Spis treści

1 Wprowadzenie

Teoria szeregowania zadań zajmuje się problemami polegającymi na przydzieleniu pewnych zadań do dostępnych maszyn w taki sposób, aby pewne kryterium było optymalizowane. Będziemy się zajmować deterministycznymi problemami, czyli takimi, w których wszystkie dane są znane z góry.

2 Postać problemu

Standardowo, problem jest skonstruowany z następujących składowych:

- zadania $\mathcal{J} = \{J_1, \ldots, J_n\}$
- maszyny $\mathcal{P} = \{P_1, \ldots, P_m\}$
- zasoby $\mathcal{R} = \{R_1, \ldots, R_s\}$ dostępnych w $m_1, \ldots, m_s$ jednostkach

2.1 Maszyny

W problemie w zależności od wykonywanego zadania i maszyn mogą występować różne ograniczenia i różnice między maszynami. Jeśli mamy do czynienia z kilkoma maszynami równoległymi to te maszyny mogą być:

- P - identycznościowe czyli z jednakową szybkością
- Q - jednorodne czyli z różną szybkością między maszynami
- R - dowolne czyli z różniącą się szybkością między zadaniami i maszynami

Jeśli mamy do czynienia z maszynami dedykowanymi, gdzie każde zadanie składa się z operacji wykonywanych na różnych maszynach to maszyny mogą być:

- F - system przepływowy czyli każde zadanie przechodzi przez maszyny w tej samej kolejności
- O - system otwarty czyli kolejność wykonywania operacji jest dowolna
- J - system gniazdowy czyli każde zadanie ma ustaloną własną kolejność przechodzenia przez maszyny

2.2 Zadania

Zadanie J opisują następujące atrybuty:

- p_j - czas wykonania zadania J_j
- r_j - czas przygotowania zadania J_j
- d_j - pożądany czas zakończenia zadania J_j
- w_j - waga zadania J_j

2.3 Parametry zadań

Zbiór zadań $\mathcal{J}$ jako całość opisują ograniczenia kolejnościowe (acykliczny graf skierowany), oraz podzielność czyli czy zadania można przerywać i wznawiać.

2.4 Uszeregowanie

Uszeregowaniem nazywamy przypisanie każdemu zadaniu maszyny i zasobów w czasie. Koniecznym jest aby następujące warunki były spełnione:

- w każdej chwili maszyna wykonuje tylko jedno zadanie
- w każdej chwili każde zadanie jest wykonywane przez jedną maszynę
- Każde zadanie jest wykonywane w całości
- Spełnione są ograniczenia kolejnościowe
- Jeśli zadania są podzielne to są one przerywane skończoną ilość razy

2.4.1 Parametry uszeregowania

- moment rozpoczęcia S_j
- moment zakończenia C_j
- czas przepływu $F_j = C_j - S_j$
- opóźnienie $L_j = C_j - d_j$
- spóźnienie $T_j = \max(0, L_j)$
- przyspieszenie $E_j = \max(0, d_j - C_j)$
- liczbę spóźnionych zadań $U_j = |\{i : C_i > d_i\}|$

2.4.2 Kryteria optymalizacji

Typowo w szeregowaniu optymalizujemy jakąś funkcję składającą się z parametrów uszeregowania. Przykładowe funkcje to: $C_{max} = \max(C_j)$, $C_{sum} = \sum C_j$ czy $T_{sum} = \sum T_j$.

2.5 Notacja Trójpolowa

$$\alpha|\beta|\gamma$$

gdzie α określa ograniczenia maszyn, β określa ograniczenia zadań, a γ określa kryterium optymalizacji.

3 Problemy na jednej maszynie

Poniżej są opisane raczej trywialne problemy z minimalnymi utrudnieniami.

3.1 $1||C_{max}$

Jest jedna maszyna i n zadań z czasami przetwarzania p_j. Każde uszeregowanie bez przestojów jest optymalne. $C_{max} = \sum_{j=1}^{n} C_j$.

3.2 $1|r_j|C_{max}$

Jest jedna maszyna i n zadań z czasami przetwarzania p_j i czasami, od których są dostępne r_j. Tutaj możliwe, że przestoje są nieuniknione. Aby rozwiązać ten problem, sortujemy zadania po r, po czym kolejno je szeregujemy.

