

# AAK – Fundament

Komplexitätstheorie und Approximation ohne Beweise  
Grundwissen zu Serie 9–13 und Skript Kapitel 6–7

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

Sommersemester 2026

**Wie du dieses Dokument benutzt.** Dieses PDF baut das Fundament auf: alle Begriffe, Probleme und Resultate des Klausurstoffs – bewusst *ohne* Beweise. Zu jedem Satz steht höchstens eine einzeilige Beweisidee. Arbeite es von vorne nach hinten durch. Wenn alles sitzt, nimm dir **Uebung.pdf** vor: Dort lernst du die Beweistechniken und trainierst die konkreten Klausuraufgabentypen.

**Klausurstoff:** Serien 9–13 und Skript Kapitel 6 (Komplexitätstheorie) und 7 (Approximative Algorithmen). Mehr kommt nicht dran.

## Inhaltsverzeichnis

|          |                                                            |           |
|----------|------------------------------------------------------------|-----------|
| <b>1</b> | <b>Grundbegriffe: Sprachen und Entscheidungsprobleme</b>   | <b>3</b>  |
| <b>2</b> | <b>Die Klassen P und NP</b>                                | <b>3</b>  |
| 2.1      | P: effizient lösbar . . . . .                              | 3         |
| 2.2      | NP: effizient überprüfbar . . . . .                        | 4         |
| <b>3</b> | <b>Reduktionen, NP-Schwere, NP-Vollständigkeit</b>         | <b>5</b>  |
| 3.1      | Polynomielle Reduktionen . . . . .                         | 5         |
| 3.2      | NP-schwer und NP-vollständig . . . . .                     | 6         |
| 3.3      | Der Startpunkt: Satz von Cook–Levin . . . . .              | 7         |
| <b>4</b> | <b>Problemkatalog</b>                                      | <b>7</b>  |
| 4.1      | Logik-Probleme . . . . .                                   | 7         |
| 4.2      | Graphprobleme . . . . .                                    | 7         |
| 4.3      | Mengen- und Zahlprobleme . . . . .                         | 10        |
| 4.4      | Scheduling . . . . .                                       | 11        |
| <b>5</b> | <b>Die Reduktionslandkarte</b>                             | <b>11</b> |
| <b>6</b> | <b>ETH: Exponentialzeit-Hypothese und untere Schranken</b> | <b>11</b> |
| <b>7</b> | <b>Approximative Algorithmen</b>                           | <b>12</b> |
| 7.1      | Gütebegriff und Approximationsschemata . . . . .           | 12        |
| 7.2      | TSP: $\Delta\text{TSP}_1$ und Christofides . . . . .       | 13        |
| 7.3      | Knapsack: Greedy, MGA, FPTAS . . . . .                     | 14        |
| 7.4      | Scheduling: List Scheduling und LPT . . . . .              | 15        |
| 7.5      | MAX-3-SAT: Güte 2 . . . . .                                | 15        |
| <b>8</b> | <b>Halteproblem und Unentscheidbarkeit</b>                 | <b>15</b> |



# 1 Grundbegriffe: Sprachen und Entscheidungsprobleme

Die Komplexitätstheorie klassifiziert, *wie schwer* Probleme sind. Ausgangspunkt: Für viele Probleme (z. B. Clique) kennt man nur Algorithmen mit exponentieller Laufzeit. Der naive Clique-Algorithmus (teste alle  $k$ -elementigen Teilmengen) braucht im schlimmsten Fall mindestens  $2^{|V|/2}$  Schritte (Lemma 6.2 im Skript). Die Frage „Geht es schneller?“ führt auf P, NP und die NP-Vollständigkeit.

## Definition 1.1 (Alphabet, Wort, Sprache)

- Ein *Alphabet*  $\Sigma$  ist eine endliche Menge von Symbolen.
- $\Sigma^*$  ist die Menge aller endlichen Wörter über  $\Sigma$ ;  $|x|$  ist die Länge von  $x$ .
- Eine Teilmenge  $L \subseteq \Sigma^*$  heißt *Sprache*.

## Definition 1.2 (Entscheidungsproblem)

Ein *Entscheidungsproblem* ist ein Tripel  $(L, U, \Sigma)$  mit Sprachen  $L \subseteq U \subseteq \Sigma^*$ . Ein Algorithmus  $A$  *entscheidet*  $L$  (bzgl.  $U$ ), wenn für alle  $x \in U$  gilt:  $x \in L \Rightarrow A(x) = 1$  und  $x \notin L \Rightarrow A(x) = 0$ . Meist ist  $U = \Sigma^*$ . Die Antwort ist immer *Ja* oder *Nein*.

Jedes konkrete Problem wird als Sprache kodiert. Beispiel Cliquesproblem:

$$\text{CLIQUE} = \{ u\#v \mid u \text{ kodiert Adjazenzmatrix von } G, v \text{ kodiert } k, \\ G \text{ hat Clique mit } k \text{ Knoten} \}.$$

Dabei hat  $u$  Länge  $O(|V|^2)$  und  $v$  Länge  $O(\log |V|)$  – Zahlen werden *binär* kodiert, sind also exponentiell größer als ihre Kodierungslänge. Das wird bei SubSet Sum wichtig.

## Definition 1.3 (Laufzeit)

$T_A(x)$  ist die Anzahl der Operationen von Algorithmus  $A$  auf Eingabe  $x$ . Die Worst-Case-Laufzeit für Eingabelänge  $n$  ist  $T_A(n) = \max\{ T_A(x) \mid x \in \Sigma^*, |x| = n \}$ .

## Merke

Klausuraufgaben arbeiten fast immer mit *Entscheidungsvarianten*: Aus einem Optimierungsproblem („finde minimales Vertex Cover“) wird eine Ja/Nein-Frage mit Schranke („gibt es ein Vertex Cover mit  $\leq k$  Knoten?“).

# 2 Die Klassen P und NP

## 2.1 P: effizient lösbar

### Definition 2.1 (Klasse P)

P ist die Menge aller Sprachen, die ein (deterministischer) Algorithmus in polynomieller Zeit entscheidet:

$$P = \{ L \subseteq \Sigma^* \mid \exists \text{ Algorithmus } A, T_A(n) = O(n^d) \text{ mit } d = O(1), A \text{ entscheidet } L \}.$$

Intuition: P = „effizient lösbar“. Beispiele: Sortieren, kürzeste Wege, minimale Spannbäume, Matching.

## 2.2 NP: effizient überprüfbar

Für NP gibt es **zwei gleichwertige Definitionen**. Beide musst du kennen – die Klausur SS23 hat genau diesen Dreischritt (Zertifikat, Verifizierer, NTM) abgefragt.

