

Die 7 Aufgabentypen

Musterlösungen, Rezepte und Bewertungshinweise
Analyse von Algorithmen und Komplexität – CAU Kiel, SS 2026
Schritt 3: die Muster kennen und richtig anwenden

*Jeder Klausur-Aufgabentyp einmal vorgemacht –
mit Rezept, kompletter Musterlösung und dem, worauf der Korrektor achtet.*

Wofür dieses Heft. Du hast das Fundament verstanden. Jetzt lernst du, es *anzuwenden*. Alle bisherigen Klausuren, Haus- und Präsenzaufgaben lassen sich auf **sieben Aufgabentypen** zurückführen. Zu jedem zeigt dieses Heft: ein *Rezept* (Schritt für Schritt), eine *vollständig gelöste* Musteraufgabe im richtigen Stil, die *Bewertungslogik* und die *häufigen Fehler*.

Die Reihenfolge ist Absicht. Die Typen bauen aufeinander auf. Arbeite sie von vorne nach hinten durch:

#	Aufgabentyp	baut auf
1	NP-Zugehörigkeit zeigen	Grundbaustein
2	Reduktion konstruieren (NP-Schwere)	braucht 1
3	Wahr oder falsch?	festigt 1+2
4	ETH-Schranke herleiten	braucht 2
5	Approximationsalgorithmus anwenden	neues Teilgebiet
6	Güte beweisen + Worst-Case-Instanz	braucht 5
7	Turingmaschine entwerfen	eigenständig

Der Stil ist echt. Die Musterlösungen sind im Stil bewerteter Abgaben gehalten: knapp, formal, jeder Schritt sichtbar begründet. Die Bewertungshinweise stammen aus echten Punkteschemata und Korrektor-Anmerkungen.

Inhalt

1	NP-Zugehörigkeit zeigen	3
2	Reduktion konstruieren (NP-Schwere)	5
3	Wahr oder falsch? Mit Begründung	8
4	ETH-Schranke herleiten	10
5	Approximationsalgorithmus anwenden	12
6	Güte beweisen und Worst-Case-Instanz bauen	14
7	Turingmaschine entwerfen	17

1 NP-Zugehörigkeit zeigen

★ In diesem Kapitel

Der erste und einfachste Typ. Du zeigst, dass ein Problem in NP liegt – indem du ein Zertifikat angibst und beschreibst, wie man es prüft. Dieser Baustein steckt später in *jedem* NP-Vollständigkeitsbeweis.

1.1 Woran du diesen Typ erkennst

Die Aufgabe klingt so (wörtlich aus der Klausur SS23): „Zeigen Sie auf *verschiedene Weisen*, dass das Problem in NP liegt: (a) geben Sie ein Zertifikat an, (b) beschreiben Sie einen Verifizierer mit konkreter Laufzeit, (c) beschreiben Sie in Worten eine nichtdeterministische Turingmaschine.“

1.2 Das Rezept

► Rezept: Der NP-Dreischritt

(a) **Zertifikat.** Nenne die „Lösung“ als Objekt (Teilmenge, Belegung, Reihenfolge). Sage, wie sie kodiert ist und dass sie *polynomiell lang* ist.

(b) **Verifizierer.** Ein deterministischer Algorithmus, der (Instanz, Zertifikat) bekommt. Hake drei Punkte ab:

1. *Beschreibung:* Welche Prüfungen laufen in welcher Reihenfolge?
2. *Korrektheit – beide Richtungen:* Ja-Instanz $\Rightarrow$ es gibt ein akzeptiertes Zertifikat; Nein-Instanz $\Rightarrow$ keines wird akzeptiert.
3. *Laufzeit:* konkret in O -Notation, nicht nur „polynomiell“.

(c) **NTM.** „Rate und prüfe.“ Rate das Zertifikat Stück für Stück nichtdeterministisch, prüfe dann deterministisch wie in (b).

1.3 Musteraufgabe

□ Aufgabe: Longest Path (Klausur SS23, Aufgabe 2)

Gegeben ein Graph $G = (V, E)$ und eine Zahl k . Gibt es einen *einfachen* Pfad (kein Knoten doppelt) der Länge $\geq k$? Zeigen Sie auf drei Weisen, dass LONGEST PATH in NP liegt.

✓ Musterlösung

(a) **Zertifikat.** Eine Folge von $k + 1$ paarweise verschiedenen Knoten $(v_0, v_1, \dots, v_k)$ – der behauptete Pfad. Jeder Knoten braucht $O(\log |V|)$ Bits, die Folge also $O(k \log |V|)$ Bits. Das ist polynomiell in der Eingabegröße.

(b) **Verifizierer.** Er bekommt (G, k) und ein Zertifikat $(v_0, \dots, v_k)$ und prüft:

- ob alle v_i paarweise verschieden sind – $O(k^2)$ Vergleiche (oder $O(k \log k)$ durch Sortieren);
- ob $\{v_i, v_{i+1}\} \in E$ für alle $i = 0, \dots, k - 1$ gilt – das sind k Nachschläge in der Adjazenzmatrix zu je $O(1)$, also $O(k)$.

Akzeptiere genau dann, wenn beide Prüfungen bestehen.

