

Metody Optimalizacji w Zarządzaniu

Spis treści

1	Przydział miejsc w parlamencie	3
1.1	Tu kilka stron o początkowych tematach	3
1.2	Tijdeman	4
1.2.1	Sformułowanie problemu	4
1.2.2	Rozwiązanie	4
1.2.3	Wyznaczanie oddziałów spełniających nierówność . . .	4
1.3	PRV	4
1.4	Problem Liu-Laylanda	4
1.4.1	Algorytm statyczny	5
1.4.2	Algorytm dynamiczny	5
1.5	MAD - Mean Absolute Deviation	5
1.5.1	Sformułowanie problemu	5
1.5.2	Własności problemu	6
1.6	WSAD weighted sum of absolute deviations	6
1.6.1	Sformułowanie problemu	6
1.6.2	Własności	7
1.7	WSAD unrestriced	7
1.8	WSAD restriced	7
1.8.1	Właściwości	7
1.8.2	Omówienie algorytmu	7
1.9	Wsad z elastycznym czasem wykonania zadania	8
1.9.1	Sformułowanie problemu	8
1.9.2	Funkcja celu	8
1.9.3	Właściwości	8
1.10	Problem z symetrycznymi wagami	9
1.10.1	Sformułowanie problemu	9
1.10.2	Funkcja celu	9
1.10.3	Własności	9
1.10.4	Algorytm programowania dynamicznego	9
1.11	TWET - total weighted...	10
1.11.1	Sformułowanie problemu	10
1.11.2	Funkcja celu	10
1.11.3	Własności	10
1.11.4	Algorytm pełnego przeglądu	11
1.12	CTV - Completion Time Variance	11
1.12.1	Sformułowanie problemu	11
1.12.2	Cel	11
1.12.3	Własności	11
1.12.4	Algorytm	12

1.12.5	Algorytm programowania dynamicznego	12
1.13	Ten algorytm z tym takim wykresem fajnym	13
1.13.1	Sformułowanie problemu	13
1.13.2	Funkcja celu	13
1.13.3	Dodatkowe ograniczenie	13
1.13.4	Właściwości	13
1.13.5	Programowanie liniowe	13
1.13.6	Przykład algorytm zachłanny	14

1 Przydział miejsc w parlamencie

1.1 Tu kilka stron o początkowych tematach

1.2 Tijdeman

1.2.1 Sformułowanie problemu

k oddziałów towarzystwa

λ_i znormalizowana liczba członków w oddziale i , $i=1, \dots, k$

$$\sum_i^K \lambda_i = 1 \quad (1)$$

ω_i numer oddziału, którego pochodził przewodniczący w okresie j ; $j=1, \dots$;

$A\omega(i, n)$ liczba przewodniczących z oddziału i w okresie $[1, n]$;

zminimalizować $D(\omega)=$

1.2.2 Rozwiązanie

$$\min_{i,n} |\lambda_i n - A\omega(i, n)| \leq 1 - \frac{1}{2K-2} \quad (2)$$

1.2.3 Wyznaczanie oddziałów spełniających nierówność

$$\sigma_i = \lambda_i n - A\omega(i, n-1) \geq \frac{1}{2K-2} \quad (3)$$

$$\frac{1 - \frac{1}{2K-2} - \sigma_i}{\lambda_i} \rightarrow \min \quad (4)$$

1.3 PRV

1.4 Problem Liu-Laylanda

- jeden proces,
- n cyklicznych, niezależnych, podzielnych zadań o czasie wykonania C_i i okresie T_i ,
- zadanie musi być wykonane w okresie T_i

1.4.1 Algorytm statyczny

- posortuj zadania rosnąco według okresów T_i
- zadanie o krótszym okresie ma większy priorytet
- wykonuj zadanie tak długo jak nie pojawi się zadanie o wyższym priorytecie

