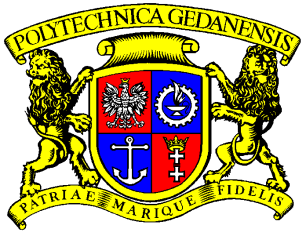
Szeregowanie zadań w modelu deterministycznym



dr hab. inż. Krzysztof Giaro
Politechnika Gdańska, Wydział ETI
pok. 207

Plan wykładu

1. Wstęp do deterministycznego szeregowania zadań
2. Metoda ścieżki krytycznej
3. Podstawowe problemy optymalizacji dyskretnej
4. Minimalizacja kryterium $C_{\max}$
5. Minimalizacja kryterium ΣC_i
6. Minimalizacja kryterium $L_{\max}$
7. Minimalizacja liczby spóźnionych zadań
8. Szeregowanie zadań na maszynach dedykowanych

Wstęp do deterministycznego szeregowania zadań

Dziedzina ta zajmuje się ***szeregowaniem*** (układaniem harmonogramów) ***zadań*** (programów, czynności, prac) na ***maszynach*** (procesorach, obrabiarkach, stanowiskach obsługi).

Szukamy harmonogramu wykonania dla danego zbioru zadań w określonych warunkach, tak by zminimalizować przyjęte ***kryterium oceny*** (koszt) uszeregowania.

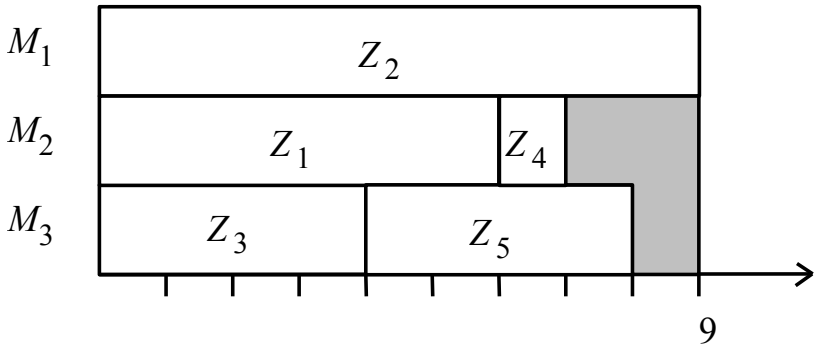
Model deterministyczny: parametry systemu i zadań są od początku znane.

Geneza i motywacje praktyczne:

- harmonogramowanie produkcji przemysłowej,
- planowanie projektów,
- organizacja pracy,
- plany zajęć szkolnych, spotkań i konferencji,
- przetwarzanie procesów w wielozadaniowych systemach operacyjnych,
- organizacja obliczeń rozproszonych.

Wstęp do deterministycznego szeregowania zadań

Przykład. Pięć zadań o czasach wykonania $p_1,\ldots,p_5=6,9,4,1,4$ należy uszeregować na trzech maszynach tak, by zakończyły się one możliwie jak najszybciej.



Reprezentacja graficzna harmonogramu – diagram Gantta

Dlaczego ten harmonogram jest poprawny?

Klasyczna **zasada** poprawności harmonogramu:

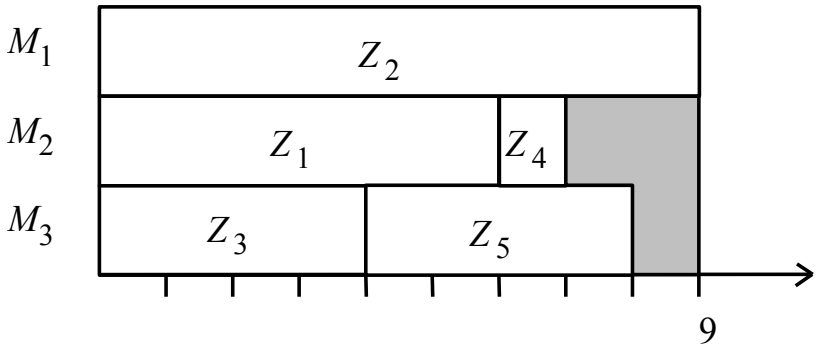
- żadne zadanie nie może być jednocześnie wykonywane przez różne maszyny,
- żaden procesor nie pracuje równocześnie nad różnymi zadaniami,
- inne – wprowadzimy za chwilę ...

Wstęp do deterministycznego szeregowania zadań

Sposoby obsługi zadań

Procesory ***równoległe*** (każdy procesor może obsłużyć każde zadanie):

- ***procesory identyczne*** – wszystkie są jednakowo szybkie,
- ***procesory jednorodne*** – mają różne szybkości, ale stosunki czasów wykonania zadań są niezależne od maszyn,
- ***procesory dowolne*** – prędkości zależą od wykonywanych zadań.



Uszeregowanie na maszynach równoległych

Wstęp do deterministycznego szeregowania zadań

Sposoby obsługi zadań

Procesory ***dedykowane***

- zadania są podzielone na operacje (zadanie Z_j zawiera operacje O_{ij} do wykonania na maszynach M_i, o długościach czasowych p_{ij}). Zadanie kończy się wraz z wykonaniem swej najpóźniejszej operacji,
- dopuszcza się sytuację, gdy zadanie nie wykorzystuje wszystkich maszyn (***operacje puste***),
- żadne dwie operacje tego samego zadania nie mogą wykonywać się równocześnie,
- żaden procesor nie może równocześnie pracować nad różnymi operacjami.

Trzy główne typy systemów obsługi dla maszyn dedykowanych:

- ***system przepływowy*** (***flow shop***) – operacje każdego zadania są wykonywane przez procesory w tej samej kolejności wyznaczonej przez numery maszyn,
- ***system otwarty*** (***open shop***) – kolejność wykonania operacji w obrębie zadań jest dowolna,
- ***system gniazdowy*** (***job shop***) – dla każdego zadania mamy dane przyporządkowanie maszyn operacjom oraz wymagana kolejność.

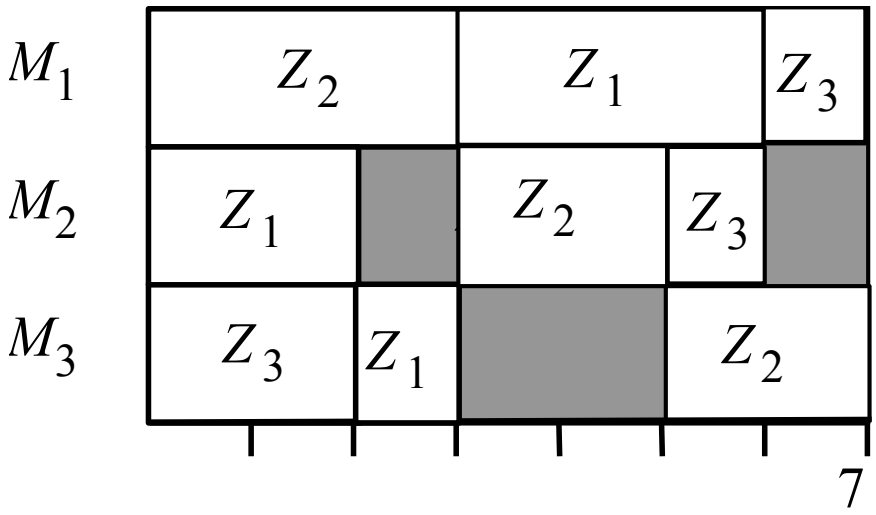
Wstęp do deterministycznego szeregowania zadań

Sposoby obsługi zadań

Procesory ***dedykowane*** – system otwarty (kolejność operacji dowolna).

Przykład. Jednodniowy plan zajęć szkolnych.

		Nauczyciele		
		M_1	M_2	M_3
	Z_1	3	2	1
Klasy	Z_2	3	2	2
	Z_3	1	1	2

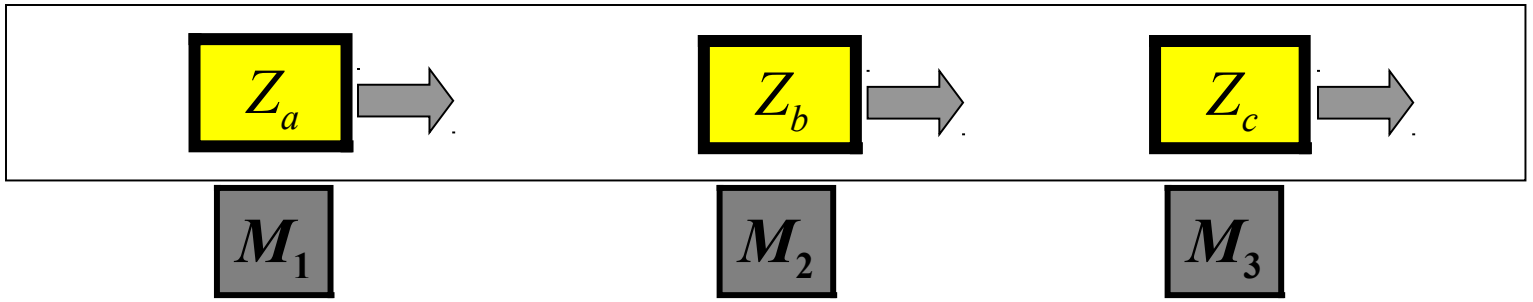


Wstęp do deterministycznego szeregowania zadań

Sposoby obsługi zadań

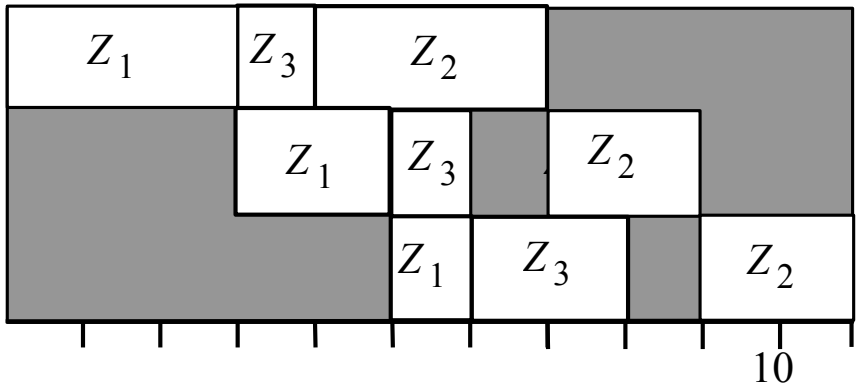
Procesory ***dedykowane*** – system przepływowy (kolejność operacji musi być zgodna z numeracją maszyn).

Przykład. Taśma produkcyjna.



Roboty

Detale	M_1	M_2	M_3
Z_1	3	2	1
Z_2	3	2	2
Z_3	1	1	2



Maszyny dedykowane zostawimy na później ...

Wstęp do deterministycznego szeregowania zadań

Parametry zadań

Dane są: zbiór *n* zadań $Z=\{Z_1,...,Z_n\}$ oraz *m* maszyn (procesorów) $M=\{M_1,...,M_m\}$.

Zadanie Z_j :

• ***Czas wykonywania***. Dla procesorów identycznych jest on niezależny od maszyny i wynosi p_j. Procesory jednorodne M_i charakteryzują się współczynnikami szybkości b_i, wtedy czas dla M_i to p_j/b_i. Dla maszyn dowolnych mamy czasy p_{ij} zależne od zadań i procesorów.

• ***Moment przybycia*** (***release time***) r_j. Czas, od którego zadanie może zostać podjęte. Wartość domyślna – zero.

• ***Termin zakończenia*** d_j. Opcjonalny parametr. Występuje w dwóch wariantach. Może oznaczać czas, od którego nalicza się spóźnienie (***due date***), lub termin, którego przekroczyć nie wolno (***deadline***).

• ***Waga*** w_j – opcjonalny parametr, określający ważność zadania przy naliczaniu kosztu harmonogramu. Domyślnie zadania są jednakowej

Wstęp do deterministycznego szeregowania zadań

Parametry zadań

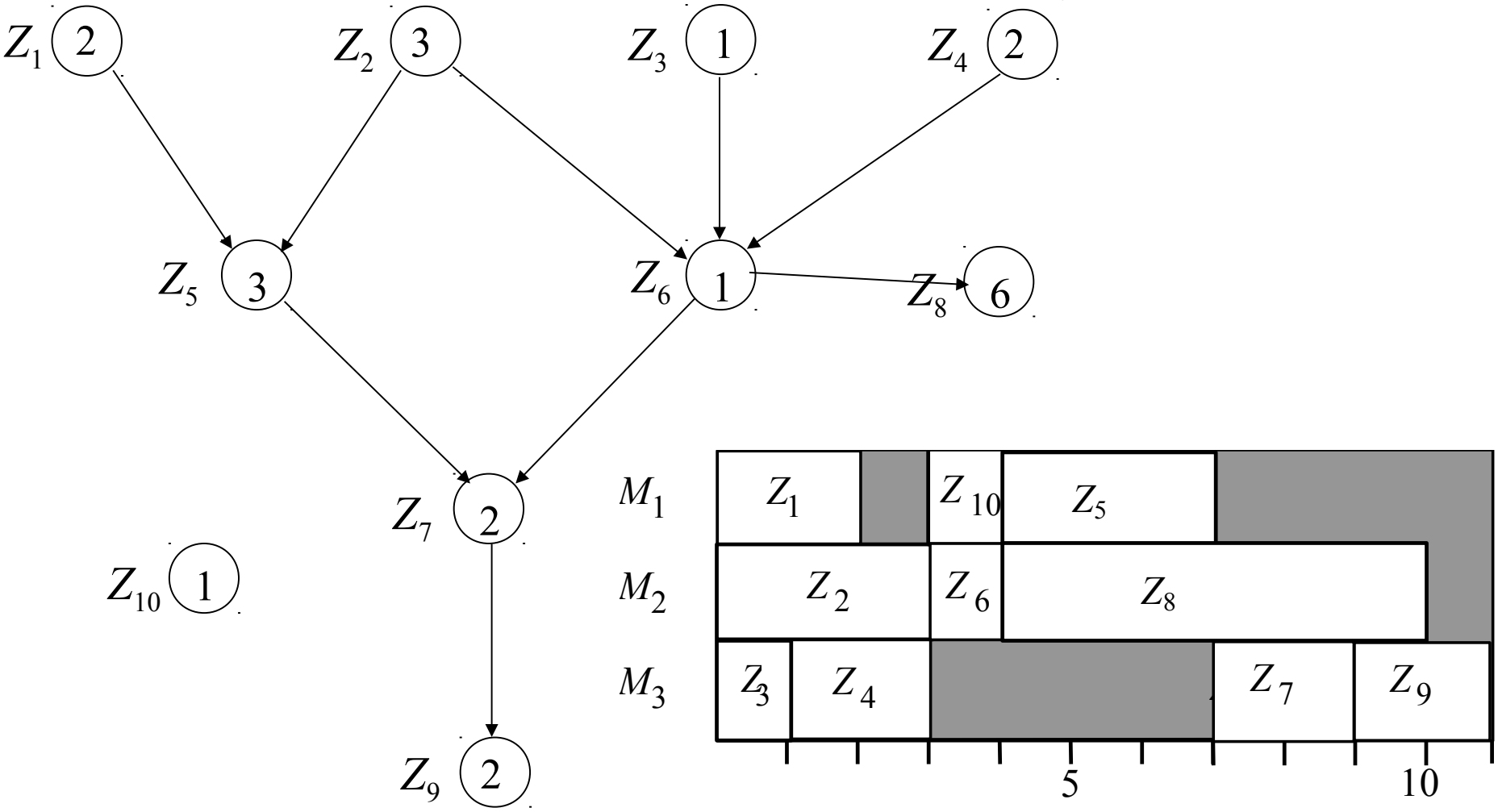
Zadania zależne:

- W zbiorze zadań Z można wprowadzić ***ograniczenia kolejnościowe*** w postaci dowolnej relacji częściowego porządku. Wówczas $Z_i \bullet Z_j$ oznacza, że zadanie Z_j może się zacząć wykonywać dopiero po zakończeniu Z_i (**czemu?** np. Z_j korzysta z wyników pracy Z_i).
- Jeśli ograniczenia te nie występują, mówimy o ***zadaniach niezależnych*** (tak się przyjmuje domyślnie) w przeciwnym razie są one ***zależne***.
- Relację zwykle podaje się w postaci acyklicznego digrafu o wierzchołkach z Z (droga z Z_i do Z_j oznacza, że $Z_i \bullet Z_j$) z łukami przechodnimi, lub bez (tylko relacje nakrywania – *diagram Hassego*).

Wstęp do deterministycznego szeregowania zadań

Parametry zadań

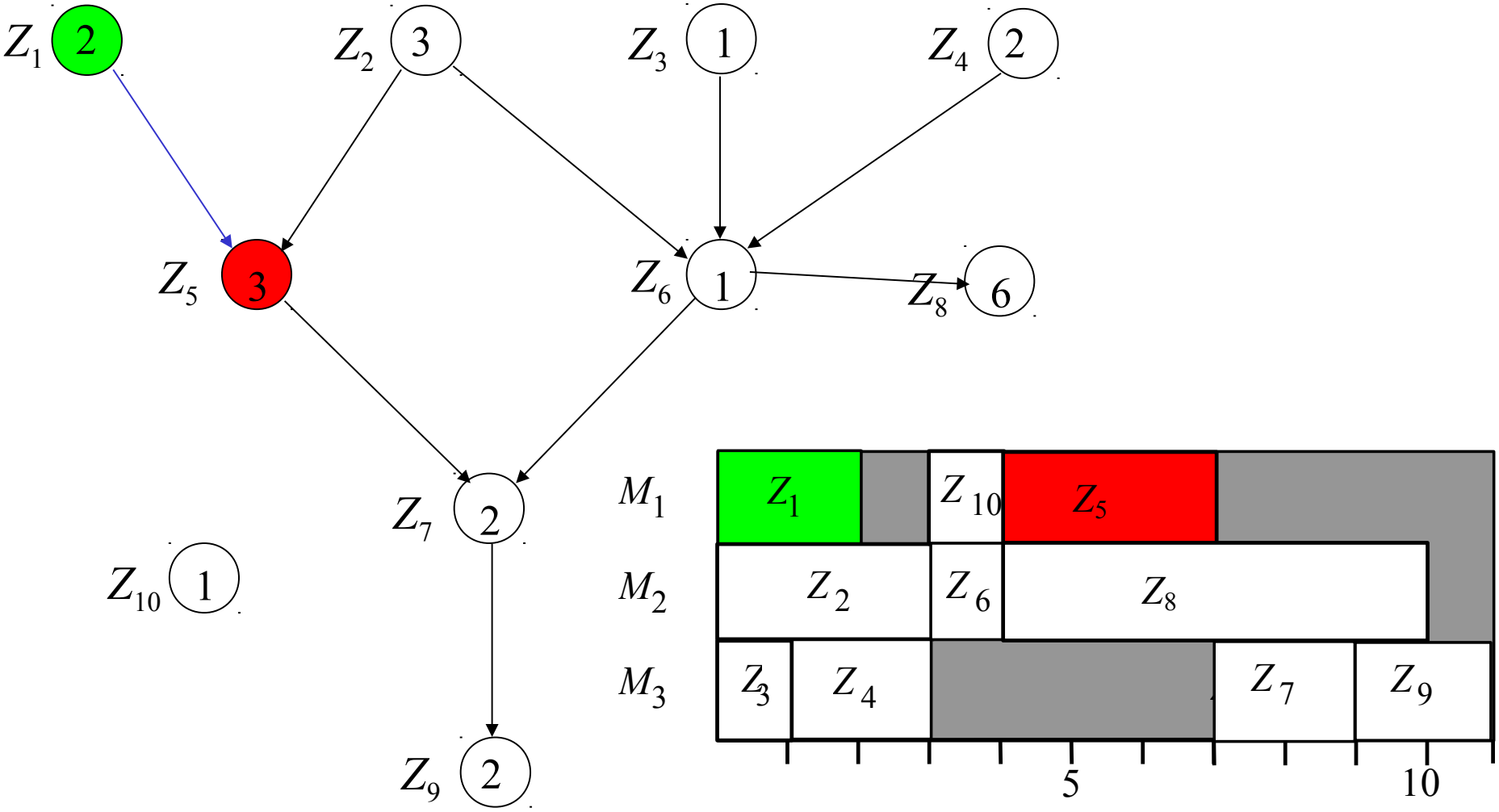
Przykład. Harmonogram dla zadań zależnych (p_j podano w kółkach).



Wstęp do deterministycznego szeregowania zadań

Parametry zadań

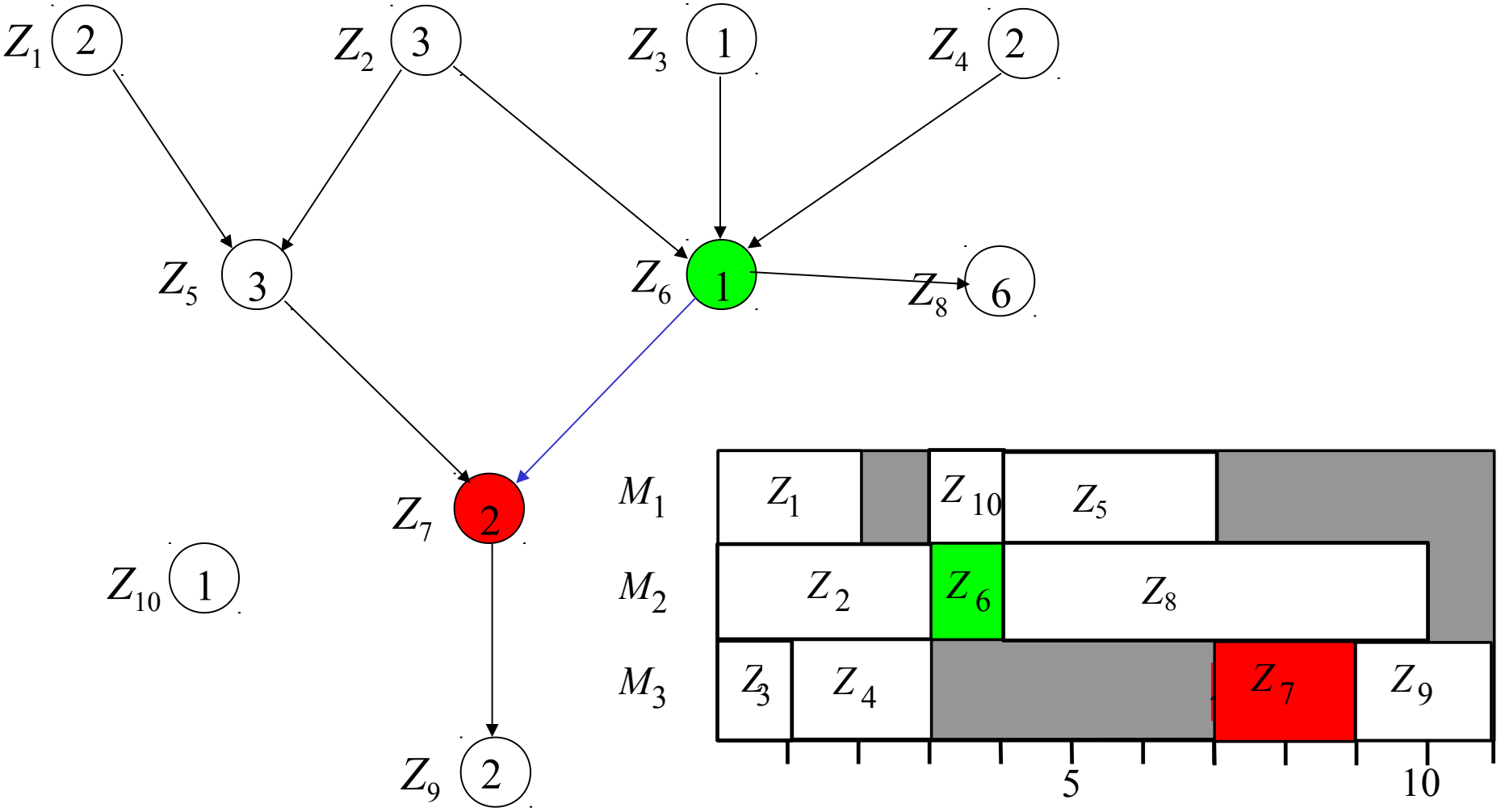
Przykład. Harmonogram dla zadań zależnych (p_j podano w kółkach).



Wstęp do deterministycznego szeregowania zadań

Parametry zadań

Przykład. Harmonogram dla zadań zależnych (p_j podano w kółkach).

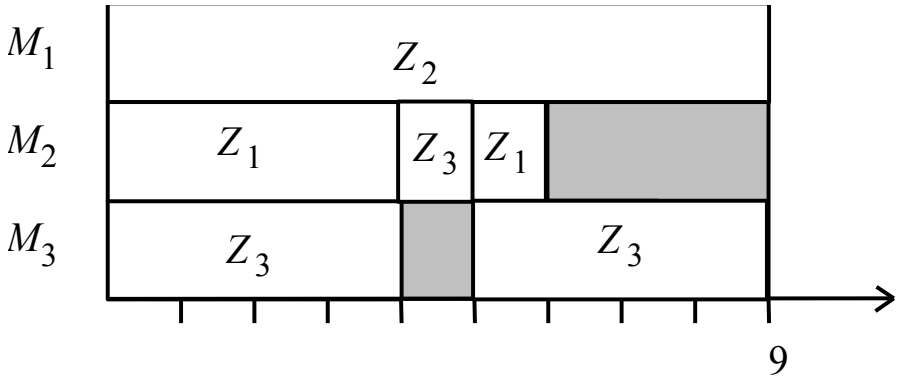


Wstęp do deterministycznego szeregowania zadań

Parametry zadań

Zadania mogą być:

- ***niepodzielne*** – przerwy w wykonaniu są niedopuszczalne (domyślnie),
- ***podzielne*** – wykonanie można przerwać i podjąć ponownie, w przypadku maszyn równoległych nawet na innym procesorze.



Uszeregowanie zadań podzielnych na maszynach równoległych

Wstęp do deterministycznego szeregowania zadań

Zasady poprawności harmonogramu (już w całości):

- w każdej chwili procesor może wykonywać co najwyżej jedno zadanie,
- w każdej chwili zadanie może być obsługiwane przez co najwyżej jeden procesor,
- zadanie Z_j wykonuje się w całości w przedziale czasu $[r_j,\infty)$,
- spełnione są ograniczenia kolejnościowe,
- w przypadku zadań niepodzielnych każde zadanie wykonuje się nieprzerwanie w pewnym domknięto–otwartym przedziale czasowym, dla zadań podzielnych czasy wykonania tworzą skończoną sumę rozłącznych przedziałów.

Wstęp do deterministycznego szeregowania zadań

Kryteria kosztu harmonogramu

Położenie zadania Z_i w gotowym harmonogramie:

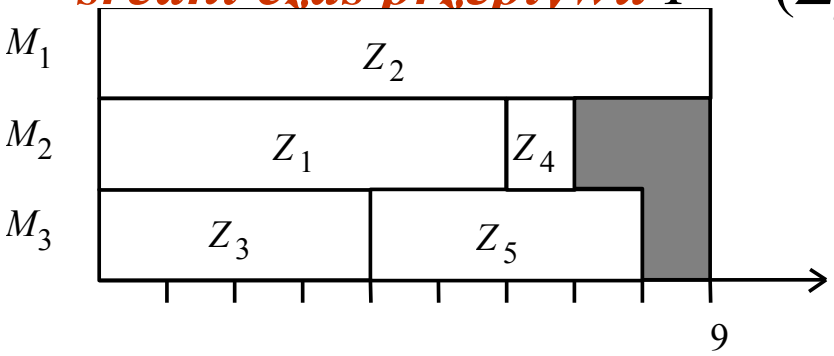
- ***moment zakończenia*** C_i (ang. *completion time*),
- ***czas przepływu*** przez system (*flow time*) $F_i=C_i-r_i$,
- ***opóźnienie*** (*lateness*) $L_i=C_i-d_i$,
- ***spóźnienie*** (*tardiness*) $T_i=\max\{C_i-d_i,0\}$,
- "***znacznik spóźnienia***" $U_i=\mathrm{w}(C_i>d_i)$, a więc odpowiedź (0/1 czyli Nie/Tak) na pytanie „czy zadanie się spóźniło?"

Wstęp do deterministycznego szeregowania zadań

Kryteria kosztu harmonogramu

Najczęściej stosowane:

- ***długość uszeregowania*** $C_{\max}=\max\{C_j : j=1,\ldots,n\}$,
- ***całkowity (łączny) czas zakończenia zadania*** $\Sigma C_j = \Sigma_{i=1,\ldots,n} C_i$,
- ***średni czas przepływu*** $\overline{F} = (\Sigma_{i=1,\ldots,n} F_i)/n$.



$C_{\max} = 9,$

$\Sigma C_j = 6+9+4+7+8 = 34$

Uszeregowanie na trzech maszynach równoległych. $p_1,\ldots,p_5=6,9,4,1,4$.

