_H
1	$H(1,0,0,1,0,0)$	38	$(4,5,6)$	25.5	$-\infty$	6	$(\circ)$

Tabela 4.6. Podsumowanie przebiegu szóstej iteracji Algorytmu 4.3

Szablon nr 7 – $H = (1,0,1,0,0,0)$

Iteracja 1 Obliczamy długość uszeregowania $T = 31$ oraz konstruujemy zbiór $NS(G_H) = \{4,5,6\}$. Wartości funkcji kosztów wynoszą odpowiednio $f_6(31) = 23.25 < f_5(31) = 28.27 < f_4(31) = 28.83$, więc wybieramy zadanie J_6. Typ zadania J_6 jest zgodny z $H[6]$ oraz $f_6(31) < F_{\min} = 25.5$, więc umieszczamy je w uszeregowaniu $\sigma_H = (6)$. Aktualizujemy wartość $T = 31 - 6 = 25$ oraz zbiór wierzchołków $V(G_H)$ i łuków $A(G_H)$. Ustawiamy pozycję $r = 5$ oraz wartość zmiennej $F_H = 23.25$.

Iteracja 2 Konstruujemy zbiór $NS(G_H) = \{3,4,5\}$. Ponownie obliczamy wartości funkcji kosztów dla zadań z tego zbioru. Wybieramy zadanie J_4, ponieważ $f_4(25) = 18.75 < f_3(25) = 20.5 < f_5(25) = 26$. Typ zadania J_4 jest zgodny z $H[5]$ oraz wartość $f_4(25) < F_{\min}$, więc umieszczamy je w uszeregowaniu $\sigma_H = (4,6)$. Aktualizujemy wartość $T = 25 - 4 = 21$, zbiory $V(G_H)$ i $A(G_H)$ oraz ustawiamy pozycję $r = 4$.

Iteracja 3 Konstruujemy zbiór $NS(G_H) = \{1,3,5\}$ i obliczamy wartości funkcji kosztów $f_3(21) = 18.5 < f_5(21) = 24.33 < f_1(21) = 31.3$. Typ zadania J_3 jest zgodny z analizowanym szablonem oraz wartość $f_3(21) < F_{\min}$, więc umieszczamy je w $\sigma_H = (3,4,6)$. Aktualizujemy wartość $T = 21 - 3 = 18$, zbiory $V(G_H)$ i $A(G_H)$ oraz pozycję $r = 3$.

Iteracja 4 Konstruujemy zbiór zadań $NS(G_H) = \{1,5\}$. Wybieramy zadanie J_5, ponieważ $f_5(18) = 22.87 < f_1(18) = 30.40$. Typ zadania J_5 jest zgodny z zgodny z $H[3]$ oraz wartość $f_5(18) < F_{\min}$, więc umieszczamy je w uszeregowaniu $\sigma_H = (5,3,4,6)$. Aktualizujemy wartość $T = 18-5-9 = 4$, zbiory $V(G_H)$ i $A(G_H)$ oraz ustawiamy pozycję $r = 2$.

Iteracja 5 Konstruujemy zbiór $NS(G_H) = \{1, 2\}$. Wybieramy zadanie J_2, ponieważ $f_2(4) = 4 < f_1(4) = 26.20$. Typ zadania J_2 jest niezgodny z H [2]. Przerywamy iterację i generujemy następny szablon.

i	H	T	$NS(G_H)$	$F_{\min}$	F_H	r	σ_H
1	$H(1,0,1,0,0,0)$	31	$(4,5,6)$	25.5	23.25	6	(6)
2	$H(1,0,1,0,0,0)$	25	$(3,4,5)$	25.5	23.25	5	$(4,6)$
3	$H(1,0,1,0,0,0)$	21	$(4,5)$	25.5	23.25	4	$(3,4,6)$
4	$H(1,0,1,0,0,0)$	18	$(1,5)$	25.5	23.25	3	$(5,3,4,6)$
5	$H(1,0,1,0,0,0)$	4	$(1,2)$	25.5	23.25	2	$(5,3,4,6)$

Tabela 4.7. Podsumowanie przebiegu siódmej iteracji Algorytmu 4.3

Szablon nr 8 – $H = (1,1,0,0,0,0)$

Iteracja 1 Obliczamy długość uszeregowania $T = 26$ oraz konstruujemy zbiór $NS(G_H) = \{4,5,6\}$. Wartości funkcji kosztów wynosza odpowiednio $f_6(26) = 19.5 < f_4(26) = 20.28 < f_5(26) = 26.40$, więc wybieramy zadanie J_6. Typ zadania J_6 jest zgodny z H [6] oraz $f_6(26) < F_{\min} = 25.5$, więc umieszczamy je w uszeregowaniu $\sigma_H = (6)$. Aktualizujemy wartość $T = 26 - 6 = 20$ oraz zbiór wierzchołków $V(G_H)$ i łuków $A(G_H)$. Ustawiamy pozycję $r = 5$ oraz wartość zmiennej $F_H = 19.5$.

Iteracja 2 Zbiór zadań $NS(G_H) = \{3,4,5\}$. Ponownie obliczamy wartości funkcji kosztów dla zadań z tego zbioru. Wybieramy zadanie J_4, ponieważ $f_4(20) = 12 < f_3(20) = 18 < f_5(20) = 23.89$. Typ zadania J_4 jest zgodny z H [5] oraz wartość $f_4(20) < F_{\min}$, więc umieszczamy je w uszeregowaniu $\sigma_H = (4,6)$. Aktualizujemy wartość $T = 20 - 4 = 16$, zbiory $V(G_H)$ i $A(G_H)$ oraz ustawiamy pozycję $r = 4$.

Iteracja 3 Zbiór zadań $NS(G_H) = \{1,3,5\}$. Wybieramy zadanie J_3, ponieważ $f_3(16) = 16 < f_5(16) = 22 < f_1(16) = 29.8$. Typ zadania J_3 jest zgodny z analizowanym szablonem oraz wartość $f_3(16) < F_{\min}$, więc umieszczamy je w uszeregowaniu $\sigma_H = (3,4,6)$. Aktualizujemy wartość $T = 16 - 3 = 13$, zbiory $V(G_H)$ i $A(G_H)$ oraz ustawiamy pozycję $r = 3$.

Iteracja 4 Zbiór zadań $NS(G_H) = \{1,5\}$. Wybieramy zadanie J_5, ponieważ $f_5(13) = 20.42 < f_1(13) = 28.90$. Typ zadania J_5 jest niezgodny z H [3]. Przerywamy iterację i generujemy następny szablon.

i	H	T	$NS(G_H)$	$F_{\min}$	F_H	r	σ_H
1	$H(1,1,0,0,0,0)$	26	$(4,5,6)$	25.5	19.5	6	(6)
2	$H(1,1,0,0,0,0)$	20	$(3,4,5)$	25.5	19.5	5	$(4,6)$
3	$H(1,1,0,0,0,0)$	16	$(4,5)$	25.5	19.5	4	$(3,4,6)$
4	$H(1,1,0,0,0,0)$	13	$(1,5)$	25.5	19.5	3	$(3,4,6)$

Tabela 4.8. Podsumowanie przebiegu ósmej iteracji Algorytmu 4.3

Szablon nr 9 – $H = (0,0,0,0,1,1)$

Iteracja 1 Obliczamy długość uszeregowania $T = 82$ oraz konstruujemy zbiór $NS(G_H) = \{4,5,6\}$. Wartości funkcji kosztów wynoszą odpowiednio $f_5(82) = 42.22 < f_6(82) = 61.5 < f_4(82) = 201.72$, więc wybieramy zadanie J_5. Typ zadania J_5 jest zgodny z $H[6]$, ale $f_5(82) > F_{\min} = 25.5$, więc przerywamy iterację i generujemy następny szablon.

i	H	T	$NS(G_H)$	$F_{\min}$	F_H	r	σ_H
1	$H(0,0,0,0,1,1)$	82	$(4,5,6)$	25.5	$-\infty$	6	$(\circ)$

Tabela 4.9. Podsumowanie przebiegu dziewiątej iteracji Algorytmu 4.3

Szablon nr 10 – $H = (0,0,0,1,0,1)$

Iteracja 1 Obliczamy długość uszeregowania $T = 73$ oraz konstruujemy zbiór $NS(G_H) = \{4,5,6\}$. Wartości funkcji kosztów wynoszą odpowiednio $f_5(73) = 40.18 < f_6(73) = 54.75 < f_4(73) = 159.87$, więc wybieramy zadanie J_5. Typ zadania J_5 jest zgodny z $H[6]$, ale $f_5(73) > F_{\min} = 25.5$, więc przerywamy iterację i generujemy następny szablon.

i	H	T	$NS(G_H)$	$F_{\min}$	F_H	r	σ_H
1	$H(0,0,0,1,0,1)$	73	$(4,5,6)$	25.5	$-\infty$	6	$(\circ)$

Tabela 4.10. Podsumowanie przebiegu dziesiątej iteracji Algorytmu 4.3

Szablon nr 11 – $H = (0,0,0,1,1,0)$

Iteracja 1 Obliczamy długość uszeregowania $T = 62$ oraz konstruujemy zbiór $NS(G_H) = \{4,5,6\}$. Wartości funkcji kosztów wynoszą odpowiednio $f_5(62) = 37.5 < f_6(62) = 46.5 < f_4(62) = 115.32$, więc wybieramy zadanie J_5. Typ zadania J_5 jest zgodny z $H[6]$, ale $f_5(62) > F_{\min} = 25.5$, więc przerywamy iterację i generujemy następny szablon.

i	H	T	$NS(G_H)$	$F_{\min}$	F_H	r	σ_H
1	$H(0,0,0,1,1,0)$	62	$(4,5,6)$	25.5	$-\infty$	6	$(\circ)$

Tabela 4.11. Podsumowanie przebiegu jedenastej iteracji Algorytmu 4.3

Szablon nr 12 – $H = (0,0,1,0,0,1)$

Iteracja 1 Obliczamy długość uszeregowania $T = 66$ oraz konstruujemy zbiór $NS(G_H) = \{4,5,6\}$. Wartości funkcji kosztów wynoszą odpowiednio $f_5(66) = 38.5 < f_6(66) = 49.5 < f_4(66) = 130.68$, więc wybieramy zadanie J_5. Typ zadania J_5 jest zgodny z H [6], ale $f_5(66) > F_{\min} = 25.5$, więc przerywamy iterację i generujemy następny szablon.

i	H	T	$NS(G_H)$	$F_{\min}$	F_H	r	σ_H
1	$H(0,0,1,0,0,1)$	66	$(4,5,6)$	25.5	$-\infty$	6	$(\circ)$

Tabela 4.12. Podsumowanie przebiegu dwunastej iteracji Algorytmu 4.3

Szablon nr 13 – $H = (0,0,1,0,1,0)$

Iteracja 1 Obliczamy długość uszeregowania $T = 55$ oraz konstruujemy zbiór $NS(G_H) = \{4,5,6\}$. Wartości funkcji kosztów wynoszą odpowiednio $f_5(55) = 35.66 < f_6(55) = 41.25 < f_4(55) = 90.75$, więc wybieramy zadanie J_5. Typ zadania J_5 jest niezgodny z H [6] oraz $f_5(55) > F_{\min} = 25.5$, więc przerywamy iterację i generujemy następny szablon.

i	H	T	$NS(G_H)$	$F_{\min}$	F_H	r	σ_H
1	$H(0,0,1,0,1,0)$	55	$(4,5,6)$	25.5	$-\infty$	6	$(\circ)$

Tabela 4.13. Podsumowanie przebiegu trzynastej iteracji Algorytmu 4.3

Szablon nr 14 – $H = (0,0,1,1,0,0)$

Iteracja 1 Obliczamy długość uszeregowania $T = 46$ oraz konstruujemy zbiór $NS(G_H) = \{4,5,6\}$. Wartości funkcji kosztów wynoszą odpowiednio $f_5(46) = 33.13 < f_6(46) = 34.5 < f_4(46) = 63.48$, więc wybieramy zadanie J_5. Typ zadania J_5 jest niezgodny z H [6] oraz $f_5(46) > F_{\min} = 25.5$, więc przerywamy iterację i generujemy następny szablon.

i	H	T	$NS(G_H)$	$F_{\min}$	F_H	r	σ_H
1	$H(0,0,1,1,0,0)$	46	$(4,5,6)$	25.5	$-\infty$	6	$(\circ)$

Tabela 4.14. Podsumowanie przebiegu czternastej iteracji Algorytmu 4.3

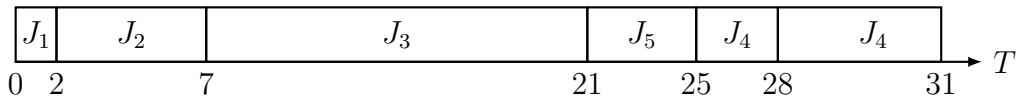
Szablon nr 15 – $H = (0, 1, 0, 0, 0, 1)$

Iteracja 1 Obliczamy długość uszeregowania $T = 61$. Konstruujemy zbiór zadań $NS(G_H) = \{4, 5, 6\}$, obliczamy wartości funkcji kosztów $f_5(61) = 37.24 < f_6(61) = 45.75 < f_4(61) = 111.63$, i wybieramy zadanie J_5. Typ zadania J_5 jest zgodny z $H[6]$, ale $f_5(61) > F_{\min} = 25.5$. Przerywamy iterację i generujemy następny szablon.

i	H	T	$NS(G_H)$	$F_{\min}$	F_H	r	σ_H
1	$H(0, 1, 0, 0, 0, 1)$	61	$(4, 5, 6)$	25.5	$-\infty$	6	$(\circ)$

Tabela 4.15. Podsumowanie przebiegu piętnastej iteracji Algorytmu 4.3

Dla danej instancji problemu, Algorytm 4.3 wygenerował optymalne uszeregowanie $\sigma^\star = (1, 2, 5, 4, 3, 6)$ z wartością $f_{\max} = 25.5$.



Rysunek 4.6. Optymalne uszeregowanie w Przykładzie 4.4

Kolejny z omawianych przez nas algorytmów służy rozwiązaniu problemu szeregowania postaci $1|p_{j,r}(t) \in \{t * r^a, n-k; p_j * t, k\}, prec|f_{\max}$, w którym czasy wykonywania zadań zależą zarówno od pozycji jak i od czasu rozpoczęcia zadania. Na wejściu i wyjściu algorytmu mamy podobne dane jak poprzednio, z dodatkowymi k proporcjonalnymi czasowo-zależnymi zadaniami opisanymi przez współczynniki wydłużenia p_j. Co więcej, wspomniany szablon H jest nadal n-wymiarowym wektorem 0–1, z tym, że $H[i] = 0$ ($H[i] = 1$) jeżeli J_i jest proporcjonalne (zmienne).

Dla tego problemu nadal zachodzi własność dotycząca przestojów z Lematu 2.1 i własność o niezmienności czasu zakończenia ostatniego zadania z Lematu 2.1, zatem możemy rozwiązać go modyfikując funkcje CalcT: linia 4 i 5 są teraz odpowiednio następującej postaci '**If** $H[r] = 0$ **then** $T \leftarrow T * (1 + p_r)$' i '**else** $T \leftarrow T * (1 + r^a)$'. W podobny sposób modyfikuje-

my Algorytm 4.3: linia 14 i 15 są teraz odpowiednio następującej postaci '**if** zadanie J_k jest proporcjonalne **then** $T \leftarrow \frac{T}{1+p_k}$' i '**else** $T \leftarrow \frac{T}{1+r^a}$'.

Algorytm 4.4 dla problemu $1|p_{j,r}(t) \in \{t * r^a, n-k; p_j * t, k\}, prec|f_{\max}$

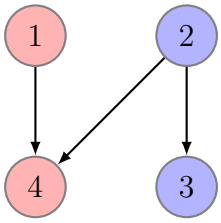
```
1:  H_P ← FirstTemplate(P)
2:  H ← H_P
3:  F_min ← ∞
4:  while H ≠ H_0 do
5:      F_H ← −∞
6:      σ_H ← (∘)
7:      G_H ← G
8:      T ← CalcT(H, t_0)
9:      r ← n
10:     while V(G_H) ≠ ∅ do
11:         NS(G_H) ← {j ∈ V(G_H) : deg⁺(j) = 0}
12:         Znajdź k ∈ NS(G_H) gdzie f_k(T) = min{f_j(T) : j ∈ NS(G_H)}
13:         if zadanie J_k jest typu H[r] and f_k < F_min then
14:             if zadanie J_k jest proporcjonalne then T ← T/(1+p_k)
15:             else T ← T/(1+r^a)
16:             end if
17:             σ_H ← (k|σ_H)
18:             V(G_H) ← V(G_H) \ {k}
19:             A(G_H) ← A(G_H) \ {(i, k) : i ∈ V(G_H)}
20:             r ← r − 1
21:             if F_H < f_k then F_H ← f_k
22:             end if
23:         else
24:             break
25:         end if
26:     end while
27:     if F_min > F_H and r = 0 then
28:         F_min ← F_H
29:         σ* ← σ_H
30:     end if
31:     H ← NextTemplate(H, H_P)
32: end while
33: return σ*
```

Twierdzenie 4.4. *Algorytm 4.4 konstruuje optymalne uszeregowanie dla problemu* $1|p_{j,r}(t) \in \{t * r^a, n-k; p_j * t, k\}, prec|f_{\max}$ *w czasie* $O\left(n^{k+2}\right)$.

Dowód Twierdzenia 4.4 pominięto, ponieważ jest podobny do dowodu Twierdzenia 4.3.

Zilustrujemy działanie Algorytmu 4.4 następującym przykładem.