1.4.2 Algorytm dynamiczny

- wyznacz dla każdego zadania linię krytyczną
- zadanie posiadające najbliższą linię krytyczną ma największy priorytet
- współczynnik wykorzystania pracy procesora

$$U = \sum_{i=1}^n \frac{C_i}{T_i} \quad (5)$$

- dla algorytmu ze stałym priorytetem

$$U = m(\sqrt[m]{2} - 1) \quad (6)$$

1.5 MAD - Mean Absolute Deviation

1.5.1 Sformułowanie problemu

- szeregowanie zadań na jednej maszynie
- n niezależnych i niepodzielnych zadań
- jeden, wspólny, żądany termin zakończenia dla wszystkich zadań $d_i = d$ dla wszystkich i
- t_i czas wykonania zadania i
- $f_i(e_i) = e_i$; $g_i(t_i) = t_i$
- wyprowadzenie funkcji celu

$$\sum_{i=1}^n |C_i - d| \quad (7)$$

1.5.2 Własności problemu

- najdłuższe zadanie uszeregowane jako pierwsze
- no idle time - brak przerwy między zadaniami
- V-shape uszeregowane zadania przed terminem zakończenia są uporządkowane według nierosnących czasów wykonania zadania, natomiast zadania które są uszeregowane po terminie zakończenia są uporządkowane według niemalejących czasów wykonania
- jedno zadanie kończy się dokładnie w żądanym terminie zakończenia $\left\lceil \frac{n}{2} \right\rceil$ zadań kończy się przed lub w żądanym terminie zakończenia
- problem restryktywny może przecinać zadanie

Algorytm Kaneta

1. $E = \phi, T = \phi$

2. WHILE $J \neq \phi$ DO

BEGIN Remove a job k from J such that $p_i = \max p_i$ Insert job k into the last position in E . If $J \neq \emptyset$ DO BEGIN remove a job k from J such that $p_i = \max(p_i)$ insert a job k into the first position in T END END
 $S = (E, T)$ (konkatenacja)

1.6 WSAD weighted sum of absolute deviations

1.6.1 Sformułowanie problemu

- n niezależnych, niepodzielnych zadań
- p_i czas wykonania zadania $i, i=1, \dots, n$
- d wspólny, żądany termin zakończenia
- α jednostkowy koszt wykonania zadania przed terminem
- β jednostkowy koszt wykonania zadania po terminie
- cel: zminimalizować

$$\sum_{i=1}^n (\alpha e_i + \beta t_i) \quad (8)$$

1.6.2 Własności

- no idle time
- V-shape
- dla problemu unrestricted zadanie k, gdzie $\left\lceil k^* = \frac{\beta n}{\alpha + \beta} \right\rceil$ kończy się w chwili d

1.7 WSAD unrestriced

Algorytm unrestricted (Bagchi, Sullivan, Chang, 1987)

- Zadania uporządkowane są według LPT (od najdłuższego)
- —E— liczba zadań w zbiorze E (przed terminem)
- —T— liczba zadań w zbiorze T (po terminie)

Krok 1. Zbiór E, T=0 Krok 2 For k=1 TO n DO BEGIN usuń zadanie k ze zbioru J IF $(\alpha * |E|) < (\beta * (|T| + 1))$ dodaj zadanie jako ostatnie do zbioru E ELSE dodaj na początek zbioru T END S=(E+T)

1.8 WSAD restriced

1.8.1 Właściwości

- NP trudny w zwykłym sensie
- zadania uporządkowane SPT (niemalejąco)

1.8.2 Omówienie algorytmu

Szereguję zadanie (k+1)

- szereguję z lewej strony
$$h_{k+1}^L = [\max(s + p_{k+1} - d, 0)\beta + \alpha \max(d - s - p_{k-1}, 0)] + h_k^*(s + p_{k+1})$$
- szereguję z prawej strony
$$h_{k+1}^p = \beta \left| \left(s + \sum_i^k p_i \right) \right| + h_k^*(s), \text{ dla spełnionego warunku } (s + \sum_{i=1}^{|n|} p_i \geq d)$$
- $h_{k+1}^*(s) = \min(h_{k+1}^L(s), h_{k+1}^p(s))$
- $h_n^* = \min h_n^*(s)$

1.9 Wsad z elastycznym czasem wykonania zadania

1.9.1 Sformułowanie problemu

- n zadań niezależnych i niepodzielnych
- p_i nominalny czas wykonania zadania i
- m_i maksymalne skrócenie zadania i
- p'_i rzeczywisty czas po skróceniu
- x_i rzeczywiste skrócenie
- γ_i koszt skrócenia zadania i o jednostkę
- α, β, d jak poprzednio, d nierestryktywne