Można wprowadzać wagi (priorytety) zadań:

- *całkowity ważony czas zakończenia* $\Sigma w_j C_j = \Sigma_{i=1,\ldots,n} w_i C_i$,

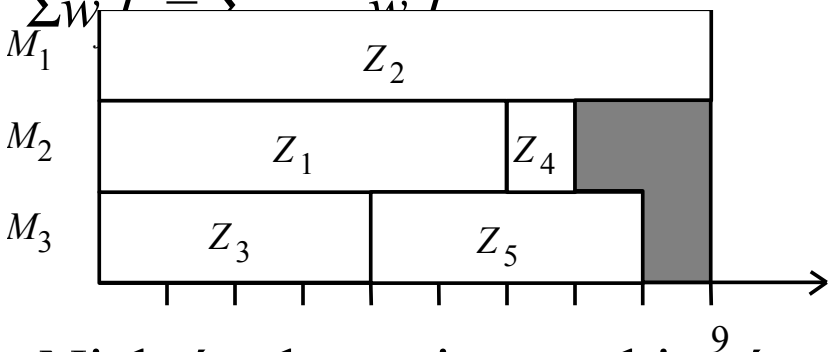
$w_1,\ldots,w_5=1,2,3,1,1$ $\quad\quad$ $\Sigma w_j C_j = 6+18+12+7+8 = 51$

Wstęp do deterministycznego szeregowania zadań

Kryteria kosztu harmonogramu

Oparte na wymaganych terminach zakończenia:

- ***maksymalne opóźnienie*** $L_{\max}=\max\{L_j : j=1,\ldots,n\}$,
- ***maksymalne spóźnienie*** $T_{\max}=\max\{T_j : j=1,\ldots,n\}$,
- ***całkowite spóźnienie*** $\Sigma T_j = \Sigma_{i=1,\ldots,n}\, T_i$,
- ***liczba spóźnionych zadań*** $\Sigma U_j = \Sigma_{i=1,\ldots,n}\, U_i$,
- można wprowadzać wagi zadań, np *łączne ważone spóźnienie* $\Sigma w_i T_i = \Sigma \ldots w_i T_i$



Zadanie:	Z_1	Z_2	Z_3	Z_4	Z_5
$d_i=$	7	7	5	5	8
$L_i=$	–1	2	–1	2	0
$T_i=$	0	2	0	2	0

$L_{\max} = T_{\max} = 2$

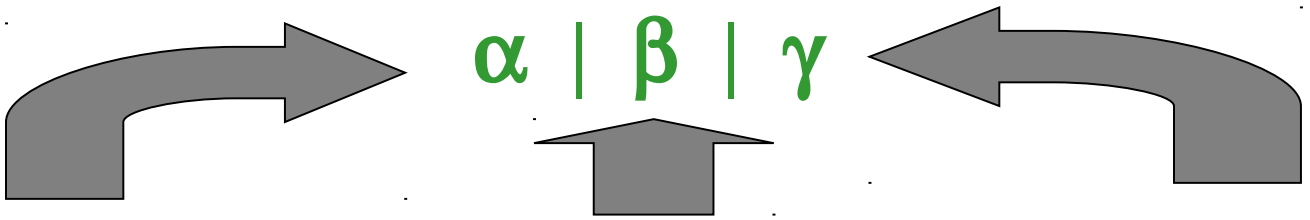
$\Sigma T_j = 4, \Sigma U_j = 2$

Niektóre kryteria są sobie równoważne

$\Sigma L_i = \Sigma C_i - \Sigma d_i$, $\bar{F} = (\Sigma C_i)/n - (\Sigma r_i)/n$.

Wstęp do deterministycznego szeregowania zadań

Jak to opisać? Notacja trójpolowa.



środowisko maszynowe

charakterystyka zadań

kryterium optymalizacji

α może mieć postać:

- *P* – procesory *identyczne*
- *Q* – procesory *jednorodne*
- *R* – procesory *dowolne*
- *O* – *system otwarty* (*open shop*)
- *F* – *system przepływowy* (*flow shop*)
- *PF* – *„permutacyjny” flow shop*
- *J* – *system ogólny* (*job shop*)

Ponadto:

- po symbolu można podać liczbę maszyn np. *O*4,
- dla jednej maszyny piszemy cyfrę 1 bez symbolu (wtedy model nie ma znaczenia),
- piszemy – przy braku maszyn (czynności bezstanowiskowe).

Wstęp do deterministycznego szeregowania zadań

Jak to opisać? Notacja trójpolowa.

β puste to cechy domyślne: zadania są niepodzielne, niezależne, z r_j=0, czasy wykonania i ewentualne wymagane terminy zakończenia d_j dowolne.

β Możliwe wartości:

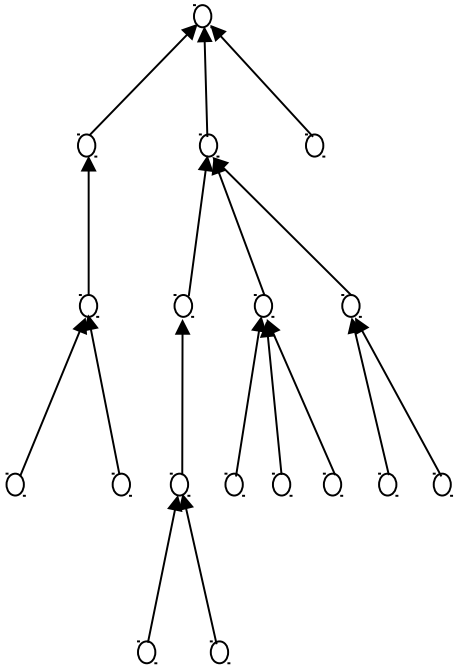
- *pmtn* – zadania podzielne (*preemption*),
- *res* – wymagane są dodatkowe zasoby (nie omawiamy),
- *prec* – zadania zależne,
- r_j – występują różne wartości momentów przybycia,
- p_j=1 lub *UET* – zadania o jednostkowym czasie wykonania,
- $p_{ij} \in \{0,1\}$ lub *ZUET* – operacje w zadaniach są jednostkowe lub puste (procesory dedykowane),
- $C_j \leq d_j$ – istnieją wymagane i nieprzekraczalne terminy zakończenia zadań,
- *no–idle* – procesory muszą pracować w sposób ciągły, bez okienek,
- *no–wait* – okienka między operacjami w zadaniach są zabronione

Wstęp do deterministycznego szeregowania zadań

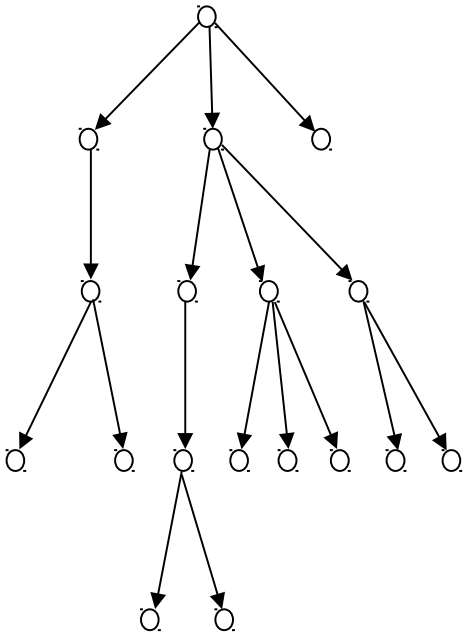
Jak to opisać? Notacja trójpolowa.

β Możliwe wartości:

• *in–tree, out–tree, chains* ... – różne szczególne postaci relacji zależności kolejnościowych (*prec*).



in–tree



out–tree

Wstęp do deterministycznego szeregowania zadań

Jak to opisać? Notacja trójpolowa.

Przykłady.

$P3|prec|C_{max}$ – szeregowanie niepodzielnych zadań zależnych na trzech identycznych maszynach równoległych w celu zminimalizowania długości harmonogramu.

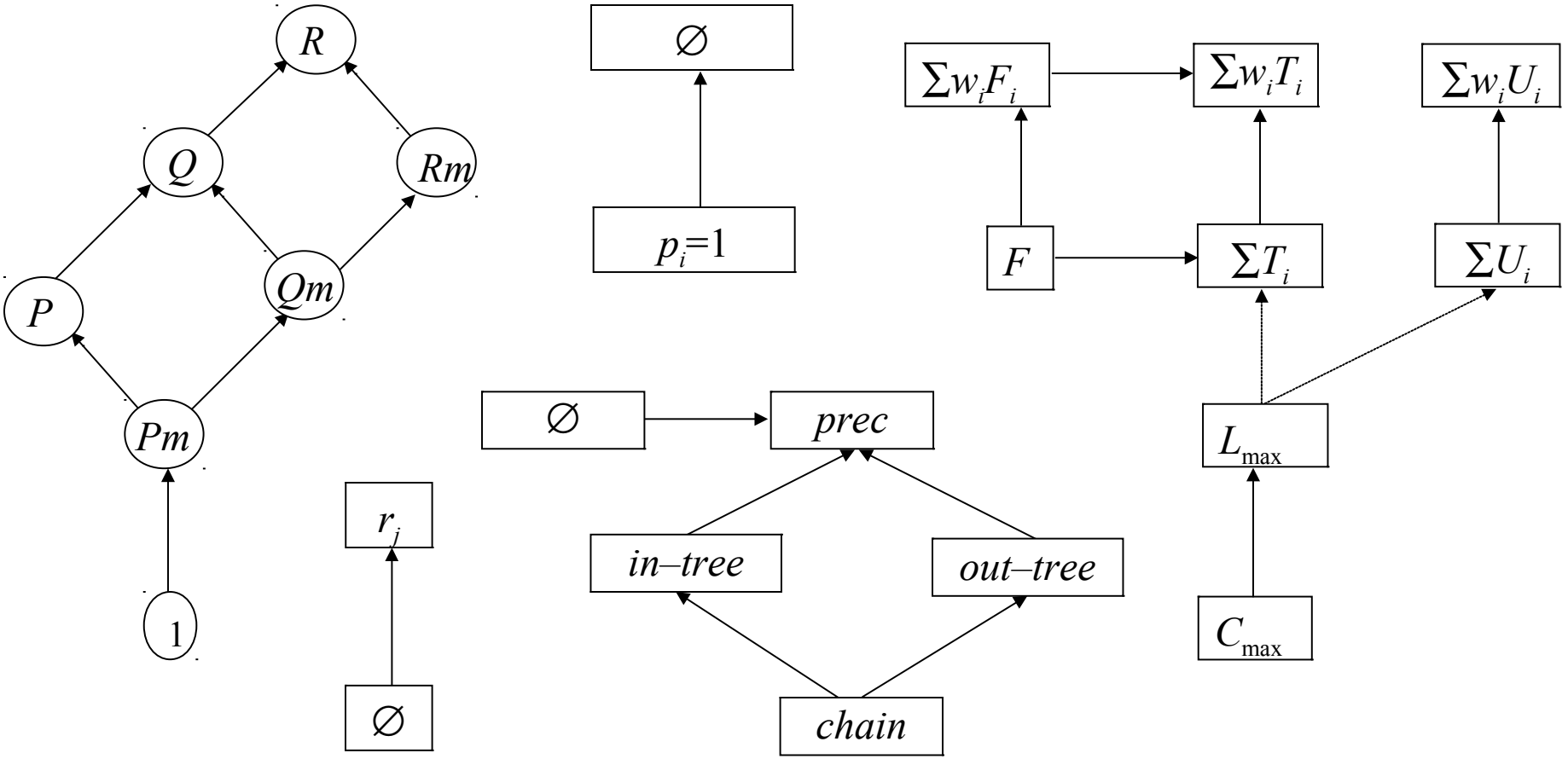
$R|pmtn,prec,r_i|\Sigma U_i$ – szeregowanie podzielnych zadań zależnych z różnymi czasami przybycia i terminami zakończenia na równoległych dowolnych maszynach (liczba procesorów jest częścią danych) w celu minimalizacji liczby zadań spóźnionych.

$1|r_i,C_i \leq d_i|-$ – pytanie o istnienie (brak kryterium kosztu, więc nic nie optymalizujemy!) uszeregowania zadań niepodzielnych i niezależnych o różnych momentach przybycia na jednej maszynie, tak by żadne zadanie nie było spóźnione.

Wstęp do deterministycznego szeregowania zadań

Redukcje podproblemów do problemów ogólniejszych

Przykłady.



Wstęp do deterministycznego szeregowania zadań

Złożoność problemów szeregowania

Jeżeli uwzględnimy tylko liczby maszyn 1,2,3,•, to istnieje 4536 problemów, z których:

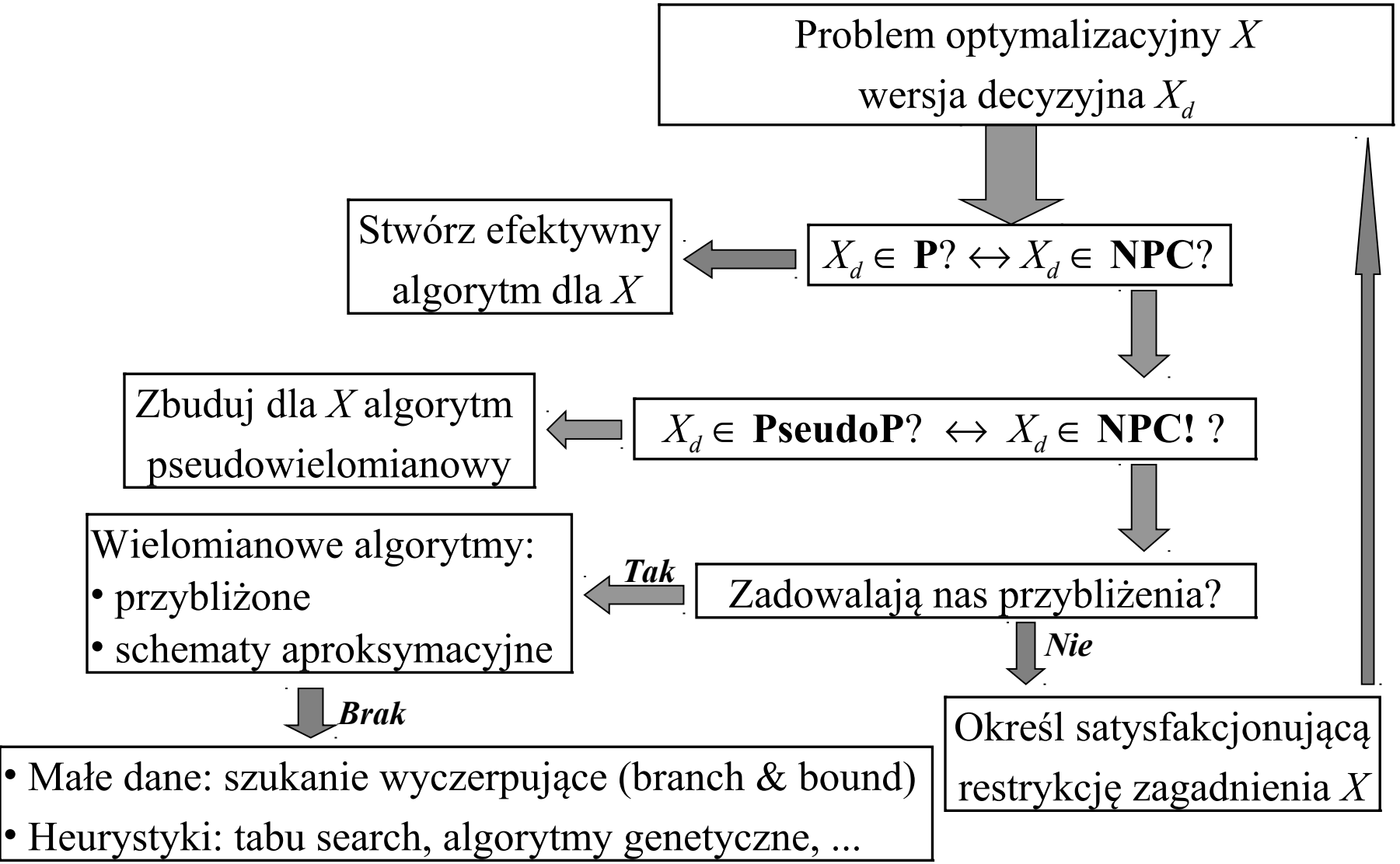
- 416 – wielomianowe,
- 3817 – NP–trudne,
- 303 – otwarte.

Jak sobie radzić z NP–trudnością?

- wielomianowe algorytmy *przybliżone* o gwarantowanej dokładności względnej,
- dokładne algorytmy *pseudowielomianowe*,
- algorytmy dokładne, szybkie tylko w *średnim przypadku*,
- *heurystyki* wyszukujące (np. *tabu search, algorytmy genetyczne*),
- dla małych rozmiarów danych – wykładnicze *przeszukiwanie wyczerpujące* (np. *branch–bound*).

Wstęp do deterministycznego szeregowania zadań

Ogólny schemat analizy zagadnienia



Szeregowanie operacji bezprocesorowych. Metoda ścieżki krytycznej.

Model –$|prec|C_{max}$ operacji o różnych czasach wykonania, z zależnościami kolejnościowymi, ale nie wymagających procesorów. Celem jest znalezienie najkrótszego możliwego harmonogramu.

Relacja zależności kolejnościowych • to **częściowy porządek** (bez przekątnej) w zbiorze zadań, czyli jest ona:

- przeciwzwrotna: $\forall_{Zi} \neg Z_i \bullet Z_i$
- przechodnia $\forall_{Zi,Zj,Zk,}(Z_i \bullet Z_j \wedge Z_j \bullet Z_k) \Rightarrow Z_i \bullet Z_k$

Szeregowanie operacji bezprocesorowych. Metoda ścieżki krytycznej.

Metody reprezentacji relacji zależności kolejnościowych • za pomocą ***digrafu acyklicznego***.

Sieć AN (*activity on node*):

• wierzchołki odpowiadają operacjom, ich wagi (liczby naturalne) są równe czasom wykonywania,

• $Z_i \bullet Z_j \Leftrightarrow$ w sieci istnieje ścieżka skierowana z wierzchołka Z_i do wierzchołka Z_j,

• zwykle usuwa się *łuki przechodnie* (jak w diagramie Hassego).

Sieć AA (*activity on arc*):

• łuki odpowiadają operacjom, ich długości są równe czasom wykonywania,

• przez każdy wierzchołek przechodzi droga z Z (źródło) do U (ujście),

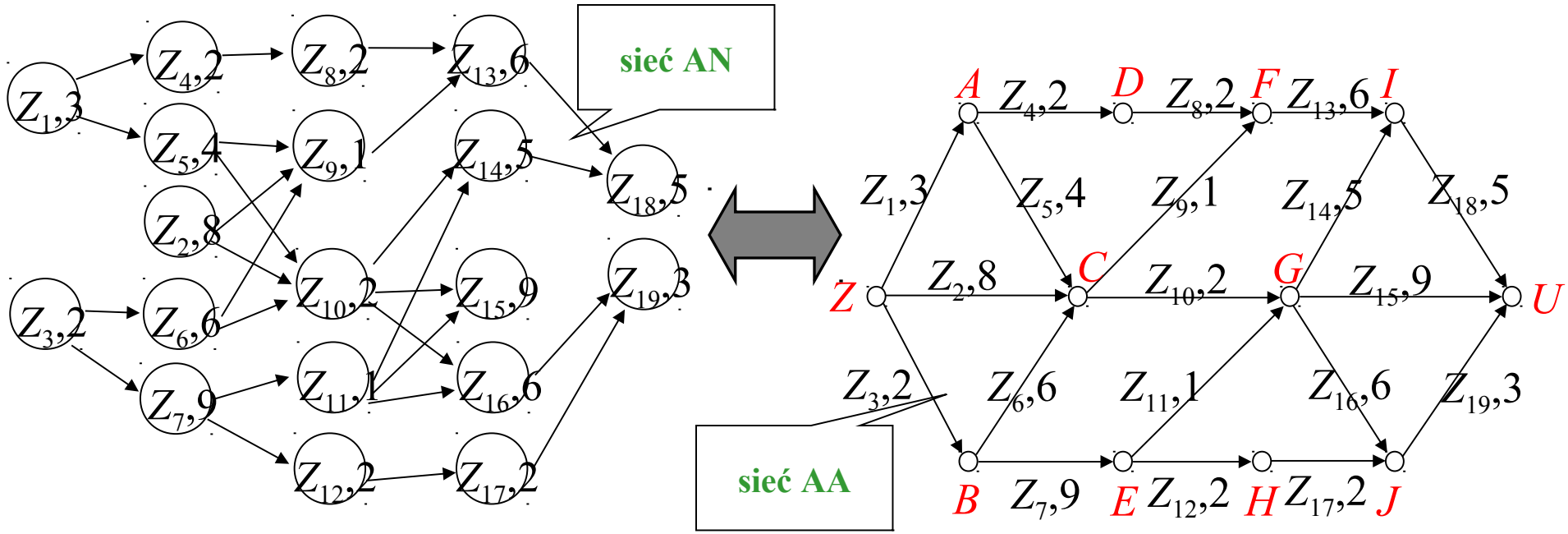
• $Z_i \bullet Z_j \Leftrightarrow$ łuk Z_i kończy się w początku łuku Z_j, lub też w sieci istnieje ścieżka skierowana z końca łuku Z_i do początku Z_j,

• można wprowadzać *operacje pozorne* – łuki o zerowej długości.

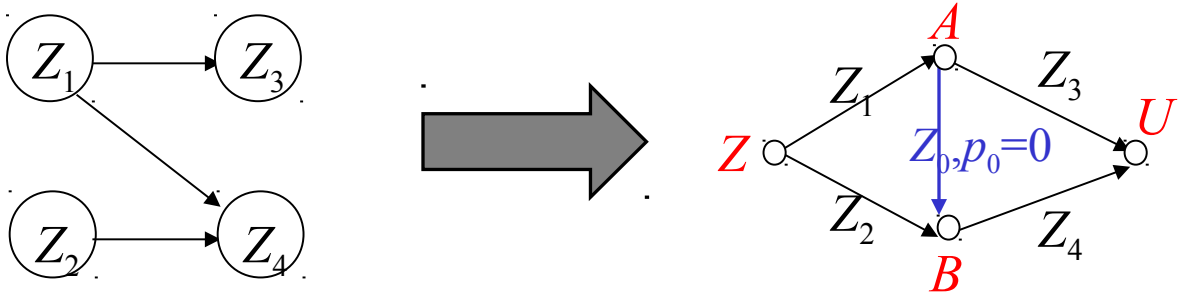
Szeregowanie operacji bezprocesorowych. Metoda ścieżki krytycznej.

Metody reprezentacji relacji • za pomocą ***digrafu acyklicznego***.

Przykład. Ta sama relacja porządku dla zbioru 19 operacji.



Przykład. Przy translacji AN ➔ AA niekiedy trzeba wprowadzić (zerowe) operacje pozorne.



Szeregowanie operacji bezprocesorowych. Metoda ścieżki krytycznej.

Model –$|prec|C_{max}$ operacji o różnych czasach wykonania, z zależnościami kolejnościowymi, ale nie wymagających procesorów. Celem jest znalezienie najkrótszego możliwego harmonogramu.

Zasada: dla każdej operacji określamy najwcześniejszy możliwy moment uruchomienia tj. maksymalną „długość" ścieżki doń prowadzącej.

Jak to zrobić?

Algorytm dla ***AN***:

1. numeruj wierzchołki „topologicznie" (brak łuków „pod prąd"),
2. wierzchołkom Z_a bez poprzedników nadaj etykietę $l(Z_a)=0$, a kolejnym wierzchołkom Z_i przypisuj $l(Z_i)=\max\{l(Z_j)+p_j$: istnieje łuk z Z_j do $Z_i\}$,

Wynik: $l(Z_i)$ jest najwcześniejszym możliwym terminem rozpoczęcia Z_i.

Algorytm dla ***AA***:

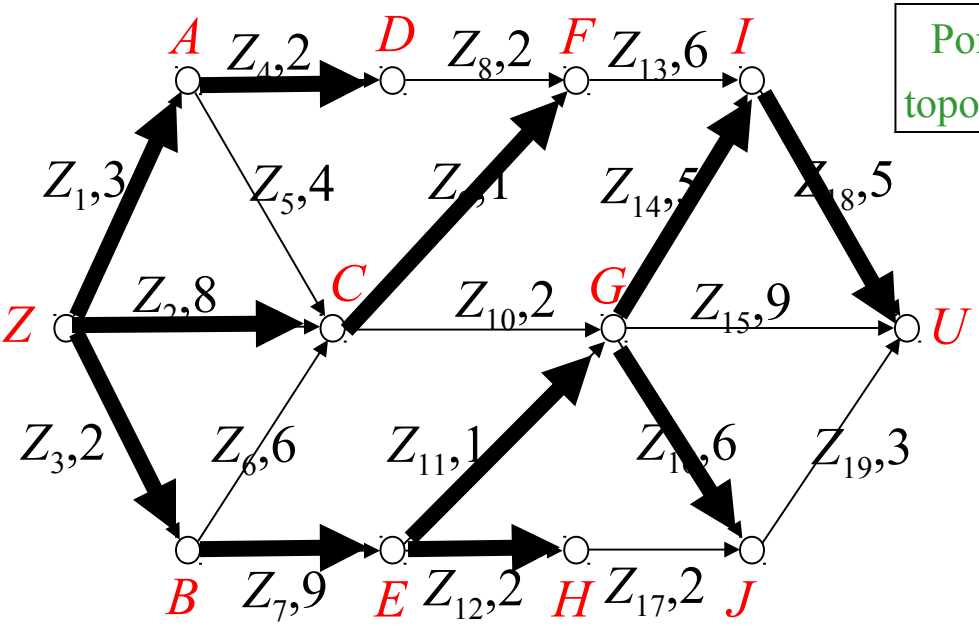
1. numeruj wierzchołki „topologicznie",
2. źródłu Z nadaj etykietę $l(Z)=0$, a kolejnym wierzchołkom v przypisuj $l(v)=\max\{l(u)+p_j$: łuk Z_j prowadzi z u do $v\}$,

Wynik: $l(v)$ wierzchołka początkowego Z_j jest najwcześniejszym możliwym terminem rozpoczęcia tej operacji. $l(U)$ to termin zakończenia harmonogramu.

Szeregowanie operacji bezprocesorowych. Metoda ścieżki krytycznej.

Model –$|prec|C_{max}$ operacji o różnych czasach wykonania, z zależnościami kolejnościowymi, ale nie wymagających procesorów. Celem jest znalezienie najkrótszego możliwego harmonogramu.

Przykład. Harmonogram dla sieci AA złożonej z 19 operacji.

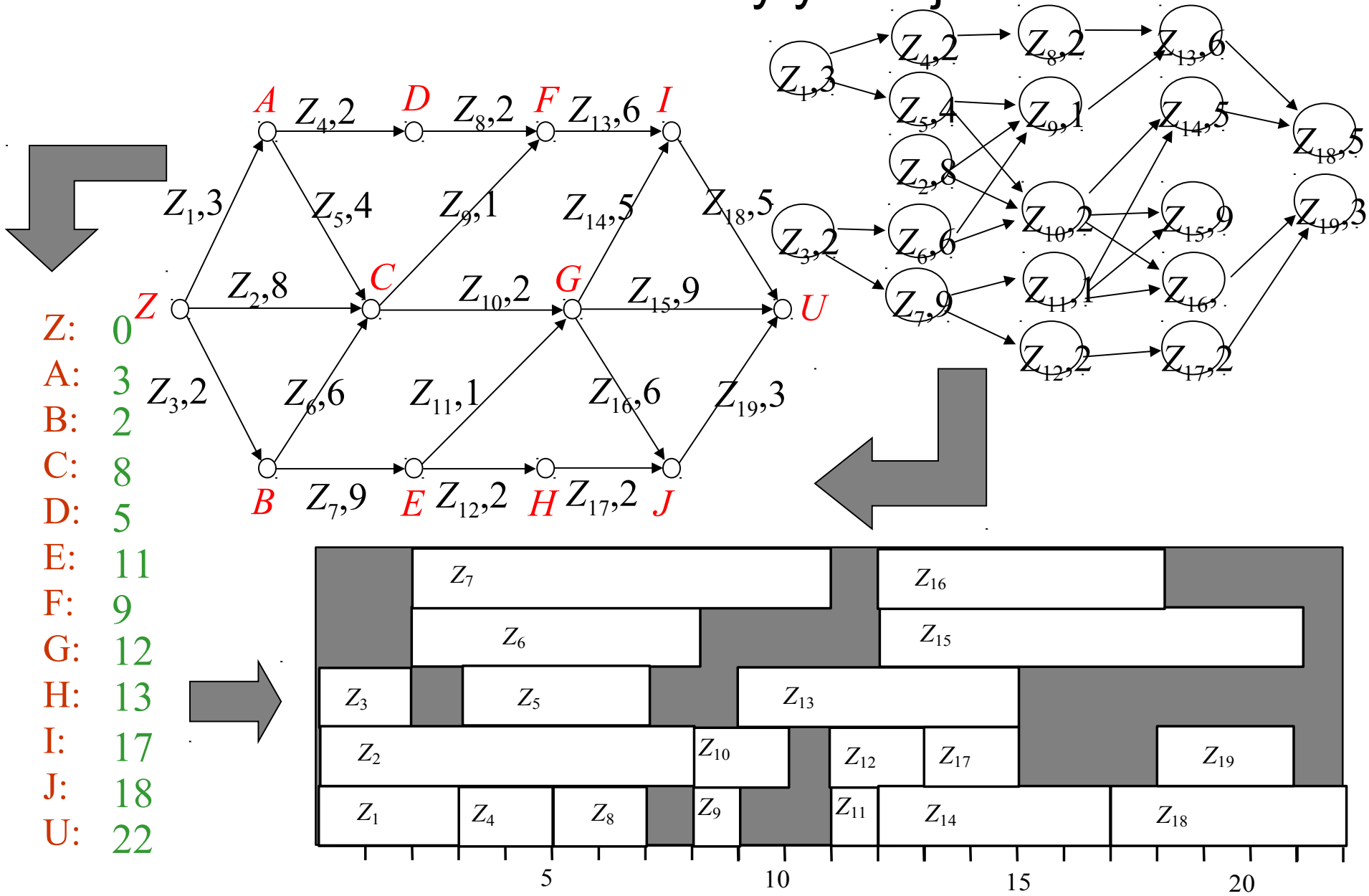


Porządek topologiczny

Terminy uruchomienia

Z:	0	
A:	3	Z:0+3
B:	2	Z:0+2
C:	8	Z:0+8, A:3+4, B:2+6
D:	5	A:3+2
E:	11	B:2+9
F:	9	C:8+1, D:5+2
G:	12	C:8+2, E:11+1
H:	13	E:11+2
I:	17	F:9+6, G:12+5
J:	18	G:12+6, H:13+2
U:	22	I:17+5, J:18+3, G:12+9

Szeregowanie operacji bezprocesorowych. Metoda ścieżki krytycznej.



Szeregowanie operacji bezprocesorowych. Metoda ścieżki krytycznej.

• Algorytmy ścieżki krytycznej minimalizują nie tylko C_{max}, ale wszystkie zdefiniowane wcześniej funkcje kryterialne.

• Możemy wprowadzić do modelu różne wartości terminów przybycia r_j dla zadań Z_j – dodając „sztuczne” zadania (o długości r_j):

- jako wierzchołki – poprzednicy w modelu AN,
- jako łuk prowadzący ze źródła Z do początku łuku Z_j w modelu AA.

