Wrocław, 15 kwietnia 2021

Sterowanie **P**rocesami **D**yskretnymi

Laboratorium 3

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

prowadzący: *mgr inż. Radosław Idzikowski*

1 Cel laboratorium

Celem laboratorium jest zapoznanie się z podstawami teorii szeregowania zadań oraz sposobami modelowania oraz rozwiązywania podestowych problemów wielomaszynowych na przykładzie permutacyjnego problemu przepływowego (*Permutation Flow Shop Problem* lub *Permutation Flow Shop Scheduling Problem*). Obejmuje to odpowiednie zdefiniowane problemu (ograniczeń i funkcji celu), danych wejściowych oraz implementacje dedykowanych algorytmów, a także interpretację wyników.

2 Przebieg zajęć

Laboratorium obejmuje zajęcia nr 5-7 (6 godzin zajęć w tym oddanie). Praca odbywa się w ramach dwuosobowych zespołów. Każdy zespół otrzymuje do opracowania ten problem. W tym wypadku $FP||C_{\max}$ oraz dedykowane dla niego algorytmy.

W trakcie zajęć nr 5 w wybranym języku `python`, `C/C++`, `Java` lub `C#` należy zaimplementować metodę generowania instancji przy użyciu załączonych generatorów dla zadanych parametrów (źródło, rozmiar, zakres). Kolejnym krokiem jest implementacja metody oceniania rozwiązania (liczenia funkcji celu). Następnie należy zacząć implementacje algorytmu Johnsona dla dwóch maszyn.

W trakcie zajęć nr 6 należy dokończyć rozpoczęte zadanie z poprzednich zajęć. Następnie należy zaimplementować algorytm przeglądu zupełnego oraz rozpocząć pracę nad metodą podziału i ograniczeń.

Podczas ostatnich zajęć z drugiego tematu (zajęcia nr 7) będzie czas na dokończenie rozpoczętych zadań oraz na ocenę efektu końcowego. Na ocenę **3.0** należy poprawnie zaimplementować algorytm Johnsona dla $FP2||C_{\max}$. Za rozszerzenie algorytmu na więcej niż dwie maszyny ($m > 2$) jest przewidziane dodatkowe `+0.25` do oceny. Napisanie metody siłowej przy użyciu standardowych bibliotek podnosi oceną o `+0.25` lub w formie budowania drzewa o `+0.5`. Podstawowa wersja algorytmu bazującego na metodzie podziału i ograniczeń przewidziane jest `+0.5` oraz maksymalnie `+1.0` w zależności od jakości dolnych i górnych ograniczeń.

- Na ocenę `3.0` należy poprawnie zaimplementować algorytm Johnsona dla problemu $FP2||C_{\max}$.
- Na ocenę `4.0` należy poprawnie zaimplementować algorytm Johnsona dla problemu $FP2||C_{\max}$ oraz prosty BF i bardzo prosty BnB dla $FP||C_{\max}$ lub algorytm Johnsona oraz prosty BnB dla $FP||C_{\max}$.
- Na ocenę `5.0` należy poprawnie zaimplementować algorytm Johnsona, BF w formie drzewiastej oraz BnB z dobrym dolnym i górnym ograniczeniem dla $FP||C_{\max}$.

3 Problem

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 π 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.

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

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

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:

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

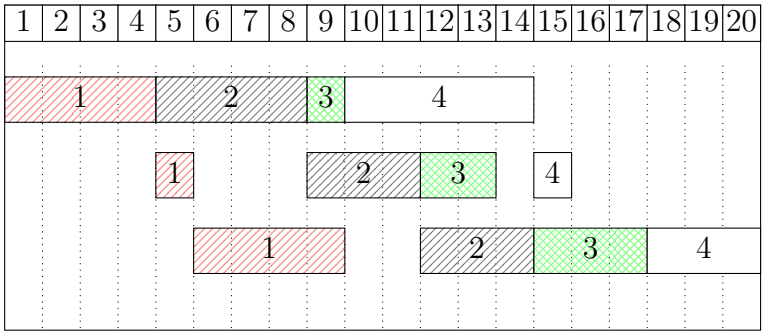
Rozpatrywanym kryterium optymalizacyjnym jest czas zakończenia wykonywania wszystkich zadań $C_{\max}$