3.3 $1||\sum C_j$

Jest jedna maszyna i n zadań z czasami przetwarzania p_j. Chcemy minimalizować sumę ich czasów zakończeń, więc lepiej najpierw wykonać najkrótsze zadania. Nazywamy to SPT.

3.4 $1||\sum w_j C_j$

Jest jedna maszyna i n zadań z czasami przetwarzania p_j i wagami w_j. Chcemy minimalizować ważoną sumę ich zakończeń, co za tym idzie najwyżej ważone zadania chcemy wykonywać jako pierwsze. Aby to osiągnąć wystarczy posortować zadania po $\frac{p_j}{w_j}$. Nazywamy to WSPT i jest to ogólniejszy przypadek SPT.

3.5 $1|pmtn, r_j|\sum C_j$

Jest jedna maszyna i n zadań z czasami przetwarzania p_j, z czasami gotowości r_j, które **są podzielne**. Tutaj ponownie stosujemy algorytm SPT, ale w momencie, gdy skończymy zadanie, lub zadanie stanie się dostępne, zmieniamy wykonywane zadanie na to o najkrótszym czasie przetwarzania. Nazywamy to SRPT.

3.6 Problemy kolejnościowe

W momencie dodania ograniczeń kolejnościowych, z reguły opisanych grafem kolejności, złożoność problemu znacznie rośnie. Załóżmy, że graf jest reprezentowany przez listę sąsiedztwa.

3.6.1 1|prec|$\sum C_j$

W tym problemie mamy jedną maszynę i n zadań nieprzerywanlnych z czasami przetwarzania p_j. Próba rozwiązania tego problemu tylko przy pomocy DFS nie wystarczy. Konieczny jest algorytm Kahna, który sortuje topologicznie graf. Algorytm Kahna jest bardzo podobny do DFS, ale zamiast używania stosu, używamy kolejki priorytetowej, czyli najpierw zwiedzamy te wierzchołki o najkrótszym czasie przetwarzania.

3.6.2 1|prec|f_{max}

W tym problemie mamy jedną maszynę i n zadań nieprzerywalnych z czasami przetwarzania p_j. Wystarczy rozważać szeregowania bez przestojów. Problem ten rozwiązuje algorytm Lawler'a, w którym budujemy uszeregowanie od końca. W każdym kroku algorytmu rozważamy te zadania, bez uszeregowanych następników, i wybieramy te które w danym momencie generuje najmniejszy koszt. Wybrane zadanie trafia na początek uszeregowania.

3.6.3 1|out-tree|$\sum w_j C_j$

W tym problemie mamy jedną maszynę i n zadań nieprzerywalnych z czasami przetwarzania p_j oraz wagami. Graf kolejnościowy jest podany jako out-tree, czyli korzeń nie ma poprzedników. Trik aby rozwiązać ten problem, to łączyć kolejno zadania o najmniejszej wartości kryterium $(\frac{w_j}{p_j})$ z swoimi poprzednikami, pamiętając w poprzednikach o kolejności łączenia.

3.6.4 1|in-tree|$\sum w_j C_j$

Graf kolejnościowy jest podany jako in-tree, czyli korzeń ma poprzedników, zgodnie z tym ile ma dzieci. Ten problem mapuje się 1-1 do problemu poprzedniego, wystarczy tylko odwrócić krawędzie i nadać zadaniom wagi przeciwne.

3.7 Problemy z opóźnieniem

W tych problemach, optymalizujemy opóźnienie, czyli $C_j - d_j$.

3.7.1 1||L_{max}

Mamy jedną maszynę i n zadań nieprzerywalnych z czasami przetwarzania p_j i czasami oczekiwanymi d_j. Aby rozwiązać ten problem, wystarczy zastosować algorytm EDD (regułę Jacksona), w którym wybieramy najpierw zadanie o najwcześniejszym czasie oczekiwania.

3.7.2 1|prec|L_{max}

Mamy jedną maszynę i n zadań nieprzerywalnych oraz graf kolejności. Ten problem rozwiązuje algorytm analogiczny do problemu 1|prec|C_{max}, czyli sortujemy topologicznie, ale tym raze po d_j a nie p_j.