Korrektheit (beide Richtungen). Hat G einen einfachen Pfad der Länge $\geq k$, so bilden seine ersten $k + 1$ Knoten ein Zertifikat, das akzeptiert wird. Wird umgekehrt ein Zertifikat akzeptiert, so ist es eine Folge verschiedener Knoten mit lauter echten Kanten – also ein einfacher Pfad der Länge k .

Laufzeit. Gesamt $O(k^2) \subseteq O(|V|^2)$, polynomiell. Somit ist dies ein polynomieller Verifizierer, und $\text{LONGEST PATH} \in \text{NP}$.

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

1.4 Bewertung und häufige Fehler

📖 Bewertung – worauf es ankommt

Typisches Punkteschema pro Teil: *Beschreibung* 2 P., *Korrektheit* 1 P., *Laufzeit* 1 P. Der Korrektor will sehen:

- eine *konkrete* O -Notation – „polynomiell“ allein reicht nicht;
- *beide* Korrektheitsrichtungen, nicht nur „Pfad $\Rightarrow$ Zertifikat“;
- dass die *Kardinalität* ($|C| \leq k$ bzw. Länge $\geq k$) mitgeprüft wird.

✗ Stolperstein

Die drei Klassiker: (1) nur „polynomiell“ schreiben, wo eine konkrete Schranke verlangt ist. (2) Die Rückrichtung der Korrektheit weglassen. (3) Beim Verifizierer vergessen, die Größe zu prüfen – ein Zertifikat könnte sonst „zu klein“ oder „zu groß“ sein und würde trotzdem akzeptiert.

1.5 Zweite Variante (kurz): Knapsack $\in$ NP

Damit du das Muster auf ein Zahlenproblem überträgst:

✓ Musterlösung

Zertifikat: eine Teilmenge $S \subseteq \{1, \dots, n\}$ der Gegenstände (ein Bit pro Gegenstand).
Verifizierer: berechne $\sum_{i \in S} w_i$ und $\sum_{i \in S} p_i$, prüfe $\sum_{i \in S} w_i \leq B$ und $\sum_{i \in S} p_i \geq P$; akzeptiere entsprechend. Beide Summen und Vergleiche in $O(n)$. **Korrektheit:** Existiert eine zulässige Auswahl mit genügend Profit, wird sie akzeptiert; sonst keine. Also KNAPSACK $\in$ NP. $\square$

2 Reduktion konstruieren (NP-Schwere)

★ In diesem Kapitel

Der zentrale und häufigste Klausurtyp. Du beweist, dass ein *neues* Problem schwer ist, indem du ein bekannt schweres Problem darauf reduzierst. Fast immer ist das neue Problem eine *Variante* eines Standardproblems.

2.1 Woran du diesen Typ erkennst

„Zeigen Sie, dass X NP-vollständig (oder NP-schwer) ist.“ Meist ist X ein leicht abgewandeltes bekanntes Problem – mit einer Zusatzeigenschaft, gerichtet statt ungerichtet, mit einer Nebenbedingung.

2.2 Das Rezept

► Rezept: NP-Vollständigkeit in 6 Schritten

1. $\in$ NP. Verifizierer angeben (oft: „wie beim Originalproblem“). Bei reiner NP-Schwere entfällt dieser Schritt.
2. **Quellproblem wählen.** Ein bekanntes NP-vollständiges X , das dem Ziel Y ähnlich ist.
Richtung: $X \leq_p Y$ (bekannt $\leq_p$ neu).
3. **Konstruktion.** Wie wird aus einer X -Instanz eine Y -Instanz? Präzise, mit *allen* Parametern – auch die Schranke k anpassen!
4. **Polynomialität.** Ein Satz: „Die Konstruktion läuft 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.** „Da X NP-vollständig, $X \leq_p Y$ und $Y \in$ NP: Y ist NP-vollständig.“

2.3 Musteraufgabe 1

□ Aufgabe: Feedback Vertex Set (Hausaufgabe 10.1)

FEEDBACK VERTEX SET (FVS): Gegeben ein *gerichteter* Graph $G = (V, E)$ und k . Gibt es $X \subseteq V$ mit $|X| \leq k$, sodass $G \setminus X$ *kreisfrei* ist? Zeigen Sie: FVS ist NP-vollständig.

✓ Musterlösung

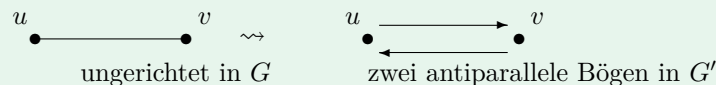
1. $\in \text{NP}$. Zertifikat $X \subseteq V$. Der Verifizierer prüft $|X| \leq k$ und ob $G \setminus X$ kreisfrei ist – Letzteres mit dem Algorithmus zur *topologischen Sortierung* (findet genau dann eine Sortierung, wenn der Graph kreisfrei ist). Beides in $O(|V| + |E|)$, also $\text{FVS} \in \text{NP}$.

2. **Quellproblem.** VERTEX COVER ist NP-vollständig. Wir zeigen $\text{VC} \leq_p \text{FVS}$.

