

AAK – Übung

Beweistechniken und Klausurtraining

Aufgabentypen, Musterbeweise und Selbsttests zu Serie 9–13

Klausurvorbereitung „Analyse von Algorithmen und Komplexität“ (CAU Kiel)

Sommersemester 2026

Wie du dieses Dokument benutzt. Voraussetzung ist **Fundament.pdf** – alle Begriffe und Problemdefinitionen werden hier als bekannt vorausgesetzt. Dieses PDF trainiert die *Aufgabentypen der Klausur*: Pro Kapitel bekommst du ein Rezept, komplett ausformulierte Musterbeweise (aus Musterlösungen der Serien und dem Skript) und 1–2 *Selbsttest-Aufgaben* aus echten Altklausuren. Löse die Selbsttests schriftlich unter Zeitdruck, bevor du die Lösungen im Anhang [A](#) liest.

Der Klausur-Bauplan. Alle vier Altklausuren (SS23/SS24, je Haupt- und Nachklausur) folgen demselben Skelett – 5–6 Aufgaben, 120 Minuten, ein handbeschriebenes Blatt erlaubt:

#	Aufgabentyp	Punkte	Kapitel hier
1	Approx-Algorithmus <i>anwenden</i> (LPT, Christofides, Greedy, ΔTSP_1)	~10	4
2	<i>Vorlesungsbeweis</i> reproduzieren	10	2
3	<i>NP-Schwere per Reduktion</i> (Variantenprobleme, 2 Stück)	5+5	3
4	<i>Güte-Beweis</i> + Worst-Case-Instanz konstruieren	~10	4.2
5	<i>ETH-Lower-Bounds</i> aus vorgegebener Reduktion	5+5	5
(6)	NP-Zugehörigkeit (Zertifikat/Verifizierer/NTM) – nur SS23	10	1

Inhaltsverzeichnis

1	NP-Zugehörigkeit zeigen	3
1.1	Musterlösung: KNAPSACK $\in$ NP (Präsenz 9.1)	3
1.2	Musterlösung: FVS $\in$ NP per Verifizierer (HA 9.2)	3
1.3	Klausurniveau durchgerechnet: LONGEST PATH (SS23 A2)	3
2	Die vier Vorlesungsbeweise (der 10-Punkte-Block)	4
2.1	Beweis 1: SAT $\leq$ 3-SAT (Skript Satz 6.25)	4
2.2	Beweis 2: k -CLIQUE $\in$ NP (Skript Lemma 6.13)	5
2.3	Beweis 3: SAT $\leq k$ -CLIQUE (Skript Satz 6.26)	5
2.4	Beweis 4: List Scheduling hat Güte $2 - \frac{1}{m}$ (Skript Satz 7.24a)	6
3	NP-Schwere per Reduktion: das Handwerk	6
3.1	Muster 1: Zusatzeigenschaft erzwingen – k -CLIQUE UNIVERSAL (HA 10.2)	7
3.2	Muster 2: Komplement-Dualität – CLIQUE $\leq$ VC (Präsenz 10.2)	7
3.3	Muster 3: Antiparallele Bögen – VC $\leq$ FVS (HA 10.1)	8
3.4	Muster 4: Gadget + o.B.d.A.-Bereinigung – VC $\leq \Delta$ -COVER (HA 11.1)	8
3.5	Muster 5: Padding und Shift – SUBSET SUM $\leq$ SUBSET SUM CARDINALITY (Präsenz 11.3)	8
3.6	Muster 6: Dummy-Knoten – CLIQUE-NOMEMBER (Präsenz 11.2)	9
3.7	Gadget-Katalog	9

4	Approximation: anwenden, beweisen, Worst Case bauen	10
4.1	Anwenden: die Rechenaufgabe	10
4.2	Güte-Beweise: die drei Beweismuster	11
4.3	Worst-Case-Instanzen konstruieren	12
5	ETH-Lower-Bounds: das Schema	13
5.1	Die Vorlesungs-Reduktionskette mit allen Schranken (Präsenz 13.1)	14
5.2	Musterlösung 1: Lower Bounds für VERTEX COVER (HA 13.1)	14
5.3	Musterlösung 2: Lower Bounds für HITTING SET (HA 13.2)	14
6	Kurz: NP-Schwere via Unentscheidbarkeit (Halteproblem)	15
7	Klausur-Fahrplan und Checklisten	15
A	Lösungen zu den Selbsttests	15
A.1	Lösung zu Selbsttest 1: DOMINATING SET	15
A.2	Lösung zu Selbsttest 2: CLIQUEANDINDEPENDENTSET	16
A.3	Lösung zu Selbsttest 3: SUBSET SUM mit Teilbarkeit	16
A.4	Lösung zu Selbsttest 4: HAMILTONIANPATH $\leftrightarrow$ HAMILTONIANCYCLE	16
A.5	Lösung zu Selbsttest 5: APPROXIMATESUBSETSUM	16
A.6	Lösung zu Selbsttest 6: MIN-EDGE-COVER	17
A.7	Lösung zu Selbsttest 7: Lower Bounds für k -COLOR	17
A.8	Lösung zu Selbsttest 8: Lower Bounds für SET COVER	17

1 NP-Zugehörigkeit zeigen

Aufgabenformat (Klausur SS23, beide Male identisch): „Zeigen Sie auf verschiedene Weisen, dass das Problem in NP liegt: (a) Zertifikat angeben, (b) Verifizierer beschreiben + Laufzeit konkret, (c) NTM in Worten + Laufzeit.“

Rezept: Der NP-Dreischritt

(a) Zertifikat: Die „Lösung“ des Problems als Objekt – Teilmenge, Belegung, Permutation. Angeben, wie es kodiert ist (z. B. ein Bit pro Element) und dass es polynomiell lang ist.

(b) Verifizierer: Deterministischer Algorithmus, der (Instanz, Zertifikat) bekommt und *alle* geforderten Eigenschaften prüft. Danach drei Punkte abhaken:

1. *Beschreibung:* Welche Checks laufen in welcher Reihenfolge?
2. *Korrektheit:* Ja-Instanz $\Rightarrow$ es existiert ein Zertifikat, das akzeptiert wird; Nein-Instanz $\Rightarrow$ kein Zertifikat wird akzeptiert.
3. *Laufzeit:* pro Check abschätzen, Summe polynomiell (in der Klausur: konkret in O -Notation!).

(c) NTM: „Rate und prüfe“ – für jedes Element nichtdeterministisch entscheiden, ob es zur Lösung gehört; danach deterministisch dieselben Checks wie in (b). Korrektheit und Laufzeit wie oben (polynomiell viele Rateschritte + polynomielle Prüfung).

Merke

Zusammenhang (a)–(c): Die Folge der Rateschritte der NTM *ist* das Zertifikat. Beide Wege nutzen die Extra-Ressource, um eine Lösungskandidatin zu bestimmen und dann nur noch zu *prüfen*.

1.1 Musterlösung: Knapsack $\in$ NP (Präsenz 9.1)

NTM: Für jeden Gegenstand entscheide nichtdeterministisch, ob er in den Rucksack kommt. Prüfe anschließend $\sum_{i \in S} w_i \leq K$ und $\sum_{i \in S} p_i \geq P$; akzeptiere genau dann. *Korrektheit:* Existiert eine zulässige Füllung mit genügend Profit, gibt es eine Folge von Entscheidungen, die sie findet – diese wird akzeptiert. Existiert keine, lehnt jeder Rechenweg ab. *Laufzeit:* n Rateschritte, Summation und Vergleich in $O(n)$ Arithmetikschritten – polynomiell.

Verifizierer: Zertifikat ist $S \subseteq [n]$ (ein Bit pro Gegenstand). Berechne $\sum_{i \in S} w_i$ und $\sum_{i \in S} p_i$, vergleiche mit K und P , akzeptiere entsprechend. *Korrektheit und Laufzeit:* analog.

1.2 Musterlösung: FVS $\in$ NP per Verifizierer (HA 9.2)

Zertifikat: $X \subseteq V$ (ein Bit pro Knoten). Prüfe (1) $|X| \leq k$, (2) $G \setminus X$ ist azyklisch – dafür den Algorithmus zur *topologischen Sortierung* aus der Vorlesung nutzen: Er findet genau dann eine topologische Sortierung, wenn der Graph kreisfrei ist. Beide Checks polynomiell $\Rightarrow$ polynomieller Verifizierer.

Merke

Nutze bekannte polynomielle Algorithmen als Bausteine des Verifizierers (topologische Sortierung, DFS/BFS, Sortieren) – mit Namen zitieren statt neu erfinden.

1.3 Klausurniveau durchgerechnet: Longest Path (SS23 A2)

Gegeben $G = (V, E)$ und k ; gibt es einen einfachen Pfad der Länge $\geq k$?

(a) **Zertifikat:** Eine Folge von $k + 1$ paarweise verschiedenen Knoten $(v_0, v_1, \dots, v_k)$ – der behauptete Pfad. Länge: $O(k \log |V|)$ Bits, polynomiell.