Przykład 4.5. Danych jest $n = 4$ zadań z czasami wykonywania odpowiednio postaci $p_1 = p_4 = t * r^2$ oraz $p_2 = 2 * t$ i $p_3 = 3 * t$. Czas rozpoczęcia $t_0 = 1$. Funkcje kosztów są następującej postaci $f_1 = 0.3 \times C_1 + 5$, $f_2 = C_2$, $f_3 = 0.5 \times C_3 + 8$, $f_4 = 0.03 \times C_4^2$. Ograniczenia kolejnościowe przedstawione są na Rysunku 4.7, gdzie kolorem czerwonym i niebieskim oznaczono odpowiednio zadania z czasami wykonywania postaci $p_{j,r} = t * r^a$ i $p_{j,r} = p_j * t$.



Rysunek 4.7. Ograniczenia kolejnościowe w Przykładzie 4.5

Musimy rozważyć $\binom{n}{k} = \binom{4}{2} = 6$ unikalnych szablonów. Używając procedury FirstTemplate generujemy początkowy szablon H. Ten szablon ma postać $H = [1, 0, 0, 1]$ i $F_{\min} = \infty$. W każdej iteracji ustalamy $F_H = -\infty$, $\sigma_H = (\circ)$, G_H jako kopię acyklicznego grafu skierowanego G i $r = 4$.

Opiszemy teraz działanie Algorytmu 4.4 dla każdego szablonu, prezentując kolejne iteracje potrzebne do znalezienia optymalnego uszeregowania.

Szablon nr 1 – $H = (1, 0, 0, 1)$

Iteracja 1 Obliczamy wartość czasu zakończenia ostatniego zadania $T = 30$ i ustawiamy pozycję $r = 4$. Konstruujemy zbiór zadań bez następników $NS(G_H) = \{3, 4\}$. Następnie obliczamy wartości funkcji kosztów dla zadań ze zbioru $NS(G_H)$. Ponieważ $f_3(30) = 23 < f_4(30) = 27$ wybieramy zadanie J_3. Typ zadania nie jest zgodny z $H[4]$. Przerywamy iterację i generujemy następny szablon.

i	H	T	$NS(G_H)$	$F_{\min}$	F_H	r	σ_H
1	$H(1, 0, 0, 1)$	30	$(3, 4)$	∞	$-\infty$	4	$(\circ)$

Tabela 4.16. Podsumowanie przebiegu pierwszej iteracji Algorytmu 4.4

Szablon nr 2 – $H = (1, 0, 1, 0)$

Iteracja 1 Obliczamy długość uszeregowania $T = 32$ oraz konstruujemy zbiór $NS(G_H) = \{3, 4\}$. Następnie obliczamy wartości funkcji kosztów dla zadań ze zbioru $NS(G_H)$. Ponieważ $f_3(32) = 24 < f_4(32) = 30.72$ wybieramy zadanie J_3. Typ zadania J_3 jest zgodny z $H[4]$ oraz wartość $f_3(32) < F_{\min}$, więc umieszczamy je w uszeregowaniu $\sigma_H = (3)$. Aktualizujemy wartość $T = 32/4 = 8$ oraz zbiór wierzchołków $V(G_H)$ i łuków $A(G_H)$. Ustawiamy pozycję $r = 3$ oraz wartość zmiennej $F_H = 24$.

Iteracja 2 Konstruujemy zbiór zadań $NS(G_H) = \{4\}$. Wybieramy zadanie J_4. Ponieważ $f_4(8) = 1.92 < F_{\min}$ oraz typ zadania J_4 jest zgodny z aktualnie analizowanym szablonem, więc umieszczamy je w uszeregowaniu $\sigma_H = (4, 3)$. Aktualizujemy wartość $T = 8/\frac{4}{3} = 6$, zbiory $V(G_H)$ i $A(G_H)$ oraz ustawiamy pozycję $r = 2$.

Iteracja 3 Konstruujemy zbiór zadań $NS(G_H) = \{1, 2\}$. Wybieramy zadanie J_2, ponieważ $f_2(6) = 6 < f_1(6) = 6.8$. Typ zadania J_2 jest zgodny z $H[2]$ oraz wartość $f_2(6) < F_{\min}$, więc umieszczamy je w uszeregowaniu $\sigma_H = (2, 4, 3)$. Aktualizujemy wartość $T = 2$, zbiory $V(G_H)$ i $A(G_H)$ oraz ustawiamy pozycję $r = 1$.

Iteracja 4 Konstruujemy zbiór zadań $NS(G_H) = \{1\}$. Wybieramy zadanie J_1. Ponieważ $f_1(2) = 5.6 < F_{\min}$ oraz typ zadania J_1 jest zgodny z aktualnie analizowanym szablonem, więc umieszczamy je w uszeregowaniu $\sigma_H = (1, 2, 4, 3)$. Aktualizujemy wartość $T = 1$, zbiory $V(G_H)$ i $A(G_H)$ oraz ustawiamy pozycję $r = 0$.

Poprawny szablon Dla analizowanego szablonu znaleźliśmy uszeregowanie z najmniejszą wartością maksymalnego kosztu $F_H = 24$. Ponieważ $F_H = 24 < F_{\min} = \infty$, więc $\sigma^\star \leftarrow (1, 2, 4, 3)$ oraz $F_{\min} \leftarrow 24$.

i	H	T	$NS(G_H)$	$F_{\min}$	F_H	r	σ_H
1	$H(1,0,1,0)$	32	$(3,4)$	∞	24	4	(3)
2	$H(1,0,1,0)$	8	(4)	∞	24	3	$(4,3)$
3	$H(1,0,1,0)$	6	$(1,2)$	∞	24	2	$(2,4,3)$
4	$H(1,0,1,0)$	2	(1)	24	24	1	$(1,2,4,3)$

Tabela 4.17. Podsumowanie przebiegu drugiej iteracji Algorytmu 4.4

Szablon nr 3 – $H = (1, 1, 0, 0)$

Iteracja 1 Obliczamy wartość czasu zakończenia ostatniego zadania $T = 36$ i ustawiamy pozycję $r = 4$. Konstruujemy zbiór zadań bez następników $NS(G_H) = \{3, 4\}$. Następnie obliczamy wartości funkcji kosztów dla zadań ze zbioru $NS(G_H)$. Ponieważ $f_3(36) = 26 < f_4(36) = 38.88$ wybieramy zadanie J_3. Typ zadania J_3 jest zgodny z $H[4]$, ale wartość $f_3(36) > F_{\min}$. Przerywamy iterację i generujemy następny szablon.

i	H	T	$NS(G_H)$	$F_{\min}$	F_H	r	σ_H
1	$H(1, 1, 0, 0)$	36	$(3, 4)$	24	26	4	$(\circ)$

Tabela 4.18. Podsumowanie przebiegu trzeciej iteracji Algorytmu 4.4

Szablon nr 4 – $H = (0, 0, 1, 1)$

Iteracja 1 Obliczamy długość uszeregowania $T = 20$ oraz konstruujemy zbiór $NS(G_H) = \{3, 4\}$. Następnie obliczamy wartości funkcji kosztów dla zadań ze zbioru $NS(G_H)$. Ponieważ $f_4(20) = 12 < f_3(20) = 18$ wybieramy zadanie J_4. Typ zadania J_4 jest zgodny z $H[4]$ oraz wartość $f_4(20) < F_{\min}$, więc umieszczamy je w uszeregowaniu $\sigma_H = (4)$. Aktualizujemy wartość $T = 20/\frac{5}{4} = 16$ oraz zbiór wierzchołków $V(G_H)$ i łuków $A(G_H)$. Ustawiamy pozycję $r = 3$ oraz wartość zmiennej $F_H = 12$.

Iteracja 2 Konstruujemy zbiór zadań $NS(G_H) = \{1, 3\}$. Następnie obliczamy wartości funkcji kosztów dla zadań ze zbioru $NS(G_H)$. Ponieważ $f_1(16) = 9.8 < f_3(16) = 16$ wybieramy zadanie J_1. Typ zadania J_1 jest zgodny z $H[3]$ oraz wartość $f_1(16) < F_{\min}$, więc umieszczamy je w uszeregowaniu $\sigma_H = (1, 4)$. Aktualizujemy wartość $T = 16/\frac{4}{3} = 12$, zbiory $V(G_H)$ i $A(G_H)$ oraz ustawiamy pozycję $r = 2$.

Iteracja 3 Konstruujemy zbiór zadań $NS(G_H) = \{3\}$. Wybieramy zadanie J_3. Ponieważ $f_3(12) = 14 < F_{\min}$ oraz typ zadania J_3 jest zgodny z aktualnie analizowanym szablonem, więc umieszczamy je w uszeregowaniu $\sigma_H = (3, 1, 4)$. Aktualizujemy wartości $T = 12/4 = 3$ i $F_H = 14$, zbiory $V(G_H)$ i $A(G_H)$ oraz ustawiamy pozycję $r = 1$.

Iteracja 4 Konstruujemy zbiór zadań $NS(G_H) = \{2\}$. Wybieramy zadanie J_2. Ponieważ $f_2(3) = 3 < F_{\min}$ oraz typ zadania J_3 jest zgodny z aktualnie

analizowanym szablonem, więc umieszczamy je w uszeregowaniu $\sigma_H = (2,3,1,4)$. Aktualizujemy wartość $T = 1$, zbiory $V(G_H)$ i $A(G_H)$ oraz ustawiamy pozycję $r = 0$.

Poprawny szablon Dla analizowanego szablonu znaleźliśmy uszeregowanie z najmniejszą wartością maksymalnego kosztu $F_H = 14$. Ponieważ $F_H = 14 < F_{\min} = 24$, więc $\sigma^\star \leftarrow (2,3,1,4)$ oraz $F_{\min} \leftarrow 14$.

i	H	T	$NS(G_H)$	$F_{\min}$	F_H	r	σ_H
1	$H(0,0,1,1)$	20	$(3,4)$	24	12	4	(4)
2	$H(0,0,1,1)$	16	$(1,3)$	24	12	3	$(1,4)$
3	$H(0,0,1,1)$	12	(3)	24	14	2	$(3,1,4)$
4	$H(0,0,1,1)$	3	(2)	14	14	1	$(2,3,1,4)$

Tabela 4.19. Podsumowanie przebiegu czwartej iteracji Algorytmu 4.4

Szablon nr 5 – $H = (0,1,0,1)$

Iteracja 1 Obliczamy wartość czasu zakończenia ostatniego zadania $T = 22.5$ i ustawiamy pozycję $r = 4$. Konstruujemy zbiór zadań bez następników $NS(G_H) = \{3,4\}$. Następnie obliczamy wartości funkcji kosztów dla zadań ze zbioru $NS(G_H)$. Ponieważ $f_4(22.5) = 15.19 < f_3(22.5) = 19.25$ wybieramy zadanie J_4. Typ zadania J_3 jest zgodny z $H[4]$, ale wartość $f_3(36) > F_{\min}$. Przerywamy iterację i generujemy następny szablon.

i	H	T	$NS(G_H)$	$F_{\min}$	F_H	r	σ_H
1	$H(0,1,0,1)$	22.5	$(3,4)$	14	15.19	4	$(\circ)$

Tabela 4.20. Podsumowanie przebiegu piątej iteracji Algorytmu 4.4

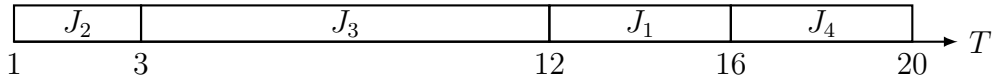
Szablon nr 6 – $H = (0,1,1,0)$

Iteracja 1 Obliczamy wartość czasu zakończenia ostatniego zadania $T = 24$ i ustawiamy pozycję $r = 4$. Konstruujemy zbiór zadań bez następników $NS(G_H) = \{3,4\}$. Następnie obliczamy wartości funkcji kosztów dla zadań ze zbioru $NS(G_H)$. Ponieważ $f_4(24) = 17.28 < f_3(24) = 20$ wybieramy zadanie J_4. Typ zadania J_4 nie jest zgodny z $H[4]$. Przerywamy iterację i generujemy następny szablon.

i	H	T	$NS(G_H)$	$F_{\min}$	F_H	r	σ_H
1	$H(0,1,1,0)$	24	$(3,4)$	14	$-\infty$	4	$(\circ)$

Tabela 4.21. Podsumowanie przebiegu szóstej iteracji Algorytmu 4.4

Dla danej instancji problemu, Algorytm 4.4 wygenerował optymalne uszeregowanie $\sigma^\star = (2,3,1,4)$ z wartością $f_{\max} = 14$.



Rysunek 4.8. Optymalne uszeregowanie w Przykładzie 4.5

Rozdział 5

Rodzina problemów postaci

$$1|p_{j,r}(t) \in \left\{p^1_{j,r}(t), n_1; \ldots; p^k_{j,r}(t), n_k\right\}, k\text{–}part|f_{\max}$$

W tym rozdziale zaprezentowana zostanie druga z rozważanych w rozprawie rodzin jednomaszynowych problemów szeregowania zadań zależnych z mieszanymi czasami wykonywania, k–dzielnymi ograniczeniami kolejnościowymi między zadaniami oraz kryterium maksymalnego kosztu $f_{\max}$. Podrozdział 5.1 zawiera sformułowanie omawianej rodziny problemów. W podrozdziale 5.2 zaprezentowany zostanie wielomianowy algorytm rozwiązujący dowolny problem z tej jednomaszynowej rodziny problemów szeregowania zadań.

5.1. Sformułowanie problemów rodziny

Rozważmy rodzinę problemów zdefiniowaną w następujący sposób. Zbiór zadań $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ gotowych do wykonania począwszy od czasu $t_0 \geqslant 0$. Rozszerzymy notację z [12], zastępując parametr *prec* w polu β symbolami k–*part* i 2–*part* dla problemów szeregowania z ograniczeniami kolejnościowymi odpowiednio w postaci pełnego acyklicznego k-dzielnego grafu skierowanego (Definicja 2.2) i pełnego acyklicznego dwudzielnego grafu skierowanego (Definicja 2.4).

Zakładamy, że czasy wykonywania zadań $p^i_{j,r}(t)$ mogą przyjąć postać:

$$p^i_{j,r}(t) = p_j, \tag{1}$$
$$p^i_{j,r}(t) = p_j * (A + B * t), \tag{2}$$
$$p^i_{j,r}(t) = p_j + B * r^a, \tag{3}$$
$$p^i_{j,r}(t) = r^a * (A + B * t). \tag{4}$$

gdzie $1 \leqslant j \leqslant n$, $1 \leqslant i \leqslant k$, $p_j > 0$, $A \geqslant 0$, $B > 0, a \neq 0$ i $r \geqslant 1$. Przez p_j, A, B oraz a i r oznaczmy odpowiednio podstawowe czasy wykonywania oraz parametr odpowiedzialny za efekt uczenia się (jeżeli $a < 0$) lub starzenia się (jeżeli $a > 0$) i pozycję zadania w uszeregowaniu. Zbiór V_i zawiera jedynie indeksy zadań z czasami wykonywanie w postaci $p^i_{j,r}(t)$.

Z każdym zadaniem $J_j \in \mathcal{J}$ powiązana jest niemalejąca funkcja kosztu $f_j(t)$, mierząca koszt zakończenia zadania J_j w chwili t. Zakładamy, że wartość $f_j(t)$ może być obliczona w stałym czasie dla każdego $t \geqslant t_0$. Szukamy uszeregowania σ, o najmniejszej wartości $f_{\max}(\sigma) = \max\limits_{1 \leqslant j \leqslant n} \{f_j(C_j(\sigma))\}$ spośród wszystkich uszeregowań zgodnych z ograniczeniami kolejnościowymi, gdzie $C_j(\sigma)$ oznacza czas zakończenia zadania $J_j \in \mathcal{J}$ w uszeregowaniu σ.

5.2. Algorytm dla k–dzielnych ograniczeń kolejnościowych

W tym podrozdziale zaprezentujemy Algorytm 5.1, rozwiązujący problemy z rodziny jednomaszynowych problemów szeregowania zależnych zadań z mieszanymi czasami wykonywania, k-dzielnymi ograniczeniami kolejnościowymi między zadaniami i kryterium maksymalnego kosztu następujących postaci $1|p_{j,r}(t) \in \left\{p^1_{j,r}(t), n_1; \ldots; p^k_{j,r}(t), n_k\right\}, k\text{–part}|f_{\max}$. Zbiory wierzchołków V_1, $V_2, \ldots$, V_k zawierają odpowiednio jedynie numery zadań z czasami wykonywania postaci $p^1_{j,r}(t)$, $p^2_{j,r}(t)$, $\ldots$, $p^k_{j,r}(t)$. Algorytm wykorzystuje trzy pomocnicze funkcje: JobForm, ReduceT oraz CompT.