3.7.3 1|r_j|L_{max}

To jest problem silnie NP-trudny, trudność wynika z niepodzielności zadań. Warto wspomnieć, że ten problem nie jest NP-trudny, jeśli $r_j = 1$, i wtedy rozwiązuje go EDD.

3.7.4 1|pmtn,r_j|L_{max}

W odróżnieniu od poprzedniego problemu, tutaj zadania są podzielne. Aby rozwiązać ten problem wystarczy zastosować regułę EDD, pamiętając o podzielności zadań. Zawsze jak zadanie kończy wykonywanie, lub nowe zadanie jest dostępne, to wybieramy zadanie o najmniejszym d_j.

3.8 $1||\sum U_j$

Tutaj optymalizujemy ilość spóźnionych zadań. W pewnym sensie nie interesuje nas ile jesteśmy spóźnieni, ale czy jesteśmy spóźnieni. Czyli mamy n zadań o czasie przetwarzania p_j i czasie oczekiwania d_j. Aby rozwiązać ten problem, wybieramy niespóźnione zadania w porządku EDD, potem robimy te spóźnione w dowolnym porządku. Czyli dobierasz do zbioru zadania, jeśli dodanie kolejnego zadania spowoduje, że zadanie jest spóźnione to usuń z zbioru zadanie o największym czasie wykonania.

3.9 $1||\sum T_j$

Problem ten jest słabo NP-trudny, generalizacja z wagami ($\sum w_j T_j$) jest silnie NP-trudna. Wersja z przerywaniem też jest silnie NP-trudna. Z kolei $p_j = 1$ znacznie ułatwia obydwa problemy i powoduje że należą do klasy P.

3.10 $1||\sum w_j E_j$

Problem ten jest równoważny z problemem $1||\sum w_j T_j$.

4 Problemy wielu maszyn

Problemy wielu maszyn mają okropną tendencję bycia NP-trudnymi. Problem $2||C_{max}$ jest NP-trudny. Problemy wielu maszyn często aproksymują algorytmy listowe, w których do n wolnych maszyn przyporządkowujemy kolejno zadania z *jakiejś* kolejki. Im większa jest różnica między najdłuższym a najkrótszym czasem wykonania zadania, tym bardziej mylne rozwiązania dają algorytmy listowe.

4.1 $\mathbf{P}||\sum C_j$

Ten problem rozwiązuje algorytm LPT, czyli sortujemy zadania nierosnąco wobec p_j i przypisujemy je kolejno do maszyn. Jest to algorytm listowy, w którym zadania w liście są posortowane nierosnąco wobec p_j.

4.2 $\mathbf{P}|\mathbf{prec}|C_{max}$

Ten problem może rozwiązać LPT, z modyfikacją w której jeśli zadanie nie ma spełnionych wymagań to nie może być zdjęte z kolejki. Niestety, ten problem jest szczególnie podatny na anomalie szeregowania listowego, gdzie wejścia "łatwiejsze" mogą dawać gorsze rezultaty niż "trudniejsze".

4.3 $\mathbf{P}|\textbf{in-tree,}p_j = 1|C_{max}$

Mamy m identycznych maszyn równoległych Poziom zadania to jego odległość od korzenia. Problem ten może rozwiązać algorytm Hu, gdzie w każdej chwili t, wybieramy m zadań z najwyższymi poziomami.

4.4 $\mathbf{P}|\mathbf{pmtn}|C_{max}$

Mamy m identycznych maszyn równoległych i n podzielnych zadań.

$$C^*_{max} = \max\{\frac{1}{m}\sum_{j=1}^{n} p_j, \max_{j=1}^{n} p_j\}$$

Optymalne uszeregowanie konstruuje algorytm McNaughton'a. Zadania są przypisywane do kolejnych maszyn w przedziale $[0, C^*_{max})$. Jeśli zadanie miałoby się kończyć w czasie $> C^*_{max}$, to jego "overflow" trafia na następną maszynę.

4.5 $\mathbf{P}|p_j = 1, r_j|L_{max}$

Jako, że zadania są równej długości, to jest to zdecydowanie prostszy problem. Kolejno wybieramy m zadań do wykonania w "turze", najpierw wykonując te, które mają najwcześniejsze terminy zakończenia.