(b) **Verifizierer:** Prüfe (1) alle v_i paarweise verschieden – $O(k^2)$ Vergleiche (oder Sortieren), (2) $\{v_i, v_{i+1}\} \in E$ für alle $i = 0, \dots, k - 1$ – k Lookups in der Adjazenzmatrix, je $O(1)$, also $O(k)$. Gesamt: $O(k^2) \subseteq O(|V|^2)$, polynomiell in der Eingabegröße. *Korrektheit:* Hat G einen Pfad der Länge $\geq k$, so bilden seine ersten $k + 1$ Knoten ein akzeptiertes Zertifikat; wird umgekehrt ein Zertifikat akzeptiert, ist es ein einfacher Pfad der Länge k .

(c) **NTM:** Rate nacheinander $k + 1$ Knoten auf ein Arbeitsband ($O(k \log |V|)$ nichtdeterministische Schritte). Prüfe dann deterministisch Verschiedenheit und Kantenbedingung wie in (b). Laufzeit: Raten $O(k \log |V|)$ plus Prüfung $\text{poly}(|V|)$ – insgesamt polynomiell.

Achtung — typische Falle

Häufige Fehlerquellen bei diesem Aufgabentyp:

- *Korrektheit nur halb:* Es gehören **beide** Richtungen dazu (Ja-Instanz $\Rightarrow$ akzeptierendes Zertifikat existiert; akzeptiert $\Rightarrow$ Ja-Instanz).
- *Laufzeit vergessen oder nur „polynomiell“ sagen,* wenn konkrete O -Notation verlangt ist.
- Beim Verifizierer die *Kardinalitätsprüfung* ($|C| \leq k$ bzw. $\geq k$) unterschlagen.

Punkteschema der Serien: Beschreibung 2 P., Korrektheit 1 P., Laufzeit 1 P.

Selbsttest: 1 – Dominating Set (Klausur SS23-N, A2; 3+3+4 P.)

$G = (V, E)$ ungerichtet, $k \in \mathbb{N}_0$. Gibt es $D \subseteq V$, $|D| \leq k$, sodass jeder Knoten in D liegt oder zu einem Knoten in D benachbart ist?

- Geben Sie ein Zertifikat an.
- Beschreiben Sie einen polynomiellen Verifizierer; Laufzeit konkret in O -Notation.
- Beschreiben Sie in Worten eine polynomielle NTM; Laufzeit konkret in O -Notation.

Lösung: Anhang A.1.

2 Die vier Vorlesungsbeweise (der 10-Punkte-Block)

In **jeder** Altklausur gibt es genau eine Aufgabe „Beweis zur Vorlesung“ für 10 Punkte. Bisher kamen dran: $\text{SAT} \leq k\text{-CLIQUE}$ (SS24-H), $\text{SAT} \leq 3\text{-SAT}$ (SS24-N), $k\text{-CLIQUE}$ NP-vollständig (SS23-H), List-Scheduling-Güte $2 - 1/m$ (SS23-N). Diese vier Beweise musst du *reproduzieren* können – sie gehören auf dein erlaubtes handschriftliches Blatt.

2.1 Beweis 1: $\text{SAT} \leq 3\text{-SAT}$ (Skript Satz 6.25)

Aussage: 3-SAT ist NP-vollständig. (In der Klausur SS24 durfte $3\text{-SAT} \in \text{NP}$ angenommen werden; sonst: Belegung raten und prüfen.)

Beweis. Zu zeigen: $\text{SAT} \leq 3\text{-SAT}$. Gesucht ist eine polynomzeitberechenbare Transformation $\alpha \mapsto \bar{\alpha}$ mit $\alpha \in \text{SAT} \iff \bar{\alpha} \in 3\text{-SAT}$.

Idee: Eine lange Klausel wird mit Hilfsvariablen in eine Kette von 3er-Klauseln aufgespalten:

$$(y_1 \vee \dots \vee y_n) \text{ erfüllbar} \iff (y_1 \vee y_2 \vee x_1) \wedge (\neg x_1 \vee y_3 \vee x_2) \wedge \dots \wedge (\neg x_{n-3} \vee y_{n-1} \vee y_n) \text{ erfüllbar},$$

wobei $x_1, \dots, x_{n-3}$ neue Hilfsvariablen sind (pro Klausel eigene).

$\Rightarrow$: Sei $(y_1 \vee \dots \vee y_n)$ wahr, etwa y_i wahr. Setze $x_1 = \dots = x_{i-2} = \text{true}$ und $x_{i-1} = \dots = x_{n-3} = \text{false}$. Dann ist jede Kettenklausel erfüllt: Die ersten $i - 2$ durch ihr x -Literal, die Klausel mit y_i durch y_i , die restlichen durch ihr $\neg x$ -Literal.

$\Leftarrow$: Sei die rechte Seite erfüllt und angenommen, alle y_i wären falsch. Dann erzwingt die erste Klausel $x_1 = \text{true}$, damit die zweite $x_2 = \text{true}$, induktiv $x_{n-3} = \text{true}$ – aber die letzte Klausel $(\neg x_{n-3} \vee y_{n-1} \vee y_n)$ ist dann falsch, Widerspruch. Also ist ein y_i wahr und die Originalklausel erfüllt.

Wende dies klauselweise an. Es gilt $|\bar{\alpha}| \leq c|\alpha|$ mit $c = O(1)$, und die Transformation ist in Polynomialzeit berechenbar. Da SAT NP-vollständig und $3\text{-SAT} \in \text{NP}$ ist, ist 3-SAT NP-vollständig. $\square$

2.2 Beweis 2: $k\text{-Clique} \in \text{NP}$ (Skript Lemma 6.13)

Dieser kleine Beweis ist Teil des Vorlesungsbeweises „ $k\text{-CLIQUE}$ ist NP-vollständig“.

Beweis. Polynomieller Verifizierer: Neben der Eingabe $(G = (V, E), k)$ erhält der Algorithmus ein Zertifikat $C \subseteq V$. Er prüft: (1) für alle zweielementigen Teilmengen $\{u, v\} \subseteq C$, ob $\{u, v\} \in E$ – Zeit $O(|C|^2) = O(|V|^2)$; (2) ob $|C| \geq k$ – Zeit $O(|V|)$. Ausgabe Ja genau dann, wenn beide Tests positiv sind. Da die Eingabelänge $\Omega(|V| + |E|)$ ist, ist die Laufzeit polynomiell. $\square$

2.3 Beweis 3: $\text{SAT} \leq k\text{-Clique}$ (Skript Satz 6.26)

Achtung — typische Falle

Die Klausur SS24 verlangte ausdrücklich die Reduktion vom *allgemeinen* SAT (beliebig lange Klauseln) – **nicht** von 3-SAT! Die Konstruktion unten funktioniert für beide.

Aussage: $k\text{-CLIQUE}$ ist NP-vollständig.

Beweis. (a) $k\text{-CLIQUE} \in \text{NP}$: siehe Beweis 2.

(b) $\text{SAT} \leq k\text{-CLIQUE}$: Sei $F = F_1 \wedge \dots \wedge F_m$ in KNF mit Klauseln $F_i = (y_{i1} \vee \dots \vee y_{i\ell_i})$. Konstruktion von $G = (V, E)$ in polynomieller Zeit:

$$\begin{aligned} V &= \{[i, j] \mid 1 \leq i \leq m, 1 \leq j \leq \ell_i\} && \text{(ein Knoten pro Literalvorkommen)} \\ E &= \{[i, j], [i', j'] \mid i \neq i' \text{ und } y_{ij} \neq \neg y_{i'j'}\} && \text{(verschiedene Klausel, nicht widersprüchlich)} \end{aligned}$$

Setze $k = m$. *Behauptung:* F erfüllbar $\iff G$ enthält eine Clique mit m Knoten.

$\Rightarrow$: Sei ψ erfüllende Belegung. Dann gibt es für jedes i ein r_i mit $\psi(y_{ir_i}) = \text{true}$. Setze $C = \{[i, r_i] \mid 1 \leq i \leq m\}$. Wäre $\{[i, r_i], [j, r_j]\} \notin E$ für $i \neq j$, dann müsste (nach Definition von E) $y_{ir_i} = \neg y_{jr_j}$ gelten – aber beide sind unter ψ wahr, Widerspruch. Also ist C eine $m\text{-Clique}$.

$\Leftarrow$: Sei C eine $m\text{-Clique}$. Da Knoten derselben Klausel nie verbunden sind, hat C die Form $C = \{[1, r_1], \dots, [m, r_m]\}$. Definiere ψ mit $\psi(y_{1r_1}) = \dots = \psi(y_{mr_m}) = \text{true}$. Das ist widerspruchsfrei, da $y_{ir_i} \neq \neg y_{jr_j}$ für alle $i \neq j$ (sonst keine Kante). Dann ist jede Klausel F_i erfüllt, also $\psi(F) = \text{true}$.

