

Christian-Albrechts-Universität zu Kiel
Institut für Informatik, Arbeitsgruppe Algorithmen und Komplexität
Prof. Dr. K. Jansen

Musterlösung – Probeklausur 2
Analyse von Algorithmen und Komplexität
SS 2026

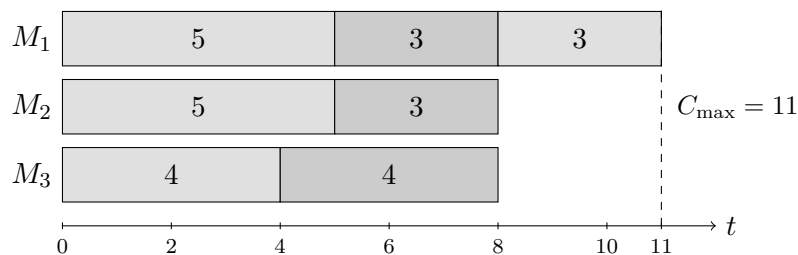
Aufgabe 1 ANWENDUNG – LPT

(5 + 2 + 3 Punkte)

a) LPT sortiert die Jobs absteigend und wendet darauf LISTSCHEDULING an; Lastvektor (M_1, M_2, M_3) , bei Gleichstand kleinster Index.

- Absteigend sortiert: 5, 5, 4, 4, 3, 3, 3.
- $5 \rightarrow M_1$: (5, 0, 0); $5 \rightarrow M_2$: (5, 5, 0); $4 \rightarrow M_3$: (5, 5, 4).
- $4 \rightarrow M_3$ (kleinste Last 4): (5, 5, 8).
- $3 \rightarrow M_1$ (kleinste Last 5, Gleichstand M_1, M_2): (8, 5, 8).
- $3 \rightarrow M_2$ (kleinste Last 5): (8, 8, 8).
- $3 \rightarrow M_1$ (kleinste Last 8, Gleichstand, kleinster Index): (11, 8, 8).

Belegung: $M_1 = \{5, 3, 3\}$ (Last 11), $M_2 = \{5, 3\}$ (Last 8), $M_3 = \{4, 4\}$ (Last 8). Makespan $LPT = C_{\max} = 11$.



Bemerkung: Wegen $\sum_j p_j = 27 = 3 \cdot 9$ und der Aufteilung $\{5, 4\}, \{5, 4\}, \{3, 3, 3\}$ (Lasten 9, 9, 9) ist $OPT = 9$. LPT ist hier also *nicht* optimal: die Rate ist $\frac{11}{9}$.

b) LPT hat die approximative Güte

$$\frac{4}{3} - \frac{1}{3m}.$$

Für $m = 3$ ergibt das $\frac{4}{3} - \frac{1}{9} = \frac{11}{9}$; die Instanz aus a) erreicht diese Schranke also exakt und ist ein worst-case-Beispiel.

c) Instanz mit $m = 2$ und Jobs in der Reihenfolge $p = (1, 1, 2)$. LISTSCHEDULING legt jeden Job auf die aktuell leerste Maschine (Lastvektor (M_1, M_2)):

- $1 \rightarrow M_1$: (1, 0); $1 \rightarrow M_2$: (1, 1); $2 \rightarrow M_1$ (Gleichstand, kleinster Index): (3, 1).

LS = 3. Optimal ist $\{1, 1\}$ auf einer Maschine und $\{2\}$ auf der anderen (Lasten 2, 2), also $OPT = 2$. Die Rate ist $\frac{LS}{OPT} = \frac{3}{2} = 2 - \frac{1}{2} = 2 - \frac{1}{m}$. $\square$

Aufgabe 2 VORLESUNGSBEWEIS – TRANSITIVITÄT**(10 Punkte)**

Zu zeigen: Aus $L_1 \leq_p L_2$ und $L_2 \leq_p L_3$ folgt $L_1 \leq_p L_3$.

Voraussetzung. Sei R_1 eine Polynomialzeitreduktion $L_1 \leq_p L_2$ mit Laufzeit $O(n^a)$ und R_2 eine Polynomialzeitreduktion $L_2 \leq_p L_3$ mit Laufzeit $O(n^b)$, wobei $a, b \geq 1$ konstant sind. Nach Definition gilt für alle x, y :

$$x \in L_1 \iff R_1(x) \in L_2, \quad y \in L_2 \iff R_2(y) \in L_3.$$

Konstruktion. Definiere die Komposition $R := R_2 \circ R_1$, also $R(x) = R_2(R_1(x))$: Berechne zuerst $R_1(x)$, wende darauf R_2 an.