1.9.2 Funkcja celu

Zminimalizować $\sum_i^n (\alpha e_i + \beta t_i + \gamma_i x_i)$

1.9.3 Właściwości

- no idle time
- V-shape względem czasów p'_i
- k kończy się w d , $k = \left\lceil \frac{n\beta}{\alpha+\beta} \right\rceil$
- Wzór na długość zadań przed i po terminie wykonania:
$$\Sigma = \alpha(L_1 + 2L_2 + \dots + (k-1)L_k) + \beta((n-k)R_1 + (n-k-1)R_2 + \dots + R_{n-k})$$
- koszt uszeregowania zadania i na pozycji k : $C_{ik} = p_i \omega_k$
- $\gamma_i < \omega_k, \gamma_i = \omega_k$ opłaca się skracać maksymalnie
- $\gamma_i > \omega_k$ nie opłaca się skracać
- $C_{ik} = \begin{cases} p'_i \omega_k + m_i \gamma_i, & \text{gdy } \gamma_i < \omega_k \\ p_i \omega_k, & \text{gdy } \gamma_i > \omega_k \end{cases}$
- Problem przydziału

– Zminimalizować $\sum_{i=1}^n \sum_{k=1}^n C_{ik} y_{ik}$ przy ograniczeniach:

- $\sum_{i=1}^n y_{ik} = 1, k=1, \dots, n$
- $\sum_{i=1}^k y_{ik} = 1, i=1, \dots, n$
- $y_{ik} \in \{0, 1\}$
- $y_{ik} = \begin{cases} 1, & \text{gdy zadanie } i \text{ jest uszeregowane na pozycji } k \\ 0, & \text{w przeciwnym razie} \end{cases}$

1.10 Problem z symetrycznymi wagami

1.10.1 Sformułowanie problemu

- n niepodzielnych, niezależnych zadań
- p_i czas wykonania zadania i
- $\alpha_i = \beta_i$ jednostkowy koszt wykonania zadania przed lub po terminie
- d wspólny żądany termin zakończenia

1.10.2 Funkcja celu

Wyprowadzenie: $\sum_{i=1}^n (e_i \alpha_i + \beta_i t_i) = \sum_{i=1}^n \alpha_i (e_i + t_i) = \sum_{i=1}^n \alpha_i |C_i - d|$

Funkcja celu: $\sum_{i=1}^n \alpha_i |C_i - d|$

1.10.3 Własności

- no idle time
- V-shape względem $\frac{p_i}{\alpha_i}$
- NP-trudny, nawet w wersji nieretryktywnej
- zadania posortowane według niemalejącej wartości $\frac{p_i}{\alpha_i}$

1.10.4 Algorytm programowania dynamicznego

- $h_k^L(s) = h_{k-1}^*(s + p_k) + \alpha_k |d - (s + p_k)|$
- $h_k^p(s) = h_{k-1}^*(s) + \alpha_k \left| \sum_{i=1}^k p_i + s - d \right|$

- $h_k^*(s) = \min(h_k^L(s), h_k^P(s))$
- $h_0 = 0$
- $h_n^* = h_k^*(s), s = 0, 1, \dots$

1.11 TWET - total weighted...

1.11.1 Sformułowanie problemu

- n niezależnych, niepodzielnych zadań
- p_i czas wykonania zadania i
- α_i jednostkowy koszt wykonania zadania i przed terminem
- β_i jednostkowy koszt wykonania zadania i po terminie
- d wspólny żądany termin zakończenia

1.11.2 Funkcja celu

zminimalizować $\sum_{i=1}^n (\alpha_i e_i + \beta_i t_i)$

1.11.3 Własności

- no idle time
- non-restricted - jedno zadanie kończy się w due-date
- NP-trudny nawet dla przypadku nierestryktywnego
- $E \in \{Ci < d\}, T \in \{Ci \geq d\}$, zadania w zbiorze T są uszeregowane według niemalejącej wartości $\frac{p_i}{\beta_i}$, a zadania w zbiorze E są uszeregowane według nierosnącej wartości $\frac{p_i}{\alpha_i}$
- uszeregowanie optymalne

$$\sum_{i \in T} \beta_i \leq \sum_{j \in E} \alpha_j \quad (9)$$

- jeżeli znamy optymalny podział zadań na zbiory E i T, to znalezienie uszeregowania optymalnego jest łatwe