Da SAT NP-vollständig ist, $\text{SAT} \leq k\text{-CLIQUE}$ gilt und $k\text{-CLIQUE} \in \text{NP}$, ist $k\text{-CLIQUE}$ NP-vollständig. $\square$

Beispiel: Konstruktion konkret (Skript Beispiel 6.27)

$F = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2) \wedge (x_1 \vee \neg x_2 \vee \neg x_3)$, also $m = 3$. Knoten: $[1, 1], [1, 2], [1, 3]$ (Klausel 1), $[2, 1], [2, 2]$ (Klausel 2), $[3, 1], [3, 2], [3, 3]$ (Klausel 3). Kanten zwischen allen Paaren aus verschiedenen Klauseln *außer* z.B. $\{[1, 1], [2, 1]\}$ (x_1 vs. $\neg x_1$) oder $\{[2, 2], [3, 2]\}$ (beide $\neg x_2$: kein Widerspruch, Kante existiert!). Nur *komplementäre* Literalpaare bekommen keine Kante. Die Clique $\{[1, 1], [2, 2], [3, 1]\}$ ($x_1, \neg x_2, x_1$) liefert $\psi(x_1) = \text{true}, \psi(x_2) = \text{false}$ (x_3 frei).

2.4 Beweis 4: List Scheduling hat Güte $2 - \frac{1}{m}$ (Skript Satz 7.24a)

Aussage: Für alle Eingaben $I = (L, m)$ gilt $\text{LS}(I)/\text{OPT}(I) \leq 2 - 1/m$.

Beweis. O.B.d.A. habe Maschine M_1 nach der Zuordnung die höchste Last $L = \sum_{j \in B_1} p_j$, d. h. $\text{LS}(I) = L$. Sei J_k der letzte Job, der M_1 zugeordnet wurde. *Kernbeobachtung:* Zum Zeitpunkt der Zuordnung von J_k hatte M_1 die *kleinste* Last $L - p_k$ – also haben *alle* Maschinen mindestens Last $L - p_k$. Daraus folgt

$$\sum_{i=1}^n p_i \geq m(L - p_k) + p_k.$$

Außerdem gelten die beiden Standard-Schranken für das Optimum:

$$\text{OPT}(I) \geq \frac{1}{m} \sum_{i=1}^n p_i \quad \text{und} \quad \text{OPT}(I) \geq p_k.$$

(Die Gesamtlast verteilt sich auf m Maschinen; und jeder einzelne Job muss irgendwo laufen.)
Kombinieren:

$$\text{OPT}(I) \geq \frac{1}{m} \sum_{i=1}^n p_i \geq \frac{m(L - p_k) + p_k}{m} = L - \left(1 - \frac{1}{m}\right)p_k = \text{LS}(I) - \left(1 - \frac{1}{m}\right)p_k.$$

Mit $p_k \leq \text{OPT}(I)$ folgt $\text{OPT}(I) \geq \text{LS}(I) - (1 - \frac{1}{m})\text{OPT}(I)$, also $\text{LS}(I) \leq (2 - \frac{1}{m})\text{OPT}(I)$. $\square$

Merke

Die zwei OPT-Schranken $\text{OPT} \geq \frac{1}{m} \sum p_i$ (Durchschnittslast) und $\text{OPT} \geq p_{\max}$ (größter Job) sind das Universalwerkzeug *aller* Scheduling-Güte-Beweise – auch für RoundRobin (Kapitel 4.2) und LPT.

Selbsttest: Selbstkontrolle ohne Lösung im Anhang

Reproduziere jeden der vier Beweise handschriftlich, ohne in dieses Dokument zu schauen. Kontrolliere gegen die jeweilige Stelle hier. Wiederhole jeden Beweis, bei dem du einen Schritt ausgelassen hast – besonders gern vergessen: die Polynomialität der Transformation und die Rückrichtung.

3 NP-Schwere per Reduktion: das Handwerk

Der häufigste Klausur-Aufgabentyp: Ein *Variantenproblem* (bekanntes Problem mit Zusatzeigenschaft) soll als NP-schwer oder NP-vollständig bewiesen werden.

Rezept: NP-Vollständigkeits-Boilerplate (6 Schritte)

1. **NP:** Verifizierer angeben (oft: „wie beim Originalproblem, da nur die Instanzen eingeschränkt sind“). Bei reiner NP-*Schwere*-Frage entfällt dieser Schritt.
2. **Quellproblem wählen:** ein bekanntes NP-vollständiges Problem X , das dem Zielproblem Y möglichst ähnlich ist. Richtung: $X \leq Y$!
3. **Konstruktion angeben:** Wie wird aus einer X -Instanz eine Y -Instanz? Präzise, mit allen Parametern (auch k anpassen!).
4. **Polynomialität:** Konstruktion läuft in Polynomialzeit (meist 1 Satz: „naiv in $O(\cdot)$ “).
5. **Korrektheit, beide Richtungen:** $\Rightarrow$ Ja-Instanz von X liefert Ja-Instanz von Y ; $\Leftarrow$ Ja-Instanz von Y liefert Ja-Instanz von X .
6. **Schlusssatz:** „Also existiert eine polynomielle Reduktion von X auf Y . Da X NP-vollständig ist und $Y \in \text{NP}$ gilt, ist Y NP-vollständig.“

Punkteschema (HA 11.1): Verifizierer 1+0,5+0,5 P.; Angabe Reduktion 1 P.; Korrektheit 3 P. (1,5 pro Richtung); Laufzeit 1 P.; Beweisführung 1 P.

Achtung — typische Falle

Die drei Klassiker-Fehler:

- **Falsche Richtung:** Es muss das *bekannte* Problem auf das *neue* reduziert werden ($X_{\text{bekannt}} \leq Y_{\text{neu}}$).
- **Rückrichtung vergessen** – kostet 1,5 von 5 Punkten.
- Die konstruierte Instanz erfüllt die *Zusatzeigenschaft* des Zielproblems nicht (z. B. universeller Knoten fehlt, Grad-Bedingung verletzt).

3.1 Muster 1: Zusatzeigenschaft erzwingen – k -Clique Universal (HA 10.2)

Das Musterbeispiel; im Aufgabentext wörtlich als „klassische Klausuraufgabe“ bezeichnet. Gegeben: k -CLIQUE, aber die Instanz enthält garantiert einen *universellen Knoten* (mit allen verbunden).

Beweis. $\in \text{NP}$: Jeder Verifizierer für k -CLIQUE eignet sich, da lediglich eine Zusatzeigenschaft der Instanz hinzukam.

Reduktion $k\text{-CLIQUE} \leq k\text{-CLIQUE UNIVERSAL}$: Gegeben $(G = (V, E), k)$. Definiere $\bar{k} = k + 1$ und $\bar{G} = (\bar{V}, \bar{E})$ mit $\bar{V} = V \cup \{u\}$ für einen neuen Knoten $u \notin V$ und $\bar{E} = E \cup \{\{u, v\} \mid v \in V\}$. Der Knoten u ist universell, also ist $(\bar{G}, \bar{k})$ eine gültige Instanz; berechenbar in $O(|V|)$.

$\Rightarrow$: Sei C eine Clique in G mit $|C| \geq k$. Da u mit allen Knoten verbunden ist, ist $\bar{C} = C \cup \{u\}$ eine Clique in $\bar{G}$ mit $|\bar{C}| \geq k + 1 = \bar{k}$.

$\Leftarrow$: Sei $\bar{C}$ eine Clique in $\bar{G}$ mit $|\bar{C}| \geq \bar{k}$. Dann ist $C = \bar{C} \setminus \{u\} \subseteq V$ eine Clique in G (Teilmenge einer Clique; u ist der einzige neue Knoten) mit $|C| \geq \bar{k} - 1 = k$.

Also ist $k\text{-CLIQUE UNIVERSAL}$ NP-vollständig. $\square$

Merke

Schema „bekanntes Problem + Zusatzeigenschaft“: Verifizierer erben; die Reduktion muss nur die Zusatzstruktur *herstellen* (Knoten/Item hinzufügen, skalieren) und den Parameter k nachziehen. Prüfe immer: Erfüllt *jede* konstruierte Instanz die Zusatzeigenschaft?

3.2 Muster 2: Komplement-Dualität – Clique $\leq$ VC (Präsenz 10.2)

Beweis. $\text{VC} \in \text{NP}$: bekannt (HA 9.1).

Reduktion: Gegeben $(G = (V, E), k)$ mit $n = |V|$ (o.B.d.A. $k \leq n$). Invertiere den Graphen zu G' (Komplementgraph: genau die Nicht-Kanten von G). Gib $(G', n - k)$ als VC-Instanz aus.

$\Rightarrow$: G enthalte eine Clique C , $|C| \geq k$. Dann ist $V' := V \setminus C$ ein Vertex Cover von G' mit $|V'| \leq n - k$: In G' gibt es keine Kante zwischen Knoten aus C (in G waren alle da), also hat jede Kante von G' einen Endpunkt außerhalb von C , d.h. in V' .

