Badania Operacyjne

Na podstawie wykładu
prof. dr hab. inż Jana Węglarza

Skład: *yacoob, nuoritoveri, Jarek*

23 czerwca 2009

Wykład 1 02.03.2009

Spis treści

1 Przedmiot badań operacyjnych **2**
1.1 Opisowe definicje zakresu badań operacyjnych . . . 2

2 Probabilistyczne (kolejkowe) problemy szeregowania zadań **3**
2.1 Proces Poissona . . . 3
2.2 Podstawowe pojęcia teorii kolejek . . . 6
2.3 Podstawowe algorytmy (regulaminy) szeregowania (obsługi) . . . 9
2.3.1 Algorytm FIFO . . . 9
2.3.2 Algorytmy ze stałymi priorytetami . . . 12
2.3.3 Algorytmy cykliczne (karuzelowe) . . . 13
2.3.4 Algorytm LIFO . . . 15
2.3.5 Systemy obsługi z momentami przybycia i/lub czasami wykonywania zadań zależnymi od stanu . . . 16

3 Deterministyczne problemy szeregowania zadań **20**
3.1 Podstawowe pojęcia, założenia i ich interpretacje . . . 20
3.2 Przykładowe problemy, metody i algorytmy . . . 27
3.2.1 Niezależne zadania . . . 28
3.2.2 Dowolne maszyny . . . 29
3.2.3 Zależne zadania . . . 30

4 Wybrane problemy analizy sieci zadań **32**
4.1 Problem analizy czasowej, przypadek deterministyczny: sformułowanie, metoda CPM . . . 32
4.2 Problem analizy czasowej, przypadek probabilistyczny: sformułowanie, metoda PERT . . . 35
4.3 Problem analizy czasowo-kosztowej: sformułowanie, metoda CPM/MCX, metoda PL . . . 36
4.4 Problemy analizy czasowo-zasobowej . . . 37

1 Przedmiot badań operacyjnych

Określenie **badania operacyjne** (bryt.: *operational research*, amer.: *operations research*) pojawiło się po raz pierwszy bezpośrednio przed wybuchem *II Wojny Światowej* w Wielkiej Brytanii i było związane z działalnością grupy ekspertów pod kierunkiem *Patricka M. Blackett'a (1897-1974)*, nazwany później *cyrkiem Blackett'a*, której zadaniem było opracowanie wariantów przeciwdziałania w wypadku ataku powietrznego państw Osi na Wielką Brytanię.

Jeśli chodzi o przedmiot i zakres **badań operacyjnych** to, podobnie jak w wypadku innych działów nauki, trudno je określić bardzo precyzyjnie. Wymagało by to bowiem, w skrajnym ujęciu, aksjomatyzacji takiej, jakiej dokonano np. w teorii prawdopodobieństwa, arytmetyce teoretycznej, czy geometrii.

Na ogół zatem pozostaje się przy podejściach opisowych, które podają tylko pewne cechy charakterystyczne definiowanych działów nauki.

W dalszym ciągu podamy kilka przykładowych opisowych definicji zakresu badań operacyjnych, a następnie dokonamy pewnego uściślenia tego zakresu.

Wykład 2 09.03.2009

1.1 Opisowe definicje zakresu badań operacyjnych

1. *„Badania operacyjne to budowanie modeli, a ściślej mówiąc* modeli użytecznych. *Mogą to być modele dowolnego rodzaju i dowolnie złożone. Użyteczne będą wówczas, gdy ich zachowanie będzie analogiczne do zachowania* systemu rzeczywistego *w takim stopniu, iż można będzie przewidywać zachowanie tego typu systemu i zmienić je w* pożądany sposób*"*[1]

2. *„Celem badań operacyjnych jest opracowanie systematycznego i racjonalnego podejścia do rozwiązywania głównych zadań sterowania systemami."*[2]

3. *„Badania operacyjne rozwijają się w postaci nauki modelującej* procesy decyzyjne *bez względu na przedmiot i podmiot danego procesu."*[3]

Jak widać powyższe określenia przedmiotu i zakresu badań operacyjnych są bardzo szerokie, co uniemożliwia wyodrębnienie innych nauk systemowych. Pewnym uściśleniem tego zakresu jest kolejna definicja, pochodząca ze statutu brytyjskiego towarzystwa badań operacyjnych:

4 *„Badania operacyjne to zastosowanie metod nauki do złożonych problemów powstających w kierowaniu i zarządzaniu* złożonymi systemami ludzi, maszyn, materiałów i pieniędzy *w przemyśle, administracji, biznesie i obronie"*

W celu dalszego uszczegółowienia przedmiotu i zakresu badań operacyjnych podajmy pewien ogólny schemat klasyfikacyjny interesującego nas obszaru nauki.

Traktując badania operacyjne jako naukę systemową musimy zdefiniować klasę systemów, którymi się ona zajmuje. Zdefiniujmy ją, dla naszych celów, jako systemy złożone z *zasobów* (ludzie, maszyny, materiały, pieniądze, *etc.*) oraz *zadań* (czynności, operacji), które *współubiegają się* o te zasoby, powiązanych *ograniczeniami kolejnościowymi*. Systemy tego typu noszą nazwy: *sieć (kompleks) zadań (czynności, operacji)*

[1] Churchman, Ackoff, Arnoff, *Introduction to Operations Research*, John Wiley & Sons New York 1957
[2] Ackhoff, Sasieni, *Fundamentals of Operations Research*, John Wiley & Sons New York 1968
[3] *Badania operacyjne w nowoczesnym zarządzaniu*, praca zbiorowa pod red. T. Kasprzaka, PWE, 1974

2 Probabilistyczne (kolejkowe) problemy szeregowania zadań

2.1 Proces Poissona

Definicja 2.1. *Proces losowy* $\{X(t), t \in < 0, \infty)\}$*, gdzie* $X(t)$ *jest liczbą zdarzeń losowych (zgłoszeń, sygnałów, klientów) w przedziale* $\langle 0, t)$*, nazywamy* procesem sygnałowym*. Wynika z tego, że* $\forall_t \forall_{t_2 > t_1}$ *zarówno X(t) jak i* $X(t_2) - X(t_1)$ *przymują wartości naturalne.*

W dalszym ciągu założymy, że:

$$P\left[X(0) = 0\right] = 1$$

Proces sygnałowy nazywamy *procesem sygnałowym o przyrostach stacjonarnych (jednorodnych)* jeśli rozkład prawdopodobieństwa liczby zgłoszeń w dowolnym przedziale o długości Δ nie zależy od umieszczenia tego przedziału na osi czasu.
W szczególności oznacza to, że średnia prędkość przybywania zgłoszeń dla takiego procesu jest stała.

Definicja 2.2. *Proces sygnałowy nazywamy procesem sygnałowym o* przyrostach niezależnych *jeśli liczby zgłoszeń w dowolnych, rozłącznych przedziałach czasowych* $\langle 0, t_1), \langle t_1, t_2), \ldots, \langle t_{n-1}, t_n)$ *są zmiennymi losowymi niezależnymi. Zauważmy, iż oznacza to, że również odstępy czasowe pomiędzy* kolejnymi *zgłoszeniami.są zmiennymi niezależnymi.*

Zauważmy także, iż na podstawie warunku 2.1 proces sygnałowy o przyrostach niezależnych jest procesem **Markowa** (szczególny przypadek warunku $P\left[X(t_0) = c\right] = 1$).

Definicja 2.3. *Proces sygnałowy o przyrostach niezależnych nazywamy* ***procesem Poissona****, jeśli* $\forall_{t \in \langle 0, \infty)}$*,* $X(t)$ *ma rozkład Poissona, tzn.* $\exists \lambda(t)$*, że* $\forall_{n \in \mathbb{N}}$*:*

$$P\left[X(t) = n\right] = \frac{\left[\lambda(t)\right]^n}{n!} e^{-\lambda(t)}$$

Jak pamiętamy, dla rozkładu Poissona:

$$E\left[X(t)\right] = \lambda(t)$$

stąd

$$E\left[X(t_2) - X(t_1)\right] = \lambda(t_2) - \lambda(t_1), t_2 > t_1$$

Zatem dla: $\forall_{t_2 > t_1}$ oraz $\forall_{n \in \mathbb{N}}$

$$P\left[X(t_2) - X(t_1)\right] = \frac{\left[\lambda(t_2) - \lambda(t_1)\right]^n}{n!} e^{-\left[\lambda(t_2) - \lambda(t_1)\right]}$$

Wykład 3 16.03.2009

W szczególności jeśli $\lambda(t) = \lambda \cdot t(\frac{\lambda(t)}{t} =$*średnia prędkość przybywania zgłoszeń* $= \lambda)$ to mówimy o *stacjonarnym (jednorodnym) procesie Poissona.* Wówczas:

$$P\left[X(t)=n\right]=\frac{(\lambda\cdot t)^n}{n!}e^{-\lambda t}$$
$$P\left[X(t_2)-X(t_1)=n\right]=\frac{\lambda^n(t_2-t_1)^n}{n!}e^{-\lambda(t_2-t_1)}$$

Wykażemy obecnie bardzo ważne twierdzenie:

Twierdzenie 2.1. *Proces sygnałowy o przyrostach niezależnych i stacjonarnych, w którym zgłoszenia zachodzą bezpośrednio i „błyskawicznie" jest stacjonarnym procesem Poissona.*

Założenia o procesie sygnałowym:

- Przyrosty niezależne
- Przyrosty stacjonarne
- Pojedynczość zachodzenia zgłoszeń
- Błyskawiczność zachodzenia zgłoszeń

Powyższe świadczą o tym, iż zjawisko zachodzi wg. definicji stacjonarnego procesu Poissona.
Jest to bardzo ważne twierdzenie które tłumaczy częstość występowania w praktyce rozkładu Poissona.

Dowód. Korzystając z założenia o *stacjonarności* przyrostu, nie musimy rozpatrywać położenia przedziału o długości t na osi czasu. Wystarczy zatem wprowadzić oznaczenie:

$$P\left[X(t)=n\right]=P_n(t)$$

Zdefiniujmy, korzystając z wprowadzonego oznaczenia, „błyskawiczność" zachodzenia zgłoszeń, która oznacza, że:

$$P_1\left(\Delta t\right)=\lambda\Delta t+o(\Delta t)$$

gdzie:

$$\lim_{\Delta t\to 0}\frac{o\left(\Delta t\right)}{\Delta t}=0$$

inaczej:

$$\lim_{\Delta t\to 0}\frac{P_1(\Delta t)}{\Delta t}=\lambda$$

Załóżmy, że nasz proces losowy znajduje się w stanie $X(t)\geqslant 0$. Rozważmy prawdopodobieństwa możliwych przejść stanów procesu:

$$X(t)\to X(t+\Delta t),\Delta t\to 0$$

Zauważmy, że ze względu na *pojedynczość* zachodzenia zgłoszeń mamy tylko dwie możliwości o poniższych prawdopodobieństwach:

$P\left[X(t+\Delta t)=X(t)+1\right]=$ *P(Przybyło jedno zgłoszenie)* $=\lambda\Delta t+o(\Delta t)$

$P\left[X(t+\Delta t)=X(t)\right]=$ *P(Przybyło zero zgłoszeń)* $=1-\lambda\Delta t-o(\Delta t)$

Przeanalizujmy obecnie prawdopodobieństwo $P_n(t+\Delta t)$. Łatwo zauważyć, że:
$P_n(t+\Delta t) = \mathrm{P}($*w przedziale o długości t zaszło (n-1) zgłoszeń i w przedziale o długości* Δt *zaszło 1 zgłoszenie*$) + \mathrm{P}($*w przedziale o długości t zaszło n zgłoszeń i w przedziale o długości* Δt *zaszło 0 zgłoszeń*$)$

Zatem z założenia o *niezależności przyrostów*:

$$P_n\left(t+\Delta t\right) = P_{n-1}(t)\cdot\left(\lambda\Delta t + o(\Delta t)\right) + P_n(t)\left[1-\lambda\Delta t - o(\Delta t)\right]$$

Czyli:

$$P_n\left(t+\Delta t\right) = \lambda P_{n-1}(t)\Delta t + P_n(t) - \lambda P_n(t)\Delta t + o(\Delta t)$$

Dzieląc obustronnie przez Δt i przechodząc do granicy $\Delta t \to 0$ otrzymujemy zatem:

$$\frac{P_n(t+\Delta t - P_n(t)}{\Delta t} = \lambda P_{n-1}(t) - \lambda P_n(t) + \frac{o(\Delta t)}{\Delta t}$$

i przechodząc do granicy $\Delta t \to 0$

$$P_n'(t) = \lambda P_{n-1}(t) - \lambda P_n(t), n = 0, 1, 2, \ldots$$

Otrzymaliśmy zatem, dla wyznaczenia $P_n(t), n = 0, 1, 2, \ldots$, nieskończony układ równań różniczkowo-różnicowych, który możemy rozwiązać, ponieważ znamy wszystkie warunki początkowe:

Istnieją ogólne metody rozwiązywania takich równań, których jednak nie będziemy tu omawiać. Zamiast tego rozwiążemy nasze równanie dla początkowych wartości n.

- **n = 0**

  $$P_0'(t) + \lambda P_0(t) = 0, P_0(0) = 1$$

  Łatwo zauważyć, że rozwiązanie tego równania ma postać:

  $$P_0(t) = e^{-\lambda t}$$

- **n = 1**

  $$P_1'(t) = \lambda P_0(t) - P_0(t) - P_1(t), P_1(0) = 0$$

  Podstawiając wyznaczoną wyżej wartość $P_0(t)$ mamy:

  Łatwo zauważyć, że rozwiązanie tego równania ma postać:

  $$P_1(t) = \lambda t e^{-\lambda t}$$

Postępując analogiczne otrzymalibyśmy:

$$P_n(t) = \frac{(\lambda t)^n}{n!}e^{-\lambda t}, n = 0, 1, 2, \ldots$$

□

Uwaga: można wykazać, że założenie „błyskawiczności" można pominąć, gdyż wynika ono z pozostałych trzech założeń.

Z twierdzenia 2.1 wynika prawie natychmiast poniższy wniosek, który jednak z uwagi na jego znaczenie zapiszemy w postaci kolejnego twierdzenia.