- istnieje algorytm pseudo-wielomianowy dla szczególnego przypadku, gdy dla każdego i i j

$$\left(\frac{p_i}{\alpha_i} < \frac{p_j}{\alpha_j}\right) \Rightarrow \left(\frac{p_i}{\beta_i} < \frac{p_j}{\beta_j}\right) \quad (10)$$

1.11.4 Algorytm pełnego przeglądu

1. Porządkowanie zadań według nierosnących wartości $\frac{p_i}{\alpha_i}$
2. Wszystkie zadania umieszczamy w zbiorze E
3. Na każdym poziomie przenosimy jedno zadanie ze zbioru E na początek zbioru T
4. Jeżeli naruszone są warunki (I) lub (II) to kończymy gałąź. Jeżeli uszeregowanie jest dopuszczalne, to obliczamy jego koszt.
5. Rozwijamy drzewo do sprawdzenia wszystkich dopuszczalnych uszeregowień

1.12 CTV - Completion Time Variance

1.12.1 Sformułowanie problemu

- n niezależnych, niepodzielnych zadań
- d wspólny, żądany termin zakończenia
- p_i czas wykonania zadania $i=1, \dots, n$
- $\alpha_i = \beta_i = 1$ $i=1, \dots, n$

1.12.2 Cel

Zminimalizować $\sum_{i=1}^n (C_i - d)^2$

1.12.3 Własności

- $d = \frac{1}{n} \sum_{i=1}^n n C_i$
- $\sum_{i=1}^n (C_i - \frac{1}{n} \sum_{i=1}^n C_i)^2 = \sum_{i=1}^n (C_i - \bar{C})^2$ wariancja C_i
- no idle time

- najdłuższe zadanie uszeregowane jako pierwsze
- V-shape
- Algorytm Kaneta nie daje poprawnego rozwiązania problemu
- problem NP-trudny
- Zminimalizować $\sum_{i=1}^n (C_i - d)^k k > 0$
 - $0 < k \leq 1$ problem wielomianowy, algorytm Kaneta
 - $k \geq 2$ problem NP-trudny
 - $1 < k < 2$ problem otwarty

1.12.4 Algorytm

- algorytm programowania dynamicznego
- algorytm podziału i ograniczeń
- metaheurystyki
- algorytmy spektralne

1.12.5 Algorytm programowania dynamicznego

- zamiast wartości bezwzględnej, kwadrat
- $h_k^L(s) = h_{k-1}^*(s + p_k) + (s + p_k - d)^2$
- $h_k^p(s) = h_{k-1}^*(s) + (s + \sum_{i=1}^k p_i - d)^2$
- $h_k^* = \min\{h_k^L(s), h_k^p(s)\}$
- $h_0^*(s) = 0$
- $h_n^* = \min_n h_n^*(s)$

1.13 Ten algorytm z tym takim wykresem fajnym

1.13.1 Sformułowanie problemu

- n niezależnych, niepodzielnych zadań
- jedna maszyna
- d_i żądany termin zakończenia zadania i
- α_i jednostkowy koszt wykonania zadania i przed terminem
- β_i jednostkowy koszt wykonania zadania i po terminie

1.13.2 Funkcja celu

$$\sum_{i=1}^n (\lambda_i e_i + \beta_i t_i)$$

1.13.3 Dodatkowe ograniczenie

brak przerw w pracy procesora

1.13.4 Właściwości

- nawet dla przypadku $\alpha_i = \beta_i = 1$ problem jest silnie NP-trudny
- w literaturze rozważa się również przypadek z dodatkowym ograniczeniem na brak przerw w pracy procesora
- dla znanej sekwencji zadań ich optymalne położenie na osi czasu można znaleźć w czasie $O(n)$ (wielomianowym zależnym od liczby zadań)
 - model programowania liniowego
 - algorytm zachłanny $O(n \log n)$

1.13.5 Programowanie liniowe

- Zmienna decyzyjna C_i (termin zakończenia zadania i)
- Funkcja celu $\sum_{i=1}^n (\alpha_i e_i + \beta_i t_i)$
- Ograniczenia
 - $e_i \geq d_i - C_i$

- $e_i \geq 0$
- $t_i \geq C_i - d_i$
- $t_i \geq 0$
- $C_{i+1} \geq C_i + p_{i+1}$
- $C_1 \geq p_1$

- Liczba zmiennych: $3n$
- Liczba ograniczeń: $5n$

1.13.6 Przykład algorytm zachłanny

I tutaj ten wykresik. Wymiekam