3. **Konstruktion.** Sei $(G = (V, E), k)$ eine VC-Instanz (ungerichtet). Baue den gerichteten Graphen $G' = (V, E')$ mit

$$E' = \{ (u, v), (v, u) \mid \{u, v\} \in E \},$$

also: *jede ungerichtete Kante wird zu zwei antiparallelen Bögen* (einem gerichteten Kreis der Länge 2). Die Schranke bleibt $k' = k$. Ausgabe (G', k) .



4. **Polynomialität.** Pro Kante werden zwei Bögen erzeugt, also $O(|E|)$ – polynomiell.

5. **Korrektheit.**

$\Rightarrow$ Sei C ein Vertex Cover mit $|C| \leq k$. Dann berührt C jede Kante, also enthält $G \setminus C$ keine Kante. Damit enthält $G' \setminus C$ keinen Bogen und erst recht keinen Kreis – C ist ein FVS.

$\Leftarrow$ Sei X ein FVS mit $|X| \leq k$. Angenommen, X wäre *kein* Vertex Cover. Dann gäbe es eine Kante $\{u, v\} \in E$ mit $u \notin X$ und $v \notin X$. In G' bleiben dann beide Bögen (u, v) und (v, u) übrig – also der Kreis (u, v, u) in $G' \setminus X$. Das ist ein Widerspruch dazu, dass X ein FVS ist. Also ist X ein Vertex Cover.

6. **Schluss.** Da VC NP-vollständig ist, $\text{VC} \leq_p \text{FVS}$ gilt und $\text{FVS} \in \text{NP}$, ist FVS NP-vollständig. □

2.4 Musteraufgabe 2 (kurz): Clique $\leq_p$ Vertex Cover

Die kürzeste Reduktion überhaupt – reine Komplement-Umformung, kein Gadget.

✓ Musterlösung

Konstruktion. Aus der CLIQUE-Instanz (G, k) mit $n = |V|$ mache die VC-Instanz $(\bar{G}, n - k)$: nimm den Komplementgraphen und setze die Schranke auf $n - k$.

Korrektheit. $\Rightarrow$ Ist C eine Clique der Größe $\geq k$ in G , so ist C ein Independent Set in $\bar{G}$, und $V \setminus C$ ein Vertex Cover in $\bar{G}$ der Größe $\leq n - k$. $\Leftarrow$ Ist D ein Vertex Cover der Größe $\leq n - k$ in $\bar{G}$, so ist $V \setminus D$ ein Independent Set der Größe $\geq k$ in $\bar{G}$ – also eine Clique der Größe $\geq k$ in G .

Laufzeit. Komplementgraph in $O(n^2)$. □

2.5 Bewertung und häufige Fehler

📖 Bewertung – worauf es ankommt

Punkteschema (aus Hausaufgabe 11.1): Verifizierer 1+0,5+0,5 P.; Angabe der Reduktion 1 P.; *Korrektheit* 3 P. – davon **1,5 pro Richtung**; Laufzeit 1 P.; Beweisführung 1 P. Merke: Die Rückrichtung zu vergessen kostet also glatt 1,5 Punkte.

✗ Stolperstein

Die vier teuren Fehler:

- *Falsche Richtung.* Immer **bekannt** $\leq_p$ **neu** ($X_{\text{bekannt}} \leq_p Y_{\text{neu}}$), nie umgekehrt.
- *Rückrichtung vergessen* – kostet 1,5 von 5 Punkten.
- *Zusatzeigenschaft nicht hergestellt.* Prüfe, dass jede konstruierte Y -Instanz die geforderte Sonderstruktur wirklich hat (universeller Knoten, Grad-Bedingung, ...).
- **Nicht alle Fälle in der Korrektheit.** Siehe der echte Punktabzug unten.

📖 Bewertung – worauf es ankommt

Echter Punktabzug (Hausaufgabe 11.1, 9/10). Bei der Reduktion $VC \leq_p \Delta\text{-COVER}$ (jede Kante $e = \{u, v\}$ wird zum Dreieck $\{v_e, u, v\}$ aufgeblasen) stand im Beweis: „jedes Dreieck in G' hat die Form $\{v_e, u, v\}$ “. Das ist falsch: Weil $E \subseteq E'$ gilt, *erbt* G' auch alle Dreiecke, die G schon hatte (drei paarweise verbundene Original-Knoten). Diese haben *nicht* die Gadget-Form. Der Beweis muss zusätzlich zeigen, dass auch so ein geerbtes Dreieck $\{a, b, c\}$ vom Cover getroffen wird – was aus dem Vertex Cover folgt (die Kante $\{a, b\}$ ist gedeckt, also $a \in C$ oder $b \in C$). *Lehre: In der Korrektheit immer **alle** Objekte abdecken, auch die aus dem Originalgraphen geerbten.*

3 Wahr oder falsch? Mit Begründung

★ In diesem Kapitel

Dieser Typ testet, ob du die Begriffe wirklich beherrschst. Du bekommst mehrere Aussagen über P, NP, Reduktionen und Vollständigkeit und musst jede als wahr oder falsch *begründen*. Er festigt genau das, was du in Kapitel 1 und 2 gelernt hast.