Twierdzenie 2.2. *Dla stacjonarnego procesu Poissona odstęp czasowy pomiędzy* kolejnymi *zgłoszeniami ma rozkład wykładniczy.*

Dowód. Oznaczmy przez U zmienną losową, oznaczającą odstęp między kolejnymi zgłoszeniami. Łatwo zauważyć, że dystrybuanta tej zmiennej losowej wyraża się wzorem:

$$F_U(t) = P(U < t) = 1 - \underbrace{P(U \geqslant t)}_{=P_0(t)} = \textit{(z tw. 2.1)}\ 1 - e^{-\lambda t}$$

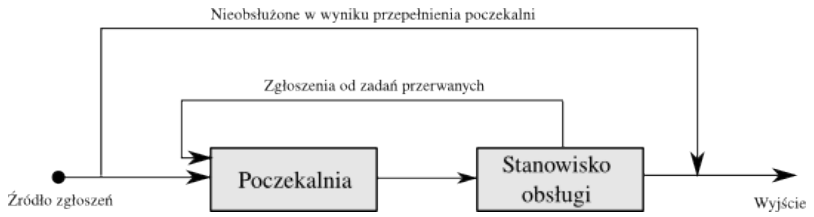
□

Uwaga: Ponieważ rozkład wykładniczy jest jedynym rozkładem zmiennej losowej typu ciągłego o własności braku pamięci, więc twierdzenie odwrotne jest również prawdziwe.

2.2 Podstawowe pojęcia teorii kolejek

Teoria kolejek (teoria masowej obsługi) rozwija się od roku 1909, w którym duński matematyk Agner K. Erlang opublikował pracę dotyczącą modelu matematycznego działania cetrali telefonicznej. Teoria ta znalazła następnie zastosowanie do modelowania ruchu w portach lotniczych i morskich, konserwacji maszyn, kontroli zapasów, ruchu ulicznego, a począwszy od lat '60 XX wieku, także obsługi procesów w systemach komputerowych.

W najprostszym ujęciu, istota problemu praktycznego polega na tym, że klienci (zadania) losowo generują zgłoszenia (żądania) obsługi (wykonywania) przez stanowisko obsługi (jednostkę zasobu). Widzimy zatem, że problem taki można zamodelować za pomocą sieci zadań, których ograniczenia kolejnościowe dane są w postaci rozkładu prawdopodobieństwa odstępu czasowego między kolejnymi żądaniami. Podkreślmy, że będziemy zamiennie i w zależności od kontekstu używali pojęć zgłoszenie – zadanie, obsługa – wykonywanie, regulamin (algorytm obsługi) – algorytm szeregowania.



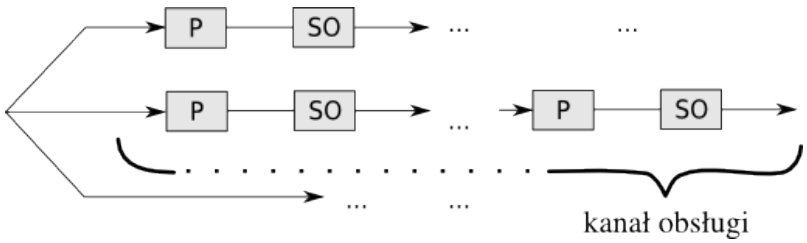
Rysunek 1: Najprostszy (jednostanowiskowy) system masowej obsługi (sieć zadań pojawiających się losowo z jedną jednostką zasobu)

Wykład 4 — 23.03.2009

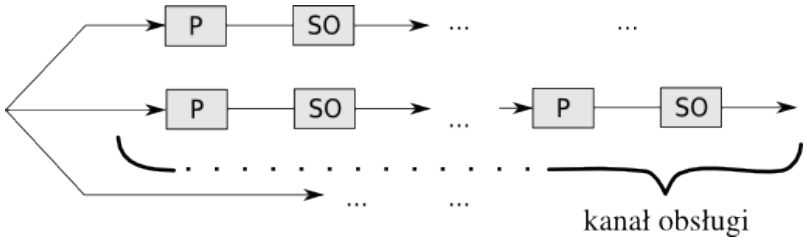
W poczekalni tworzy się *kolejka* (ang. *queue*) lub system kolejek. W ogólności rozpatruje się bardziej złożone systemy masowej obsługi, np. szeregowo-równoległe:

lub ogólne *sieci kolejkowe* – rys. 3 – (ang. *opening networks*) *otwarte* (ang. *opened*) lub *zamknięte* (ang. *closed*)

W dalszym ciągu zajmiemy się na razie jednostanowiskowym systemem masowej obsługi. Jest on scharakteryzowany przez:



Rysunek 2: System szeregowo-równoległy



Rysunek 3: Sieć kolejkowa

- wymiar źródła zgłoszeń[4]
- pojemność poczekalni
- rozkład wejścia – czyli rozkład (prawdopodobieństwa) odstępu U między kolejnymi zgłoszeniami
- rozkład czasu obsługi (wykonywania zadania) V
- regulamin (algorytm) obsługi (szeregowania)

Dla systemów masowej obsługi z algorytmem naturalnym (FIFO), złożonych z identycznych, równoległych stanowisk obsługi stosuje się często poniższą notację, zwaną *notacją Kendalla*:

$$X|Y|m|L|k$$

- **X** – rozkład wejścia U
- **Y** – rozkład obsługi V
- **m** – liczba równoległych stanowisk obsługi
- **L** – pojemność poczekalni (deterministyczna)
- **k** – wymiar źródła

gdzie literały X, Y przymują wartości opisujące konkretne rozkłady, np.:

- **D** – rozkład jednopunktowy
- **M** – rozkład wykładniczy

[4]Mówimy, że źródło zgłoszeń ma wymiar k jeśli może generować nowe zgłoszenia tylko wtedy, gdy aktualna liczba zgłoszeń w systemie (tzn. w poczekalni i na stanowisku obsługi) jest **mniejsza od** k.

- **G** – rozkład dowolny

Jeśli $L = k = \infty$ to notacja *Kendalla* ogranicza się do trzech pierwszych pól i ma postać:

$$X|Y|m$$

Przykładowo oznaczenie $M|M|1$ oznacza jednostanowiskowy system masowej obsługi z algorytmem FIFO, z nieograniczoną pojemnością poczekalni i nieskończenie wymiarowym źródłem zgłoszeń, oraz z wykładniczym rozkładem wejścia i wykładniczym rozkładem czasu obsługi. W dalszym ciągu, dopóki nie powiemy inaczej, będziemy rozpatrywali takie właśnie systemy masowej obsługi (z dokładnością do rozkładu czasu obsługi).

Założenie o wykładniczości rozkładu wejścia oznacza, jak pamiętamy, że na wejściu systemów mamy strumień zgłoszeń, będący stacjonarnym procesem Poissona. Strumień taki bywa nazywany *stacjonarnym strumieniem prostym*, lub krótko *strumieniem prostym*.

Uwaga: Notacja *Kendalla* bywa uogólniana do oznaczania szerszej klasy systemów masowej obsługi, ale wówczas wymaga dodatkowego komentarza.

Wprowadźmy kilka kolejnych oznaczeń:

średnią liczbę zgłoszeń przybywających w **jednostce czasu** nazywać będziemy *intensywnością strumienia zgłoszeń* (dla strumienia prostego pokrywa się ona z *parametrem strumienia* definiowanym jako $\lim_{\Delta t \to 0} \frac{P_1(\Delta t)}{\Delta t}$) i oznaczać przez λ.

Łatwo zauważyć, że:

$$\frac{1}{\lambda} = \overline{U} = \int_0^\infty u dF(u) = \int_0^\infty t dF_u(t)$$

Średnią liczbę zgłoszeń obsługiwanych w jednostce czasu oznaczymy przez μ. Łatwo zauważyć, że

$$\frac{1}{\mu} = \overline{V} = \int_0^\infty v dF(v) = \int_0^\infty t dF_v(t)$$

Stosunek λ do μ nazywamy intensywnością ruchu i oznaczamy przez ρ:

$$\rho = \frac{\lambda}{\mu}$$

Stan systemu masowej obsługi w chwili t, czyli liczbę zgłoszeń w tym systemie w chwili t, oznaczać będziemy przez $N(t)$. Jest to oczywiście w ogólności proces losowy, który zastąpi oznaczenie $X(t)$. Zatem $P_n(t) = P\left[N(t) = n\right]$.

Czas oczekiwania na obsługę (zmienna losowa) oznaczymy przez W.
Czas odpowiedzi (zmienna losowa) oznaczymy przez $T = W + V$.

W analizie systemów masowej obsługi wyróżnia się tzw. analizę wartości chwilowych oraz analizę w stanie równowagi statystycznej. W dalszym ciągu zajmiemy się tylko tą drugą analizą. Zakłada się w niej, że w systemie masowej obsługi został osiągnięty tzw. *stan równowagi statystycznej* (ang. *statistical equilibrium, steady state*). Czyli zostały osiągnięte następujące granice:

$$\lim_{t \to \infty} N(t) = N, \overline{N} < \infty$$

(granica jest zmienną losową, której wartość średnia jest skończona)

$$\lim_{t \to \infty} P_n(t) = p_n, n = 0, 1, 2, \ldots$$

(prawdopodobieństwa stacjonarne niezależne już od czasu – stan ustalony zwany **stanem równowagi statystycznej**)

Można wykazać, że przy bardzo ogólnych założeniach, spełnionych przez wszystkie rozpatrywane przez nas systemy, warunek dostateczny w osiąganiu stanu równowagi statystycznej ma postać:

$$\rho < m$$

albo

$$\lambda < m \cdot \mu$$

W szczególności dla $m = 1$ mamy $\rho < 1$ lub $\lambda < \mu$

Systemy obsługi, dla których ten warunek jest spełniony nazywa się stabilnymi. W systemach tych interesują nas trzy zmienne losowe: N, W, T, a w szczególności ich wartości średnie $\overline{N}, \overline{W}, \overline{T}$, między którymi zachodzą nastepujące związki:

$$\overline{T} = \overline{W} + \overline{V} = \overline{W} + \frac{1}{\mu}$$

(oczywisty z definicji)

$$\overline{N} = \lambda \overline{T}$$

(tzw. wzór *Little'a*)

Widzimy zatem, że w systemie stabilnym wszystkie wyżej wymienione **średnie są skończone**.

W systemach komputerowych ważne jest zapewnienie preferencji zadaniom krótkim, czyli zadaniom o małych wartościach $V = t$. Żeby taką preferencję móc opisać, definiuje się rozkłady warunkowe zmiennych losowych W i T pod warunkiem, że $V = t$. Rozkłady te są zatem opisane przez warunkowe dystrybuanty:

$$F_W(x|t) = P(W < x|V = t)$$
$$F_T(x|t) = P(T < x|V = t)$$

W szczególności interesują nas wartości średnie tych rozkładów $\overline{W}(t)$ i $\overline{T}(t)$, między którymi zachodzi oczywisty związek:

$$\overline{T}(t) = \overline{W}(t) + t$$

Wykład 5 30.03.2009

2.3 Podstawowe algorytmy (regulaminy) szeregowania (obsługi)

2.3.1 Algorytm FIFO

W systemach obsługi z algorytmem FIFO (*first in first out*, FCFS = *first come first served*, algorytm naturalny) zgłoszenia są obsługiwane w kolejności przychodzenia do systemu *bez przerwań*. Nadejście zgłoszenia od nowego zadania nie przerywa zadania aktualnie wykonywanego.
Rozpatrzymy system $M|G|1$, dla którego wyznaczymy wszystkie interesujące nas średnie w stanie równowagi statystycznej. Zacznijmy od wyprowadzenia wzoru na $\overline{N}$[5].
Oznaczmy przez N_1, N_2 liczby zgłoszeń w systemie bezpośrednio po obsłużeniu dwóch kolejnych zgłoszeń z_1, z_2 (wykonaniu dwóch kolejnych zadań) oraz przez R liczbę zgłoszeń, które nadeszły w czasie obsługiwania zgłoszenia z_2 (w czasie V).
Łatwo zauważyć, że:

[5] średnie N

Definicja 2.4. *Jeśli* $N_1 = 0$*, to* $N_2 = R$*, natomiast jeśli* $N_1 \neq 0, to N_2 = N_1 + R - 1$ *(*z_2 *wzięte z* N_1*)*

Wprowadźmy teraz zmienną losową *bool'owską* Γ wzorem:

$$\Gamma = \begin{cases} 1 & N_1 = 0 \\ 0 & w.p.p. \end{cases}$$

Korzystając z tej zmiennej losowej możemy napisać wzory powyżej (2.4) za pomocą jednego związku:

$$N_2 = N_1 + R - 1 + \Gamma$$

Weźmy wartość średnią obu stron:

$$\overline{N}_2 = \overline{N}_1 + \overline{R} - 1 + \overline{\Gamma}(*)$$

Ponieważ rozpatrujemy system w stanie równowagi statystycznej, więc $\overline{N}_2 = \overline{N}_1$

Wyznaczymy obecnie $\overline{R}$.
Ponieważ zmienna losowa R zależy od zmiennej losowej V. W celu wyznaczenia $\overline{R}$ (lub każdego innego momentu), należy to uczynić dwuetapowo:

1. Liczymy $\overline{R}$ dla ustalonego $V = v$, co oznaczymy jako $\overline{R}(v) = \lambda v$
2. Uśredniamy otrzymany wynik po rozkładzie zmiennej losowej V, czyli

$$\overline{R} = \int_0^\infty \lambda v dF(v) = \lambda \int_0^\infty v dF(v) = \lambda \overline{V} = \frac{\lambda}{\mu} = \rho$$

Wracając do wzoru (*) i zakładając stan równowagi statystycznej, otrzymujemy zatem: $\overline{\Gamma} = 1 - \overline{R} = 1 - \rho$
Podnieśmy teraz wzór (*) obustronnie do kwadratu:

$$N_2^2 = N_1^2 + 2N_1R + R^2 - 2N_1 + 2N_1\Gamma - 2R + 2R\Gamma + 1 - 2\Gamma + \Gamma^2$$

Z definicji: $2N_1\Gamma = 0$ oraz $\Gamma^2 = \Gamma$ (??)
Wyliczmy wartość średnią obu stron powyższego wzoru:

$$\overline{N_2^2} = \overline{N_1^2} + 2\overline{N_1R} + \overline{R}^2 - 2\overline{N}_1 + 2N_1\Gamma - 2\overline{R} + 2\overline{R\Gamma} + 1 - 2\Gamma + \Gamma^2$$

W stanie równowagi statystycznej oczywiście:

$$\overline{N_2^2} = \overline{N_1^2}$$

Zauważmy ponadto, że zmienne losowe R i N_1, a w konsekwencji także R i Γ są zależne. Możemy zatem napisać:

$$0 = 2\overline{N_1R} + \overline{R^2} - 2\overline{N_1} - 2\overline{R} + 2\overline{R\Gamma} + 1 - \overline{\Gamma}$$