Funkcja JobForm dla danego numeru zadania j zwraca numer postaci czasu wykonywania zadania powiązanego z tym zadaniem.

```
1: function JobForm(j)
2:     case p_{j,r}(t) ma postać :
3:         p_j : JobForm ← 1
4:         p_j * (A + B * t) : JobForm ← 2
5:         p_j + B * r^a : JobForm ← 3
6:         r^a * (A + B * t) : JobForm ← 4
7:     end case
```

Funkcja CompT dla danego acyklicznego grafu skierowanego ograniczeń

kolejnościowych G i czasu rozpoczęcia t_0 oblicza w czasie $O(n)$ wartość czasu zakończenia ostatniego zadania T.

```
1: function CompT(k, G, t_0)
2:     T ← t_0
3:     r ← 1
4:     for i ← 1 to k do
5:         for all j in V_i do
6:             case JobForm(j) :
7:                 1 : T ← T + p_j
8:                 2 : T ← T * (1 + B * p_j) + A * p_j
9:                 3 : T ← T + (p_j + B * r^a)
10:                4 : T ← T * (1 + B * r^a) + A * r^a
11:            end case
12:            r ← r + 1
13:        end for
14:    end for
15:    return T
```

Funkcja ReduceT dla danego numeru zadania j, pozycji r zadania J_j w uszeregowaniu i czasu T redukuje wartość czasu T zgodnie z numerem postaci czasu wykonywania zadania zwróconego przez funkcję JobForm.

```
1: function ReduceT(j, r, T)
2:     case JobForm(j) :
3:         1 : T ← T − p_j
4:         2 : T ← (T − A * p_j) / (1 + B * p_j)
5:         3 : T ← T − (p_j + B * r^a)
6:         4 : T ← (T − A * r^a) / (1 + B * r^a)
7:     end case
8:     return T
```

Jeżeli ograniczenia kolejnościowe między zadaniami opisane są przy pomocy pełnego acyklicznego k-dzielnego grafu skierowanego, to możemy rozważać problemy, w których występują czasy wykonywania zadań opisane funkcjami z więcej niż dwóch rodzin. Zakładamy, że czasy wykonywania zadań $p^i_{j,r}(t)$ mogą przyjmować postać (1)–(4). Taka postać ograniczeń kolejnościowych dodatkowo gwarantuje, że wszystkie indeksy zadań bez następników są w ostatnim zbiorze V_k. Wiemy, że każdy ze zbiorów V_i, gdzie $1 \leqslant i \leqslant k$, zawiera jedynie zadania z czasami wykonywania $p^i_{j,r}(t)$ opisanymi funkcjami z tej samej rodziny, więc na podstawie Lematu 2.2 wiemy, że czas zakończenia ostatniego zadania w każdym ze zbiorów V_i nie zależy od uszeregowania.

Dlatego czas zakończenia ostatniego zadania T nie zależy od uszeregowania zadań. Na podstawie Lematu 2.1 wiemy, że optymalne uszeregowanie nie zawiera przestojów. Jeżeli acykliczny k-dzielny graf skierowany G nie będzie pełny, to nie możemy zagwarantować niezmienności czasu zakończenia ostatniego zadania T, ponieważ w zbiorze zadań bez następników mogą znajdować się zadania o różnych czasach wykonywania. Z tego powodu nie będziemy mogli zastosować Lematu 2.2.

Algorytm 5.1 bazuje na Algorytmie 4.1 i jego główna idea jest następująca: przy użyciu funkcji CompT obliczamy czas zakończenia ostatniego zadania T (linia 2). Następnie w każdej iteracji głównej pętli '**while**' (linie 5–11) wybieramy spośród zadań z ostatniego zbioru V_i zadanie o najmniejszym koszcie na danej pozycji (linia 6). Następnie umieszczamy to zadanie na początku uszeregowania $\sigma^\star$ (linia 7) i uaktualniamy czas T przy pomocy funkcji ReduceT (linia 8). Na koniec aktualizujemy zbiór V_i i liczbę pozostałych zadań r (linie 9–10). Jeżeli zbiór V_i jest pusty, to przechodzimy do poprzedzającego go zbioru V_{i-1} (linia 11). Powtarzamy te kroki do momentu, gdy wszystkie zadania zostaną uszeregowane.

Poniżej prezentujemy pseudokod Algorytmu 5.1.

Algorytm 5.1 dla $1|p_{j,r}(t) \in \left\{p^1_{j,r}(t), n_1; \ldots; p^k_{j,r}(t), n_k\right\}, k\text{–}part|f_{\max}$

```
1:  σ* ← (∘)
2:  T ← CompT(k, G, t_0)
3:  r ← n
4:  i ← k
5:  while r > 0 do
6:      Znajdź j ∈ V_i taki, że f_j(T) = min{f_v(T) : v ∈ V_i}
7:      σ* ← (j|σ*)
8:      T ← ReduceT(j, r, T)
9:      V_i ← V_i \ {j}
10:     if V_i = ∅ then i ← i − 1
11:     end if
12:     r ← r − 1
13: end while
14: return σ*
```

Jeżeli $k = 2$, to mamy do czynienia z rodziną problemów następującej postaci $1|p_{j,r}(t) \in \left\{p^1_{j,r}(t), n_1; p^2_{j,r}(t), n_2\right\}, 2\text{–}part|f_{\max}$, dla której n_1 zadań

z V_1 ma czasy wykonywania postaci $p^1_{j,r}(t)$ i n_2 zadań z V_2 ma czasy wykonywania postaci $p^2_{j,r}(t)$. Dodatkowo $V_1 \cap V_2 = \emptyset$ i $n_1 + n_2 = n$. Mieszane czasy wykonywania zadań mogą być postaci (1)–(4). Graf ograniczeń kolejnościowych $G = (V, A)$ jest pełnym acyklicznym dwudzielnym grafem skierowanym. Dla tej rodziny problemów prawdziwe jest następujące twierdzenie.

Twierdzenie 5.1. *Algorytm 5.1 dla dowolnego problemu z rodziny problemów* $1|p_{j,r}(t) \in \left\{p^1_{j,r}(t), n_1; p^2_{j,r}(t), n_2\right\}, 2\text{–}part|f_{\max}$ *z czasami wykonywania zadań postaci* $p^1_{j,r}(t)$, $p^2_{j,r}(t) \in \{p_j,\ p_j{*}(A + B * t),\ p_j{+}B{*}r^a,\ r^a{*}(A + B * t)\}$ *znajduje optymalne uszeregowanie w czasie* $O(n^2)$.

Dowód. Zauważmy, że Algorytm 5.1 generuje jedynie dopuszczalne uszeregowania, które nie naruszają ograniczeń kolejnościowych między zadaniami, ponieważ w (liniach 5–11) rozważamy jedynie zadania bez następników.

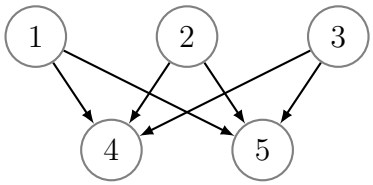
Każdy ze zbiorów V_1 i V_2 zawiera indeksy zadań z czasami wykonywania opisanymi funkcjami z tej samej rodziny. Spełnione są założenia Lematu 2.2 i czas zakończenia ostatniego zadania w każdym z tych zbiorów nie zależy od uszeregowania. Dodatkowo na podstawie Lematu 2.1 wiemy, że optymalne uszeregowanie nie zawiera przestojów.

Uszeregowanie $\sigma^\star$ wygenerowane przez Algorytm 5.1 ma minimalny koszt, ponieważ w każdym kroku algorytmu wybieramy zadanie, które ma minimalny koszt na danej pozycji i umieszczamy je na początku uszeregowania. Powtarzamy powyższą procedurę dla wszystkich pozostałych zadań. Po uszeregowaniu wszystkich zadań uzyskujemy optymalne uszeregowanie.

Złożoność obliczeniową Algorytmu 5.1 wyznaczamy w następujący sposób. Linia 2 wykonuje się w czasie $O(n)$. Linie 5–11 potrzebują czasu $O(n^2)$, ponieważ w każdej iteracji głównej pętli wybieramy spośród zadań ze zbioru V_i to, które ma najmniejszy koszt na danej pozycji. To może być zrobione w $|V_i| - 1$ porównaniach i musi być powtórzone $n_i - 1$ razy dla V_i. Łączna liczba porównań jest dana formułą $\frac{1}{2}(n_1^2 - n_1 + n_2^2 - n_2) = \frac{1}{2}(n^2 - n) - n_1 n_2$. Ponieważ, $0 \leqslant n_1 n_2 \leqslant \frac{1}{4}n^2$, stąd $\frac{1}{4}n^2 - \frac{1}{2}n \leqslant \frac{1}{2}(n^2 - n) - n_1 n_2 \leqslant \frac{1}{2}(n^2 - n)$. Złożoność obliczeniowa Algorytmu 5.1 wynosi zatem $O(n^2)$. □

Zilustrujemy działanie Algorytmu 5.1 na poniższym przykładzie:

Przykład 5.1. Danych jest $n = 5$ zadań z czasami wykonywania postaci $p_1 = p_2 = p_3 = t * r^2$, $p_4 = 4, p_5 = 5$ i czasem rozpoczęcia $t_0 = 1$. Funkcje kosztów są następującej postaci $f_1 = C_1 + 1$, $f_2 = C_2 + 15$, $f_3 = 2 \times C_3 + 2$, $f_4 = 0.5 * C_4 + 2$, $f_5 = C_5 + 10$. Ograniczenia kolejnościowe między zadaniami przedstawiono na Rysunku 5.1.



Rysunek 5.1. Ograniczenia kolejnościowe w Przykładzie 5.1

Iteracja 1 Obliczamy wartość czasu zakończenia ostatniego zadania $T = 109$, ustawiamy pozycję $r = 5$ oraz liczbę podziałów grafu $i = 2$. Dla zadań o numerach należących do zbioru $V_2 = \{4, 5\}$, obliczamy wartości funkcji kosztów. Ponieważ $f_4(109) = 56.5 < f_5(109) = 119$ zadanie J_4 umieszczamy w uszeregowaniu $\sigma^\star = (4)$. Aktualizujemy wartość $T = 109 - 4 = 105$ oraz zbiór V_2. Zbiór V_2 zawiera jeszcze zadanie, więc przechodzimy do kolejnej iteracji zmniejszając pozycję $r = 4$.

Iteracja 2 Dla zadania J_5, obliczamy wartości funkcji kosztu $f_5(105) = 115$ i umieszczamy je na przedostatniej pozycji w $\sigma^\star = (5, 4)$. Aktualizujemy wartość $T = 105 - 5 = 100$ oraz zbiór V_2. Zbiór V_2 jest już pusty, więc przechodzimy do zbioru V_1, zmniejszając pozycję $r = 3$.

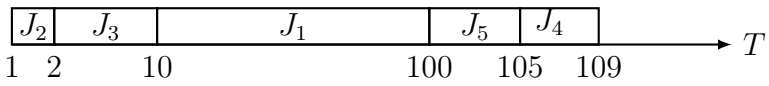
Iteracja 3 Dla zadań o numerach należących do zbioru $V_1 = \{1, 2, 3\}$, obliczamy wartości funkcji kosztów. Ponieważ $f_1(100) = 101 < f_2(100) = 115 < f_3(100) = 202$ zadanie J_1 umieszczamy w uszeregowaniu $\sigma^\star = (1, 5, 4)$. Aktualizujemy wartość $T = 100/10 = 10$ oraz zbiór V_1. Zbiór V_1 zawiera jeszcze zadania, więc przechodzimy do kolejnej iteracji zmniejszając pozycję $r = 2$.

Iteracja 4 Dla zadań o numerach należących do zbioru $V_1 = \{2, 3\}$, obliczamy wartości funkcji kosztów. Ponieważ $f_3(10) = 22 < f_2(10) = 25$ zadanie J_3 umieszczamy w uszeregowaniu $\sigma^\star = (3, 1, 5, 4)$. Aktualizujemy

wartość $T = 10/5 = 2$ oraz zbiór V_1. Zbiór V_1 zawiera jeszcze zadania, więc przechodzimy do kolejnej iteracji zmniejszając pozycję $r = 1$.

Iteracja 5 Dla zadania J_2, obliczamy wartości funkcji kosztu $f_2(2) = 17$ i umieszczamy je na pierwszej pozycji w $\sigma^\star = (2, 3, 1, 5, 4)$. Aktualizujemy wartość $T = 2/2 = 1$ oraz zbiór V_1. Zbiór V_1 jest już pusty oraz $r = 0$, więc kończymy działanie naszego algorytmu.

Dla danej instancji problemu, Algorytm 5.1 wygenerował optymalne uszeregowanie $\sigma^\star = (2, 3, 1, 5, 4)$ z maksymalnym kosztem równym $f_{\max} = 115$.



Rysunek 5.2. Optymalne uszeregowanie w Przykładzie 5.1

W przypadku dla $k \geqslant 2$, mamy do czynienia z rodziną problemów opisaną jako $1|p_{j,r}(t) \in \left\{p^1_{j,r}(t), n_1; \ldots; p^k_{j,r}(t), n_k\right\}, k\text{–part}|f_{\max}$. Niech G będzie pełnym acyklicznym k–dzielnym grafem skierowanym i każdy ze zbiorów V_1, $\ldots, V_k$ zawiera odpowiednio jedynie indeksy zadań z czasami wykonywania $p^1_{j,r}(t)$, $\ldots, p^k_{j,r}(t)$. Poprawność Algorytmu 5.1 dla $k \geqslant 2$ może być pokazana w ten sam sposób jak w dowodzie Twierdzenia 5.1 oraz w oparciu o indukcję matematyczną po k. Całkowita liczba porównań jest wyrażona wzorem $\frac{1}{2}\sum\limits_{i=1}^{k}(n_i^2 - n_i) = \frac{1}{2}(n^2 - n) - \sum\limits_{i=1}^{k-1}\sum\limits_{j=i+1}^{k}(n_i n_j)$. Złożoność obliczeniową oszacujemy w następujący sposób.

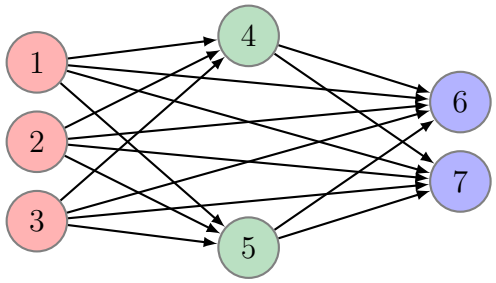
Mamy dane $n_1 + n_2 + \ldots + n_k = n$. Oszacujemy $\sum\limits_{i=1}^{k-1}\sum\limits_{j=i+1}^{k}(n_i n_j)$ w następujący sposób. $\sum\limits_{i=1}^{k-1}\sum\limits_{j=i+1}^{k}(n_i n_j) = n_1(n_2 + \ldots + n_k) + n_2(n_3 + \ldots + n_k) + \ldots + n_{k-1}n_k = n_1(n - n_1) + n_2(n - (n_1 + n_2)) + \ldots + n_{k-1}(n - (n_1 + \ldots + n_{k-1})) \leqslant nn_1 + nn_2 + \ldots nn_{k-1} = n(n - n_k) \leqslant n^2$. Otrzymaliśmy zatem następujące oszacowanie $0 \leqslant \sum\limits_{i=1}^{k-1}\sum\limits_{j=i+1}^{k}(n_i n_j) \leqslant n^2$, więc całkowita liczba porównań $\frac{1}{2}(n^2 - n) - \sum\limits_{i=1}^{k-1}\sum\limits_{j=i+1}^{k}(n_i n_j) \leqslant \frac{1}{2}(n^2 - n)$. Wynika stąd, że złożoność obliczeniowa naszego problemu wynosi $O(n^2)$.

Stosując powyższe rozważania otrzymujemy następujące twierdzenie:

Twierdzenie 5.2. *Algorytm 5.1 dla dowolnego problemu z rodziny problemów* $1|p_{j,r}(t) \in \left\{p^1_{j,r}(t), n_1; \ldots; p^k_{j,r}(t), n_k\right\}, k\text{–}part|f_{\max}$ *z czasami wykonywania zadań postaci* $p^1_{j,r}(t), \ldots, p^k_{j,r}(t) \in \{p_j,\ p_j * (A + B * t),\ p_j + B * r^a,\ r^a * (A + B * t)\}$ *znajduje optymalne uszeregowanie w czasie* $O(n^2)$.

Zilustrujemy działanie Algorytmu 5.1 na poniższym przykładzie:

Przykład 5.2. Danych jest $n = 7$ zadań z czasami wykonywania postaci $p_1 = p_2 = p_3 = t * r^2$, $p_4 = 2, p_5 = 1$ oraz $p_6 = 2 + r^2, p_7 = 3 + r^2$ i czasem rozpoczęcia $t_0 = 1$. Funkcje kosztów są następującej postaci $f_1 = 0.3 \times C_1 + 17$, $f_2 = C_2 + 13$, $f_3 = 0.5 \times C_3 + 13$, $f_4 = C_4 + 5$, $f_5 = 4 \times C_5^{0.5} + 20$, $f_6 = 2 \times C_6$, $f_7 = C_7 + 30$. Ograniczenia kolejnościowe między zadaniami przedstawiono na Rysunku 5.3.



Rysunek 5.3. Ograniczenia kolejnościowe między zadaniami w Przykładzie 5.2

Iteracja 1 Obliczamy wartość czasu zakończenia ostatniego zadania $T = 193$, ustawiamy pozycję $r = 7$ oraz liczba podziałów grafu $i = 3$. Dla zadań o numerach należących do zbioru $V_3 = \{6, 7\}$, obliczamy wartości funkcji kosztów. Ponieważ $f_7(193) = 223 < f_6(193) = 386$ zadanie J_7 umieszczamy w uszeregowaniu $\sigma^\star = (7)$. Aktualizujemy wartość $T = 193 - 49 - 3 = 141$ oraz zbiór V_3. Zbiór V_3 zawiera jeszcze zadania, więc przechodzimy do kolejnej iteracji zmniejszając pozycję $r = 6$.

Iteracja 2 Dla zadania J_6 obliczamy wartości funkcji kosztu $f_6(141) = 282$ i umieszczamy je na przedostatniej pozycji w $\sigma^\star = (6, 7)$. Aktualizujemy wartość $T = 141 - 2 - 36 = 103$ oraz zbiór V_3. Zbiór V_3 jest już pusty, więc przechodzimy do zbioru V_2, zmniejszając pozycję $r = 5$.