3.1 Das Rezept

► Rezept: So knackst du jede Aussage

1. Führe die Aussage auf die *Definitionen* zurück. Zwei Merksätze reichen fast immer:
 - $NP\text{-schwer}$ = untere Schranke („mindestens so schwer wie alles in NP“). $\in NP$ = obere Schranke.
 - $A \leq_p B$ heißt „B mindestens so schwer wie A“. Die Richtung ist alles.
2. Ist die Aussage *falsch*, gib ein **konkretes Gegenbeispiel** (oft $L = \emptyset$ oder ein triviales $L \in P$).
3. Ist sie *wahr*, begründe in einem Satz aus der Definition.

3.2 Musteraufgabe

□ Aufgabe: Fragen über Fragen (Präsenz 11.1)

Entscheide jeweils wahr/falsch und begründe. Sei L eine Sprache.

- (a) $L \in NP$ und $L \leq_p 3\text{-SAT} \Rightarrow L$ ist NP-vollständig.
- (b) $L \leq_p 3\text{-SAT}$ und $3\text{-SAT} \leq_p L \Rightarrow L$ ist NP-vollständig.
- (c) L NP-vollständig $\Rightarrow (L \in P \iff P = NP)$.
- (d) Es ist möglich, dass $3\text{-SAT} \in P$ und $\text{CLIQUE} \notin P$.

✓ Musterlösung

(a) **Falsch.** $L \leq_p 3\text{-SAT}$ heißt nur, dass sich L auf 3-SAT reduziert – das macht L *nicht* schwer, sondern höchstens „so leicht wie 3-SAT“. Gegenbeispiel: $L = \emptyset$. Es liegt in NP und ist auf 3-SAT reduzierbar, aber sicher nicht NP-vollständig. Für NP-Schwere bräuchte man die *andere* Richtung $3\text{-SAT} \leq_p L$.

(b) **Wahr.** Das ist genau die Definition. Aus $3\text{-SAT} \leq_p L$ folgt, dass L NP-schwer ist (3-SAT ist NP-vollständig, Schwere vererbt sich). Aus $L \leq_p 3\text{-SAT}$ folgt $L \in \text{NP}$ (denn $3\text{-SAT} \in \text{NP}$, und die Reduktion überträgt die Zugehörigkeit). Schwer und in NP = NP-vollständig.

(c) **Wahr.** Ist L NP-vollständig, so reduziert sich jedes $L' \in \text{NP}$ auf L . Läge L in P, wäre damit jedes $L' \in \text{P}$, also $\text{P} = \text{NP}$. Umgekehrt folgt aus $\text{P} = \text{NP}$ trivial $L \in \text{P}$. (Das ist der zentrale Satz über NP-vollständige Probleme.)

(d) **Falsch.** CLIQUE lässt sich auf 3-SAT reduzieren (3-SAT ist NP-vollständig, also ist jedes NP-Problem auf es reduzierbar). Läge 3-SAT in P, folgte über diese Reduktion auch $\text{CLIQUE} \in \text{P}$. Die beschriebene Situation kann es also nicht geben. $\square$

3.3 Zweite Variante (kurz): Präsenz 10.1

✓ Musterlösung

(i) „Löst eine NDTM ein Problem in Polyzeit, dann auch eine gewöhnliche DTM.“ – *Nicht beweisbar wahr*: Das würde $\text{P} = \text{NP}$ bedeuten, und das ist offen.

(ii) „ A NP-schwer $\Rightarrow A \in \text{P}$ ausgeschlossen / $A \in \text{NP}$.“ – *Falsch*: NP-schwer sagt nichts über die Zugehörigkeit zu NP. Das HALTEPROBLEM ist NP-schwer, liegt aber nicht in NP (es ist nicht einmal entscheidbar). $\square$

3.4 Bewertung und häufige Fehler

📖 Bewertung – worauf es ankommt

Hier zählt *ausschließlich die Begründung*, nicht das Kreuz. Ein richtiges „falsch“ ohne Gegenbeispiel gibt kaum Punkte. Nenne bei falschen Aussagen immer ein *konkretes* Gegenbeispiel; bei wahren die *eine* Definition, aus der es folgt.

✗ Stolperstein

Der Dauerbrenner ist die *Reduktionsrichtung*. „ $L \leq_p 3\text{-SAT}$ “ macht L nicht schwer – es macht L höchstens leicht. Verwechselst du die Richtung, drehst du fast jede Aussage ins Gegenteil.

4 ETH-Schranke herleiten

★ In diesem Kapitel

Die typische *letzte* Klausuraufgabe. Aus einer *vorgegebenen* Reduktion leitest du untere Laufzeitschranken her – unter Annahme der Exponentialzeit-Hypothese (ETH). Du musst hier keine Reduktion erfinden, sondern sauber *rechnen* und argumentieren.

4.1 Woran du diesen Typ erkennst

„Gegeben die Reduktion Zeigen Sie unter Annahme der ETH, dass X nicht in Zeit $2^{o(p)} \cdot \text{poly}$ lösbar ist – für die Parameter $p \in \{\dots\}$.“ Meist sind mehrere Parameter gefragt (Universumsgröße, Mengenzahl, k), und bei manchen brauchst du das Sparsification-Lemma, bei anderen nicht.