Podstawiając wartości otrzymane na $\overline{R}$ i $\overline{\Gamma}$ otrzymujemy zatem:

$$0 = 2\overline{N_1}\rho + \overline{R^2} - 2\overline{N_1} - 2\rho + 2\rho(1-\rho) + 1 - \rho$$

Zatem:

$$2\overline{N_1} - 2\overline{N_1}\rho = \overline{R^2} - 2\rho + 2\rho(1-\rho) + \rho$$

$$2\overline{N_1}(1-\rho) = R^2 - \rho - \rho + 2\rho(1-\rho) + \rho | : 2(1-\rho)$$

$$\overline{N_1} = \overline{N} = \frac{\overline{R^2} - \rho}{2(1-\rho)} + \rho$$

Pozostaje zatem wyznaczenie $\overline{R^2}$, co znowu zrobimy dwuetapowo:

1. Wyliczymy $\overline{R^2}$ dla ustalonego $V = v$.
   Ponieważ na wejściu mamy stacjonarny strumień prosty (Poissona), więc prawdopodobieństwo tego, że w ustalonym czasie v nadejdzie r zgłoszeń wyraża się wzrorem: $\frac{(\lambda v)^r}{r!}e^{-\lambda v}$.
   Zatem
   $$\overline{R^2}(v) = \sum_{r=1}^{\infty} r^2 \frac{(\lambda v)^r}{r!} e^{-\lambda v} = e^{-\lambda v} \sum_{r=1}^{\infty} r^2 \frac{(\lambda v)^r}{r!} = \lambda v + \lambda^2 v^2$$

2. Uśrednimy obecnie otrzymany wynik względem rozkładu zmiennej losowej V: w
   $$\overline{R^2} = \int_0^{\infty} \left(\lambda v + \lambda^2 v^2\right) dF(v) = \lambda \int_0^{\infty} v dF(v) + \lambda^2 \int_0^{\infty} v^2 dF(v) = \rho + \lambda^2 \sigma_V^2 + \rho^2$$

   Podstawiając ten wynik do wzoru na $\overline{N}$ otrzymujemy:

   $$\overline{N} = \frac{\rho^2 + \lambda^2 \sigma_V^2}{2(1-\rho)} + \rho$$

Korzystając z tego wyniku, łatwo wyliczymy pozostałe interesujące nas średnie:
Ze wzoru *Little'a* mamy:

$$\overline{N} = \lambda \overline{T} = \lambda(\overline{V} + \overline{W}) = \lambda \overline{V} + \lambda \overline{W} | : \lambda \overline{V} = \rho$$

Czyli:

$$\frac{\overline{W}}{\overline{V}} = \frac{\overline{N}}{\rho} - 1 = \frac{\rho}{2(1-\rho)} \left(1 + \mu^2 \sigma_V^2\right)$$

(wzór *Pollaczka*)
Otrzymujemy zatem:

$$\overline{W} = \frac{\rho}{2\mu(1-\rho)} \left(1 + \mu^2 \sigma_V^2\right) = \frac{\lambda}{2\mu^2(1-\rho)} \left(1 + \mu^2 \sigma_V^2\right) = \frac{\lambda \overline{V^2}}{2(1-\rho)}$$

Ponieważ algorytm FIFO nie preferuje żadnych zadań, więc:

$$\overline{W}(t) = \overline{W}$$

Natomiast:

$$\overline{T} = \overline{W} + \overline{V} = \frac{\lambda \overline{V^2}}{2(1-\rho)} + \frac{1}{\mu}$$

Oraz:

$$\overline{T}(t) = \overline{W}(t) + t = \overline{W} + t$$

Rozpatrzymy teraz dwa przypadki szczególne rozkładu zmiennej losowej:

- **System** $M|D|1$ – oznacza to, że $P(V = v_0) = 1$.
  Oczywiście $\overline{V} = v_0, {\sigma_V}^2 = 0$
  $\overline{W} = \dfrac{\rho v_0}{2(1-\rho)}, \overline{T} = \overline{W} + v_0$

Wówczas $\overline{V} = \dfrac{1}{\mu}, \sigma_V^2 = \dfrac{1}{\mu^2}$ (wyliczyć!!!), zatem:

$$\overline{W} = \frac{\rho}{\mu(1-\rho)}, \overline{T} = \overline{W} + \frac{1}{\mu} = \frac{1}{\mu(1-\rho)}$$

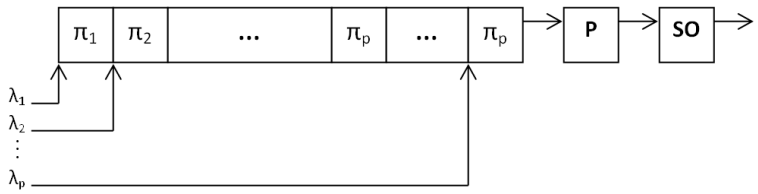
Porównajmy obecnie system $M|D|1$ z systemem $M|M|1$, zakładając, że w obu z nich V jest takie samo. Dzieląc obustronnie wzory *Pollaczka* dla obu z tych systemów, otrzymujemy:

$$\frac{\overline{W}_{M|M|1}}{\overline{W}_{M|D|1}} = 2$$

Wykład 6 06.04.2009

2.3.2 Algorytmy ze stałymi priorytetami

W algorytmach tych (*head of the line priority scheduling algorithm*) zadanie przychodzące otrzymuje określony priorytet, dołącza do podkolejki o tym priorytecie i pozostaje w niej aż do zakończenia. Kolejka jest podzielona na podkolejki odpowiadające priorytetom i jest obbsługiwana według algorytmu FIFO.
Załóżmy, że mamy p priorytetów $\Pi_i, i = 1, 2, \ldots, p$, odpowiadających p strumeniom wejściowym prostym o intensywności λ_i i p-rozkładu czasu obsługi o średnich $\dfrac{1}{\mu_i}, i = 1, 2, \ldots, p$. System obsługi ma zatem postać:



Rysunek 4: System obsługi ze stałymi priorytetami

Zauważmy, że w takim systemie warunek równowagi statystycznej przyjmuje postać zbioru warunków:

$$\rho_i + \rho_{i+1} + \ldots + +\rho_p < 1, i = 1, 2, \ldots, p-1$$

gdzie

$$\rho_i = \frac{\lambda_i}{\mu_i}, i = 1, 2, \ldots, p-1$$

Można wykazać, że dla omawianego algorytmu słuszny jest następujący wzór:

$$\overline{W}_i(t) = \overline{W}_i = \frac{\overline{W}_0}{(1-\gamma_i)(1-\gamma_{i+1})}, i = 1, 2, \ldots, p-1$$

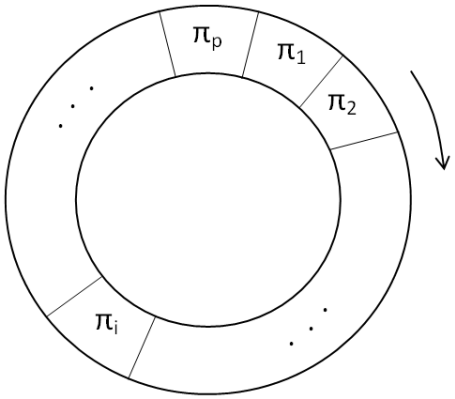
gdzie

$$\overline{W}_0 = \frac{1}{2}\sum_{j=1}^{p} \lambda_j \overline{V_j^2}$$

$$\gamma_i = \sum_{j=1}^{p} \rho_j$$

2.3.3 Algorytmy cykliczne (karuzelowe)

Dla systemów obsługi z tymi algorytmami charakterystyczna jest technika przerwań: zadanie jest przerywane w trakcie wykonywania, a SO (procesor) przechodzi do obsługi kolejnego zadania w kolejce, itd. Kolejka ma zatem charakter cykliczny.



Rysunek 5: Algorytm cykliczny

W ogólności nowo przychodzące zadanie o priorytecie Π_i dołączy na koniec kolejki Π_i. ??
W systemach komputerowych stosowane są dwa typy algorytmów cyklicznych. W pierwszym z nich przerwanie (wykonywania) zadania następuje po zarządaniu przez nie operacji wejścia/wyjścia. Oczywiście system z takim algorytmem preferuje zadania rzadko korzystające z tych urządzeń, jednak jakiekolwiek wzory analityczne są tu raczej trudne do uzyskania.
Znacznie częściej stosowany jest drugi tym algorytmu cyklicznego, zwany algorytmem RR (*round robin*) (UWAGA: wielu autorów mówiąc algorytm cykliczny (karuzelowy) mają na myśli właśnie ten algorytm).
W systemie obsługi z algorytmem RR przerwanie zadania następuje po upływie kwantu czasu Δ_i, który w systemach komputerowych jest parametrem systemu operacyjnego.
Rozpatrzymy najpierw system, który nie ma priorytetów. W analizie teoretycznej zakłada się, że kwant czasu dąży do zera, co pozwala zaniechać w analizie inne zdarzenia (zakończenie zadania, żądanie operacji wejścia-wyjścia), które zbiegają się z czasem zakończenia tego kwantu witrualnego. Zakładając ponadto, że rozkład czasu obsługi jest wykładniczy, można wykazać, że:

$$\overline{W}_i(t) = \frac{\rho t}{1-\rho}$$

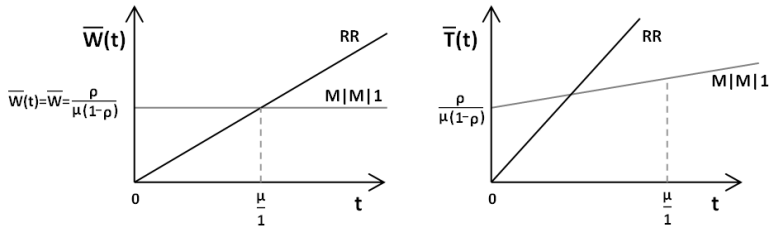
$$\overline{T}_i(t) = \frac{t}{1-\rho}$$

Jak widać system z tym algorytmem zapewnia *liniową preferencję* zadań w funkcji ich długości (zadanie dwukrotnie dłuższe spędzi średnio w kolejce (systemie) n-razy więcej czasu). Zauważmy, że bezwarunkowe wartości średnie w takim systemie (już bez wymogu wykładniczości czasu obsługi) mają postać:

$$\overline{W} = \frac{\rho}{\mu(1-\rho)}$$

$$\overline{T} = \frac{1}{\mu(1-\rho)}$$

Średnie te są zatem identyczne jak w systemie $M|M|1$.
Porównajmy teraz system z algorytmem RR z systemem $M|M|1$ z punktu widzenia średnich warunkowych.



Rysunek 6: Porównanie systemu z algorytmem RR z z systemem $M|M|1$ z punktu widzenia średnich warunkowych.

System z algorytmem RR preferuje zadanie o długości mniejszej od $\frac{1}{\mu}$, a dyskryminuje zadanie o długości większej od tej wartości.
Załóżmy teraz, że kolejka jest podzielona na podkolejki o priorytetach $i = 1, 2, \ldots, p$ i jest obsługiwana cyklicznie, przy czym kwant czasu w podkolejce o priorytecie Π_i wyraża się wzorem:

$$\Delta_i = \frac{c_i}{\sum_{j=1}^{p} c_j \overline{N}_j}$$

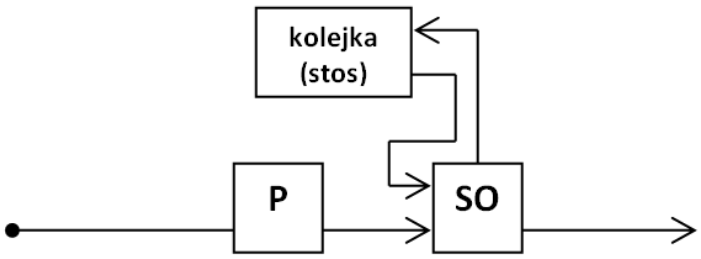
gdzie c_i jest stałą dodatnią, a $\overline{N}_j$ jest średnią liczbą zadań o priorytecie Pi_j w stystemie.
Można wykazać, że w systemie z takim algorytmem słuszne są następujące wzory:

$$\overline{W}_i(t) = \frac{\rho t}{1-\rho}\left[1 + \sum_{j=1}^{p}\left(\frac{c_j}{c_i} - 1\right)\frac{\rho_j}{\rho}\right], i = 1, 2, \ldots, p$$

$$\overline{T}_i(t) = \frac{t}{1-\rho}\left[1 + \sum_{j=1}^{p}\left(\frac{c_j}{c_i} - 1\right)\rho_j\right], i = 1, 2, \ldots, p$$

2.3.4 Algorytm LIFO

W systemie z algorytmem LIFO (*Last in last out. LCFS - last come, first served* zadanie przychodzące natychmiast otrzymuje obsługę. Jeśli w tym czasie było wykonywane inne zadanie to zostaje ono przerwane (w sensie wykonywania) i umieszczone na początku kolejki. Po stwierdzeniu wykonania zadania jest ono usuwane z systemu, a SO (procesor) przechodzi do obsługi pierwszego zadania w kolejce (czyli zadania ostatnio przerwanego). Widzimy zatem, że kolejka jest zorganizowana w postaci stosu (*stack*).



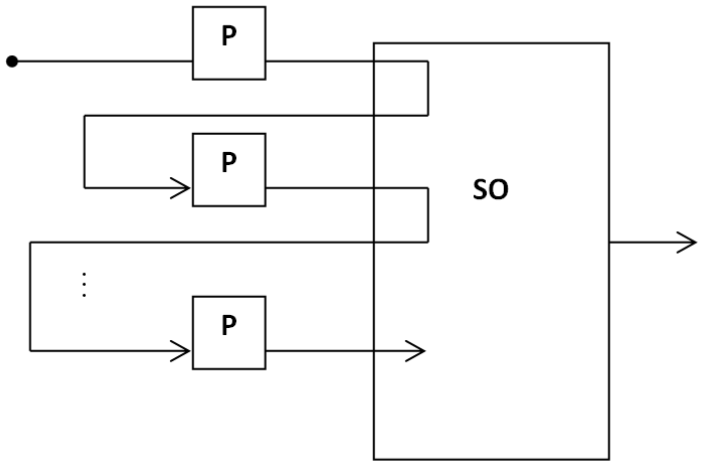
Rysunek 7: Organizacja algorytmu LIFO

Można wykazać (przy założeniu wykładniczości rozkładu czasów obsługi, że wzory na $\overline{W}(t), \overline{T}(t), \overline{W}, \overline{T}$ są identyczne jak dla systemu z algorytmem RR (bez priorytetów). Jednak wariancje odpowiednich rozkładów są tu większe, niż w systemie z algorytmem RR, a zatem system z algorytmem LIFO ma gorsze własności, niż ten ostatni (RR).

