Wrocław, 13 maja 2021

# **S**terowanie **P**rocesami **D**yskretnymi

Laboratorium 5

*Opracowanie i implementacja algorytmów dla problemu $FP||C_{\max}$ lub $J||C_{\max}$*

prowadzący: *mgr inż. Radosław Idzikowski*

Celem laboratorium jest zapoznanie się metodami przybliżonymi na przykładzie problemów wielomaszynowych. Obejmuje to odpowiednie zdefiniowane problemu (ograniczeń i funkcji celu), danych wejściowych oraz implementacje dedykowanych algorytmów, a także interpretację wyników.

## 1 Przebieg zajęć

Laboratorium obejmuje zajęcia nr 10-11 (4 godzin zajęć, w tym oddanie). Praca odbywa się w ramach dwuosobowych zespołów. Każdy zespół otrzymuje do opracowania problem do wyboru. W tym wypadku $FP||C_{\max}$ lub $J||C_{\max}$ oraz dedykowane dla niego algorytmy.

W trakcie pierwszych zajęć w wybranym języku `python`, `C/C++`, `Java` lub `C#` należy zaimplementować metodę generowania instancji problemu przy użyciu załączonych generatorów dla zadanych parametrów (źródło, rozmiar, zakres). Kolejnym krokiem jest implementacja metody oceniania rozwiązania (liczenia funkcji celu). Następnie należy zaimplementować algorytm konstrukcyjny.

Podczas ostatnich zajęć z tego tematu (zajęcia nr 11) będzie czas na dokończenie rozpoczętych zadań oraz na ocenę efektu końcowego. Na ocenę `3.0` należy poprawnie zaimplementować algorytm konstrukcyjny NEH dla problemu $FP||C_{\max}$ za zaimplementowanie dwóch wybranych modyfikacji można uzyskać maksymalnie `+1.0` do oceny. Za implementacje algorytmu konstrukcyjnego INSA dla problemu $J||C_{\max}$ jest przewidzenia ocena `5.0`.

## 2 Problem

### 2.1 Permutacyjny problem przepływowy

Problem $FP||C_{\max}$ jest szczególnym przypadkiem problemu ogólniejszego $F||C_{\max}$, gdzie na każdej maszynie będziemy mieli taką samą kolejność wykonywania zadań. Mamy zbiór $n$ zadań wykonywanych na maszynach:

$$\mathcal{J} = \{1, 2, \ldots, n\}, \tag{1}$$

które należy wykonać na $m$ maszynach:

$$\mathcal{M} = \{1, 2, \ldots, m\}, \tag{2}$$

każde $j$-te zadanie składa się dokładnie z $m$ operacji

$$\mathcal{O}_j = \{o_{1j}, o_{2j}, \ldots, o_{mj}\}. \tag{3}$$

Każda operacja $o_{ij}$ z zadania $j$ jest wykonywana kolejno na następnej maszynie $i$ według kolejności technologicznej w tym wypadku $1 \rightarrow 2 \rightarrow \ldots \rightarrow m$ . Czas wykonania (*performed time*) operacji $o_{ij}$ wynosi $p_{ij}$. Na każdej maszynie naraz może wykonywać się tylko jedna operacja. W problemie przepływowym w ramach zadań musi być zachowany porządek technologiczny, tzn. aby mogła się zacząć

wykonywać kolejna operacja najpierw musi się wykonać operacja poprzednia z tego samego zadania. Przez $\pi$ oznaczymy kolejność wykonywania zadań (w permutacyjnym problemie przepływowym na każdej maszynie mamy tą samą kolejność wykonywania zadań).