### Weg 1: Verifizierer und Zertifikat.

#### Definition 2.2 (Verifizierer, Zertifikat)

Ein Algorithmus  $A$  auf  $\Sigma^* \times \Sigma^*$  heißt *Verifizierer* für  $L$ , wenn

$$L = \{x \in \Sigma^* \mid \exists c \in \Sigma^* \text{ mit } A(x, c) = 1\}.$$

Ein Wort  $c$  mit  $A(x, c) = 1$  heißt *Zertifikat* für  $x \in L$ .  $A$  ist ein *polynomieller* Verifizierer, wenn es für jedes  $x \in L$  ein Zertifikat  $c$  gibt mit  $T_A(x, c) = O(|x|^d)$  für eine Konstante  $d$ .

#### Definition 2.3 (Klasse NP)

$\text{NP} = \{L \subseteq \Sigma^* \mid \text{es existiert ein polynomieller Verifizierer für } L\}.$

Intuition: Eine Lösung zu *finden* mag schwer sein – eine vorgelegte Lösung zu *prüfen* ist leicht. Das Zertifikat ist die vorgelegte Lösung.

#### Beispiel: Typische Zertifikate

| Problem       | Zertifikat                  | Prüfung                              |
|---------------|-----------------------------|--------------------------------------|
| $k$ -CLIQUE   | Knotenmenge $C \subseteq V$ | alle Paare in $E$ ? $ C  \geq k$ ?   |
| SAT           | Belegung $\psi$             | wertet $\alpha$ zu true aus?         |
| VERTEX COVER  | Knotenmenge $C$             | jede Kante überdeckt? $ C  \leq k$ ? |
| SUBSET SUM    | Teilmenge $S$               | $\sum_{j \in S} c_j = K$ ?           |
| HAMILTONKREIS | Permutation $\pi$           | alle Folgekanten in $E$ ?            |

Das Zertifikat ist polynomiell lang, die Prüfung läuft in Polynomialzeit.

### Weg 2: Nichtdeterminismus.

Ein *nichtdeterministischer Algorithmus* darf in jedem Schritt „raten“ (mehrere Rechenwege). Er entscheidet  $L$ , wenn (i) jedes  $w \in L$  mindestens einen akzeptierenden Rechenweg hat, (ii) jeder akzeptierende Rechenweg polynomiell lang ist und (iii) für  $w \notin L$  jeder Rechenweg ablehnt. Muster: *Rate die Lösung, prüfe sie deterministisch.*

### Definition 2.4 (Nichtdeterministische Turingmaschine (NDTM))

$M = (Q, \Gamma, q_0, \delta, F)$  mit endlicher Zustandsmenge  $Q$ , Arbeitsalphabet  $\Gamma \supset \Sigma$ , Startzustand  $q_0$ , Endzuständen  $F \subset Q$  und Übergangsfunktion

$$\delta : Q \times \Gamma \longrightarrow 2^{Q \times \Gamma \times \{-1, 0, 1\}}.$$

Zu  $(q, a)$  kann es also *mehrere* (oder keine) Übergänge geben. Eine *Konfiguration*  $uqav$  speichert Bandinschrift, Zustand  $q$  und Arbeitsfeld  $a$ ;  $C \vdash C'$  bezeichnet den Übergang zur Folgekonfiguration. Eine Berechnung  $C_0, \dots, C_k$  *akzeptiert*, wenn  $C_k$  eine Stopkonfiguration mit Zustand in  $F$  ist.  $L \in \text{NP} \iff$  eine NDTM akzeptiert  $L$  in nichtdeterministischer polynomieller Zeit. Ist  $|\delta(q, a)| \leq 1$  überall, heißt  $M$  *deterministisch* (DTM);  $L \in \text{P} \iff$  eine DTM entscheidet  $L$  in polynomieller Zeit.

### Satz 2.5 (Äquivalenz der NP-Definitionen)

Die Definition über Verifizierer und die über NDTMs beschreiben dieselbe Klasse NP.

*Beweisidee:* Die Folge der Rateschritte einer NDTM ist ein Zertifikat; umgekehrt rät die NDTM das Zertifikat (Phase 1) und simuliert den Verifizierer (Phase 2).

### Satz 2.6 ( $\text{P} \subseteq \text{NP}$ )

Jede Sprache in P liegt auch in NP.

*Beweisidee:* Der entscheidende Algorithmus ist ein Verifizierer, der das Zertifikat einfach ignoriert.

### Merke

Ob  $\text{P} = \text{NP}$  gilt, ist **offen** – eines der sieben Millennium-Probleme (Clay Institute, 1 Mio. Dollar). Alles Weitere (NP-Vollständigkeit, ETH) ist der Umgang mit dieser Unwissenheit.

## 3 Reduktionen, NP-Schwere, NP-Vollständigkeit

### 3.1 Polynomielle Reduktionen

#### Definition 3.1 (Polynomielle Reduktion $L_1 \leq L_2$ )

Seien  $L_1 \subseteq \Sigma_1^*$ ,  $L_2 \subseteq \Sigma_2^*$ . Eine Funktion  $f : \Sigma_1^* \rightarrow \Sigma_2^*$  mit

$$w \in L_1 \iff f(w) \in L_2 \quad \forall w \in \Sigma_1^*$$

heißt *Transformation* von  $L_1$  auf  $L_2$ . Sie heißt *polynomiell*, wenn ein Algorithmus  $f(w)$  in polynomieller Zeit berechnet (dann ist auch  $|f(w)| \leq \text{poly}(|w|)$ ). Notation:  $L_1 \leq L_2$  („ $L_1$  ist polynomiell reduzierbar auf  $L_2$ “).

Intuition:  $L_1 \leq L_2$  heißt „ $L_2$  ist mindestens so schwer wie  $L_1$ “ – wer  $L_2$  lösen kann, kann damit auch  $L_1$  lösen.

### Achtung — typische Falle

**Die Richtung der Reduktion** ist der häufigste Fehler. Um zu zeigen, dass ein *neues* Problem  $Y$  schwer ist, reduzierst du ein *bekanntes* schweres Problem  $X$  auf  $Y$ : also  $X \leq Y$  („bekannt  $\leq$  neu“). Die Reduktion nimmt eine  $X$ -Instanz und baut daraus eine  $Y$ -Instanz – niemals umgekehrt.

### Satz 3.2 (Transitivität)

$L_1 \leq L_2$  und  $L_2 \leq L_3 \Rightarrow L_1 \leq L_3$ .

*Beweisidee:* Reduktionen hintereinanderschalten; die Ausgabe der ersten ist polynomiell groß, also bleibt die Komposition polynomiell.