Uwaga: Zauważmy, że w dotychczas rozpatrywanych algorytmach preferencja zadań w funkcji ich długości (jeśli istniała), była liniowa. Nie zawsze pozwala to uzyskać maksymalne wykorzystanie mocy obliczeniowej systemu. Stosuje się w takich wypadkach algorytmy zapewniające nieliniową preferencję zadań funkcji ich długości. Obszerną klasę takich algorytmów są np. *algorytmy wielokolejkowe* (*multilevel processor sharing scheduling algorithm*, zorganizowane następująco:

Zadanie przychodzące umieszczane jest na końcu kolejki o najwyższym priorytecie, jest obsługiwane przez kwant czasu obowiązujący w tej kolejce, przy czym jeśli się nie zakończy, spada do kolejki o priorytecie bezpośrednio mniejszym, itd.
Jednym z często stosowanych algorytmów wielokolejkowych jest *algorytm dwukolejkowy*, zwany też algorytmem FB (*foreground–background*), w którym pierwsza kolejka jest obsługiwana według algorytmu FIFO (z kwantem czasu!!!), a druga według algorytmu RR. Można wykazać, że otrzymujemy wówczas typ preferencji opisany poniższą krzywą:

Zauważmy, że w systemach z algorytmami wielokolejkowymi, zadania długie są silnie dyskryminowane i nie zawsze wystarcza fakt, że kwant czasu w tych systemach rośnie wraz z maleniem priorytetu. Żeby temu zapobiec, stosuje się w tych systemach pewne dodatkowe mechanizmy, np. tasssk zwaną częściową, czy generalną amnestię.



Rysunek 8: Algorytm wielokolejkowy

Wykład 7 20.04.2009

2.3.5 Systemy obsługi z momentami przybycia i/lub czasami wykonywania zadań zależnymi od stanu

Zacznijmy od wyprowadzenia wzorów na prawdopodobieństwa stacjonarne dla systemów M—M—1. Załóżmy, że system jest w stanie $N(t) > 0$ i oceńmy prawdopodobieństwa przejścia $N(t + \Delta t \rightarrow N(t)$ dla $\Delta t \rightarrow 0$. Załóżmy, że prawdopodobieństwo przybycia zadania dla $\Delta t \rightarrow 0$ wynosi $\lambda\Delta t + o(\Delta t)$ oraz, że prawdopodobieństwo opuszczenia systemu przez zadanie w tym przedziale wynosi $\mu\Delta t + o(\Delta t)$.
Zatem prawdopodobieństwo możliwych przejść stanów wynoszą:

- P(1 przybycie i 0 opuszczeń)

$$P\left[N(t+\Delta t) = N(t)+1\right] = \left[\lambda\Delta t + o(\Delta t)\right]\left[1 - \mu\Delta t - o(\Delta t)\right] = \lambda\Delta t + o(\Delta t)$$

- P(0 przybyć i 0 opuszczeń lub 1 przybycie i 1 opuszczenie)

$$\begin{aligned} P\left[N(t+\Delta t) = N(t)\right] = \\ = \left[1 - \lambda\Delta t - o(\Delta t))\right]\left[1 - \mu\Delta t - o(\Delta t)\right] + \left[\lambda\Delta t + o(\Delta t)\right]\left[\mu\Delta t + o(\Delta t)\right] = \\ = 1 - (\lambda + \mu)\Delta t \end{aligned}$$

- P(0 przybyć i 1 opuszczenie)

$$P\left[N(t+\Delta t) = N(t)-1\right] = \left[1 - \lambda\Delta t - o(\Delta t)\right]\left[\mu\Delta t + o(\Delta t)\right] = \mu\Delta t + o(\Delta t)$$

Zatem (z uwagi na brak pamięci rozkładu wykładniczego na wejściu i *wyjściu*):

$$P_n(t+\Delta t) = P_{n-1}(t)\cdot\left[\lambda\Delta t + o(\Delta t)\right] + P_n(t)\cdot\left[1 - (\lambda+\mu)\Delta t + o(\Delta t)\right] + P_{n+1}(t)\cdot\left[\mu\Delta t + o(\Delta t)\right]$$

czyli:

$$P_n(t+\Delta t) = \lambda P_{n-1}(t)\Delta t + P_n(t) - (\lambda+\mu)P_n(t)\Delta t + \mu P_{n+1}(t)\Delta t + o(\Delta t)$$
$$\frac{P_n(t+\Delta t) - P_n(t)}{\Delta t} = \lambda P_{n-1}(t) - (\lambda+\mu)P_n(t) + \mu P_{n+1}(t) + \frac{o(\Delta t)}{\Delta t}$$

przechodząc do granicy przy $\Delta t \to 0$ otrzymujemy zatem:

$$P_n'(t) = \lambda P_{n-1}(t) - (\lambda+\mu)P_n(t) + \mu P_{n+1}(t), n = 1, 2, \ldots$$

Zauważmy, że dla $n = 0$ musimy uwzględnić możliwość opuszczenia systemu w drugim składniku, a zatem otrzymujemy równanie:

$$P_0'(t) = \mu P_1(t) - \lambda P_0(t)$$

Zauważmy, że jeśli przyjmiemy $\mu = 0$ (tylko przybywanie) to otrzymamy to samo, co wyprowadzono poprzednio (dla FIFO) (??)

Załóżmy teraz, że w systemie został osiągnięty stan równowagi statystycznej, czyli $P_n(t) = p_n, n = 0, 1, 2, \ldots$ ($P_n' = 0, n = 0, 1, 2, \ldots$). Mamy zatem:

$$\lambda p_{n-1} - (\lambda+\mu)p_n + \mu p_{n+1} = 0, n = 1, 2, \ldots$$
$$\mu p_1 - \lambda p_0 = 0$$

Rozwiążmy powstały układ równań, poczynając od drugiego z nich, stąd mamy:

$$p_1 = \frac{\lambda_0}{\mu_1} p_0$$
$$p_2 = \frac{\lambda_0}{\mu_1}\frac{\lambda_1}{\mu_2} p_0$$

Postępując tak można wykazać, że:

$$p_u = p_0 \prod_{i=1}^{n} \frac{\lambda_i - 1}{\mu_i}$$

Dodatkowe równanie otrzymujemy z warunku:

$$\sum_{n=0}^{\infty} p_n = 1$$

zapisując je w postaci:

$$p_0 + \sum_{n=0}^{\infty} p_n = 1$$

i podstawiając wyżej wyliczoną wartość p_n mamy ostatecznie:

$$p_0 = \left[1 + \sum_{n=1}^{\infty}\prod_{i=1}^{n}\frac{\lambda_{i-1}}{\mu_i}\right]^{-1}$$

Wyznaczyliśmy zatem wszystkie prawdopodobieństwa stacjonarne, a zatem rozkład prawdopodobieństwa stanu naszego systemu w stanie równowagi statystycznej. Na tej podstawie możemy wyznaczyć wszelkie parametry tego rozkładu, a więc w szczególności wartość średnią:

$$\overline{N} = \sum_{n=1}^{\infty} n p_n$$

Dalej ze wzoru *Little'a* możemy wyliczyć:

$$\overline{T} = \frac{\overline{N}}{\lambda}$$

Uwaga: Powyższe rozważanie jest słuszne dla dowolnych *poisson'owskich* (na wejściu i na wyjściu) systemów obsługi z algorytmami obsługi niezależnymi od czasu wykonywania zadania. A zatem dla wszystkich dotychczas rozpatrywanych systemów z wyjątkiem systemu z algorytmem ze stałymi priorytetami, których priorytety zależą od czasu wykonywania zadań.

Zauważmy ponadto, że otrzymane równania na prawdopodobieństwa stacjonarne można natychmiastowo uogólnić na przypadek, gdy momenty przybycia zadań i/lub ich czasy wykonywania zależą od stanu n. Oznaczając bowiem odpowiednie parametry występujące w tych równaniach, indeksem wskazującym na stan otrzymujemy:

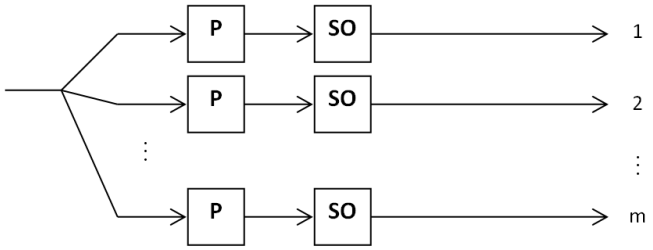
$$\lambda n - 1 p_{n-1} - (\lambda_n + \mu_n) p_n + \mu_{n+1} p_{n+1} = 0, n = 1, 2, \ldots$$
$$\mu_1 p_1 = \lambda_0 p_0 = 0$$

Rozpatrzymy teraz kilka przypadków szczególnych istotnych w systemach komputerowych:

Systemy typu $M|M|1|L$ Zauważmy, że teraz (pojemność poczekalni $L < \infty$ mamy:

$$\lambda_u = \begin{cases} \lambda & n = 0, 1, 2, \ldots, L \\ 0 & n > L \end{cases}$$

Podstawiając powyższe do ogólnych wzorów na p_n moglibyśmy wyliczyć np. $\overline{N}$ i $\overline{T}$.



Rysunek 9: System z kilkoma stanowiskami obsługi

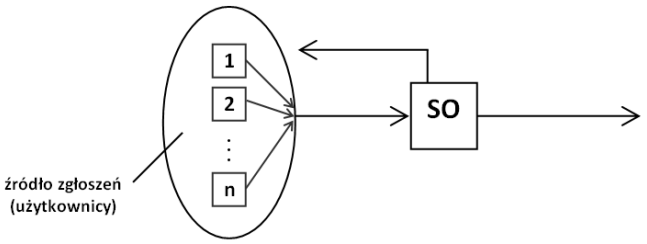
Systemy typu $M|M|m$ Łatwo zauważyć, że w tej sytuacji $\lambda_n = \lambda, n = 0, 1, 2, \ldots$

$$\mu_n = \begin{cases} n \cdot \mu & 1 \leqslant n \leqslant m \\ m \cdot \mu & n \geqslant m \end{cases}$$

Podstawiając to do wzoru na p_n możemy wyliczyć np. $\overline{N}$ i $\overline{T}$.

Systemy typu $M|M|1|\infty|k$ (czyli ze skończenie wymiarowym źródłem zgłoszeń) Rozpatrzymy system komputerowy, w którym jedno stanowisko obliczeniowe obsługuje k-końcówek użytkowników w trybie interaktywnym. Jak wiadomo, w systemach takich jeśli z danej końcówki zostanie wysłane zgłoszenie, to po wykonaniu odpowiedniego zadania zostaje do tej końcówki wysłana odpowiedź i po pewnym czasie, zwanym czasem zastanawiania się (ang. *thinking time*) może z tej końcówki znaleźć kolejne zgłoszenie. Przyjmując wykładniczy rozkład czasu zastanawiania się o średniej $\frac{1}{\lambda}$ otrzymamy szczególny przypadek rozważanych w tym punkcie (3.5) systemów.

Przedtem jednak zauważmy, że mamy do czynienia wtedy istotnie z k-wymiarowym źródłem zgłoszeń.



Wykład 8 27.04.2009

Zauważmy, że w rozpatrywanej sytuacji:

$$\lambda_n = \begin{cases} (k-n) & n < k \\ 0 & n \geqslant k \end{cases}$$

Podstawiając powyższe do podanych wzorów, otrzymujemy:

$$\overline{T} = \frac{k}{\mu(1-p_0)} - \frac{1}{\lambda}$$

gdzie

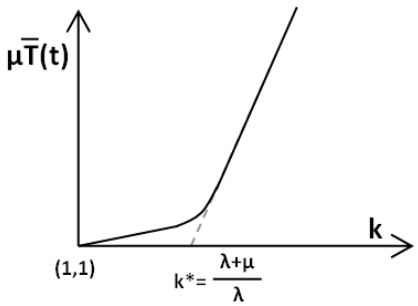
$$p_o = \left[\sum_{i=0}^{k} \frac{k!}{(k-i)!} \rho^i \right]^{-1}$$

Wynika stąd wykres charakteryzujący nasz system obsługi:

Zauważmy, że dla $k = 1$, $\overline{W} = 0$, $\overline{T} = \overline{V} = \frac{1}{\mu}$. Widzimy zatem, że dla skończonego k, nasz system nigdy nie wejdzie w nasycenie ($\overline{T} < \infty$), jednak dla $k \geqslant k^* = \frac{\lambda + \mu}{\lambda}$ i zwanego liczbą nasycenia następuje *gwałtowne* pogorszenie się jakości pracy systemu.
Projektując zatem taki system należy unikać przekraczania k^* jako liczby użytkowników (końcówek).

Uwaga: Jeśli w naszym systemie zastosować algorytm RR, to można wykazać (por. Kleinrock-Queuing Systems Theory vol.2):

$$\overline{T}(t) \approx \mu t \overline{T}$$



Rysunek 10: Wykres charakteryzujący omawiany system obsługi

3 Deterministyczne problemy szeregowania zadań

3.1 Podstawowe pojęcia, założenia i ich interpretacje

Rozpatrzymy najpierw zbiór zadań $\mathcal{Z} = \{Z_1, Z_2, \ldots, Z_n\}$ oraz zbiór maszyn $\mathcal{M} = \{M_1, M_2, \ldots M_m\}$ Założymy, że zbiór $\mathcal{M}$ jest zbiorem maszyn równoległych, tzn. każda maszyna z tego zbioru może wykonać każde zadanie ze zbioru $\mathcal{Z}$.
Przyjmiemy również dwa podstawowe założenia klasycznej (deterministycznej) teorii szeregowania:

1. każde zadanie ze zbioru $\mathcal{Z}$ może być w każdej chwili wykonywane na co najwyżej jednej maszynie

2. każda maszyna ze zbioru $\mathcal{M}$ może w każdej chwili wykonywać co najwyżej jedno zadanie

Poniżej scharakteryzujemy bliżej oba powyższe zbioru, zaczynając od zbioru $\mathcal{Z}$.