4.2 Das Rezept

► Rezept: ETH-Schranke in 3 Schritten

1. Parameter durch n und m ausdrücken. Drücke jede gefragte Instanzgröße (Knoten, Mengen, k , . . .) durch die Variablenzahl n und die Klauselzahl m der 3-SAT-Formel aus. Nutze $n \leq 3m$.

2. Kontrapositions-Baustein hinschreiben. Für jeden Parameter p :

Angenommen, es gäbe einen Algorithmus, der X in $2^{o(p)} \cdot |I|^{O(1)}$ löst. Zusammen mit der Reduktion ergäbe das einen Algorithmus für 3-SAT in $2^{o(\cdot)}$. Nach der ETH (bzw. dem Sparsification-Lemma) ist das ausgeschlossen – Widerspruch.

3. Das Sparsification-Lemma nur zitieren, wenn nötig. Hängt p an n , brauchst du es *nicht* (die ETH spricht direkt über n). Hängt p an m , *musst* du es zitieren (die ETH allein verbietet $2^{o(m)}$ nicht).

4.3 Musteraufgabe

□ Aufgabe: Lower Bounds für Hitting Set (Hausaufgabe 12.2)

HITTING SET: Universum U , Mengen $F_1, \dots, F_r \subseteq U$, Zahl k . Gibt es $S \subseteq U$, $|S| \leq k$, das jede Menge trifft ($S \cap F_i \neq \emptyset$)? Gegeben ist die Reduktion $3\text{-SAT} \leq_p \text{HITTING SET}$:

- $U = \{x_1, \bar{x}_1, \dots, x_n, \bar{x}_n\}$ (alle Literale),
- für jede Variable i die Menge $F_i = \{x_i, \bar{x}_i\}$,
- für jede Klausel j die Menge $F_{n+j} = \{\text{Literale von } C_j\}$,
- $k = n$.

Geben Sie (1) die Parameter $|U|, r, k$ an und leiten Sie (2) unter der ETH die unteren Schranken bezüglich $|U|$, r und k her.

✓ Musterlösung

Teil 1 – Parameter. Es gilt

$$|U| = 2n, \quad r = n + m, \quad k = n.$$

Teil 2 – untere Schranken.

Bezüglich $|U|$: kein $2^{o(|U|)}$. Angenommen, ein Algorithmus löste HITTING SET in $2^{o(|U|)} \cdot \text{poly}$. Wegen $|U| = 2n$ wäre das $2^{o(n)}$. Über die Reduktion ergäbe das einen $2^{o(n)}$ -Algorithmus für 3-SAT – Widerspruch *direkt* zur ETH. (Kein Sparsification-Lemma nötig, da $|U|$ nur an n hängt.)

Bezüglich r : kein $2^{o(r)}$. Hier ist $r = n + m$. Wegen $n \leq 3m$ gilt $r \leq 4m = O(m)$. Ein $2^{o(r)}$ -Algorithmus wäre also ein $2^{o(m)}$ -Algorithmus und ergäbe über die Reduktion einen $2^{o(m)}$ -Algorithmus für 3-SAT. Nach dem **Sparsification-Lemma** ist das unter der ETH ausgeschlossen – Widerspruch. (Hier ist das Lemma *zwingend*, weil der Parameter an m hängt.)

Bezüglich k : kein $2^{o(k)}$. Wegen $k = n$ ist das genau wie im ersten Fall: $2^{o(k)} = 2^{o(n)}$ ergäbe $2^{o(n)}$ für 3-SAT – direkter Widerspruch zur ETH. □

4.4 Bewertung und häufige Fehler

📖 Bewertung – worauf es ankommt

Wiederhole den Schluss-Baustein *in jedem* Teilpunkt vollständig – der Korrektor will pro Parameter das ganze Argument sehen, nicht nur beim ersten. Entscheidend ist, ob du bei jedem Parameter richtig erkennst, *ob* das Sparsification-Lemma gebraucht wird.

✗ Stolperstein

Der Kern: n oder m ? Hängt der Parameter an der Variablenzahl n , argumentierst du *direkt* mit der ETH. Hängt er an der Klauselzahl m (oder an $n + m$, was $O(m)$ ist), *musst* du das Sparsification-Lemma nennen. Es einfach immer oder nie zu zitieren, kostet Punkte.

5 Approximationsalgorithmus anwenden

★ In diesem Kapitel

Ein Themenwechsel: weg von den Härte-Beweisen, hin zur Approximation. Dieser Typ ist eine reine *Rechenaufgabe* – du lässt einen bekannten Algorithmus Schritt für Schritt auf eine konkrete Instanz laufen. Wenig Kreativität, viel Sorgfalt.

5.1 Woran du diesen Typ erkennst

„Wenden Sie den Algorithmus ... auf die folgende Instanz an. Geben Sie alle Zwischenschritte an.“
Kandidaten: List Scheduling / LPT, ΔTSP_1 , Christofides, Knapsack-Greedy / Modified Greedy.