## 3.2 NP-schwer und NP-vollständig

### Definition 3.3 (NP-schwer, NP-vollständig)

- $L_0$  heißt *NP-schwer*  $\iff \forall L \in \text{NP} : L \leq L_0$ .
- $L_0$  heißt *NP-vollständig*  $\iff L_0 \in \text{NP}$  und  $L_0$  ist NP-schwer.

### Merke

Eselsbrücke: *NP-schwer* = untere Schranke (mindestens so schwer wie alles in NP).  $\in \text{NP}$  = obere Schranke (nicht schwerer als NP). *NP-vollständig* = beides. Es gibt NP-schwere Probleme, die nicht in NP liegen – z.B. das Halteproblem (Abschnitt 8).

### Satz 3.4 (NP-vollständig und $P=NP$ )

Sei  $L_0$  NP-vollständig. Dann gilt:  $P = NP \iff L_0 \in P$ .

*Beweisidee* ( $\Leftarrow$ ): Für beliebiges  $L \in \text{NP}$  schalte die Reduktion  $L \leq L_0$  vor den Polynomialzeit-Algorithmus für  $L_0$ .

Konsequenz: Findet jemand für *ein* NP-vollständiges Problem einen Polynomialzeit-Algorithmus, kollabiert alles:  $P = NP$ . Solange das nicht passiert, gilt ein NP-Vollständigkeitsbeweis als starkes Indiz, dass es keinen effizienten Algorithmus gibt.

### Satz 3.5 (Vererbungskorollar – das wichtigste Werkzeug)

Ist  $L_0$  NP-vollständig,  $L_0 \leq L_1$  und  $L_1 \in \text{NP}$ , so ist  $L_1$  NP-vollständig.

*Beweisidee:*  $L \leq L_0 \leq L_1$  für alle  $L \in \text{NP}$  (Transitivität).

Dieses Korollar ist das Standardwerkzeug der Klausur: Statt „alle  $L \in \text{NP}$ “ zu betrachten, reicht *eine* Reduktion von einem bekannten NP-vollständigen Problem plus der Nachweis  $L_1 \in \text{NP}$ .

### 3.3 Der Startpunkt: Satz von Cook–Levin

#### Satz 3.6 (Cook (1971) / Levin (1973))

SAT ist NP-vollständig.

*Beweisidee:* Für beliebiges  $L \in \text{NP}$  mit NDTM  $M$  wird die Berechnung von  $M$  auf  $u$  als KNF-Formel  $\alpha_u$  kodiert:  $\alpha_u = \alpha_{\text{Anfang}} \wedge \alpha_{\text{Ende}} \wedge \alpha_{\text{Eindeutig}} \wedge \alpha_{\text{Übergang}}$  mit Variablen für „Zustand zur Zeit  $t$ “, „Arbeitsfeld zur Zeit  $t$ “, „Bandinhalt Feld  $i$  zur Zeit  $t$ “. Erfüllende Belegungen entsprechen genau akzeptierenden Berechnungen; die Formel hat polynomielle Größe  $O(T(n)^3 \log T(n))$ .

Ab hier werden alle weiteren NP-Vollständigkeitsbeweise per Reduktionskette aus SAT aufgebaut (Kapitel 5).

## 4 Problemerkatalog

Alle Probleme des Klausurstoffs im Steckbrief-Format. Präge dir zu jedem ein: Eingabe, Frage, Status, und *woher* die NP-Vollständigkeit kommt.

### 4.1 Logik-Probleme

#### Problem: SAT

**Gegeben:** Boolescher Ausdruck  $\alpha = C_1 \wedge \dots \wedge C_m$  in konjunktiver Normalform (KNF): Klauseln  $C_i$  sind Disjunktionen von Literalen ( $x_j$  oder  $\neg x_j$ ).

**Entscheide:** Gibt es eine erfüllende Belegung der Variablen?

**Status:** NP-vollständig (Cook–Levin). *Das* Ur-Problem.

#### Problem: 3-SAT

**Gegeben:** KNF-Formel mit *höchstens 3 Literalen pro Klausel*.

**Entscheide:** Erfüllbar?

**Status:** NP-vollständig, via  $\text{SAT} \leq 3\text{-SAT}$  (lange Klauseln mit Hilfsvariablen aufspalten). Startpunkt fast aller ETH-Reduktionen. Variante 3-SAT' (*genau* 3 Literale) ist ebenfalls NP-vollständig.

#### Problem: MAX-3-SAT (Optimierungsproblem)

**Gegeben:** KNF-Formel, jede Klausel mit 3 Literalen.

**Gesucht:** Belegung  $\beta$ , die die Anzahl  $v(\beta)$  erfüllter Klauseln maximiert.

**Status:** Optimierungsvariante von 3-SAT; in der Klausur als *Approximationsaufgabe* (Güte-2-Algorithmus, siehe Abschnitt 7.5).

### 4.2 Graphprobleme

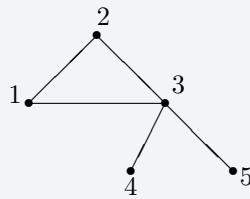
#### Problem: $k$ -Clique

**Gegeben:** Ungerichteter Graph  $G = (V, E)$ , Zahl  $k \geq 1$ . Eine *Clique* ist  $C \subseteq V$  mit  $\{u, v\} \in E$  für alle  $u \neq v$  aus  $C$ .

**Entscheide:** Existiert eine Clique mit  $|C| \geq k$ ?

**Status:** NP-vollständig, via  $\text{SAT} \leq k\text{-CLIQUE}$  (Vorlesungsbeweis – auswendig können!).

### Beispiel: Clique



$C = \{1, 2, 3\}$  ist eine Clique der Größe 3;  $\{2, 3, 4\}$  nicht (Kante  $\{2, 4\}$  fehlt).

### Problem: Independent Set (IS)

**Gegeben:**  $G = (V, E)$ , Zahl  $k$ . Ein *Independent Set* ist  $I \subseteq V$  ohne Kante zwischen zwei Knoten aus  $I$ .

**Entscheide:** Gibt es ein Independent Set mit  $|I| \geq k$ ?

**Status:** NP-vollständig –  $C$  ist Clique in  $G \iff C$  ist Independent Set im Komplementgraphen  $\bar{G}$ .

### Problem: Vertex Cover (VC)

**Gegeben:**  $G = (V, E)$ , Zahl  $k$ . Ein *Vertex Cover* ist  $C \subseteq V$ , sodass jede Kante mindestens einen Endpunkt in  $C$  hat.