Podstawowe problemy optymalizacji dyskretnej

- Zagadnienie ***maksymalnego przepływu w sieci***. Dany jest multidigraf bez pętli $D(V,E)$ o łukach obciążonych wagami $w:E\to N$ (przepustowość) i dwóch wyróżnionych i różnych wierzchołkach z (źródło) i u (ujście). Znajdź ***przepływ*** $p:E\to N\cup\{0\}$ o maksymalnej możliwej ***objętości***.

Co to jest przepływ o objętości P?

- $\forall_{e\in E}\, p(e)\le w(e)$, (nie wolno przekroczyć przepustowości łuków)
- $\forall_{v\in V-\{z,u\}}\ \Sigma_{e \text{ wchodzi do } v}\, p(e) - \Sigma_{e \text{ wychodzi z } v}\, p(e) = 0$,

(do „zwykłego” wierzchołka „wpływa” tyle ile „wypływa”)

- $\Sigma_{e \text{ wchodzi do } u}\, p(e) - \Sigma_{e \text{ wychodzi z } u}\, p(e) = P$,

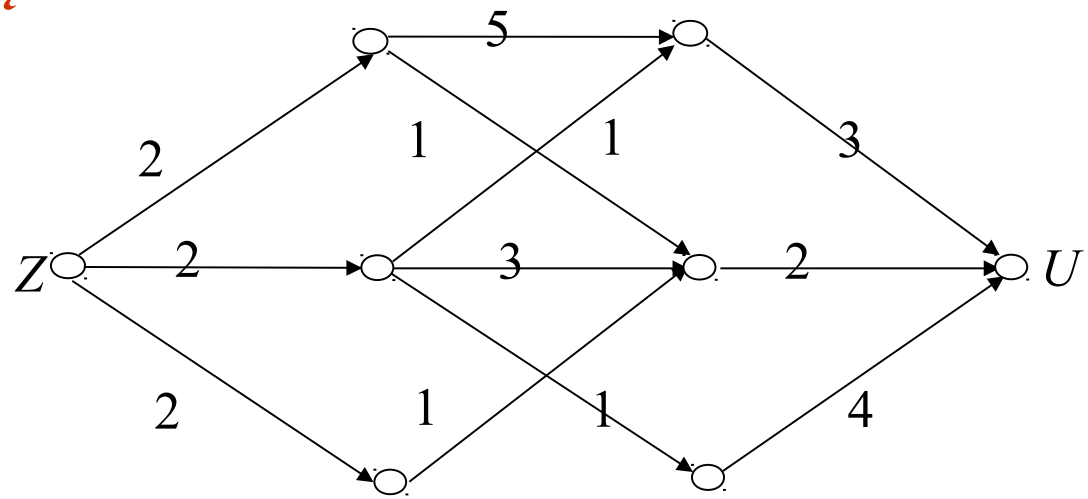
(przez ujście „wypływa” z sieci P jednostek)

- $\Sigma_{e \text{ wchodzi do } z}\, p(e) - \Sigma_{e \text{ wychodzi z } z}\, p(e) = -P$.

(wniosek: do źródła „wpływa” P jednostek)

Podstawowe problemy optymalizacji dyskretnej

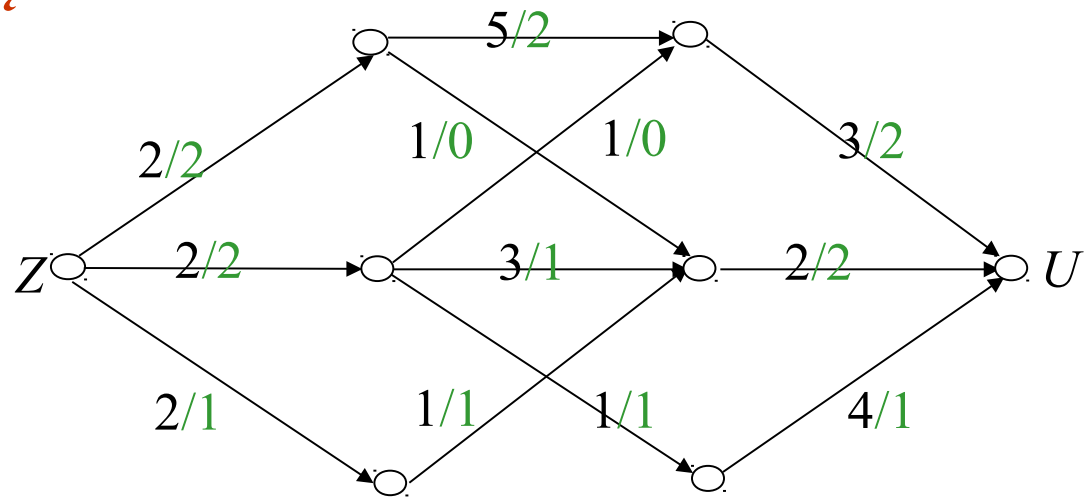
- Zagadnienie ***maksymalnego przepływu w sieci***. Dany jest multidigraf bez pętli $D(V,E)$ o łukach obciążonych wagami $w:E\rightarrow N$ (przepustowość) i dwóch wyróżnionych i różnych wierzchołkach z (źródło) i u (ujście). Znajdź ***przepływ*** $p:E\rightarrow N\cup\{0\}$ o maksymalnej możliwej ***objętości***.



Sieć, przepustowości łuków.

Podstawowe problemy optymalizacji dyskretnej

- Zagadnienie ***maksymalnego przepływu w sieci***. Dany jest multidigraf bez pętli $D(V,E)$ o łukach obciążonych wagami $w:E\to N$ (przepustowość) i dwóch wyróżnionych i różnych wierzchołkach z (źródło) i u (ujście). Znajdź ***przepływ*** $p:E\to N\cup\{0\}$ o maksymalnej możliwej ***objętości***.



... i przepływ. $P=5$

Złożoność $O(|V||E|) \le O(|V|^3)$.

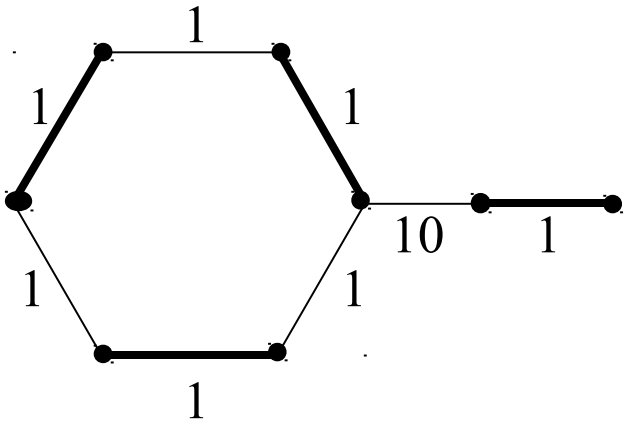
Podstawowe problemy optymalizacji dyskretnej

- Różne modele ***kolorowania grafów***.
- Problemy ***najdłuższej (najkrótszej) drogi*** w grafie.
- Zagadnienia ***programowania liniowego*** – są rozwiązywalne w czasie wielomianowym.

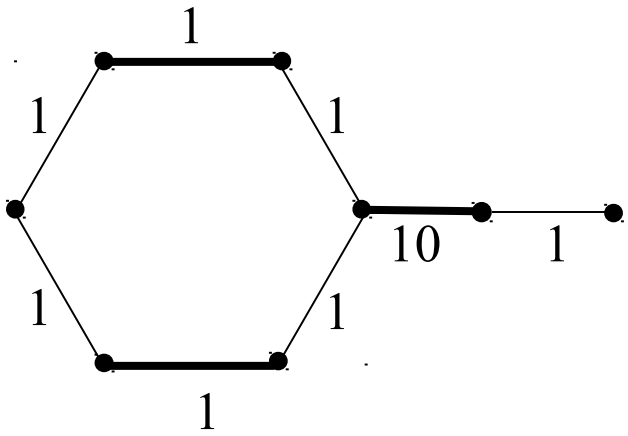
- Wyszukiwanie ***skojarzeń w grafach***. Dany jest graf $G(V,E)$ i funkcja wag zadana na krawędziach $w:E\rightarrow N\cup\{0\}$. ***Skojarzeniem*** nazywamy dowolny podzbiór $A\subseteq E$ o krawędziach niesąsiadujących.

- ***Największe skojarzenie***: znajdź skojarzenie o maksymalnej możliwej liczbie krawędzi ($\alpha(L(G))$). **Złożoność $O(|E||V|^{1/2})$.**
- ***Najcięższe (najlżejsze) skojarzenie o danym rozmiarze***. Dla danej liczby $k\leq\alpha(L(G))$ znajdź skojarzenie o k krawędziach i maksymalnej (minimalnej) możliwej sumie wag.
- ***Najcięższe skojarzenie***. Znajdź skojarzenie o maksymalnej możliwej sumie wag. **Złożoności $O(|V|^3)$** dla grafów dwudzielnych i **$O(|V|^4)$** dla dowolnych grafów.

Podstawowe problemy optymalizacji dyskretnej



Liczność: 4
Waga: 4



Liczność: 3
Waga: 12

Największe skojarzenie nie musi być najcięższym i odwrotnie.

Minimalizacja długości harmonogramu. Maszyny równoległe.

Procesory identyczne, zadania niezależne

Zadania podzielne $P|pmtn|C_{max}$.

Algorytm ***McNaughtona*** Złożoność $O(n)$

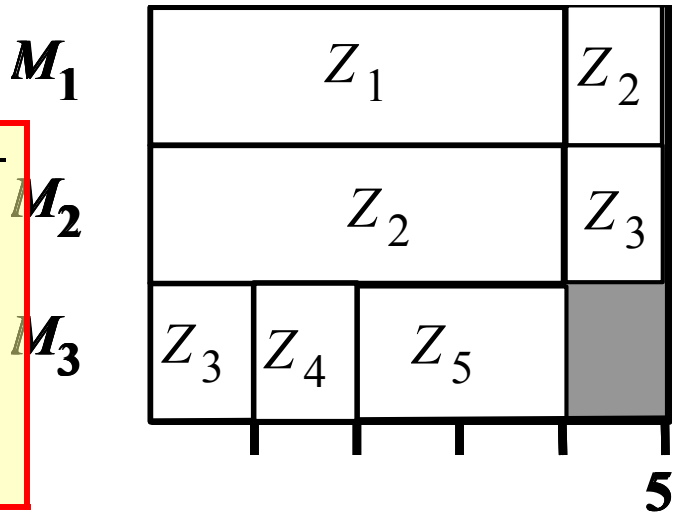
1. Wylicz optymalną długość $C_{max}^* = \max\{\Sigma_{j=1,\ldots,n} p_j/m, \max_{j=1,\ldots,n} p_j\}$,
2. Szereguj kolejno zadania na maszynie, po osiągnięciu C_{max}^* przerwij zadanie i (jeśli się nie zakończyło) kontynuuj je na następnym procesorze począwszy od chwili 0.

Przykład. $m=3$, $n=5$, $p_1,\ldots,p_5=4,5,2,1,2$.

$\Sigma_{i=1,\ldots,5} p_i=14$, $\max p_i=5$,

$C_{max}^* = \max\{14/3,5\}=5$.

Uwaga oznaczenie: przez *X (gdzie *X* – nazwa kryterium) będziemy rozumieli *wartość optymalną* (tj. najmniejszą możliwą) tego kryterium dla konkretnej instancji problemu szeregowania np. C_{max}^*, L_{max}^*.**



Minimalizacja długości harmonogramu. Maszyny równoległe.

Procesory identyczne, zadania niezależne

Zadania niepodzielne $P||C_{max}$.

Problem jest NP–trudny już na dwóch maszynach ($P2||C_{max}$).

Dowód. ***Problem podziału***: dany jest ciąg $a_1,...a_n$ liczb naturalnych o $S=\Sigma_{i=1,...,n} a_i$ parzystej. Czy istnieje jego podciąg o sumie $S/2$?

Redukcja PP ➔ $P2||C_{max}$: bierzemy n zadań o $p_j=a_j$ ($j=1,...,n$), dwie maszyny, pytamy o istnienie uszeregowania z $C_{max} \leq S/2$.

Dokładny algorytm dynamiczny o czasie pracy $O(nC^m)$, gdzie $C \geq C_{max}$*.

Minimalizacja długości harmonogramu. Maszyny równoległe.

Procesory identyczne, zadania niezależne

Zadania niepodzielne $P||C_{max}$.

Wielomianowe algorytmy przybliżone.

Szeregowanie listowe (*List Scheduling* LS) – stosowane w rozmaitych zagadnieniach:

- Ustal kolejność zadań na liście,
- Za każdym razem, gdy zwalnia się jakaś maszyna/maszyny, wybieraj pierwsze (według „listy") *wolne* (w tym momencie) zadania i przypisuj je do zwalniających się procesorów.

Dotyczy problemów z zależnościami kolejnościowymi. Zadanie Z_i jest *wolne* od chwili, w której ukończony został jej ostatni poprzednik Z_j (tj. $Z_j \bullet Z_i$).

Zadania niezależne zawsze są wolne.

Minimalizacja długości harmonogramu. Maszyny równoległe.

Procesory identyczne, zadania niezależne

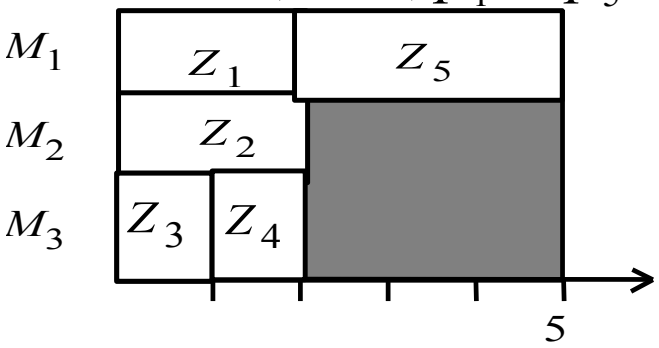
Zadania niepodzielne $P||C_{max}$.

Wielomianowe algorytmy przybliżone.

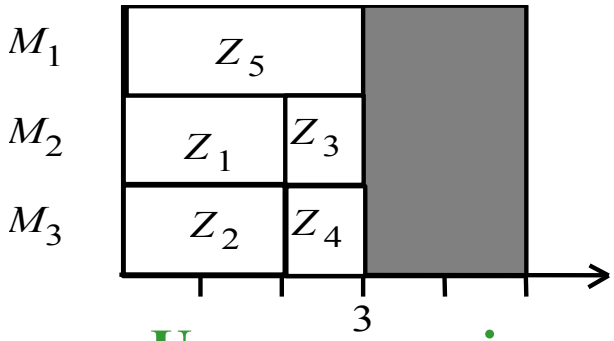
Szeregowanie listowe (*List Scheduling* LS) – stosowane w rozmaitych zagadnieniach:

- Ustal kolejność zadań na liście,
- Za każdym razem, gdy zwalnia się jakaś maszyna/maszyny, wybieraj pierwsze (według „listy") *wolne* (w tym momencie) zadania i przypisuj je do zwalniających się procesorów.

Przykład. m=3, n=5, $p_1,...,p_5$=2,2,1,1,3.



Uszeregowanie listowe



Uszeregowanie optymalne

Minimalizacja długości harmonogramu. Maszyny równoległe.

Procesory identyczne, zadania niezależne

Zadania niepodzielne $P||C_{\max}$.

Wielomianowe algorytmy przybliżone.

Szeregowanie listowe (*List Scheduling* LS) w skrócie:

• Z ustalonego ciągu zadań wybieraj pierwsze wolne (według „listy"), przypisując je zawsze do zwalniającego się procesora.

Dokładność. LS jest 2–przybliżone: $C_{\max}(\mathrm{LS}) \le (2 - m^{-1}) C_{\max}^*$.

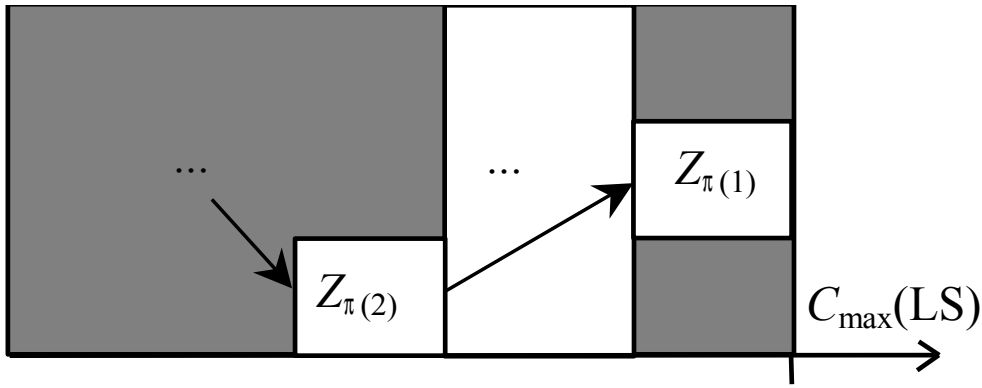
Dowód (obejmuje ogólniejszy model zadań z zależnościami kolejnościowymi $P|prec|C_{\max}$). W harmonogramie LS znajdujemy łańcuch zadań $Z_{\pi(1)},\ldots, Z_{\pi(k)}$: $Z_{\pi(1)}$ – skończone najpóźniej, $Z_{\pi(2)}$ – jego skończony najpóźniej poprzednik (tj. $Z_{\pi(2)} \bullet Z_{\pi(1)}$) itd. aż do zadania bez poprzednika.

$$C^*_{\max}{}^{(pmtn)} \le C_{\max}^* \le$$

$$\le C_{\max}(\mathrm{LS}) \le \Sigma_{i=1,\ldots,k}\, p_{\pi(i)} + \Sigma_{i \notin \pi} p_i/m =$$

$$= (1 - 1/m)\Sigma_{i=1,\ldots,k}\, p_{\pi(i)} + \Sigma_i p_i/m \le$$

$$\le (2 - 1/m) C^*_{\max}{}^{(pmtn)} < (2 - 1/m)\, C_{\max}^*$$



Minimalizacja długości harmonogramu. Maszyny równoległe.

Procesory identyczne, zadania niezależne

Zadania niepodzielne $P||C_{max}$.

Wielomianowe algorytmy przybliżone.

Szeregowanie LPT (*Longest Processing Time*):

• Szereguj listowo, przy czym zadania na liście są wstępnie posortowane według nierosnących czasów wykonania p_i.

Dokładność. LS jest 4/3–przybliżone: $C_{max}(LPT) \leq (4/3 - (3m)^{-1}) C_{max}^*$.

Znany jest ***wielomianowy schemat aproksymacyjny*** oparty na całkowitoliczbowym programowaniu liniowym.

Procesory dowolne, zadania niezależne

Zadania podzielne $R|pmtn|C_{max}$

Istnieje algorytm wielomianowy – wrócimy do tego ...

Zadania niepodzielne $R||C_{max}$

• Oczywiście problem jest NP–trudny (uogólnienie $P||C_{max}$).

• Podproblem $Q|p_i=1|C_{max}$ można rozwiązać w czasie wielomianowym.

• W praktyce stosuje się LPT.

Minimalizacja długości harmonogramu. Maszyny równoległe.

Procesory identyczne, zadania zależne

Zadania podzielne $P|pmtn,prec|C_{max}$.

- W ogólności jest to problem NP–trudny.
- Istnieje algorytm $O(n^2)$ dla $P2|pmtn,prec|C_{max}$ i $P|pmtn,forest|C_{max}$.
- Pomiędzy optymalnym harmonogramem z przerwami i bez zachodzi:

$$C^*_{max} \leq C_{max}(\mathrm{LS}) \leq (2 - m^{-1}) C^{*(pmtn)}_{max}$$

Dowód. Analogiczny jak w przypadku szeregowania listowego.

Minimalizacja długości harmonogramu. Maszyny równoległe.

Procesory identyczne, zadania zależne

Zadania niepodzielne $P|prec|C_{max}$.

- Oczywiście problem jest NP–trudny.
- Najbardziej znane przypadki wielomianowe dotyczą zadań jednostkowych:
  - $P|p_i=1,in\text{–}forest|C_{max}$ i $P|p_i=1,out\text{–}forest|C_{max}$ (***Algorytm Hu***, złożoność $O(n)$),
  - $P2|p_i=1,prec|C_{max}$ (***Algorytm Coffmana–Grahama***, złożoność $O(n^2)$),
- Już $P|p_i=1,opositing\text{–}forest|C_{max}$ i $P2|p_i\in\{1,2\},prec|C_{max}$ są NP–trudne.

Algorytm Hu:

- Redukcja *out–forest* ➔ *in–forest*: odwrócenie relacji *prec*, a po uzyskaniu harmonogramu – odwrócenie go,
- *in–forest* ➔ *in–tree*: dodanie "dodatkowego korzenia" dla wszystkich drzew, a po uzyskaniu harmonogramu usunięcie go.
- Procedura Hu w skrócie: szeregowanie listowe z ograniczeniami kolejnościowymi + lista utworzona wg. nierosnącej odległości od korzenia drzewa.

Minimalizacja długości harmonogramu. Maszyny równoległe.

Procesory identyczne, zadania zależne

Zadania niepodzielne

Algorytm Hu ($P|p_i=1,in\text{–}tree|C_{max}$):

• *Poziom zadania* – liczba węzłów na drodze do korzenia.

• Zadanie jest *wolne w chwili t* – jeżeli wcześniej wykonane zostały wszystkie zadania poprzedzające je.

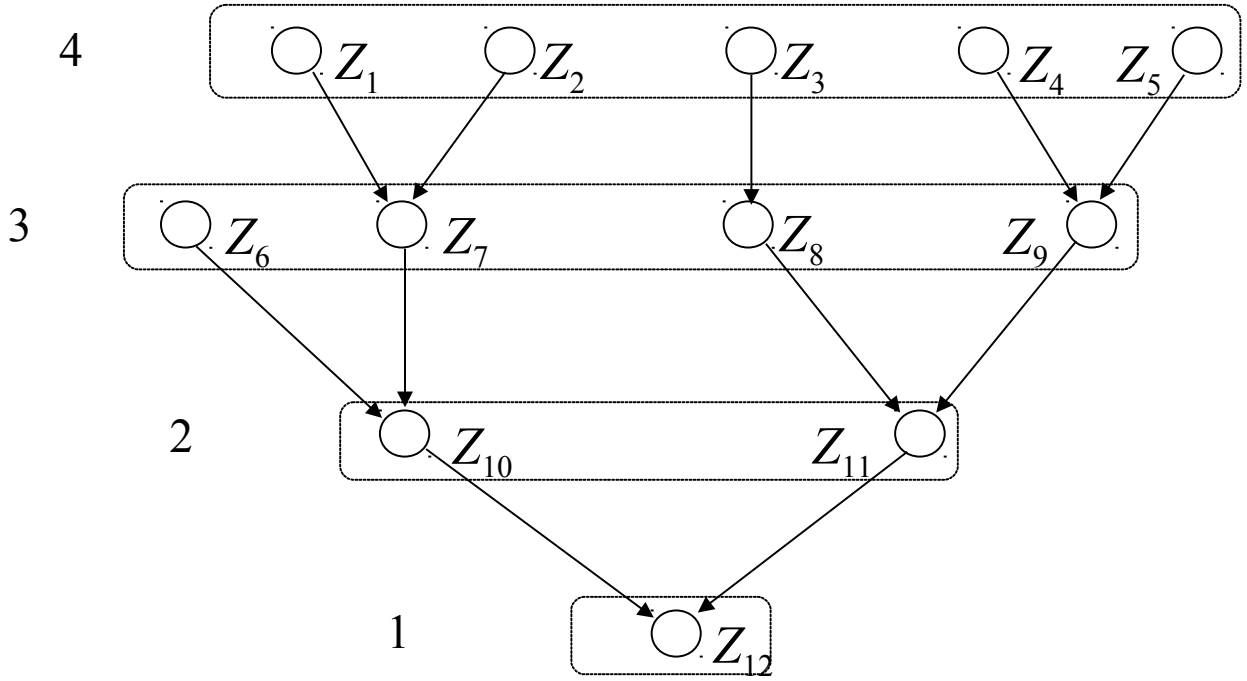
Policz poziomy zadań;
t:=1;
repeat
  Wyznacz listę L_t zadań wolnych w chwili t;
  Uporządkuj L_t według nierosnącego poziomu;
  Przypisz m (lub mniej) zadań z początku L_t do maszyn;
  Usuń przypisane zadania z grafu;
  t:=t+1;
until uszeregowano wszystkie zadania;

Minimalizacja długości harmonogramu. Maszyny równoległe.

Procesory identyczne, zadania zależne

Zadania niepodzielne

Przykład. Algorytm Hu. n=12, m=3.



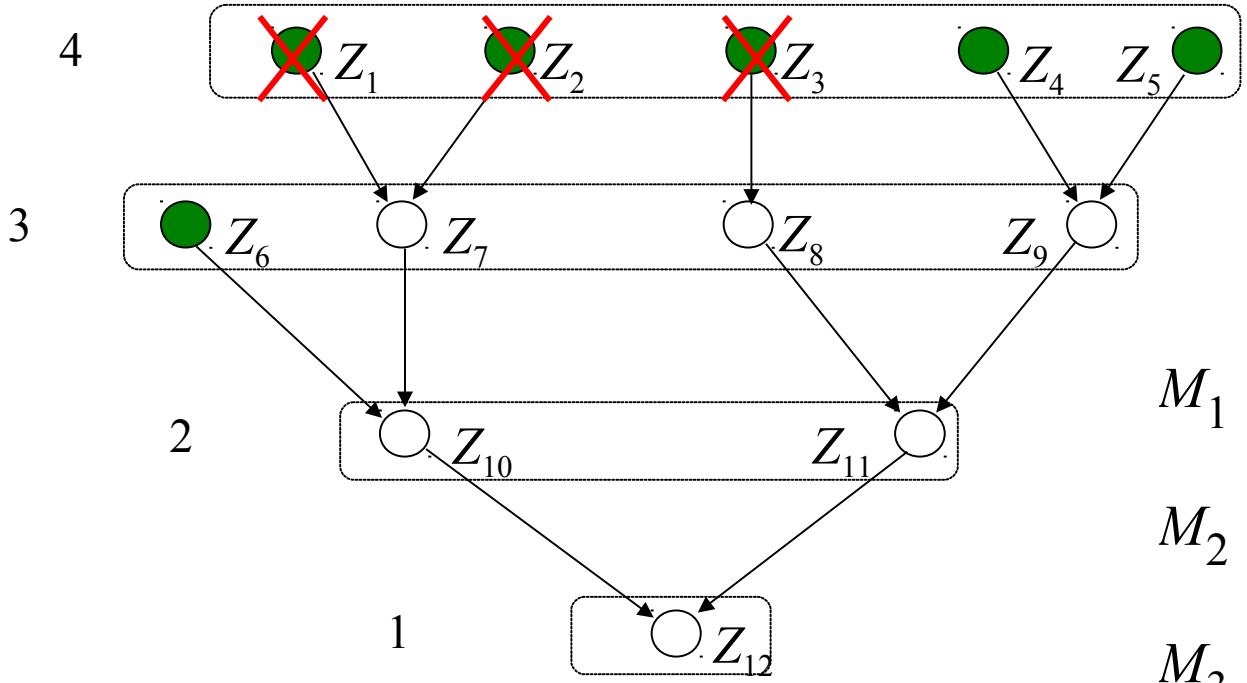
- zadanie dostępne (wolne)

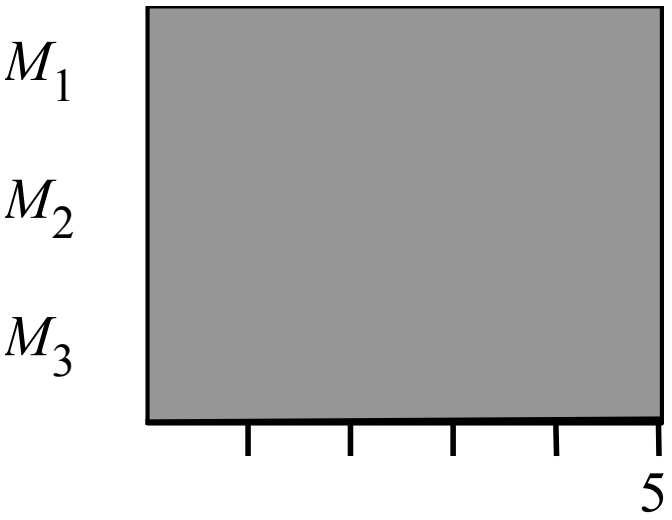
Minimalizacja długości harmonogramu. Maszyny równoległe.

Procesory identyczne, zadania zależne

Zadania niepodzielne

Przykład. Algorytm Hu. n=12, m=3.



