Übungsaufgabe

Scheduling-Algorithmen: Sequenzierung, Heuristiken und Optimierungsmetriken

Universität:	Technische Universität Berlin
Kurs/Modul:	Algorithmen und Datenstrukturen
Erstellungsdatum:	September 6, 2025

Aufgabe 1: Sequenzierung auf einer Maschine

In dieser Aufgabe geht es um die zeitliche Abfolge von Jobs auf einer einzigen Maschine. Sei $J = \{1, \ldots, n\}$ die Menge der Jobs, $p_j > 0$ die Bearbeitungszeit von Job j. Sei S_j der Startzeitpunkt und $C_j = S_j + p_j$ der Abschluss von Job j. Eine zulässige Sequenz entspricht einer Permutation π von J. Falls keine Idle-Zeit vorgesehen ist, gilt für die in der Sequenz π belegten Jobs:

$$S_{\pi_i} = \sum_{k=1}^{i-1} p_{\pi_k}, \qquad C_{\pi_i} = S_{\pi_i} + p_{\pi_i}, \qquad i = 1, \ldots, n.$$

Zielgrößen:

$$C_{\max} = \max_{j \in J} C_j \quad \text{minimiere.}$$

a) Formuliere das Scheduling-Problem als eine Optimierungsaufgabe mit Ziel $\min C_{\max}$ auf einer Maschine, inklusive der Variablen S_j und C_j sowie der Restriktion, dass die Jobs sequentiell ablaufen (ohne Unterbrechung) und ggf. Idle-Zeiten explizit zulässig sind.

b) Gegeben seien $n = 4$ Jobs mit Bearbeitungszeiten

$$p_1 = 3, \quad p_2 = 1, \quad p_3 = 2, \quad p_4 = 4,$$

und die Sequenz $\pi = (1, 2, 3, 4)$. Berechne anhand der obigen Formeln die Startzeiten S_j, die Abschlusszeiten C_j und $C_{\max}$ für diese Sequenz. (Nutze die Gleichungen aus Teil a).)

c) Diskutiere kurz, wie sich die eingeführte Idle-Zeit auf $C_{\max}$ auswirken würde. Unter welchen Bedingungen könnte Idle-Time sinnvoll sein, wenn das Ziel $C_{\max}$ ist?

Aufgabe 2: Heuristiken zur Sequenzierung mehrerer Jobs

Für eine Menge von Jobs mit Bearbeitungszeiten p_j und multizieligen Terminen werden gängige Heuristiken verwendet, um schnell eine gute Sequenz zu finden. Betrachte folgenden Datensatz:

$$J = \{1, 2, 3, 4, 5\}, \quad p = (3, 5, 2, 4, 6), \quad d = (7, 9, 5, 10, 12).$$

a) Wende die EDD-Heuristik (Earliest Due Date) an und erzeuge eine Sequenz π^{EDD} durch Sortieren der Jobs nach aufsteigendem DAtuendatum d_j. Gib die resultierende Reihenfolge an.

b) Wende die SPT-Heuristik (Shortest Processing Time) an und erzeuge eine Sequenz π^{SPT} durch Sortieren der Jobs nach absteigender oder absteigender Bearbeitungszeit? (Wähle die übliche SPT-Variante: aufsteigend nach p_j.) Gib die resultierende Reihenfolge an.

c) Für beide Sequenzen (π^{EDD} und π^{SPT}) definiere die tardiness- oder lateness-Kennzahl, die du verwenden möchtest (z. B. maximale Abweichung von der Frist), und erkläre, wie du diese Kennzahl berechnest:

$$L_j = \max(C_j - d_j,\, 0), \quad T_{\max} = \max_j L_j \quad \text{bzw.} \quad \text{falls } C_j < d_j, \text{ dann } L_j = 0.$$

Berechne ohne Lösung anzugeben die entsprechenden Werte für die beiden Sequenzen.

d) Diskutiere kurz, welcher Heuristik für diese Daten tendenziell bessere Ergebnisse liefern könnte und warum.