**Entscheide:** Gibt es ein Vertex Cover mit  $|C| \leq k$ ?

**Status:** NP-vollständig, via  $\text{CLIQUE} \leq \text{VC}$ .

### Merke

**Dualitätsdreieck** (Kern vieler Aufgaben): In  $G$  mit  $n$  Knoten gilt

$$\begin{aligned} C \text{ ist Clique der Größe } k \text{ in } G &\iff C \text{ ist IS der Größe } k \text{ in } \bar{G} \\ &\iff V \setminus C \text{ ist VC der Größe } n - k \text{ in } \bar{G}. \end{aligned}$$

Merksatz: *Vertex Cover* = *Komplement eines Independent Set* (in demselben Graphen), *Clique* = *Independent Set im Komplementgraphen*.

### Problem: $k$ -Color (Färbungsproblem)

**Gegeben:**  $G = (V, E)$ , Zahl  $k$ . Eine  $k$ -Färbung ist  $f : V \rightarrow \{1, \dots, k\}$  mit  $f(u) \neq f(v)$  für alle  $\{u, v\} \in E$ .

**Entscheide:** Ist  $G$   $k$ -färbbar?

**Status:** NP-vollständig, via  $3\text{-SAT} \leq k\text{-COLOR}$  (Konstruktion mit  $n + 1$  Farben, Knoten  $x_i, \bar{x}_i, v_i$ , Klauselknoten  $F_j$  und Zentrum  $z$ ). Auch  $3\text{-COLOR}$  ist NP-vollständig.

### Problem: Hamiltonkreis (HK) und Hamiltonpfad

**Gegeben:**  $G = (V, E)$  ungerichtet (Aussage gilt auch gerichtet).

**Entscheide:** Gibt es einen Kreis, der jeden Knoten *genau einmal* besucht (formal: Permutation  $\pi$  mit  $\{v_{\pi(i)}, v_{\pi(i+1)}\} \in E$  zyklisch)?

**Status:** NP-vollständig, via  $3\text{-SAT}' \leq \text{HK}$  (Gadget-Konstruktion mit A-/B-Komponenten; nur Beweisskizze im Skript). HAMILTONPFAD (Pfad statt Kreis) ist ebenfalls NP-vollständig; Klausur SS23 verlangte die Reduktionen zwischen beiden in *beide* Richtungen.

### Problem: TSP (Traveling Salesman, Entscheidungsvariante)

**Gegeben:** Knoten  $\{1, \dots, n\}$ , Distanzen  $d(i, j)$ , Schranke  $L$ .

**Entscheide:** Gibt es eine Rundreise (jeder Knoten genau einmal) mit Länge  $\leq L$ ?

**Status:** NP-vollständig, via  $\text{HK} \leq \text{TSP}$  (Kanten  $\mapsto$  Distanz 1, Nicht-Kanten  $\mapsto |V| + 1$ ,  $L = |V|$ ). Bleibt NP-vollständig im metrischen Fall ( $\Delta$ -Ungleichung). Als Optimierungsproblem zentral in Kapitel 7.

### Problem: Feedback Vertex Set (FVS)

**Gegeben:** Gerichteter Graph  $G = (V, E)$ , Zahl  $k$ .

**Entscheide:** Gibt es  $X \subseteq V$  mit  $|X| \leq k$ , sodass  $G \setminus X$  kreisfrei ist?

**Status:** NP-vollständig, via  $\text{VC} \leq \text{FVS}$  (Serie 9/10: jede ungerichtete Kante wird zu zwei antiparallelen Bögen).

### Problem: $\Delta$ -Cover (Dreiecksüberdeckung)

**Gegeben:**  $G = (V, E)$  ungerichtet, Zahl  $k$ .

**Entscheide:** Gibt es  $C_\Delta \subseteq V$ ,  $|C_\Delta| \leq k$ , das *jede 3-Clique* (Dreieck) von  $G$  trifft?

**Status:** NP-vollständig, via  $\text{VC} \leq \Delta\text{-COVER}$  (Serie 11: jede Kante zu einem Dreieck aufblasen).

### Problem: Dominating Set

**Gegeben:**  $G = (V, E)$ , Zahl  $k$ .

**Entscheide:** Gibt es  $D \subseteq V$ ,  $|D| \leq k$ , sodass jeder Knoten in  $D$  liegt oder einen Nachbarn in  $D$  hat?

**Status:** NP-vollständig. In Klausuren als NP-Zugehörigkeits- und ETH-Beispiel (Reduktion von 3-SAT vorgegeben).

### Problem: Longest Path

**Gegeben:**  $G = (V, E)$ , Zahl  $k$ .

**Entscheide:** Gibt es einen einfachen Pfad der Länge  $\geq k$ ?

**Status:** NP-vollständig (verallgemeinert Hamiltonpfad). In der Klausur SS23 als NP-Zugehörigkeits-Aufgabe (Zertifikat/Verifizierer/NTM).

### 4.3 Mengen- und Zahlprobleme

#### Problem: 3-dimensionales Matching (3-DM)

**Gegeben:** Mengen  $U, V, W$  mit  $|U| = |V| = |W|$  und  $T \subseteq V \times W \times U$ .

**Entscheide:** Gibt es  $M \subseteq T$  mit  $|M| = |U|$ , sodass keine zwei Tripel in einer Komponente übereinstimmen?

**Status:** NP-vollständig, via  $\text{SAT} \leq 3\text{-DM}$  (Gadgets:  $a/b$ -Knoten für Variablensetzung,  $v/w$  für Klauselwahl,  $c/d$  als Garbage Collection).