5.2 Das Rezept

➤ Rezept: Was du flüssig können musst

- **LPT / List Scheduling:** Jobs (bei LPT erst absteigend sortieren) der Reihe nach auf die Maschine mit *kleinster* Last; Schedule als Gantt-Balken zeichnen; Güte nennen.
- ΔTSP_1 : minimaler Spannbaum $\rightarrow$ Kanten verdoppeln $\rightarrow$ Eulerkreis $\rightarrow$ abkürzen.
- **Christofides:** MST $\rightarrow$ ungerade Knoten $\rightarrow$ min. Matching darauf $\rightarrow$ Eulerkreis $\rightarrow$ abkürzen. *Alle Zwischengraphen angeben.*
- **Knapsack-Greedy / MGA:** nach Profitdichte p_i/w_i sortieren, einpacken was passt; MGA zusätzlich mit dem besten Einzel-Item vergleichen.

5.3 Musteraufgabe

□ Aufgabe: Knapsack Greedy und Modified Greedy (Klausur SS23, Aufgabe 1)

Kapazität $B = 16$. Gegenstände als (p_i, w_i) :

$(1, 4), (1, 1), (1, 3), (11, 13), (3, 3), (5, 6), (1, 5)$.

Führen Sie Greedy (GA) und Modified Greedy (MGA) aus und vergleichen Sie mit dem Optimum.

✓ Musterlösung

Schritt 1 – Profitdichten p_i/w_i berechnen und sortieren.

(p, w)	(1, 4)	(1, 1)	(1, 3)	(11, 13)	(3, 3)	(5, 6)	(1, 5)
Dichte	0,25	1	0,33	0,85	1	0,83	0,2

Absteigend sortiert: $(1, 1), (3, 3), (11, 13), (5, 6), (1, 3), (1, 4), (1, 5)$.

Schritt 2 – Greedy einpacken (Kapazität $B = 16$, laufende Last):

- $(1, 1)$: passt, Last 1.
- $(3, 3)$: passt, Last 4.
- $(11, 13)$: $4 + 13 = 17 > 16$ – *passt nicht*.
- $(5, 6)$: $4 + 6 = 10$, passt, Last 10.
- $(1, 3)$: $10 + 3 = 13$, passt, Last 13.
- $(1, 4)$: $13 + 4 = 17 > 16$ – passt nicht. $(1, 5)$: $13 + 5 = 18$ – passt nicht.

Greedy-Profit: $GA = 1 + 3 + 5 + 1 = 10$.

Schritt 3 – Modified Greedy. Vergleiche mit dem besten Einzel-Item. Das ist $(11, 13)$ mit Profit $11 > 10$. Also $MGA = 11$.

Schritt 4 – Optimum und Vergleich. Optimal ist $\{(11, 13), (3, 3)\}$: Gewicht $13 + 3 = 16 \leq B$, Profit $OPT = 14$. Greedy verfehlt das schwere, profitable Item wegen seiner mittelmäßigen Dichte; MGA repariert das teilweise (garantierte Güte 2). □

5.4 Zweite Variante: Christofides mit allen Zwischenschritten

□ Aufgabe: Christofides (Klausur SS23-N, Aufgabe 1)

Knoten a, b, c, d mit $d(a, b) = 1$, $d(b, c) = 2$, $d(a, d) = 2$, $d(a, c) = 3$, $d(c, d) = 3$, $d(b, d) = 3$. Wenden Sie Christofides an.

✓ Musterlösung

- (1) **MST** (Kruskal): $\{a, b\}$ (1), $\{b, c\}$ (2), $\{a, d\}$ (2); Gewicht 5.
 - (2) **Ungerade Grade im MST**: a hat Grad 2, b Grad 2, c Grad 1, d Grad 1 $\Rightarrow X = \{c, d\}$.
 - (3) **Minimales perfektes Matching auf X** : nur die Kante $\{c, d\}$, Kosten 3.
 - (4) **Multigraph $T + K$** : Kanten $a-b, b-c, a-d, c-d$ – alle Grade jetzt gerade. **Eulerkreis** ab a : $[a, b, c, d, a]$, Kosten $1 + 2 + 3 + 2 = 8$.
 - (5) **Abkürzen**: kein Knoten kommt doppelt vor – dieser Schritt ändert hier *nichts* (trotzdem erwähnen!). Tour $[a, b, c, d, a]$, Länge 8.
- Hier ist sogar $8 = \text{OPT}$. Güte-Garantie: 1,5; benötigt Symmetrie *und* Dreiecksungleichung.

□

5.5 Bewertung und häufige Fehler

📝 Bewertung – worauf es ankommt

Punkte gibt es für die *Zwischenschritte*, nicht nur das Endergebnis. Bei Christofides: alle Zwischengraphen angeben. Bei Scheduling: den Schedule als Gantt-Diagramm zeichnen. Und stets die *Güte* des Verfahrens dazuschreiben.

✗ Stolperstein

Der Klassiker: den *Abkürzen*-Schritt weglassen, wenn er nichts ändert – er gehört trotzdem hin. Und bei Greedy die Dichte-Sortierung nicht sauber hinschreiben, sodass die Einpack-Reihenfolge nicht nachvollziehbar ist.

6 Güte beweisen und Worst-Case-Instanz bauen

★ In diesem Kapitel

Der anspruchsvollste Approximationstyp, fast immer *zweiteilig*: (a) beweise, dass ein Algorithmus eine bestimmte Güte einhält, und (b) konstruiere eine Instanz, die diese Schranke (fast) erreicht – die also zeigt, dass die Schranke *scharf* ist.

