Enumeration der Pläne

Gegeben: Partieller Plan $LR(SIJ)$ über $SIJ \subset I \times J$ und eine Operation $(i, j) \notin SIJ$.

Frage: Wie lässt sich die Operation (i, j) in den Plan einfügen, so dass

- alle Vorrangbedingungen aus $LR(SIJ)$ auch in $LR(SIJ \cup \{(i, j)\}$ enthalten sind und
- wieder ein Plan vorliegt?

Eigenschaften der Einfügung:

- Der neue Knoten muss so eingefügt werden, dass die **Sequence-Bedingung** erfüllt ist:
  Zu jeder Zahl $rk_{ij} > 1$ existiert entweder in Zeile i oder in Spalte j oder in beiden die Zahl $rk_{ij} - 1$.

- Von allen Nachfolgern der eingefügten Operation müssen die Ränge aktualisiert werden.

Es bezeichne $NF(i, j)$ die Menge aller Nachfolger von (i, j).

Enumeration der Pläne (2)

(1) (i, j) wird Quelle: $rk_{ij} = 1$. Aktualisiere alle rk_{kl} mit $(k, l) \in NF(i, j)$.

(2) (i, j) wird als direkter Nachfolger einer Operation (i, j^*) für Auftrag A_i oder einer Operation (i^*, j) auf Maschine M_j eingeordnet, d. h. setze
$rk_{ij} = rk_{ij^*} + 1$ für alle $(i, j^*) \in SIJ$ oder $rk_{ij} = rk_{i^*j} + 1$ für alle $(i^*, j) \in SIJ$.
Aktualisiere alle rk_{kl} mit $(k, l) \in NF(i, j)$.

(3) (i, j) wird als direkter Nachfolger einer Operation für A_i oder auf M_j und *gleichzeitig* als direkter Vorgänger einer Operation für M_j bzw. A_i mit nicht größerem Rang angeordnet.

(3-a) Wenn zwei Operationen (i, j^*) und (i^*, j) existieren, für die $rk_{ij^*} \geq rk_{i^*j}$ gilt, und es gibt keinen Weg von $(i^*, j) \rightarrow (i, j^*)$ in $G(LR(SIJ))$, dann setze:
$rk_{ij} := rk_{ij^*} + 1$ und $rk_{i^*j} := rk_{ij} + 1$.

(3-b) Wenn zwei Operationen (i^*, j) und (i, j^*) existieren, für die $rk_{i^*j} \geq rk_{ij^*}$ gilt, und es gibt keinen Weg von $(i^*, j) \rightarrow (i, j^*)$ in $G(LR(SIJ))$, dann setze:
$rk_{ij} := rk_{i^*j} + 1$ und $rk_{ij^*} := rk_{ij} + 1$.

Aktualisiere in beiden Fällen rk_{kl} für alle $(k, l) \in NF(i, j)$.

Beispiel zur Einfügung (1)

Beispiel 1. Gegeben ist der folgende partielle Plan $LR(SIJ)$, in den zusätzlich die Operation $(i, j) = (1, 3)$ eingefügt werden soll:

$$LR(SIJ) = \begin{pmatrix} 3 & 1 & . & 2 \\ . & 4 & . & 3 \\ . & 3 & 2 & 1 \\ 2 & . & 1 & . \end{pmatrix}$$

(1) liefert einen neuen Plan:

$$LR(SIJ) = \begin{pmatrix} 3^* & 1^* & \underline{1} & 2^* \\ . & 4^* & . & 3^* \\ . & 3^* & 2^* & 1 \\ 2^* & . & 1^* & . \end{pmatrix} \overset{rk_{13}=1}{\longrightarrow} LR(SIJ \cup \{(1,3)\} = \begin{pmatrix} 4 & 2 & 1 & 3 \\ . & 5 & . & 4 \\ . & 4 & 3 & 1 \\ 3 & . & 2 & . \end{pmatrix}$$

Die zu aktualisierenden Ränge sind mit * gekennzeichnet.