W celu utworzenia harmonogramu dla zadanej permutacji $\pi$ musi utworzyć macierz $S$ momentów rozpoczęcia operacji oraz macierz $C$ momentów zakończenia operacji.

$$S = \begin{bmatrix} S_{11} & S_{12} & S_{13} & \dots & S_{1n} \\ S_{21} & S_{22} & S_{23} & \dots & S_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ S_{m1} & S_{m2} & S_{m3} & \dots & S_{mn} \end{bmatrix} \tag{4}$$

$$C = \begin{bmatrix} C_{11} & C_{12} & C_{13} & \dots & C_{1n} \\ C_{21} & C_{22} & C_{23} & \dots & C_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ C_{m1} & C_{m2} & C_{m3} & \dots & C_{mn} \end{bmatrix} \tag{5}$$

Przy układaniu harmonogramu należy pamiętać o dwóch ograniczeniach. Po pierwsze, że aby mogła się zacząć wykonywać operacja aktualnego zadania na tej samej maszynie, najpierw musi się zakończyć operacja z zadania poprzedniego:

$$S_{i\pi(j+1)} \geqslant C_{i\pi(j)} \tag{6}$$

Po drugie, musi być zachowany porządek technologiczny, więc aby móc rozpocząć wykonywania aktualnej operacji z zadania to musi się zakończyć wykonywanie poprzedniej operacji w ramach tego samego zadania na poprzedniej maszynie:

$$S_{i+1\pi(j)} \geqslant C_{i\pi(j)} \tag{7}$$

Uwzględniając oba ograniczenia możemy wyznaczyć czas rozpoczęcia operacji wzorem:

$$S_{i\pi(j)} = max\{C_{i-1\pi(j)}, C_{i\pi(j-1)}\}. \tag{8}$$

Rozpatrywanym kryterium optymalizacyjnym jest czas zakończenia wykonywania wszystkich zadań $C_{\max}$

$$C_{\max} = \max_{j \in \mathcal{J}} C_{m,\pi(j)}. \tag{9}$$

## 2.2 Problem gniazdowy

Podobnie jak w problemie przepływowym, w problemie $J||C_{\max}$ mamy zbiór $n$ zadań $\mathcal{J}$ oraz zbiór $m$ maszyn $\mathcal{M}$. Najważniejsze różnice:

- Każde zadanie może mieć inną liczbę operacji.
- Każde zadanie może mieć inną marszrutę technologiczną, więc może zaczynać się na dowolnej maszynie, a także część maszyn może zostać pominięta.
- W ramach jednego zadania różne operacje mogą wykonywać na tej samej maszynie, więc liczba operacji może być większa niż liczba maszyn.

Podsumowując każe zadanie składa się z $o_j$ operacji dla $j \in \mathcal{J}$. Dla ułatwienia możemy zmienić numerację operacji, numerując kolejno wszystkie operacje ze wszystkich zadani, wtedy zbiór wszystkich operacji będzie można zapisać następująco:

$$\mathcal{O} = \{1, 2, \dots, o_j, o_j + 1, \dots, o_j + o_{j+1}, o_j + o_{j+1} + 1, \dots, \sum_{k \in \mathcal{J}} o_k\} \tag{10}$$

Do rozwiązania problemu można użyć dwóch reprezentacji.

### 2.2.1 Krotka permutacji

W zależności od reprezentacji dane wejściowe będą się trochę różnić:

- czas wykonywania $i$-tej operacji w zadaniu $j$-tym wynosi $p_{i,j}$,
- $i$-tą operacje w zadaniu $j$-tym należy wykonać na maszynie $\mu_{i,j}$

Rozwiązaniem będzie krotka permutacji $\boldsymbol{\pi} = (\pi_1, \pi_2, \ldots, \pi_m)$, gdzie $\pi_i$ to kolejność wykonywania zadań na $i$-tej maszynie. W celu utworzenia harmonogramu dla zadanej permutacji $\boldsymbol{\pi}$ należy wyznaczyć momenty rozpoczęcia wykonywania oraz momenty zakończenia wykonywania operacji. Należy pamiętać, że na wszystkich maszynach może być różna liczba operacji.

### 2.2.2 Jedna permutacja

W zależności od reprezentacji dane wejściowe będą się trochę różnić:

- czas wykonywania $k$-tej operacji wynosi $\rho_k$,
- $k$-tą operacje należy wykonać na maszynie $\eta_k$,

Rozwiązaniem będzie jedna kolejność wykonywania wszystkich operacji $\pi$ o długości równiej liczbie operacji $|\mathcal{O}|$. W celu utworzenia harmonogramu dla zadanej permutacji $\pi$ należy wyznaczyć momenty rozpoczęcia wykonywania oraz momenty zakończenia wykonywania operacji. Należy uważać, ponieważ nie wszystkie rozwiązania są dopuszczalne.