6.1 Das Rezept

► Rezept: Die zwei Teile

Teil (a) – Güte beweisen. Meist eines von drei Mustern:

- *OPT von unten abschätzen* über eine Struktur (z. B. ein Matching: jede Optimallösung braucht pro Matching-Kante einen eigenen Knoten).
- *Widerspruch + Zählen* (z. B. MAX-3-SAT: jede Klausel wird von einer der beiden Belegungen erfüllt).
- *Scheduling-Schranken*: $\text{OPT} \geq \frac{1}{m} \sum p_i$ und $\text{OPT} \geq p_{\max}$.

Teil (b) – Worst-Case-Instanz.

1. Zwingen den Algorithmus zu einer lokal guten, global schlechten Entscheidung (Greedy-Falle).
2. Parametrisiere die Instanz und berechne $A(I)$ und $\text{OPT}(I)$ *explizit*.
3. Zeige $A(I)/\text{OPT}(I) \rightarrow$ Schranke – mit *Grenzwert*.

6.2 Musteraufgabe

□ Aufgabe: MAX-3-SAT, Güte 2 (Hausaufgabe 13.1 = Klausur SS23)

Algorithmus A : werte die Formel unter β_0 (alle Variablen *falsch*) und β_1 (alle *wahr*) aus, gib die bessere zurück. (a) Zeige $v(A(\varphi)) \geq \frac{1}{2} v(\text{OPT}(\varphi))$. (b) Gib eine Formel an, bei der A genau die Hälfte des Optimums erreicht.

✓ Musterlösung

(a) **Güte-Beweis (Widerspruch + Zählen).** Sei φ eine Formel mit m Klauseln. *Schlüsselbeobachtung:* Jede Klausel enthält mindestens ein Literal – ist es positiv, wird die Klausel von β_1 (alle *wahr*) erfüllt; ist es negativ, von β_0 (alle *falsch*). Jede Klausel wird also von *mindestens einer* der beiden Belegungen erfüllt. Summiert man:

$$v(\beta_0) + v(\beta_1) \geq m.$$

Also erfüllt die bessere der beiden mindestens $m/2$ Klauseln: $v(A(\varphi)) = \max\{v(\beta_0), v(\beta_1)\} \geq \frac{m}{2}$. Da eine optimale Belegung höchstens alle m Klauseln erfüllt, ist $v(\text{OPT}(\varphi)) \leq m$. Zusammen:

$$v(A(\varphi)) \geq \frac{m}{2} \geq \frac{1}{2} v(\text{OPT}(\varphi)).$$

Somit hat A Güte 2.

(b) **Scharfe Instanz.** Wähle

$$\varphi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_0 \vee \neg x_2 \vee \neg x_3).$$

Optimum: die Belegung $x_0 = \text{falsch}$, $x_1 = x_2 = x_3 = \text{wahr}$ erfüllt *beide* Klauseln, also $\text{OPT} = 2$. Der Algorithmus aber: β_1 (alle *wahr*) erfüllt nur Klausel 1 (Klausel 2 hat lauter negative Literale, alle falsch); β_0 (alle *falsch*) erfüllt nur Klausel 2. Also $v(A(\varphi)) = \max\{1, 1\} = 1 = \frac{1}{2}\text{OPT}$. Die Schranke wird exakt erreicht. *Bauprinzip: eine Klausel nur positiv, eine nur negativ.* $\square$

6.3 Zweite Variante: Approximate Subset Sum (Klausur SS24-H)

□ Aufgabe: Greedy für Approximate Subset Sum

Maximiere $\sum_{j \in S} a_j \leq T$. Algorithmus GA: sortiere absteigend, nimm der Reihe nach, *stoppe* beim ersten Element, das nicht mehr passt. (a) Instanzen, die Güte 2 annähern; (b) zeige $\text{GA}(I) \geq \text{OPT}(I)/2$.

✓ Musterlösung

(a) **Worst-Case-Familie.** Für gerades T : $a_1 = \frac{T}{2} + 1$, $a_2 = a_3 = \frac{T}{2}$. GA nimmt zuerst a_1 (das größte) und stoppt, denn a_2 passt nicht mehr ($\frac{T}{2} + 1 + \frac{T}{2} > T$). Also $\text{GA} = \frac{T}{2} + 1$. Optimal sind $a_2 + a_3 = T$, also $\text{OPT} = T$. Rate: $\frac{T}{T/2+1} \rightarrow 2$ für $T \rightarrow \infty$.

(b) **Güte-Beweis.** Sei S die von GA gewählte Menge und a_ℓ das erste Element, das nicht mehr passte. Dann ist $\text{GA}(I) + a_\ell > T \geq \text{OPT}(I)$. Wegen der absteigenden Sortierung enthält S ein Element $a_j \geq a_\ell$, also $\text{GA}(I) \geq a_\ell$. Einsetzen:

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

somit $\text{GA}(I) \geq \text{OPT}(I)/2$. $\square$

6.4 Bewertung und häufige Fehler

Bewertung – worauf es ankommt

Beide Teile geben Punkte – der Güte-Beweis (a) und die Konstruktion (b). Bei (b) reicht keine einzelne Instanz: Gib eine *parametrisierte Familie* an und zeige den *Grenzwert* $A/OPT \rightarrow$ Schranke. $A(I)$ und $OPT(I)$ müssen *explizit* berechnet sein.