#### Problem: 3-Exact Cover (3-EC)

**Gegeben:** Familie  $F = \{S_1, \dots, S_n\}$  von 3-elementigen Teilmengen einer Menge  $U$  mit  $|U| = 3m$ .

**Entscheide:** Gibt es  $m$  Mengen aus  $F$ , die  $U$  exakt überdecken (disjunkt, Vereinigung  $= U$ )?

**Status:** NP-vollständig – jede 3-DM-Instanz ist eine 3-EC-Instanz (Uminterpretation).

#### Problem: SubSet Sum

**Gegeben:** Ganze Zahlen  $c_1, \dots, c_n$  und Zielwert  $K$ .

**Entscheide:** Gibt es  $S \subseteq \{1, \dots, n\}$  mit  $\sum_{j \in S} c_j = K$ ?

**Status:** NP-vollständig, via  $3\text{-EC} \leq \text{SUBSET SUM}$  (Mengen als Bitvektoren zur Basis  $n+1$ ).

**Klausur-Spitzenreiter:** In jeder Altklausur taucht eine SubSet-Sum-Variante auf.

#### Problem: Partition

**Gegeben:** Ganze Zahlen  $c_1, \dots, c_n$ .

**Entscheide:** Gibt es  $S$  mit  $\sum_{j \in S} c_j = \frac{1}{2} \sum_{j=1}^n c_j$ ?

**Status:** NP-vollständig, via  $\text{SUBSET SUM} \leq \text{PARTITION}$  (zwei Zusatzzahlen  $N - K$  und  $K + 1$ ).

#### Problem: Rucksackproblem (Knapsack, Entscheidungsvariante)

**Gegeben:**  $n$  Gegenstände mit Größen  $c_j$  und Profiten  $p_j$ , Kapazität  $K$ , Zielprofit  $P$ .

**Entscheide:** Gibt es  $S$  mit  $\sum_{j \in S} c_j \leq K$  und  $\sum_{j \in S} p_j \geq P$ ?

**Status:** NP-vollständig – SubSet Sum ist der Spezialfall  $c_j = p_j$ ,  $P = K$ . Als Optimierungsproblem zentral in Kapitel 7 (Greedy, FPTAS).

#### Problem: Hitting Set

**Gegeben:** Universum  $U$ , Teilmengen  $F_1, \dots, F_r \subseteq U$ , Zahl  $k$ .

**Entscheide:** Gibt es  $S \subseteq U$ ,  $|S| \leq k$ , mit  $S \cap F_i \neq \emptyset$  für alle  $i$  („ $S$  trifft jede Menge“)?

**Status:** NP-vollständig; via  $3\text{-SAT} \leq \text{HITTING SET}$  (Serie 13). Standard-Beispiel für ETH-Lower-Bounds.

#### Problem: Set Cover

**Gegeben:** Universum  $U$ , Mengen  $S_1, \dots, S_r \subseteq U$ , Zahl  $k$ .

**Entscheide:** Gibt es  $\leq k$  Mengen, deren Vereinigung  $U$  ist?

**Status:** NP-vollständig; dual zu Hitting Set (Klausur SS23:  $\text{SET COVER} \leq \text{HITTING SET}$  für ETH-Schranken).

## 4.4 Scheduling

**Problem:**  $P \parallel C_{\max}$  (Scheduling auf identischen Maschinen)

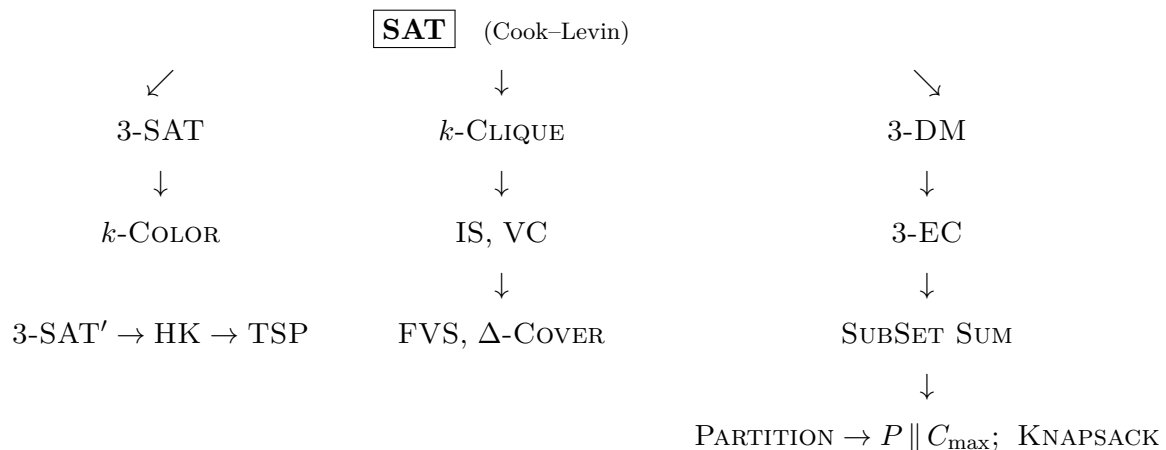
**Gegeben:**  $n$  Jobs mit Laufzeiten  $p_1, \dots, p_n$ ,  $m$  identische Maschinen.

**Gesucht:** Partition der Jobs in  $B_1, \dots, B_m$  mit minimalem *Makespan*  $C_{\max} = \max_i \sum_{J_j \in B_i} p_j$ .

**Status:** NP-vollständig schon für  $m = 2$  (Reduktion von PARTITION). Zentrale Approximationsalgorithmen: List Scheduling, LPT (Kapitel 7).

## 5 Die Reduktionslandkarte

Alle NP-Vollständigkeitsbeweise des Stoffs hängen an dieser Kette (Pfeil = „wird reduziert auf“):



Die Konstruktionsideen im Überblick (Details und Beweise in Uebung.pdf):

| Reduktion                              | Kernidee der Konstruktion                                         | Quelle      |
|----------------------------------------|-------------------------------------------------------------------|-------------|
| $SAT \leq 3-SAT$                       | lange Klauseln per Hilfsvariablen-Kette aufspalten                | Skript 6.25 |
| $SAT \leq k-CLIQUE$                    | Knoten $[i, j]$ pro Literal; Kanten = konsistent, andere Klausel  | Skript 6.26 |
| $3-SAT \leq k-COLOR$                   | $n+1$ Farben; Clique $\{v_1, \dots, v_n, z\}$ erzwingt Setzung    | Skript 6.28 |
| $CLIQUE \leq VC$                       | Komplementgraph; $C$ Clique $\iff V \setminus C$ VC, $k' = n - k$ | Präsenz 10  |
| $VC \leq FVS$                          | Kante $\rightarrow$ zwei antiparallele Bögen (2-Kreise)           | Serie 10    |
| $VC \leq \Delta-COVER$                 | Kante $\{u, v\} \rightarrow$ Dreieck $\{v_e, u, v\}$              | Serie 11    |
| $SAT \leq 3-DM$                        | Variablen-Rad, Klausel-Tripel, Garbage Collection                 | Skript 6.30 |
| $3-DM \leq 3-EC$                       | Uminterpretation (Tripel = 3er-Mengen)                            | Skript 6.31 |
| $3-EC \leq SUBSET SUM$                 | Mengen als Bitvektoren zur Basis $n+1$ ; kein Übertrag            | Skript 6.32 |
| $SUBSET SUM \leq PARTITION$            | Zusatzzahlen $N-K$ und $K+1$                                      | Skript 6.33 |
| $3-SAT' \leq HK$                       | A-Komponenten (XOR), B-Komponenten (Klauseln)                     | Skript 6.42 |
| $HK \leq TSP$                          | Kanten Distanz 1, Nicht-Kanten $ V +1$ , $L =  V $                | Skript 7.2  |
| $PARTITION \leq P2 \parallel C_{\max}$ | Zahlen = Jobs; Makespan $\frac{1}{2} \sum c_j$ ?                  | Skript 7.23 |
| $3-SAT \leq SUBSET SUM$ (streng)       | Ziffern zur Basis 10; $ A  = O(m)$ – für ETH                      | Skript 6.40 |

## 6 ETH: Exponentialzeit-Hypothese und untere Schranken

NP-Vollständigkeit sagt nur „vermutlich nicht polynomiell“. Die ETH ist eine *stärkere* Annahme, mit der man konkrete exponentielle *untere Laufzeitschranken* herleitet. In **jeder** Altklausur ist die letzte Aufgabe eine ETH-Aufgabe.