Beispiel zur Einfügung (2)

(2) liefert 3 neue Pläne: Die Operation $(1,3)$ wird als direkter Nachfolger von einer der Operationen aus $(1,1),(1,2),(1,4),(3,3),(4,3)$ eingefügt, d. h. $rk_{ij} \in \{2,3,4\}$. Wählt man $rk_{13} = 3$, so ergibt sich:

$$LR(SIJ) = \begin{pmatrix} 3^* & 1 & \underline{3} & 2 \\ . & 4 & . & 3 \\ . & 3 & 2 & 1 \\ 2 & . & 1 & . \end{pmatrix} \overset{rk_{13}=3}{\longrightarrow} LR(SIJ \cup \{(1,3)\}) \begin{pmatrix} 4 & 1 & 3 & 2 \\ . & 4 & . & 3 \\ . & 3 & 2 & 1 \\ 2 & . & 1 & . \end{pmatrix}$$

(3) liefert die folgenden Fälle für die in Frage kommenden Knotenpaare:

(a) $(1,1)$ und $(3,3)$: $\nexists$ Weg von $(3,3)$ nach $(1,1)$,
(b) $(1,1)$ und $(4,3)$: $\exists$ Weg von $(4,3)$ nach $(1,1)$,
(c) $(1,2)$ und $(3,3)$: $\nexists$ Weg von $(1,2)$ nach $(3,3)$,
(d) $(1,2)$ und $(4,3)$: $\nexists$ Weg von $(1,2)$ nach $(4,3)$ und umgekehrt.

Beispiel zur Einfügung (3)

Damit lassen sich 4 neue Pläne erzeugen, als Beispiel sei der Fall (d) beschrieben. Da die beiden Operationen $(1,2)$ und $(4,3)$ den gleichen Rang haben, kann kein Weg zwischen ihnen in $G(LR(SIJ))$ existieren, so dass die Fälle (3-a) und (3-b) zur Anwendung kommen:

$$LR(SIJ) = \begin{pmatrix} 3^* & \underline{1} & \underline{2} & 2^* \\ . & 4^* & . & 3^* \\ . & 3^* & 2^* & 1 \\ 2^* & . & \underline{1} & . \end{pmatrix} \stackrel{(d)}{\Longrightarrow} LR(SIJ \cup \{(1,3)\}) = \begin{pmatrix} 5 & \underline{1} & \underline{2} & 3 \\ . & 6 & . & 4 \\ . & 5 & 4 & 1 \\ 4 & . & \underline{3} & . \end{pmatrix}$$

$$LR(SIJ) = \begin{pmatrix} 3^* & \underline{1} & \underline{2} & 2^* \\ . & 4^* & . & 3^* \\ . & 3^* & 2^* & 1 \\ 2 & . & \underline{1} & . \end{pmatrix} \stackrel{(d)}{\Longrightarrow} LR(SIJ \cup \{(1,3)\}) = \begin{pmatrix} 5 & \underline{3} & \underline{2} & 4 \\ . & 6 & . & 5 \\ . & 4 & 3 & 1 \\ 2 & . & \underline{1} & . \end{pmatrix}$$

Enumeration der Pläne

Enumeration aller Pläne $\mathbf{LR[n, m, r]}$ mit $\mathbf{\max\{n, m\} \leq r \leq nm}$ über $\mathbf{SIJ = I \times J}$

Eingabe: n, m, Einfügereihenfolge der Operationen: $(i_1, j_1), (i_2, j_2), \ldots, (i_{nm}, j_{nm})$;
Ausgabe: Menge aller Pläne $MLR(SIJ)$ über $SIJ = I \times J$.
BEGIN
$k := 0$; $SIJ := \emptyset$; $MLR(SIJ) := \emptyset$;
WHILE $k \leq nm$ **DO**
BEGIN
$k := k + 1$;
Füge die Operation (i_k, j_k) in jeden Plan aus $MLR(SIJ)$ nach den Fällen $(1) - (3)$ der Einfügung ein und bestimme $MLR(SIJ \cup \{(i_k, j_k)\})$;
$SIJ := SIJ \cup \{(i_k, j_k)\}$;
END;
END.

Enumeration aller Pläne für Shop-Probleme

Satz 1. *Der Enumerationsalgorithmus (Insertion-Algorithmus) erzeugt alle Pläne eines Open-Shop Problems mit* $SIJ = I \times J$.

Folgerung: Der Algorithmus ist unmittelbar zur Enumeration aller Pläne mit $SIJ \subset I \times J$ anwendbar.