$\Leftarrow$: G' enthalte ein VC V' mit $|V'| \leq n - k$. Dann ist $C := V \setminus V'$ eine Clique in G mit $|C| \geq k$: Für $\{u, v\} \subseteq C$ gilt $\{u, v\} \notin E'$ (sonst wäre V' kein VC), also $\{u, v\} \in E$.

Laufzeit: Invertieren naiv in $O(|V|^2)$, polynomiell. Mit dem Satzsatz folgt: VC ist NP-vollständig. $\square$

3.3 Muster 3: Antiparallele Bögen – VC $\leq$ FVS (HA 10.1)

Beweis. FVS $\in$ NP: bekannt (HA 9.2, topologische Sortierung).

Reduktion: Für eine VC-Instanz $(G = (V, E), k)$ erzeuge den gerichteten Graphen $G' = (V, E')$ mit $E' := \{(u, v), (v, u) \mid \{u, v\} \in E\}$ – jede ungerichtete Kante wird zu *zwei antiparallelen Bögen*. Gib (G', k) aus.

$\Rightarrow$: Sei C ein VC mit $|C| \leq k$. Dann enthält $G \setminus C$ keine Kanten, also enthält $G' \setminus C$ keine Bögen und damit keine Kreise.

$\Leftarrow$: Sei X ein FVS mit $|X| \leq k$. Dann ist X ein VC in G : Gäbe es eine Kante $\{u, v\} \in E$ mit $u \notin X$ und $v \notin X$, so wäre (u, v, u) ein Kreis in $G' \setminus X$ – Widerspruch.

Laufzeit: $O(|E|)$. Satzsatz $\Rightarrow$ FVS NP-vollständig. $\square$

3.4 Muster 4: Gadget + o.B.d.A.-Bereinigung – VC $\leq$ Δ -Cover (HA 11.1)

Beweis. $\in$ NP: Zertifikat $C_\Delta \subseteq V$; prüfe $|C_\Delta| \leq k$ und für alle $\leq |V|^3$ Knotentripel, ob sie ein Dreieck bilden und dann getroffen werden; $O(|V|^4)$, polynomiell.

Reduktion: Für eine VC-Instanz $(G = (V, E), k)$ bilde $G' = (V', E')$ mit $V' = V \cup \{v_e \mid e \in E\}$ und $E' = E \cup \{\{v_e, e_1\}, \{v_e, e_2\} \mid e = \{e_1, e_2\} \in E\}$ – **jede Kante wird zu einem Dreieck** $\{v_e, e_1, e_2\}$ aufgeblasen. Gib (G', k) aus.

$\Rightarrow$: Sei C ein VC von G , $|C| \leq k$. Jedes Dreieck in G' ist entweder ein Dreieck aus G (drei paarweise verbundene Knoten, deren Kanten alle von C überdeckt sind, also ein Eckknoten in C) oder ein Gadget-Dreieck $\{v_e, e_1, e_2\}$ – und $e_1 \in C$ oder $e_2 \in C$, da C die Kante e überdeckt. Also ist C eine Dreiecksüberdeckung.

$\Leftarrow$: Sei C_Δ eine Dreiecksüberdeckung von G' , $|C_\Delta| \leq k$. **O.B.d.A. gilt** $C_\Delta \subseteq V$: Enthält C_Δ einen Gadget-Knoten v_e , ersetze ihn durch e_1 – die Überdeckung bleibt erhalten (alle Dreiecke durch v_e enthalten e_1 oder e_2 ; das einzige Dreieck mit v_e ist $\{v_e, e_1, e_2\}$) und die Menge wird nicht größer. Nun trifft C_Δ jedes Gadget-Dreieck $\{v_e, e_1, e_2\}$ in e_1 oder e_2 – also ist für jede Kante $e = \{e_1, e_2\} \in E$ ein Endpunkt in C_Δ : ein Vertex Cover.

Laufzeit: $O(|E|)$. Satzsatz $\Rightarrow$ Δ -COVER NP-vollständig. $\square$

Merke

Die **o.B.d.A.-Bereinigung** (Gadget-Knoten in der Rückrichtung durch Original-Knoten ersetzen) ist ein wiederkehrender Trick: Sie macht die Rückrichtung sauber, ohne Fallunterscheidungen zu multiplizieren. Immer begründen, warum die Ersetzung (i) die Lösungseigenschaft erhält und (ii) die Menge nicht vergrößert.

3.5 Muster 5: Padding und Shift – SubSet Sum $\leq$ SubSet Sum Cardinality (Präsenz 11.3)

Zielproblem: wie SubSet Sum, aber zusätzlich $|S| = n/2$ gefordert.

Beweis. Reduktion: Für $I = (c_1, \dots, c_n, K)$ definiere $c'_i := c_i + 1$ für $i \leq n$ („Shift“) und $c'_i := 1$ für $i \in \{n+1, \dots, 2n\}$ (n „Padding-Items“). Ausgabe: $I' = (c'_1, \dots, c'_{2n}, K+n)$ mit Kardinalitätsforderung n (von $2n$ Items).

$\Rightarrow$: Sei S mit $\sum_{i \in S} c_i = K$. Dann $\sum_{i \in S} c'_i = K + |S|$. Fülle mit Padding-Items $S' = \{n+1, \dots, 2n - |S|\}$ auf: $|S \cup S'| = n$ und $\sum_{i \in S \cup S'} c'_i = K + |S| + (n - |S|) = K + n$.

$\Leftarrow$: Sei S' mit $|S'| = n$ und $\sum_{i \in S'} c'_i = K + n$. Setze $S := S' \cap \{1, \dots, n\}$. Die $n - |S|$ Padding-Items in S' tragen je 1 bei, also $\sum_{i \in S} (c_i + 1) = K + n - (n - |S|) = K + |S|$, d. h. $\sum_{i \in S} c_i = K$.

Laufzeit: $O(n)$. *Schlusssatz.* □

Merke

Der SubSet-Sum-Werkzeugkasten (für die Klausur-Varianten):

- *Skalieren:* $c'_i = \lambda c_i$, $K' = \lambda K$ ändert die Lösbarkeit nicht – erzwingt Teilbarkeitseigenschaften (z. B. $\lambda = 3$: alle Größen durch 3 teilbar / keine Größe ist Zweierpotenz).
- *Shift + Padding:* $c'_i = c_i + 1$ plus Einsen-Items – erzwingt Kardinalitäten (Muster 5).
- *Skalieren + Sonder-Item:* $c'_i = 2c_i$, neues Item der Größe 1, $K' = 2K + 1$ – erzwingt, dass das Sonder-Item in jeder Lösung liegt (Parität!).

In jeder Altklausur kam genau eine solche SubSet-Sum-Variante dran.

3.6 Muster 6: Dummy-Knoten – Clique-Nomember (Präsenz 11.2)

Kurzfassung: Gegeben CLIQUE-Instanz (G, k) ; füge einen neuen *isolierten* Knoten v hinzu und frage nach einer k -Clique ohne v in $G' = (V \cup \{v\}, E)$. $\Rightarrow$: Eine k -Clique in G enthält v nicht (v ist neu) – fertig. $\Leftarrow$: Eine k -Clique ohne v in G' liegt komplett in G . Laufzeit $O(1)$ zusätzlich. Der isolierte Knoten ist das einfachste Gadget überhaupt.

3.7 Gadget-Katalog

Zusatzeigenschaft / Ziel	Trick	Beispiel
Verbotener/erzwungener Knoten	isolierten bzw. universellen Knoten hinzufügen, k anpassen	Clique-Nomember; Clique Universal
ungerichtet $\rightarrow$ gerichtet	Kante $\rightarrow$ zwei antiparallele Bögen (2-Kreise)	$VC \leq FVS$
Kanten- $\rightarrow$ Dreiecksstruktur	Kante zu Dreieck aufblasen (neuer Knoten pro Kante)	$VC \leq \Delta$ -Cover
Komplement-Sicht	Graph invertieren; Clique $\leftrightarrow$ IS $\leftrightarrow$ VC	Clique $\leq$ VC
Kardinalität erzwingen Zahleneigenschaften	+1-Shift + Padding-Einsen skalieren ($\times \lambda$), Sonder-Item, Parität	SubSetSum Cardinality SubsetSum-Klausurvarianten
Grad-Bedingungen ($\deg \geq d$)	Gadget an jeden Knoten hängen, das Grad erhöht, ohne Lösungen zu ändern	Hitchhiker's-HK (SS24-N)

Selbsttest: 2 – CliqueAndIndependentSet (Klausur SS24-H, A3a; 5 P.)

Gegeben $G = (V, E)$ und $k \geq 1$. Entscheide: Gibt es in G *sowohl* ein Independent Set der Größe k *als auch* eine Clique der Größe k ? Zeigen Sie die NP-Schwere durch Reduktion eines bekannten NP-schweren Problems. Lösung: Anhang A.2.