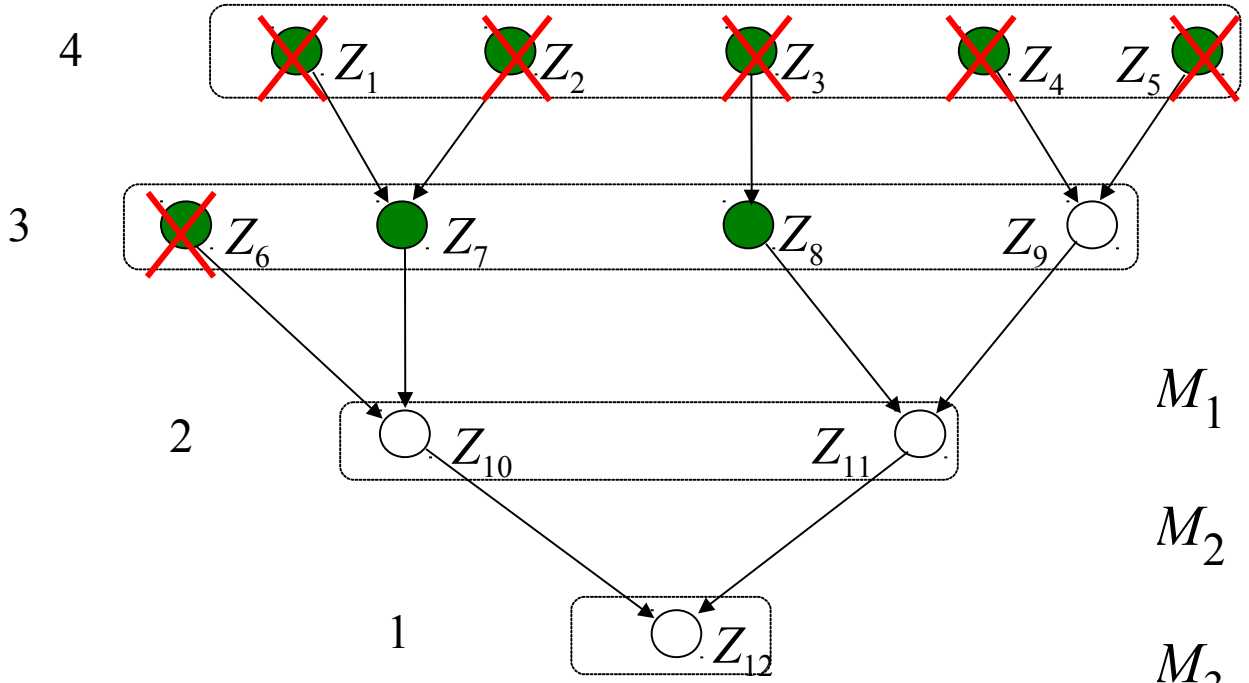


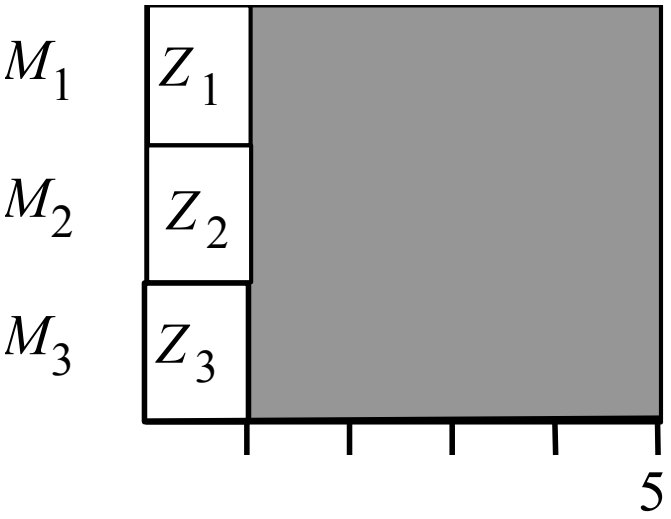
Minimalizacja długości harmonogramu. Maszyny równoległe.

Procesory identyczne, zadania zależne

Zadania niepodzielne

Przykład. Algorytm Hu. n=12, m=3.



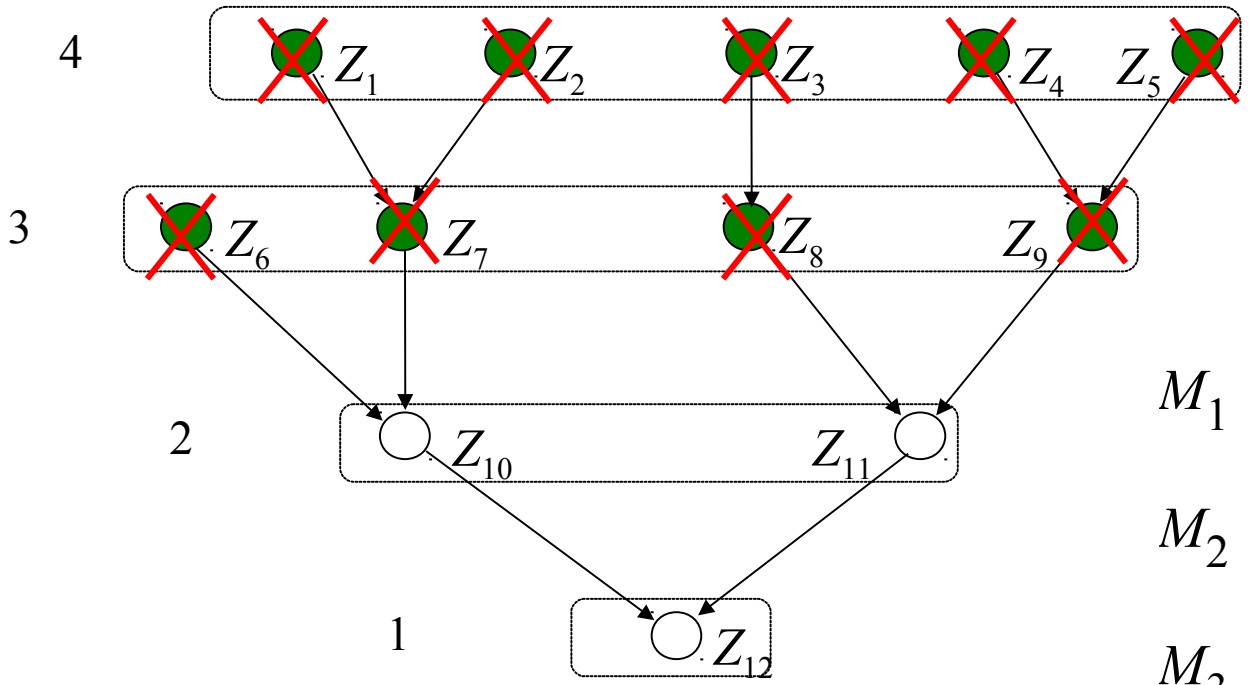


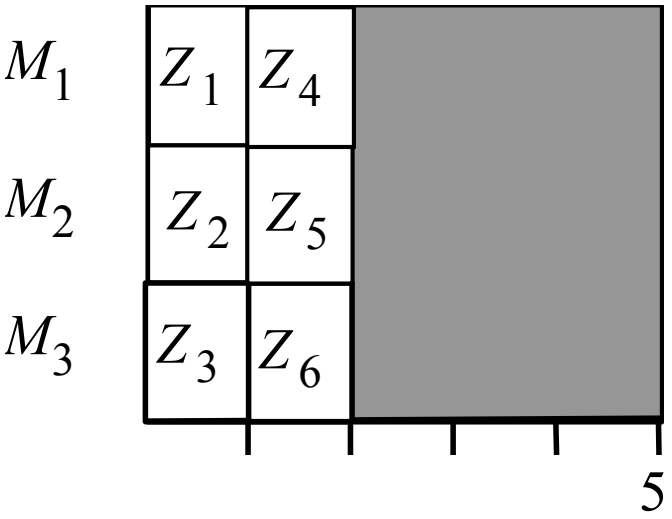
Minimalizacja długości harmonogramu. Maszyny równoległe.

Procesory identyczne, zadania zależne

Zadania niepodzielne

Przykład. Algorytm Hu. n=12, m=3.



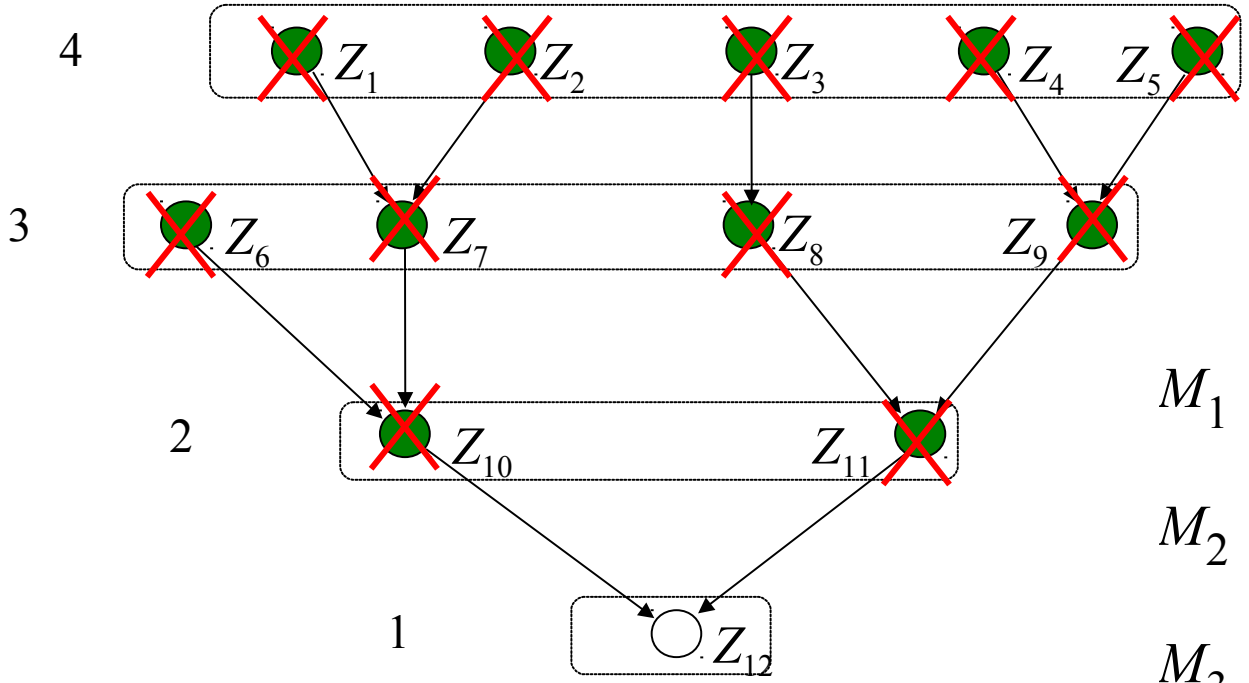


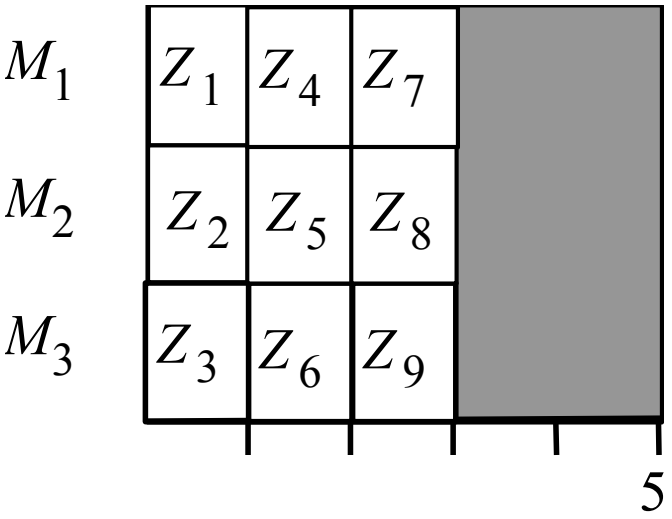
Minimalizacja długości harmonogramu. Maszyny równoległe.

Procesory identyczne, zadania zależne

Zadania niepodzielne

Przykład. Algorytm Hu. n=12, m=3.



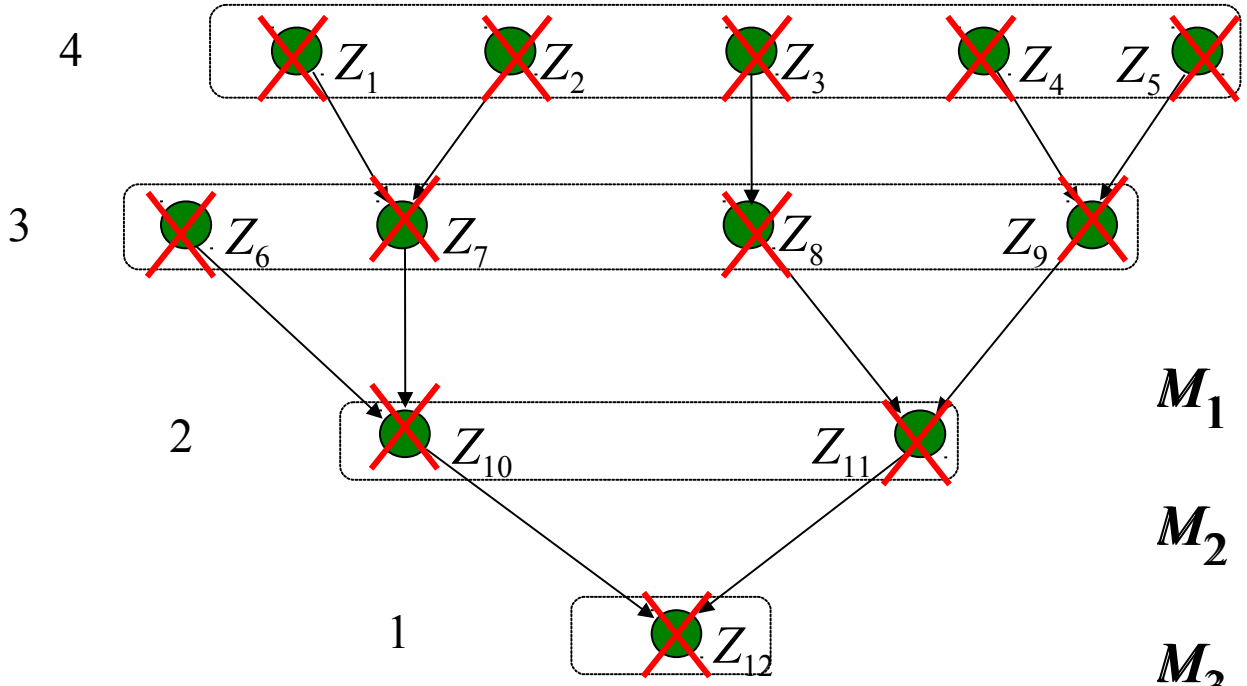


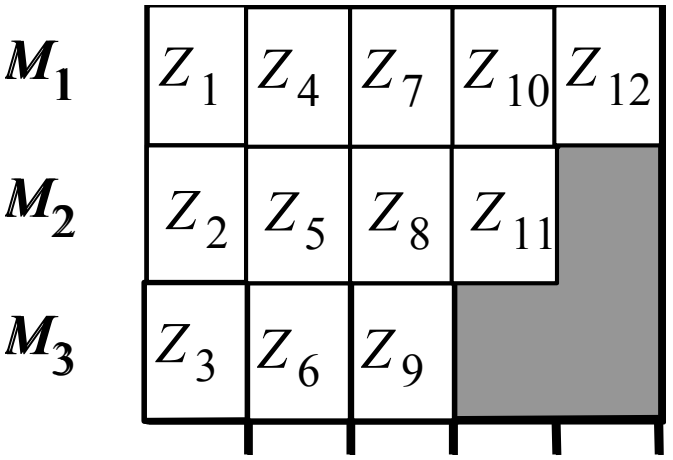
Minimalizacja długości harmonogramu. Maszyny równoległe.

Procesory identyczne, zadania zależne

Zadania niepodzielne

Przykład. Algorytm Hu. n=12, m=3.





Minimalizacja długości harmonogramu. Maszyny równoległe.

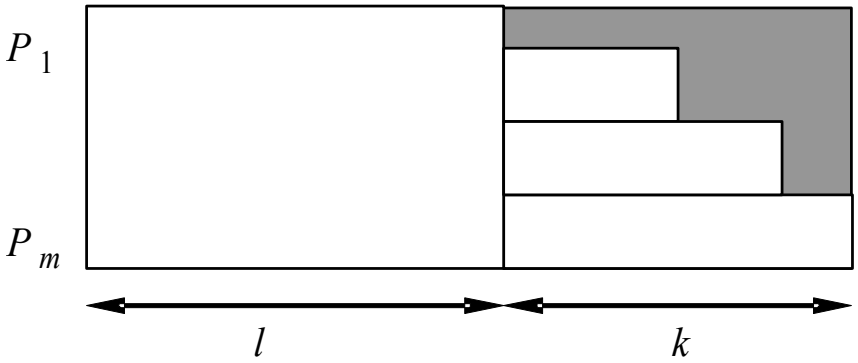
Procesory identyczne, zadania zależne

Zadania niepodzielne

Algorytm Hu ($P|p_i=1,in(out)–forest|C_{max}$)

Dowód. Porządek *in-forest,* indukcja ze względu na liczbę zadań (krok 2):

- W kolejnych krokach algorytmu liczba wolnych zadań nie wzrasta.
- Wniosek: w kolejnych chwilach liczba zajętych procesorów nie rośnie.
- Jeśli $k \in \{0,1\}$ lub $l=0$, to harmonogram jest optymalny.



- Niech $Z' \subset Z$ oznacza podzbiór zadań z poziomów $\geq k$. W chwili $l+1$ wykonano ostatnie zadanie z Z'. Wykreślając pozostałe zadania otrzymamy harmonogram Hu (czyli optymalny) dla Z'.
- Zatem w każdym harmonogramie dla Z jest zadanie z Z' wykonywane najwcześniej w chwili $l+1$, a po nim występuje jeszcze łańcuch $k–1$ zadań.
- Wniosek: nasz harmonogram jest optymalny.

Minimalizacja długości harmonogramu. Maszyny równoległe.

Procesory identyczne, zadania zależne

Zadania niepodzielne

Algorytm Coffmana–Grahama ($P2|p_i=1,prec|C_{max}$):

1. numeruj zadania przypisując im etykiety l od 1 do n,
2. szereguj listowo, przy czym kolejność na liście odpowiada malejącym etykietom zadań.

Faza 1 – numerowanie zadań;

Początkowo zadania nie mają list ani etykiet l;

for i:=1 **to** n **do begin**

A:=zbiór zadań bez etykiet l, których wszystkie bezpośrednie następniki już mają etykiety;

for each $Z \in A$ **do** przypisz do $list(Z)$ malejący ciąg etykiet l jego bezpośrednich następników;

wybierz $Z \in A$ o leksykograficznie najmniejszym $list(Z)$;

$l(Z)$:=i;

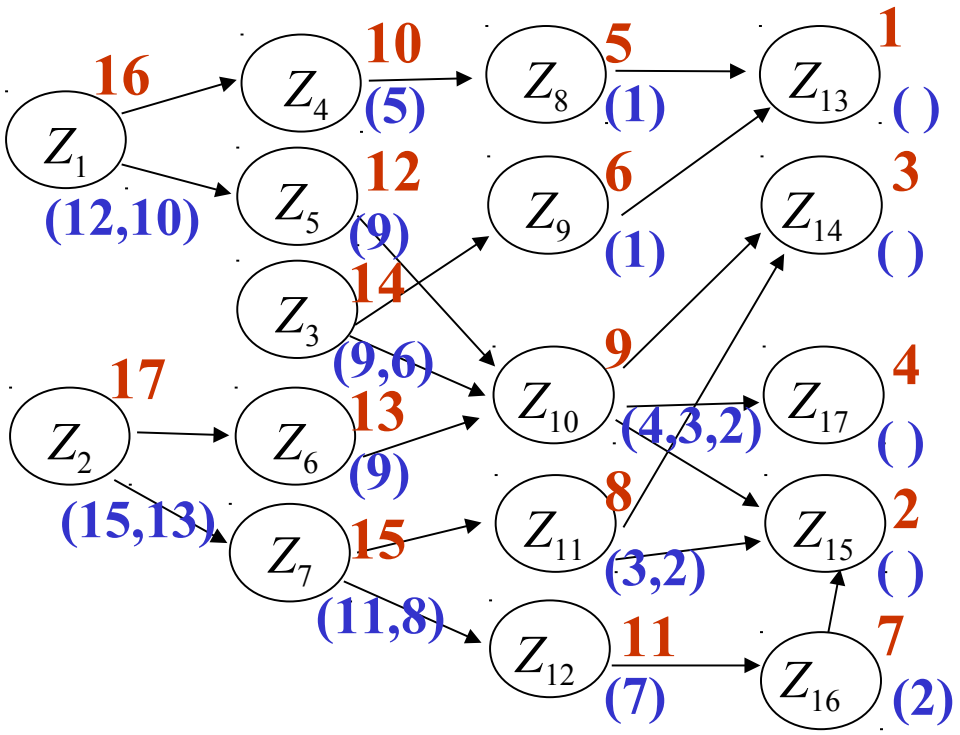
end;

Minimalizacja długości harmonogramu. Maszyny równoległe.

Procesory identyczne, zadania zależne

Zadania niepodzielne

Przykład. Algorytm Coffmana–Grahama, n=17.

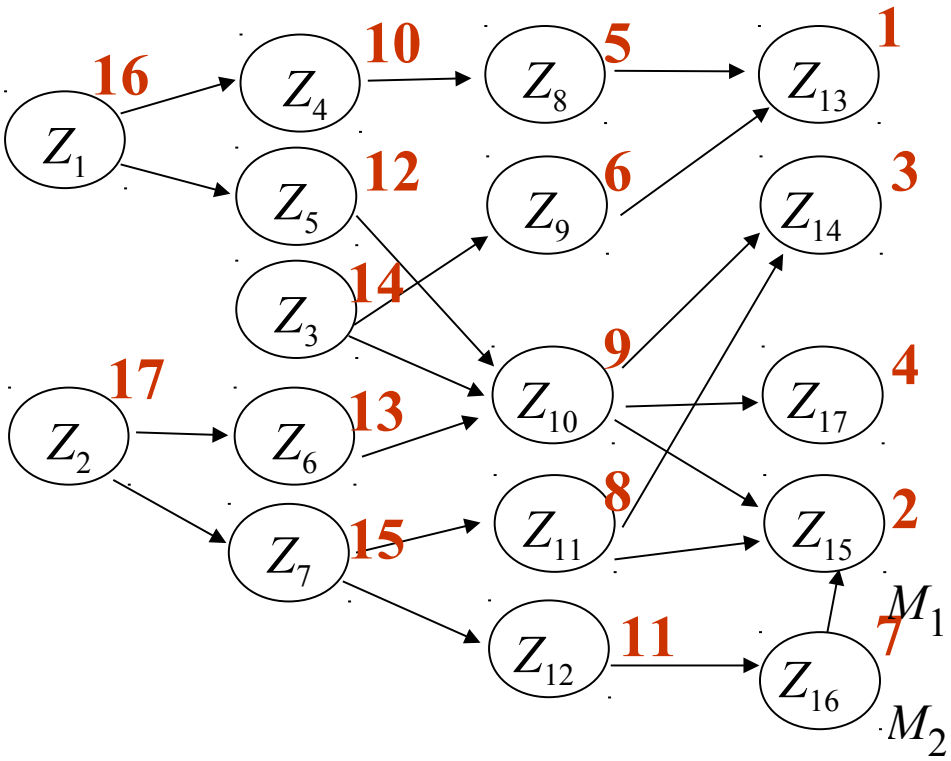


Minimalizacja długości harmonogramu. Maszyny równoległe.

Procesory identyczne, zadania zależne

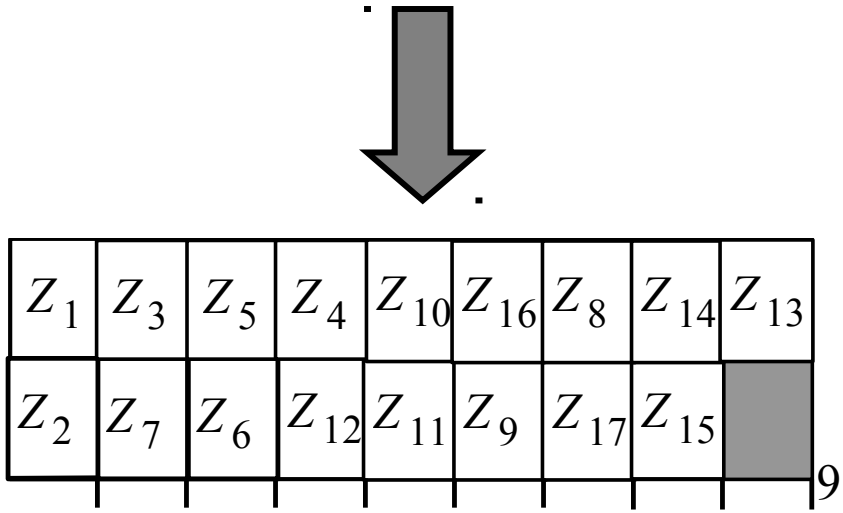
Zadania niepodzielne

Przykład. Algorytm Coffmana–Grahama, n=17.



Kolejność na liście:

$Z_2, Z_1, Z_7, Z_3, Z_6, Z_5, Z_{12}, Z_4, Z_{10},$
$Z_{11}, Z_{16}, Z_9, Z_8, Z_{17}, Z_{14}, Z_{15}, Z_{13}.$



Minimalizacja długości harmonogramu. Maszyny równoległe.

Procesory identyczne, zadania zależne

Zadania niepodzielne

Dla $P|prec|C_{max}$ można stosować heurystykę *LS*. W ogólności jest ona 2–przybliżona: $C_{max}(\mathrm{LS}) \leq (2 - m^{-1}) C_{max}^*$.

Dowód. Już był ...

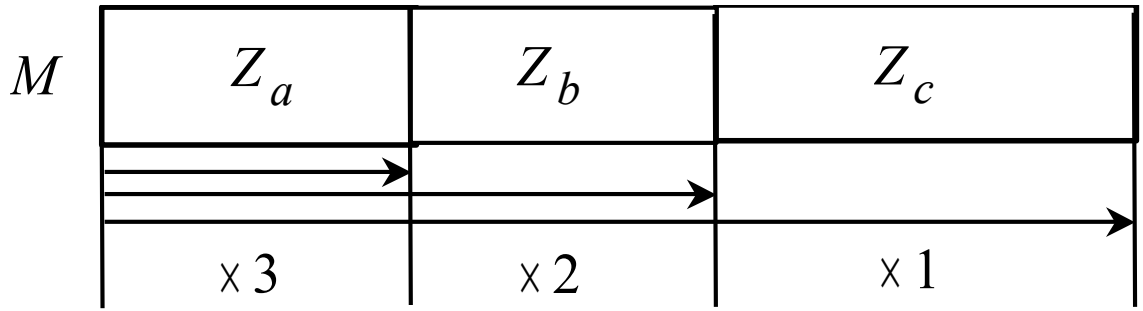
Kolejność zadań na liście (priorytety) ustala się różnymi metodami. Mogą się pojawiać anomalie polegające na wydłużaniu się harmonogramu przy:

- wzroście liczby maszyn,
- zmniejszaniu czasu wykonania zadań,
- zmniejszaniu relacji *prec,*
- zmianie kolejności na liście.

Minimalizacja średniego czasu przepływu na maszynach równoległych

Procesory identyczne, zadania niezależne

Własność: zadanie Z_j na maszynie M_i umieszczone na k–tej pozycji od końca dodaje do kryterium ΣC_j wartość kp_j (lub kp_{ij} dla maszyn $R|$).



Wnioski.

- długość pierwszego zadania jest mnożona przez największy współczynnik, dla kolejnych zadań współczynniki maleją,
- minimalizując ΣC_j powinniśmy umieszczać krótkie zadania na początku (są mnożone przez największe współczynniki),
- optymalne uszeregowanie jest zgodne z regułą ***SPT*** (*Shortest Processing Times*) – zadania na maszynach są podejmowane w kolejności niemalejących czasów wykonania,
- **ale jak znaleźć optymalne przypisanie zadań do procesorów?**

Minimalizacja średniego czasu przepływu na maszynach równoległych

Procesory identyczne, zadania niezależne

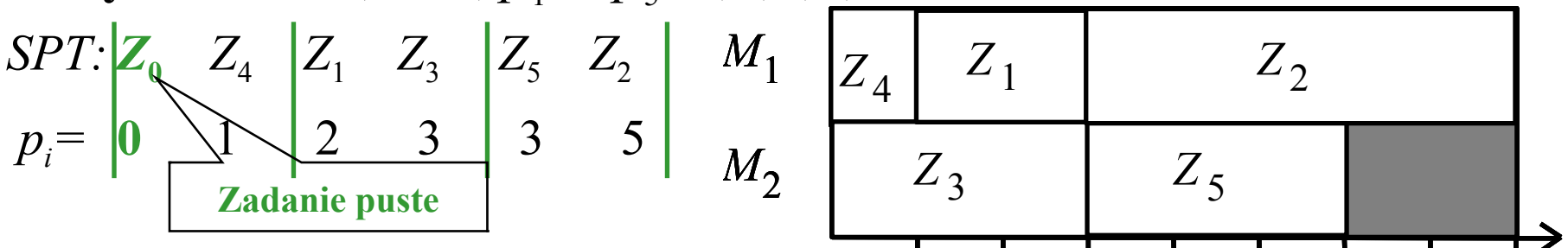
Zadania podzielnie i niepodzielne

Przypadki $P||\Sigma C_i$ i podzielnych $P|pmtn|\Sigma C_i$ można rozpatrywać razem (optymalny harmonogram podzielny nie musi dzielić zadań).

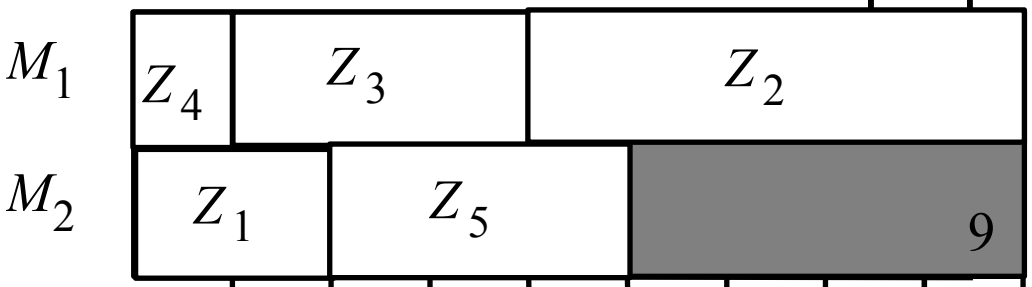
Algorytm optymalny $O(n \log n)$:

1. Przyjmij, że liczba zadań dzieli się przez m (ew. wprowadź zadania puste),
2. Uporządkuj je według ***SPT***,
3. Przypisuj kolejne m–tki zadań w sposób dowolny do różnych maszyn.

Przykład. m=2, n=5, $p_1,...,p_5$=2,5,3,1,3.



Można i tak:



$\Sigma C_j^*=21$

Minimalizacja średniego czasu przepływu na maszynach równoległych

Procesory identyczne, zadania niezależne

Zadania podzielnie i niepodzielne

Przypadki $P||\Sigma C_i$ i podzielnych $P|pmtn|\Sigma C_i$ można rozpatrywać razem (optymalny harmonogram podzielny nie musi dzielić zadań).

Algorytm optymalny $O(n \log n)$:

1. Przyjmij, że liczba zadań dzieli się przez m (ew. wprowadź zadania puste),
2. Uporządkuj je według ***SPT***,
3. Przypisuj kolejne m–tki zadań w sposób dowolny do różnych maszyn.

Dowód (przypadek niepodzielny).