**Definition 6.1 (Klein-o-Notation)**

$f(x) = o(g(x)) \iff$  für jedes  $c > 0$  gilt ab einem  $x_0$ :  $|f(x)| \leq c \cdot |g(x)|$ ; äquivalent  $f(x)/g(x) \rightarrow 0$ . Beispiele:  $\sqrt{n} = o(n)$ ,  $n/\log n = o(n)$ , aber  $\delta n \neq o(n)$  für festes  $\delta > 0$ . Ein „ $2^{o(n)}$ -Algorithmus“ ist also einer, der *subexponentiell* in  $n$  läuft.

### Satz 6.2 (Exponentialzeit-Hypothese (ETH; Impagliazzo, Paturi, Zane 2001))

Es existiert ein  $\delta > 0$ , sodass 3-SAT mit  $n$  Variablen und  $m$  Klauseln *nicht* in Zeit  $2^{\delta n}(n+m)^{O(1)}$  gelöst werden kann.

Äquivalent formuliert: Es gibt keinen  $2^{o(n)} \cdot \text{poly}$ -Algorithmus für 3-SAT. (Unbewiesene Annahme; alle bekannten 3-SAT-Algorithmen brauchen  $c^n$ , derzeit bestes  $c \approx 1,307$ .)

### Satz 6.3 (Sparsification-Lemma)

Unter der ETH existiert  $\delta' > 0$ , sodass 3-SAT nicht in Zeit  $2^{\delta' m}(n+m)^{O(1)}$  lösbar ist – also auch kein  $2^{o(m)}$ -Algorithmus (Schranke in der *Klauselzahl*  $m$ ).

### Merke

Warum braucht man das Lemma? Die ETH spricht über  $n$  (Variablen). Reduktionen erzeugen Instanzgrößen aber oft abhängig von  $m$  (Klauseln). Es gilt zwar  $n \leq 3m$  (jede Variable komme vor), aber  $m$  kann bis  $O(n^3)$  groß sein – eine  $2^{o(m)}$ -Schranke folgt daher *nicht* direkt aus der ETH. Das Sparsification-Lemma schließt genau diese Lücke. In Klausurlösungen: Immer wenn dein Parameter an  $m$  hängt, das Sparsification-Lemma zitieren.

### Satz 6.4 (ETH-Konsequenzen (Skript 6.37–6.40))

Unter Annahme der ETH gibt es keinen  $2^{o(n)} \cdot \text{poly}$ -Algorithmus für: 3-COLOR ( $n$  = Knotenzahl; via Reduktion mit  $O(n+m)$  Knoten), CLIQUE, VERTEX COVER, INDEPENDENT SET, sowie SUBSET SUM und PARTITION ( $n$  = Anzahl Zahlen; via strenger Reduktion mit  $|A| = O(m)$  Items).

Das Beweisschema (Kontraposition) lernst du in Uebung.pdf, Kapitel 5: *Gäbe es einen  $2^{o(\text{Parameter})}$ -Algorithmus für das Zielproblem, ergäbe die Reduktion einen  $2^{o(m)}$ -Algorithmus für 3-SAT – Widerspruch zum Sparsification-Lemma.* Entscheidend ist, die Parameter der konstruierten Instanz (Knoten, Kanten,  $k, \dots$ ) durch  $n$  und  $m$  abzuschätzen.

## 7 Approximative Algorithmen

Wenn exakte Lösungen (vermutlich) exponentielle Zeit brauchen, berechnet man in Polynomialzeit *beweisbar gute Näherungen*.

### 7.1 Gütebegriff und Approximationsschemata

#### Definition 7.1 (Multiplikative (Worst-Case-)Güte)

Sei  $A$  ein Algorithmus für ein Optimierungsproblem,  $\text{OPT}(I)$  der optimale Wert zu Instanz  $I$ .  $A$  hat Güte  $\alpha$ , wenn für *alle* Instanzen  $I$ :

- Minimierungsproblem:  $A(I) \leq \alpha \cdot \text{OPT}(I)$  (z. B. TSP, VC,  $C_{\max}$ ),
- Maximierungsproblem:  $A(I) \geq \frac{1}{\alpha} \cdot \text{OPT}(I)$  (z. B. Knapsack, MAX-3-SAT).

Die Güte ist *scharf*, wenn es Instanzen gibt, die den Faktor (beliebig genau) erreichen.

### Definition 7.2 (PTAS, EPTAS, FPTAS)

Eine Familie  $(A_\varepsilon)$  von Algorithmen mit Güte  $1 + \varepsilon$  für jedes  $\varepsilon > 0$  heißt

- *PTAS* (polynomielles Approximationsschema): Laufzeit polynomiell in  $n$  für jedes feste  $\varepsilon$  (z. B.  $O(n^{1/\varepsilon})$ ),
- *EPTAS*: Laufzeit  $f(1/\varepsilon) \cdot \text{poly}(n)$  (z. B.  $2^{O(1/\varepsilon \log^2(1/\varepsilon))} + O(n)$  für  $P \parallel C_{\max}$ ),
- *FPTAS*: Laufzeit polynomiell in  $n$  und  $1/\varepsilon$  (z. B.  $O(n^3/\varepsilon)$  für Knapsack). Stärkste Form.

*Pseudo-polynomiell* heißt eine Laufzeit, die polynomiell in den *Zahlenwerten* der Eingabe ist (z. B.  $O(n^2 p_{\max})$  beim Knapsack-DP) – aber exponentiell in der Kodierungslänge.

## 7.2 TSP: $\Delta\text{TSP}_1$ und Christofides

### Satz 7.3 (Allgemeines TSP ist nicht approximierbar)

Für das allgemeine TSP gibt es *keinen* Approximationsalgorithmus mit beschränkter Güte  $\alpha$ , außer  $P = NP$ .