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

4 Przykład

Na Rysunku 1 pokazano schemat gantta przedstawiający harmonogram dla danej kolejności π. Czas wykonywania operacji na maszynie jest zaprezentowany w formie bloczku. Operacje w ramach jednego zadania są oznaczone tym samym numerkiem oraz kolorem. Na schemacie dobrze widać, że na maszynach pojawiają się przerwy (przestoje). Można również łatwo zaobserwować, że $C_{\max}$ będzie zawsze równy momentowi zakończenia operacji na maszynie m z ostatniego zadania.

$$C_{\max} = C_{mn}. \tag{10}$$



Rysunek 1: Harmonogram problemu FSP dla $n = 4$, $m = 3$ oraz $\pi = (1, 2, 3, 4)$

Tabela 1: Instancja o rozmiarze $n = 4$ i $m = 3$

j	p_{1j}	p_{2j}	p_{3j}
1	4	1	4
2	4	3	3
3	1	2	3
4	5	1	3

5 Sposób generowania instancji

Dla parametru n, m oraz ziarna Z:

1. init(Z).
2. Dla każdego $j \in \mathcal{J}$:
   2.1 Dla każdego $i \in \mathcal{M}$: $p_{ij} \leftarrow \text{nextInt}(1, 29)$

6 Metody rozwiązania

6.1 Johnson

Algorytm Johnsona jest metodą dedykowaną dla problemu $FP2||C_{\max}$. Ogólna idea algorytmu polega na kolejnym znajdywaniu operacji o najkrótszym czasie wykonywania na pierwszej lub drugiej maszynie. Następnie należy przyporządkować całe zadania na początku kolejności wykonywania zdaań π jeśli krótsza operacja jest na maszynie pierwszej lub w przeciwnym wypadku na końcu. Procedura dla $FP2||C_{\max}$ zwraca rozwiązanie optymalne.

Algorithm 1 Johnson's Algorithm

procedure JOHNSON
  $l \leftarrow 1$
  $k \leftarrow n$
  $\mathcal{N} \leftarrow \mathcal{J}$
  while $\mathcal{N} \neq \emptyset$ **do**
    $j^*, i^* \leftarrow \underset{j \in \mathcal{N}, i \in \mathcal{M}}{\operatorname{argmin}} p_{ij}$
    if $p_{1j^*} < p_{2j^*}$ **then**
      $\pi(l) \leftarrow j^*$
      $l \leftarrow l + 1$
    else
      $\pi(k) \leftarrow j^*$
      $k \leftarrow k - 1$
    end if
    $\mathcal{N} \leftarrow \mathcal{N} \setminus \{j^*\}$
  end while
  return π
end procedure

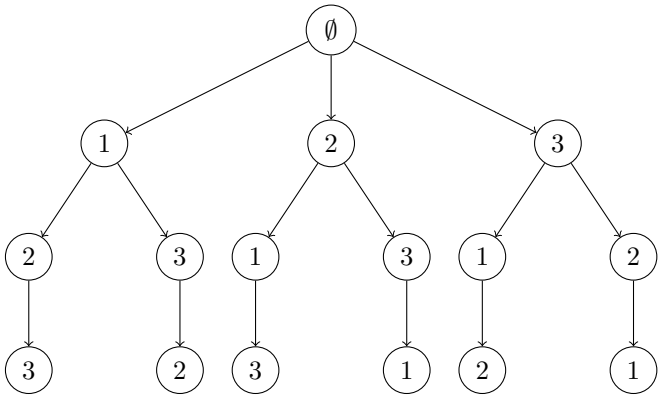
Istnieje wiele modyfikacji algorytmu Johnsona dla problemów o większej liczbie maszyn niż 2. Niestety wersje dla $m > 2$ nie mają gwarancji znalezienia optymalnego rozwiązania. Podstawa modyfikacja dla $m = 3$ polega to uwzględnieniu czas na „środkowej" maszyny, tzn. należy stworzyć dwie wirtualne maszyny. Przez p'_{ij} oznaczymy czas wykonywania na wirtualnych maszynach następująco:

$$\begin{matrix} p'_{1j} = p_{1j} + p_{2j} \\ p'_{2j} = p_{2j} + p_{3j} \end{matrix} \quad \text{dla } j \in \mathcal{J}. \tag{11}$$

Dla zmodyfikowanych danych może się posłużyć już podstawową wersją algorytmu Johnsona. Dla $m > 3$ można policzyć analogicznie jak dla $m = 3$ lub wziąć pod uwagę jedynie czasy wykonywania operacji na pierwszej i ostatniej maszynie.