Lemat. Dane są dwa ciągi liczb $a_1,...,a_n$ i $b_1,...,b_n$. W jaki sposób należy je popermutować, by iloczyn skalarny $a_{\pi(1)}b_{\tau(1)}+a_{\pi(2)}b_{\tau(2)}+...+a_{\pi(n-1)}b_{\tau(n-1)}+a_{\pi(n)}b_{\tau(n)}$ był możliwie:

- największy? – oba posortować niemalejąco,
- najmniejszy? – jeden posortować niemalejąco, a drugi nierosnąco.

Przykład. Mamy ciągi (3,2,4,6,1) i (5,7,8,1,2).
(1,2,3,4,6) i (1,2,5,7,8) ➔ 1+4+15+28+48=96
(1,2,3,4,6) i (8,7,5,2,1) ➔ 8+14+15+8+6=51

Minimalizacja średniego czasu przepływu na maszynach równoległych

Procesory identyczne, zadania niezależne

Zadania podzielnie i niepodzielne

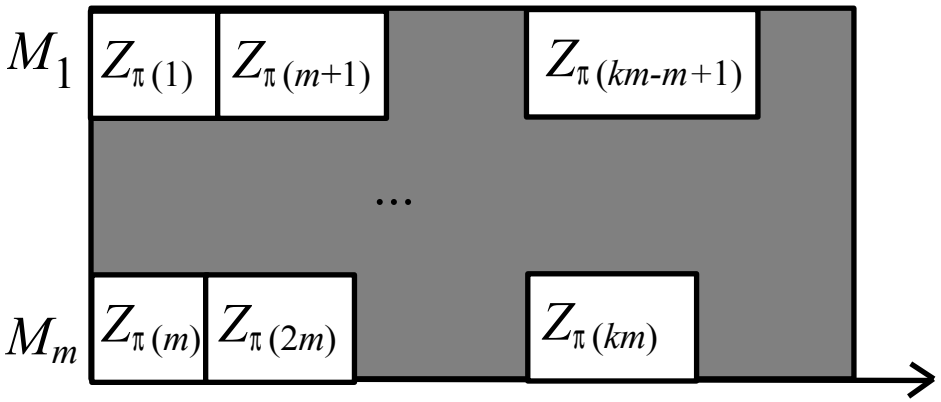
Przypadki $P||\Sigma C_i$ i podzielnych $P|pmtn|\Sigma C_i$ można rozpatrywać razem (optymalny harmonogram podzielny nie musi dzielić zadań).

Algorytm optymalny $O(n \log n)$:

1. Przyjmij, że liczba zadań dzieli się przez m (ew. wprowadź zadania puste),
2. Uporządkuj je według ***SPT***,
3. Przypisuj kolejne m–tki zadań w sposób dowolny do różnych maszyn.

Dowód (przypadek niepodzielny). Rozważamy uszeregowanie optymalne. Można przyjąć, że na każdej maszynie jest k zadań (ew. zadania puste).

$$\Sigma C_i = kp_{\pi(1)}+\ldots+kp_{\pi(m)} + \\ +(k-1)p_{\pi(m+1)}+\ldots+(k-1)p_{\pi(2m)} + \\ +1p_{\pi(km-m+1)}+\ldots+1p_{\pi(km)}$$



Przestawienie zadań według SPT nie zwiększy ΣC_i.

Minimalizacja średniego czasu przepływu na maszynach równoległych

Procesory identyczne, zadania niezależne

Zadania niepodzielne

Już wersja ważona ***P2*||Σw_jC_j** (a także równoważna ***P2*|*pmtn*|Σw_jC_j**) jest NP–trudna.

Dowód. Jak w ***P2*||C_{max}**. Redukcja *PP* ➔ ***P2*||Σw_iC_i**: bierzemy n zadań o $p_j = w_j = a_j$ (j=1,...,n), dwie maszyny. Wyznacz liczbę $C(a_1,...,a_n)$ taką, że istnieje uszeregowanie o $\Sigma w_jC_j \le C(a_1,...,a_n) \Leftrightarrow C_{max}^* = \Sigma_{i=1,...,n}\, a_i/2$ (ćwiczenie).

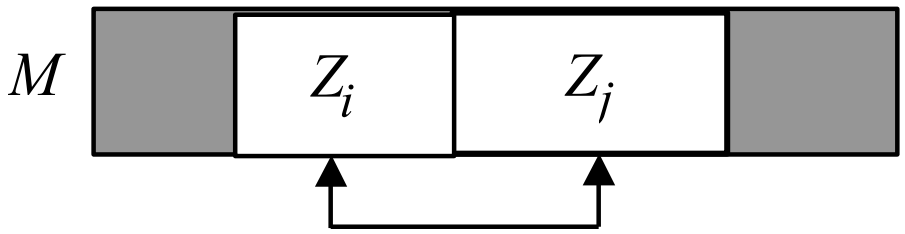
Wariant ważony jednomaszynowy (**1||Σw_jC_j**) można rozwiązać w czasie ***O*(*n*log *n*)** szeregując według ***reguły Smitha*** (uogólnione SPT):

- ustaw zadania w kolejności niemalejącego p_j/w_j.

Dowód. Rozważamy przyrost kryterium po zamianie dwóch kolejnych zadań.

$w_jp_j + w_i(p_i+p_j) - w_ip_i - w_j(p_i+p_j) =$

$= w_ip_j - w_jp_i \le 0 \Leftrightarrow p_j/w_j \le p_i/w_i$



Naruszenie reguły Smitha sprawi, że wartość Σw_iC_i zmaleje po zamianie.

Minimalizacja średniego czasu przepływu na maszynach równoległych

Procesory identyczne, zadania niezależne

Zadania niepodzielne

Próbą pogodzenia kryteriów C_{max} i ΣC_i jest ***algorytm RPT***:

1. Zastosuj szeregowanie LPT.
2. Na każdej maszynie posortuj zadania według SPT.

Dokładność: $1 \leq \Sigma C_i\,(\mathrm{RPT}) / \Sigma C_i^* \leq m$ (zwykle jest lepsza)

Procesory identyczne, zadania zależne

- Już $1|prec|\Sigma C_i$, $P2|prec,p_j=1|\Sigma C_i$, $P2|chains|\Sigma C_i$ i $P2|chains,pmtn|\Sigma C_i$ są NP–trudne.
- Wielomianowy algorytm dla $P|out\text{–}tree,p_j=1|\Sigma C_i$ (adaptacja algorytmu Hu).
- W wersji ważonej nawet przypadek jednomaszynowy z zadaniami jednostkowymi $1|prec,p_j=1|\Sigma w_i C_i$ jest NP–trudny.

Minimalizacja średniego czasu przepływu na maszynach równoległych

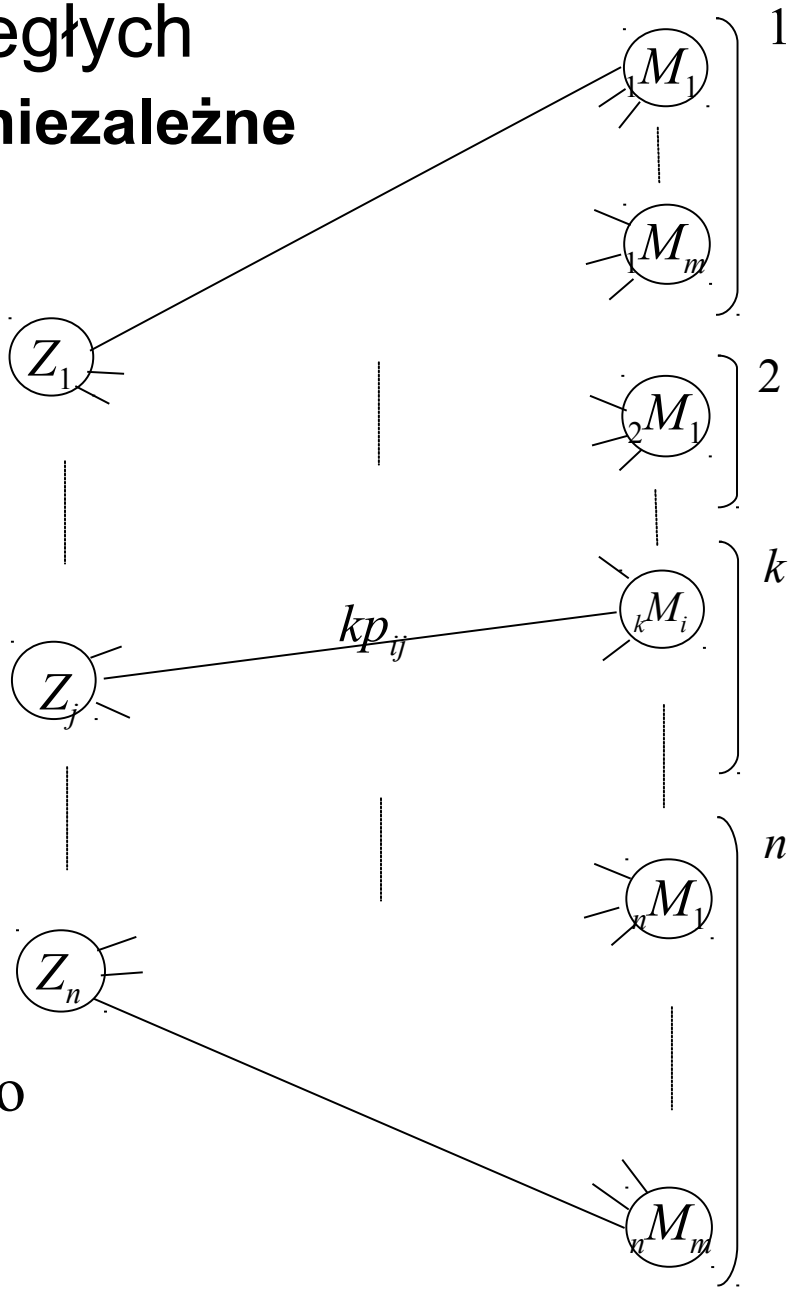
Procesory dowolne, zadania niezależne

Algorytm $O(n^3)$ dla $R||\Sigma C_i$

bazuje na problemie skojarzeń w grafach. Graf dwudzielny z krawędziami obciążonymi wagami:

- W partycji V_1 zadania $Z_1,...,Z_n$.
- W partycji V_2 każdy procesor n razy: ${}_kM_i$, i=1...m, k=1...n.
- Krawędź z Z_j do ${}_kM_i$ ma wagę kp_{ij} – oznacza ona zadanie Z_j na maszynie M_i, pozycja k–ta od końca.

Szukamy najlżejszego skojarzenia o n krawędziach. Przedstawia ono szukany harmonogram.



Minimalizacja maksymalnego opóźnienia na maszynach równoległych

Własności:

• Aby opóźnienie $L_i=C_i-d_i$ zadania Z_i w harmonogramie było określone, zadania muszą być wyposażone w oczekiwane terminy zakończenia d_i.

• Spóźnienie zadania $T_i=\max\{L_i,0\}$ nie bierze pod uwagę wykonania się zadań przed terminem.

• Wniosek: $T_{\max}=\max\{L_{\max},0\}$. Dlatego kryterium $T_{\max}$ nie rozważamy osobno – harmonogram $L_{\max}$-optymalny jest też $T_{\max}$-optymalny.

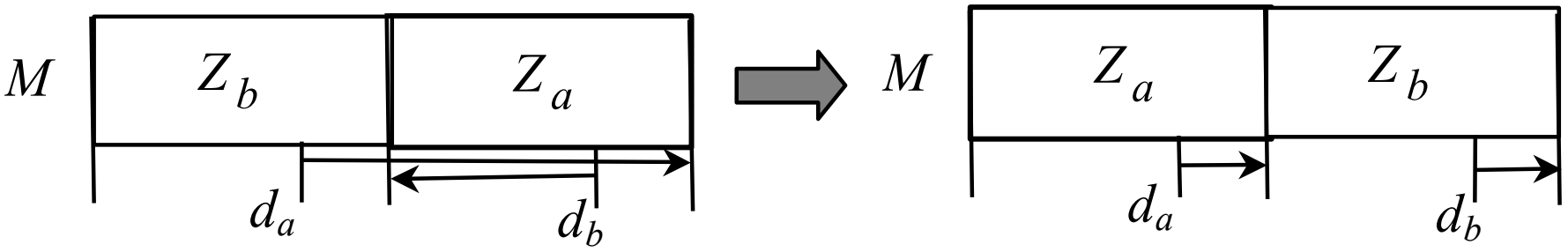
• $L_{\max}^*$ to najmniejsza liczba x, taka że przedłużenie terminów $d_i'=d_i+x$ pozwala nie spóźnić się z żadnym zadaniem (spełnione są nowe deadline-y $C_i\leq d_i'$ zadań Z_i).

• Wniosek: minimalizacja $L_{\max}$ i szukanie (jeśli istnieje) harmonogramu respektującego nieprzekraczalne deadline-y (tj. pytania ...|...|$L_{\max}$ i ...|....$C_i\leq d_i$|–) to problemy „jednakowo trudne”.

Minimalizacja maksymalnego opóźnienia na maszynach równoległych

Własności:

• kryterium L_{max} jest uogólnieniem C_{max}, zagadnienia NP–trudne dla C_{max} pozostaną takie w przypadku L_{max},



• mając do wykonania wiele prac z różnymi oczekiwanymi terminami zakończenia spóźnimy się „najmniej” zaczynając zawsze od „najpilniejszej” pracy,

• to samo innymi słowy: w różnych wariantach stosujemy ***regułę EDD*** (*Earliest Due Date*) – wybieraj zadania Z_j w kolejności niemalejących oczekiwanych terminów zakończenia d_j,

• problem zadań niepodzielnych na jednej maszynie (**$1||L_{max}$**) rozwiązuje właśnie szeregowanie według ***EDD***.

Minimalizacja maksymalnego opóźnienia na maszynach równoległych

Procesory identyczne, zadania niezależne

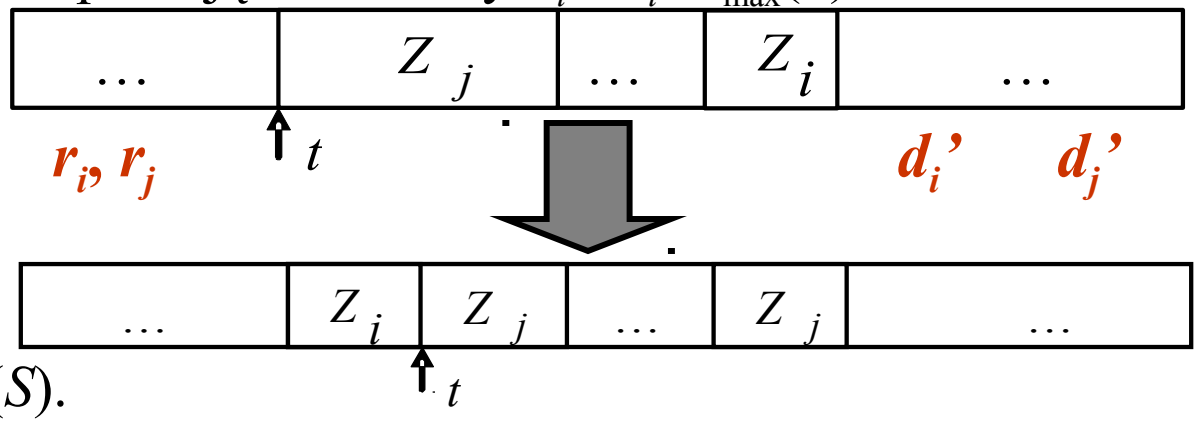
Zadania podzielne

Jedna maszyna: ***Algorytm Liu*** $\boldsymbol{O(n^2)}$**,** oparty na regule *EDD*, działający nawet przy $1|r_i,pmtn|L_{\max}$:

1. Spośród dostępnych zadań przydziel maszynę temu, które ma najmniejszy wymagany termin zakończenia,
2. Jeśli zadanie zostało zakończone, lub przybyło nowe – wróć do 1 (w drugim przypadku przerywamy zadanie).

Dowód. S_{Liu} – harmonogram uzyskany algorytmem Liu. S – inny harmonogram. Zadania Z_i respektują deadline-y $d_i'=d_i+L_{\max}(S)$.

Zaczynając od t=0 przekształcimy S w S_{Liu} nie naruszając d_i'.



- W S_{Liu} w chwili t uruchomiono Z_i

Wniosek: $L_{\max}(S_{\text{Liu}}) \le L_{\max}(S)$.

Minimalizacja maksymalnego opóźnienia na maszynach równoległych

Procesory identyczne, zadania niezależne

Zadania podzielne

Jedna maszyna: ***Algorytm Liu*** $O(n^2)$, oparty na regule *EDD*, działający nawet przy $1|r_i,pmtn|L_{max}$:

1. Spośród dostępnych zadań przydziel maszynę temu, które ma najmniejszy wymagany termin zakończenia,
2. Jeśli zadanie zostało zakończone, lub przybyło nowe – wróć do 1 (w drugim przypadku przerywamy zadanie).

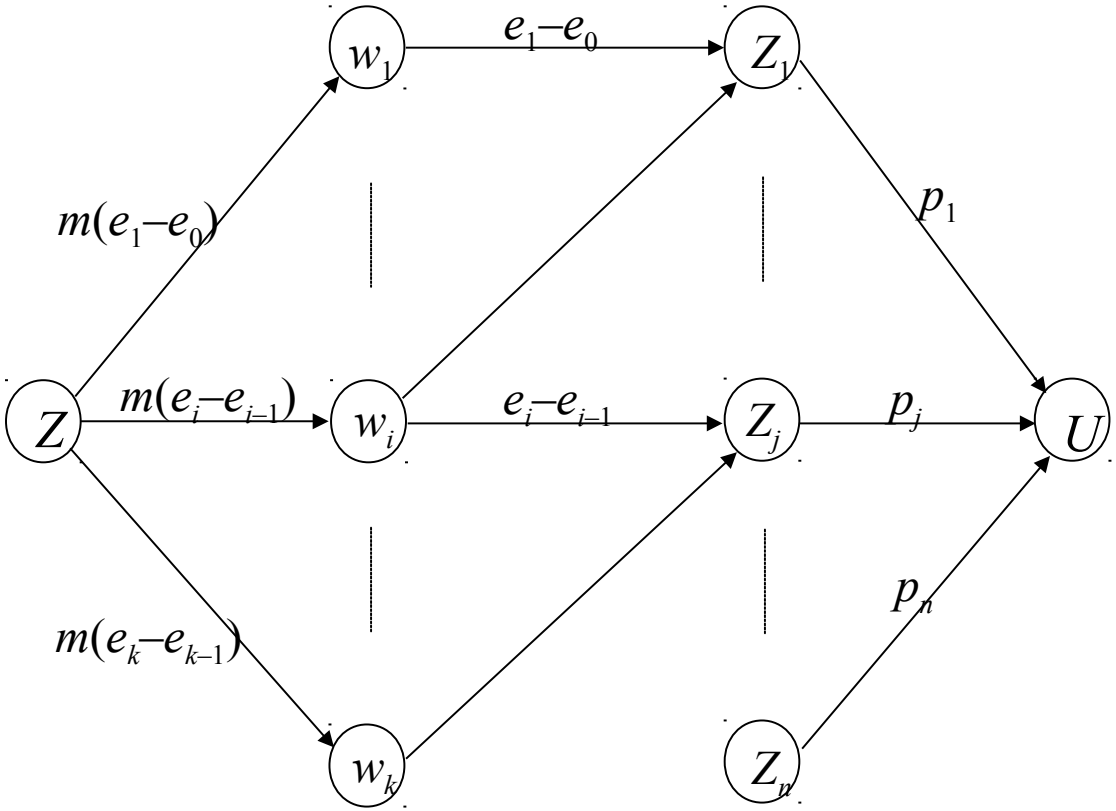
Więcej maszyn ($P|r_i,pmtn|L_{max}$). Również algorytm wielomianowy: korzystamy z podprocedury rozwiązującej wersję z "twardymi" terminami zakończenia $P|r_i,C_i \leq d_i,pmtn|-$, szukamy optymalnego L_{max} metodą połowienia.

Minimalizacja maksymalnego opóźnienia na maszynach równoległych

P|r_i,C_i≤d_i,pmtn| – sprowadzamy do problemu przepływu. Ustawiamy wszystkie r_i i d_i w ciąg $e_0<e_1<...<e_k$.

Tworzymy sieć:

- Ze źródła wychodzi k łuków o przepustowości $m(e_i–e_{i–1})$ do wierzchołków w_i, i=1,...,k.
- Do ujścia wchodzą łuki o przepustowości p_i z wierzchołków Z_i, i=1,...,n.
- Między w_i a Z_j biegnie łuk o przepustowości $e_i–e_{i–1}$, jeżeli zachodzi $[e_{i–1},e_i]\subseteq[r_j,d_j]$.



Uszeregowanie istnieje ⇔ istnieje przepływ o objętości $\Sigma_{i=1,...,n}\, p_i$ (można rozdysponować moce obliczeniowe procesorów do zadań w odpowiednich odcinkach czasu, tak by wykonać wszystkie).

Minimalizacja maksymalnego opóźnienia na maszynach równoległych

Zadania niezależne

Zadania niepodzielne

Niektóre przypadki NP–trudne: $P2||L_{max}$, $1|r_j|L_{max}$.

Przypadki wielomianowe:

- dla zadań jednostkowych $P|p_j=1,r_j|L_{max}$.
- podobnie dla maszyn jednorodnych $Q|p_j=1|L_{max}$ (redukcja do programowania liniowego),
- dla jednej maszyny rozwiązanie optymalne $1||L_{max}$ uzyskamy szeregując według *EDD* (to już było ...).

Minimalizacja maksymalnego opóźnienia na maszynach równoległych

Zadania zależne

Zadania podzielne

Dla jednej maszyny $1|pmtn,prec,r_j|L_{max}$ zmodyfikowany algorytm Liu $O(n^2)$:

1. określ ***zmodyfikowane terminy zakończenia*** zadań:

$$d_j^* = \min\{d_j, \min_i\{d_i : Z_j \bullet Z_i\}\}$$

2. szereguj według EDD dla nowych d_j^* z *wywłaszczaniem* zadania, gdy pojawia się nowe, wolne, z mniejszym zmodyfikowanym terminem zakończenia,
3. powtarzaj 2 aż do uszeregowania wszystkich zadań.

- Inne przypadki wielomianowe:
  $P|pmtn,in\text{–}tree|L_{max}$, $Q2|pmtn,prec,r_j|L_{max}$.
- Stosuje się też algorytmy pseudowielomianowe.

Minimalizacja maksymalnego opóźnienia na maszynach równoległych

Zadania zależne

Zadania niepodzielne

- Już $P|p_j=1,out\text{–}tree|L_{max}$ jest NP–trudny.
- istnieje wielomianowy algorytm dla $P2|prec,p_j=1|L_{max}$.
- $P|p_j=1,in\text{–}tree|L_{max}$ rozwiązuje ***algorytm Bruckera*** $O(n\log n)$:

$next(j)$ = bezpośredni następnik zadania Z_j.

1. wylicz *zmodyfikowane terminy zakończenia* zadań:
dla korzenia $d_{root}^*=1-d_{root}$ i dla pozostałych $d_k^*=\max\{1+d_{next(k)}^*,1-d_k\}$,
2. szereguj zadania dostępne podobnie jak w algorytmie Hu, ale remisy rozstrzygaj wybierając zadania według nierosnących zmodyfikowanych terminów zakończenia, a nie według poziomów w drzewie.

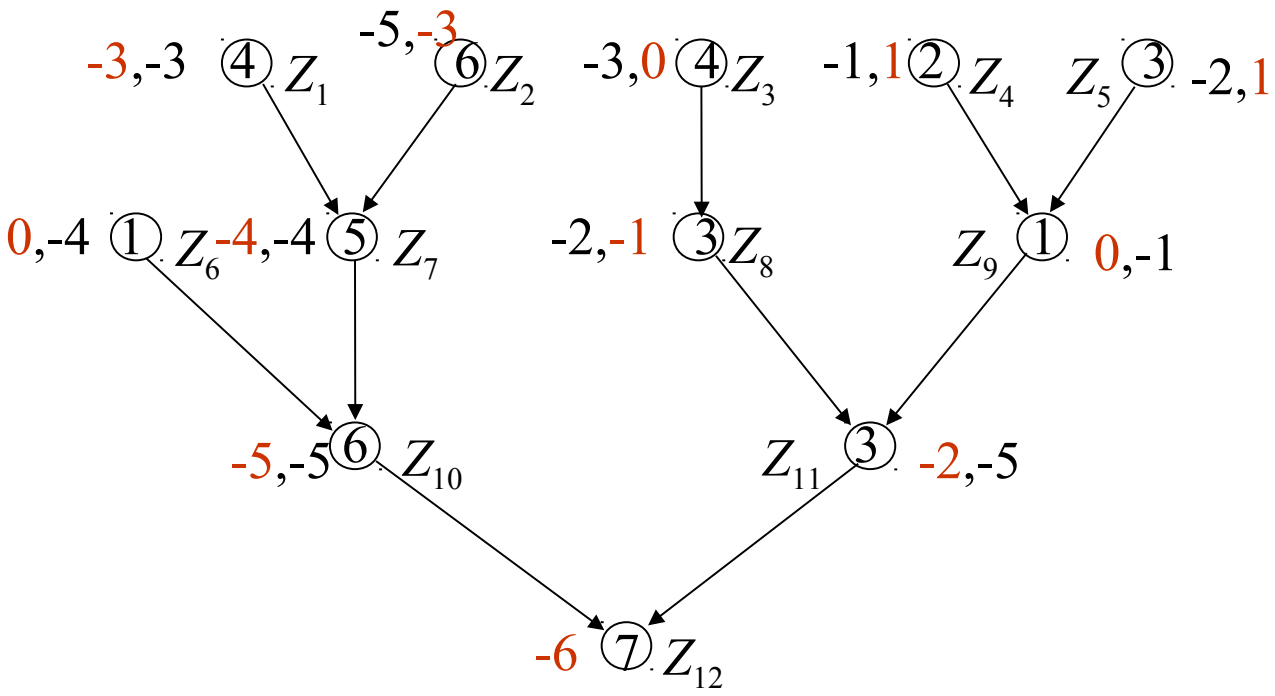
Czyli znowu szeregowanie listowe z inną metodą wyznaczania kolejności na liście.

Minimalizacja maksymalnego opóźnienia na maszynach równoległych

Zadania zależne

Zadania niepodzielne

Przykład. Algorytm Bruckera, n=12, m=3, terminy zakończenia w kółkach.

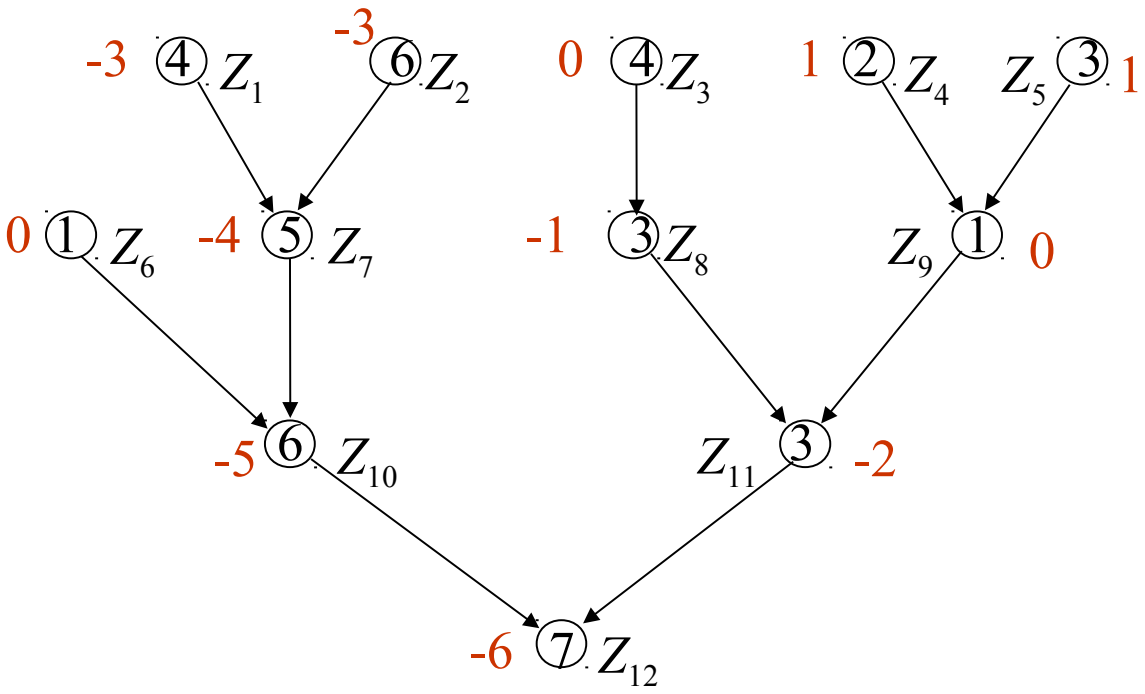


Minimalizacja maksymalnego opóźnienia na maszynach równoległych

Zadania zależne

Zadania niepodzielne

Przykład. Algorytm Bruckera, n=12, m=3, terminy zakończenia w kółkach.