Korrektheit. Für jede Eingabe x gilt

$$x \in L_1 \iff R_1(x) \in L_2 \iff R_2(R_1(x)) \in L_3,$$

wobei die erste Äquivalenz aus der Korrektheit von R_1 und die zweite aus der Korrektheit von R_2 (angewandt auf $y = R_1(x)$) folgt. Insgesamt also $x \in L_1 \iff R(x) \in L_3$ – in beiden Richtungen.

Laufzeit. Sei $|x| = n$.

- Die Berechnung von $R_1(x)$ braucht $O(n^a)$ Zeit. Da in dieser Zeit höchstens $O(n^a)$ Symbole geschrieben werden, hat die Ausgabe $R_1(x)$ die Größe $|R_1(x)| = O(n^a)$.
- Auf dieser Eingabe der Größe $O(n^a)$ braucht R_2 die Zeit $O((n^a)^b) = O(n^{ab})$.
- Gesamtlaufzeit $O(n^a) + O(n^{ab}) = O(n^{ab})$; wegen $a, b \geq 1$ ist n^{ab} ein Polynom in n .

Damit ist R eine in Polynomialzeit berechenbare Funktion mit $x \in L_1 \iff R(x) \in L_3$, also eine Reduktion $L_1 \leq_p L_3$. $\square$

Sei L das Problem SUBSETSUM-OHNE-DREIERPOTENZEN.

∈ **NP**: Eingabe: Zahlen $c_1, \dots, c_n$, Zielwert K und Zertifikat $S \subseteq [n]$. Der Verifizierer arbeitet wie folgt:

- Prüfe für jede Größe c_i , dass sie keine Dreierpotenz ist (teile c_i so lange wie möglich durch 3; ist das Ergebnis $\neq 1$ bzw. tritt ein Rest auf, so ist c_i keine Dreierpotenz); lehne sonst ab.
- Prüfe $\sum_{i \in S} c_i = K$; lehne sonst ab.
- Sonst akzeptiere.

Korrektheit: Ist die Instanz eine gültige Variante mit Lösung, so ist diese ein akzeptiertes Zertifikat; andernfalls scheitert jedes Zertifikat an einer Prüfung.

Laufzeit: Je Größe der Dreier-Test $O(\log c_i)$ und die Summe bilden $O(n)$ – polynomiell. Also gilt $L \in \text{NP}$.

NP-schwer: Zeige die NP-Schwere durch die Reduktion

$$\text{SUBSETSUM} \leq_p L.$$

SUBSETSUM ist bereits als NP-vollständig bekannt.

Konstruktion: Sei $(c_1, \dots, c_n, K)$ eine SUBSETSUM-Instanz mit $c_i \in \mathbb{N}_{>0}$. Skaliere mit 2:

$$c'_i := 2c_i \quad (i \in [n]), \quad K' := 2K.$$

Jede neue Größe $c'_i = 2c_i \geq 2$ ist gerade. Dreierpotenzen 3^t sind dagegen für jedes $t \in \mathbb{N}_0$ ungerade (Produkt ungerader Faktoren bzw. $3^0 = 1$). Eine gerade Zahl ≥ 2 ist also nie eine Dreierpotenz; insbesondere ist $c'_i \neq 1$ und $c'_i \neq 3^t$ für alle $t \geq 1$. Damit liegt die konstruierte Instanz $(c'_1, \dots, c'_n, K')$ in L .

Beweis SubsetSum nach Variante:

- Sei S mit $\sum_{i \in S} c_i = K$.
- Dann gilt $\sum_{i \in S} c'_i = \sum_{i \in S} 2c_i = 2 \sum_{i \in S} c_i = 2K = K'$.
- Also ist S eine Lösung der Variante.

Beweis Variante nach SubsetSum:

- Sei S mit $\sum_{i \in S} c'_i = K'$.
- Das heißt $2 \sum_{i \in S} c_i = 2K$.
- Division durch 2 liefert $\sum_{i \in S} c_i = K$.
- Also ist S eine SUBSETSUM-Lösung.

Laufzeit: Jede der n Zahlen und K mit 2 multiplizieren – $O(n)$.