# 3 Metody rozwiązania

## 3.1 Algorytm NEH

Algorytm o charakterze konstrukcyjnym dla $FP||C_{\max}$ opiera się na metodzie wstawiania coraz to mniejszych zadań konsekwentnie na wszystkich możliwych pozycjach. Należy zacząć od posortowania malejąco zadań względem sumy czasów wykonywania operacji wewnątrz każdego zadania. Można użyć odwróconej kolejki priorytetowej lub stworzyć zbiór $\mathcal{W}$ oraz następnie go posortować. Teraz w pętli należy wziąć zadanie $j^*$ o największej sumie operacji oraz wstawić je na wszystkich kolejnych, możliwych pozycjach w permutacji częściowej $\pi'$ (dla zadania $k$-tego w kolejności będzie $k$ wstawień) oraz wybrać tą pozycję, dla której $C_{max}$ jest najmniejszy. Następnie przejść do kolejnego zadania. Schemat metody pokazano w Algorytmie 1.

### 3.1.1 NEH+

Algorytm NEH można poprawić poprzez dodanie drugiej fazy wstawień dla zadań już przydzielonych. Należy wybrać zadanie $x$ według jednej z czterech reguł pomijając zadanie, które właśnie wstawiliśmy:

1. Zadanie zawierające najdłuższą operacje na ścieżce krytycznej.
2. Zadanie zawierające największą sumę operacji wchodzących w ścieżkę krytyczną.
3. Zadanie zawierające największą liczbę operacji wchodzących w ścieżkę krytyczną
4. Zadanie, którego usunięcie spowoduję największe zmniejszenie wartości $C_{max}$.

**Algorithm 1** NEH Algorithm

1: $\mathcal{N} \leftarrow \mathcal{J}$
2: $k \leftarrow 1$
3: **procedure** NEH($\mathcal{N}$)
4: **for each** $j \in \mathcal{N}$ **do**
5: $\omega_j \leftarrow \sum_{i \in \mathcal{M}} p_{ij}$
6: $\mathcal{W}$.PUSH($\omega_j, j$)
7: **end for**
8: **while** $\mathcal{W} \neq \emptyset$ **do**
9: $j^* \leftarrow \underset{j \in \mathcal{W}}{\operatorname{argmax}}\, \omega_j$
10: **for** $l \leftarrow 1, l \leqslant k$ **do**
11: **if** $\pi'$.INSERT($j^*, l$).$C_{max} < \pi^*.C_{max}$ **then**
12: $\pi^* \leftarrow \pi'$.INSERT($j^*, l$)
13: **end if**
14: $l \leftarrow l + 1$
15: **end for**
16: $\pi' \leftarrow \pi^*$
17: $\mathcal{W} \leftarrow \mathcal{W} \setminus \{j^*\}$
18: $k \leftarrow k + 1$
19: **end while**
20: **end procedure**

### 3.1.2 Algorytm INSA

Algorytm INSA dla problemu $J||C_{\max}$ działa analogicznie jak algorytm NEH dla $FP||C_{\max}$. Największą różnicą jest to, że należy dokonać wstawiań operacji niezależnie na wszystkich pozycjach na odpowiedniej maszynie.

# 4 Sposób generowania instancji

Dla parametru $n$ oraz ziarna $Z$:

1. init($Z$).
2. Dla każdego $j \in \mathcal{J}$ :
   (a) $o_j \leftarrow \text{nextInt}(1, \lfloor m \cdot 1,2 \rfloor)$.
   (b) Dla każdego $k \in \{1, 2, \ldots, o_j\}$ : $p_{k,j} \leftarrow \text{nextInt}(1, 29)$.
3. Dla każdego $j \in \mathcal{J}$ :
   (a) Dla każdego $k \in \{1, 2, \ldots, o_j\}$ : $\mu_{k,j} \leftarrow \text{nextInt}(1, m)$.

opracował: *Radosław Idzikowski*