Selbsttest: 3 – SubSet Sum mit Teilbarkeit (Klausur SS23-H, A5a; 4 P.)

Beweisen Sie die NP-Vollständigkeit von SUBSET SUM, bei dem jede Itemgröße durch 3 oder durch 7 teilbar ist. Lösung: Anhang A.3.

Selbsttest: 4 – HamiltonianPath $\leftrightarrow$ HamiltonianCycle (Klausur SS23-N, A5b/c; 5+6 P.)

Geben Sie polynomielle Reduktionen an: (b) HAMILTONIANPATH $\leq$ HAMILTONIANCYCLE; (c) HAMILTONIANCYCLE $\leq$ HAMILTONIANPATH. Lösung: Anhang A.4.

4 Approximation: anwenden, beweisen, Worst Case bauen

Dieser Aufgabenkomplex besteht in der Klausur aus zwei Aufgaben: einer *Anwendungsaufgabe* (Algorithmus auf konkrete Instanz, Aufgabe 1) und einem *Güte-Beweis mit Worst-Case-Konstruktion* (Aufgabe 4).

4.1 Anwenden: die Rechenaufgabe

Rezept: Was du flüssig rechnen können musst

- **LPT/List Scheduling:** Jobs (ggf. sortieren), der Reihe nach auf die Maschine mit kleinster Last; Schedule *grafisch* zeichnen (Maschinen-Balken mit Zeitachse); Güte nennen ($2 - \frac{1}{m}$ bzw. $\frac{4}{3} - \frac{1}{3m}$).
- **Δ TSP₁:** MST $\rightarrow$ Kanten verdoppeln $\rightarrow$ Eulerkreis (Startknoten beachten!) $\rightarrow$ Abkürzen; erklären, warum Güte 2 und wozu die Δ -Ungleichung.
- **Christofides:** MST $\rightarrow$ ungerad-gradige Knoten $X \rightarrow$ min. perfektes Matching auf $X \rightarrow$ Eulerkreis $\rightarrow$ Abkürzen. Alle Zwischengraphen angeben – auch Schritte nennen, die nichts ändern!
- **Knapsack-Greedy/MGA:** nach Profitdichte p_i/w_i sortieren, einpacken was passt; MGA: Vergleich mit bestem Einzel-Item; Güte 2 nennen; Verbesserungsidee: k -Enumeration (Sahni) $\rightarrow 3/2$ bei $k = 2$, $4/3$ bei $k = 3$.

Beispiel: Knapsack-Greedy durchgerechnet (Klausur SS23-H, A1)

Instanz (Items als (p_i, w_i) , Kapazität $B = 16$): $(1, 4), (1, 1), (1, 3), (11, 13), (3, 3), (5, 6), (1, 5)$.

GA: Profitdichten: $1/4, 1/1, 1/3, 11/13, 3/3, 5/6, 1/5 = 0,25; 1; 0,33; 0,85; 1; 0,83; 0,2$. Sortierung: $(1, 1), (3, 3), (11, 13), (5, 6), (1, 3), (1, 4), (1, 5)$. Einpacken: $(1, 1)$ [Last 1], $(3, 3)$ [4], $(11, 13)$ passt nicht ($17 > 16$), $(5, 6)$ [10], $(1, 3)$ [13], $(1, 4)$ passt nicht, $(1, 5)$ passt nicht. GA = $1 + 3 + 5 + 1 = 10$.

MGA: bestes Einzel-Item $(11, 13)$ mit Profit $11 > 10 \Rightarrow$ MGA = 11.

Vergleich: Optimal ist $\{(11, 13), (3, 3)\}$: Gewicht 16, Profit OPT = 14. GA verfehlt wegen des Dichte-Fokus das schwere profitable Item; MGA repariert das teilweise (Güte 2 garantiert).

Beispiel: Christofides durchgerechnet (Klausur SS23-N, A1)

Graph: Knoten a, b, c, d ; Distanzen $d(a, b) = 1$, $d(b, c) = 2$, $d(a, d) = 2$, $d(a, c) = 3$, $d(c, d) = 3$ (fehlende Paare über kürzeste Wege, z. B. $d(b, d) = 3$).

(1) **MST** (Kruskal): $\{a, b\}$ (1), $\{b, c\}$ (2), $\{a, d\}$ (2); Gewicht 5.

(2) **Ungerade Grade im MST**: a : Grad 2, b : Grad 2, c : Grad 1, d : Grad 1 $\Rightarrow X = \{c, d\}$.

(3) **Min. perfektes Matching auf X** : einzige Möglichkeit $K = \{\{c, d\}\}$, Kosten 3.

(4) **Multigraph $T+K$** : alle Grade gerade. **Eulerkreis** ab a : $[a, b, c, d, a]$, Kosten $1+2+3+2 = 8$.

(5) **Abkürzen**: kein Knoten doppelt – dieser Schritt ändert hier nichts (in der Klausur trotzdem erwähnen!). Tour $R = [a, b, c, d, a]$, Länge 8.

Hier gilt sogar $R = \text{OPT}$. Güte-Garantie: 1,5; benötigt Symmetrie und Dreiecksungleichung.

4.2 Güte-Beweise: die drei Beweismuster

Rezept: Muster A: OPT von unten durch Struktur abschätzen (Matching)

Für Minimierungsprobleme: Finde eine Struktur in der Algorithmus-Lösung, die *jede* Optimallösung zwingt, groß zu sein.

Musterbeweis 2ApproxVC (Präsenz 12.1): Der Algorithmus nimmt für jede noch unüberdeckte Kante *beide* Endpunkte. Sei A die Menge dieser Kanten; dann $|C| = 2|A|$. Die Kanten in A sind paarweise knotendisjunkt (ein *Matching*) – sonst wäre eine schon überdeckt gewesen. Jedes VC C^* braucht pro Kante aus A einen *eigenen* Knoten, also $|C^*| \geq |A|$. Zusammen: $|C| = 2|A| \leq 2|C^*|$.

Rezept: Muster B: Widerspruch + Zählen (MAX-3-SAT, HA 12.1 = Klausur SoSe23)

Algorithmus A : werte β_0 (alles false) und β_1 (alles true) aus, gib die bessere zurück. **Zu zeigen**: $v(A(\varphi)) \geq \frac{1}{2}v(\text{OPT}(\varphi))$.

Beweis (Widerspruch): Angenommen $v(A(\varphi)) < \frac{1}{2}v(\text{OPT}(\varphi))$ für ein φ mit m Klauseln. Wegen $v(\text{OPT}(\varphi)) \leq m$ folgt $v(\beta_0) < \frac{m}{2}$ und $v(\beta_1) < \frac{m}{2}$. Aber jede Klausel enthält mindestens ein positives oder ein negatives Literal und wird daher von β_0 oder β_1 erfüllt; also $v(\beta_0) + v(\beta_1) \geq m$, d. h. eine der beiden erfüllt $\geq \frac{m}{2}$ Klauseln – Widerspruch.

Scharfe Instanz (Teil b): $\varphi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_0 \vee \neg x_2 \vee \neg x_3)$. Mit $\beta = \{x_0 \mapsto \text{false}, x_1, x_2, x_3 \mapsto \text{true}\}$ sind beide Klauseln erfüllt: $\text{OPT} = 2$. Aber β_0 erfüllt nur Klausel 2, β_1 nur Klausel 1: $v(A(\varphi)) = 1 = \frac{1}{2}\text{OPT}$. *Bauprinzip: komplementäre Klauseln.*

Rezept: Muster C: Scheduling-Schranken (RoundRobin, Klausur SS24-N A4)

RoundRobin: Jobs absteigend sortiert, reihum verteilt (J_j auf Maschine $((j-1) \bmod m) + 1$).

Zu zeigen (per Widerspruch): Güte 2.

Angenommen $RR(I) > 2 \cdot OPT(I)$. Sei M_i die Maschine mit maximaler Last; sie trägt die Jobs $J_i, J_{i+m}, J_{i+2m}, \dots$. Für $l \geq 1$ gilt wegen der Sortierung $p_{i+lm} \leq p_j$ für alle $j \leq (l-1)m + m = lm$, insbesondere

$$p_{i+lm} \leq \frac{1}{m} \sum_{j=(l-1)m+1}^{lm} p_j.$$

Summieren über $l \geq 1$: $RR(I) - p_i \leq \frac{1}{m} \sum_j p_j \leq OPT(I)$. Mit $p_i \leq p_1 \leq OPT(I)$ folgt $RR(I) \leq 2 OPT(I)$ – Widerspruch zur Annahme.

Kern: dieselben zwei OPT-Schranken wie bei List Scheduling (Durchschnittslast, größter Job).