Da SUBSETSUM NP-vollständig ist, $\text{SUBSETSUM} \leq_p L$ gilt und $L \in \text{NP}$, ist $L = \text{SUBSETSUM-OHNE-DREIERPOTENZEN}$ NP-vollständig. □

Teil 1: k -IndependentSet.

a) Der Komplementgraph hat dieselben Knoten und als Kanten genau die Nicht-Kanten von G :

$$|V'| = |V|, \quad |E'| = \binom{|V|}{2} - |E| = \frac{1}{2}|V|(|V| - 1) - |E| \leq |V|^2.$$

b)

Bzgl. $|V'|$:

- Angenommen, k -INDEPENDENTSET wäre in $2^{o(|V'|)} \cdot |I|^{O(1)}$ lösbar.
- Ein Independent Set in $G' = \bar{G}$ ist genau eine Clique in G ; die Reduktion ist in Polynomialzeit berechenbar. Der Algorithmus löst also k -CLIQUE.
- Wegen $|V'| = |V|$ wird daraus ein $2^{o(|V|)} \cdot |I|^{O(1)}$ -Algorithmus für k -CLIQUE.
- Widerspruch zum Hinweis.

Bzgl. $|E'|$:

- Angenommen, k -INDEPENDENTSET wäre in $2^{o(\sqrt{|E'|})} \cdot |I|^{O(1)}$ lösbar.
- Wegen $|E'| \leq |V|^2$ ist $\sqrt{|E'|} \leq |V|$, also $\sqrt{|E'|} = O(|V|)$ und damit $o(\sqrt{|E'|}) \subseteq o(|V|)$.
- Via Komplementbildung wird daraus ein $2^{o(|V|)} \cdot |I|^{O(1)}$ -Algorithmus für k -CLIQUE.
- Widerspruch zum Hinweis.

Bounds: kein $2^{o(|V'|)}$, kein $2^{o(\sqrt{|E'|})}$.

Teil 2: SubsetSum.

c) Da die Reduktion pro Menge $S_j \in F$ genau ein Item erzeugt, gilt für die Itemzahl

$$n = |F|.$$

d)

Bzgl. n :

- Angenommen, SUBSETSUM wäre in $2^{o(\sqrt[4]{n})} \cdot |I|^{O(1)}$ lösbar.
- Die Reduktion 3 -EXACTCOVER $\leq_p$ SUBSETSUM ist in Polynomialzeit berechenbar und erzeugt eine Instanz mit $n = |F|$ Items.
- Verkettet man sie mit dem angenommenen Algorithmus, so entsteht wegen $n = |F|$ ein $2^{o(\sqrt[4]{|F|})} \cdot |I|^{O(1)}$ -Algorithmus für 3 -EXACTCOVER.
- Widerspruch zum Hinweis.

Bound: kein $2^{o(\sqrt[4]{n})}$.

□

Aufgabe 5 APPROXIMATIVE ALGORITHMEN – 2ApproxVC (2 + 5 + 3 + (10) Punkte)

a) Der Algorithmus fügt bei einer Kante beide Endpunkte hinzu, falls beide noch frei (nicht in C) sind. Kantenreihenfolge $\{1, 2\}, \{1, 3\}, \{2, 3\}, \{3, 4\}, \{4, 5\}, \{4, 6\}, \{5, 6\}$:

- $\{1, 2\}$: beide frei $\Rightarrow$ aufnehmen, $C = \{1, 2\}$.
- $\{1, 3\}$: $1 \in C \Rightarrow$ überspringen.
- $\{2, 3\}$: $2 \in C \Rightarrow$ überspringen.
- $\{3, 4\}$: beide frei $\Rightarrow$ aufnehmen, $C = \{1, 2, 3, 4\}$.
- $\{4, 5\}$: $4 \in C \Rightarrow$ überspringen.
- $\{4, 6\}$: $4 \in C \Rightarrow$ überspringen.
- $\{5, 6\}$: beide frei $\Rightarrow$ aufnehmen, $C = \{1, 2, 3, 4, 5, 6\}$.

Der Algorithmus liefert $C = \{1, 2, 3, 4, 5, 6\}$ mit $|C| = 6$.