Bemerkung: Die Insertion-Idee kann auch für Shop-Probleme mit beliebigen Vorrangbedingungen für die Operationen modifiziert werden: Man beachte beim Einfügen jeder weiteren Operation zusätzlich die gegebenen Vorrangbedingungen.

Modifikation für Job-Shop und Flow-Shop Probleme:
Man beachte beim Einfügen jeder weiteren Operation die gegebenen technologischen Reihenfolgen für den in der Operation enthaltenen Auftrag.

Branch & Bound Algorithmen: Voraussetzungen

Branch & Bound Algorithmen sind *implizite Enumerationsverfahren*: Man sucht auf intelligente Weise in der Menge aller zulässigen Lösungen nach einer optimalen Lösung eines Problems der folgenden Form:

$$\text{Bestimme } x_0 \text{ mit } f(x_0) = \min\{f(x) \mid x \in S,\ S \text{ endlich }\}.$$

Voraussetzungen:

- **Branching-Regel:** Jedes $S^* \subseteq S$ muss in Teilmengen $S_1^*, S_2^*, \ldots S_r^*$ zerlegbar sein, wobei $\cup_{k=1}^{r} S_k^* = S^*$ und $|S_k^*| < |S^*|$ für alle k erfüllt sein muss.

- **Lower Bound:** Für jedes Teilproblem lässt sich eine untere Schranke für den optimalen Zielfunktionswert des Teilproblems angeben.

$$LB(S^*) \leq \min\{f(x) \mid x \in S^*\}$$

- **Auswahlregel:** Sie legt fest, welches Teilproblem im nächsten Schritt zerlegt werden müssen. Häufig ist es das Teilproblem mit der kleinsten unteren Schranke.

Branch & Bound Algorithmen - Beschreibung

Beachte: Die Mengen S_k werden nicht als disjunkt vorausgesetzt (häufig ist dies aber der Fall).

Hinweis: In der Regel wird gefordert, dass mit $|S^*| = 1$ die Bedingung $LB(B^*) = f(x)$ mit $x \in S^*$ gelten soll (dies ist aber zur Anwendung der Methode nicht notwendig).

Methode: Teile den Lösungsbereich in Teilmengen und bestimme für jedes so entstehende Teilproblem eine untere Schranke für den besten Zielfunktionswert in diesem Teilproblem. Wiederhole dies für den Lösungsbereich des durch die Auswahlfunktion ausgewählten Teilproblems. Aus der Zerlegung entsteht ein sogenannter **Branching Tree** mit Wurzel in S. Die Teilmengen, in die $S^* \subseteq S$ zerlegt wird, heißen **Kinder (children)** von S. Damit symbolisiert jeder Knoten ein Teilproblem. Man hat eine optimale Lösung gefunden, wenn man zu einer einelementigen Teilmenge gelangt ist, wobei der Zielfunktionswert des Elements dieser Menge kleiner oder gleich den unteren Schranken aller nichtverzweigten Knoten des Baumes ist.

Verkürzung des Verfahrens: Falls ein $\bar{x} \in S$ mit $f(\bar{x})$ bekannt ist, eliminiere alle S^* mit $LB(S^*) \geq f(\bar{x}) = UB$ (*upper bound*). S^* kann keine bessere Lösung enthalten.

Branch & Bound Algorithmen

Algorithmus Branch & Bound

Eingabe: Problem: $\min\{f(x) \mid x \in S, S \neq \emptyset, S \text{ endlich}\}$, Näherungslösung $\overline{x}$;
Ausgabe: Optimale Lösung: $BEST$ mit Zielfunktionswert UB (Upper Bound).
BEGIN $LIST := \{S\}$ (Menge der Teilprobleme); $BEST := \overline{x}$; $UB := f(\overline{x})$;
  WHILE $LIST \neq \emptyset$ **DO**
    BEGIN Wähle ein Teilproblem S^* aus $LIST$ aus; Entferne S^* aus $LIST$;
      Generiere die Kinder aus S^*: $CHILD(i),\ i = 1, \ldots, n_{S^*}$;
      Berechne für alle Kinder untere Schranken LB_i;
      FOR $i := 1$ **TO** n_{S^*} **DO**
        IF $LB_i < UB$ **THEN**
          IF $|CHILD(i)| = 1$ **THEN**
            BEGIN $UB := LB_i$; $BEST := CHILD(i)$; **END**;
            ELSE Füge $CHILD(i)$ in $LIST$ ein;
    END;
END.