Iteracja 3 Dla zadań o numerach należących do zbioru $V_2 = \{4, 5\}$ obliczamy wartości funkcji kosztów. Ponieważ $f_5(103) = 60.6 < f_4(103) = 108$ zadanie J_5 umieszczamy w uszeregowaniu $\sigma^\star = (5, 6, 7)$. Aktualizujemy wartość $T = 103-1 = 102$ oraz zbiór V_2. Zbiór V_2 zawiera jeszcze zadania, więc przechodzimy do kolejnej iteracji zmniejszając pozycję $r = 4$.

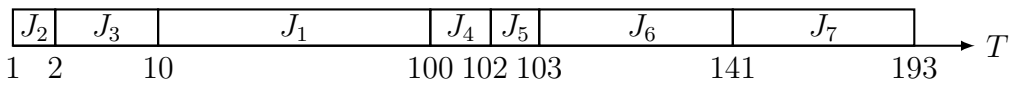
Iteracja 4 Dla zadania J_4 obliczamy wartości funkcji kosztu $f_4(102) = 107$ i umieszczamy je na pierwszej pozycji w $\sigma^\star = (4, 5, 6, 7)$. Aktualizujemy wartość $T = 102 - 2 = 100$ oraz zbiór V_2. Zbiór V_2 jest już pusty, więc przechodzimy do zbioru V_1, zmniejszając pozycję $r = 3$.

Iteracja 5 Dla zadań o numerach należących do zbioru $V_1 = \{1, 2, 3\}$ obliczamy wartości funkcji kosztów. Ponieważ $f_1(100) = 47 < f_3(100) = 63 < f_2(100) = 113$ zadanie J_1 umieszczamy w uszeregowaniu $\sigma^\star = (1, 4, 5, 6, 7)$. Aktualizujemy wartość $T = 100/10 = 10$ oraz zbiór V_1. Zbiór V_1 zawiera jeszcze zadania, więc przechodzimy do kolejnej iteracji zmniejszając pozycję $r = 2$.

Iteracja 6 Dla zadań o numerach należących do zbioru $V_1 = \{2, 3\}$ obliczamy wartości funkcji kosztów. Ponieważ $f_3(10) = 18 < f_2(10) = 23$ zadanie J_3 umieszczamy w uszeregowaniu $\sigma^\star = (3, 1, 4, 5, 6, 7)$. Aktualizujemy wartość $T = 10/5 = 2$ oraz zbiór V_1. Zbiór V_1 zawiera jeszcze zadania, więc przechodzimy do kolejnej iteracji zmniejszając pozycję $r = 1$.

Iteracja 7 Dla zadania J_2 obliczamy wartości funkcji kosztu $f_2(2) = 15$ i umieszczamy je na pierwszej pozycji w $\sigma^\star = (2, 3, 1, 4, 5, 6, 7)$. Aktualizujemy wartość $T = 2/2 = 1$ oraz zbiór V_1. Zbiór V_1 jest już pusty, oraz $r = 0$, więc kończymy działanie naszego algorytmu.

Dla danej instancji problemu Algorytm 5.1 wygenerował optymalne uszeregowanie $\sigma^\star = (2, 3, 1, 4, 5, 6, 7)$ z maksymalnym kosztem $f_{\max} = 282$.



Rysunek 5.4. Optymalne uszeregowanie w Przykładzie 5.2

Rozdział 6

Rodzina problemów postaci

$$1|p_{j,r}(t) \in \{a_j, n_1; b_j * t, n_2; a_j + b_j * t, n_3\}, prec|f_{\max}$$

W tym rozdziale zaprezentowana zostanie trzecia z rozważanych w rozprawie rodzin jednomaszynowych problemów szeregowania zadań ze zmiennymi mieszanymi czasami wykonywania, dowolnymi ograniczeniami kolejnościowymi między zadaniami oraz kryterium maksymalnego kosztu $f_{\max}$. Podrozdział 6.1 zawiera sformułowanie omawianej rodziny problemów. W podrozdziale 6.2 zaprezentowane zostaną dwie wersje algorytmu dokładnego. W podrozdziale 6.3 zaprezentowane zostanie kilkanaście konstrukcyjnych algorytmów heurystycznych wraz z hybrydowym algorytmem heurystycznym. W podrozdziale 6.4 przedstawione zostaną wyniki eksperymentu numerycznego badające jakość uszeregowań generowanych przez hybrydowy algorytm heurystyczny.

6.1. Sformułowanie problemów rodziny

W tym podrozdziale zaprezentowana zostania rodzina problemów szeregowania postaci $1|p_{j,r}(t) \in \{a_j, n_1; b_j * t, n_2; a_j + b_j * t, n_3\}, prec|f_{\max}$, gdzie n_1, n_2 i n_3 oznaczają odpowiednio liczbę zadań ze stałymi, proporcjonalnymi i liniowymi czasami wykonywania oraz $n_1+n_2+n_3 = n$. Dodatkowo zakładamy, że $a_j \geqslant 0, b_j \geqslant 0$, przy czym a_j i b_j nie zerują się jednocześnie. Pierwsze zadanie zaczyna się w chwili początkowej $t_0 \geqslant 0$.

Dla każdego zadania $J_j \in \mathcal{J}$ znana jest niemalejąca funkcja kosztu $f_j(t)$. Mierzy ona koszt zakończenia zadania J_j w chwili t i zakładamy, że jej wartość możemy oszacować w stałym czasie. Ograniczenia kolejnościowe mię-

dzy zadaniami ze zbioru $\mathcal{J}$ są opisane przez acykliczny graf skierowany $G = (V, A)$, gdzie V i A oznaczają odpowiednio zbiór wierzchołków oraz zbiór łuków. Naszym celem jest znalezienie uszeregowania σ o najmniejszej wartości $f_{\max}(\sigma) = \max\limits_{1 \leqslant j \leqslant n} \{f_j(C_j(\sigma))\}$ spośród wszystkich uszeregowań zgodnych z ograniczeniami kolejnościowymi, gdzie $C_j(\sigma)$ oznacza czas zakończenia zadania $J_j \in \mathcal{J}$ w uszeregowaniu σ.

6.2. Algorytmy dokładne

Omawiana rodzina problemów stanowi uogólnienie NP-trudnego problemu minimalizacji maksymalnej nieterminowości dla zbioru zadań z mieszanymi czasami wykonywania i niepustymi ograniczeniami kolejnościowymi zaprezentowanego w pracy [12]. Wynika to stąd, iż jeżeli funkcje kosztu będą następującej postaci $f_j = C_j - d_j$, to otrzymamy szczególny przypadek kryterium maksymalnego kosztu $f_{\max}$, a mianowicie $L_{\max}$, o którym była mowa w rozdziale 2. Ponadto, jeżeli zbiór zadań $\mathcal{J}$ będzie złożony z następującej liczby zadań ze stałymi, proporcjonalnymi i liniowymi czasami wykonywania odpowiednio $n_1 = 1$, $n_2 = n - 1$ i $n_3 = 0$, to wówczas nasza rodzina problemów redukuje się do problemu $1|p_{j,r}(t) \in \{a_j, 1; b_j * t, n - 1\}, prec|L_{\max}$ omawianego w pracy [12]. Ze względu na to, że problem ten jest NP-trudny, także nasza rodzina zawiera problemy NP-trudne, dla których nie istnieją wielomianowe algorytmy dokładne, o ile $P \neq NP$. Z tego powodu w kolejnych podrozdziałach konstruować będziemy algorytmy heurystyczne dla omawianych NP-trudnych problemów z tej rodziny. Jakość uszeregowań uzyskiwanych za pomocą algorytmów heurystycznych będzie badana za pomocą algorytmów dokładnych zaprezentowanych poniżej.

Algorytm 6.1 jest pierwszą wersją algorytmu dokładnego dla omawianej rodziny problemów opublikowaną w pracy [7]. Algorytm 6.2 jest ulepszoną wersją Algorytmu 6.1, w której zredukowano złożoność pamięciową. Algorytm 6.2 będzie używany do weryfikacji jakości uszeregowań generowanych przez hybrydowy algorytm heurystyczny, który przedstawiony zostanie w następnym podrozdziale. Na początku zaprezentowany zostanie pseudokod Al-

gorytmu 6.1 wraz z trzema pomocniczymi funkcjami: CalculateCT, GenerateAllTopSorts oraz AllTopSorts.

Funkcja CalculateCT dla danego uszeregowania H i czasu początkowego t_0 generuje wektor wszystkich czasów zakończenia zadań. Pseudokod tej funkcji wygląda następująco:

```
1: function CalculateCT(H[i], t_0)
2:     C[0] ← t_0
3:     for i ← 1 to n do
4:         C[i] ← a_i + (1 + b_i) * C[i − 1]
5:     end for
6:     return C
```

Funkcja GenerateAllTopSorts dla danego zbioru zadań $\mathcal{J}$ i grafu skierowanego G generuje listę wszystkich posortowań topologicznych H o rozmiarze SizeOf(H) używając algorytmu zaprezentowanego w [16]. Algorytm wykorzystuje następujące struktury danych: listę Vertices, tablicę predsCount oraz listę VeP, które zawierają odpowiednio wierzchołki, liczbę poprzedników dla każdego wierzchołka oraz wierzchołki bez poprzedników. Dla danego $v \in V(G)$, atrybut PredCount zawiera liczbę poprzedników v, podczas gdy metoda Add wstawia wierzchołek do listy VeP.

```
1: function GenerateAllTopSorts(G)
2:     for all v in V(G) do
3:         Vertices[v.Index] ← v
4:         predsCount[v.Index] ← v.PredCount
5:         if predsCount[v.Index] = 0 then
6:             VeP.Add(v.Index)
7:         end if
8:     end for
9:     AllTopSorts(0)
10:    return H
```

Funkcja GenerateAllTopSorts po stworzeniu wspomnianych struktur danych wywołuje funkcję AllTopSorts.

Funkcja AllTopSorts zwraca wszystkie posortowania topologiczne jako lista H. Działa ona na liście VeP zawierającej jedynie takie wierzchołki bez poprzedników, które nie zostały jeszcze przetworzone. Funkcja AllTopSorts może spowodować tymczasowe zmiany w VeP i predsCo-

UNT, ale na wyjściu są one przywracane do początkowych postaci. Funkcje ERASEALLRELATIONS(q) i RETRIEVEALLRELATIONS(q) odpowiednio usuwają i przywracają wszystkie łuki $(q, i) \in A(G)$ dla danego wierzchołka $q \in V(G)$. Tablica OUTPUT zawiera ciąg dotychczas przetworzonych wierzchołków i jeżeli ich liczba odpowiada liczbie wszystkich wierzchołków w grafie G, ciąg jest umieszczany na liście H.

```
1: function AllTopSorts(k)
2:     if (VeP.Count > 0) then
3:         base ← VeP[VeP.Count]
4:         do
5:             q ← VeP[VeP.Count]
6:             VeP ← VeP\ {q};
7:             EraseAllRelations(q)
8:             output[k] ← q;
9:             if k = Vertices.Count-1 then
10:                H.Add(output)
11:            end if
12:            AllTopSorts(k + 1)
13:            RetrieveAllRelations(q)
14:            VeP.Insert(1,q)
15:        while VeP[VeP.Count] ! = base
16:    end if
```

Idea Algorytmu 6.1 jest następująca. Używając funkcji GENERATEALLTOPSORTS tworzymy wszystkie posortowania topologiczne i zapisujemy je w tablicy H. Każde takie posortowanie odpowiada uszeregowaniu zapisanemu jako $H[i]$ w H. Dla każdego uszeregowania σ odpowiadającemu $H[i]$, obliczamy czasy zakończenia zadań poprzez funkcję CALCULATECT i przechowujemy je w tablicy C. Dla każdego zadania J_j obliczamy jego koszt w σ i wyznaczamy maksymalny koszt $F_{\max}$ dla aktualnego uszeregowania. Na koniec porównujemy wartość $F_{\max}$ z poprzednio zachowaną wartością $F_{\min}$ odpowiadającą najmniejszemu maksymalnemu kosztowi spośród wszystkich przeanalizowanych uszeregowań. Jeżeli $F_{\max} < F_{\min}$, to $F_{\min} \leftarrow F_{\max}$. W podobny sposób sprawdzamy wszystkie pozostałe posortowania w H.

Na wejściu Algorytmu 6.1 mamy dany acykliczny graf skierowany G zawierający ograniczenia kolejnościowe między zadaniami ze zbiór zadań $\mathcal{J}$. Na wyjściu algorytmu dostajemy optymalne uszeregowania $\sigma^\star$ z najmniejszym maksymalnym kosztem $f_{\max}$.

Algorytm 6.1 dla $1|p_{j,r}(t) \in \{a_j, 0; b_j * t, 0; a_j + b_j * t, n\}, prec|f_{\max}$

```
1: H ← GenerateAllTopSorts(G)
2: k ← SizeOf(H)
3: i ← 1
4: F_min ← ∞
5: while i ⩽ k do
6:     F_max ← −∞
7:     C ← CalculateCT(H[i], t_0)
8:     r ← n
9:     while r ⩾ 1 do
10:        j ← H[i][r]
11:        f ← f_j(C[r])
12:        if F_max < f then F_max ← f
13:        end if
14:        r ← r − 1
15:    end while
16:    if F_min > F_max and r = 0 then
17:        F_min ← F_max
18:        σ* ← H[i]
19:    end if
20:    i ← i + 1
21: end while
22: return σ*
```

Ze względu na dużą złożoność pamięciową Algorytmu 6.1, zaprojektowana została ulepszona wersja algorytmu, która nie przechowuje w pamięci wszystkich posortowań topologicznych. Zanim zaprezentowany zostanie pseudokod Algorytmu 6.2 przedstawione zostaną dwie pomocnicze funkcje CalculateFmax oraz MinTopSorts.

Funkcja CalculateFMax dla danego uszeregowania H i czasu początkowego t_0 zwraca maksymalny koszt. Pseudokod tej funkcji jest następujący:

```
1: function CalculateFMax(H, t_0)
2:     F_max ← −∞
3:     C[0] ← t_0
4:     for i ← 1 to n do
5:         j ← H[i]
6:         C[i] ← a_j + (1 + b_j) * C[i − 1]
7:         f ← f_j(C[i])
8:         if f > F_max then
9:             F_max ← f
10:        end if
11:    end for
12:    return F_max
```

Funkcja MinTopSorts przeszukuje wszystkie posortowania topologiczne w celu znalezienia uszeregowanie z minimalnym kosztem. Jest to modyfikacja funkcji AllTopSorts, gdzie zastąpiono zapamiętywanie znalezionego posortowania topologicznego (linia 10) poprzez jego analizę (linie 10–15). Funkcja zwraca optymalne uszeregowanie z minimalnym kosztem. Pseudokod tej funkcji wygląda następująco.

```
 1: function MinTopSorts(k, t_0, F_min, σ*)
 2:     if (VeP.Count > 0) then
 3:         base ← VeP[VeP.Count]
 4:         do
 5:             q ← VeP[VeP.Count]
 6:             VeP ← VeP\ {q};
 7:             EraseAllRelations(q)
 8:             output[k] ← q;
 9:             if k = Vertices.Count-1 then
10:                 H ← output
11:                 F_max ← CalculateFMax(H, t_0)
12:                 if F_min > F_max and r = 0 then
13:                     F_min ← F_max
14:                     σ* ← H
15:                 end if
16:             end if
17:             MinTopSorts(k + 1, t_0, F_min, σ*)
18:             RetrieveAllRelations(q)
19:             VeP.Insert(1,q)
20:         while VeP[VeP.Count] ! = base
21:     end if
```

Algorytm 6.1 generował wszystkie posortowania topologiczne i zapamiętywał je w tablicy H. Następnie analizowane było każde z tych posortowań. W Algorytmie 6.2 ograniczona została złożoność pamięciową tylko do jednej zmiennej H, która zawiera aktualnie rozpatrywane posortowanie topologiczne. Idea tego algorytmu dokładnego jest następująca. Po ustaleniu początkowych wartości zmiennych $F_{\min}$, $\sigma^\star$ oraz struktur danych predsCount oraz VeP, wywołana zostaje funkcja MinTopSorts. Tablica output zawiera ciąg dotychczas przetworzonych wierzchołków i jeżeli ich liczba odpowiada liczbie wszystkich wierzchołków w grafie G, ciąg jest umieszczany na liście H. Następnie dla uszeregowania H obliczamy maksymalny koszt $F_{\max}$ przy pomocy funkcji CalculateFMax. Na koniec porównujemy wartość $F_{\max}$ z poprzednio zapamiętaną wartością $F_{\min}$, która oznacza najmniejszy maksy-

malny koszt spośród wszystkich dotychczas przeanalizowanych uszeregowań i jeżeli $F_{\max}$ jest mniejsza, $F_{\min} \leftarrow F_{\max}$ oraz $\sigma^* \leftarrow H$. Po sprawdzeniu wszystkich uszeregowań na wyjściu Algorytmu 6.2 otrzymujemy optymalne uszeregowanie σ^* z najmniejszym maksymalnym kosztem.

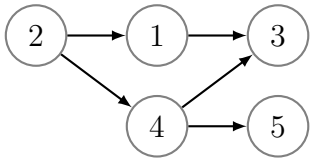
Na wejściu algorytmu mamy dane ograniczenia kolejnościowe w postaci acyklicznego grafu skierowanego $G(V, A)$, zbiór zadań $\mathcal{J}$ z czasami wykonywania postaci $p_{j,r}(t) \in \{a_j, n_1; b_j * t, n_2; a_j + b_j * t, n_3\}$ i funkcje kosztów f_j, gdzie $1 \leqslant j \leqslant n$. Pseudokod Algorytmu 6.2 wygląda następująco.