Vergleich mit dem Optimum: G besteht aus den beiden knotendisjunkten Dreiecken $\{1, 2, 3\}$ und $\{4, 5, 6\}$ (verbunden durch die Brücke $\{3, 4\}$). Jedes Dreieck erzwingt mindestens 2 Knoten im Cover (bei nur einem bleibt eine Kante ungedeckt), und die Dreiecke sind disjunkt, also $|C^*| \geq 4$. Die Menge $\{1, 2, 4, 5\}$ deckt alle Kanten ab (auch die Brücke via 4), somit $|C^*| = 4$. Die Rate in diesem Fall ist $\frac{|C|}{|C^*|} = \frac{6}{4} = \frac{3}{2}$.

b) Zu zeigen: $|C| \leq 2|C^*|$ für ein minimales Vertex Cover C^* . Sei A die Menge der Kanten, bei denen *beide* Endpunkte aufgenommen wurden; da jede solche Kante genau zwei neue Knoten beisteuert und kein Knoten doppelt aufgenommen wird, gilt $|C| = 2|A|$.

A ist ein Matching. Hätten zwei Kanten aus A einen gemeinsamen Endpunkt w , so wäre bei der später betrachteten dieser Kanten der Knoten w bereits in C gewesen – die Bedingung „beide Endpunkte frei“ wäre verletzt und die Kante nicht aufgenommen worden. Also sind die Kanten aus A paarweise knotendisjunkt.

Schranke für das Optimum. C^* deckt jede Kante aus A mit mindestens einem Endpunkt ab. Da die Kanten aus A knotendisjunkt sind, braucht jede einen *eigenen* Knoten in C^* , also $|C^*| \geq |A|$.

Kombination.

$$|C| = 2|A| \leq 2|C^*|.$$

Somit hat 2APPROXVC die Güte 2. □

c) Der Kreis C_n (mit n gerade) hat als größte unabhängige Menge die $n/2$ „jeden zweiten Knoten“; also ist die maximale Unabhängigkeitszahl $n/2$ und ein minimales Vertex Cover hat Größe

$$|C^*| = n - \frac{n}{2} = \frac{n}{2}.$$

Schlechteste Kantenreihenfolge: Man ordnet die $n/2$ paarweise disjunkten Kanten $\{v_1, v_2\}, \{v_3, v_4\}, \dots, \{v_{n-1}, v_n\}$ zuerst an (danach beliebig die restlichen Kanten). Beim Abarbeiten dieser $n/2$ Kanten sind jeweils beide Endpunkte noch frei, sodass der Algorithmus *alle* n Knoten aufnimmt; die übrigen Kanten haben dann stets einen bereits bedeckten Endpunkt. Es folgt $|C| = n$ und die Rate

$$\frac{|C|}{|C^*|} = \frac{n}{n/2} = 2,$$

also der nach b) größtmögliche Wert.

d) BONUS. Wir geben die Familie der *ungeraden* Kreise C_{2k+1} für $k \geq 2$ an, mit Knoten $v_1, \dots, v_{2k+1}$ und Kanten $\{v_i, v_{i+1}\}$ (Indizes modulo $2k+1$).

- *Optimum:* Die größte unabhängige Menge in C_{2k+1} hat Größe k (man kann höchstens jeden zweiten Knoten wählen, im ungeraden Kreis also k). Damit ist $|C^*| = (2k+1) - k = k+1$.
- *Ungünstige Reihenfolge:* Ordne die k paarweise disjunkten Kanten $\{v_1, v_2\}, \{v_3, v_4\}, \dots, \{v_{2k-1}, v_{2k}\}$ zuerst an. Beim Abarbeiten sind jeweils beide Endpunkte frei, also nimmt der Algorithmus die $2k$ Knoten $v_1, \dots, v_{2k}$ auf. Der einzige unbedeckte Knoten wäre v_{2k+1} ; seine beiden Kanten $\{v_{2k}, v_{2k+1}\}$ und $\{v_{2k+1}, v_1\}$ haben aber je einen Endpunkt in C , werden also übersprungen. Somit $|C| = 2k$.

- *Rate:*

$$\frac{|C|}{|C^*|} = \frac{2k}{k+1} = 2 - \frac{2}{k+1} \xrightarrow{k \rightarrow \infty} 2,$$

und die Rate ist streng monoton wachsend sowie für jedes endliche k echt kleiner als 2 (z.B. C_5 : $\frac{4}{3}$, C_7 : $\frac{3}{2}$, C_9 : $\frac{8}{5}$, C_{11} : $\frac{5}{3}$).

Die Approximationsrate von 2APPROXVC konvergiert also entlang der Familie $(C_{2k+1})_{k \geq 2}$ gegen 2; zusammen mit b) ist die Güte 2 damit asymptotisch scharf. $\square$