Branch & Bound Algorithmus für das Problem $O \,||\, C_{max}$

- Ordne die Operationen nach nichtsteigenden Bearbeitungszeiten.
- Benutze den Insertion-Algorithmus als Branching Rule.
- Benutze als LB den *Makespan* des Schedules zum aktuellen partiellen Plan oder:
- Benutze als LB das *Gewicht eines schwersten Weges*, der die neu eingefügte Operation enthält.
- Verzweige den partiellen Plan mit kleinster unteren Schranke.

Beispiel 2. Gesucht ist die optimale Lösung eines $O \,||\, C_{max}$ Problems mit der folgenden Bearbeitungsmatrix sowie dem folgenden partiellen Startplan, der nur triviale Wege enthält. Es gilt: $C_{max} \geq 35$.

$$P = \begin{pmatrix} 10 & 12 & \underline{8} & \underline{5} \\ \underline{4} & \underline{6} & 15 & \underline{7} \\ . & 9 & 10 & 11 \\ 14 & 27 & 33 & 23 \end{pmatrix} \begin{matrix} 35 \\ 32 \\ 30 \end{matrix}, \quad LR(SIJ) = \begin{pmatrix} 2 & 1 & . & . \\ . & . & 2 & . \\ . & 3 & 1 & 2 \end{pmatrix}.$$

Insertion-Reihenfolge der Operationen:
$(1,3), (2,4), (2,2), (1,4), (2,1)$, da $p_{13} \geq p_{24} \geq \ldots \geq p_{21}$.

Branch & Bound Algorithmus - Beispiel

Eine optimale Lösung mit $C_{max} = 35$ ergibt sich wie folgt:

$$LR = \begin{pmatrix} 3 & 1 & 2 & 4 \\ 1 & 3 & 4 & 2 \\ . & 4 & 1 & 3 \end{pmatrix}, \qquad C = \begin{pmatrix} 30 & 12 & 20 & 35 \\ 4 & 18 & 35 & 11 \\ . & 31 & 10 & 22 \end{pmatrix}.$$

Bemerkung: In den folgenden partiellen Plan $LR(SIJ)$ soll die Operation $(1, 4)$ mit dem Rang 2 eingefügt werden:

$$LR(SIJ) = \begin{pmatrix} 3 & 1 & 2 & . \\ . & 2 & 3 & 1 \\ . & 3 & 1 & 2 \end{pmatrix} \rightarrow LR(SIJ \cup \{(1,4)\}) = \begin{pmatrix} 4 & 1 & 3 & 2 \\ . & 2 & 4 & 1 \\ . & 4 & 1 & 3 \end{pmatrix}.$$

Branch & Bound Algorithmen - Beispiel

$$H+P+T = \begin{pmatrix} 20 & 0 & 12 & . \\ . & 12 & 20 & 0 \\ . & 21 & 0 & 10 \end{pmatrix} + \begin{pmatrix} 10 & 12 & 8 & . \\ . & 6 & 15 & 7 \\ . & 9 & 10 & 11 \end{pmatrix} + \begin{pmatrix} 0 & 23 & 15 & . \\ . & 15 & 0 & 21 \\ . & 0 & 23 & 9 \end{pmatrix}.$$

Mit bekannten Matrizen H und T über SIJ lässt sich sehr einfach das Gewicht eines schwersten Weges über die Operation $(1, 4)$ mit Rang 2 bestimmen:

$$\begin{aligned} w_{14} &= \max\{p_{12} + h_{12}, p_{24} + h_{24}\} + p_{14} + \max\{p_{13} + t_{13}, p_{34} + t_{34}\} \\ &= 12 + 5 + 23 = 40 \end{aligned}$$

Damit ist der längste Weg in $O(1)$ berechenbar, falls die Matrizen H und T abgespeichert sind.

Dynamische Optimierung

Voraussetzung: Das Problem ist ein n-*stufiges Entscheidungsproblem* und die Zielfunktion setzt sich *additiv* (oder *multiplikativ* bzw. als *Bottleneck-Funktion*) aus den Stufen zusammen! (**Separabilität der Zielfunktion**)

Die dynamische Optimierung beruht auf dem **Bellmanschen Optimalitätsprinzip** und der rekursiven Auswertung der **Bellmanschen Funktionalgleichungen**. In einem ersten Durchlauf durch die Stufen wird der *optimale Zielfunktionswert* ermittelt, und in einem zweiten Durchlauf in entgegengesetzter Richtung eine *optimale Lösung*.