Algorytm 6.2 dla $1|p_{j,r}(t) \in \{a_j, n_1; b_j * t, n_2; a_j + b_j * t, n_3\}, prec|f_{\max}$

```
1: F_min ← ∞
2: σ* ← (∘)
3: for all v in V(G) do
4:     VERTICES[v.Index] ← v
5:     PREDSCOUNT[v.Index] ← v.PredCount
6:     if PREDSCOUNT[v.Index] = 0 then
7:         VEP.Add(v.Index)
8:     end if
9: end for
10: (σ*, F_min) ← MINTOPSORTS(0, t_0, F_min, σ*)
11: return (σ*, F_min)
```

Zilustrujemy działanie Algorytmu 6.2 na poniższym przykładzie.

Przykład 6.1. Danych jest $n = 5$ zadań z następującymi mieszanymi czasami wykonywania postaci $p_1 = 7 + 4 * t$, $p_2 = 5 + 2 * t$, $p_3 = 6 * t$, $p_4 = 3$, $p_5 = t$. Czas rozpoczęcia zadań $t_0 = 0$. Funkcje kosztów mają następującą postać $f_1 = 3 \times C_1 + 3$, $f_2 = 4 \times C_2 + 4$, $f_3 = 4 \times C_3 + 8$, $f_4 = C_4 + 2$, $f_5 = 4 \times C_5 + 6$. Ograniczenia kolejnościowe między zadaniami przedstawiono na Rysunku 6.1.



Rysunek 6.1. Ograniczenia kolejnościowe między zadaniami w Przykładzie 6.1

Dla ograniczeń kolejnościowych przedstawionych na Rysunku 6.1 istnieje 5 posortowań topologicznych zaprezentowanych w Tabeli 6.1.

Posortowania topologiczne
$\sigma_1 = (2, 4, 5, 1, 3)$
$\sigma_2 = (2, 4, 1, 3, 5)$
$\sigma_3 = (2, 4, 1, 5, 3)$
$\sigma_4 = (2, 1, 4, 5, 3)$
$\sigma_5 = (2, 1, 4, 3, 5)$

Tabela 6.1. Wszystkie posortowania topologiczne w Przykładzie 6.1

W pierwszej iteracji funkcji MinTopSorts tablica OUTPUT zawiera następujące posortowanie $H = [2, 4, 5, 1, 3]$. Funkcja CalculateFMax zwraca wartość $F_{\max} = 3284$. Zatem $F_{\min} \leftarrow 3284$ oraz $\sigma^\star = [2, 4, 5, 1, 3]$.

W drugiej iteracji posortowanie topologiczne wygenerowane przez funkcję MinTopSorts to $H = [2, 4, 1, 3, 5]$ z $F_{\max} = 3478$.

Następne posortowanie topologiczne wygenerowane przez funkcję MinTopSorts to $H = [2, 4, 1, 5, 3]$ z $F_{\max} = 3480$.

Następne posortowanie topologiczne wygenerowane przez funkcję MinTopSorts to $H = [2, 1, 4, 5, 3]$ z $F_{\max} = 2808$. Ponieważ $F_{\min} > F_{\max}$, więc $F_{\min} \leftarrow 2808$ oraz $\sigma^\star = [2, 1, 4, 5, 3]$.

Ostatnie posortowanie topologiczne wygenerowane przez funkcję MinTopSorts to $H = [2, 1, 4, 3, 5]$ z $F_{\max} = 2806$. Ponieważ $F_{\min} > F_{\max}$, więc $F_{\min} \leftarrow 2806$ oraz $\sigma^\star = [2, 1, 4, 3, 5]$.

Optymalne uszeregowanie wygenerowane przez Algorytm 6.2 ma postać $\sigma^\star = [2, 1, 4, 3, 5]$ z maksymalnym kosztem $f_{\max} = 2806$.

6.3. Algorytmy heurystyczne

W tym podrozdziale zaprezentowane zostaną dla naszej rodziny problemów dwa algorytmy heurystyczne oznaczane dalej odpowiednio przez AH1 oraz HAH. Algorytm AH1 jest pierwszą wersją algorytmu heurystycznego dla omawianej rodziny problemów opublikowaną w pracy [7]. Algorytm HAH jest nową wersją Algorytmu AH1, w której poprawiono jakość uzyskiwanych uszeregowań.

Poniżej prezentujemy pseudokody Algorytmu AH1 oraz trzech pomocniczych funkcji CalculateFMax, GenerateRates i FindMinimum.

Funkcja CalculateFMax dla danego uszeregowania σ i początkowego czasu t_0 oblicza maksymalny koszt $F_{\max}$.

```
1: function CalculateFMax(σ, t_0)
2:     C[0] ← t_0
3:     F_max ← −∞
4:     for all j in σ do
5:         C[j] ← a_j + (1 + b_j) * C[j − 1]
6:         f ← f_j(C[j])
7:         if F_max < f then F_max ← f
8:         end if
9:     return F_max
```

Funkcja GenerateRates dla każdego zadania ze zbioru zadań $\mathcal{J}$ zwraca parę dwóch zbiorów. Pierwszy zbiór zawiera wartości $\frac{b_j}{a_j}$ dla każdego zadania $J_j \in \mathcal{J}$. Wartości te są używane do wygenerowania dwóch różnych uszeregowań. Drugi zbiór zawiera wartości funkcji kosztu powiązanych z każdym zadaniem $J_j \in \mathcal{J}$ w chwili $t = Random(1, 100)$. Na ich podstawie tworzymy uszeregowania w porządku nierosnącym po tych wartościach.

```
1: function GenerateRates(J)
2:     for all j in J do
3:         t ← Random(1, 100)
4:         ptRates[j] ← b_j / a_j
5:         fRates[j] ← f_j(t)
6:     return (ptRates, fRates)
```

Funkcja FindMinimum używając funkcji CalculateFMax dla każdego z czterech uszeregowań $\sigma_1, \sigma_2, \sigma_3, \sigma_4$ oblicza maksymalny koszt i zwraca spośród nich uszeregowanie o najmniejszym maksymalnym koszcie.

```
1: function FindMinimum(σ_1, σ_2, σ_3, σ_4, t_0)
2:     for i ← 1 to 4 do
3:         F[i] ← CalculateFMax(σ_i, t_0)
4:     F_min ← F[1]
5:     for i ← 2 to 4 do
6:         if F[i] < F_min then
7:             F_min ← F[i]
8:             σ ← σ_i
9:         end if
10:    return (σ, F_min)
```

Na wejściu Algorytmu AH1 mamy dane acykliczny graf skierowany G zawierający ograniczenia kolejnościowe między zadaniami ze zbioru $\mathcal{J}$. Przez G_T, $V(G_T)$, $A(G_T)$ i $NS(G_T)$ będziemy oznaczać odpowiednio kopię grafu G, zbiór wierzchołków w G_T, zbiór łuków w G_T i zbiór zadań bez następników w G_T. Algorytm AH1 generuje na wyjściu optymalne uszeregowanie $\sigma^\star$.

Algorytm AH1 dla problemu $p_{j,r}(t) \in \{a_j, 0; b_j * t, 0; a_j + b_j * t, n\}$

for $i \leftarrow 1$ **to** 4 **do**
  $G_T \leftarrow G$
  while $V(G_T) \neq \emptyset$ **do**
    $NS(G_T) \leftarrow \left\{j \in V(G_T) : \deg^+(j) = 0\right\}$
    $min \leftarrow \infty$
    $max \leftarrow -\infty$
    GenerateRates($\mathcal{J}$)
    for all j **in** $NS(G_T)$ **do**
      if $i = 1$ **and** $ptRates[j] < min$ **then**
        $min \leftarrow ptRates[j]$
        $k \leftarrow j$
      else if $i = 2$ **and** $ptRates[j] > max$ **then**
        $max \leftarrow ptRates[j]$
        $k \leftarrow j$
      else if $i = 3$ **and** $fRates[j] > max$ **then**
        $max \leftarrow fRates[j]$
        $k \leftarrow j$
      end if
    end for
    if $i = 1$ **then** $\sigma_1 \leftarrow (k|\sigma_1)$
    else if $i = 2$ **then** $\sigma_2 \leftarrow (k|\sigma_2)$
    else if $i = 3$ **then** $\sigma_3 \leftarrow (k|\sigma_3)$
    else
      $x \leftarrow$ SizeOf$(NS(G_T))$
      $k \leftarrow x$ **div** $Random(1, x)$
      $\sigma_4 \leftarrow (NS(G_T, k)|\sigma_4)$
    end if
    $V(G_T) \leftarrow V(G_T) \setminus \{k\}$
    $A(G_T) \leftarrow A(G_T) \setminus \{(i, k) : i \in V(G_T)\}$
  end while
end for
$(\sigma^\star, F_{\min}) \leftarrow$ FindMinimum$(\sigma_1, \sigma_2, \sigma_3, \sigma_4, t_0)$
return $\sigma^\star, F_{\min}$

Idea Algorytmu AH1 jest następująca. Cztery iteracje zewnętrznej pętli **for** (linie 1–29) są odpowiedzialne za stworzenie czterech różnych uszeregowań $\sigma_1, \sigma_2, \sigma_3, \sigma_4$. Uszeregowania σ_1 i σ_2 są dopuszczalnymi uszeregowaniami konstruowanymi zgodnie z niemalejącym porządkiem i nierosnącym porządkiem w zbiorze po *ptRates* dostępnych w zbiorze $NS(G_T)$. Uszeregowanie σ_3 odpowiada nierosnącemu porządkowi wartości funkcji kosztów obliczonych przez funkcję GenerateRates. Ostatnie uszeregowanie σ_4 jest losowym dopuszczalnym uszeregowaniem. W każdej iteracji wewnętrznej pętli (linie 3–28) tworzony jest zbiór zadań bez następników (linia 4) i uszeregowania $\sigma_1, \sigma_2, \sigma_3, \sigma_4$ (linie 7–27). W każdej iteracji pętli **for** (linie 8–18) wybierane jest zadanie do σ_1 (linie 9–11), σ_2 (linie 12–14) i σ_3 (linie 15–17). Instrukcje warunkowe **if** w liniach 19–21 są odpowiedzialne za dodanie wybranego zadania do odpowiedniego uszeregowania (σ_1, σ_2 lub σ_3), podczas gdy w liniach 22–25 wybieramy losowe zadanie ze zbioru $NS(G_T)$ i umieszczamy je w uszeregowaniu σ_4. Następnie w liniach 26–27 aktualizujemy zbiory $V(G_T)$ i $A(G_T)$. Powtarzamy te instrukcje, aż do chwili gdy $V(G_T) = \emptyset$.

Jakość uszeregowań generowanych przez Algorytm AH1 okazała się niesatysfakcjonująca. Chcąc polepszyć jakość uszeregowań zaprojektowany został Hybrydowy Algorytm Heurystyczny HAH. Zanim sformułowany zostanie pseudokod tego algorytmu, zdefiniowane zostaną trzy pomocnicze funkcje: jForm, FindMinimum oraz UpdateDAG.

Funkcja jForm dla danego numeru zadania j zwraca numer postaci czasu wykonywania zadania powiązanego z tym zadaniem.

```
1: function jForm(j)
2:     case p_{j,r}(t) ma postać :
3:         a_j  :  jForm ← 1
4:         b_j * t  :  jForm ← 2
5:         a_j + b_j * t  :  jForm ← 3
6:     end case
```

Funkcja FindMinimum przy użyciu funkcji CalculateFMax wybiera spośród proponowanej listy uszeregowań Π suboptymalne uszeregowanie σ° wraz z jego maksymalnym kosztem.

```
1: function FindMinimum(Π, t_0)
2:     n ← Π.Count()
3:     for all σ in Π do
4:         F[i] ← CalculateFMax(σ, t_0)
5:     F_min ← F[1]
6:     for i ← 2 to n do
7:         if F[i] < F_min then
8:             F_min ← F[i]
9:             σ° ← Π[i]
10:        end if
11:    return (σ°, F_min)
```

Funkcja UpdateDAG dla danego wierzchołka k i grafu G usuwa ten wierzchołek z grafu, aktualizuje zbioru wierzchołków i łuków. Następnie zwraca zakutalizowany graf G.

```
1: function UpdateDAG(k, G)
2:     V(G) ← V(G) \ {k}
3:     A(G) ← A(G) \ {(i, k) : i ∈ V(G)}
4:     return G
```

Idea Hybrydowego Algorytmu Heurystycznego HAH jest następująca. Algorytm ten konstruuje przy pomocy osiemnastu heurystyk (Heurystyka H1 – H18), osiemnaście uszeregowań. Następnie przy użyciu funkcji FindMinimum wybiera to z najmniejszym maksymalnym kosztem. Poniżej prezentujemy listę algorytmów heurystycznych.

Heurystyka H1 konstruuje następujące uszeregowanie. Spośród zbioru zadań bez następników $NS(G)$ wybieramy zadanie J_j o najmniejszym b_j dla zadań z liniowymi lub proporcjonalnymi czasami wykonywania i umieszczamy je na początku dotychczas uszeregowanych zadań. Jeżeli w $NS(G)$ są tylko zadania stałe to o najmniejszym a_j. Następnie usuwamy to zadanie ze zbioru G i powtarzamy powyższe instrukcje dla pozostałych zadań. Po uszeregowaniu wszystkich zadań algorytm generuje uszeregowanie σ_1°.

Heurystyka H1 dla $1|p_{j,r}(t) \in \{a_j, n_1; b_j * t, n_2; a_j + b_j * t, n_3\}, prec|f_{\max}$

while $V(G) \neq \emptyset$ **do**
  $NS(G) \leftarrow \{j \in V(G) : \deg^+(j) = 0\}$
  $k = 0$
  Znajdź k taki, że $b_k = \min\{b_j : j \in NS(G) \wedge \text{JFORM}(j) \neq 1\}$
  if $k = 0$ **then**
    Znajdź $k \in NS(G)$ taki, że $a_k = \min\{a_j : j \in NS(G)\}$
  end if
  $\sigma^\circ \leftarrow (k|\sigma^\circ)$
  UPDATEDAG(k, G)
end while
return σ_1°

Heurystyka H2 konstruuje następujące uszeregowanie. Spośród zbioru zadań bez następników $NS(G)$ wybieramy zadanie J_j o największym b_j dla zadań z liniowymi lub proporcjonalnymi czasami wykonywania i umieszczamy je na początku dotychczas uszeregowanych zadań. Jeżeli w $NS(G)$ są tylko zadania stałe to o największym a_j. Następnie usuwamy to zadanie ze zbioru G i powtarzamy powyższe instrukcje dla pozostałych zadań.

Heurystyka H2 dla $1|p_{j,r}(t) \in \{a_j, n_1; b_j * t, n_2; a_j + b_j * t, n_3\}, prec|f_{\max}$

while $V(G) \neq \emptyset$ **do**
  $NS(G) \leftarrow \{j \in V(G) : \deg^+(j) = 0\}$
  $k = 0$
  Znajdź k taki, że $b_k = \max\{b_j : j \in NS(G) \wedge \text{JFORM}(j) \neq 1\}$
  if $k = 0$ **then**
    Znajdź k taki, że $a_k = \max\{a_j : j \in NS(G)\}$
  end if
  $\sigma^\circ \leftarrow (k|\sigma^\circ)$
  UPDATEDAG(k, G)
end while
return σ_2°

Heurystyka H3 konstruuje uszeregowanie w następujący sposób. Spośród zbioru zadań bez następników $NS(G)$ wybieramy zadanie J_j, które ma największą wartość funkcji kosztu w chwili $T = Random(1, 100)$ i umieszczamy je na początku dotychczas uszeregowanych zadań. Następnie usuwamy to zadanie ze zbioru G i powtarzamy powyższe instrukcje dla pozostałych zadań. Po uszeregowaniu wszystkich zadań algorytm generuje uszeregowanie σ_3°.

Heurystyka H3 dla $1|p_{j,r}(t) \in \{a_j, n_1; b_j * t, n_2; a_j + b_j * t, n_3\}, prec|f_{\max}$

while $V(G) \neq \emptyset$ **do**
  $NS(G) \leftarrow \{j \in V(G) : \deg^+(j) = 0\}$
  $T \leftarrow Random(1, 100)$
  Znajdź $k \in NS(G)$ taki, że $f_k(t) = \max\{f_j(t) : j \in NS(G)\}$
  $\sigma^\circ \leftarrow (k|\sigma^\circ)$
  UpdateDAG(k, G)
end while
return σ_3°

Heurystyka H4 konstruuje uszeregowanie w następujący sposób. Na początku konstruujemy macierz C czasów zakończenia zadań i macierz F funkcji kosztów oraz zbiór zadań bez poprzedników NP. Dla czasu t wyznaczamy w $C[i,j]$ różnicę $C_j - C_i$, gdzie C_j oraz C_i oznaczają czas zakończenia zadania odpowiednio J_j w uszeregowaniu (i,j) oraz J_i w uszeregowaniu (j,i). Następnie w $F[i,j]$ wyznaczamy różnicę $F_j(C_j) - F_i(C_i)$. Wybieramy zadanie J_j takie, że $f_{ij} = \min F[i,j]$ i umieszczamy je na początku dotychczas uszeregowanych zadań. Następnie usuwamy to zadanie z grafu G, wyznaczamy $t \leftarrow t + p_j(t)$ i powtarzamy powyższe instrukcje dla pozostałych zadań. Po uszeregowaniu wszystkich zadań algorytm generuje uszeregowanie σ_4°.