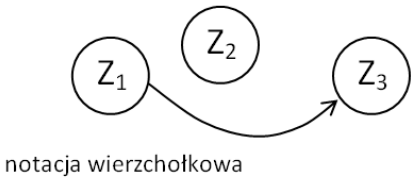
Powiemy, że $\mathcal{Z}$ jest zbiorem zadań *podzielnych* (*splittable*, *preemtable*), jeśli każde zadanie z tego zbioru (a dokładniej jego wykonywanie) może być przerwane w dowolnej chwili i następnie wznowione, *bez straty czasu*, być może na innej maszynie. Jeśli żadne zadanie ze zbioru $\mathcal{Z}$ nie może zostać przerwane to powiemy o zbiorze zadań *niepodzielnych* (*non-splittable*, *non-preemtable*).

Jeśli w zbiorze zadań $\mathcal{Z}$ wszystkie zadania mogą być wykonywane jednocześnie, tzn. w zbiorze tym nie jest zdefiniowane żadne ograniczenie kolejnościowe, to powiemy o zbiorze zadań niezależnych. W przeciwnym razie będziemy mówili o zbiorze zadań zależnych (*dependant*). Będziemy mówili, że pomiędzy zadaniami Z_j, Z_k jest zdefiniowane *ograniczenie kolejnościowe* (*precedence constraint*), co zapisujemy $Z_j \prec Z_k$ (poprzedza), jeśli zadanie Z_j musi się zakończyć przed rozpoczęciem zadania Z_k. Widzimy zatem, że zbiór zadań zależnych $\mathcal{Z}$ jest częściowo uporządkowany przez relację binarną $\{\mathcal{Z}, \prec\}$. W przypadku, gdy zbiór zadań jest zbiorem zadań niezależnych $\prec = \varnothing$.

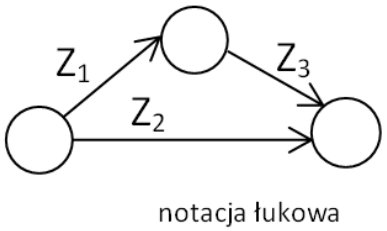
Relację poprzedzania przedstawia się na ogół w postaci grafów skierowanych (*digrafów*) w konwencji wierzchołkowej lub łukowej. W *konwencji wierzchołkowej* (*task-on-mode*) zadania odpowiadają wierzchołkom, a łuki ograniczają kolejnościowo, natomiast w *konwencji łukowej* (*task-on-arc*) zadania odpowiadają łukom, a wierzchołki zdarzeniom czasowym typu *and* (zadania wychodzące z takiego wierzchołka mogą się rozpocząć po zakończeniu *wszystkich* zadań dochodzących do niego).

By wyjaśnić obie te konwencje, rozważmy prosty przykład:

$$\mathcal{Z} = \{Z_1, Z_2, Z_3, \}$$
$$\prec: Z_1 \prec Z_3$$



Rysunek 11: Konwencja (notacja) wierzchołkowa



Rysunek 12: Konwencja (notacja) łukowa

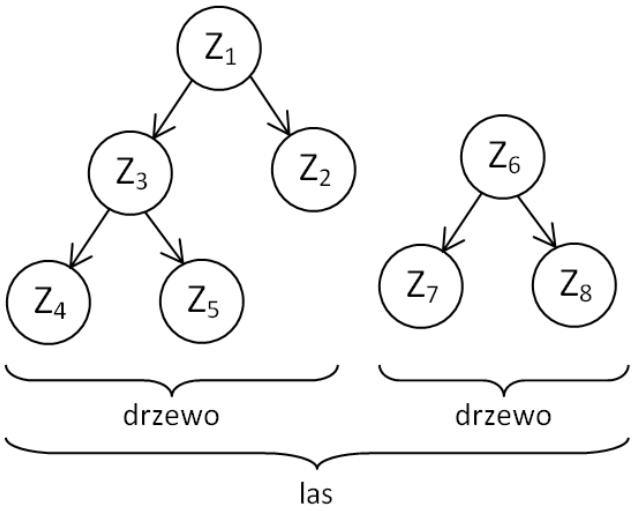
W dalszym ciągu, o ile nie powiemy inaczej, będziemy stosowali konwencję wierzchołkową. W konwencji tej wyróżnimy kilka struktur:

- Lasem (*out-forest*) nazywamy *digraf*, w którym każdy wierzchołek ma co najwyżej jeden, bezpośredni poprzednik
- Antylasem (*in-forest*) nazywamy *digraf*, w którym każdy wierzchołek ma co najwyżej jeden, bezpośredni nastepnik
- Drzewem (*out-tree*) nazywamy *las*, w którym istnieje jeden wierzchołek bez poprzedników
- Antydrzewem (*in-tree*) nazywamy *antylas*, w którym istnieje jeden wierzchołek bez następników

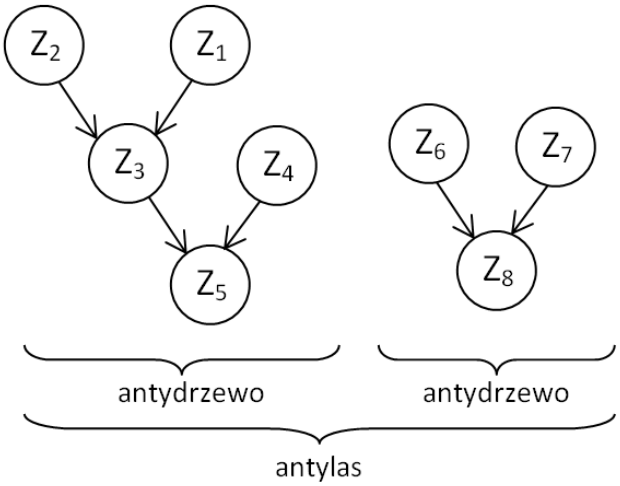
Proste przykłady są następujące:

Uwagi: *Las* i *antylas* bywają definiowane odwrotnie. Ponadto pojęcie *lasu* definiuje się również dla grafów nieskierowanych: *lasem* jest tu dowolny graf acykliczny, a *drzewem* graf acykliczny spójny.

Dla każdego zadania $Z_j \in \mathcal{Z}$ zdefiniowane są w ogólności:



Rysunek 13: Przykładowy *las*



Rysunek 14: Przykładowy *antylas*

- moment przybycia (gotowości – *arrival, ready time*) $0 \leqslant r_j < \infty$
- wektor czasu wykonywania $[\tau_{1j}, \tau_{2j}, \ldots, \tau_{mj}]^T$ gdzie $0 < \tau_{ij} < \infty$ jest czasem wykonywania zadania (*processing, execution time*) Z_j na maszynie M_i.
- waga = priorytet (*weight, priority*) $1 \leqslant w_j < \infty$
- żądany termin zakończenia wykonywania (*due date*) $0 < d_j \leqslant \infty$ – jeśli termin ten nie może

być przekroczony, to przy tym samym oznaczeniu nazwiemy go linią krytyczną (*deadline*)

Uwaga: deterministyczna teoria szeregowania obejmuje również problemy, w których $r_j, j = 1, 2, \ldots n$ nie są znane *a priori*.

Wykład 9 — 04.05.2009

sztywny czas rzeczywisty, szeregowanie LMAX

W tej sytuacji zachodzi konieczność konstrukcji algorytmów szeregowania *on-line*, nie wykorzystujących żadnej a priorycznie wiedzy o parametrach zadań.

Przejdźmy obecnie do scharakteryzowania zbioru maszyn $\mathcal{M}$. Powiemy o zbiorze maszyn *identycznych* (*identical*) jeśli:

$$\forall_{Z_j \in Z} \tau_{ij} = \tau_j, i = 1, 2, \ldots, m$$

Powiemy o zbiorze maszyn *jednorodnych* (*uniform*), jeśli:

$$\forall_{Z_j \in Z} \tau_{ij} = \frac{\tau_j}{b_i}, i = 1, 2, \ldots, m$$

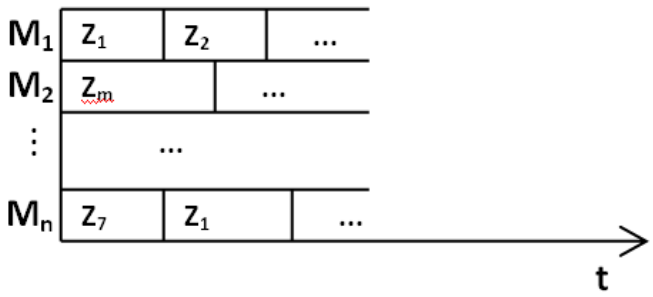
gdzie τ_j jest tzw. standardowym czasem wykonywania zadania Z_j na tzw. maszynie standardowej; przyjmiemy, że jest to maszyna najwolniejsza, wówczas:

$$b_i \geqslant 1, i = 1, 2, \ldots, m$$

Powiemy natomiast o zbiorze maszyn *dowolnych* (*unrelated*) w pozostałych wypadkach.

Problem polega na znalezieniu takiego przydziału maszyn ze zbioru $\mathcal{M}$ do zadań ze zbioru $\mathcal{Z}$, że każde zadanie ze zbioru $\mathcal{Z}$ zostanie wykonane przy spełnieniu *wszystkich* przyjętych założeń (organiczeń). Taki przydział nazywać będziemy *uszeregowaniem dopuszczalnym* (*feasible schedule*), a procedurę jego wyznaczania *algorytmem szeregowania* (*scheduling algorithm*).

Uszeregowania dopuszczalne będziemy przedstawiali w postaci tzw. *wykresów Gantta* (*Gantt chart*) – rysunek 15.



Rysunek 15: Wykres Gantta

W danym uszeregowaniu dopuszczalnym dla każdego zadania Z_j możemy określić m.in.:

- moment zakończenia (*completion time, finishing time*) — C_j
- czas przepływu = czas przebywania w systemie maszyn (*flow time*) — $F_j = C_j - r_j$

- opóźnienie = nieterminowość (*lateness*) — $L_j = C_j - d_j$

Do oceny uszeregowań dopuszczalnych służą *kryteria szeregowania* (*scheduling criteria, schedule performance measures*). Główne z nich są następujące:

- długość uszeregowania = czas wykonywania zbioru zadań (*schedule length, makespan*)

$$C_{max} = \max_j\{C_j\}$$

- średni (średni ważony) czas przepływu (*mean (mean weighted) flow time*)

$$\overline{F} = \frac{1}{n}\sum_{j=1}^{n} F_j$$

$$\overline{F_W} = \frac{\sum_{j=1}^{n} w_j F_j}{\sum_{j=1}^{n} w_j}$$

- maksymalne spóźnienie (*maximum lateness*)

$$L_{max} = \max_j\{L_j\}$$

Uszeregowanie dopuszczalne minimalizujące przyjęte kryterium szeregowania K nazywać będziemy *uszeregowaniem optymalnym* (*optimal schedule*) i wyróżniać nadindeksem * (np. K^*).

Dokonajmy teraz interpretacji przyjętych dotychczas założeń (wyłączając założenia podstawowe). Wyróżnimy przy tym dwa środowiska (rzeczywistości):

- systemów operacyjnych komputerów
- systemów produkcyjnych

Zacznijmy od założenia podzielności zadań. W systemach produkcyjnych nie jest ona na ogół dopuszczalna, a jeśli jest, to przerwanie wiąże się z tzw. *czasem przezbrojenia* (*set-up time*). Natomiast w systemach operacyjnych założenie podzielności zadań jest naturalne. Ponadto, jeśli z przerwaniem zadania nie wiąże się jego przepisanie pomiędzy pamięcią operacyjną, a zewnętrzną, to przerwanie to nie wiąże się ze stratą czasu. Jeśli natomiast przepisanie między pamięciami musi nastąpić, to jego czas w danym systemie można wyliczyć i rozwiązując dany problem szeregowania dla zadań podzielnych oraz dla zadań niepodzielnych ocenić zysk płynący z dopuszczenia podzielności.

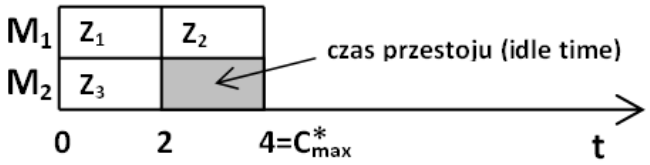
Rozpatrzymy prosty przykład:
Załóżmy, że:

$$\mathcal{Z} = \{Z_1, Z_2, Z_3\}$$
$$\prec = \varnothing$$
$$\mathcal{M} = \{M_1, M_2\}$$
$$\tau_1 = \tau_2 = \tau_3 = 2, K = C_{max}$$

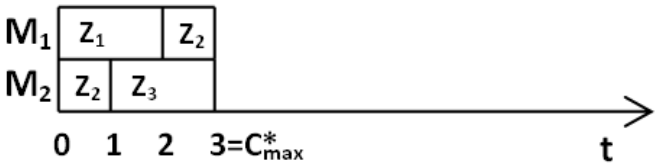
Znajdźmy uszeregowanie optymalne tego problemu przy założeniu, że zbiór $\mathcal{Z}$ jest

a. zbiorem zadań niepodzielnych (rys. 16)

b. zbiorem zadań podzielnych (rys. 17)



Rysunek 16: Wykres Gantta dla zadań niepodzielnych



Rysunek 17: Wykres Gantta dla zadań podzielnych

Jeśli chodzi o ograniczenia kolejnościowe w zbiorze $\mathcal{Z}$, to w systemach produkcyjnych wynikają one z ograniczeń technologicznych, natomiast w systemach operacyjnych mogą one wynikać z:

- dekompozycji
- zapewnienia określoności wyników
- efektywności obliczeń poprzez paralelizację

Przejdźmy obecnie do interpretacji założeń, które fundamentalnie różnią deterministyczną i kolejkową teorię szeregowania, a mianowicie do założenia znajomości momentu przybycia zadań i czasów ich wykonywania. W systemach produkcyjnych znajomość tych parametrów wynika z wysokiej powtarzalności procesów produkcyjnych (w stanie ustalonym).

Natomiast jeśli chodzi o systemy operacyjne i znajomość momentów gotowości zadań, to oczywiście jest ona naturalna w systemach obliczeniowych pracujących w trybie wsadowym (*off-line*). Warto jednak zauważyć, że momenty te są również znane w systemach komputerowych zastosowanych do sterowania, gdy okresowo (z okresami zależnymi od zmienności sterowanych parametrów) pobierane są dane dla algorytmów sterowania.