Die Methodik wird am Beispiel der Lösung des Problems $1 \mid\mid \sum f_i(C_i)$ erklärt.

Bezeichnung:

$$V(I) = \sum_{i \in I} f_i(C_i).$$

Dynamische Optimierung: Vorwärts-/Rückwärtsstrategie (1)

Anfangswerte: $V(\{i\}) = f_i(p_i)$ für alle $i \in I$;

Rekursion: $$V(I) = \min_{i \in I} \left\{ V(I \backslash \{i\}) + f_i \left(\sum_{k \in I} p_k \right) \right\};$$

Optimaler Zielfunktionswert: $V(\{1, \ldots, n\})$.

Vorwärtsrechnung: Die Anfangswerte führen nach $(n-1)$-maliger Rekursion zum optimalen Zielfunktionswert $V(I)$.

Rückwärtsrechnung: Der optimale Zielfunktionswert ermöglicht rückwärts die Bestimmung des Auftrags an Position k, $k = n, \ldots, 1$.

Dynamische Optimierung (2)

Dynamische Optimierung: Rückwärts-/Vorwärtsstrategie (2)

Anfangswerte: $V(\{1,\ldots,i-1,i+1,\ldots,n\}) = f_i(C_{max})$;

Rekursion: $$V(I) = \min_{i\in\{1,\ldots,n\}} \left\{ V(I\cup\{i\}) + f_i(\sum_{k\in I\cup\{i\}} p_k) \right\};$$

Optimaler Zielfunktionswert: $V(\emptyset)$.

Rückwärtsrechnung: Ausgehend von den Anfangswerten wird durch schrittweise Rekursion rückwärts $V(\emptyset)$ erreicht.

Vorwärtsrechnung: Der optimale Zielfunktionswert $V(\emptyset)$ ermöglicht vorwärts die Bestimmung des Auftrags an Position k, $k = 1,\ldots,n$.

Aufwand: $O(2^n)$

Bemerkung: Die dynamische Optimierung kann einen *exponentiellen, polynomialen oder pseudo-polynomialen* Aufwand haben!

Dynamische Optimierung - Beispiel (1)

Beispiel 3. *Gegeben sind die folgenden Eingangsdaten für ein* $1 \mid\mid \sum f_i(C_i)$ *Problem:*

i	1	2	3
p_i	4	5	7
$f_i(C_i)$	$10 \cdot C_1$	C_2^2	$C_3 + 20$

Vorwärts-/Rückwärtsstrategie

- **Vorwärtsrechnung**:

  Anfangswerte:

  (Stufe 1) $V(\{1\}) = f_1(C_1) = 40$

  $V(\{2\}) = f_2(C_2) = 25$

  $V(\{3\}) = f_3(C_3) = 27$

Dynamische Optimierung - Beispiel (2)

Rekursion:

$$\begin{aligned}
\textit{(Stufe 2)}\ V(\{1,2\}) &= \min\{V(\{1\}) + f_2(p_1+p_2), \underline{V(\{2\}) + f_1(p_1+p_2)}\} \\
&= \min\{40+81, \underline{25+90}\} = 115 \\
V(\{1,3\}) &= \min\{\underline{V(\{1\}) + f_3(p_1+p_3)}, V(\{3\}) + f_1(p_1+p_3)\} \\
&= \min\{\underline{40+31}, 27+110\} = 71 \\
V(\{2,3\}) &= \min\{\underline{V(\{2\}) + f_3(p_2+p_3)}, V(\{3\}) + f_2(p_2+p_3)\} \\
&= \min\{\underline{25+32}, 27+144\} = 57 \\
\textit{(Stufe 3)}\ V(\{1,2,3\}) &= \min\{\underline{V(\{1,2\}) + f_3(16)}, V(\{1,3\}) + f_2(16), \\
&\qquad \underline{V(\{2,3\}) + f_1(16)}\} \\
&= \min\{\underline{115+36}, 71+256, 57+160\} = 151
\end{aligned}$$