6.2 Metoda siłowa

Sprawdzenie wszystkich możliwych kombinacji (*Brute Force*). Dla zbioru $\{1, 2, 3\}$ mamy $n!$ kombinacji: $(1, 2, 3)$, $(1, 3, 2)$, $(2, 1, 3)$, $(2, 3, 1)$, $(3, 1, 2)$ i $(3, 2, 1)$. Przeglądanie drzewa stanu jest analogiczne



Rysunek 2: Drzewo przestrzeni stanów dla $n = 3$

to sposobu przeglądania w metodzie podziału i ograniczeń z pominięciem „odcięcia".

6.3 Metoda podziału i ograniczeń

Podobnie jak przy algorytmie przeglądu zupełnego posłużymy się drzewem przestrzeni stanów przedstawionym na Rysunku 2. Algorytm w najgorszym wypadku może sprawdzić również wszystkie możliwe rozwiązania. Jednak aby tego uniknąć będziemy starać się odcinać "niekorzystne" rozwiązania oraz stosować różne strategie przeglądania.

Musimy zacząć od każdego wierzchołka, ponieważ badany problem nie ma charakteru cyklicznego. Następnie w danym węźle musimy przydzielić zadanie do maszyny oraz następnie obliczyć dolne oszacowanie (*lower bound*, LB). LB ogranicza rozwiązanie z dołu, to znaczy, że wartość $C_{\max}$ w rozpatrywanej gałęzi nie możne być lepsza niż wartość LB. Następnie jeśli węzeł będzie korzystny ($LB < UB$) czyli dolne oszacowanie będzie miało mniejszą wartość od górnego oszacowania (*upper bound*, UB) to węzeł będzie rozwijany. UB ogranicznia rozwiązanie z góry, to znaczy, że wartość $C_{\max}$ w rozpatrywanej gałęzi nie może być gorsza niż UB, za które jest podstawiona wrtość funkcji celu najlepszego znalezionego do tej pory rozwiązania. W przypadku dojścia do liścia należy policzyć rzeczywistą wartość $C_{\max}$ oraz uaktualnić UB, jeśli zachodzi taka potrzeba. W celu poprawienia wydajności algorytmu można stosować różne początkowe UB np.: rozwiązanie otrzymane algorytmem zachłannym. Oczywiście przy poprawianiu górnego ograniczenia należy pamiętać o zapamiętaniu permutacji π.

W ubogiej wersji algorytmu można podstawić początkowo nieskończoność za wartość górnego ograniczenia. Jednak jeśli chcemy zacząć odcinać jeszcze przed dojściem do liścia, możemy jako początkowe rozwiązanie wybrać: (1) permutację naturalną, (2) losowe (3) najlepsze z k-losowych (4) po innym algorytmie (np.: zmodyfikowany Johnson). Opisana metoda rekurencyjna reprezentuje sposób przeglądania drzewa stanu w głąb (*deep first search*). W literaturze można często spotkać również przeszukiwanie typu najpierw najlepszy (*best first search*) implementowane iteracyjne przechowując węzły w kolejce priorytetowej.

Sercem algorytmu jest funkcja licząca LB. Zazwyczaj dla jednego problemu istnieje więcej niż jedna metoda liczenia LB. W tym wypadku trzeba uważać na dwie rzeczy: (1) jeśli przeszacujemy i zaniżmy wartość może się okazać, że nie odetniemy ani jednej gałęzi, przez co algorytm wykona się dłużej niż przegląd zupełny, (2) jeśli zawyżymy wartość, możemy odciąć dojście do optymalnego rozwiązania, przez co algorytm nie będzie działał poprawnie.