Wykład 10 — 11.05.2009

Jeśli natomiast chodzi o czasy wykonywania zadań, to w systemach operacyjnych praktycznie nigdy nie są one znane. Mimo to rozwiązanie deterministycznego problemu szeregowania może mieć ważne znaczenie praktyczne. Wymieńmy trzy główne sytuacje tego typu:

1. W charakterze czasu wykonywania zadań występują ich wartości średnie lub górne granice. W pierwszym wypadku rozwiązując deterministyczny problem szeregowania (*DPS*) możemy skonstruować estymator czasu wykonania zbioru zadań i zbadać jego własności. Drugi wypadek ma szczególne znaczenie w systemach sztywnego czasu rzeczywistego, w których musimy zapewnić, że:
$$\forall_j C_j \leqslant d_j$$
Wówczas jedynym podejściem jest przyjęcie w charakterze czasu wykonywania zadań ich górnych granic. Wynikających z funkcji złożoności obliczeniowych odpowiednich algorytmów.

2. W celu zidetyfikowania mechanizmu szeregowania dla algorytmu przybliżonego często rozwiązujemy *DPS* przy pewnych mocnych założeniach dotyczących parametrów problemu, na przykład przy założeniu, że czasy wykonywania wszystkich zadań są jednakowe (jednostkowe).

3. Wyznaczamy czasy wykonywania zadań *po* wykonaniu specjalnie skonstruowanego zbioru zadań testowych i rozwiązując odpowiedni(-e)*DPS* dla różnych kryteriów oceniamy pracującą procedurę szeregującą i podejmujemy ewentualnie decyzję o jej zmianie.

Na zakończenie zinterpretujmy krótko rozpatrywane przez nas kryteria szeregowania. Zauważmy, że minimalizacja C_{max} jest równoważna maksymalizacji współczynnika wykorzystania zbioru maszyn. Kryterium to reprezentuje zatem punkt widzenia *właściciela* zbioru maszyn. Natomiast minimalizacja $\overline{F}$ (lub $\overline{F}_W$) prowadzi do minimalizacji średniego czasu odpowiedzi, a zatem reprezentuje punkt widzenia *użytkownika* zbioru maszyn. Widzimy zatem, że kryteria te są w ogólności przeciwstawne.

Z kolei minimalizacja L_{max}, oprócz oczywistej interpretacji jest szczególnie istotna w systemach sztywnego czasu rzeczywistego, gdyż $(L^*_{max} \leqslant 0) \Rightarrow \forall_j C_j \leqslant d_j$

Dla dalszych rozważań niezbędne jest ściślejsze zdefiniowanie problemów szeregowania.

Definicja 3.1. *Przez problem szeregowania* Π *będziemy rozumieli skończony, uporządkowany ciąg parametrów, zawierający informacje odpowiednio o:*

- *zbiorze maszyn* $\mathcal{M}$
- *zbiorze zadań* $\mathcal{Z}$
- *kryterium szeregowania* $\mathcal{K}$

Przy czym parametry są tu rozumiane ogólnie: (parametry liczbowe, zbiory, relacje, wektory, ...) i nie wszystkie muszą mieć nadane wartości.

Przykładowo zapis: *P2 | podz., las |* C_{max} oznacza szeregowanie zadań podzielnych w ograniczeniach kolejnościowych w postaci lasu na dwóch identycznych maszynach równoległych w celu minimalizacji C_{max}.

Jeśli dla problemu szeregowania Π określimy wartości *wszystkich* jego parametrów, to otrzymamy tzw. *instancję* (ang. *instance*) I tego problemu. Zbiór wszystkich instancji problemu szeregowania *Pi* nazywać będziemy dziedziną tego problemu i oznaczać przez D_Π. Możemy teraz definiować algorytmy szeregowania.

Definicja 3.2. Algorytmem szeregowania *dla problemu* Π *nazywamy skończony ciąg dobrze zdefiniowanych operacji, który* $\forall_{I \in D_\Pi}$ *znajduje uszeregowanie dopuszczalne jeśli ono istnieje.*

Definicja 3.3. Algorytmem dokładnym, optymalizacyjnym *(ang. exact, optimization) dla problemu* Π *nazywamy algorytm szeregowania dla tego problemu, który* $\forall_{I \in D_\Pi}$ *znajduje uszeregowanie optymalne jeśli ono istnieje.*

Definicja 3.4. *Algorytm szeregowania dla problemu* Π*, który nie jest algorytmem dokładnym nazywać będziemy* **algorytmem przybliżonym** *(ang. approximation),* heurystycznym *(ang. heuristic) lub* suboptymalnym *(ang. suboptimal).*

Uwaga: W dokładniejszych rozważaniach typy algorytmów z ostatniej definicji są precyzyjnie zdefiniowane i rozróżniane.

Możemy teraz powiedzieć, że dla danego problemu szeregowania Π dążymy do konstrukcji algorytmu dokładnego, który byłby obliczeniowo jak najprostszy. Wymagania te jednak są w ogólności sprzeczne. Dla dokładniejszego naświetlenia tej sprzeczności musimy sobie przypomnieć podstawowe pojęcia z zakresu złożoności obliczeniowej algorytmów i problemów kombinatorycznych. Jak pamiętamy, klasy złożonościowe problemów kombinatorycznych są zdefiniowane dla *problemów decyzyjnych*, czyli sformułowanych w postaci pytania, na które odpowiedź brzmi „tak" lub „nie". Natomiast problemu szeregowania to zasadniczo problemy optymalizacyjne, w których chodzi o optymalizację kryterium K. Żeby zatem móc korzystać do analizy tych problemów z teorii złożoności obliczeniowej, musimy powiązać między sobą obie te klasy problemów. W tym celu zauważmy najpierw, że z każdym problemem optymalizacyjnym możemy związać jego wersję decyzyjną pytając o istnienie rozwiązania (tu: uszeregowania dopuszczalnego) o wartości $K \leqslant k$ (minimalizacja) lub $K \geqslant k$ (maksymalizacja).

Zauważmy dalej, że powstała w ten sposób wersja optymalizacyjna jest obliczeniowo nietrudniejsza od problemu wyjściowego.

Poniżej podsumujemy tylko krótko sposób postępowania przy badaniu złożoności obliczeniowej problemy szeregowania Π. Otóż mamy tu dwie możliwości:

1. Podanie algorytmu dokładnego wielomianowego dla problemu Π (optymalizacyjnego) – jeśli to się uda, to znaczy że wersja decyzyjna problemu Π jest obliczeniowo łatwa ($\in \mathbb{P}$).

2. Wykazanie $\mathbb{NP}$-trudności problemu Π – żeby tego dokonać musimy przetransformować wielomianowo dowolny znany problem $\mathbb{NP}$-trudny ($\mathbb{NP}$-zupełny) do wersji decyzyjnej badanego problemu Π.

Uwaga: Kolejność wyboru powyższych dróg, jak również wybór problemu $\mathbb{NP}$-trudnego ($\mathbb{NP}$-zupełnego) do transformacji opisanej w punkcie 2. zależy od intuicji badającego.

Jeśli obie powyższe drogi zawiodą to mówimy, że badany problem Π jest otwarty (ang. *open*) z punktu widzenia złożoności obliczeniowej.

3.2 Przykładowe problemy, metody i algorytmy

W tym punkcie zilustrujemy główne aspekty metodologiczne dotyczące kombinatorycznych problemów optymalizacyjnych. Uczynimy to na prostym przykładzie dotyczącym problemów szeregowania w celu minimalizacji C_{max} przy założeniu, że

$$\bigwedge_j (r_j = 0) \wedge (w_j = 1) \wedge (d_j = \infty)$$

Uczynimy to dla niezależnych i zależnych zadań oraz maszyn identycznych i dowolnych.

3.2.1 Niezależne zadania

Identyczne maszyny Jak pamiętamy, podzielność zadań dla kryterium C_{max} ma wpływ na minimalną wartość tego kryterium, zatem rozpatrzymy oddzielny przypadek zadań podzielnych oraz niepodzielnych.

Podzielne zadania Przy przyjętych założeniach łatwo zauważyć, że możemy z góry wyliczyć minimalną wartość naszego kryterium, która dana jest wzorem:

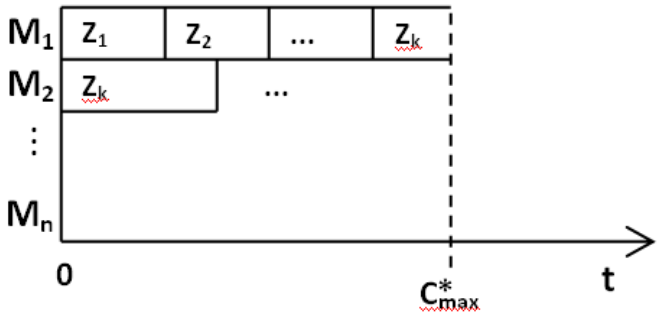
$$C^*_{max} = \max\left\{\max_j \tau_j, \frac{1}{m}\sum_{j=1}^{n}\tau_j\right\}$$

Możliwe są tu bowiem dwie sytuacje, mianowicie:

1. C^*_{max} jest wyznaczone przez czas wykonywania najdłuższego zadania
2. C^*_{max} jest wyznaczone przez w pełni równoległe wykonanie zbioru zadań

Oczywiście w pierwszej sytacji wystąpią przestoje w pracy co najmniej 1 maszyny.

W tej sytuacji pozostaje nam znalezienie uszeregowania dopuszczalnego o długości C^*_{max}, czyli rozwiązania optymalnego. Łatwo wykazać, że uszeregowanie takie dla każdej instancji rozpatrywanego problemu znajduje *algorytm McNaughtona*.



Rysunek 18: Wykres Gantta dla algorytmu McNaughtona

1. Wybierz dowolne zadanie i rozpocznij jego wykonywanie na dowolnej maszynie w chwili $t = 0$
2. Wybietz dowolne, nieuszeregowane jeszcze zadanie i rozpocznij jego wykonywanie na tej samej maszynie w chwili zakończenia poprzedniego zadania. Powtarzaj, aż wszystkie zadania zostaną wykonane, lub osiągniemy z czasem bieżącym $t = C^*_{max}$
3. Część zadania pozostałą po osiągnięciu $t = C^*_{max}$ przydziel do dowolnej wolnej maszyny, rozpoczynając jej wykonywanie od $t = 0$

4. Wracaj do punktu 2, aż wszystkie zadania zostaną wykonane

Uwaga: wszystkie stwierdzenia występujące w opisie algorytmu szeregowania służą wyłącznie konstrukcji urzeregowania, które później jest realizowane zgodnie z uzyskanym wykresem *Gantta*. Łatwo zauważyć, że złożoność obliczeniowa algorytmu *McNaughtona* jest rzędu n ($O(n)$). Jest to minimalna złożoność obliczeniowa algorytmu dokładnego dla naszego problemu szregowania.

Problem nasz jest zatem w ten sposób do końca rozwiązany.

Nieodzielne zadania W tym wypadku już dla $m = 2$ (zauważmy, że dla $m = 1$ rozpatrywany problem szeregowania jest trywialny) i dla całkowitych czasów wykonywania zadań, rozpatrywany problem szeregowania jest $\mathbb{NP}$-trudny (prosta transformacja wielomianowa od problemów PODZIAŁU ZBIORU podana jest w książce [1]). W tej sytacji mamy dwie możliwości: zastosowania algorytmu dokładnego o złożżoności wykładniczej, albo konstrukcja algorytmu wielomianowego, ale przybliżonego. Poniżej podamy przykładowe realizacje obu tych możliwości.

Zaczynając od pierwszej z nich, sprowadzimy nasz problem szeregowania do zerojedynkowego problemu programowania liniowego (w skrócie: 0-1PL). Wprowadźmy zmienne decyzyjne $x_{ij}, i = 1, 2, \ldots, m; j = 1, 2, \ldots, n$ wzorem:

$$x_{ij} \begin{cases} 1 & Z_j \textit{ wykonywane na } M_i \\ 0 & w \textit{ p.p.} \end{cases}$$

Za pomocą tych zmiennych, łatwo otrzymujemy następujący problem 0-1PL:
Zminimalizować C_{max} *przy ograniczeniach*

$$C_{max} - \sum_{j=1}^{n} \tau_j x_{ij} \geqslant 0, i = 1, 2, \ldots, m$$

$$\sum_{i=1}^{m} x_{ij} = 1, j = 1, 2, \ldots, n$$

Oczywiście problem 0-1PL jest $\mathbb{NP}$-zupełny, zatem dla jego dokładnego rozwiązania niezbędne są algorytmy wykładnicze. Podkreślmy, że sprowadzenie naszego problemu szeregowania do problemu 0-1PL nie jest niezbędne dla jego (dokładnego) rozwiązania; możemy bowiem podejść do tego problemu w sposób bezpośredni.

Przykładową realizacją drugiej możliwości jest algorytm LPT (ang. *longest processing time*), szeregujący zadania według najdłuższych czasów wykonywania. Jego złożoność obliczeniowa jest oczywiście $O(n \log n)$.

Graham wykazał, że zachodzi następująca nierówność:

$$\frac{C^*_{max(LPT)}}{C^*_{max}} \leqslant \frac{4}{3} - \frac{1}{3m}$$

3.2.2 Dowolne maszyny

Podzielne zadania Pokażemy obecnie dwa podejścia do dokładnego rozwiązania naszego problemu szeregowania, wykorzystujące istotnie różne transformacje tego problemu do problemu programowania liniowego (PL). Pierwsze z podejść do tzw. *podejście dwuetapowe*, w którym transformacja do problemu PL stanowi treść pierwszego etapu.

Wprowadźmy zmienne decyzyjne $x_{ij}, i = 1, 2, \ldots, m; j = 1, 2, \ldots, n$, przy czym $0 \leqslant x_{ij} \leqslant 1$ oznacza sumaryczną część zadania Z_j wykonywaną na maszynie M_i (część czasu wykonywania

τ_j). Za pomocą tych zmiennych dokonamy transformacji problemu znalezienia C^*_{max} i odpowiadających jej $\forall_{i,j} x^*_{ij}$ do problemu PL.

Zauważmy, że dwa zbiory ograniczeń w problemie PL są prawie identyczne jak w rozpatrywanym poprzednio problemie O-1PL (v_j zastępujemy przez v_{ij}). Jednak ze względu na podzielność zadań, musimy dodać do nich jeszcze jeden zbiór ograniczeń, który gwarantuje, że czas wykonywania każdego z zadań nie przekroczy C_{max} (gdyż wówczas części tego zadania musiały by być wykonywane równocześnie).