*Beweisidee:* Reduktion von HK: Nicht-Kanten bekommen Distanz  $\infty$  (bzw.  $\alpha|V| + 1$ ); ein  $\alpha$ -Approximationsalgorithmus könnte dann Hamiltonkreise erkennen. (Klausuraufgabe SS23!)

Deshalb: *metrisches TSP* ( $\Delta$ -TSP) mit symmetrischen Distanzen und Dreiecksungleichung  $d(i, j) \leq d(i, k) + d(k, j)$ . Bleibt NP-vollständig, ist aber approximierbar.

### Rezept: Algorithmus $\Delta\text{TSP}_1$ (MST-Verdopplung) – Güte 2

1. Berechne minimalen Spannbaum  $T$  von  $K_n$  mit Distanzen  $d$ .
2. Verdopple alle Kanten von  $T \rightarrow$  Multigraph  $G$  (alle Grade gerade).
3. Bestimme Eulerkreis  $K$  in  $G$ .
4. Kürze  $K$  zu Rundreise  $R$  ab (bereits besuchte Knoten überspringen).

Güte-Argument (nur Idee):  $w(T) \leq \text{OPT}$  (Tour minus Kante = Spannbaum),  $d(K) = 2w(T)$ , Abkürzen verlängert wegen  $\Delta$ -Ungleichung nicht. Also  $d(R) \leq 2 \text{OPT}$ . Schranke ist asymptotisch scharf (Stern-Beispiel:  $\text{OPT} = n$ , Ausgabe  $2n - 2$ ).

### Definition 7.4 (Eulerkreis)

Ein *Eulerkreis* in einem Multigraphen besucht jede *Kante* genau einmal. Existenzkriterium:  $G$  zusammenhängend und *jeder Knoten hat geraden Grad* (Königsberger Brückenproblem).

### Beispiel: $\Delta\text{TSP}_1$ durchgerechnet (Skript-Beispiel 7.7)

$V = \{A, B, C, D, E\}$ ; optimale Tour  $[C, B, D, E, A, C]$  mit Länge 8. Der MST hat Gewicht 6 (Kanten  $CB, BA$  je 1,  $CD, AE$  je 2). Verdopplung: Multigraph mit Gewicht 12. Eulerkreis  $K = [C, A, E, A, C, B, C, D, C]$ , Abkürzen liefert  $R = [C, A, E, B, D, C]$  mit Länge  $10 \leq 2 \cdot 8$ .

### Rezept: Algorithmus $\Delta\text{TSP}_2$ (Christofides) – Güte 1,5

1. Berechne minimalen Spannbaum  $T$ .
2.  $X :=$  Knoten mit *ungeradem* Grad in  $T$  ( $|X|$  ist gerade).
3. Berechne ein perfektes Matching  $K$  mit minimalem Gewicht auf  $X$  (geht in  $O(|V|^3)$ , Lawler 1976).
4.  $G := T + K$  (alle Grade gerade), Eulerkreis, abkürzen.

Güte-Argument (nur Idee): Die optimale Tour zerfällt auf  $X$  in zwei Matchings  $M_1, M_2$  mit  $d(M_1) + d(M_2) \leq \text{OPT}$ ; also  $d(K) \leq \text{OPT}/2$  und  $d(R) \leq w(T) + d(K) \leq 1,5 \text{ OPT}$ . Auch diese Schranke ist asymptotisch scharf (Leiter-Beispiel, Präsenzserie 12: Rate  $\rightarrow 3/2 - 1/n$ ).

### Achtung — typische Falle

Klausur-Klassiker: *Warum braucht Christofides die  $\Delta$ -Ungleichung?* Beim Abkürzen des Eulerkreises darf der Weg nicht länger werden – das garantiert nur die Dreiecksungleichung. Und: Das Matching wird auf den *ungerad-gradigen Knoten des MST* gebildet, nicht auf allen.

## 7.3 Knapsack: Greedy, MGA, FPTAS

### Rezept: Greedy GA – Güte unbeschränkt!

Sortiere nach Profitdichte  $p_i/w_i$  absteigend; nimm jeden Gegenstand, der noch passt. Laufzeit  $O(n \log n)$ .

**Aber:** Instanz  $(w_0, p_0) = (1, 1)$ ,  $(w_1, p_1) = (B, B - 1)$ : GA nimmt Gegenstand 0 (Dichte 1), Gewinn 1; optimal ist  $B - 1$ . Güte  $\geq B - 1$ , also *keine* konstante Güte.

### Rezept: Modified Greedy MGA – Güte 2

Berechne  $S_1$  mit GA und  $S_2 = \{\text{Gegenstand mit maximalem Einzelprofit}\}$ ; gib die bessere Lösung aus.

Güte-Argument (nur Idee): Die fraktionale Relaxierung packt nach Dichte und schneidet beim *Split-Item*  $k+1$  ab:  $\text{OPT} \leq \text{OPT}_f \leq \text{GA}(I) + p_{k+1} \leq \text{GA}(I) + p_{\max} \leq 2 \cdot \text{MGA}(I)$ .

### Satz 7.5 (Sahni-Algorithmus $A_k$ ( $k$ -Enumeration))

Probiere alle Teilmengen  $S$  mit  $|S| \leq k$  als Vorplatzierung, fülle mit GA auf, nimm das Beste. Güte  $\leq 1 + 1/k$ , Laufzeit  $O(n^{k+1})$  – ein PTAS.

### Satz 7.6 (FPTAS für Knapsack (Profit-Skalierung))

Skaliere Profite  $p'_i = \lfloor p_i/K \rfloor$  mit  $K = \frac{\varepsilon p_{\max}}{(1+\varepsilon)n}$ , löse mit dem pseudopolynomiellen DP ( $O(n^2 p'_{\max})$ ), gib dessen Lösung aus. Laufzeit  $O(n^3/\varepsilon)$ , Güte  $1 + \varepsilon$ .

*Idee:* Durch Abrunden verliert jede Lösung höchstens  $K$  pro Item, also  $\leq Kn$  gesamt; wegen  $\text{OPT} \geq p_{\max}$  ist der relative Fehler  $\leq \varepsilon$ .

## 7.4 Scheduling: List Scheduling und LPT

### Rezept: List Scheduling – Güte $2 - \frac{1}{m}$

Gehe die Jobs in Listenreihenfolge durch; weise jeden Job der Maschine mit aktuell *kleinster Last* zu.