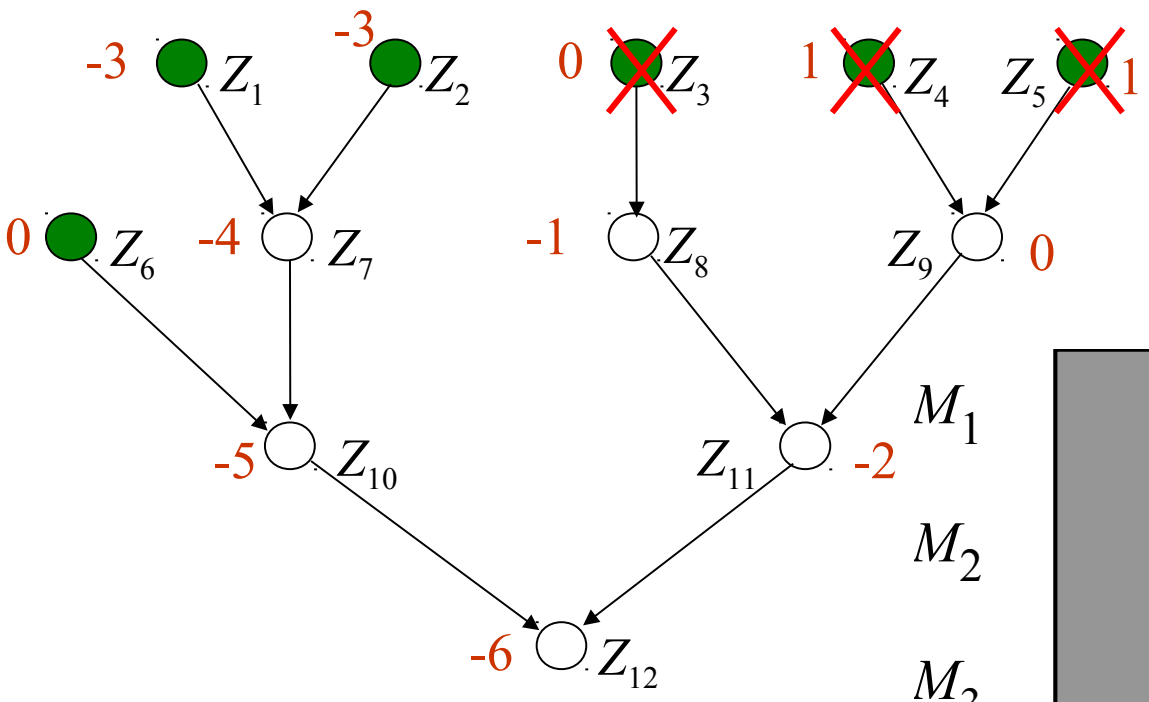


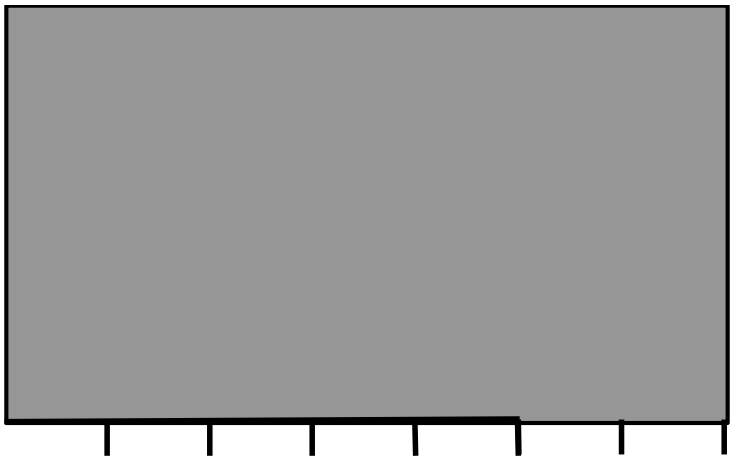
Minimalizacja maksymalnego opóźnienia na maszynach równoległych

Zadania zależne

Zadania niepodzielne

Przykład. Algorytm Bruckera, n=12, m=3, terminy zakończenia w kółkach.



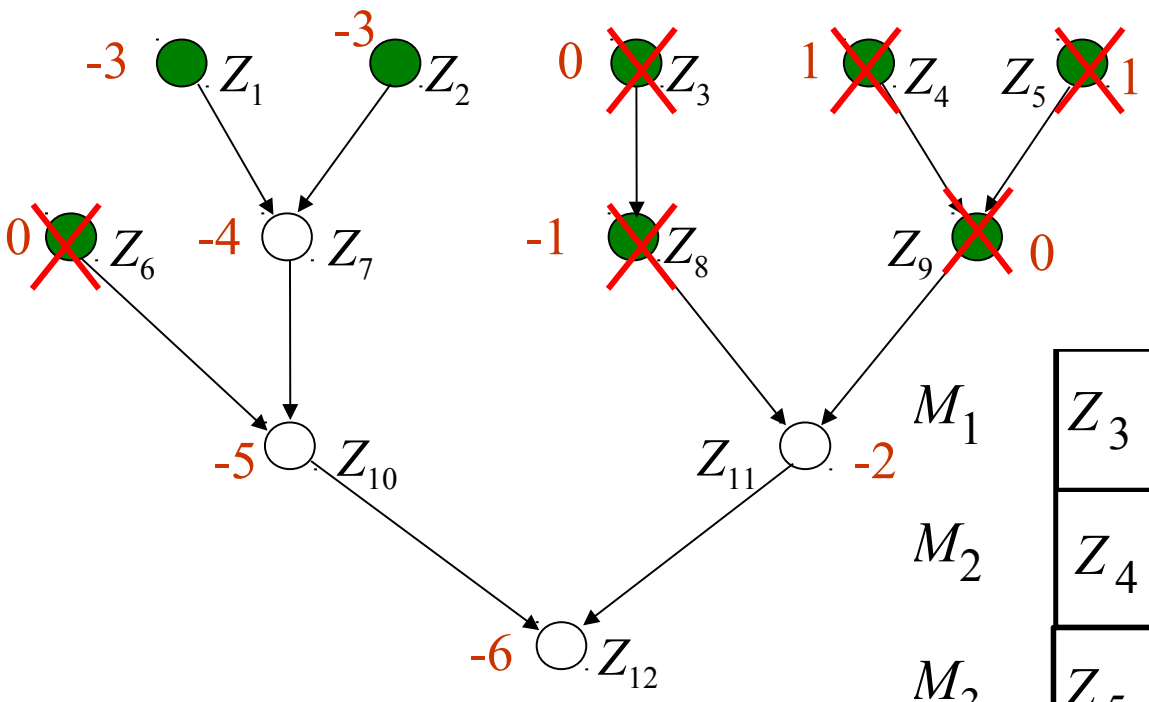


Minimalizacja maksymalnego opóźnienia na maszynach równoległych

Zadania zależne

Zadania niepodzielne

Przykład. Algorytm Bruckera, n=12, m=3, terminy zakończenia w kółkach.

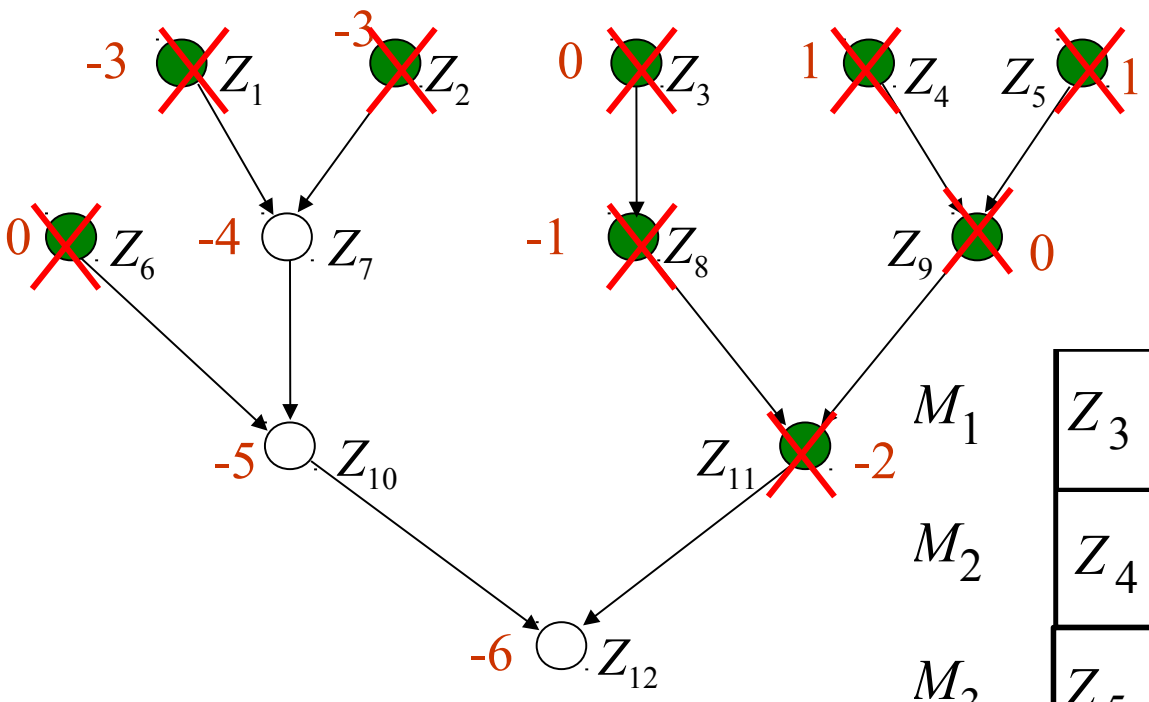


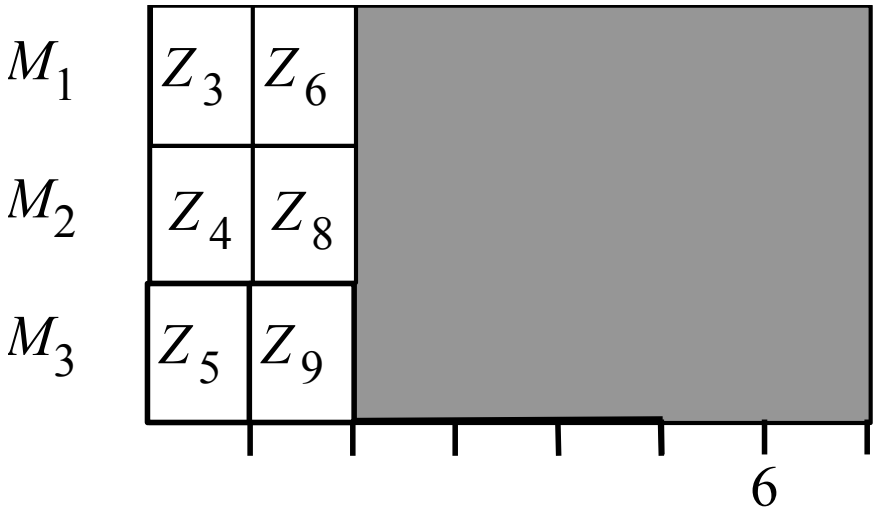
Minimalizacja maksymalnego opóźnienia na maszynach równoległych

Zadania zależne

Zadania niepodzielne

Przykład. Algorytm Bruckera, n=12, m=3, terminy zakończenia w kółkach.



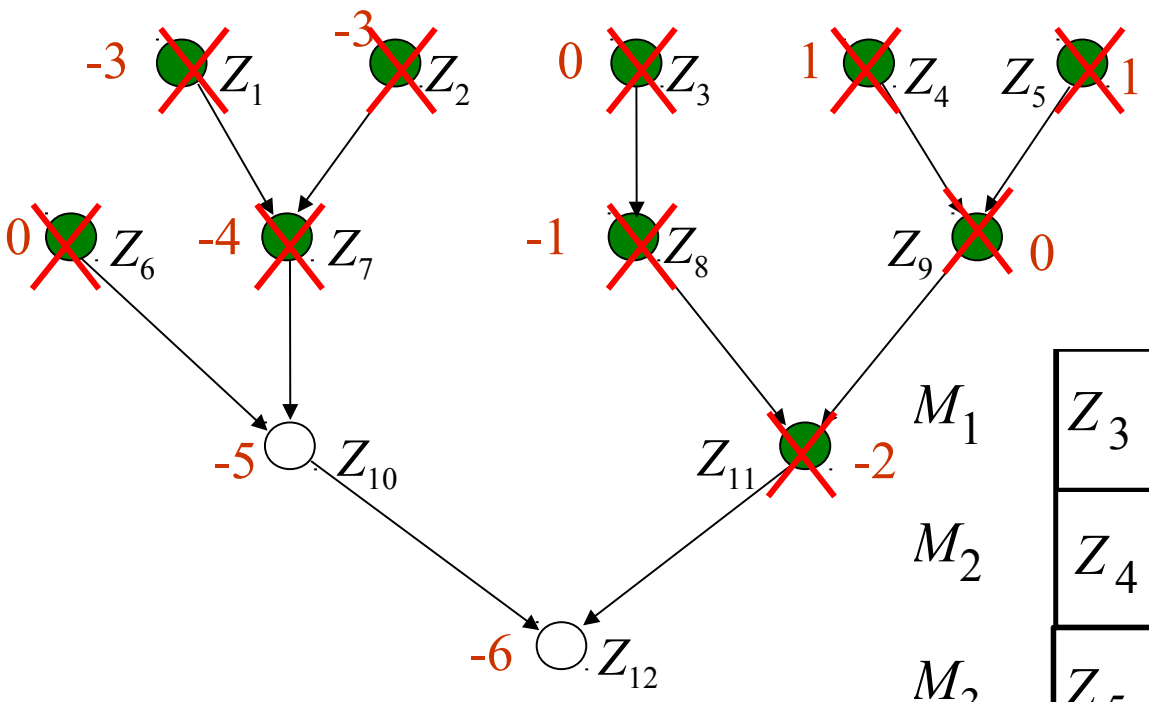


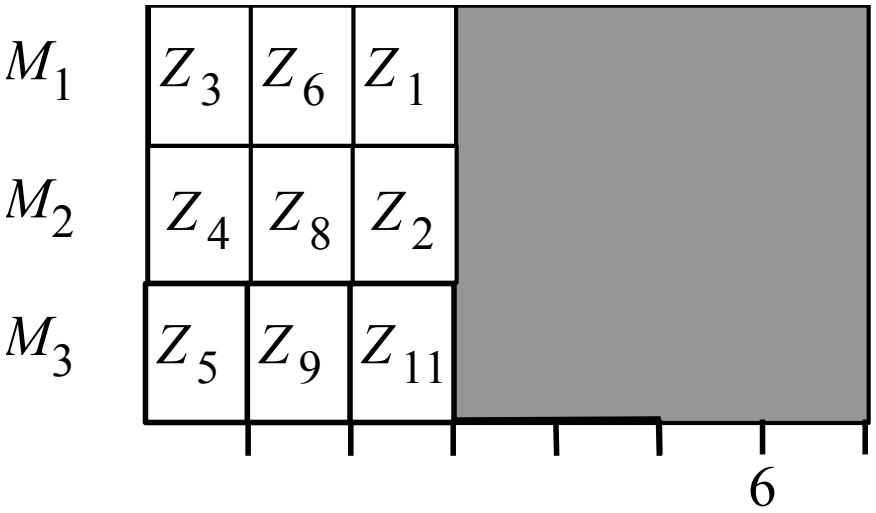
Minimalizacja maksymalnego opóźnienia na maszynach równoległych

Zadania zależne

Zadania niepodzielne

Przykład. Algorytm Bruckera, n=12, m=3, terminy zakończenia w kółkach.



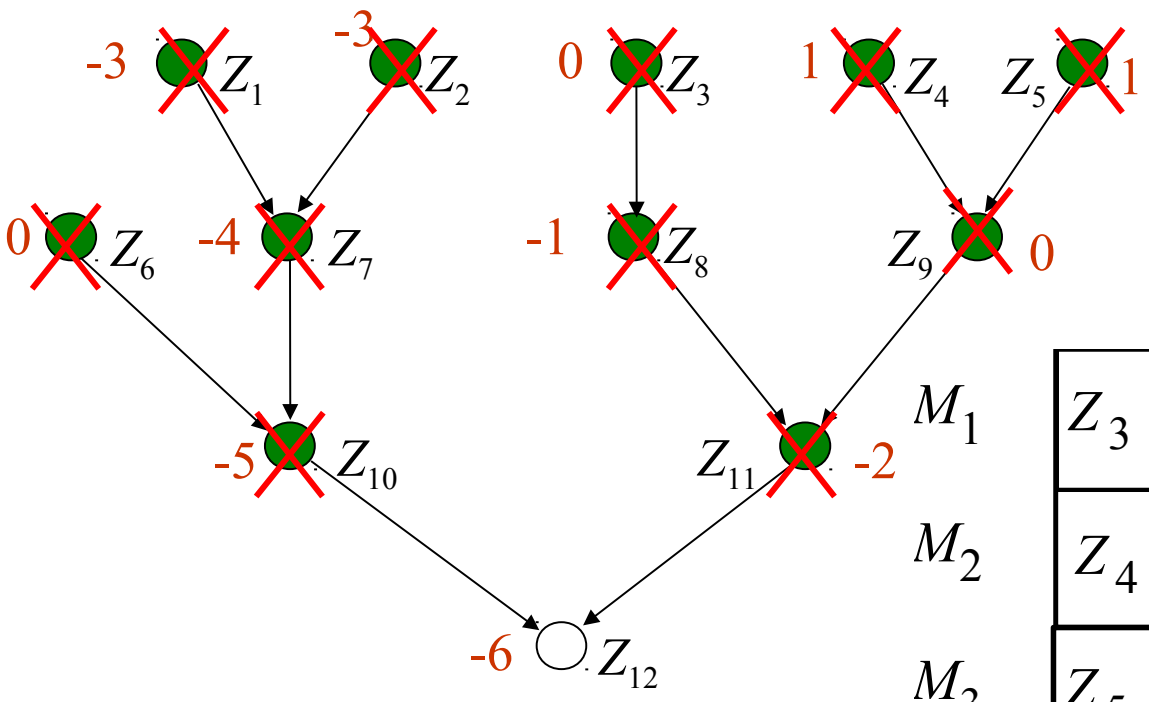


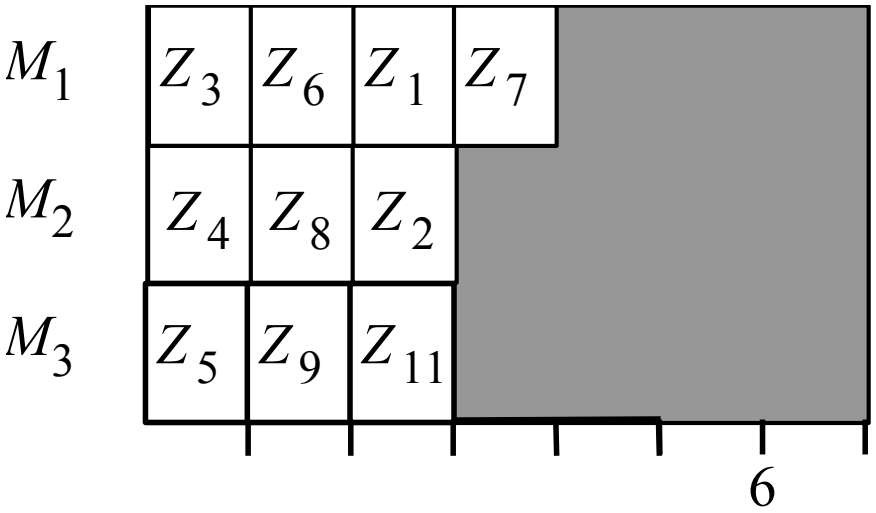
Minimalizacja maksymalnego opóźnienia na maszynach równoległych

Zadania zależne

Zadania niepodzielne

Przykład. Algorytm Bruckera, n=12, m=3, terminy zakończenia w kółkach.



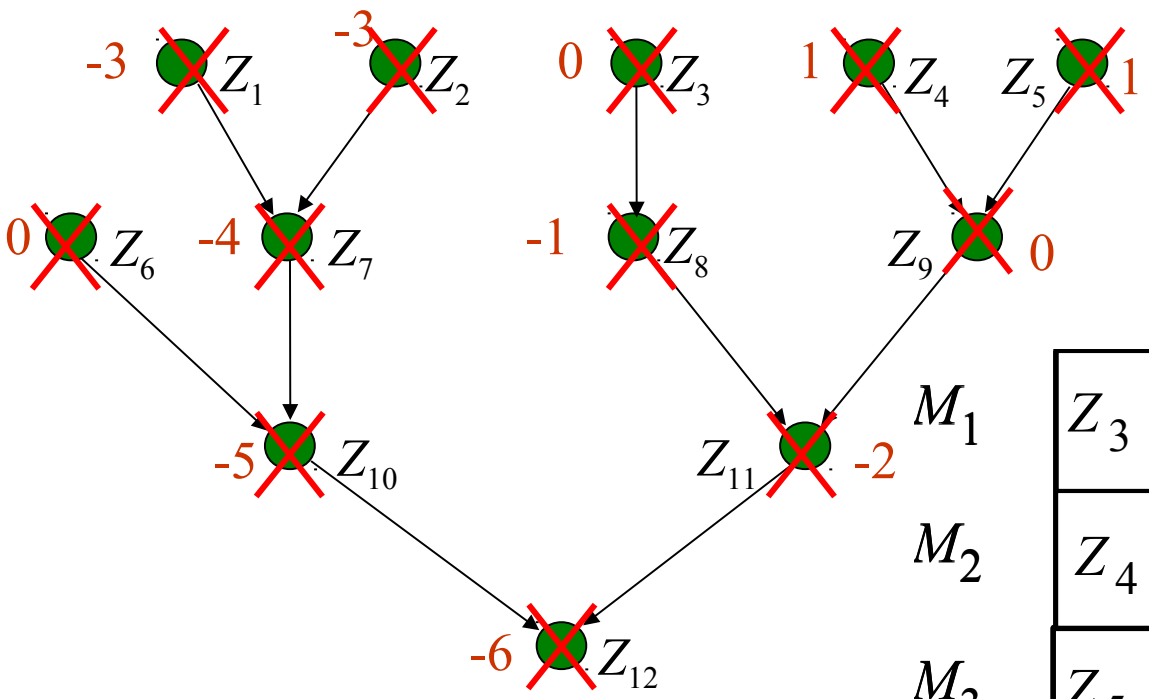


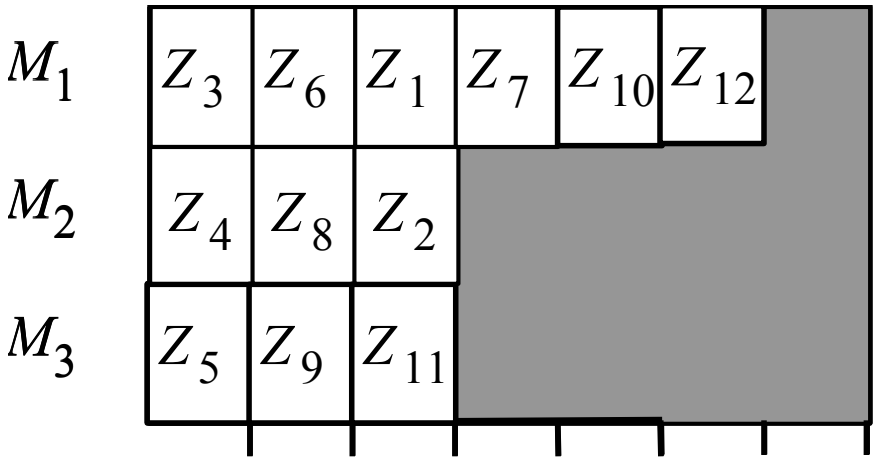
Minimalizacja maksymalnego opóźnienia na maszynach równoległych

Zadania zależne

Zadania niepodzielne

Przykład. Algorytm Bruckera, n=12, m=3, terminy zakończenia w kółkach.





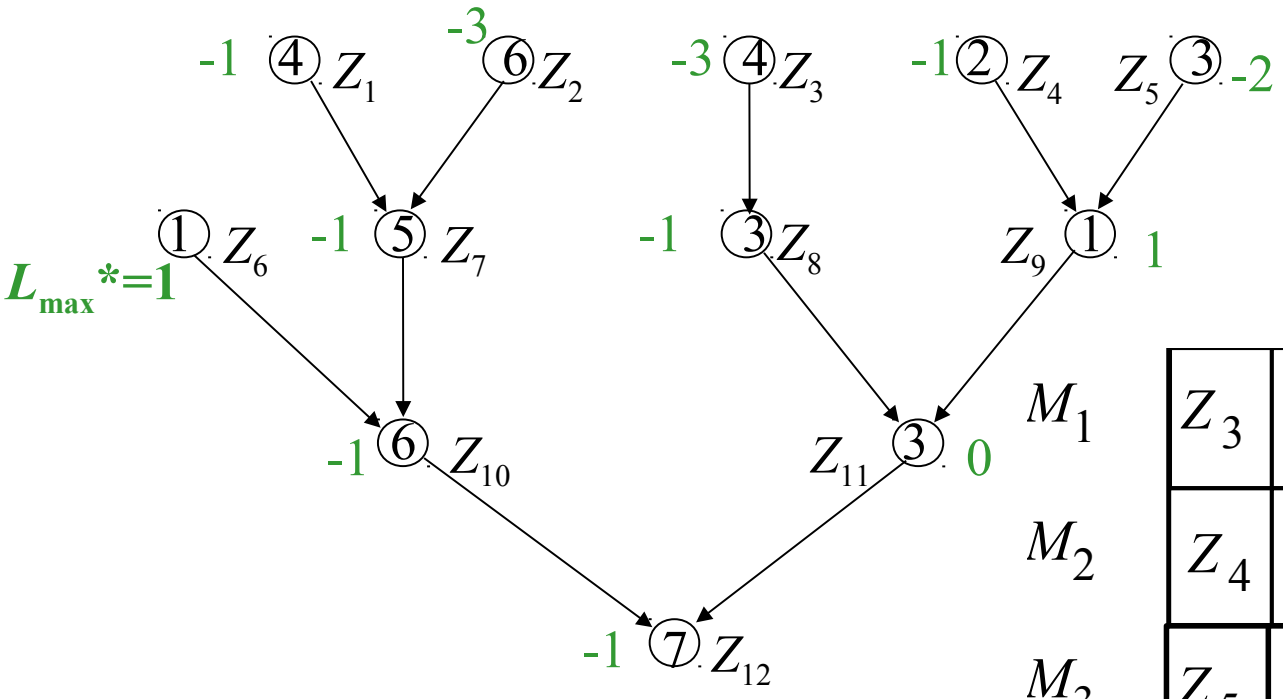
Minimalizacja maksymalnego opóźnienia na maszynach równoległych

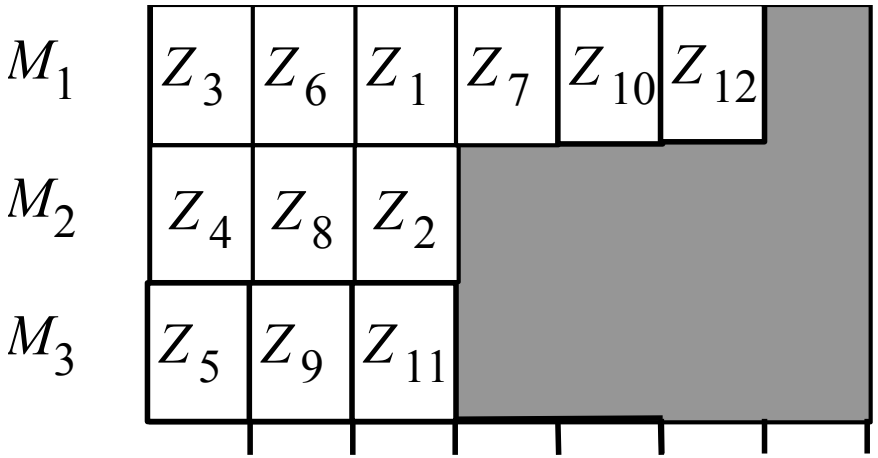
Zadania zależne

Zadania niepodzielne

Przykład. Algorytm Bruckera, n=12, m=3, terminy zakończenia w kółkach.

Opóźnienia:





Minimalizacja liczby spóźnionych zadań na jednej maszynie

Zadania niezależne i niepodzielne

Oczywiście nawet $P2||\Sigma U_i$ i $P2||\Sigma T_i$ są NP-trudne.

Dowód. Analogiczny jak dla $P2||C_{max}$.

Dalej skoncentrujemy się na przypadku jednoprocesorowym.
Minimalizacja liczby spóźnionych zadań $1||\Sigma U_i$ jest wielomianowa

Algorytm Hodgsona $O(n \log n)$:

Uporządkuj zadania według EDD: $Z_{\pi(1)}, Z_{\pi(2)}, \ldots, Z_{\pi(n)}$;
$A := \emptyset$;
for $i := 1$ **to** n **do begin**
  $A := A \cup \{Z_{\pi(i)}\}$;
  if $\Sigma_{Zj \in A} p_j > d_{\pi(i)}$ **then** usuń z A najdłuższe zadanie;
end;

A to najliczniejszy podzbior zbioru $Z'=\{Z_{\pi(1)},\ldots,Z_{\pi(i)}\}$ możliwy do uszeregowania bez spóźnień (jak? - EDD).

Minimalizacja liczby spóźnionych zadań na jednej maszynie

Zadania niezależne i niepodzielne

Oczywiście nawet $P2||\Sigma U_i$ i $P2||\Sigma T_i$ są NP-trudne.

Dowód. Analogiczny jak dla $P2||C_{\max}$.

Dalej skoncentrujemy się na przypadku jednoprocesorowym.
Minimalizacja liczby spóźnionych zadań $1||\Sigma U_i$ jest wielomianowa

Algorytm Hodgsona $O(n \log n)$:

Uporządkuj zadania według EDD: $Z_{\pi(1)}, Z_{\pi(2)}, \ldots, Z_{\pi(n)}$;
$A := \emptyset$;
for $i := 1$ **to** n **do begin**
$A := A \cup \{Z_{\pi(i)}\}$;
if $\Sigma_{Zj \in A} p_j > d_{\pi(i)}$ **then** usuń z A najdłuższe zadanie;
end;

Dla k=0,...,$|A|$ najkrótszym (w sensie sumy długości zadań) k–elementowym podzbiorem zbioru Z', możliwym do uszeregowania bez spóźnień jest A po skreśleniu jego $|A|–k$ najdłuższych zadań.

Minimalizacja liczby spóźnionych zadań na jednej maszynie

Zadania niezależne i niepodzielne

Oczywiście nawet $P2||\Sigma U_i$ i $P2||\Sigma T_i$ są NP-trudne.

Dowód. Analogiczny jak dla $P2||C_{\max}$.

Dalej skoncentrujemy się na przypadku jednoprocesorowym.

Minimalizacja liczby spóźnionych zadań $1||\Sigma U_i$ jest wielomianowa

Algorytm Hodgsona $O(n \log n)$:

Uporządkuj zadania według EDD: $Z_{\pi(1)}, Z_{\pi(2)}, \ldots, Z_{\pi(n)}$;

$A := \emptyset$;

for $i := 1$ **to** n **do begin**

$A := A \cup \{Z_{\pi(i)}\}$;

if $\Sigma_{Zj \in A} p_j > d_{\pi(i)}$ **then** usuń z A najdłuższe zadanie;

end;

A – najliczniejszy możliwy podzbiór zadań, które można wykonać bez spóźnienia.