Otrzymujemy zatem następujący problem PL:
Zminimalizować C_{max} przy ograniczeniach:

$$C_{max} - \sum_{i=1}^{m} \tau_{ij} x_{ij} \geqslant 0, j = 1, 2, \ldots, n$$

$$\sum_{i=1}^{m} x_{ij} = 1, j = 1, 2, \ldots, n, \forall_{i,j} 0 \leqslant x_{ij} \leqslant 1$$

Rozwiązując powyższy problem PL, np. jednym z algorytmów simpleksów otrzymamy C^*_{max} oraz $x^*_{ij}, \forall_{i,j}$ na tej podstawie, dzięki zapisanym ograniczeniom, będziemy w stanie skonstruować uszeregowanie optymalne o tych parametrach.

Wykład 12 25.05.2009

Problem ten jest częścią drugiego etapu naszego podejścia. Można wykazać, że sprowadza się do problemu przepływu w odpowiedniej sieci i może być rozwiązany w czasie $O(n^3)$. Zauważmy zatem, że wyjściowy problem szeregowania jest problemem obliczeniowo łatwym. Wynika to z faktu wielomianowości przedstawionej transformacji do problemu PL (1 etap) i z faktu, że problem PL jest również obliczeniowo łatwy (algorytmy niesimpleksowe). Wspomnieliśmy również o wielomianowości drugiego etapu.

Pokażemy teraz, że nasz problem szeregowania można sprowadzić do problemu PL także w inny, istotnie różny sposób, który jest treścią tzw. *podejścia jednoetapowego*. Utwórzmy najpierw wszytskie *możliwe warianty* wykonywania zadań ze zbioru $\mathcal{Z}$ na maszynach ze zbioru $\mathcal{M}$. Oznaczmy liczbę tych wariantów przez N i przypiszmy k-temu wariantowi czas realizacji $x_k \geq 0, k = 1, 2, \ldots, N$, który pootraktujemy jako zmienną decyzyjną.
Otrzymujemy następujący problem PL: *Zminimalizować* $\sum_{k=1}^{N} x_k$ *przy ograniczeniach* $\sum_{i=1}^{m} \sum_{k \in K_{ij}} \frac{x_k}{\tau_{ij}}$, *gdzie* K_{ij} *jest zbiorem numerów wariantów, w których zadanie* Z_j *jest wykonywane na maszynie* M_i.

Zauważmy jednak, że przedstawiona wyżej transformacja jest wykładnicza, gdyż liczba możliwych wariantów N jest rzędu $O(n^m)$. Podejście to jednak ma tę przewagę nad podejściem dwuetapowym, że może być w naturalny sposób uogólnione na sytuację, gdy oprócz maszyn mamy również inne zasoby żądane w sposób dyskretny.

Niepodzielne zadania. Rozpatrywany problem szeregowania można sprowadzić do problemu 0-1PL zastępując w ograniczeniach definiujących $C_{\max}$ τ_j przez τ_{ij}.

3.2.3 Zależne zadania

Rozpatrzymy tylko przypadek *identycznych* maszyn.

Podzielne zadania *Muntz* i *Coffman* wykazali, że stosunek minimalnej długości uszeregowania zbioru zadań przy przyjętych założeniach do długości uszeregowania gdzie nie ma założenia o niepodzielności wyraża się wzrorem:

$$\frac{C^*_{\max(np)}}{C^*_{\max(p)}} \leqslant 2 - \frac{1}{m}$$

Podali oni również algorytm dokładny dla tego problemu o złożoności $O(n^2)$ i przy założeniu, że digraf ograniczeń kolejnościowych jest *typu lasu* (*las* lub *antylas*), lub $m = 2$.
Algorytm ten wykorzystuje pojęcie poziomu (ang. *level*) zadania, który definiuje się jako długość najdłuższej ścieżki łączącej dwa zadania z jednym z zadań końcowych.

Algorytm *Muntza-Coffmana*

1. Wyznacz poziomy wszystkich zadań (lub ich części).
2. Przydziel zadanie (bez poprzedników) o najwyższym poziomie (np. n_1 zadań) do wolnych maszyn (np. m_1 maszyn) w następujący sposób.
   - Jeśli $n_1 > m_1$, to przydziel każdemu z zadań $\alpha = \frac{m_1}{n_1}$ mocy wykonawczej maszyn.
   - W przeciwnym razie przydziel każdemu zadaniu maszynę. Jeśli są jeszcze wolne maszyny, to rozpatrz w analogiczny sposób zadania o poziomie bezpośrednio mniejszym itd. Przydział ten kontynuuj, aż jakieś z zadań się zakończy, lub do momentu począwszy od którego zadanie o wyższym poziomie byłoby wykonywane wolniej (z mniejszym α) niż zadanie o niższym poziomie. Wracaj do punktu 1. aż wszystkie zadania zostaną wykonane.
3. W celu skonstruowania uszeregowania optymalnego zastosuj algorytm *McNaughtona* do odcinków uszeregowania pomiędzy chwilami określonymi w punkcie 2.

Uwaga: Algorytm *Muntza-Coffmana* wymaga również założenia, że czasy wykonywania zadań są wielokrotnościami przyjętej jednostki. Rozpatrzmy prosty przykład:
Wyznaczyć uszeregowanie minimalizujące $C_{\max}$ podanego zbioru zadań podzielnych na dwóch identycznych maszynach
Ponieważ $m = 2$ i czasy wykonywania zadań są wielokrotnościami przyjętej jednostki, możemy zastosować algorytm *Muntza-Coffmana*.

Wykład 13 — 28.05.2009

Dla dowolnych organiczeń kolejnościowych i dowolnego n rozpatrywany problem szeregowania jest $\mathbb{NP}$-trudny, natomiast dla ustalonego n (już dla $n = 3$) otwarty[6].

Niepodzielne zadania W tym wypadku znane są dwa algorytmy dokładne wielomianowe, skonstruowane przy założeniu, że czasy wykonywania zadań są jednostkowe (jednakowe). Jest to algorytm *Hu*, który dodatkowo zakłada, że digraf ograniczeń kolejnościowych jest *antylasem*, i algorytm *Coffmana-Grahama*, który dodatkowo zakłada, że $m = 2$. Ten drugi algorytm ma złożoność $O(n^2)$ i wykorzystuje pojęcie etykiety (cechy) zadania jako poziomu. Poniżej sformułujemy algorytm *Hu*.

[6] Jego status złożonościowy nie jest znany

Algorytm *Hu*

1. Wyznacz poziomy wszystkich zadań.
2. Jeśli liczba zadań bez poprzedników $\leqslant m$ to przydziel im maszyny i przejdź do punktu 3. W przeciwnym razie wybierz spośród nich m zadań o najwyższych poziomach (spośród zadań o jednakowym poziomie wybór jest dowolny) i przydziel im maszyny.
3. Usuń przydzielone zadania z grafu. Wracaj do punktu 2, aż graf będzie pusty.

Znaleźć uszeregowanie minimalizujące $C_{\max}$ podanego zbioru zadań niepodzielnych na 3 identycznych maszynach, przy założeniu, że czasy wykonywania zadań są jednostkowe.

Uwagi: Złożoność algorytmu *Hu* jest $O(n)$. Jeżli digraf ograniczeń kolejnościowych jest *lasem*, to odwracamy zwroty wszystkich łuków, stosujemy algorytm *Hu*, a następnie odracamy kolejność zadań na każdej z maszyn.

Rozpatrywany problem szeregowania zadań niepodzielnych jest w ogólności silnie $\mathbb{NP}$-trudny, jest on też $\mathbb{NP}$-trudny dla zadań jednostkowych i dowolnych ograniczeń kolejnościowych i dowolnego m. Natomiast dla ustalonego m już dla $m = 3$ jest otwarty. Pozostają nam zatem algorytmy dokładne wykładnicze, lub algorytmy wielomianowe, ale przybliżone. Spośród tych ostatnich szczególnie często stosowane są algorytmy *szeregowania listowego* (ang. *list scheduling*), w których zadania umieszczone są na liście według malejących priorytetów. Algorytm szeregowania przegląda tę listę i przydziela wolną maszynę do pierwszego zadania, którego wszystkie poprzedniki zostały wykonane. Oczywiście złożoność takiego algorytmu jest $O(n \log n)$. Obszerne eksperymenty obliczeniowe pozwoliły stwierdzić, że spośród wielu przebadanych reguł przydziału priorytetów najlepsze wyniki dały dwie:

- HLFET (ang. *highest level first with estimated times*) — jest to po prostu zastosowanie algorytmu *Hu* w sytuacji, gdy czasy wykonywania zadań są znane, ale nie są jednostkowe.
- HLFNET (ang. *highest level first with non-estimated times*) — jest to również zastosowanie algorytmu *Hu* w sytuacji, gdy czasy wykonywania zadań nie są znane, ale przyjmujemy, że są jednostkowe.

Pierwsza z powyższych reguł daje uszeregowania optymalne w ponad 90% przypadków, a średnia odległość od optimum jest $< 5\%$ (takie reguły nazywa się prawie optymalnymi).

4 Wybrane problemy analizy sieci zadań

4.1 Problem analizy czasowej, przypadek deterministyczny: sformułowanie, metoda CPM

Wchodzimy obecnie w *problematykę analizy sieci zadań* (programowania sieciowego, sterowania siecią (kompleksem) zadań (operacji)) zwaną w terminologii angielskiej *project scheduling*. W tych problemach mówi się raczej o czynnościach (ang. *activity*) niż zadaniach i raczej o czasach trwania czynności (ang. *activity duration*). W dalszym ciągu zachowamy jednak dotychczasową terminologię. W ogólności w problemach analizy sieci zadań, zadania mogą żądać do swego wykonania dowolnych zasobów. W analizie czasowej sieci zadań nie rozpatruje się *explicite* żadnych zasobów czasowych, co praktycznie oznacza, że są one nieograniczone. W takiej sytuacji każde zadanie jest scharakteryzowane (w przypadku deterministycznym) wyłącznie przez czas jego wykonywania (czyli praktycznie przy założeniu, że wszystkie zasoby, których żadało zostały mu

przydzielone). Łatwo zauważyć, że znalezienie minimalnego czasu wykonania zbioru zadań przy tym założeniu oznacza znalezienie minimalnych wartości tego czasu w sytuacji, gdy zasoby są ograniczone.

W problemach analizy czasowej sieci zadań zakłada się ponadto, że digraf ograniczeń kolejnościowych jest dany w przedstawieniu łukowym, czyli tworzy sieć zadań (ang. *activity network*).

Przedmiotem analizy czasowej sieci zadań są następujące parametry:

- Najwcześniejszy i najpóźniejszy termin zajścia wszystkich zadań w sieci.
- Najwcześniejszy i najpóźniejszy termin rozpoczęcia i zakończenia poszczególnych zadań w sieci.
- Luzy (zapasy) czasowe zdarzeń i zadań.
- Ścieżka(-i) krytyczna(-e)

Poniżej wyznaczymy wymienione parametry. Założymy w tym celu, że wierzchołki (zdarzenia) sieci są uporządkowane od $1 \dots s$ w ten sposób, że zajście wierzchołka i jest niepóźniejsze od zajścia wierzchołka j ($i < j$). Uporządkowanie takie jest zawsze możliwe i może być otrzymane np. metodą skreślania łuków (wierzchołek bez łuków wejściowych otrzymuje nr 1, skreślamy wszystkie łuki wychodzące z niego i znajdujemy wierzchołki bez łuków wejściowych, które otrzymują – w sposób dowolny – kolejne numery. Skreślamy łuki wychodzące z wszystkich tych wierzchołków, itd.).

Wykład 14 — 01.06.2009

Zadanie oznaczać będziemy jako uporządkowaną parę $(i, j), i < j$, gdzie i, j są wierzchołkami (zdarzeniami), między którymi jest ono wykonywane, natomiast czas wykonywania tego zadania oznaczymy przez $\tau_{i,j}$.

Stosując metodę CPM (ang. *Critical Path Method*) wyznaczymy obecnie wymienione na wstępie charakterystyki, zaczynając od najwcześniejszych terminów zajścia poszczególnych zdarzeń w sieci.

Oznaczając przez t_j^w najwcześniejszy termin zajścia zdarzenia j zauważamy, że:

$$t_j^w = 0$$
$$t_j^w = \max\left\{T(P_j^k)\right\}, j = 2, 3, \dots, 5$$

gdzie P_j^k jest k-tą ścieżką od zdarzenia j do zdarzenia k tej sieci, a $T(P_j^k) = \sum_{(i,j)\in P_j^k)} \tau_{ij}$.

Łatwo zauważyć, że dla wyznaczenia t_j^w możemy zastosować wzór rekurencyjny:

$$t_j^w = 0$$
$$t_j^w = \max_{i\in A_j}\left\{t_i^w + \tau_{ij}\right\}, j = 2, 3, \dots, 5$$

gdzie A_j jest zbiorem wszystkich wierzchołków, w którym rozpoczynają się zadania dochodzące do wierzchołka j.

Najpóźniejszy termin zajścia wierzchołka i, t_i^p, który nie powoduje opóźnienia zajścia zdarzenia końcowego s wyznaczymy analogicznie, ale poruszając się od końca sieci. Łatwo zauważyć, że:

$$t_s^p = t_s^w$$
$$t_i^p = \min_k\left\{t_s^p - T(\hat{P}_i^k)\right\}, i = s-1, s-2, \dots, 1$$

gdzie $\hat{P}$ jest k-tą ścieżką (w sieci z odwróconymi zwrotami łuków) od zdarzenia s do zdarzenia i, a T od tego jest długością tej ścieżki. Łatwo zauważyć, że dla wyznaczenia t_i^p można zastosować wzór rekurencyjny:

$$t_s^p = t_s^w$$
$$t_i^p = \min_{j \in B_i} \left\{ t_j^p - \tau_{ij} \right\}, i = s-1, s-2, \ldots, 1$$

gdzie B_i jest zbiorem wszystkich wierzchołków, w których kończą się zadania wychodzące z wierzchołka i.

Przejdźmy obecnie do wyznaczenia najwcześniejszych i najpóźniejszych terminów rozpoczęcia i zakończenia poszczególnych zadań w sieci.
Łatwo zauważyć, że:

- t_i^w = najwcześniejszy termin rozpoczęcia zadań wychodzących z wierzchołka i (EST = ang. *Early Start Time*).
- $t_j^w + \tau_{ij}$ = najwcześniejszy termin zakończenia zadania (i, j) (EFT = ang. *Early Finishing Time / Completion Time*).
- $t_j^p - \tau_{ij}$ = najpóźniejszy termin rozpoczęcia zadania (i, j) (LST = ang. *Late Start Time*).
- t_j^p = najpóźniejszy termin zakończenia zadań dochodzących do wierzchołka j (LFT = ang. *Late Finishing Time / Completion Time*).