Heurystyka H4 dla $1|p_{j,r}(t) \in \{a_j, n_1; b_j * t, n_2; a_j + b_j * t, n_3\}, prec|f_{\max}$

$t \leftarrow t_0$
while $V(G) \neq \emptyset$ **do**
  $NP \leftarrow \{j \in V(G) : \deg^-(j) = 0\}$
  for all i in NP **do**
    for all j in NP **do**
      $\sigma \leftarrow (i, j)$
      $C(i,j) \leftarrow C_j(\sigma) - C_i(\sigma)$
      $F(i,j) \leftarrow f_j(C_j(\sigma)) - f_i(C_i(\sigma))$
    end for
  end for
  Znajdź $k \in NP$ taki, że $f_k = \min\{F_{i,k} : i \in NP\}$
  $T \leftarrow T + p_k(t)$
  $\sigma^\circ \leftarrow (k|\sigma^\circ)$
  UpdateDAG(k, G)
end while
return σ_4°

Heurystyka H5 konstruuje uszeregowanie w następujący sposób. Na początku konstruujemy macierz C czasów zakończenia zadań oraz zbiór zadań bez poprzedników NP. Dla czasu t wyznaczamy w $C[i,j]$ różnicę $C_j - C_i$, gdzie C_j oraz C_i oznaczają czas zakończenia zadania odpowiednio J_j w uszeregowaniu (i,j) oraz J_i w uszeregowaniu (j,i). Wybieramy zadanie J_j takie, że $C_{ij} = \min C[i,j]$, $t \leftarrow t + p_j(t)$ i umieszczamy je na początku dotychczas uszeregowanych zadań. Następnie usuwamy to zadanie ze zbioru G, wyznaczamy $t \leftarrow t + p_j(t)$ i powtarzamy powyższe instrukcje dla pozostałych zadań. Po uszeregowaniu wszystkich zadań algorytm generuje uszeregowanie σ_5°.

Heurystyka H5 dla $1|p_{j,r}(t) \in \{a_j, n_1; b_j * t, n_2; a_j + b_j * t, n_3\}, prec|f_{\max}$

```
1: t ← t_0
2: while V(G) ≠ ∅ do
3:     NP ← {j ∈ V(G) : deg⁻(j) = 0}
4:     for all i in NP do
5:         for all j in NP do
6:             σ ← (i, j)
7:             C(i, j) ← C_j(σ) − C_i(σ)
8:         end for
9:     end for
10:    Znajdź k ∈ NP taki, że C_k = min{C_{i,k} : i ∈ NP}
11:    T ← T + p_k(t)
12:    σ° ← (k|σ°)
13:    UpdateDAG(k, G)
14: end while
15: return σ_5^°
```

Heurystyka H6 konstruuje uszeregowanie w następujący sposób. Na początku konstruujemy macierz C czasów zakończenia zadań i macierz F funkcji kosztów oraz zbiór zadań bez poprzedników NP. Dla czasu t wyznaczamy w $C[i,j]$ różnicę $C_j - C_i$, gdzie C_j oraz C_i oznaczają czas zakończenia zadania odpowiednio J_j w uszeregowaniu (i,j) oraz J_i w uszeregowaniu (j,i). Następnie w $F[i,j]$ wyznaczamy różnicę $F_j(C_j) - F_i(C_i)$. Wybieramy zadanie J_j takie, że $f_{ij} = \max F[i,j]$ i umieszczamy je na początku dotychczas uszeregowanych zadań. Następnie usuwamy to zadanie ze zbioru G, wyznaczamy $t \leftarrow t + p_j(t)$ i powtarzamy powyższe instrukcje dla pozostałych zadań. Po uszeregowaniu wszystkich zadań algorytm generuje uszeregowanie σ_6°.

Heurystyka H6 dla $1|p_{j,r}(t) \in \{a_j, n_1; b_j * t, n_2; a_j + b_j * t, n_3\}, prec|f_{\max}$

1: $t \leftarrow t_0$
2: **while** $V(G) \neq \emptyset$ **do**
3: $NP \leftarrow \{j \in V(G) : \deg^-(j) = 0\}$
4: **for all** i in NP **do**
5: **for all** j in NP **do**
6: $\sigma \leftarrow (i, j)$
7: $C(i,j) \leftarrow C_j(\sigma) - C_i(\sigma)$
8: $F(i,j) \leftarrow f_j(C_j(\sigma)) - f_i(C_i(\sigma))$
9: **end for**
10: **end for**
11: Znajdź $k \in NP$ taki, że $f_k = \max\{F_{i,k} : i \in NP\}$
12: $T \leftarrow T + p_k(t)$
13: $\sigma^\circ \leftarrow (k|\sigma^\circ)$
14: UpdateDAG(k, G)
15: **end while**
16: **return** σ_6°

Heurystyka H7 ze zbioru NP wybiera zadanie J_j o najmniejszym b_j dla zadań z liniowymi lub proporcjonalnymi czasami wykonywania. Jeżeli w NP są tylko zadania stałe to o najmniejszym a_j. Umieszczamy zadanie J_j na końcu uszeregowanych zadań w σ_1. Usuwamy to zadanie z grafu G. Następnie konstruujemy zbiór zadań bez następników $NS(G)$ i wybieramy z niego zadanie J_j o największej wartości funkcji kosztu w chwili $T = Random(1, 100)$ i umieszczamy je na początku uszeregowanych zadań w σ_2. Usuwamy to zadanie i powtarzamy powyższe kroki dla pozostałych zadań. Po uszeregowaniu wszystkich zadań algorytm generuje uszeregowanie $\sigma_7^\circ = [\sigma_1, \sigma_2]$.

Heurystyka H7 dla $1|p_{j,r}(t) \in \{a_j, n_1; b_j * t, n_2; a_j + b_j * t, n_3\}, prec|f_{\max}$

1: **while** $V(G) \neq \emptyset$ **do**
2: $NP \leftarrow \{j \in V(G) : \deg^-(j) = 0\}$
3: $k = 0$
4: Znajdź k taki, że $b_k = \min\{b_j : j \in NP \wedge \text{JForm}(j) \neq 1\}$
5: **if** $k = 0$ **then** Znajdź k taki, że $a_k = \min\{a_j : j \in NP\}$
6: $\sigma_1 \leftarrow (\sigma_1|k)$
7: UpdateDAG(k, G)
8: $NS(G) \leftarrow \{j \in V(G) : \deg^+(j) = 0\}$
9: $T \leftarrow Random(1, 100)$
10: Znajdź k taki, że $f_k(t) = \max\{f_j(t) : j \in NS(G)\}$
11: $\sigma_2 \leftarrow (k|\sigma_2)$
12: UpdateDAG(k, G)
13: **end while**
14: $\sigma_7^\circ \leftarrow (\sigma_1, \sigma_2)$
15: **return** σ_7°

Heurystyka H8 konstruuje uszeregowanie w następujący sposób. Ze zbioru zadań bez poprzedników NP wybieramy zadanie J_j na dwa sposoby w zależności od aktualnej pozycji r. Na nieparzystych pozycjach wybieramy zadanie J_j o najmniejszym b_j dla zadań z liniowymi lub proporcjonalnymi czasami wykonywania, jeżeli w NP są tylko zadania stałe to o najmniejszym a_j. Na parzystych pozycjach wybieramy zadanie J_j o największej wartości funkcji kosztu w chwili $T = Random(1, 100)$. Umieszczamy zadanie J_j na końcu uszeregowanych zadań. Usuwamy to zadanie z grafu G i powtarzamy powyższe instrukcje dla pozostałych zadań. Po uszeregowaniu wszystkich zadań algorytm generuje uszeregowanie σ_8°.

Heurystyka H8 dla $1|p_{j,r}(t) \in \{a_j, n_1; b_j * t, n_2; a_j + b_j * t, n_3\}, prec|f_{\max}$

```
1: r ← 1
2: while V(G) ≠ ∅ do
3:     NP ← {j ∈ V(G) : deg⁻(j) = 0}
4:     if r mod 2 = 1 then
5:         Znajdź k taki, że b_k = min{b_j : j ∈ NP ∧ JFORM(j) ≠ 1}
6:         if k = 0 then Znajdź k taki, że a_k = min{a_j : j ∈ NP}
7:     else
8:         T ← Random(1, 100)
9:         Znajdź k taki, że f_k(t) = max{f_j(t) : j ∈ NS(G)}
10:    σ₈° ← (σ₈°|k)
11:    UPDATEDAG(k, G)
12: end while
13: return σ₈°
```

Heurystyka H9 konstruuje uszeregowanie w następujący sposób. Ze zbioru zadań bez poprzedników NP wybieramy zadanie J_j na dwa sposoby w zależności od aktualnej pozycji r. Na parzystych pozycjach wybieramy zadanie J_j o najmniejszym b_j dla zadań z liniowymi lub proporcjonalnymi czasami wykonywania, jeżeli w NP są tylko zadania stałe to o najmniejszym a_j. Na nieparzystych pozycjach wybieramy zadanie J_j o największej wartości funkcji kosztu w chwili $T = Random(1, 100)$. Umieszczamy zadanie J_j na końcu uszeregowanych zadań. Usuwamy to zadanie z grafu G i powtarzamy powyższe instrukcje dla pozostałych zadań. Po uszeregowaniu wszystkich zadań algorytm generuje uszeregowanie σ_9°.

Heurystyka H9 dla $1|p_{j,r}(t) \in \{a_j, n_1; b_j * t, n_2; a_j + b_j * t, n_3\}, prec|f_{\max}$

$r \leftarrow 1$
while $V(G) \neq \emptyset$ **do**
  $NP \leftarrow \{j \in V(G) : \deg^-(j) = 0\}$
  if r **mod** $2 = 0$ **then**
    Znajdź k taki, że $b_k = \min\{b_j : j \in NP \wedge \text{JFORM}(j) \neq 1\}$
    if $k = 0$ **then** Znajdź k taki, że $a_k = \min\{a_j : j \in NP\}$
  else
    $T \leftarrow Random(1, 100)$
    Znajdź k taki, że $f_k(t) = \max\{f_j(t) : j \in NS(G)\}$
  $\sigma_9^\circ \leftarrow (\sigma_9^\circ | k)$
  UPDATEDAG(k, G)
end while
return σ_9°

Heurystyka H10 konstruuje uszeregowanie w następujący sposób. Ze zbioru zadań bez poprzedników NP wybieramy J_j o największej wartości funkcji kosztu w chwili $T = Random(1, 100)$. Umieszczamy zadanie J_j na końcu uszeregowanych zadań w σ_1. Usuwamy to zadanie z grafu G. Następnie konstruujemy zbiór zadań bez następników $NS(G)$ i wybieramy z niego zadanie J_j o najmniejszym b_j dla zadań z liniowymi lub proporcjonalnymi czasami wykonywania. Jeżeli w $NS(G)$ są tylko zadania stałe, to wybieramy zadanie z najmniejszym a_j. Umieszczamy to zadanie na początku uszeregowanych zadań w σ_2. Usuwamy to zadanie i powtarzamy powyższe instrukcje dla pozostałych zadań. Po uszeregowaniu wszystkich zadań algorytm generuje uszeregowanie $\sigma_{10}^\circ = [\sigma_1, \sigma_2]$.

Heurystyka H10 dla $1|p_{j,r}(t) \in \{a_j, n_1; b_j * t, n_2; a_j + b_j * t, n_3\}, prec|f_{\max}$

while $V(G) \neq \emptyset$ **do**
  $NP \leftarrow \{j \in V(G) : \deg^-(j) = 0\}$
  $T \leftarrow Random(1, 100)$
  Znajdź k taki, że $f_k(t) = \max\{f_j(t) : j \in NP\}$
  $\sigma_1 \leftarrow (\sigma_1 | k)$
  UPDATEDAG(k, G)
  $NS(G) \leftarrow \{j \in V(G) : \deg^+(j) = 0\}$
  $k = 0$
  Znajdź k taki, że $b_k = \min\{b_j : j \in NS(G) \wedge \text{JFORM}(j) \neq 1\}$
  if $k = 0$ **then** Znajdź k taki, że $a_k = \min\{a_j : j \in NS(G)\}$
  $\sigma_2 \leftarrow (k | \sigma_2)$
  UPDATEDAG(k, G)
$\sigma_{10}^\circ \leftarrow (\sigma_1, \sigma_2)$
return σ_{10}°

Heurystyka H11 konstruuje uszeregowanie w następujący sposób. Ze zbioru NP wybieramy zadanie J_j o najmniejszej wartości funkcji kosztu w chwili $T = Random(1, 100)$. Umieszczamy to zadanie na końcu uszeregowanych zadań w σ_1. Usuwamy to zadanie z grafu G, konstruujemy zbiór zadań bez następników $NS(G)$ i wybieramy z niego zadanie o najmniejszym b_j dla zadań z liniowymi lub proporcjonalnymi czasami wykonywania. Jeżeli w $NS(G)$ są tylko zadania ze stałymi czasami wykonywania, to wybieramy to z najmniejszym a_j. Umieszczamy to zadanie na początku uszeregowanych zadań w σ_2. Usuwamy je z grafu i powtarzamy powyższe kroki dla pozostałych zadań. Po uszeregowaniu wszystkich zadań algorytm generuje uszeregowanie $\sigma_{11}^{\circ} = [\sigma_1, \sigma_2]$.

Heurystyka H11 dla $1|p_{j,r}(t) \in \{a_j, n_1; b_j * t, n_2; a_j + b_j * t, n_3\}, prec|f_{\max}$

while $V(G) \neq \emptyset$ **do**
  $NP \leftarrow \{j \in V(G) : \deg^-(j) = 0\}$
  $T \leftarrow Random(1, 100)$
  Znajdź k taki, że $f_k(t) = \min\{f_j(t) : j \in NP\}$
  $\sigma_1 \leftarrow (\sigma_1|k)$
  UPDATEDAG(k, G)
  $NS(G) \leftarrow \{j \in V(G) : \deg^+(j) = 0\}$
  $k = 0$
  Znajdź k taki, że $b_k = \max\{b_j : j \in NS(G) \wedge$ JFORM$(j) \neq 1\}$
  if $k = 0$ **then** Znajdź k taki, że $a_k = \max\{a_j : j \in NS(G)\}$
  $\sigma_2 \leftarrow (k|\sigma_2)$
  UPDATEDAG(k, G)
$\sigma_{11}^{\circ} \leftarrow (\sigma_1, \sigma_2)$
return σ_{11}°

Heurystyka H12 wybiera ze zbioru NP zadanie J_j o najmniejszej wartości b_j/a_j dla zadań z liniowymi czasami wykonywania. Jeżeli w NP nie ma zadań z liniowymi czasami wykonywania to szukamy zadania z najmniejszą wartością b_j spośród zadań z proporcjonalnymi czasami wykonywania. Jeżeli w NP są tylko zadanie ze stałymi czasami wykonywania, to wybieramy zadanie z największą wartością funkcji kosztu w chwili $T = Random(1, 100)$. Umieszczamy wybrane zadanie na początku uszeregowanych zadań. Usuwamy to zadanie i powtarzamy powyższe instrukcje dla pozostałych zadań. Po uszeregowaniu wszystkich zadań algorytm generuje uszeregowanie σ_{12}°.

Heurystyka H12 dla $1|p_{j,r}(t) \in \{a_j, n_1; b_j * t, n_2; a_j + b_j * t, n_3\}, prec|f_{\max}$

while $V(G) \neq \emptyset$ **do**
  $NP \leftarrow \{j \in V(G) : \deg^-(j) = 0\}$
  $T \leftarrow Random(1, 100)$
  $k = 0$
  Znajdź k taki, że $ba_k(t) = \min\{b_j/a_j : j \in NP \wedge \text{JFORM}(j) = 3\}$
  if $k = 0$ **then**
    Znajdź k taki, że $b_k = \min\{b_j : j \in NP \wedge \text{JFORM}(j) = 2\}$
  end if
  if $k = 0$ **then**
    Znajdź k taki, że $f_k = \max\{f_j(t) : j \in NP \wedge \text{JFORM}(j) = 2\}$
  end if
  $\sigma^\circ_{12} \leftarrow (\sigma^\circ_{12}|k)$
  UPDATEDAG(k, G)
end while
return σ°_{12}

Heurystyka H13 wyznacza czas $T = Random(1, 100)$ i ze zbioru NP wybiera zadanie J_j o najmniejszej wartości funkcji kosztu w chwili T dla zadań z liniowymi czasami wykonywania. Jeżeli w NP nie ma zadań z liniowymi czasami wykonywania, to szukamy zadania z najmniejszą wartością funkcji kosztu w chwili T dla zadań z proporcjonalnymi czasami wykonywania. Jeżeli w NP są tylko zadania ze stałymi czasami wykonywania, to wybieramy zadanie z największą wartością funkcji kosztu w chwili T. Umieszczamy wybrane zadanie na początku uszeregowanych zadań. Usuwamy to zadanie i powtarzamy powyższe instrukcje dla pozostałych zadań. Po uszeregowaniu wszystkich zadań algorytm generuje uszeregowanie σ°_{13}.