Aufgabe 3: Optimierungsmetriken und Grundlagen von Branch-and-Bound

Branch-and-Bound (BB) ist eine vollständige Suchstrategie, die oft zur Lösung von Scheduling-Problemen in zulässiger Laufzeit verwendet wird. In dieser Aufgabe arbeiten wir konzeptionell mit einem kleinen Beispiel.

a) Beschreibe in eigenen Worten, wie Branch-and-Bound für Scheduling-Probleme funktionieren kann. Welche Elemente gehören typischerweise zum Baum, wie werden Zweige gebildet, und welche Rolle spielen obere und untere Schranken?

b) Formuliere eine einfache untere Schranke (Lower Bound) für das Problem der Sequenzierung zur Minimierung von $\sum w_j C_j$ auf einer einzelnen Maschine. Verwende dabei eine Teilfolge S (eine bereits feste Reihenfolge der ersten Jobs) und die verbleibenden Jobs R. Definiere klar, wie der Bound berechnet wird (ohne konkrete Zahlen).

c) Diskutiere, welche Vorteile Branch-and-Bound gegenüber reinen Heuristiken bietet und in welchen Situationen BB besonders sinnvoll ist (z. B. kleine bis mittlere Problemgrößen, exakte Ergebnisse gewünscht).

Lösungen

Aufgabe 1: Sequenzierung auf einer Maschine

In dieser Aufgabe geht es um die zeitliche Abfolge von Jobs auf einer einzigen Maschine. Sei $J = \{1, \dots, n\}$ die Menge der Jobs, $p_j > 0$ die Bearbeitungszeit von Job j. Sei S_j der Startzeitpunkt und $C_j = S_j + p_j$ der Abschluss von Job j. Eine zulässige Sequenz entspricht einer Permutation π von J. Falls keine Idle-Zeit vorgesehen ist, gilt für die in der Sequenz π belegten Jobs:

$$S_{\pi_i} = \sum_{k=1}^{i-1} p_{\pi_k}, \qquad C_{\pi_i} = S_{\pi_i} + p_{\pi_i}, \qquad i = 1, \dots, n.$$

Zielgrößen:

$$C_{\max} = \max_{j \in J} C_j \quad \text{minimiere.}$$

a) Formuliere das Scheduling-Problem als eine Optimierungsaufgabe mit Ziel $\min C_{\max}$ auf einer Maschine, inklusive der Variablen S_j und C_j sowie der Restriktion, dass die Jobs sequentiell ablaufen (ohne Unterbrechung) und ggf. Idle-Zeiten explizit zulässig sind.

Lösung (Modellierung als MILP mit Vorwärts-Sequenzierung). Sei $J = \{1, \dots, n\}$. Es seien folgende Entscheidungsvariablen zu verwenden: - $S_j \geq 0$ Startzeit von Job j, - $C_j \geq 0$ Abschlusszeit von Job j mit $C_j = S_j + p_j$, - $z_{ij} \in \{0, 1\}$ für alle $i \neq j$: $z_{ij} = 1$ bedeutet, dass Job i vor Job j beendet ist (d. h. i precedes j), - $u_j \in \{1, \dots, n\}$ MTZ-Variablen (Sequenzpositionen der Jobs), - $C_{\max} \geq 0$ als Variable der Zielfunktion.

Zielelement:

$$\min C_{\max}$$

Beschränkungen:

$$C_j \geq S_j + p_j \qquad \forall j \in J, \tag{1}$$

$$C_{\max} \geq C_j \qquad \forall j \in J, \tag{2}$$

$$S_j \geq \sum_{i \in J \setminus \{j\}} p_i \, z_{ij} \qquad \forall j \in J, \tag{3}$$

$$z_{ij} + z_{ji} = 1 \qquad \forall i \neq j, \tag{4}$$

$$z_{ij} \in \{0, 1\} \qquad \forall i \neq j, \tag{5}$$

$$S_j \geq 0, \; C_j \geq 0, \tag{6}$$

$$1 \leq u_j \leq n \qquad \forall j \in J, \tag{7}$$

$$u_i - u_j + n \, z_{ij} \leq n - 1 \qquad \forall i \neq j. \tag{8}$$

Hinweis: - Die Gleichung $S_j \geq \sum_{i \neq j} p_i z_{ij}$ leitet die Startzeit von der Gesamtdauer derjenigen Jobs ab, welche vor j liegen. - Die MTZ-Bedingungen (mit u_j) verhindern Zyklen und erzwingen eine zulässige Sequenz. - Idle-Zeiten können explizit durch zulässige Startzeiten realisiert werden, aber sie sind durch das Minimieren von $C_{\max}$ tendenziell zu vermeiden, sofern keine Release-Dates vorliegen.

b) Gegeben seien $n = 4$ Jobs mit Bearbeitungszeiten

$$p_1 = 3, \quad p_2 = 1, \quad p_3 = 2, \quad p_4 = 4,$$

und die Sequenz $\pi = (1, 2, 3, 4)$. Berechne anhand der obigen Formeln die Startzeiten S_j, die Abschlusszeiten C_j und $C_{\max}$ für diese Sequenz. (Nutze die Gleichungen aus Teil a).)