Optimaler Zielfunktionswert: $V(\{1,2,3\}) = 151$

- **Rückwärtsrechnung**:

$$\begin{aligned}
V(\{1,2,3\}) &= 151, && \text{wenn } V(\{1,2\}) \text{ erreicht wird, d. h. } \pi(3) = 3; \\
V(\{1,2\}) &= 115, && \text{wenn } V(\{2\}) \text{ erreicht wird, d. h. } \pi(2) = 1; \\
V(\{2\}) &= 25, && \pi(1) = 2, \text{ d.h. die optimale Reihenfolge ist } \pi = (2,1,3).
\end{aligned}$$

Dynamische Optimierung - Beispiel (3)

Beispiel 4. Gegeben sind die folgenden Eingangsdaten für ein $1 \mid\mid \sum f_i(C_i)$ Problem:

i	1	2	3
p_i	4	5	7
$f_i(C_i)$	$10 \cdot C_1$	C_2^2	$C_3 + 20$

Rückwärts-/Vorwärtsstrategie

Mit $C_{max} = 16$ ergibt sich:

- **Rückwärtsrechnung**:
  Anfangswerte:
  (Stufe 3) $V(\{1,2\}) = f_3(C_{max}) = f_3(16) = 36$
  $V(\{1,3\}) = f_2(C_{max}) = f_2(16) = 256$
  $V(\{2,3\}) = f_1(C_{max}) = f_1(16) = 160$

Dynamische Optimierung - Beispiel (4)

Rekursion:

$$
\begin{aligned}
\text{(Stufe 2)}\quad V(\{1\}) &= \min\{V(\{1,2\}) + f_2(9), V(\{1,3\}) + f_3(11)\} \\
&= \min\{\underline{36+81}, 256+31\} = 117 \\
V(\{2\}) &= \min\{V(\{1,2\}) + f_1(9), V(\{2,3\}) + f_3(12)\} \\
&= \min\{\underline{36+90}, 160+32\} = 126 \\
V(\{3\}) &= \min\{V(\{1,3\}) + f_1(11), V(\{2,3\}) + f_2(12)\} \\
&= \min\{256+110, \underline{160+144}\} = 304 \\
\text{(Stufe 1)}\quad V(\emptyset) &= \min\{V(\{1\}) + f_1(4), V(\{2\}) + f_2(5), V(\{3\}) + f_3(7)\} \\
&= \min\{117+40, \underline{126+25}, 304+27\} = 151
\end{aligned}
$$

Optimaler Zielfunktionswert: $V(\emptyset) = 151$.

- **Vorwärtsrechnung**:

$$
\begin{aligned}
V(\emptyset) &= 151 \quad \text{bei } V(\{2\}) + f_2(5), \text{ d. h. } \pi(1) = 2; \\
V(\{2\}) &= 126 \quad \text{bei } V(\{1,2\}) + f_1(9), \text{ d. h. } \pi(2) = 1; \\
V(\{1,2\}) &= 36 \quad \text{bei } f_3(16), \text{ d. h. } \pi(3) = 3.
\end{aligned}
$$

Damit erhält man ebenfalls die optimale Reihenfolge $\pi = (2, 1, 3)$.

Dynamische Optimierung für das Problem $1 \mid\mid \Sigma T_i$ (1)

Bemerkung: Durch Ausnutzung spezieller Eigenschaften eines gegebenen Problems aus $\mathbb{NP}$-hard gelingt es häufig, den dynamischen Optimierungsalgorithmus zu einem pseudopolynomialen Algorithmus zu entwickeln.

Lemma 1. (Emmons) *Gelten für eine Instanz des Problems* $1 \mid\mid \sum T_i$ *für zwei Aufträge* A_i *und* A_k *die Ungleichungen* $p_i \leq p_k$ *und* $d_i \leq d_k$*, dann existiert eine optimale Reihenfolge, in der* A_i *vor* A_k *bearbeitet wird.*

Als nächstes wird untersucht, wie sich eine kleine Änderung der Eingangsdaten auf die Optimalität der Lösung auswirkt. Es seien die Eingangsdaten für P_1 und P_2 gegeben durch:

P_1 :

i	1	...	n
p_i	p_1	...	p_n
d_i	d_1	...	d_n

P_2 :

i	1	...	$k-1$	k	$k+1$	...	n
p_i	p_1	...	p_{k-1}	p_k	p_{k+1}	...	p_n
d_i	d_1	...	d_{k-1}	$\max\{C'_k, d_k\}$	d_{k+1}	...	d_n

Lemma 2. *Jede für* P_2 *optimale Reihenfolge ist auch optimal für* P_1*.*

Dynamische Optimierung für das Problem $1 \mid\mid \Sigma T_i$ (2)

Lemma 3. *Für das Problem $1 \mid\mid \sum T_i$ seien die Due Dates nach der EDD-Regel sortiert, und es gelte $p_k = \max\{p_i \mid i \in I\}$. Dann gibt es ein δ mit $0 \leq \delta \leq n - k$ so, dass eine optimale Reihenfolge π existiert, in der alle Aufträge A_i mit $i \leq k + \delta$ vor Auftrag A_k und alle restlichen Aufträge A_i mit $i > k + \delta$ nach Auftrag A_k bearbeitet werden.*

Folgerung: Man hat nicht mehr alle Teilmengen zu untersuchen!
Es existiert eine optimale Reihenfolge der Indizes der Aufträge mit den folgenden Eigenschaften:

$$\{\underbrace{1, 2, \ldots, k-1, k+1, \ldots, k+\delta,}_{\text{beliebige Permutation}} \underbrace{k,}_{\substack{Position \\ ist\ fest}} \underbrace{k+\delta+1, \ldots, k+n-k}_{\text{beliebige Permutation}}\} \quad 0 \leq \delta \leq n-k$$

Also: Man hat eine optimale Reihenfolge für eine Indexmenge $\{1, \ldots, l\}$ von Aufträgen zu bestimmen, wobei die Bearbeitung zum Zeitpunkt t beginnen soll.

Dynamische Optimierung für das Problem $1 \mid\mid \Sigma T_i$ (3)

Für eine Menge mit l Aufträgen sei k' festgelegt durch $p_{k'} = \max\{p_i \mid i \in \{1, \ldots, l\}\}$. Dann gibt es für ein δ mit $0 \leq \delta \leq l - k'$ eine optimale Indexreihenfolge der Form:

$$\{\underbrace{1, \ldots, k'-1, k'+1, \ldots, k'+\delta}_{\text{permutierbar}}, \underbrace{k'}_{\text{fest}}, \underbrace{k'+\delta+1, \ldots, l}_{\text{permutierbar}}\}$$

Da die Position von k fest ist, gilt: $C_{k'}(\delta) = t + \sum_{i \leq k'+\delta} p_i$.

Für die dynamische Optimierung werden folgende Bezeichnungen verwendet:

- $I(i, l, k) = \{i, i+1, \ldots, l\} \backslash \{k\}$,
- $V(I(i, l, k), t)$ ist die Gesamtverspätung für die Aufträge aus der Indexmenge I, wenn ihre Bearbeitung zum Zeitpunkt t beginnt.

Dynamische Optimierung für das Problem $1 \mid\mid \Sigma T_i$ (4)

Dynamische Optimierung für das Problem $1 \mid\mid \Sigma T_i$

Eingabe: Instanz für das Problem $\mathbf{1 \mid\mid \sum T_i}$
Ausgabe: Optimaler Zielfunktionswert

Anfangswerte:
$$V(\emptyset, t) = 0,$$
$$V(\{i\}, t) = \max\{0, t + p_i - d_i\};$$

Rekursion:
$$V(I(i,l,k),t) = \min_{0 \le \delta \le l-k'} \{V(I(i, k'+\delta, k'), t) +$$
$$+ \max\{0, C_{k'}(\delta) - d_{k'}\} + V(I(k'+\delta+1, l, k'), C_{k'}(\delta))\},$$
k' ist durch $p_{k'} = \max\{p_{i'} \mid i' \in I(i,l,k)\}$ bestimmt;

Optimaler Zielfunktionswert:
$$V(\{1, \ldots, n\}, 0).$$

Aufwand: Es sind höchstens $O(n^3 \sum p_i)$ Rekursionen durchzuführen, jede erfordert dabei einen Aufwand von $O(n)$. Damit ergibt sich eine Gesamtkomplexität von $O(n^4 \sum p_i)$, und der Algorithmus ist pseudopolynomial.