Heurystyka H13 dla $1|p_{j,r}(t) \in \{a_j, n_1; b_j * t, n_2; a_j + b_j * t, n_3\}, prec|f_{\max}$

$t \leftarrow t_0$
while $V(G) \neq \emptyset$ **do**
  $NP \leftarrow \{j \in V(G) : \deg^-(j) = 0\}$
  $k = 0$
  Znajdź k taki, że $f_k(t) = \min\{f_j(t) : j \in NP \wedge \text{JFORM}(j) = 3\}$
  if $k = 0$ **then**
    Znajdź k taki, że $f_k(t) = \min\{f_j(t) : j \in NP \wedge \text{JFORM}(j) = 2\}$
  end if
  if $k = 0$ **then**
    Znajdź k taki, że $f_k(t) = \max\{f_j(t) : j \in NP \wedge \text{JFORM}(j) = 1\}$
  end if
  $T \leftarrow T + p_j(t)$
  $\sigma^\circ_{13} \leftarrow (\sigma^\circ_{13}|k)$
  UPDATEDAG(k, G)
end while
return σ°_{13}

Heurystyka H14 działa w następujący sposób. Ze zbioru zadań bez poprzedników NP wybieramy zadanie o najmniejszej wartości funkcji kosztu w chwili $T = Random(1, 100)$ dla zadań z proporcjonalnymi czasami wykonywania, a jeżeli ich brak, to dla zadań z liniowymi czasami wykonywania. Jeżeli w NP są tylko zadania ze stałymi czasami wykonywania, to wybieramy zadanie z największą wartością funkcji kosztu w chwili T. Umieszczamy wybrane zadanie na początku uszeregowanych zadań. Usuwamy to zadanie i powtarzamy powyższe instrukcje dla pozostałych zadań. Po uszeregowaniu wszystkich zadań algorytm generuje uszeregowanie σ_{14}°.

Heurystyka H14 dla $1|p_{j,r}(t) \in \{a_j, n_1; b_j * t, n_2; a_j + b_j * t, n_3\}, prec|f_{\max}$

1: $t \leftarrow t_0$
2: **while** $V(G) \neq \emptyset$ **do**
3: $\quad NP \leftarrow \{j \in V(G) : \deg^-(j) = 0\}$
4: $\quad k = 0$
5: $\quad$ Znajdź k taki, że $f_k(t) = \min\{f_j(t) : j \in NP \wedge \text{JFORM}(j) = 2\}$
6: $\quad$ **if** $k = 0$ **then** Znajdź k taki, że $f_k(t) = \min\{f_j(t) : j \in NP \wedge \text{JFORM}(j) = 3\}$
7: $\quad$ **if** $k = 0$ **then** Znajdź k taki, że $f_k(t) = \max\{f_j(t) : j \in NP \wedge \text{JFORM}(j) = 1\}$
8: $\quad T \leftarrow T + p_j(t)$
9: $\quad \sigma_{14}^{\circ} \leftarrow (\sigma_{14}^{\circ}|k)$
10: $\quad$ UPDATEDAG(k, G)
11: **end while**
12: **return** σ_{14}°

Heurystyka H15 składa się z następujących kroków. Wyznaczamy czas $T = Random(1, 100)$ i ze zbioru zadań bez poprzedników NP wybieramy J_j o największej wartości funkcji kosztu w chwili T dla zadań z proporcjonalnymi czasami wykonywania. Jeżeli w NP nie ma zadań z proporcjonalnymi czasami wykonywania, to szukamy zadania z największą wartością funkcji kosztu w chwili T dla zadań z liniowymi czasami wykonywania. Jeżeli w NP są tylko zadania ze stałymi czasami wykonywania, to wybieramy zadanie z największą wartością funkcji kosztu w chwili T. Umieszczamy wybrane zadanie na początku uszeregowanych zadań. Usuwamy to zadanie i powtarzamy powyższe instrukcje dla pozostałych zadań. Po uszeregowaniu wszystkich zadań algorytm generuje uszeregowanie σ_{15}°.

Heurystyka H15 dla $1|p_{j,r}(t) \in \{a_j, n_1; b_j * t, n_2; a_j + b_j * t, n_3\}, prec|f_{\max}$

1: $t \leftarrow t_0$
2: **while** $V(G) \neq \emptyset$ **do**
3: $NP \leftarrow \{j \in V(G) : \deg^-(j) = 0\}$
4: $k = 0$
5: Znajdź k taki, że $f_k(t) = \max\{f_j(t) : j \in NP \wedge \text{JFORM}(j) = 2\}$
6: **if** $k = 0$ **then** Znajdź k taki, że $f_k(t) = \max\{f_j(t) : j \in NP \wedge \text{JFORM}(j) = 3\}$
7: **if** $k = 0$ **then** Znajdź k taki, że $f_k(t) = \max\{f_j(t) : j \in NP \wedge \text{JFORM}(j) = 1\}$
8: $T \leftarrow T + p_j(t)$
9: $\sigma^\circ_{15} \leftarrow (\sigma^\circ_{15}|k)$
10: UPDATEDAG(k, G)
11: **end while**
12: **return** σ°_{15}

Heurystyka H16 konstruuje uszeregowanie w następujący sposób. Ze zbioru zadań bez poprzedników NP wybieramy J_j o najmniejszej wartości b_j spośród zadań z proporcjonalnymi czasami wykonywania. Jeżeli w zbiorze NP nie ma zadań z proporcjonalnymi czasami wykonywania, to szukamy zadania z największą wartością b_j/a_j dla zadań z liniowymi czasami wykonywania. Jeżeli w NP są tylko zadania ze stałymi czasami wykonywania, to wybieramy zadanie z najmniejszą wartością a_j. Umieszczamy wybrane zadanie na początku uszeregowanych zadań. Usuwamy to zadanie i powtarzamy powyższe instrukcje dla pozostałych zadań. Po uszeregowaniu wszystkich zadań algorytm generuje uszeregowanie σ°_{16}.

Heurystyka H16 dla $1|p_{j,r}(t) \in \{a_j, n_1; b_j * t, n_2; a_j + b_j * t, n_3\}, prec|f_{\max}$

1: $t \leftarrow t_0$
2: **while** $V(G) \neq \emptyset$ **do**
3: $NP \leftarrow \{j \in V(G) : \deg^-(j) = 0\}$
4: $k = 0$
5: Znajdź k taki, że $b_k = \min\{b_j : j \in NP \wedge \text{JFORM}(j) = 2\}$
6: **if** $k = 0$ **then** Znajdź k taki, że $ba_k = \max\{b_j/a_j : j \in NP \wedge \text{JFORM}(j) = 3\}$
7: **if** $k = 0$ **then** Znajdź k taki, że $a_k = \min\{a_j : j \in NP \wedge \text{JFORM}(j) = 1\}$
8: $T \leftarrow T + p_j(t)$
9: $\sigma^\circ_{16} \leftarrow (\sigma^\circ_{16}|k)$
10: UPDATEDAG(k, G)
11: **end while**
12: **return** σ°_{16}

Heurystyka H17 konstruuje uszeregowanie w następujący sposób. Ze zbioru zadań bez poprzedników NP wybieramy J_j o największej wartości b_j spośród zadań z proporcjonalnymi czasami wykonywania. Jeżeli w NP nie

ma zadań z proporcjonalnymi czasami wykonywania, to szukamy zadania z największą wartością b_j/a_j dla zadań z liniowymi czasami wykonywania. Jeżeli w NP są tylko zadania ze stałymi czasami wykonywania, to wybieramy zadanie z największą wartością a_j. Umieszczamy wybrane zadanie na początku uszeregowanych zadań. Usuwamy to zadanie i powtarzamy powyższe instrukcje dla pozostałych zadań. Po uszeregowaniu wszystkich zadań algorytm generuje uszeregowanie σ_{17}°.

Heurystyka H17 dla $1|p_{j,r}(t) \in \{a_j, n_1; b_j * t, n_2; a_j + b_j * t, n_3\}, prec|f_{\max}$

$t \leftarrow t_0$
while $V(G) \neq \emptyset$ **do**
  $NP \leftarrow \{j \in V(G) : \deg^-(j) = 0\}$
  $k = 0$
  Znajdź k taki, że $b_k = \max\{b_j : j \in NP \wedge \text{JFORM}(j) = 2\}$
  if $k = 0$ **then**
    Znajdź k taki, że $ba_k = \max\{b_j/a_j : j \in NP \wedge \text{JFORM}(j) = 3\}$
  end if
  if $k = 0$ **then**
    Znajdź k taki, że $a_k = \max\{a_j : j \in NP \wedge \text{JFORM}(j) = 1\}$
  end if
  $T \leftarrow T + p_j(t)$
  $\sigma_{17}^{\circ} \leftarrow (\sigma_{17}^{\circ}|k)$
  UPDATEDAG(k, G)
end while
return σ_{17}°

Heurystyka H18 konstruuje uszeregowanie w następujący sposób. Jeżeli liczba pozostałych zadań jest większa od 2, to w pierwszej kolejności z $NP(G)$ wybieramy zadanie o największej wartości funkcji kosztu w chwili t spośród zadań z proporcjonalnymi czasami wykonywania; jeżeli brak takich, to wybieramy zadanie spośród zadań z liniowymi czasami wykonywania; jeżeli brak obu wcześniejszych, to wybieramy zadanie spośród zadań ze stałymi czasami wykonywania. Jeżeli liczba pozostałych zadań jest mniejsza od 3, to w takiej samej kolejności wybieramy ze zbioru NP zadanie tym razem o najmniejszej wartości funkcji kosztu Umieszczamy zadanie J_j na końcu uszeregowanych zadań oraz $t = C_j$. Usuwamy zadanie z grafu G i powtarzamy powyższe instrukcje dla pozostałych zadań. Po uszeregowaniu wszystkich zadań algorytm generuje uszeregowanie σ_{18}°.

Heurystyka H18 dla $1|p_{j,r}(t) \in \{a_j, n_1; b_j * t, n_2; a_j + b_j * t, n_3\}, prec|f_{\max}$

```
1: t ← t_0
2: while V(G) ≠ ∅ do
3:     NP ← {j ∈ V(G) : deg⁻(j) = 0}
4:     if V(G).Count < 3 then
5:         k = 0
6:         Znajdź k taki, że f_k(t) = min{f_j(t) : j ∈ NP ∧ JFORM(j) = 2}
7:         if k = 0 then
8:             Znajdź k taki, że f_k(t) = min{f_j(t) : j ∈ NP ∧ JFORM(j) = 3}
9:         end if
10:        if k = 0 then
11:            Znajdź k taki, że f_k(t) = min{f_j(t) : j ∈ NP ∧ JFORM(j) = 1}
12:        end if
13:    else
14:        k = 0
15:        Znajdź k taki, że f_k(t) = max{f_j(t) : j ∈ NP ∧ JFORM(j) = 2}
16:        if k = 0 then
17:            Znajdź k taki, że f_k(t) = max{f_j(t) : j ∈ NP ∧ JFORM(j) = 3}
18:        end if
19:        if k = 0 then
20:            Znajdź k taki, że f_k(t) = max{f_j(t) : j ∈ NP ∧ JFORM(j) = 1}
21:        end if
22:    end if
23:    T ← T + p_j(t)
24:    σ°_18 ← (σ°_18|k)
25:    UPDATEDAG(k, G)
26: end while
27: return σ°_18
```

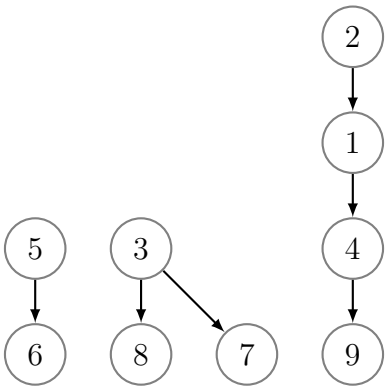
Pseudokod Hybrydowego Algorytmu HAH wygląda następująco. Na wejściu algorytmu mamy dany graf ograniczeń kolejnościowych $G = (V, A)$, zbiór zadań $\mathcal{J}$ z mieszanymi czasami wykonywania oraz funkcje kosztów. Na wyjściu Algorytmu HAH otrzymujemy suboptymalne uszeregowanie σ° z najmniejszym maksymalnym kosztem $F_{\min}$.

Algorytm HAH dla problemu $p_{j,r}(t) \in \{a_j, n_1; b_j * t, n_2; a_j + b_j * t, n_3\}$

```
1: for i ← 1 to 18 do
2:     σ°_i ← Algorytm Hi
3:     Π[i] ← σ°_i
4: end for
5: (σ°, F_min) ← FINDMINIMUM(Π, t_0)
6: return σ°, F_min
```

Zilustrujemy działanie Algorytmu HAH na poniższym przykładzie.

Przykład 6.2. Mamy dane $n = 9$ zadań, z losowymi czasami wykonywania i funkcjami kosztów jak w Tabeli 6.2. Ograniczenia kolejnościowe między zadaniami są przedstawione na Rysunku 6.2



Rysunek 6.2. Ograniczenia kolejnościowe w Przykładzie 6.2

Czasy wykonywania	Funkcje kosztów
$p_1(t) = 1$	$f_1(t) = t + 6$
$p_2(t) = 8$	$f_2(t) = 3t + 4$
$p_3(t) = 9t$	$f_3(t) = t + 3$
$p_4(t) = 6 + 3t$	$f_4(t) = t + 8$
$p_5(t) = 3t$	$f_5(t) = t + 5$
$p_6(t) = 8t$	$f_6(t) = 4t + 4$
$p_7(t) = 1 + 4t$	$f_7(t) = 3t + 9$
$p_8(t) = 7$	$f_8(t) = 4t + 9$
$p_9(t) = 8$	$f_9(t) = 3t + 7$

Tabela 6.2. Czasy wykonywania i funkcje kosztów w Przykładzie 6.2

Dla instancji naszego problemu, funkcja GenerateAllTopSorts wygenerowała 2 521 posortowań topologicznych. Każde z nich odpowiada dopuszczalnemu uszeregowaniu dla instancji z Przykładu 6.2 (patrz Tabela 6.2). Optymalne uszeregowanie uzyskane przy pomocy Algorytmu 6.1 ma postać (5,6,3,7,8,2,1,4,9) z maksymalnym kosztem równym 21 853. Najgorsze uszeregowanie znalezione przez Algorytm 6.1 ma postać (2,1,4,9,3,8,7,5,6) z maksymalnym kosztem równym 393 988. Poniżej zaprezentowano 18 uszeregowań uzyskanych za pomocą Algorytmu 6.2. Symbolem $\circ$ oznaczono suboptymalne uszeregowania.

Uszeregowanie	$f_{\max}$	Uszeregowanie	$f_{\max}$
$\sigma_1 = (2,1,4,9,3,8,5,6,7)$	295 392	$\sigma_{10} = (2,1,4,9,3,8,7,5,6)$	393 988
$\sigma_2 = (2,1,3,8,4,9,5,7,6)$	318 280	$\sigma_{11} = (3,5,2,1,4,9,8,6,7)$	29 307
$\sigma_3 = (3,5,2,1,4,9,7,6,8)$	37 873	$\sigma_{12}^{\circ} = (5,6,3,7,8,2,1,4,9)$	21 853
$\sigma_4 = (5,6,2,3,7,8,1,4,9)$	332 713	$\sigma_{13} = (3,7,5,6,8,2,1,4,9)$	22 273
$\sigma_5 = (2,1,4,5,6,3,8,7,9)$	25 213	$\sigma_{14} = (5,3,7,6,8,2,1,4,9)$	21 949
$\sigma_6 = (3,7,8,5,6,2,1,4,9)$	248 539	$\sigma_{15} = (3,7,5,6,8,2,1,4,9)$	22 273
$\sigma_7 = (3,5,2,1,4,9,7,6,8)$	37 873	$\sigma_{16}^{\circ} = (5,6,3,7,8,2,1,4,9)$	21 853
$\sigma_8 = (5,6,3,8,7,2,1,4,9)$	22 189	$\sigma_{17}^{\circ} = (3,5,6,7,2,8,1,4,9)$	21 853
$\sigma_9^{\circ} = (5,6,3,7,8,2,1,4,9)$	21 853	$\sigma_{18} = (2,3,7,8,5,6,1,4,9)$	197 917

Tabela 6.3. Wszystkie uszeregowania w Przykładzie 6.2

6.4. Eksperyment numeryczny

W tym podrozdziale zaprezentowane zostaną wyniki eksperymentu numerycznego, który przeprowadzony został w celu weryfikacji jakości uszeregowań konstruowanych przez Hybrydowy Algorytm Heurystyczny HAH.

W tym eksperymencie wygenerowano sześć zbiorów z n zadaniami, gdzie $n \in \{5,6,7,8,9,10\}$. Dla każdego n wygenerowano 40 razy losowy graf skierowany G opisujący ograniczenia kolejnościowe między zadaniami. Dla każdego takiego grafu, znaleziono najmniejszy maksymalny koszt i optymalne uszeregowanie przy użyciu Algorytmu 6.2 i suboptymalne uszeregowanie wraz z jego maksymalnym kosztem przy użyciu Hybrydowego Algorytmu Heurystycznego HAH. Łącznie wygenerowanych zostało 240 losowych instancji rozważanego problemu.

Wszystkie algorytmy zostały zaimplementowane w środowisku Microsoft Visual C# 2010 Express. Badania zostały przeprowadzone na komputerze PC o następujących parametrach:

- Płyta główna: ASUS P5K SE;
- Procesor: Intel(R) Pentium(R) Dual CPU E2180 @ 2.00GHz;
- Pamięć RAM: Patriot Memory 2GB DDR2-800;
- System operacyjny: Microsoft Windows XP Professional.