Berechnung: - Job 1 (1) beginnt bei 0: $S_1 = 0$, $C_1 = S_1 + p_1 = 0 + 3 = 3$. - Job 2 (2) beginnt nach Abschluss von Job 1: $S_2 = 3$, $C_2 = 3 + 1 = 4$. - Job 3 (3) beginnt nach Job 2: $S_3 = 4$, $C_3 = 4 + 2 = 6$. - Job 4 (4) beginnt nach Job 3: $S_4 = 6$, $C_4 = 6 + 4 = 10$. - $C_{\max} = \max\{3, 4, 6, 10\} = 10$.

Ergebnis:

$$S_1 = 0,\ C_1 = 3; \quad S_2 = 3,\ C_2 = 4; \quad S_3 = 4,\ C_3 = 6; \quad S_4 = 6,\ C_4 = 10; \quad C_{\max} = 10.$$

c) Diskutiere kurz, wie sich die eingeführte Idle-Zeit auf $C_{\max}$ auswirken würde. Unter welchen Bedingungen könnte Idle-Time sinnvoll sein, wenn das Ziel $C_{\max}$ ist?

Lösungshinweis: - Ohne Release-Dates (alle Aufträge ab Zeitpunkt 0 verfügbar) erhöht Idle-Time typischerweise den Gesamtdurchlauf (Makespan) und kann $C_{\max}$ nicht senken. Jede durch Idle verzögerte Nachfolge führt zu größeren Abschlusszeiten und damit zu einer größeren oder gleichen $C_{\max}$. - Idle-Time kann jedoch notwendig oder sinnvoll sein, falls Release-Dates oder andere Restriktionen existieren, z. B. um eine als Vorbedingung eingeführte Aufgabe erst ab einem bestimmten Zeitpunkt starten zu lassen. In solchen Fällen ist Idle-Time durch die Restriktionen vorgegeben und senkt in der Regel nicht das maximale Abschlussdatum, verhindert aber, dass frühere Aufgaben durch zu frühes Starten blockiert werden. - Insgesamt gilt: Für das einfache 1-Maschinen-Scheduling mit allen Jobs gleichzeitig verfügbar, ist Idle-Time kein Mittel zur Verringerung von $C_{\max}$.

Aufgabe 2: Heuristiken zur Sequenzierung mehrerer Jobs

Für eine Menge von Jobs mit Bearbeitungszeiten p_j und multizieligen Terminen werden gängige Heuristiken verwendet, um schnell eine gute Sequenz zu finden. Betrachte folgenden Datensatz:

$$J = \{1, 2, 3, 4, 5\}, \quad p = (3, 5, 2, 4, 6), \quad d = (7, 9, 5, 10, 12).$$

a) Wende die EDD-Heuristik (Earliest Due Date) an und erzeuge eine Sequenz π^{EDD} durch Sortieren der Jobs nach aufsteigendem DAtuendatum d_j. Gib die resultierende Reihenfolge an.

Lösung: Sortierung nach d_j (aufsteigend): $d_3 = 5,\ d_1 = 7,\ d_2 = 9,\ d_4 = 10,\ d_5 = 12$. Daraus folgt:

$$\pi^{\text{EDD}} = (3,\ 1,\ 2,\ 4,\ 5).$$

b) Wende die SPT-Heuristik (Shortest Processing Time) an und erzeuge eine Sequenz π^{SPT} durch Sortieren der Jobs nach absteigender oder absteigender Bearbeitungszeit? (Wähle die übliche SPT-Variante: aufsteigend nach p_j.) Gib die resultierende Reihenfolge an.