Güte-Argument (nur Idee): Sei  $J_k$  der letzte Job auf der vollsten Maschine. Bis zu seinem Start waren alle Maschinen beschäftigt:  $LS(I) - p_k \leq \frac{1}{m} \sum_i p_i \leq \text{OPT}$  und  $p_k \leq \text{OPT}$  ergeben  $LS(I) \leq (2 - \frac{1}{m})\text{OPT}$ . Die Schranke wird von konkreten Instanzen erreicht (Klausuraufgabe: solche Instanzen konstruieren!).

### Rezept: LPT Scheduling – Güte $\frac{4}{3} - \frac{1}{3m}$

*Longest Processing Time first*: Sortiere die Jobs absteigend ( $p_1 \geq p_2 \geq \dots$ ), dann List Scheduling.

Beweis über minimales Gegenbeispiel: Dort wäre  $p_n > \text{OPT}/3$ , also höchstens 2 Jobs pro Maschine im Optimum; per Austausch-Transformationen wird das Optimum in einen LPT-Schedule überführt – Widerspruch. Auch  $4/3 - 1/(3m)$  wird asymptotisch erreicht.

### Beispiel: List Scheduling vs. LPT, $m = 2$

Jobs mit Zeiten  $(1, 1, 2)$  in Listenreihenfolge: List Scheduling legt die beiden Einsen auf  $M_1$  und  $M_2$ , die 2 dann auf  $M_1 \Rightarrow C_{\max} = 3$ . Optimal ist  $\{2\}/\{1, 1\}$  mit  $\text{OPT} = 2$ . Die Rate ist  $3/2 = 2 - \frac{1}{m}$  – die Schranke wird exakt erreicht. LPT sortiert zuerst:  $(2, 1, 1)$  ergibt  $C_{\max} = 2 = \text{OPT}$ .

### Satz 7.7 (EPTAS für $P \parallel C_{\max}$ )

Für jedes  $\varepsilon > 0$  gibt es  $A_\varepsilon$  mit  $A_\varepsilon(I) \leq (1 + \varepsilon)\text{OPT}(I)$  und Laufzeit  $2^{O(1/\varepsilon \cdot \log^2(1/\varepsilon))} + O(n)$ .

## 7.5 MAX-3-SAT: Güte 2

### Rezept: Komplementäre Belegungen – Güte 2

Werte die Formel unter  $\beta_0$  (alle Variablen false) und  $\beta_1$  (alle true) aus; gib die bessere Belegung zurück.

Warum das funktioniert: Jede Klausel enthält ein positives oder ein negatives Literal, wird also von  $\beta_0$  oder  $\beta_1$  erfüllt. Daher  $v(\beta_0) + v(\beta_1) \geq m$ , also  $\max \geq m/2 \geq \text{OPT}/2$ . (Beweis + scharfe Instanz: Uebung.pdf – war Klausuraufgabe SoSe 23.)

## 8 Halteproblem und Unentscheidbarkeit

### Problem: $\text{HALT}_{\text{TM}}$ (Halteproblem)

**Gegeben:** Kodierung  $\langle M, w \rangle$  einer DTM  $M$  und eines Wortes  $w$ .

**Entscheide:** Hält  $M$  auf  $w$  (erreicht eine Stopkonfiguration)?

**Status:** *Unentscheidbar* – kein Algorithmus löst es, egal mit welcher Laufzeit. Trotzdem **NP-schwer** (Serie 11): via  $\text{SAT} \leq \text{HALT}_{\text{TM}}$ ; konstruiere  $M_\varphi$ , die alle Belegungen durchprobiert und genau dann hält, wenn eine erfüllende existiert.

### Merke

$\text{HALT}_{\text{TM}}$  ist NP-schwer, aber *nicht* NP-vollständig – es liegt nicht in NP (nicht einmal entscheidbar). Wichtig ist die Pointe der Reduktion:  $M_\varphi$  läuft womöglich ewig, aber die *Konstruktion* von  $\langle M_\varphi, \varepsilon \rangle$  aus  $\varphi$  ist polynomiell – nur darauf kommt es an. (In den Altklausuren kam das Thema nie dran; Serienstoff ist es trotzdem.)

## 9 Wahr oder falsch? Die Logik-Merkkiste

Aussagen aus den Präsenzserien 10/11 – ideale Selbstkontrolle:

| Aussage                                                                                      | W/F | Begründung                                                                                                                                                                  |
|----------------------------------------------------------------------------------------------|-----|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Jedes Problem, das eine NDTM in Polyzeit löst, löst auch eine DTM in Polyzeit.               | F*  | Das wäre $P = NP$ ; gilt nur, falls $P = NP$ .                                                                                                                              |
| $A$ NP-schwer $\Rightarrow A \in P$ ausgeschlossen? / $A$ NP-schwer $\Rightarrow A \in NP$ ? | F   | NP-schwer sagt nichts über Zugehörigkeit zu NP (Bsp. Halteproblem). $A \in P$ würde $P = NP$ implizieren.                                                                   |
| $L \in NP$ und $L \leq 3\text{-SAT} \Rightarrow L$ NP-vollständig.                           | F   | Gegenbeispiel $L = \emptyset$ bzw. jedes einfache $L$ : reduzierbar auf 3-SAT heißt nicht schwer.                                                                           |
| $L \leq 3\text{-SAT}$ und $3\text{-SAT} \leq L \Rightarrow L$ NP-vollständig.                | W   | $3\text{-SAT} \leq L$ gibt NP-Schwere; $L \leq 3\text{-SAT}$ gibt $L \in NP$ .                                                                                              |
| $L$ NP-vollständig: $L \in P \iff P = NP$ .                                                  | W   | Satz 6.16.                                                                                                                                                                  |
| Es ist möglich, dass $3\text{-SAT} \in P$ und $\text{CLIQUE} \notin P$ .                     | F   | $\text{CLIQUE} \leq 3\text{-SAT}$ (3-SAT ist NP-vollständig); aus $3\text{-SAT} \in P$ folgt $\text{CLIQUE} \in P$ .                                                        |
| Wenn die ETH falsch ist, gilt $P = NP$ .                                                     | F   | ETH falsch heißt nur: 3-SAT in $2^{o(n)}$ – das kann immer noch superpolynomiell sein (z.B. $2^{\sqrt{n}}$ ). Umgekehrt gilt aber: $P = NP \Rightarrow \text{ETH falsch}$ . |

\* genauer: nicht beweisbar wahr; äquivalent zu  $P = NP$ .

### Merke

**Weiter geht's mit Uebung.pdf:** Dort stehen die Schritt-für-Schritt-Rezepte für alle sechs Klausur-Aufgabentypen, die vier Vorlesungsbeweise in voller Länge, alle Reduktionen der Serien mit Musterbeweis – und Selbsttest-Aufgaben aus echten Klausuren.