Szereguj najpierw A według EDD, po nich pozostałe zadania w dowolnym porządku;

Minimalizacja całkowitego spóźnienia $1||\Sigma T_i$ jest pseudowielomianowa.

Minimalizacja liczby spóźnionych zadań na jednej maszynie

Zadania niezależne i niepodzielne

• Wersja z wagami $1||\Sigma w_i U_i$ jest NP–trudna jako uogólnienie problemu plecakowego i podobnie jak dla problemu plecakowego znany jest algorytm pseudowielomianowy.

• Podobny $1||\Sigma w_i T_i$ jest też NP–trudny.

• Zagadnienia optymalizacyjne upraszczają się dla zadań jednostkowych: $P|p_j=1|\Sigma w_i U_i$ i $P|p_j=1|\Sigma w_i T_i$ są wielomianowe np. prosta redukcja do najtańszego skojarzenia w grafie dwudzielnym.

Minimalizacja liczby spóźnionych zadań na jednej maszynie

Zadania zależne i niepodzielne

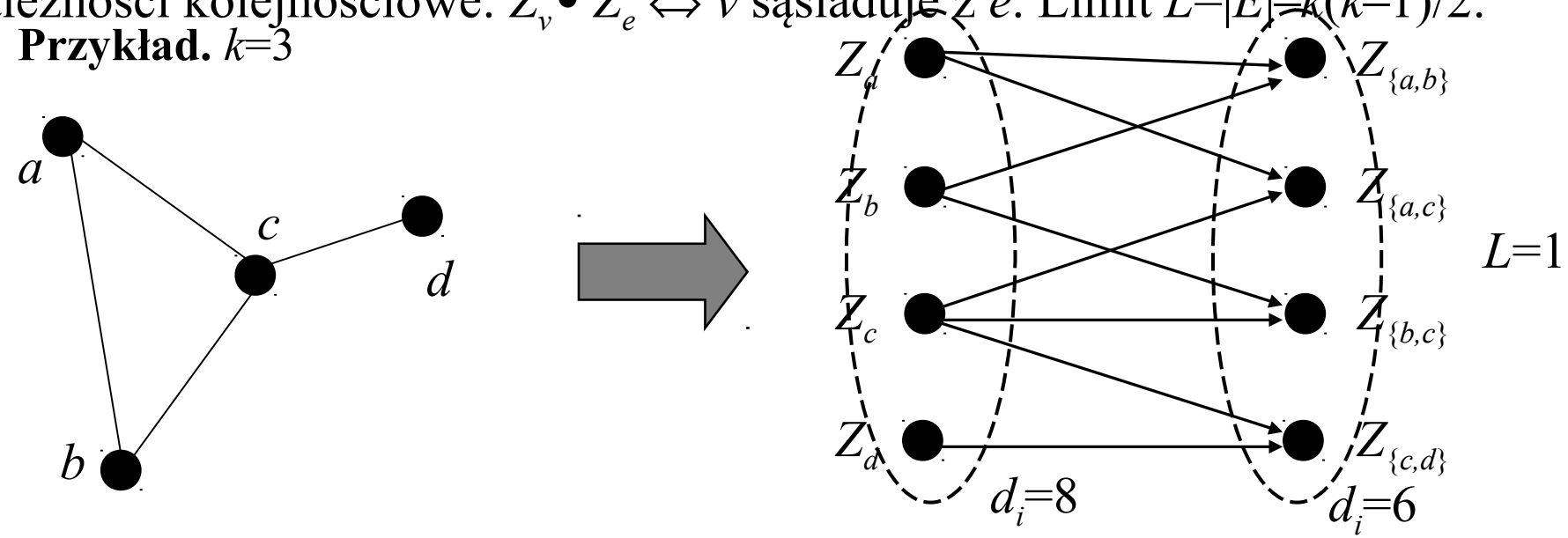
NP–trudność pojawia się nawet dla zadań jednostkowych, w zagadnieniach $1|p_j=1,prec|\Sigma U_i$ i $1|p_j=1,prec|\Sigma T_i$.

Dowód. ***Problem kliki***: dany jest graf $G(V,E)$ i liczba k. Czy w G istnieje pełny podgraf k–wierzchołkowy?

Redukcja $PK \rightarrow 1|p_j=1,prec|\Sigma U_i$: bierzemy zadania jednostkowe Z_v z $d_i=|V \cup E|$ dla wierzchołków $v \in V$ oraz Z_e z $d_i=k+k(k-1)/2$ dla krawędzi $e \in E$.

Zależności kolejnościowe: $Z_v \bullet Z_e \Leftrightarrow v$ sąsiaduje z e. Limit $L=|E|-k(k-1)/2$.

Przykład. $k=3$



Minimalizacja liczby spóźnionych zadań na jednej maszynie

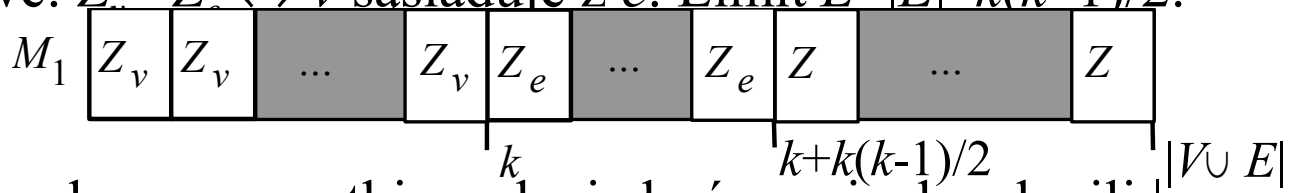
Zadania zależne i niepodzielne

NP–trudność pojawia się nawet dla zadań jednostkowych, w zagadnieniach $1|p_j=1,prec|\Sigma U_i$ i $1|p_j=1,prec|\Sigma T_i$.

Dowód. ***Problem kliki***: dany jest graf $G(V,E)$ i liczba k. Czy w G istnieje pełny podgraf k–wierzchołkowy?

Redukcja $PK \rightarrow 1|p_j=1,prec|\Sigma U_i$: bierzemy zadania jednostkowe Z_v z $d_i=|V\cup E|$ dla wierzchołków $v\in V$ oraz Z_e z $d_i=k+k(k-1)/2$ dla krawędzi $e\in E$.

Zależności kolejnościowe: $Z_v \bullet Z_e \Leftrightarrow v$ sasiaduje z e. Limit $L=|E|-k(k-1)/2$.



W uszeregowaniu optymalnym wszystkie zadania kończą się do chwili $|V\cup E|$. Jeżeli $\Sigma U_i \le L$, czyli co najmniej $k(k-1)/2$ zadań Z_e wykona się przed $k+k(k-1)/2$, ich krawędzie muszą sąsiadować z co najmniej k wierzchołkami (których zadania Z_v poprzedzają te Z_e). Jest to możliwe jedynie, gdy k wierzchołków tworzy klikę.

Podobnie przebiega redukcja $PK \rightarrow 1|p_j=1,prec|\Sigma T_i$.

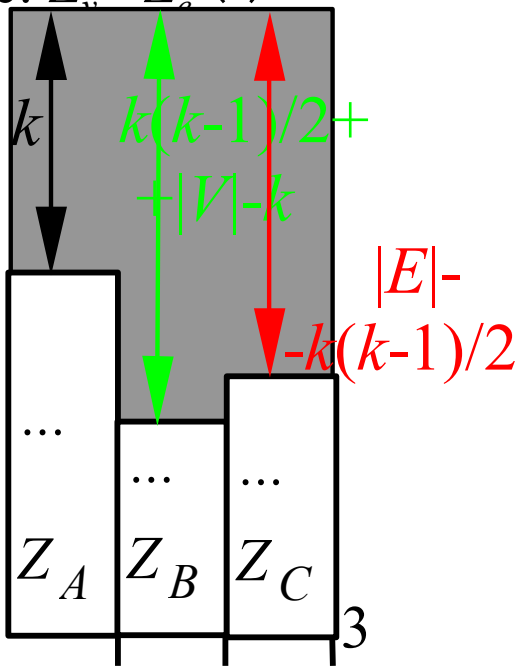
Minimalizacja liczby spóźnionych zadań na jednej maszynie

Procesory równoległe, minimalizacja C_{max} ... znowu

Znamy wielomianową redukcję *PK* ➔ $1|p_j=1,prec|\Sigma U_i$. A jak dowieść NP–trudności $P|p_j=1,prec|C_{max}$? Bardzo podobnie.

Dowód. ***Problem kliki***: dany jest graf $G(V,E)$ bez wierzchołków izolowanych i liczba k. Czy w G istnieje pełny podgraf k–wierzchołkowy?

Redukcja *PK* ➔ $P|p_j=1,prec|C_{max}$: zadania jednostkowe Z_v dla wierzchołków $v\in V$ oraz Z_e dla krawędzi $e\in E$. Zależności kolejnościowe: $Z_v \bullet Z_e \Leftrightarrow$ v sąsiaduje z e. Limit L=3. Ponadto 3 „piętra” zadań jednostkowych $Z_{A1},Z_{A2},... \bullet Z_{B1},Z_{B2},... \bullet Z_{C1},Z_{C2},...$ i liczba maszyn m taka, by harmonogram z C_{max}=3:



Jeśli rzeczywiście C_{max}*=3, to:

- wszystkie szare pola są wypełnione przez Z_v i Z_e,
- w chwili 1 są tylko Z_v, a w 3 tylko Z_e,
- w chwili 2 działa $k(k–1)/2$ zadań Z_e, a ich krawędzie sąsiadują z k wierzchołkami (których zadania Z_v działają w chwili 1) tworzącymi klikę.

Szeregowanie na procesorach dedykowanych

Przypomnienie

- zadania są podzielone na operacje (zadanie Z_j ma operację O_{ij} do wykonania na maszynie M_i, o długości czasowej p_{ij}). Zadanie kończy się wraz z wykonaniem swej najpóźniejszej operacji,
- dopuszcza się sytuację, gdy zadanie nie wykorzystuje wszystkich maszyn (*operacje puste*),
- żadne dwie operacje tego samego zadania nie mogą wykonywać się równocześnie,
- żaden procesor nie może jednocześnie pracować nad różnymi operacjami.

Systemy obsługi:

- ***system przepływowy*** (***flow shop***) – operacje każdego zadania są wykonywane przez procesory w tej samej kolejności wyznaczonej przez numery maszyn,
- ***system otwarty*** (***open shop***) – kolejność wykonania operacji w obrębie zadań jest dowolna,
- inne, ogólniejsze ...

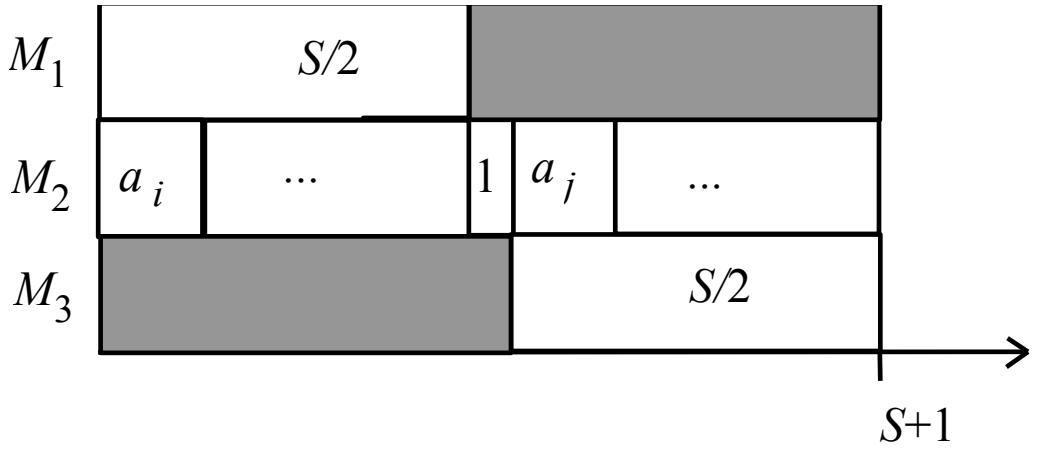
Szeregowanie na procesorach dedykowanych

System przepływowy

Już przypadek trzech maszyn ($F3||C_{max}$) jest NP–trudny.

Dowód. ***Problem podziału***: dany jest ciąg $a_1,...a_n$ liczb naturalnych o $S=\Sigma_{i=1,...,n}$ a_i parzystej. Czy istnieje jego podciąg o sumie $S/2$?

Redukcja PP ➔ $F3||C_{max}$: bierzemy n zadań o czasach $(0,a_i,0)$ $i=1,...,n$ oraz jedno z czasami $(S/2,1,S/2)$. Pytamy o istnienie uszeregowania z $C_{max} \leq S+1$.



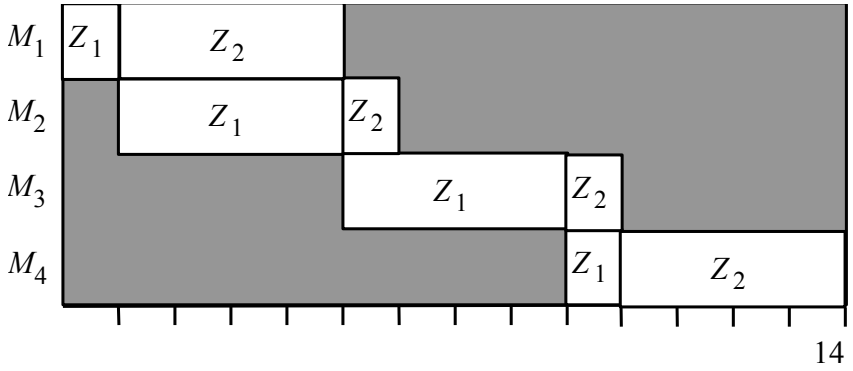
Permutacyjny system przepływowy (***PF***): system przepływowy + kolejność podejmowania operacji z poszczególnych zadań musi być jednakowa na każdej maszynie (permutacja numerów zadań).

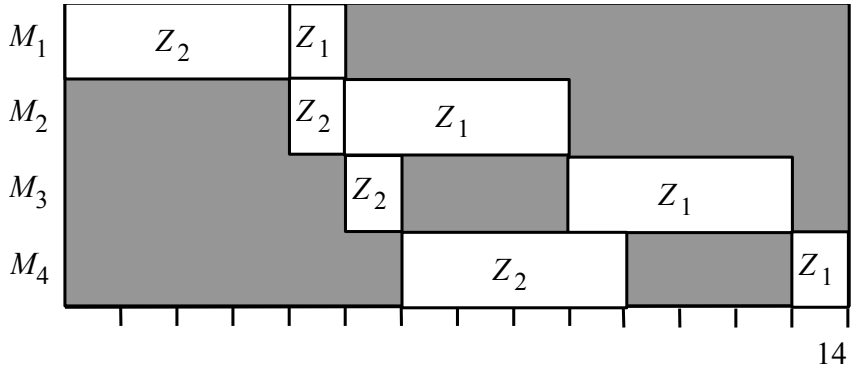
Szeregowanie na procesorach dedykowanych

System przepływowy

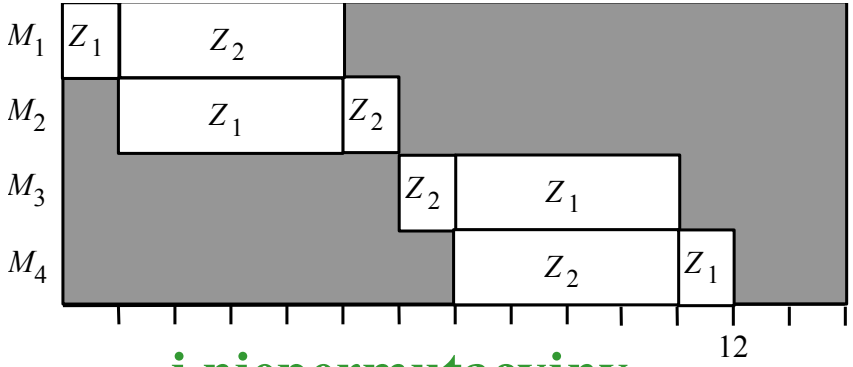
W zwykłym systemie przepływowym operacje w zadaniach wykonują się w tej samej kolejności (numeracja procesorów) ale kolejność podejmowania zadań może się zmieniać pomiędzy maszynami. Jest to możliwe nawet w harmonogramie optymalnym.

Przykład. m=4, n=2. Czasy wykonania (1,4,4,1) dla Z_1 i (4,1,1,4) dla Z_2.





Harmonogramy permutacyjne ...



i niepermutacyjny

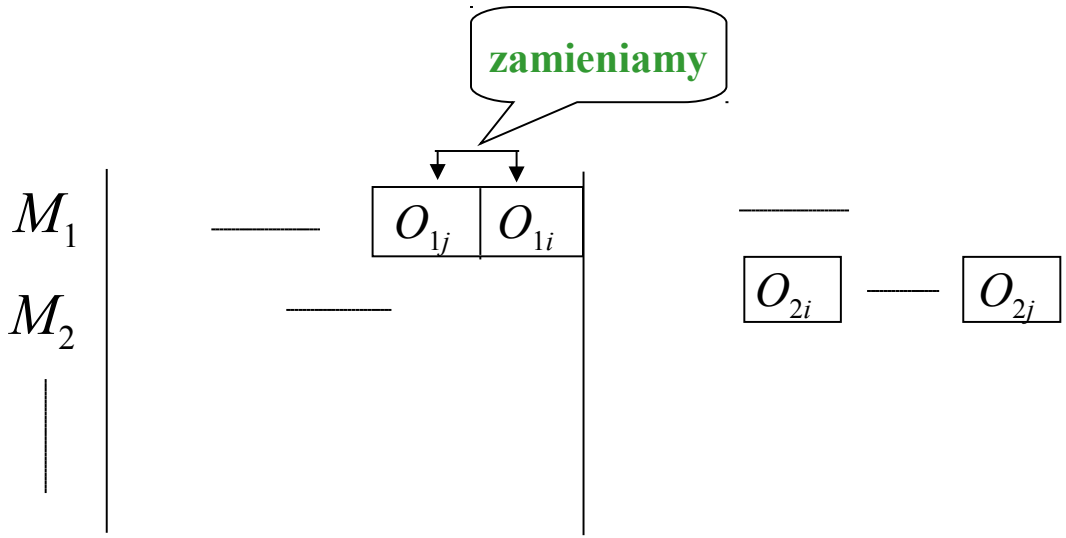
Szeregowanie na procesorach dedykowanych

System przepływowy

Jeżeli $p_{ij}>0$, to istnieje optymalne uszeregowanie flow shopu, w którym kolejność podejmowania zadań jest jednakowa na pierwszych dwóch maszynach, oraz jednakowa na ostatnich dwóch.

Wniosek. Harmonogram optymalny dla $PFm||C_{max}$ jest wtedy ($p_{ij}>0$) optymalny dla $Fm||C_{max}$ przy $m\leq 3$ (sprawdzamy więc tylko harmonogramy permutacyjne, mniej do przeszukania!).

Dowód. Na M_1 można "poprawić" kolejność operacji, by była zgodna z M_2.



Szeregowanie na procesorach dedykowanych

System przepływowy

Przypadek dwóch maszyn $F2||C_{max}$ (jak również z operacjami podzielnymi $F2|pmtn|C_{max}$), ***algorytm Johnsona*** $O(n \log n)$:

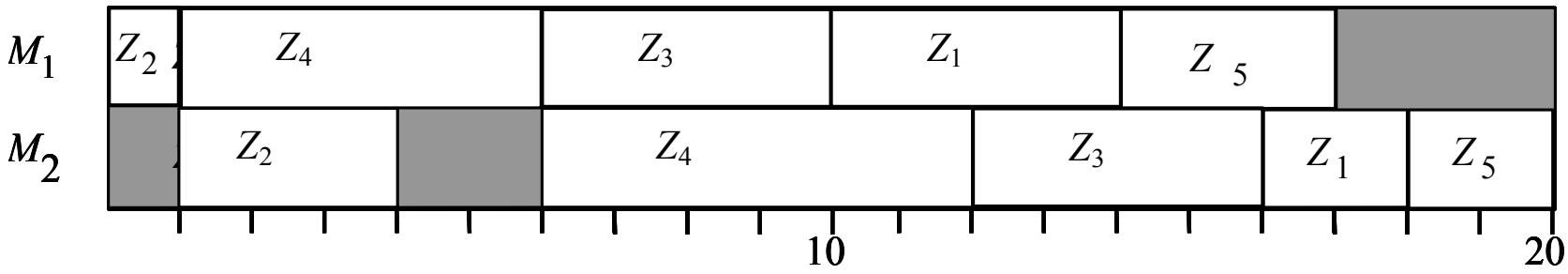
1. Podziel zadania na zbiory $N_1=\{Z_j: p_{1j}<p_{2j}\}$, $N_2=\{Z_j: p_{1j}\geq p_{2j}\}$,
2. Porządkuj N_1 w kolejności niemalejącej p_{1j} a N_2 według nierosnącego p_{2j},
3. Utwórz harmonogram permutacyjny (maksymalnie „przesunięty w lewo”) na podstawie kolejności N_1,N_2.

Przykład. Algorytm Johnsona, m=2, n=5.

	Z_1	Z_2	Z_3	Z_4	Z_5
M_1	4	1	4	5	3
M_2	2	3	4	6	2
	N_2	N_1	N_2	N_1	N_2

N_1: Z_2 Z_4 (1, 5)

N_2: Z_3 Z_1 Z_5 (4, 2, 2)



Szeregowanie na procesorach dedykowanych

System przepływowy

Przypadek dwóch maszyn $F2||C_{\max}$ (jak również z operacjami podzielnymi $F2|pmtn|C_{\max}$), ***algorytm Johnsona*** $O(n \log n)$:

1. Podziel zadania na zbiory $N_1=\{Z_j: p_{1j}<p_{2j}\}$, $N_2=\{Z_j: p_{1j}\geq p_{2j}\}$,
2. Porządkuj N_1 w kolejności niemalejącej p_{1j} a N_2 według nierosnącego p_{2j},
3. Utwórz harmonogram permutacyjny (maksymalnie „przesunięty w lewo”) na podstawie kolejności N_1, N_2.

Dowód. Lemat Jonsona: Jeśli w „zachłannym” harmonogramie permutacyjnym dla każdych kolejnych Z_j, Z_{j+1} zachodzi $\min\{p_{1j},p_{2,j+1}\}\leq \min\{p_{2j},p_{1,j+1}\}$, to ich zamiana nie zmniejszy $C_{\max}$.

Dowód. Dla pewnego s zachodzi:

$$C_{\max}=\sum_{i=1}^{s} p_{1i}+\sum_{i=s}^{n} p_{2i}=\left(\sum_{i=1}^{s} p_{1i}-\sum_{i=1}^{s-1} p_{2i}\right)+\sum_{i=1}^{n} p_{2i}=\sum_{i=1}^{n} p_{2i}+\Delta_s$$

$\Delta_s =$ **maksymalna** Δ_k

Oznaczmy składniki tej postaci (z *k* w miejscu *s*) przez Δ_k.

Po zmianie kolejności Z_j i Z_{j+1} $C_{\max}$ nie ulegnie zmniejszeniu jeśli

$$\max\left\{\Delta_j,\Delta_{j+1}\right\}\leq\max\left\{\Delta_j',\Delta_{j+1}'\right\}$$

Szeregowanie na procesorach dedykowanych

System przepływowy

Przypadek dwóch maszyn $F2||C_{max}$ (jak również z operacjami podzielnymi $F2|pmtn|C_{max}$), ***algorytm Johnsona*** $O(n \log n)$:

1. Podziel zadania na zbiory $N_1=\{Z_j: p_{1j}<p_{2j}\}$, $N_2=\{Z_j: p_{1j}\geq p_{2j}\}$,
2. Porządkuj N_1 w kolejności niemalejącej p_{1j} a N_2 według nierosnącego p_{2j},
3. Utwórz harmonogram permutacyjny (maksymalnie „przesunięty w lewo”) na podstawie kolejności N_1, N_2.

Dowód. Lemat Jonsona: Jeśli w „zachłannym” harmonogramie permutacyjnym dla każdych kolejnych Z_j, Z_{j+1} zachodzi $\min\{p_{1j},p_{2,j+1}\}\leq \min\{p_{2j},p_{1,j+1}\}$, to ich zamiana nie zmniejszy C_{max}.

Dowód. Po zmianie kolejności Z_j i Z_{j+1} C_{max} nie zmaleje jeśli

$$\max\{\Delta_j,\Delta_{j+1}\}\leq\max\{\Delta_j',\Delta_{j+1}'\}$$

$$\Leftrightarrow \max\{p_{1j},p_{1j}-p_{2j}+p_{1,j+1}\}\leq\max\{p_{1,j+1},p_{1,j+1}-p_{2,j+1}+p_{1j}\}$$

$$\Leftrightarrow (p_{1j}\leq p_{1,j+1}\wedge p_{1j}-p_{2j}+p_{1,j+1}\leq p_{1,j+1})\vee$$
$$(p_{1j}\leq p_{1,j+1}-p_{2,j+1}+p_{1j}\wedge p_{1j}-p_{2j}+p_{1,j+1}\leq p_{1,j+1}-p_{2,j+1}+p_{1j})$$

$$\Leftrightarrow p_{1j}\leq\min\{p_{2j},p_{1,j+1}\}\vee p_{2,j+1}\leq\min\{p_{2j},p_{1,j+1}\}$$

Szeregowanie na procesorach dedykowanych

System przepływowy

Przypadek dwóch maszyn $F2||C_{max}$ (jak również z operacjami podzielnymi $F2|pmtn|C_{max}$), ***algorytm Johnsona*** $O(n \log n)$:

1. Podziel zadania na zbiory $N_1=\{Z_j: p_{1j}<p_{2j}\}$, $N_2=\{Z_j: p_{1j}\geq p_{2j}\}$,
2. Porządkuj N_1 w kolejności niemalejącej p_{1j} a N_2 według nierosnącego p_{2j},
3. Utwórz harmonogram permutacyjny (maksymalnie „przesunięty w lewo”) na podstawie kolejności N_1, N_2.

Dowód. Lemat Jonsona: Jeśli w „zachłannym” harmonogramie permutacyjnym dla każdych kolejnych Z_j, Z_{j+1} zachodzi $\min\{p_{1j},p_{2,j+1}\}\leq \min\{p_{2j},p_{1,j+1}\}$, to ich zamiana nie zmniejszy C_{max}.

Ale dla dowolnej pary zadań Z_i, Z_j ($i<j$) w algorytmie Johnsona:

- oba z N_1: $p_{1i}=\min\{p_{1i},p_{2j}\}\leq \min\{p_{2i},p_{1j}\}$,
- oba z N_2: $p_{2j}=\min\{p_{1i},p_{2j}\}\leq \min\{p_{2i},p_{1j}\}$,
- Z_i jest z N_1, a Z_j z N_2: $p_{1i}\leq p_{2i}$ i $p_{2j}\leq p_{1j}$, więc $\min\{p_{1i},p_{2j}\}\leq \min\{p_{2i},p_{1j}\}$.

Wniosek: sortując bąbelkowo „zachłanny” harmonogram permutacyjny wg kolejności Johnsona przy każdej zamianie nie zwiększamy C_{max}.

Przypadek operacji podzielnych: można je scalić na obu procesorach nie zwiększając C_{max}. Zatem uszeregowanie optymalne nie musi dzielić zadań.

Szeregowanie na procesorach dedykowanych

System przepływowy

• problem $F2||\Sigma C_j$ jest NP–trudny,

• dla $F3||C_{max}$, w którym M_2 jest ***zdominowana*** przez M_1 ($\forall_{i,j}\ p_{1i} \geq p_{2j}$) lub przez M_3 ($\forall_{i,j}\ p_{3i} \geq p_{2j}$) można użyć Johnsona stosując zmodyfikowane czasy wykonania ($p_{1i}+p_{2i}$, $p_{2i}+p_{3i}$), $i=1,...,n$.

Algorytm wielomianowy (***graficzny***) dla $F||C_{max}$ z $n=2$ zadaniami i dowolną liczbą maszyn. Szkic:

1. Na osi OX odkładamy kolejne odcinki o długości p_{11}, p_{21}, ..., p_{m1} (czasy pracy maszyn nad Z_1). Na osi OY odkładamy odcinki o długości p_{12}, p_{22}, ..., p_{m2} (czasy pracy maszyn nad Z_2).

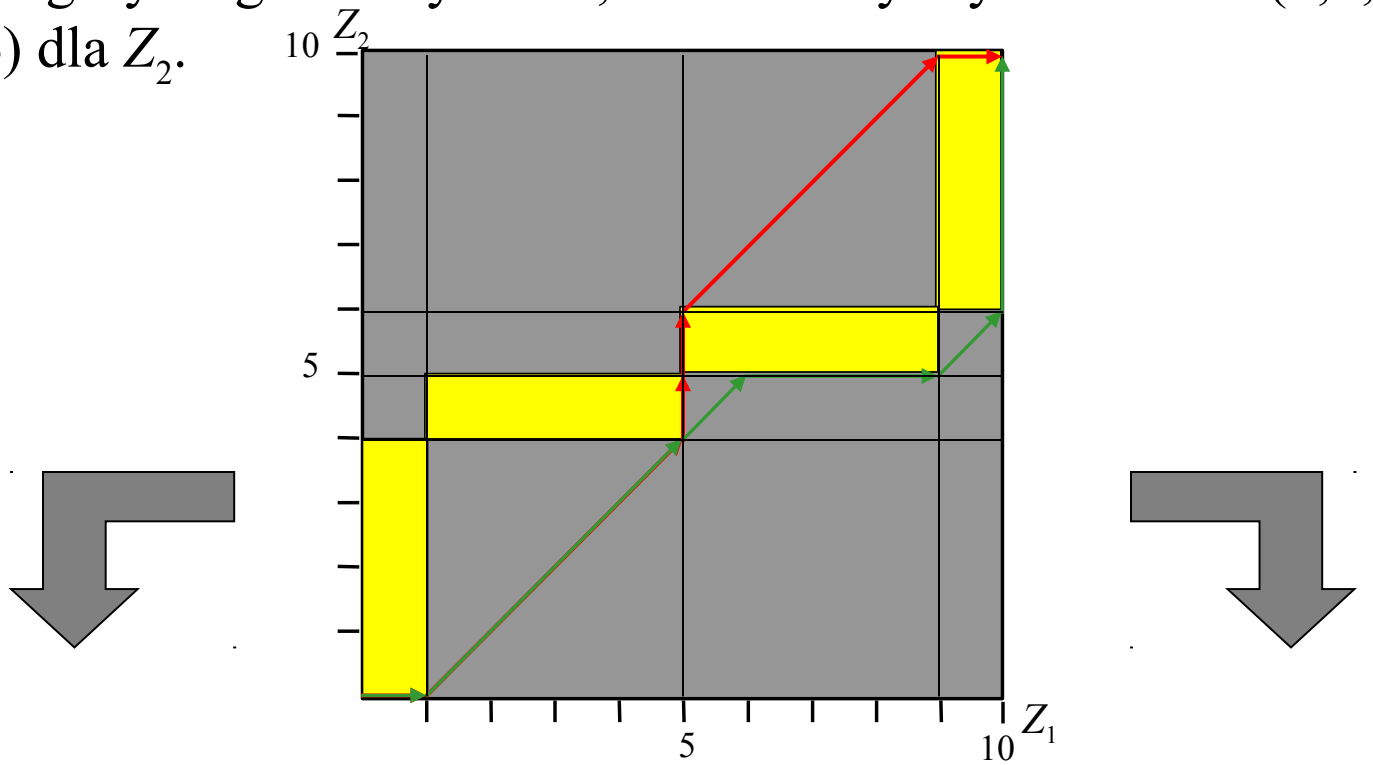
2. Zaznaczamy obszary zakazane – wnętrza prostokątów będących iloczynami kartezjańskimi odpowiednich odcinków (ta sama maszyna nie pracuje równocześnie nad dwoma zadaniami).