Lösung: Bearbeitungszeiten sortiert aufsteigend: $p_3 = 2,\ p_1 = 3,\ p_4 = 4,\ p_2 = 5,\ p_5 = 6$. Daraus folgt:

$$\pi^{\text{SPT}} = (3,\ 1,\ 4,\ 2,\ 5).$$

c) Für beide Sequenzen (π^{EDD} und π^{SPT}) definiere die tardiness- oder lateness-Kennzahl, die du verwenden möchtest (z. B. maximale Abweichung von der Frist), und erkläre, wie du diese Kennzahl berechnest:

$$L_j = \max(C_j - d_j,\ 0), \quad T_{\max} = \max_j L_j \quad \text{bzw.} \quad \text{falls } C_j < d_j, \text{ dann } L_j = 0.$$

Berechne ohne Lösung anzugeben die entsprechenden Werte für die beiden Sequenzen.

Lösung (Definition und Vorgehen): - Definiere Lateness-Latenz pro Job: $L_j = \max(C_j - d_j, 0)$. - Gesamte Kennzahl: $T_{\max} = \max_j L_j$ (maximale Verspätung). - Vorgehen: Berechne für jeden Job j nach der jeweiligen Sequenz C_j als kumulative Summe der Bearbeitungszeiten bis einschließlich j. Wende dann die obige Formel für L_j an.

Berechnungen (ohne vollständiges numerisches Ergebnis hier): - Für $\pi^{\text{EDD}} = (3, 1, 2, 4, 5)$ ergibt sich eine bestimmte Folge von C_j (durchlaufend: C_3, C_1, C_2, C_4, C_5); anschließend

$$L_3 = \max(C_3 - d_3, 0),\ L_1 = \max(C_1 - d_1, 0),\ \ldots,\ T_{\max} = \max\{L_j\}.$$

- Für $\pi^{\text{SPT}} = (3, 1, 4, 2, 5)$ similarly.

Hinweis zur Praxis: - Die konkrete numerische Auswertung der L_j hängt von den berechneten C_j ab, die aus der Sequenzordnung resultieren. Die obige Vorgehensweise liefert die erforderliche Berechnungsvorschrift.

d) Diskutiere kurz, welcher Heuristik für diese Daten tendenziell bessere Ergebnisse liefern könnte und warum.

Diskussion: - EDD neigt dazu, maximale Fristüberschreitungen zu minimieren, insbesondere wenn Fristen (Due Dates) stark variieren. In dieser Datenmenge kann EDD oft frühere Jobs mit engen Terminen rechtzeitig abarbeiten. - SPT minimiert typischerweise Ihre durchschnittliche Wartezeit bzw. Ihre Gesamtdurchlaufzeit und kann bei vielen Jobs mit unterschiedlich langen

Bearbeitungszeiten vorteilhaft sein, verschiebt aber möglicherweise spätere Aufgaben näher an deren Fristen. - In dieser konkreten Stichprobe kann EDD tendenziell bessere Lateness-Werte liefern, da einige Fristen früh liegen, während SPT häufiger längere Jobs später platziert. Dennoch kann je nach Verteilung der Bearbeitungszeiten und Fristen eine andere Reihenfolge besser abschneiden. Allgemein gilt: EDD ist oft eine gute Wahl, wenn das Ziel primär die Vermeidung von Fristüberschreitungen ist; SPT tendiert dazu, den Gesamtdurchlauf und die Wartezeiten zu minimieren.

Aufgabe 3: Optimierungsmetriken und Grundlagen von Branch-and-Bound

Branch-and-Bound (BB) ist eine vollständige Suchstrategie, die oft zur Lösung von Scheduling-Problemen in zulässiger Laufzeit verwendet wird. In dieser Aufgabe arbeiten wir konzeptionell mit einem kleinen Beispiel.

a) Beschreibe in eigenen Worten, wie Branch-and-Bound für Scheduling-Probleme funktionieren kann. Welche Elemente gehören typischerweise zum Baum, wie werden Zweige gebildet, und welche Rolle spielen obere und untere Schranken?