4.3 Worst-Case-Instanzen konstruieren

Der zweite Teil der Klausuraufgabe: „Geben Sie eine Konstruktionsvorschrift für eine Instanz an, mit der der Algorithmus beliebig nah an die Güte kommt.“

Rezept: Bauprinzip

1. Zwingen den Algorithmus zu einer lokal guten, global schlechten Entscheidung (Greedy-Falle) – oder nutze adversarielles Tie-Breaking (bei Gleichständen die schlechteste Wahl unterstellen).
2. Parametrisiere die Instanz $(n, m, T, \dots)$ und berechne $A(I)$ und $OPT(I)$ *explizit*.
3. Zeige $A(I)/OPT(I) \rightarrow$ Schranke (Grenzwert angeben).

Beispiel: List Scheduling erreicht $2 - \frac{1}{m}$

$m(m-1)$ Jobs der Größe 1, danach *ein* Job der Größe m (Liste in dieser Reihenfolge). List Scheduling verteilt die Einsen gleichmäßig (jede Maschine $m-1$), der große Job kommt obendrauf: $LS = (m-1) + m = 2m-1$. Optimal: großer Job allein auf eine Maschine, die $m(m-1)$ Einsen auf die übrigen $m-1$ Maschinen (m pro Maschine): $OPT = m$. Rate: $(2m-1)/m = 2 - \frac{1}{m}$ – exakt die Schranke.

Beispiel: RoundRobin erreicht $2 - \frac{1}{m}$ (Klausur SS24-N A4b, Bonus)

Ein Job der Größe m und $m(m-1)$ Jobs der Größe 1 (absteigend sortiert: der große zuerst). RoundRobin gibt Maschine 1 den großen Job *und* (da reihum verteilt wird) $m-1$ weitere Einsen-Jobs: $RR = m + (m-1) = 2m-1$; $OPT = m$ wie eben. Für $m=3$ (7 Jobs: 3, 1, 1, 1, 1, 1, 1): $RR = 5$, $OPT = 3$, Rate $5/3$ – genau die in der Klausur gestufte Teilpunkt-Konstruktion.

Beispiel: Christofides erreicht $3/2$ asymptotisch (Präsenz 12.2, Skizze)

Leiter-Graph mit n Knoten ($n \bmod 4 = 2$): Sprossen und Holme mit Distanz 1, Rest über kürzeste Wege. Adversariell gewählter Zickzack-MST (Kosten $n-1$) lässt nur die Knoten 1 und n mit ungeradem Grad; das Matching ist die einzelne Kante $\{1, n\}$ mit Kosten $n/2$ (kürzester Weg). Tour: $n-1 + n/2$, $OPT = n$ (außen herum). Rate $\frac{n-1+n/2}{n} = \frac{3}{2} - \frac{1}{n} \rightarrow \frac{3}{2}$. Merke: Bei MST-/Matching-Gleichständen darf der Prüfling die *schlechte* Wahl unterstellen – das ist der Sinn von „eine mögliche Ausführung“.

Selbsttest: 5 – ApproximateSubsetSum (Klausur SS24-H, A4; 3+7 P.)

Maximiere $\sum_{j \in S} a_j \leq T$ (alle $a_i \leq T$, $\sum a_i > T$). Algorithmus GA: sortiere absteigend, nimm Zahlen der Reihe nach, *stoppe* beim ersten Element, das nicht mehr passt. (a) Konstruktionsvorschrift für Instanzen, mit denen GA beliebig nah an Güte 2 kommt. (b) Zeigen Sie $GA(I) \geq OPT(I)/2$. Lösung: Anhang A.5.

Selbsttest: 6 – Min-Edge-Cover (Klausur SS23-N, A3; 6+4 P.)

Gesucht: kardinalitätsminimale Kantenmenge C , sodass jeder Knoten inzident zu einer Kante aus C ist (G zusammenhängend). Algorithmus A: gehe die Knoten in beliebiger Reihenfolge durch; ist der aktuelle Knoten v noch nicht abgedeckt, füge eine beliebige zu v inzidente Kante zu C hinzu. (a) Zeigen Sie $A(G) \leq 2 \cdot OPT(G)$. (b) Kreisgraph G_n ($n \geq 4$ gerade): Wie groß ist ein Min-Edge-Cover? Geben Sie Reihenfolge und Kantenwahl an, die zur schlechtesten Rate führen. Lösung: Anhang A.6.

5 ETH-Lower-Bounds: das Schema

Immer die letzte Klausuraufgabe: Eine Reduktion ist *vorgegeben*; du sollst untere Schranken bzgl. zweier Parameter herleiten. Das ist ein Schema-Spiel – wer das Muster kennt, sammelt hier die sichersten 10 Punkte der Klausur.

Rezept: ETH-Aufgabe in drei Schritten

Schritt 1 – Parameter abschätzen: Drücke jeden gefragten Parameter der konstruierten Instanz durch n (Variablen) und m (Klauseln) der 3-SAT-Quellinstanz aus. Nutze $n \leq 3m$ (jede Variable kommt in einer Klausel vor, jede Klausel hat ≤ 3 Literale).

Schritt 2 – Textbaustein (Kontraposition):

„Angenommen, es gäbe einen Algorithmus, der [Zielproblem] in $2^{o(p)} \cdot |I|^{O(1)}$ löst. Da $p = O(f(m))$ nach Schritt 1, existiert durch Kombination der Reduktion mit diesem Algorithmus ein Algorithmus, der 3-SAT in $2^{o(m)} \cdot |I|^{O(1)}$ löst. Nach dem Sparsification-Lemma geht das nur, wenn die ETH nicht gilt. Also existiert ein solcher Algorithmus nur, wenn die ETH falsch ist.“

Schritt 3 – Exponent justieren (Wurzel-Regel): Wächst der Parameter polynomiell, $p = O(m^c)$, dann ist die Schranke $2^{o(\sqrt[p]{p})}$: aus $q = o(\sqrt[p]{p})$ und $p = O(m^c)$ folgt $q = o(m)$. Beispiele: $|E| = O(m^2) \Rightarrow 2^{o(\sqrt{|E|})}$; $|T| = O(m^4) \Rightarrow 2^{o(\sqrt[4]{|T|})}$.

Achtung — typische Falle

- **n oder m ?** Hängt dein Parameter (auch) an m , brauchst du das *Sparsification-Lemma* und schreibst $2^{o(m)}$. Hängt er nur an n , reicht die ETH direkt ($2^{o(n)}$). Im Zweifel: Lemma zitieren schadet nicht.
- **Reduktionsketten:** Bei $X \leq Y \leq Z$ die Schranken schrittweise durchreichen und auf den vorigen Teil verweisen („nach Aufgabenteil 1 gibt es diesen nur, wenn die ETH nicht gilt“).
- Die Schlussformel „... nur, wenn die ETH falsch ist“ in *jedem* Teilpunkt wiederholen – sie ist Teil der erwarteten Lösung.

5.1 Die Vorlesungs-Reduktionskette mit allen Schranken (Präsenz 13.1)

Reduktion	Parameter der Zielinstanz	ergibt Lower Bounds
$3\text{-SAT} \leq k\text{-CLIQUE}$	$ V = O(m), E = O(m^2), k = m$	$2^{o(V)}, 2^{o(\sqrt{ E })}, 2^{o(k)}$
$k\text{-CLIQUE} \leq k\text{-IS}$	$ V' = V , E' \leq V ^2$	$2^{o(V)}, 2^{o(\sqrt{ E })}$
$\text{SAT} \leq 3\text{-DM}$	$ U = V = W = O(mn) \subseteq O(m^2), T = O(m^2 n^2) \subseteq O(m^4)$	$2^{o(\sqrt{ V })}, 2^{o(\sqrt[4]{ T })}$
$3\text{-DM} \leq 3\text{-EC}$	Spezialfall: $ U = O(V), F = T $	$2^{o(\sqrt{ U })}, 2^{o(\sqrt[4]{ F })}$
$3\text{-EC} \leq \text{SUBSET SUM}$	$n_{\text{Items}} = F $	$2^{o(\sqrt[4]{n})}$

(Über die *strenge* Wegener-Reduktion $3\text{-SAT} \leq \text{SUBSET SUM}$ mit $|A| = O(m)$ Items gilt sogar: kein $2^{o(n)} \cdot \text{poly}$ für SubSet Sum und Partition; Skript Satz 6.40.)

5.2 Musterlösung 1: Lower Bounds für Vertex Cover (HA 13.1)

Reduktion: $\text{CLIQUE} \leq \text{VC}$ aus Präsenz 10 (Komplementgraph, $k' = n - k$).

Teil 1 – Parameter: $|V'| = |V|$; $|E'| = \binom{|V|}{2} - |E| \leq |V|^2$; $k' = |V| - k \leq |V|$.

Teil 2 – Schranken (jeweils mit dem Textbaustein):

- Kein $2^{o(|V'|)} \cdot |I|^{O(1)}$: Sonst gäbe es (Reduktion + Algorithmus, $|V'| = |V|$) einen $2^{o(|V|)}$ -Algorithmus für CLIQUE – nach Präsenz 13 nur möglich, wenn die ETH falsch ist.
- Kein $2^{o(\sqrt{|E'|})} \cdot |I|^{O(1)}$: Wegen $|E'| \leq |V|^2$ ist $\sqrt{|E'|} \leq |V|$, also ergäbe sich ein $2^{o(|V|)}$ -Algorithmus für CLIQUE – ETH falsch.
- Kein $2^{o(k')} \cdot |I|^{O(1)}$: Wegen $k' \leq |V|$ analog.