Algorithm 2 Branch and Bound

```
1: 𝒩 ← 𝒥
2: UB ← InitUB
3: for each j ∈ 𝒩 do
4:     BnB(j, 𝒩, π)
5: end for
6: procedure BnB(j*, 𝒩, π)
7:     π.append(j*)
8:     𝒩 ← 𝒩 \ {j*}
9:     if 𝒩 ≠ ∅ then
10:        LB ← Bound(π, 𝒩)
11:        if LB < UB then
12:            for each j ∈ 𝒩 do
13:                BnB(j, 𝒩, π)
14:            end for
15:        end if
16:    else
17:        C_max ← Calculate(π)
18:        if C_max < UB then
19:            UB ← C_max
20:            π* ← π
21:        end if
22:    end if
23: end procedure
```

Jeśli mamy x zadań uszeregowanych oraz przez $\mathcal{N}$ oznaczymy zbiór zadań jeszcze nieuszeregowanych to dla problemu $FP||C_{\max}$ możemy wyznaczyć dolne oszacowanie na cztery sposoby:

$$LB_0 = C_{m,x} + \sum_{j \in \mathcal{N}} p_{m,j} \tag{12}$$

$$LB_1 = \max_{i \in \mathcal{M}} (C_{i,x} + \sum_{j \in \mathcal{N}} p_{i,j}) \tag{13}$$

$$LB_2 = \max_{i \in \mathcal{M}} (C_{i,x} + \sum_{j \in \mathcal{N}} p_{i,j} + \sum_{k=i+1}^{m} \min_{j \in \mathcal{J}} p_{k,j}) \tag{14}$$

$$LB_3 = \max_{i \in \mathcal{M}} (C_{i,x} + \sum_{j \in \mathcal{N}} p_{i,j} + \sum_{k=i+1}^{m} \min_{j \in \mathcal{N}} p_{k,j}) \tag{15}$$

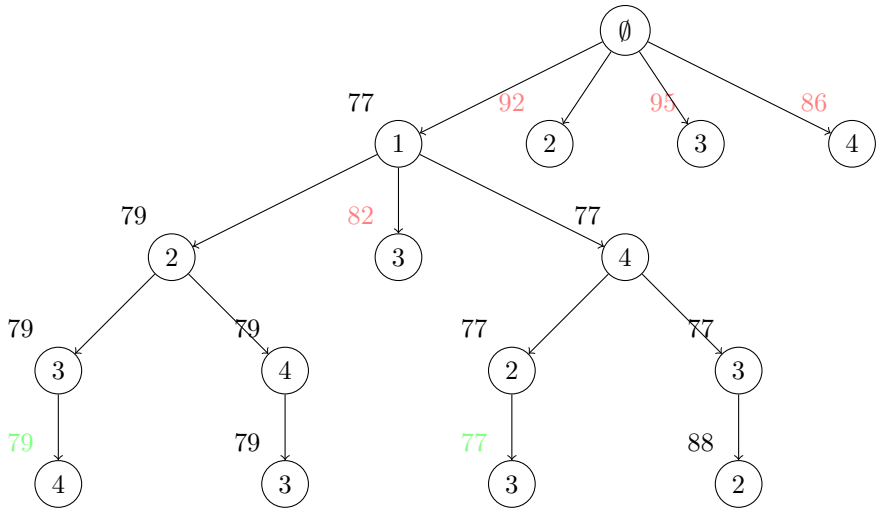
$$LB_4 = \max_{i \in \mathcal{M}} (C_{i,x} + \sum_{j \in \mathcal{N}} p_{i,j} + \min_{j \in \mathcal{N}} \sum_{k=i+1}^{m} p_{k,j}) \tag{16}$$

Załóżmy, że uszeregowane $x = 3$ zadań z $n = 6$ w kolejności $\pi = (2, 5, 1)$ na $m = 3$ maszynach, to dolne oszacowanie na podstawie wzoru (3) dla takiego rozwiązania wyniesie:

$$LB_3 = max \begin{cases} C_{1,3} + p_{1,3} + p_{1,4} + p_{1,6} + \min\{p_{2,3}, p_{2,4}, p_{2,6}\} + \min\{p_{3,3}, p_{3,4}, p_{3,6}\} \\ C_{2,3} + p_{2,3} + p_{2,4} + p_{2,6} + \min\{p_{3,3}, p_{3,4}, p_{3,6}\} \\ C_{3,3} + p_{3,3} + p_{3,4} + p_{3,6} \end{cases} \tag{17}$$

Tabela 2: Instancja o rozmiarze $n = 4$ i $m = 2$

zadanie	p_{1j}	p_{2j}
1	1	14
2	16	42
3	19	5
4	10	15



Rysunek 3: Metota podziału i ograniczeń

opracował: *Radosław Idzikowski*