4.6 P|in-tree,$p_j = 1|L_{max}$

Mamy m identycznych maszyn równoległych. Aby rozwiązać ten problem, wystarczy zmodyfkować $d_i = \min\{d_i, d_{i-1}\}$, a następnie szeregować zgodnie z EDD.

4.7 P|$p_j = 1|\sum w_j U_j$

Problem sprowadza się do wybrania zbioru zadań, które zostaną wykonane na czas. Zadania nie spóźnione można przydzielić do maszyn LPT z porządkiem EDD. Czyli najpierw sortujemy zadania według d_j, potem dodajemy do zbioru zadania zgodnie z tym porządkiem. Jeśli dodanie zadania powoduje przekroczenie limitu czasu, oraz to zadanie ma wagę wieksza niż najmniejsza waga zadania w zbiorze, to usuwamy zadanie o najmniejszej wadze.

4.8 Maszyny nie-identyczne

Maszyny niekoniecznie muszą być identyczne, ale jednorodne. Mogą być różne w pojemnościach przetwarzania. Prędkość oznaczamy v_i. Czas wykonywania zadania j na maszynie i wynosi $\frac{p_j}{v_i}$.

4.8.1 Q||$\sum C_j$

Zadania sortujemy i wykonujemy z porządkiem SPT. Jeśli $t_j = \frac{k}{v_i}$, to zadanie J_j należy uszeregować na maszynie M_i jako k-te od końca. Niech $w_i = \frac{1}{v_i} : i < m$. Następnie dla każdego zadania znajdujemy najmniejsze w_i w liście, umieszczamy zadanie na początek uszeregowania i-tej maszyny, po czym $w_i += \frac{1}{v_i}$.

4.8.2 Q|pmtn|$\sum C_j$

Najktótsze zadanie chcemy wykonywać na maszynie o najwyższej prędkości. W momencie wykonania zadania, przerzucamy drugie najkrótsze zadanie na tą maszynę, itd.

5 Programowanie dynamiczne

Programowanie dynamiczne, to technika w której algorytm rozwiązuje problem rekurencyjnie na podstawie ogromnej tablicy w której wyniki tymczasowe są przechowywane.

5.1 Problem plecakowy

Mamy n przedmiotów o wagach w_i i cenach c_i. Mamy też daną maksymalny łączny rozmiar B. Maksymalna łączna cena podzbioru przedmiotów o numerach nieprzekraczających i mieszczących się w plecaku o pojemności j:

$$A[i,j] = \begin{cases} 0 & i = 0 \lor j = 0 \\ A[i-1,j] & i,j > 0 \land w_i > j_j \\ \max\{A[i-1,j], A[i-1,j-w_i] + c_i\} & i,j > 0 \land w_i \leq j \end{cases}$$

Tworzymy tablicę A o rozmiarach $n+1 \times B+1$ i zwracamy $A[n,B]$. Aby stworzyć rozwiązanie, dla $j = B$, oraz $i = n$, jeśli $A[i,j] \neq A[i-1,j]$ wkładamy przedmiot i do plecaka, $j -= w_i$ oraz $i -= 1$.

5.2 P2||C_{max}

Mamy problem plecakowy, gdzie $w_j = c_j = p_j$, oraz $B = \lfloor \sum_{j=1}^{n} \frac{p_j}{2} \rfloor$.

5.3 Pm||C_{max}

Wykorzystujemy tablicę $m+1$ wymiarową. W komórce $A[j, t_1, \ldots, t_m]$ zapisujemy informację czy zadania o numerach nieprzekraczających j można przypisać w taki sposób, że czas działania maszyny $P_i = t_i$.

1. $$A[0, t_1, \ldots, t_m] = false$$

2.
$$A[0, 0, \dots, 0] = true$$

3.
$$A[1..n, t_1, \dots, t_m] = \bigvee_{i=1}^{n} A[j-1, t_1, \dots, t_{i-1}, t_i - p_j, t_{i+1}, \dots, t_m]$$

4. return $\min\{\max\{t_1, \dots, t_m\} | A[n, t_1, \dots, t_m] = true\}$