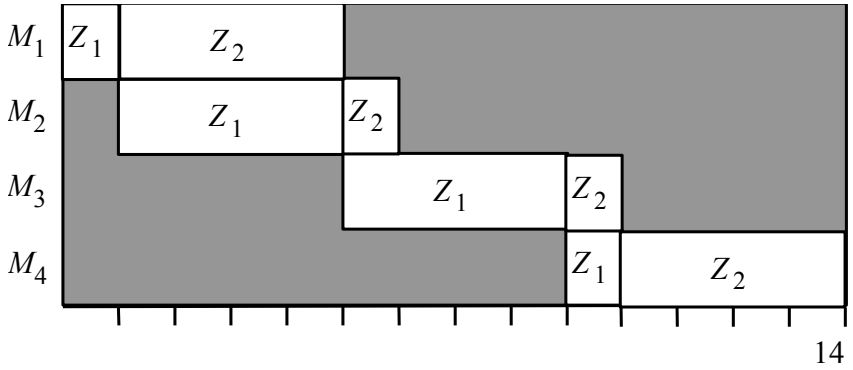
3. Szukamy najkrótszej łamanej o odcinkach równoległych do osi (praca jednej maszyny) lub biegnących pod kątem $\pi/4$ (równoczesna praca obu maszyn), łączącej (0,0) z ($\Sigma_i p_{i1}$, $\Sigma_i p_{i2}$) – używamy metryki $d((x_1,x_2),(y_1,y_2))=\max\{|x_1-x_2|, |y_1-y_2|\}$. Jej długość to długość harmonogramu.

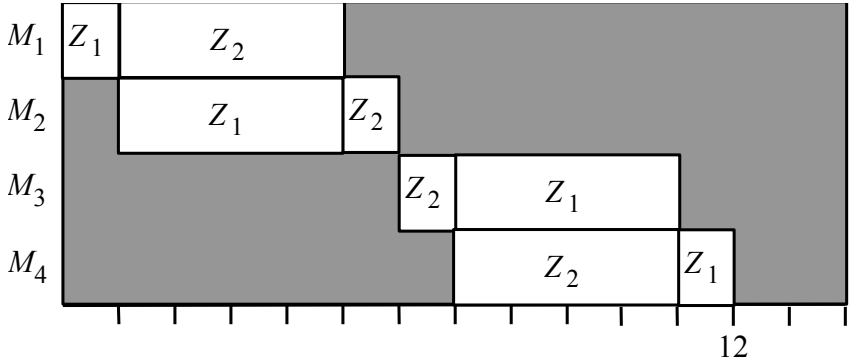
Szeregowanie na procesorach dedykowanych

System przepływowy

Przykład. Algorytm graficzny. m=4, n=2 i czasy wykonania to (1,4,4,1) dla Z_1 i (4,1,1,4) dla Z_2.





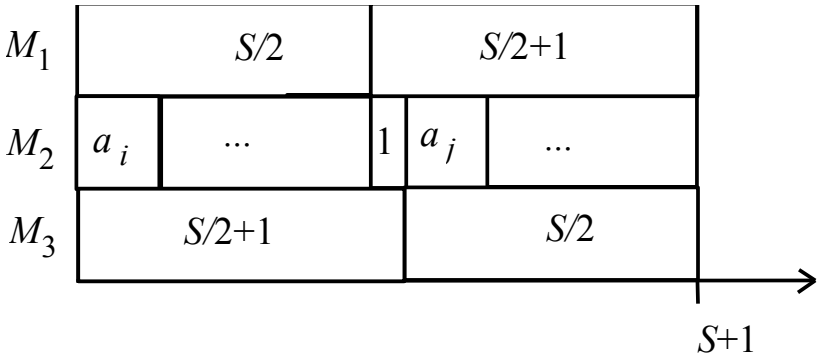


Szeregowanie na procesorach dedykowanych

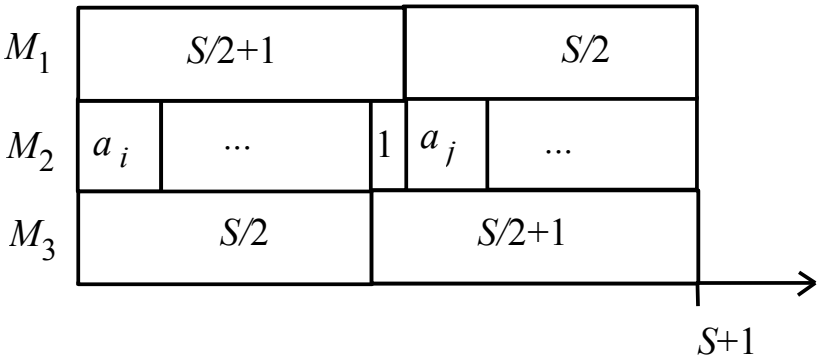
System otwarty

Znów przypadek trzech maszyn ($O3||C_{max}$) jest NP–trudny.

Dowód. Redukcja *PP* ➔ $O3||C_{max}$: bierzemy n zadań o czasach $(0,a_i,0)$ $i=1,...,n$ oraz oraz trzy zadania z czasami $(S/2,1,S/2)$, $(S/2+1,0,0)$, $(0,0,S/2+1)$. Pytamy o istnienie uszeregowania z $C_{max} \leq S+1$.



lub



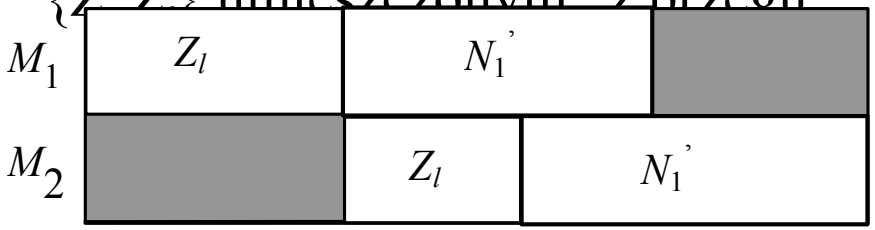
Problem $O2||\Sigma C_j$ jest NP–trudny.

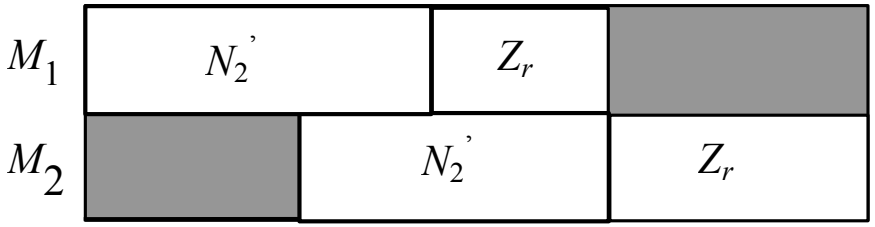
Szeregowanie na procesorach dedykowanych

System otwarty

Przypadek dwóch maszyn $O2||C_{max}$ (jak również $O2|pmtn|C_{max}$), ***algorytm Gonzalez–Sahni*** $O(n)$:

1. Podziel zadania na zbiory $N_1=\{Z_j: p_{1j}<p_{2j}\}$, $N_2=\{Z_j: p_{1j}\geq p_{2j}\}$,
2. Wybierz 2 zadania Z_r, Z_l, że: $p_{1r}\geq \max_{Zj\in N2} p_{2j}$; $p_{2l}\geq \max_{Zj\in N1} p_{1j}$;
3. $p_1:=\Sigma_i p_{1i}$; $p_2:=\Sigma_i p_{2i}$; $N_1':=N_1\setminus\{Z_r,Z_l\}$; $N_2':=N_2\setminus\{Z_r,Z_l\}$; Dla $N_1'\cup\{Z_l\}$ i $N_2'\cup\{Z_r\}$ utwórz harmonogramy (permutacyjne i no–idle) z zadaniem z $\{Z_r,Z_l\}$ umieszczonym „z brzegu”:

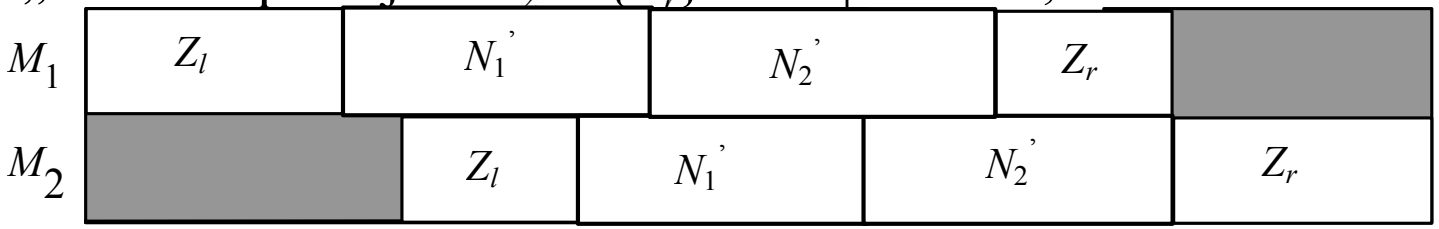




4. Sklej oba harmonogramy. **if** $p_1-p_{1l}\geq p_2-p_{2r}$ ($p_1-p_{1l}<p_2-p_{2r}$)

   then „dosuń” operacje z $N_1'\cup\{Z_l\}$ na M_2 w prawo

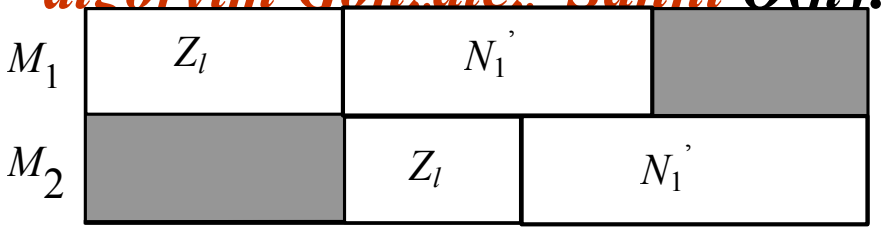
   else „dosuń” operacje z $N_2'\cup\{Z_r\}$ na M_1 w lewo;

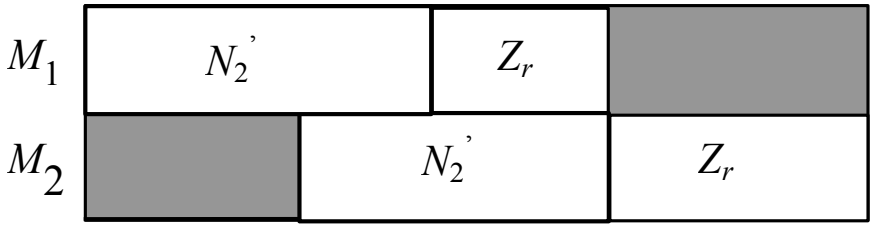


Szeregowanie na procesorach dedykowanych

System otwarty

Przypadek dwóch maszyn $O2||C_{max}$ (jak również $O2|pmtn|C_{max}$), ***algorytm Gonzalez–Sahni*** $O(n)$:

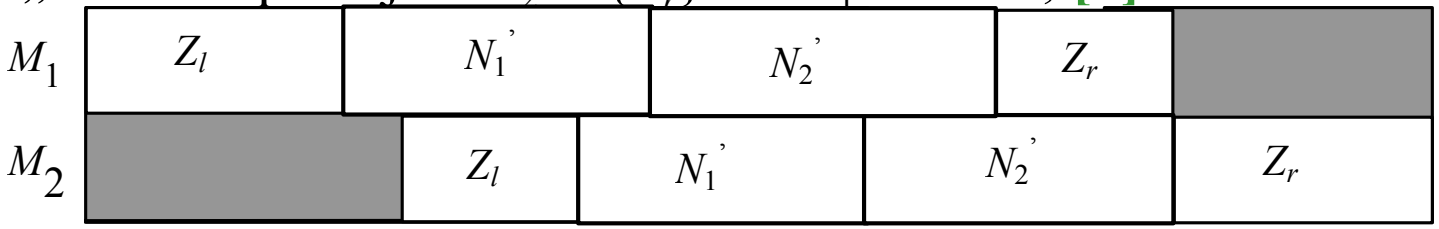




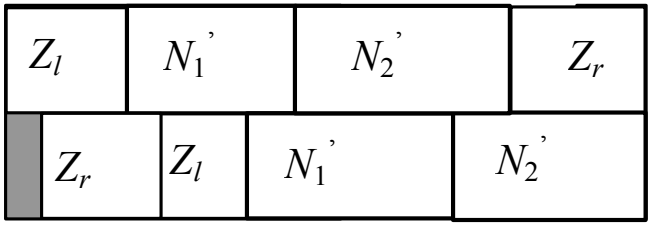
4. Sklej oba harmonogramy. **if** $p_1 - p_{1l} \geq p_2 - p_{2r}$ ($p_1 - p_{1l} < p_2 - p_{2r}$)

then „dosuń” operacje z $N_1' \cup \{Z_l\}$ na M_2 w prawo

else „dosuń” operacje z $N_2' \cup \{Z_r\}$ na M_1 w lewo; [*]

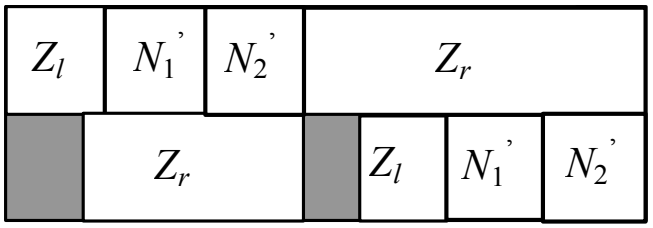


5. Operację z Z_r na M_2 ([*] Z_l na M_1) przenieś na początek ([*] koniec) i maksymalnie w prawo ([*] w lewo).



lub

$C_{max} = \max\{p_1, p_2\}$



$C_{max} = \max\{p_1, p_{1r} + p_{2r}\}$

Szeregowanie na procesorach dedykowanych

System otwarty

Przykład. Algorytm Gonzalez–Sahni, m=2, n=5.

	Z_1	Z_2	Z_3	Z_4	Z_5
M_1	4	1	4	5	3
M_2	2	3	4	6	2
	N_2	N_1	N_2	N_1	N_2

N_1: Z_2, Z_4

N_2: Z_1, Z_3, Z_5

$Z_r := Z_1$

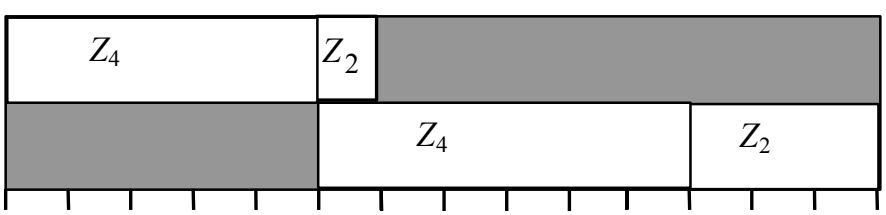
$Z_l := Z_4$

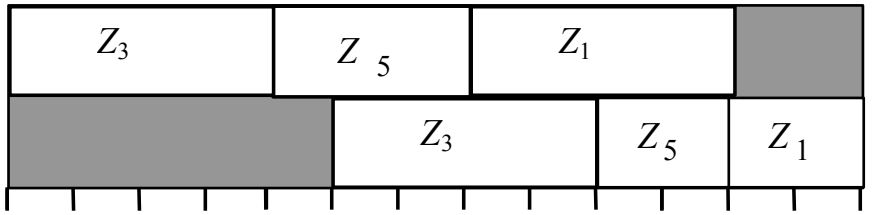
$p_{1r} \geq \max_{Zj \in N2} p_{2j} = 4$

$p_{2l} \geq \max_{Zj \in N1} p_{1j} = 5$

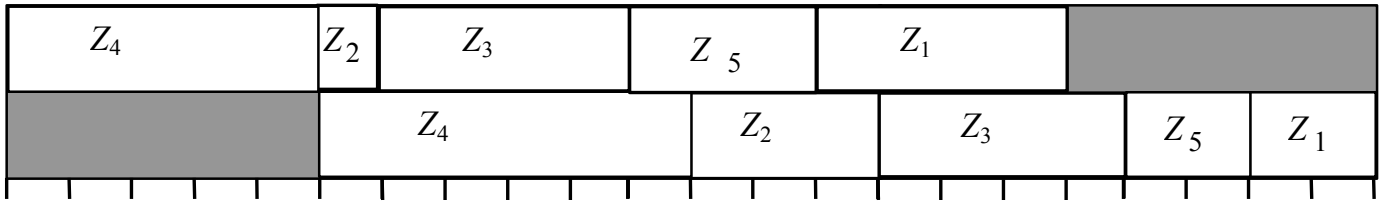
N_1': Z_2

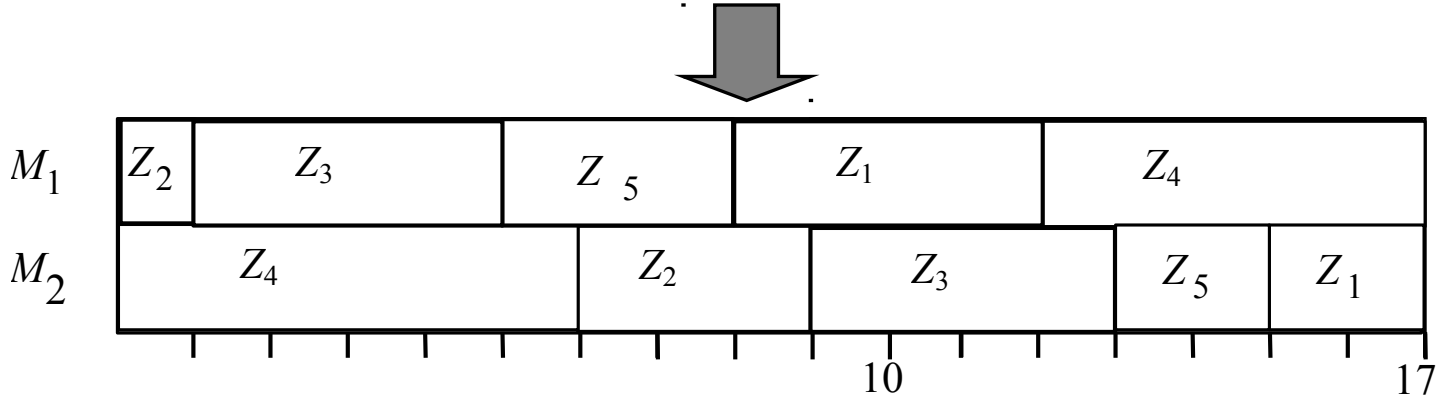
N_2': Z_3, Z_5





Scalanie:



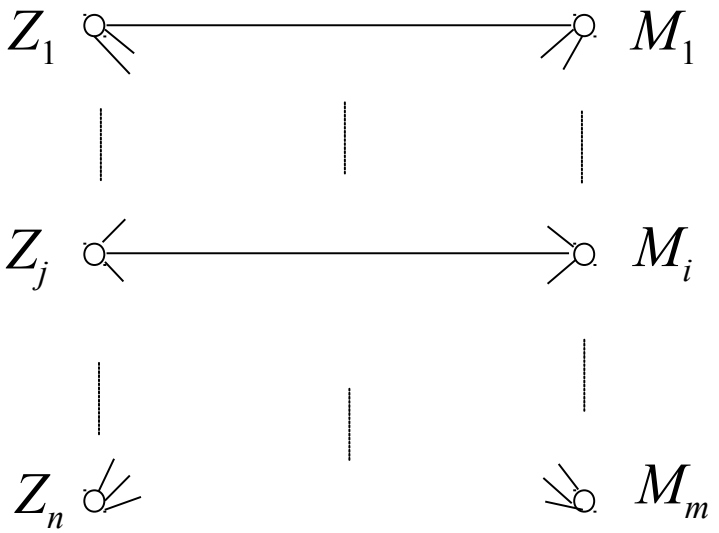


Szeregowanie na procesorach dedykowanych

System otwarty

Operacje zero–jedynkowe ($O|ZUET|C_{max}$): ***algorytm wielomianowy*** oparty na kolorowaniu krawędziowym grafów dwudzielnych.

1. Graf dwudzielny G:
   a) wierzchołki jednej partycji to zadania, a drugiej to procesory,
   b) każdej niepustej operacji O_{ij} odpowiada krawędź $\{Z_j, M_i\}$.



2. Kolorujemy krawędziowo $\Delta(G)$ kolorami, interpretując barwy jako jednostki czasu przydzielone operacjom,
(**własność:** poprawny harmonogram $\Leftrightarrow$ poprawne pokolorowanie).
3. Wtedy $C_{max}^* = \Delta(G) = \max\{\max_i \Sigma_{j=1,\ldots,n} p_{ij}, \max_j \Sigma_{i=1,\ldots,m} p_{ij}\}$. Oczywiście krótszy harmonogram nie istnieje.

Szeregowanie na procesorach dedykowanych

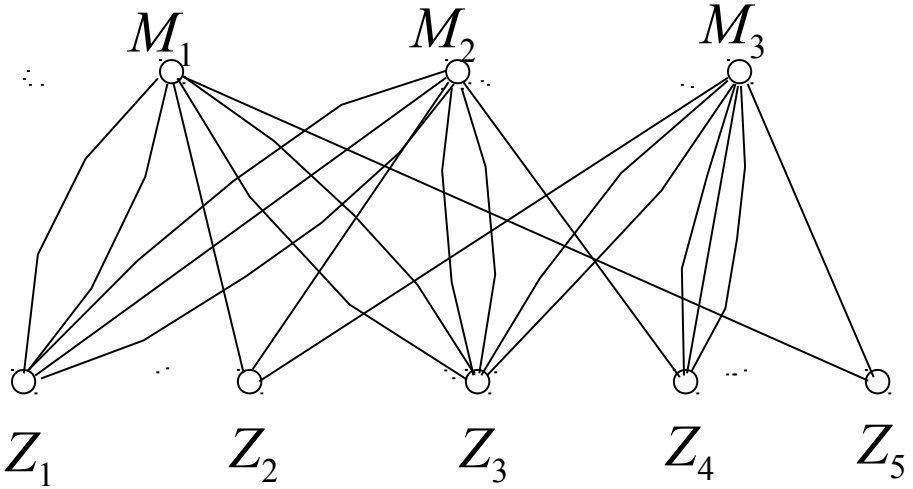
System otwarty

Operacje podzielne ($O|pmtn|C_{max}$): ***algorytm pseudowielomianowy*** podobny do przypadku $O|ZUET|C_{max}$. Różnica: G jest multigrafem dwudzielnym, niepustą operację O_{ij} dzielimy na p_{ij} „operacji" jednostkowych, odpowiadają im krawędzie równoległe.

Nadal $C_{max}^* = \max\{\max_i \Sigma_{j=1,\ldots,n} p_{ij}, \max_j \Sigma_{i=1,\ldots,m} p_{ij}\}$.

Czemu „pseudo"? Możemy uzyskać niewielomianową liczbę krawędzi ($=\Sigma_{i=1,\ldots,m;\, j=1,\ldots,n} p_{ij}$), a w uszeregowaniu niewielomianową liczbę przerwań.

Przykład. Podzielny system otwarty. $m=3$, $n=5$, $p_1=(2,3,0)$, $p_2=(1,1,1)$, $p_3=(2,2,2)$, $p_4=(0,1,3)$, $p_5=(1,0,1)$.



Szeregowanie na procesorach dedykowanych

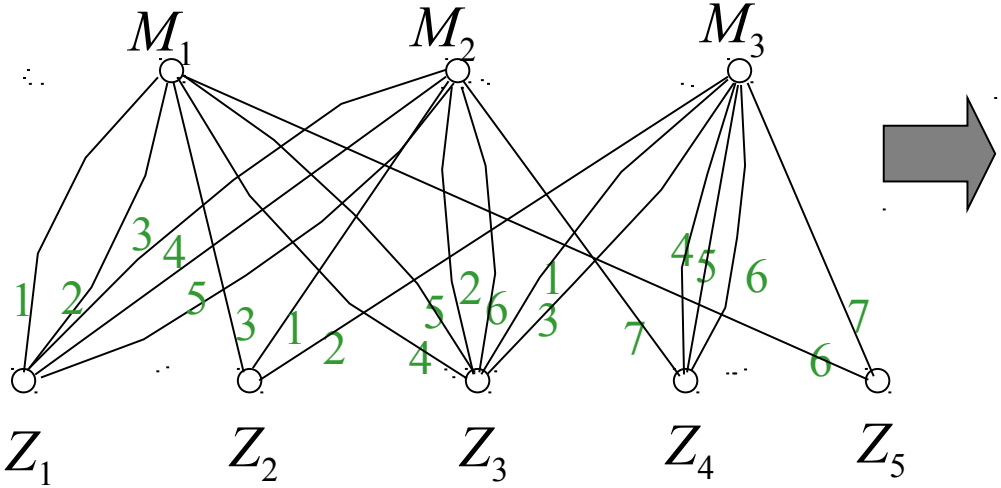
System otwarty

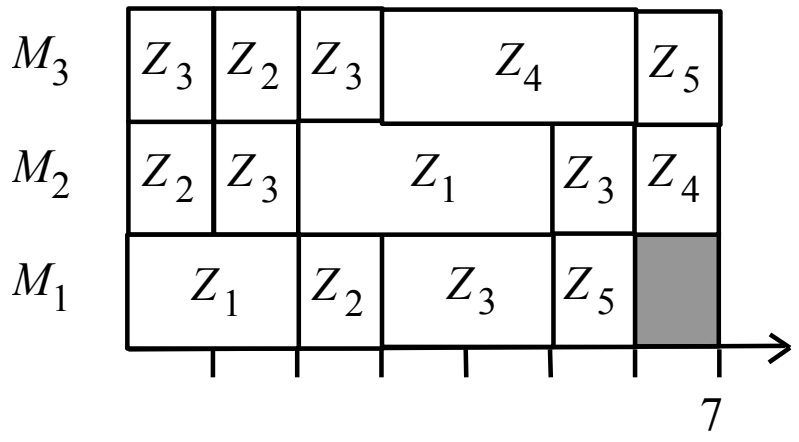
Operacje podzielne ($O|pmtn|C_{max}$): ***algorytm pseudowielomianowy*** podobny do przypadku $O|ZUET|C_{max}$. Różnica: G jest multigrafem dwudzielnym, niepustą operację O_{ij} dzielimy na p_{ij} „operacji" jednostkowych, odpowiadają im krawędzie równoległe.

Nadal $C_{max}^* = \max\{\max_i \Sigma_{j=1,\ldots,n} p_{ij}, \max_j \Sigma_{i=1,\ldots,m} p_{ij}\}$

Czemu „pseudo"? Możemy uzyskać niewielomianową liczbę krawędzi ($=\Sigma_{i=1,\ldots,m;\, j=1,\ldots,n} p_{ij}$), a w uszeregowaniu niewielomianową liczbę przerwań.

Przykład. Podzielny system otwarty. $m=3$, $n=5$, $p_1=(2,3,0)$, $p_2=(1,1,1)$, $p_3=(2,2,2)$, $p_4=(0,1,3)$, $p_5=(1,0,1)$.





Szeregowanie na procesorach dedykowanych

System otwarty

Operacje podzielne ($O|pmtn|C_{max}$):

- istnieje ***algorytm wielomianowy*** oparty na tzw. *kolorowaniu cząstkowym* krawędzi grafu z wagami (w grafie G operacji O_{ij} odpowiada jedna krawędź $\{Z_j,M_i\}$ z wagą p_{ij}),

Procesory równoległe, minimalizacja C_{max} ... znowu

Algorytm wielomianowy dla maszyn dowolnych $R|pmtn|C_{max}$.

Redukcja $R|pmtn|C_{max}$ ➔ $O|pmtn|C_{max}$. Niech x_{ij} to część zadania Z_j wykonana na M_i (więc w czasie $t_{ij}= p_{ij}x_{ij}$). Znając optymalne wartości x_{ij}, moglibyśmy zastosować powyższy algorytm traktując fragmenty zadań jak podzielne operacje przypisane do maszyn systemu otwartego (te same warunki poprawności!). **Skąd je wziąć?** Wyznaczamy minimalny *stopień ważony* grafu G, czyli $C=C_{max}$* oraz x_{ij} z programowania liniowego:

minimalizuj C przy warunkach:

$$\Sigma_{i=1,\ldots,m}\, x_{ij}=1,\ j=1,\ldots,n$$

$$C\geq\Sigma_{j=1,\ldots,n}\, p_{ij}x_{ij},\quad i=1,\ldots,m, \qquad C\geq\Sigma_{i=1,\ldots,m}\, p_{ij}x_{ij},\quad j=1,\ldots,n.$$