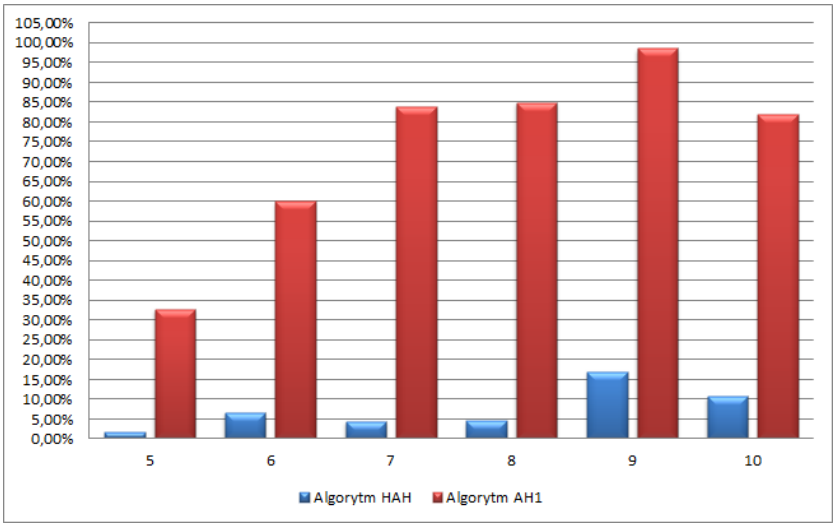
Wyniki naszego eksperymentu podsumowuje Tabela 6.2, w której przez '#sched', 'Alg. 6.2', 'Alg. HAH', 'Alg. AH1', oraz 'Najgorszy' oznaczamy odpowiednio liczbę wszystkich posortowań topologicznych, optymalny

(najmniejszy) maksymalny koszt uzyskany przez Algorytm 6.2, maksymalny koszt uszeregowania skonstruowanego przez Hybrydowy Algorytm HAH, maksymalny koszt uszeregowania skonstruowanego przez Algorytm Heurystyczny AH1 i najgorszy (największy) maksymalny koszt zapamiętany przez Algorytm 6.2. Każda z wartości w tabeli jest wielkością średnią z 40 wartości odpowiadającym losowo generowanym instancjom dla naszego problemu.

n	#sched	Alg. 6.2	Alg. HAH	Alg. AH1	Najgorszy
5	15	1 851	1 882	2 458	6 173
6	42	5 593	5 964	8 949	22 392
7	109	19 242	20 101	35 386	85 608
8	516	50 386	52 665	93 144	296 400
9	1 983	94 318	110 335	187 276	1 002 597
10	31 000	580 082	641 960	1 055 409	6 014 443

Tabela 6.4. Wyniki eksperymentu numerycznego

Jak widać z zaprezentowanej tabeli, jakość wyników uzyskiwanych za pomocą Hybrydowego Algorytmu HAH znacznie się polepszyła w stosunku do Algorytmu Heurystycznego AH1 zaprezentowanego w pracy [7].



Rysunek 6.3. Wykres błędów względnych algorytmów HAH i AH1

Rozdział 7

Podsumowanie

W niniejszej rozprawie przedstawiono wyniki dotyczące trzech rodzin jednomaszynowych problemów szeregowania zadań zależnych ze zmiennymi mieszanymi czasami wykonywania, niepustymi ograniczeniami kolejnościowymi oraz kryterium maksymalnego kosztu $f_{\max}$.

Dla rodziny jednomaszynowych problemów szeregowania zadań postaci $1|p_{j,r}(t) \in \{p_1, n-k; p_2, k\}, prec|f_{\max}$ otrzymano następujące wyniki zaprezentowane w opublikowanej pracy [6] oraz przedstawione na 25 międzynarodowej konferencji EURO (Wilno, lipiec 2012).

Skonstruowano wielomianowy Algorytm 4.1 znajdujący optymalne uszeregowanie dla problemu postaci $1|p_{j,r}(t) = p_j + b * r^a, prec|f_{\max}$, w którym czasy wykonywania zadań zależą jedynie od pozycji r zadania w uszeregowaniu. Udowodniono Twierdzenie 4.1 mówiące, że Algorytm 4.1 znajduje optymalne uszeregowania dla powyższego problemu w czasie $O(n^2)$.

Skonstruowano wielomianowy Algorytm 4.2 znajdujący optymalne uszeregowanie dla problemu postaci $1|p_{j,r}(t) = t * r^a, prec|f_{\max}$, w którym czasy wykonywania zadań zależą jednocześnie od czasu rozpoczęcia t oraz pozycji r. Udowodniono Twierdzenie 4.2 mówiące, że Algorytm 4.2 znajduje optymalne uszeregowania dla powyższego problemu w czasie $O(n^2)$.

Skonstruowano wielomianowy Algorytm 4.3 znajdujący optymalne uszeregowanie dla problemu $1|p_{j,r}(t) \in \{p_j + b * r^a, n-k; p_j, k\}, prec|f_{\max}$, w którym $n-k$ zadań posiada czasy wykonywania postaci $p_{j,r}(t) = p_j + b * r^a$ oraz k zadań ze stałymi czasami wykonywania postaci $p_{j,r}(t) = p_j$. Udowodniono

Twierdzenie 4.3 mówiące, że Algorytm 4.3 znajduje optymalne uszeregowania dla powyższego problemu w czasie $O\left(n^{k+2}\right)$.

Skonstruowano wielomianowy Algorytm 4.4 znajdujący optymalne uszeregowanie dla problemu $1|p_{j,r}(t) \in \{t * r^a, n-k; p_j * t, k\}, prec|f_{\max}$, w którym $n-k$ zadań posiada czasy wykonywania postaci $p_{j,r}(t) = t * r^a$ oraz k zadań z proporcjonalnymi czasami wykonywania $p_{j,r}(t) = p_j * t$. Udowodniono Twierdzenie 4.4 mówiące, że Algorytm 4.4 znajduje optymalne uszeregowania dla powyższego problemu w czasie $O\left(n^{k+2}\right)$.

Problem	Algorytm	Złożoność
$1\|p_{j,r}(t) = p_j + b * r^a, prec\|f_{\max}$	Algorytm 4.1	$O(n^2)$
$1\|p_{j,r}(t) = t * r^a, prec\|f_{\max}$	Algorytm 4.2	$O(n^2)$
$1\|p_{j,r}(t) \in \{p_j + b * r^a, n-k; p_j, k\}, prec\|f_{\max}$	Algorytm 4.3	$O\left(n^{k+2}\right)$
$1\|p_{j,r}(t) \in \{t * r^a, n-k; p_j * t, k\}, prec\|f_{\max}$	Algorytm 4.4	$O\left(n^{k+2}\right)$

Tabela 7.1. Podsumowanie wyników uzyskanych w rozdziale 4

Dla rodziny jednomaszynowych problemów szeregowania zadań postaci $1|p_{j,r}(t) \in \left\{p^1_{j,r}(t), n_1; \ldots; p^k_{j,r}(t), n_k\right\}, k\text{–}part|f_{\max}$ otrzymano następujące wyniki zaprezentowane w opublikowanej pracy [5] oraz przedstawione na XVIII Krajowej Konferencji Automatyzacji Procesów Dyskretnych (Zakopane, wrzesień 2012).

Skonstruowano wielomianowy Algorytm 5.1 znajdujący optymalne uszeregowanie dla dowolnego problemu z powyższej rodziny problemów. Udowodniono Twierdzenie 5.2 mówiące, że Algorytm 5.1 znajduje optymalne uszeregowania dla dowolnego problemu z powyższej rodziny w czasie $O(n^2)$, jeżeli czasy wykonywania zadań $p^1_{j,r}(t), \ldots, p^k_{j,r}(t) \in \{p_j,\ p_j * (A + B * t),\ p_j + B * r^a,\ r^a * (A + B * t)\}$.

Problem	Algorytm	Złożoność
$1\|p_{j,r}(t) \in \left\{p^1_{j,r}(t), n_1; \ldots; p^k_{j,r}(t), n_k\right\}, k\text{–part}\|f_{\max}$	Algorytm 5.1	$O(n^2)$

Tabela 7.2. Podsumowanie wyników uzyskanych w rozdziale 5

Dla rodziny jednomaszynowych problemów szeregowania zadań postaci $1|p_{j,r}(t) \in \{a_j, b_j * t, a_j + b_j * t\}, prec|f_{\max}$ otrzymano następujące wyni-

ki, przedstawione w referacie wygłoszonym na międzynarodowej konferencji FedCSIS (Wrocław, wrzesień 2012) oraz w pracy [7] przyjętej do publikacji w materiałach z tej konferencji.

Skonstruowano algorytm dokładny (Algorytm 6.2) oraz hybrydowy algorytm heurystyczny (Algorytm HAH).

Problem	Algorytm
$1\|p_{j,r}(t) \in \{a_j, n_1; b_j * t, n_2; a_j + b_j * t, n_3\}, prec\|f_{\max}$	Algorytm 6.2
$1\|p_{j,r}(t) \in \{a_j, n_1; b_j * t, n_2; a_j + b_j * t, n_3\}, prec\|f_{\max}$	Algorytm HAH

Tabela 7.3. Podsumowanie wyników uzyskanych w rozdziale 6

Dalsze badania mogą zmierzać w następujących kierunkach. Pierwszy z nich może polegać na znalezieniu innych postaci zmiennych czasów wykonywania zadań, które prowadzą do wielomianowo rozwiązywalnych problemów szeregowania zadań z mieszanymi czasami wykonywania. Drugi z nich może dotyczyć poszukiwania nowych postaci grafów ograniczeń kolejnościowych między zadaniami, które pozwolą na rozwiązanie problemów w czasie wielomianowym. Trzeci z interesujących kierunków badań może koncentrować się na identyfikacji nietrywialnych NP-trudnych problemów z omawianych rodzin problemów. Interesujące wydaje się również przeprowadzenie eksperymentów w oparciu o szybszy algorytm generowania posortowań topologicznych Varola i Rotema [20] i porównanie go z wynikami uzyskanymi za pomocą algorytmu Knutha [15].

Bibliografia

[1] M.J. Anzanello, F.S. Fogliatto, Learning curve models and applications: Literature review and research directions, International Journal of Industrial Ergonomics, 41 (2011), no. 5, 573–583.

[2] D. Biskup, A state-of-the-art review on scheduling with learning effects, European Journal of Operational Research, 188 (2008), no. 2, 315–329.

[3] P. Brucker, Scheduling Algorithms, 5th edition, Berlin - Heidelberg, Springer 2007.

[4] R.W. Conway, W.L. Maxwell, L.W. Miller, Theory of scheduling, Reading: Addison-Wesley, 1967.

[5] M. Dębczyński, Maximum cost scheduling of jobs with mixed processing times and k-partite precedence constraints, Optimization Letters, (2012), doi:10.1007/s11590-012-0582-5.

[6] M. Dębczyński, S. Gawiejnowicz, Scheduling jobs with mixed processing times, arbitrary precedence constraints and maximum cost criterion, Computers & Industrial Engineering, 64 (2013), no. 1, 273–279.

[7] M. Dębczyński, S. Gawiejnowicz, An exact algorithm and a heuristic for scheduling linearly deteriorating jobs with arbitrary precedence constraints and maximum cost criterion, Preprints of the Federated Conference on Computer Science and Information Systems, (2012), 423–427.

[8] A. Dolgui, V.S. Gordon, V.A. Strusevich, Single machine scheduling with precedence constraints and positionally dependent processing times, Computers and Operations Research, 39 (2012), no. 6, 1218–1224.

[9] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, San Francisco, Freeman 1979.

[10] S. Gawiejnowicz, Time-Dependent Scheduling, Berlin - Heidelberg, Springer 2008.

[11] S. Gawiejnowicz, T.C. Lai, M.H. Chiang, Scheduling linearly shortening jobs under precedence constraints, Applied Mathematical Modelling, 35 (2011), 2005–2015.

[12] S. Gawiejnowicz and B. M. T. Lin, Scheduling time-dependent jobs under mixed deterioration, Applied Mathematics and Computation, 216 (2010), no. 2, 438–447.

[13] V.S. Gordon, C.N. Potts, V.A. Strusevich, J.D. Whitehead, Single machine scheduling models with deterioration and learning: handling precedence constraints via priority generation, Journal of Scheduling, 11 (2008), no. 5, 357–370.

[14] R.L. Graham, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling: a survey, Annals of Discrete Mathematics, 5 (1979), 287–326.

[15] D.E. Knuth, The Art of Computer Programming, vol. 4, Fascicle 2, Addison-Wesley, 2005.

[16] D.E. Knuth, J.L. Szwarcfiter, A structured program to generate all topological sorting arrangements, Information Processing Letters, 2 (1974), no. 6, 153–157.

[17] E.L. Lawler, Optimal sequencing of a single machine subject to precedence constraints, Management Science, 19 (1973), no. 5, 544–546.

[18] E.J. Lodree, C.D. Geiger, X.C. Jiang, Taxonomy for integrating scheduling theory and human factors: Review and research opportunities, International Journal of Industrial Ergonomics, 39 (2009), no. 1, 39–51.

[19] M. Pinedo, Scheduling: Theory, Algorithms, and Systems, Englewood Cliffs: Prentice-Hall 1995.

[20] Y.L. Varol, D. Rotem, An algorithm to generate all topological sorting arrangements, The Computer Journal, 24 (1981), no. 1, 83–84.

[21] J.B. Wang, C.T. Ng, T.C.E. Cheng, Single-machine scheduling with deteriorating jobs under a series-parallel constraint, Computers & Operations Research, 35 (2008), no. 8, 2684–2693.

[22] J.B. Wang, J.J. Wang, P. Ji, Scheduling jobs with chain precedence con-

straints and deteriorating jobs, Journal of the Operational Research Society, 62 (2011), 1765–1770.

[23] J.B. Wang, J.J. Wang, Single-machine scheduling with precedence constraints and position-dependent processing times, Applied Mathematical Modelling, 37 (2013), 649–658.

Spis rysunków

2.1. Przykłady pełnych acyklicznych grafów skierowanych: a) dwudzielny b) k-dzielny 11
2.2. Przykładowe postacie ograniczeń kolejnościowych 12
2.3. Diagram Gantta optymalnego uszeregowania w Przykładzie 2.1 13
2.4. Diagram Gantta optymalnego uszeregowania w Przykładzie 2.2 14

3.1. Ograniczenia kolejnościowe między zadaniami w Przykładzie 3.1 . . . 21
3.2. Optymalne uszeregowanie w Przykładzie 3.1 22

4.1. Ograniczenia kolejnościowe w Przykładzie 4.1 28
4.2. Optymalne uszeregowanie w Przykładzie 4.1 29
4.3. Ograniczenia kolejnościowe w Przykładzie 4.2 30
4.4. Optymalne uszeregowanie w Przykładzie 4.2 31
4.5. Ograniczenia kolejnościowe w Przykładzie 4.4 38
4.6. Optymalne uszeregowanie w Przykładzie 4.4 47
4.7. Ograniczenia kolejnościowe w Przykładzie 4.5 49
4.8. Optymalne uszeregowanie w Przykładzie 4.5 53

5.1. Ograniczenia kolejnościowe w Przykładzie 5.1 59
5.2. Optymalne uszeregowanie w Przykładzie 5.1 60
5.3. Ograniczenia kolejnościowe między zadaniami w Przykładzie 5.2 . . . 61
5.4. Optymalne uszeregowanie w Przykładzie 5.2 62

6.1. Ograniczenia kolejnościowe między zadaniami w Przykładzie 6.1 . . . 69
6.2. Ograniczenia kolejnościowe w Przykładzie 6.2 87
6.3. Wykres błędów względnych algorytmów HAH i AH1 89

Spis tabel

3.1. Wyniki uzyskane w monografii [10, Rozdział 13] 23
3.2. Wyniki zaprezentowane w monografii [10] oraz w pracach [21] i [13] . 23

4.1. Podsumowanie przebiegu pierwszej iteracji Algorytmu 4.3 39
4.2. Podsumowanie przebiegu drugiej iteracji Algorytmu 4.3 40
4.3. Podsumowanie przebiegu trzeciej iteracji Algorytmu 4.3 42
4.4. Podsumowanie przebiegu czwartej iteracji Algorytmu 4.3 42
4.5. Podsumowanie przebiegu piątej iteracji Algorytmu 4.3 42
4.6. Podsumowanie przebiegu szóstej iteracji Algorytmu 4.3 43
4.7. Podsumowanie przebiegu siódmej iteracji Algorytmu 4.3 44
4.8. Podsumowanie przebiegu ósmej iteracji Algorytmu 4.3 45
4.9. Podsumowanie przebiegu dziewiątej iteracji Algorytmu 4.3 45
4.10. Podsumowanie przebiegu dziesiątej iteracji Algorytmu 4.3 45
4.11. Podsumowanie przebiegu jedenastej iteracji Algorytmu 4.3 46
4.12. Podsumowanie przebiegu dwunastej iteracji Algorytmu 4.3 46
4.13. Podsumowanie przebiegu trzynastej iteracji Algorytmu 4.3 46
4.14. Podsumowanie przebiegu czternastej iteracji Algorytmu 4.3 46
4.15. Podsumowanie przebiegu piętnastej iteracji Algorytmu 4.3 47
4.16. Podsumowanie przebiegu pierwszej iteracji Algorytmu 4.4 49
4.17. Podsumowanie przebiegu drugiej iteracji Algorytmu 4.4 50
4.18. Podsumowanie przebiegu trzeciej iteracji Algorytmu 4.4 51
4.19. Podsumowanie przebiegu czwartej iteracji Algorytmu 4.4 52
4.20. Podsumowanie przebiegu piątej iteracji Algorytmu 4.4 52
4.21. Podsumowanie przebiegu szóstej iteracji Algorytmu 4.4 53

6.1. Wszystkie posortowania topologiczne w Przykładzie 6.1 70
6.2. Czasy wykonywania i funkcje kosztów w Przykładzie 6.2 87
6.3. Wszystkie uszeregowania w Przykładzie 6.2 88
6.4. Wyniki eksperymentu numerycznego 89

7.1. Podsumowanie wyników uzyskanych w rozdziale 4 91
7.2. Podsumowanie wyników uzyskanych w rozdziale 5 91
7.3. Podsumowanie wyników uzyskanych w rozdziale 6 92