Lösung: - Baumstruktur: Jeder Knoten entspricht einer Teilreihenfolge (Präfix) bereits festgelegter Jobs; der Wurzelknoten entspricht einem leeren Präfix. Ein Kindknoten ergibt sich, indem ein weiterer noch nicht geplanter Job als nächster an das Präfix angehängt wird. - Zweige: Von einem Knoten entstehen Zweige durch das Auswählen eines der verbleibenden Jobs als nächsten in der Sequenz. - Obere Schranke (Upper Bound): Eine gute vollständige Lösung, z. B. durch eine schnelle Heuristik gefunden, dient als aktueller Obergrenze für das Ziel (z. B. $\min \sum w_j C_j$ oder $C_{\max}$). - Untere Schranke (Lower Bound): Eine rechnerisch belegbare Untergrenze des optimalen Werts ab dem aktuellen Knoten, die verwendet wird, um Zweige abzubrechen, die keine bessere Lösung liefern können. - Vorgehen: Expandiere Knoten nur dann, wenn der Lower Bound des Knotens kleiner ist als der aktuell gefundenen Obergrenze. Auf diese Weise werden viele unproduktive Teilbäume abgeschnitten.

b) Formuliere eine einfache untere Schranke (Lower Bound) für das Problem der Sequenzierung zur Minimierung von $\sum w_j C_j$ auf einer einzelnen Maschine. Verwende dabei eine Teilfolge S (eine bereits feste Reihenfolge der ersten Jobs) und die verbleibenden Jobs R. Definiere klar, wie der Bound berechnet wird (ohne konkrete Zahlen).

Lösung (Lower Bound via Smith-Regel (WSPT) als Fortführung der verbleibenden Jobs). - Sei $S = (j_1, \ldots, j_m)$ die bereits festgelegte Präfix-Reihenfolge und $R = J \setminus \{j_1, \ldots, j_m\}$ die verbleibenden Jobs. - Zeitverlauf: Nach dem Abschluss des Prefix S ist die aktuelle Zeit

$$t = \sum_{r=1}^{m} p_{j_r}.$$

- Um eine untere Schranke für die verbleibenden Jobs R zu erhalten, plane diese in der sogenannten WSPT-Reihenfolge (Smith-Regel) mit aufsteigender p_j/w_j–Quote, d. h. sortiere R nach p_j/w_j aufsteigend und bezeichne die Ordnung als $(r_1, \ldots, r_{|R|})$. - Die Kalibrierung der verbleibenden Jobs: Die Abschlusszeiten der verbleibenden Jobs in dieser idealen Fortsetzung wären

$$C_{r_k} \geq t + \sum_{\ell=1}^{k} p_{r_\ell}, \quad k = 1, \ldots, |R|.$$

- Die minimale zusätzliche gewichtete Summe, die mit dieser Fortsetzung erreichbar ist, ist daher

$$\mathrm{LB}_{\mathrm{Remaining}} = \sum_{k=1}^{|R|} w_{r_k} \left(t + \sum_{\ell=1}^{k} p_{r_\ell} \right).$$

- Die komplette untere Schranke für den aktuellen Knoten ist dann

$$\mathrm{LB}(S) = \sum_{j \in S} w_j C_j + \mathrm{LB}_{\mathrm{Remaining}},$$

wobei $\sum_{j \in S} w_j C_j$ der bislang erzielte Beitrag aus dem bereits festgelegten Prefix ist.

Begründung: - Die verbleibenden Jobs R können aufgrund der bekannten optimalen Reihenfolge (WSPT) die kleinere mögliche zusätzliche Kosten erzeugen; daher liefert diese Berechnung eine zulässige, admissible Lower Bound für den aktuellen Ast.

c) Diskutiere, welche Vorteile Branch-and-Bound gegenüber reinen Heuristiken bietet und in welchen Situationen BB besonders sinnvoll ist (z. B. kleine bis mittlere Problemgrößen, exakte Ergebnisse gewünscht).

Lösung: - Vorteile von BB: - Liefert exakte optimale Lösungen, nicht nur heuristische Näherungen. - Nutzt starke obere und untere Schranken, um große Teile des Suchraums wirksam abzuschneiden. - Kann problemstrukturspezifische Pruning-Regeln und problemnahe Bounds einsetzen, um die Praxis runtimes signifikant zu reduzieren. - Wann sinnvoll: - Bei kleinen bis mittleren Problemgrößen (z. B. $n \lesssim$ 15–25 je nach Struktur) oder wenn exakte Ergebnisse benötigt werden. - Wenn die Heuristiken zwar schnell gute Lösungen liefern, aber eine garantierte optimale Lösung gefordert ist. - In Fällen mit engen Fristen oder when signifikante Einsparungen beim Lösen erforderlich sind (z. B. in Planungssystemen mit hoher Zuverlässigkeit).