Stolperstein

In (a) die beiden OPT-Schranken vergessen (bei Scheduling: Durchschnittslast *und* größter Job). In (b) nur eine feste Instanz angeben statt einer Familie mit Grenzwert – dann ist die Schärfe nicht gezeigt.

7 Turingmaschine entwerfen

★ In diesem Kapitel

Der letzte Typ steht für sich – er braucht keine der anderen Techniken. Du konstruierst eine Turingmaschine für eine formale Sprache, begründest ihre Korrektheit und schätzt die Laufzeit ab. Manchmal kommt ein Vergleich mit dem RAM-Modell dazu.

7.1 Das Rezept

➤ Rezept: TM in drei Teilen

- 1. Die Maschine in Worten.** Beschreibe das Verhalten als nummerierte Schritte („lies das erste Symbol, laufe nach rechts bis ...“). Keine formale Zustandstabelle nötig – eine klare Wortbeschreibung genügt.
- 2. Korrektheit.** Warum akzeptiert die TM genau die Wörter der Sprache? Ein bis zwei Sätze über die Schleifeninvariante („nach jedem Durchlauf sind die äußeren Symbole abgearbeitet“).
- 3. Laufzeit.** Kosten pro Durchlauf $\times$ Anzahl Durchläufe. Denk an die *Randfälle* (leeres Wort, ein Symbol).

7.2 Musteraufgabe

Aufgabe: Palindrom-Erkenner (Hausaufgabe 10.2)

Konstruiere eine Turingmaschine für die Sprache $L = \{ w \mid w \text{ ist ein Palindrom} \}$ (liest sich vorwärts wie rückwärts). Gib Korrektheit und Laufzeit an.

✓ Musterlösung

Die Maschine.

1. Ist das Band leer, *akzeptiere* (das leere Wort ist ein Palindrom).
2. Steht nur noch ein Buchstabe, *akzeptiere*.
3. Lies den *ersten* Buchstaben, merke ihn im Zustand, laufe zum *letzten* Buchstaben. Sind die beiden *verschieden*, *verwirf*. Sind sie gleich, ersetze beide durch ein Leersymbol x (erst den letzten, dann laufe zurück zum ersten und lösche ihn).
4. Gehe zu Schritt 1.

Korrektheit. Die Maschine vergleicht wiederholt das äußerste Paar von Buchstaben und streicht es weg. Sie akzeptiert genau dann, wenn bei jedem Vergleich beide Enden gleich waren – also wenn w symmetrisch ist. Sie endet in der Mitte mit einem oder keinem Buchstaben (beides Palindrome). Bei Ungleichheit verwirft sie sofort.

Laufzeit. Ein Durchlauf (Schritte 1–3) kostet $O(n)$: der Kopf läuft einmal von links nach rechts und zurück. Pro Durchlauf werden zwei Buchstaben gestrichen, es gibt also höchstens $\lfloor n/2 \rfloor$ Durchläufe. Insgesamt $O(n) \cdot O(n) = O(n^2)$. $\square$

7.3 Zweite Variante: $L = \{0^{2^n}\}$ (Präsenz 10.3)

Etwas anspruchsvoller, mit RAM-Vergleich.

✓ Musterlösung

Die Maschine (akzeptiert Nullfolgen, deren Länge eine Zweierpotenz ist):

1. Steht genau eine 0 auf dem Band, *akzeptiere*.
2. Laufe über das Band und streiche *jede zweite* 0 (ersetze sie durch x).
3. War die Anzahl der 0en *ungerade* (die letzte gelesene war eine 0, die nicht gepaart wurde), *verwirf*.
4. Gehe zu Schritt 1.

Korrektheit. Eine Zahl ist genau dann eine Zweierpotenz, wenn wiederholtes Halbieren bei 1 endet, ohne je auf eine ungerade Zahl > 1 zu treffen. Genau das prüft die Maschine: jedes Streichen jeder zweiten 0 halbiert die Anzahl; eine ungerade Anzahl > 1 führt zum Verwerfen.

Laufzeit. Jeder Durchlauf kostet $O(n)$, und die Anzahl halbiert sich jedes Mal – also $O(\log n)$ Durchläufe. Insgesamt $O(n \log n)$.

RAM-Vergleich. Ein RAM-Modell könnte die 0en einfach zählen ($O(n)$) und dann in $O(\log n)$ prüfen, ob die Zahl eine Zweierpotenz ist – schneller, weil es beliebige Speicherzellen direkt adressiert, während die TM den Kopf über das Band bewegen muss. $\square$

7.4 Bewertung und häufige Fehler

Bewertung – worauf es ankommt

Punkteschema (Hausaufgabe 10.2): *TM-Beschreibung* 2 P., *Korrektheit* 1 P., *Laufzeit* 2 P. Die Laufzeit ist also die Hälfte der Punkte – rechne sie sauber vor (Kosten pro Durchlauf $\times$ Anzahl Durchläufe).

Stolperstein

Die Randfälle vergessen: leeres Wort, ein einzelnes Symbol. Und die Laufzeit nur behaupten statt herzuleiten – gerade hier steckt die halbe Punktzahl.