Punkteschema: Teil 1: 1 P.; Teil 2: 1,5 + 1,5 + 1 P.

5.3 Musterlösung 2: Lower Bounds für Hitting Set (HA 13.2)

Gegebene Reduktion $3\text{-SAT} \leq \text{HITTING SET}$: $U = \{x_1, \bar{x}_1, \dots, x_n, \bar{x}_n\}$; pro Variable $F_i = \{x_i, \bar{x}_i\}$; pro Klausel $F_{n+j} = C_j$; $k = n$.

Teil 1 – Parameter: $|U| = 2n$; $r = n + m$; $k = n$.

Teil 2 – Schranken:

- Kein $2^{o(|U|)}$: $|U| = 2n$, also ergäbe sich ein $2^{o(n)}$ -Algorithmus für 3-SAT – direkt ein Widerspruch zur ETH (nur n beteiligt, Lemma nicht nötig).
- Kein $2^{o(r)}$: $r = n + m \leq 3m + m = 4m = O(m)$, also ergäbe sich ein $2^{o(m)}$ -Algorithmus für 3-SAT – **hier** braucht es das Sparsification-Lemma.
- Kein $2^{o(k)}$: $k = n$, analog zum ersten Punkt.

Selbsttest: 7 – Lower Bounds für k -Color (Klausur SS24-H, A5a; 5 P.)

Gegeben ist die Vorlesungsreduktion $3\text{-SAT} \leq k\text{-COLOR}$ (Knoten $x_i, \bar{x}_i, v_i$ für jede Variable, C_j für jede Klausel, Zentrum z ; Kanten wie im Skript). Welche Lower Bounds ergeben sich unter der ETH bzgl. (i) $|V|$ und (ii) $|E|$? Lösung: Anhang A.7.

Selbsttest: 8 – Lower Bounds für Set Cover (Klausur SS23-N, A6a; 5 P.)

Bekannt: HITTING SET ist unter der ETH weder in $2^{o(r')} \langle I \rangle^{O(1)}$ noch in $2^{o(|U'|)} \langle I \rangle^{O(1)}$ lösbar. Gegeben ist die Dualitäts-Reduktion $\text{SET COVER} \leq \text{HITTING SET}$: $U' := \{1, \dots, r\}$ (Mengen-Indizes werden Elemente), für jedes $w \in U$ die Menge $F'_w := \{v \in \{1, \dots, r\} \mid w \in F_v\}$, $k' = k$. Welche Schranken folgen für SET COVER bzgl. (i) $|U|$ und (ii) r ? Lösung: Anhang A.8.

6 Kurz: NP-Schwere via Unentscheidbarkeit (Halteproblem)

Serienstoff (HA 11.2), in den vier Altklausuren nie geprüft – der Beweis ist kurz, nimm ihn mit.

Beweis (HALT_{TM} ist NP-schwer). Wir zeigen $\text{SAT} \leq \text{HALT}_{\text{TM}}$. Aus einer KNF-Formel φ konstruiere eine DTM M_φ : (1) ignoriere die Eingabe, (2) enumeriere systematisch alle Belegungen der Variablen von φ , (3) prüfe jede, (4) *halte*, sobald eine erfüllende gefunden ist, (5) sonst laufe für immer. Setze $f(\varphi) = \langle M_\varphi, \varepsilon \rangle$.

Pointe: M_φ läuft nicht polynomiell – aber sie lässt sich in Polynomialzeit *konstruieren*; nur das zählt für f .

$\Rightarrow$: Ist φ erfüllbar, findet M_φ eine erfüllende Belegung und hält. $\Leftarrow$: Hält M_φ , so nur, weil eine erfüllende Belegung gefunden wurde; also $\varphi \in \text{SAT}$. $\square$

Merke

HALT_{TM} ist NP-schwer, aber nicht NP-vollständig: Es liegt nicht in NP (nicht einmal entscheidbar). Typische Transferfrage im Stil von Präsenz 10.1 (ii).

7 Klausur-Fahrplan und Checklisten

Rezept: Zeitbudget (120 Minuten, ~55 Punkte)

Grob 2 Minuten pro Punkt: Anwendungsaufgabe ~20 min, Vorlesungsbeweis ~20 min, Reduktionen 2×12 min, Güte-Beweis ~20 min, ETH ~15 min, Puffer ~20 min. Beginne mit deinen sichersten Typen – ETH und Vorlesungsbeweis sind am besten vorhersagbar. Notenschlüssel der Altklausuren: 4,0 ab ~17,5 P., 1,0 ab ~35 P. – du musst nicht alles schaffen.

Rezept: Was auf das handbeschriebene Blatt gehört

- Die vier Vorlesungsbeweise (Kapitel 2), mindestens als Stichpunkt-Gerüst.
- Boilerplate: 6-Schritte-Reduktionsschema + Schlusssatz (Kapitel 3).
- ETH-Textbaustein + Wurzel-Regel + $n \leq 3m$ (Kapitel 5).
- Gadget-Katalog und SubSet-Sum-Werkzeugkasten.
- Die zwei OPT-Schranken fürs Scheduling; MAX-3-SAT-Formel; LS-Worst-Case-Instanz.
- Dualitätsdreieck Clique/IS/VC; NP-Dreischritt.

Rezept: Letzte Kontrolle bei jeder Beweisaufgabe

- Reduktionsrichtung richtig? ($X_{\text{bekannt}} \leq Y_{\text{neu}}$)
- Beide Korrektheitsrichtungen da?
- Polynomialität (Reduktion bzw. Verifizierer) explizit erwähnt?
- Parameter ($k, T, \dots$) in der Konstruktion angepasst?
- Schlusssatz geschrieben?

A Lösungen zu den Selbsttests

A.1 Lösung zu Selbsttest 1: Dominating Set

(a) **Zertifikat:** Eine Teilmenge $D \subseteq V$ (ein Bit pro Knoten, Länge $O(|V|)$) – die behauptete dominierende Menge.

(b) **Verifizierer:** Prüfe (1) $|D| \leq k$: Bits zählen, $O(|V|)$. (2) Dominanz: Für jeden Knoten $v \in V$ teste, ob $v \in D$ oder ein Nachbar von v in D liegt – pro Knoten Durchlauf seiner Zeile in der Adjazenzmatrix, $O(|V|)$; gesamt $O(|V|^2)$. Akzeptiere genau dann, wenn beides gilt. Laufzeit insgesamt $O(|V|^2)$, polynomiell. *Korrektheit:* Existiert ein Dominating Set der Größe $\leq k$, so wird das zugehörige Zertifikat akzeptiert; akzeptiert der Verifizierer ein D , erfüllt D per Konstruktion beide definierenden Eigenschaften.

(c) **NTM:** Rate für jeden Knoten nichtdeterministisch, ob er in D liegt ($O(|V|)$ Rateschritte). Prüfe danach deterministisch $|D| \leq k$ und die Dominanzbedingung wie in (b). Laufzeit: $O(|V|)$ Raten + $O(|V|^2)$ Prüfen = $O(|V|^2)$, polynomiell. *Korrektheit:* Ja-Instanz $\Rightarrow$ es existiert ein Rechenweg, der ein gültiges D rät und akzeptiert; Nein-Instanz $\Rightarrow$ jeder Rechenweg verwirft in der Prüfphase.

A.2 Lösung zu Selbsttest 2: CliqueAndIndependentSet

Reduktion von CLIQUE. Sei $(G = (V, E), k)$ gegeben (o.B.d.A. $k \geq 1$). Konstruiere G' , indem zu G genau k neue isolierte Knoten $u_1, \dots, u_k$ hinzugefügt werden; Ausgabe (G', k) . Laufzeit $O(k) \subseteq O(|V|)$, polynomiell.

$\Rightarrow$: Hat G eine Clique C mit $|C| \geq k$, so hat G' dieselbe Clique; außerdem ist $\{u_1, \dots, u_k\}$ ein Independent Set der Größe k (isolierte Knoten). Also Ja-Instanz.

$\Leftarrow$: Hat G' eine Clique C' der Größe k , so gilt für $k \geq 2$: C' enthält keinen der isolierten Knoten (die haben keine Kanten), also $C' \subseteq V$ und C' ist eine k -Clique in G . (Für $k = 1$ ist jede Instanz mit $V \neq \emptyset$ trivial eine Ja-Instanz beider Probleme.)

Da CLIQUE NP-schwer ist, ist CLIQUEANDINDEPENDENTSET NP-schwer. *Trick:* Das Independent Set wird „gratis“ mitgeliefert, ohne die Clique-Frage zu beeinflussen.