Przejdźmy obecnie do wyznaczenia luzów (zapasów) czasowych zdarzeń i zadań w sieci.
Liczbę

$$S_i = (t_i^p - t_i^w) \geqslant 0, i = 1, 2, \ldots, s$$

nazywamy luzem (zapasem) czasowym zdarzenia i (ang. *Event Slack Time*). Łatwo zauważyć, że wyraża on maksymalne przesunięcie terminu zajścia tego zdarzenia, nie powodujące opóźnienia zajścia zdarzenia końcowego s.
Zdarzenia, dla których $s_i = 0$ nazywać będziemy *zdarzeniami krytycznymi.*

Ponieważ każde zadanie (i, j) leży poniędzy **dwoma** zdarzeniami, więc możemy do niego określić **cztery** luzy (zapasy czasowe).

- Luz całkowity (ang. *Total Slack Time*)

$$S_{ij}^c = t_j^p - t_i^w - \tau_{ij}$$

- Luz bezpieczny (ang. *Safety Slack Time*)

$$S_{ij}^b = t_j^p - t_i^p - \tau_{ij}$$

- Luz swobodny (ang. *Free Slack Time*)

$$S_{ij}^s = t_j^w - t_i^w - \tau_{ij}$$

- Luz niezależny (ang. *Independent Slack Time*)

$$S_{ij}^n = \max \left\{ 0, t_j^w - t_i^p - \tau_{ij} \right\}$$

Na zakończenie przejdziemy do określenia i wyznaczenia ścieżki krytycznej.

Definicja 4.1. Ścieżką (drogą) krytyczną *nazywamy ścieżkę od zdarzenia 1 do zdarzenia s o maksymalnej długości w sieci. Łatwo zauważyć, że w każdej sieci istnieje co najmniej jedna taka ścieżka.*

Zadania leżace na ścieżce krytycznej nazywają się *zadaniami krytycznymi.* Łatwo zauważyć, że zdarzenia leżące na ścieżce krytycznej są zdarzeniami krytycznymi, ale ciąg zdarzeń krytycznych nie zawsze wyznacza ścieżkę krytyczną.
Można natomiast wykazać następujące twierdzenie:

Twierdzenie 4.1. *Na to, by zadanie (i, j) było zadaniem krytycznym potrzeba i wystarcza, by $S_{ij}^c = 0$. Wynika stąd, że ścieżka krytyczna jest jednoznacznie wyznaczona przez zadania, w których luz całkowity jest zerowy.*

Rozpatrzmy następujący przykład:

Dla podanej sieci wyznaczyć $t_i^w, t_i^p, S_i, S_{ij}^c$ wszystkich zadań i wierzchołków.

4.2 Problem analizy czasowej, przypadek probabilistyczny: sformułowanie, metoda PERT

Rozważymy teraz sytuację, gdy czasy wykonywania zadań w sieci zadań są zmiennymi losowymi o nieznanym rozkładzie prawdopodobieństwa. Z praktycznego punktu widzenia uzasadnione jest przyjęcie o tym rozkładzie następujących założeń:

- ciągłość
- jedno ekstremum
- styk z osią odciętych w dwóch punktach

Przykładowe wykresy funkcji gęstości prawdopodobieństwa spełniających powyższe założenia są następujące:

Jednym z rozkładów spełniających powyższe założenia jest tzw. *rozkład beta* o funkcji gęstości prawdopodobieństwa danej wzorem:

$$f(\tau) = \begin{cases} H(a-\tau)^{p-1}(\tau-b)^{q-1} & dla\ \tau \in (a,b) \\ 0 & w.p.p. \end{cases}$$

gdzie a, b, p, q są stałymi większymi od 0, a H jest pewną funkcją zależną od a, b, p, q. W metodzie PERT (ang. *Program Evaluational Review Technique*) zakłada się właśnie, że czasy wykonywania zadań w rozpatrywanej sieci zadań mają rozkład *beta*. Następnie wyznacza się w przybliżeniu wartość średnią i wariancję każdego z tych czasów korzystając z trzech wartości (dla każdego zadania), które łatwo określić ekspertom:

- optymistycznej a
- pesymistycznej b
- najbardziej prawdopodobnej τ_0 (wartość modalna)

Na tej podstawie, przy założeniu rozkładu beta, mamy:

$$m \approx \frac{a + 4\tau_0 + b}{6}$$
$$\sigma^2 \approx \frac{(b-a)^2}{36}$$

Nie znając czasu wykonywania zadań nie możemy korzystać z pojęcia ścieżki krytycznej i zadań krytycznych, dlatego *metoda PERT jest ukierunkowana na zdarzenia*, a ściśle chodzi w niej o znalezienie prawdopodobieństw ujemnych luzów czasowych poszczególnych zdarzeń, czyli o wyznaczenie prawdopodobieństw $P(S_i < 0), i = 1, 2, \ldots, s$. Jak pamiętamy $S_i = t_i^p - t_i^w$, przy czym t_i^p i t_i^w są ostatecznie sumami pewnej liczby czasów wykonywania zadań, czyli w naszym przypadku pewnej liczby niezależnych zmiennych losowych. Na podstawie centralnego twierdzenia granicznego można w przybliżeniu przyjąć, że mają one rozkład asymptotycznie normalny. Zatem również $S_i = t_i^p - t_i^w$ ma również rozkład asymptotycznie normalny, a ściślej rozkład $N\left(m_i^p - m_i^w, \sqrt{(\sigma_i^w)^2 + (\sigma_i^p)^2}\right)$.

Po unormowaniu tego rozkładu możemy skorzystać z tablic rozkładu $N(0,1)$ i znaleźć interesujące nas prawdopodobieństwo. Przyjmuje się, że jeśli $P(S_i < 0) > 0,75$ to ryzyko związane z dotrzymaniem terminu zajścia tego zdarzenia jest zbyt wysokie.

Wykład 15 — 15.06.2009

4.3 Problem analizy czasowo-kosztowej: sformułowanie, metoda CPM/MCX, metoda PL

W tym problemie występuje jeden zasób, którym jest koszt. Koszt K wykonania danego zbioru zadań składa się z:

- kosztu bezpośredniego K_b, będącego sumą kosztów bezpośrednich K_ij wykonania *poszczególnych* zadań (i, j) (robocizna, energia, materiały itd.)
- kosztu pośredniego K_p odniesionego do *całego zbioru* zadań (administracja, zamrożenie kapitału)

Załóżmy, że:

- koszt bezspośredni K_{ij} zadania (i, j) można zmieniać w określonych granicach, zmieniając czas jego wykonywania τ_{ij} zgodnie z daną krzywą czas-koszt tego zadania
- koszt pośredni K_p *rośnie liniowo* ze wzrostem czasu wykonania zbioru zadań C_{max}

Typowe wykresy pokazano na rysunkach poniżej.

Problem polega na wyznaczeniu takich czasów wykonywania zadań τ_{ij}, dla których łączny koszt K wykonania zbioru zadań osiąga minimum (time-cost trade-off - najlepszy kompromis pomiędzy czasem i kosztem). Istnieje wiele metod, dokładnych i przybliżonych, do znalezienia rozwiązania tego problemu. Poniżej podamy dwie metody przybliżone. Zakłada się w nich znajomość dwóch punktów na każdej krzywej czas-koszt:

- punktu normalnego o współrzędnych (K_{ij}^n,τ_{ij}^n)
- punktu granicznego o współrzędnych (K_{ij}^g,τ_{ij}^g)

Metoda CPM/MCX (CMP/Minimum Cost eXpenditing) jest uogólnieniem metody CPM z uwzględnieniem zasobu, którym jest koszt. Ogólna idea tej metody jest następująca:

1. Dla czasów τ_{ij}^n zastosuj metodę CPM do wyznaczenia luzów czasowych całkowitych zadań i ścieżki krytycznej
2. Skracaj czasy wykonywania zadań krytycznych, nie przekraczając punktu (g) dla którego prowadzi to do zmniejszenia K (porównaj Siudak)

Podamy teraz drugą metodę przybliżoną rozwiązania naszego problemu, polegającą na sprowadzeniu do problemu PL po uprzednim dokonaniu lineryzacji krzywych czas-koszt poszczególnych zadań. Oznacza to, że pomiędzy punktami normalnym i granicznych prowadzimy prostą o współczynniku $a_{ij} = \frac{K_{ij}^g - K_{ij}^n}{\tau_{ij}^n - \tau_{ij}^g}$, który oznacza przyrost kosztu bezpośredniego τ_{ij} związany ze zmniejszeniem czasu jego wykonywania o jednostkę. Przy tym założeniu koszt bezpośredni K_{ij} wyraża się wzorem:

$$K_{ij} = K_{ij}^n + a_{ij}(\tau_{ij}^n - \tau_{ij}) = -a_{ij}\tau_{ij} + c$$

W konsekwencji koszt bezpośredni wykonania całego zbioru zadań wynosi:

$$K_b = \sum_{(i,j)\in G} K_{ij} = -\sum_{(i,j)\in G} a_{ij}\tau_{ij} + c_1$$

Oznaczając koszt pośredni wykonania zbioru zadań przez $K_p = \alpha C_{max}$ otrzymujemy zatem następujący problem PL:

zminimalizować $K = K_b + K_p = \alpha C_{max} - \sum_{(i,j)\in G} a_{ij}\tau_{ij}$
przy ograniczeniach:

1. $\tau_{ij}^g \leq \tau_{ij} \leq \tau_{ij}^n, \quad \forall_{(i,j)\in G}$
2. S_{ij}^c (luz całkowity) $\leq 0 \ \rightarrow \ \tau_{ij} \leq t_j^p - t_i^w, \quad i = 1..s, \quad \forall_{(i,j)\in G}$
3. $t_i^w \leq t_i^p \quad i = 1..s \quad \forall_{(i,j)\in G}$
4. $t_s^p = C_{max}$
5. $t_1^w = 0$

Otrzymaliśmy problem PL, w którym zmienne decyzyjne to C_{max}, τ_{ij}, t_i^p, t_i^w.

4.4 Problemy analizy czasowo-zasobowej

Ten punkt dotyczy najogólniejszego problemu analizy sieci zadań, z uwzględnieniem dowolnych zasobów.

Rozważmy ogólny system typu *sieć (kompleks) zadań (operacji)* podstaci:

$$\mathcal{S} = \{\mathcal{R}, \mathcal{Z}, \mathcal{K}\}$$

$\mathcal{R}$ - zbiór zasobów zadań
$\mathcal{Z}$ - zbiór zadań
$\mathcal{K}$ - zbiór kryteriów optymalności

W zbiorze *mathcalR* wyróżnia się k *typów* (rodzajów) zasobów: $R_1..R_k$, przy czym do tego samego typu zaliczamy wszystkie jednostki zasobu, niekoniecznie identyczne, które spełniają te same funkcje (np. maszyny o wykonujące to samo, pomimo, że z różnymi prędkościami).

Niezależnie od podziały na typy, zasoby dzieli się na *kategorie* z trzech punktów widzenia:

- przywłaszczalność
  - przywłaszczalne (preemptible)
  - nieprzywłaszczalne (non-preemptible)

- podzielność
  - dyskretne (discrete) - podzielne w sposób dyskretny
  - ciągłe (continuous) - podzielne w sposób ciągły
- ograniczenia zasobowe (główne kategorie)
  - odnawialne (renewable)
  - nieodnawialne (non-renewable)
  - podwójnie ograniczone (doubly constrained)

Zasób nazywamy przywłaszczalnym, jeśli każda jego jednostka może zostać odebrana zadaniu, przydzielona do innego zadania, a następnie zwrócona poprzedniemu zadaniu, którego wykonywanie może być kontynuowane bez błędów. Zasób którego jednostki nie mają tej własności nazywa się zasobem nieprzywłaszczalnym. Ten podział na kategorie dotyczy głównie zasobów w systemach komputerowych. Głównym przykładem zasobów przywłaszczalnych są procesory censtralne. Uwzględnienie zasobów nieprzywłaszczalnych wymaga rozwiązania problemu zakleszczenia (deadline).

Zasobem dyskretnym nazywamy taki, który może być przydzielany zadaniom w liczbach jednostkowych będących elementami zbioru skończonego, natomiast zasobem ciągłym nazywamy zasób, który może być przydzielany w liczbach jednostek z danego przedziału (będącego zbiorem nieprzeliczalnym).

Zasób nazwiemy odnawialnym, jeśli tylko jego chwilowa dostępność jest ograniczona, nieodnawialnym - jeśli tylko jego zużycie do danego momentu (na ogół łączne zużycie, do momentu zakończenia projektu) jest ograniczone. Zasób podwójnie ograniczony to taki, dla którego zarówno chwilowa dostępność jak i zużycie są ograniczone. Typowym przykładem zasobów odnawialnych są procesory, maszyny, pamieć, zaś nieoficjalnych są nakłady finansowe. Warto zauważyć, że wiele zasobów to zasoby podwójnie ograniczone, typowym przykładem jest moc, która podlega ograniczeniu chwilowemu, jak i ograniczeniu zużycia (np. energia).

Wykłady

Wykład 1 (02.03.2009) 1
Wykład 2 (09.03.2009) 2
Wykład 3 (16.03.2009) 3
Wykład 4 (23.03.2009) 6
Wykład 5 (30.03.2009) 9
Wykład 6 (06.04.2009) 12
Wykład 7 (20.04.2009) 15
Wykład 8 (27.04.2009) 19
Wykład 9 (04.05.2009) 23
Wykład 10 (11.05.2009) 25
Wykład 11 (18.05.2009) 27
Wykład 12 (25.05.2009) 30
Wykład 13 (28.05.2009) 31
Wykład 14 (01.06.2009) 33
Wykład 15 (15.06.2009) 36

Literatura

[1] *Badania operacyjne dla informatyków*, J. Błażewicz, W. Cellary, R. Słowiński, J. Węglarz, wyd. WNT, 1983.

[2] *Badania operacyjne*, M. Siudak, Oficyna Wydawnicza Politechniki Warszawskiej, 1994.

[3] *Handbook on Scheduling: From Theory to Applications*, Springer, 2007. (problemy maszynowe + dodatkowe zasoby)

[4] *Perspectives in Modern Scheduling*, J.Józefowska, J. Węglarz (eds.), Kluwer, 2006.

[5] *Grid resource Management*, J. Nabrzyski, J. Schapf, J. Węglarz (eds.), Kluwer, 2003. (rozdział zasobów w gridach)