

Musterkatalog

25 Aufgaben, 26 Muster — eine Präsentator-Aufgabe je Muster
Analyse von Algorithmen und Komplexität – CAU Kiel

Inhalt

1	Beweismuster: einmal verstehen, überall anwenden	2
2	Konstruktions-Familien: an einem Beispiel lernen, dann ableiten	16
3	Verfahren und Einzelmuster: einzeln lernen	37

1 Beweismuster: einmal verstehen, überall anwenden

A1 Longest Path $\in$ NP (Dreischritt)

Klausur SS23-H, A2, 3+3+4 P.

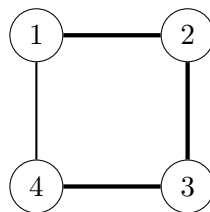
Problem Longest Path: Gegeben ein ungerichteter Graph $G = (V, E)$ und eine Zahl k . Gibt es einen *einfachen* Pfad in G (kein Knoten wird doppelt besucht) mit Länge $\geq k$?

Zeigen Sie auf verschiedene Weisen, dass das Problem in NP liegt:

- Geben Sie ein Zertifikat für LONGEST PATH an.
- Beschreiben Sie einen Verifizierer für LONGEST PATH und geben Sie dessen Laufzeit konkret an.
- Beschreiben Sie eine nichtdeterministische Turing-Maschine (in Worten), die LONGEST PATH löst. Geben Sie auch hier die Laufzeit konkret an.

Beispiel Longest Path $\in$ NP

Instanz. Das folgende Viereck mit Schranke $k = 3$ (gesucht: ein einfacher Pfad mit $k+1 = 4$ Knoten).



Kern. Zertifikat $(1, 2, 3, 4)$ (fetter Pfad): alle vier Knoten sind paarweise verschieden und jede der drei Pfadkanten existiert – der Verifizierer akzeptiert diesen einfachen Pfad der Länge $3 \geq k$. Der Rechenweg der NTM, der genau diese Folge rät, akzeptiert.

Muster $X \in$ NP beweisen

Ein Zertifikat wird geraten oder gegeben und in Polynomialzeit geprüft.

- Wähle das Zertifikat: die Lösung selbst (Pfad, Menge, Belegung).
- Schreibe den Verifizierer als Schrittliste: Größe prüfen, Eigenschaften prüfen, sonst akzeptiere.
- Korrektheit in zwei Sätzen: Ja-Instanz $\rightarrow$ akzeptiertes Zertifikat; Nein-Instanz $\rightarrow$ jede Prüfung schlägt fehl.
- Laufzeit: jede Prüfung mit Operation und $O(\cdot)$.
- Schluss: „Also gilt $X \in$ NP.“ (NDTM-Variante: Zertifikat im ersten Schritt nichtdeterministisch raten.)

Lösung.

a) Eine Folge von $k + 1$ paarweise verschiedenen Knoten $(v_0, \dots, v_k)$ – der behauptete Pfad.

b) Eingabe: Graph $G = (V, E)$, Zahl k und Zertifikat $(v_0, \dots, v_k)$.

Der Verifizierer arbeitet wie folgt:

- Falls das Zertifikat nicht aus $k + 1$ Knoten besteht, lehne ab.
- Prüfe, ob alle v_i paarweise verschieden sind; lehne sonst ab.
- Prüfe für jedes $i < k$, ob $\{v_i, v_{i+1}\} \in E$ gilt; lehne sonst ab.
- Sonst akzeptiere.

Korrektheit: Hat G einen einfachen Pfad der Länge $\geq k$, dann sind seine ersten $k + 1$ Knoten ein Zertifikat – beide Prüfungen gelingen. Hat G keinen solchen Pfad, dann scheitert jedes Zertifikat an einer Prüfung – der Verifizierer lehnt ab.

Laufzeit: Paarweise Vergleiche $O(|V|^2)$, Kantenprüfungen $O(k)$ – gesamt $O(|V|^2)$.

c) Die NTM arbeitet wie folgt:

- Rate nichtdeterministisch $k + 1$ Knoten $v_0, \dots, v_k$.
- Prüfe, ob alle geratenen Knoten paarweise verschieden sind; lehne sonst ab.
- Prüfe für jedes $i < k$, ob $\{v_i, v_{i+1}\} \in E$ gilt; lehne sonst ab.
- Sonst akzeptiere.

Korrektheit: Hat G einen einfachen Pfad der Länge $\geq k$, dann rät ein Rechenweg genau dessen erste $k + 1$ Knoten und akzeptiert. Sonst scheitert auf jedem Rechenweg eine Prüfung – die NTM lehnt ab.

Laufzeit: Rateschritte $O(k)$, paarweise Vergleiche $O(|V|^2)$, Kantenprüfungen $O(k)$ – gesamt $O(|V|^2)$.

Also gilt LONGEST PATH $\in$ NP. □

Sei $L \subseteq \Sigma^*$ eine Sprache, für die eine NDTM M über einem Alphabet $\Gamma \supset \Sigma$ und ein Polynom T existieren, sodass L von M in nicht-deterministischer Zeit T entschieden wird. Beweisen Sie: Es existiert ein polynomieller Verifizierer für L .