A.3 Lösung zu Selbsttest 3: SubSet Sum mit Teilbarkeit

$\in$ NP: Verifizierer wie bei SUBSET SUM (Teilmenge als Zertifikat, Summe prüfen) – nur die Instanzmenge ist eingeschränkt.

Reduktion von SUBSET SUM: Gegeben $(c_1, \dots, c_n, K)$. Setze $c'_i := 3c_i$ und $K' := 3K$. Jede Itemgröße ist durch 3 teilbar, also ist die konstruierte Instanz gültig; Laufzeit $O(n)$.

Korrektheit: Für jede Teilmenge S gilt $\sum_{i \in S} c'_i = 3 \sum_{i \in S} c_i$, also $\sum_{i \in S} c'_i = K' \iff \sum_{i \in S} c_i = K$ – beide Richtungen in einem Schritt. Mit dem Schlusssatz folgt die NP-Vollständigkeit.

A.4 Lösung zu Selbsttest 4: HamiltonianPath $\leftrightarrow$ HamiltonianCycle

(b) **HP $\leq$ HC:** Gegeben $G = (V, E)$. Füge einen neuen *universellen* Knoten u hinzu: $G' = (V \cup \{u\}, E \cup \{\{u, v\} \mid v \in V\})$. Laufzeit $O(|V|)$. $\Rightarrow$: Ein Hamiltonpfad $v_1, \dots, v_n$ in G wird durch u zum Kreis $v_1, \dots, v_n, u, v_1$ geschlossen. $\Leftarrow$: Ein Hamiltonkreis in G' besucht u genau einmal; Entfernen von u (und seiner zwei Kreiskanten) liefert einen Hamiltonpfad in G .

(c) **HC $\leq$ HP:** Gegeben $G = (V, E)$, o.B.d.A. $|V| \geq 3$. Wähle einen beliebigen Knoten v . Konstruiere G' : füge eine Kopie v^* mit derselben Nachbarschaft wie v hinzu sowie zwei neue Grad-1-Knoten s (nur mit v verbunden) und t (nur mit v^* verbunden). Laufzeit $O(|V| + |E|)$. $\Rightarrow$: Ein Hamiltonkreis $v, w_1, w_2, \dots, w_{n-1}, v$ in G liefert den Hamiltonpfad $s, v, w_1, \dots, w_{n-1}, v^*, t$ in G' (die letzte Kreiskante $\{w_{n-1}, v\}$ existiert auch zu v^*). $\Leftarrow$: Ein Hamiltonpfad in G' muss in den Grad-1-Knoten s und t enden, hat also die Form $s, v, \dots, v^*, t$. Der Mittelteil $v, \dots, v^*$ besucht alle Knoten von G (plus v^*); Ersetzen von v^* durch v am Ende schließt einen Hamiltonkreis in G (Nachbarschaften identisch).

A.5 Lösung zu Selbsttest 5: ApproximateSubsetSum

(a) **Konstruktionsvorschrift:** Für gerades T nimm $a_1 = T/2 + 1$, $a_2 = a_3 = T/2$. GA sortiert absteigend, nimmt a_1 , und stoppt: a_2 passt nicht mehr ($T/2 + 1 + T/2 > T$). $GA(I) = T/2 + 1$,

aber $\text{OPT}(I) = T$ (Items 2 und 3). Rate: $\frac{T}{T/2+1} \rightarrow 2$ für $T \rightarrow \infty$.

(b) Güte 2: Sei S die von GA gewählte Menge und a_ℓ das erste Element, das nicht mehr passte (existiert wegen $\sum_i a_i > T$; falls GA alle Elemente bis zum Abbruch einpackt, betrachte den Abbruchmoment). Dann gilt

$$\text{GA}(I) + a_\ell > T \geq \text{OPT}(I).$$

Wegen der absteigenden Sortierung wurde a_ℓ erst nach allen gewählten Elementen betrachtet. Ist $S = \emptyset$, wäre schon $a_1 = a_\ell > T$ – ausgeschlossen, da $a_i \leq T$. Also enthält S ein Element $a_j \geq a_\ell$ (Sortierung), somit $\text{GA}(I) \geq a_\ell$. Einsetzen:

$$2 \text{GA}(I) \geq \text{GA}(I) + a_\ell > T \geq \text{OPT}(I),$$

also $\text{GA}(I) \geq \text{OPT}(I)/2$. □

A.6 Lösung zu Selbsttest 6: Min-Edge-Cover

(a) Güte 2: Jede von A gewählte Kante wird für einen zu diesem Zeitpunkt *unabgedeckten* Knoten v gewählt; danach ist v abgedeckt. Verschiedene Auswahlsschritte gehören also zu verschiedenen Knoten: $A(G) = |C| \leq |V| \dots$ genauer: $|C| \leq \text{Anzahl der Auswahlsschritte} \leq |V|$. Andererseits deckt jede Kante höchstens 2 Knoten ab, also braucht jede Lösung $\text{OPT}(G) \geq |V|/2$ Kanten. Zusammen: $A(G) \leq |V| \leq 2 \cdot \text{OPT}(G)$. □

(b) Kreisgraph G_n : Ein minimales Edge Cover ist ein perfektes Matching des Kreises: $\text{OPT}(G_n) = n/2$ (jede Kante deckt 2 Knoten, n Knoten). *Schlechteste Ausführung:* Besuche die Knoten in der Reihenfolge $1, 3, 4, 5, \dots, n$ und wähle: für 1 die Kante $\{1, 2\}$, für 3 die Kante $\{2, 3\}$, für 4 die Kante $\{3, 4\}$, $\dots$, für n die Kante $\{n-1, n\}$ – jede neue Kante deckt nur *einen* neuen Knoten ab (der andere Endpunkt war schon abgedeckt). Das ergibt $|C| = n-1$ Kanten. Rate:

$$\frac{n-1}{n/2} = 2 - \frac{2}{n} \rightarrow 2.$$

A.7 Lösung zu Selbsttest 7: Lower Bounds für k -Color

Parameter abschätzen: Die Konstruktion erzeugt

$$|V| = \underbrace{3n}_{x_i, \bar{x}_i, v_i} + \underbrace{m}_{C_j} + \underbrace{1}_z = O(n+m) \subseteq O(m) \quad (\text{da } n \leq 3m).$$

Kanten: $\{v_i, v_j\}$: $O(n^2)$; $\{v_i, x_j\}, \{v_i, \bar{x}_j\}$: $O(n^2)$; $\{x_i, \bar{x}_i\}$: n ; $\{x_i, C_j\}, \{\bar{x}_i, C_j\}$: $O(nm)$; $\{v_i, z\}, \{C_j, z\}$: $O(n+m)$. Gesamt $|E| = O(n^2 + nm + m) \subseteq O(m^2)$.

(i) Kein $2^{o(|V|)} \cdot |I|^{O(1)}$ für k -COLOR: Sonst lieferte die Reduktion (mit $|V| = O(m)$) einen $2^{o(m)}$ -Algorithmus für 3-SAT – nach dem Sparsification-Lemma nur möglich, wenn die ETH falsch ist.

(ii) Kein $2^{o(\sqrt{|E|})} \cdot |I|^{O(1)}$: Wegen $|E| = O(m^2)$ gilt $\sqrt{|E|} = O(m)$; ein solcher Algorithmus ergäbe wieder $2^{o(m)}$ für 3-SAT – ETH falsch (Wurzel-Regel).

A.8 Lösung zu Selbsttest 8: Lower Bounds für Set Cover

Parameter der konstruierten Hitting-Set-Instanz: $|U'| = r$ (die Mengen-Indizes) und $r' = |U|$ (eine Menge F'_w pro Element $w \in U$); $k' = k$. Die Reduktion vertauscht also die Rollen von Elementen und Mengen (Dualität über den bipartiten Inzidenzgraphen) und ist in polynomieller Zeit berechenbar. Entscheidend: Sie ist eine *Reduktion von Set Cover auf Hitting Set* – ein schneller Set-Cover-Algorithmus hilft so nicht direkt. Die Schranken erhält man über

die *umgekehrte* Anwendung: Dieselbe Konstruktion, auf eine HITTING-SET-Instanz angewandt, liefert eine SET-COVER-Instanz (die Dualisierung ist involutorisch: zweimal anwenden ergibt die Ausgangsinstanz). Also gilt HITTING SET $\leq$ SET COVER mit $|U| = r'$ und $r = |U'|$.

(i) Kein $2^{o(|U|)} \cdot \langle I \rangle^{O(1)}$ für SET COVER: Sonst könnte man eine HITTING-SET-Instanz dualisieren ($|U| = r'$) und erhielte einen $2^{o(r')}$ -Algorithmus für HITTING SET – nach Voraussetzung nur möglich, wenn die ETH falsch ist.

(ii) Kein $2^{o(r)} \cdot \langle I \rangle^{O(1)}$: analog mit $r = |U'|$ – ein solcher Algorithmus ergäbe $2^{o(|U'|)}$ für HITTING SET, also ETH falsch.