- a) Geben Sie dafür zunächst für die Sprache L , die durch die folgende NDTM M akzeptiert wird, einen polynomiellen Verifizierer an, der in analoger Weise arbeitet. Die NDTM M erhält als Eingabe einen ungerichteten Graphen $G = (V, E)$ und eine Zahl $k \in \mathbb{N}_0$.

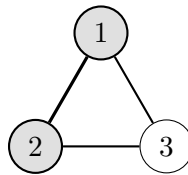
$M(G = (V, E), k)$

1. Rate nichtdeterministisch eine Menge von Knoten $C \subseteq V$.
2. Für alle $\{u, v\} \in E$: Falls $u \notin C$ und $v \notin C$, verwerfe.
3. Falls $|C| > k$, verwerfe.
4. Akzeptiere.

- b) Führen Sie nun einen allgemeingültigen Beweis.

Beispiel NDTM $\subseteq$ Verifizierer: VC am Dreieck, $k = 2$

Instanz. Das folgende Dreieck G mit $k = 2$:



Zertifikat: Knotenmenge $C = \{1, 2\}$
(Knoten 1, 2 hervorgehoben)

Kern. Die NDTM rät die Knotenmenge $C \subseteq V$; der Verifizierer erhält C als Zertifikat und prüft mit denselben Schritten 2–4 – das Raten ist durch das Zertifikat ersetzt. Für $C = \{1, 2\}$ ist jede Kante durch 1 oder 2 gedeckt und $|C| = 2 \leq k$ – er akzeptiert.

Muster Zertifikat $\leftrightarrow$ Rechenweg

Übersetzt zwischen den zwei NP-Definitionen: nichtdeterministische Entscheidungen und Zertifikat sind dasselbe Objekt.

- Richtung NDTM $\rightarrow$ Verifizierer: Zertifikat = Folge der nichtdeterministischen Entscheidungen.
- Der Verifizierer simuliert die NDTM deterministisch entlang dieser Folge.
- Rückrichtung Verifizierer $\rightarrow$ NDTM (analog): rate das Zertifikat, simuliere dann den Verifizierer.
- Korrektheit: akzeptierender Rechenweg $\leftrightarrow$ akzeptiertes Zertifikat.
- Laufzeit: Simulation Schritt für Schritt, bleibt polynomiell.

Lösung.

a) Eingabe: Graph $G = (V, E)$, Zahl k und Zertifikat c : eine Knotenmenge $C \subseteq V$.

Der Verifizierer $V(G, k; c)$ arbeitet wie folgt:

1. Für alle $\{u, v\} \in E$: Falls $u \notin C$ und $v \notin C$, verwerfe.
2. Falls $|C| > k$, verwerfe.
3. Akzeptiere.

Das Raten ist durch das Zertifikat ersetzt. Die Prüfschritte sind identisch zu den Schritten 2–4 der NDTM. **Laufzeit:** Jede Kante prüfen und $|C|$ zählen – $O(|V| + |E|)$.

b) Sei L von der NDTM M in nicht-deterministischer Zeit T akzeptiert.

Eingabe: Wort x und als Zertifikat eine Folge c von höchstens $T(|x|)$ nichtdeterministischen Entscheidungen.

Der Verifizierer $V(x, c)$ arbeitet wie folgt:

- Simuliere M auf x deterministisch, höchstens $T(|x|)$ Schritte.
- Bei jedem Schritt mit mehreren möglichen Übergängen wähle den Übergang, den die Entscheidungsfolge c vorgibt.
- Falls die Simulation akzeptierend endet, akzeptiere.
- Sonst lehne ab.

Korrektheit: Ist $x \in L$, dann hat M einen akzeptierenden Rechenweg mit höchstens $T(|x|)$ Schritten. Seine Entscheidungsfolge ist ein Zertifikat, das V akzeptiert.

Ist $x \notin L$, dann akzeptiert kein Rechenweg von M . Jede Entscheidungsfolge steuert die Simulation einen Rechenweg von M entlang – V lehnt jedes Zertifikat ab.

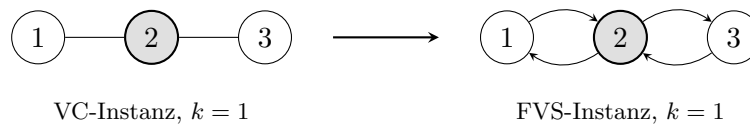
Laufzeit: Höchstens $T(|x|)$ Simulationsschritte mit konstantem Aufwand pro Schritt – $O(T(|x|))$, polynomiell. Das Zertifikat umfasst höchstens $T(|x|)$ Entscheidungen – polynomiell viele. Somit ist V ein polynomieller Verifizierer für L . $\square$

Zeigen Sie: Ist L_0 NP-vollständig, gilt $L_0 \leq_p L_1$ und ist $L_1 \in \text{NP}$, so ist auch L_1 NP-vollständig.

Hinweis: Die Relation $\leq_p$ ist transitiv.

Beispiel Vererbung: von VertexCover zu FeedbackVertexSet

Instanz. $L_0 = \text{VERTEXCOVER}$ (NP-vollständig), $L_1 = \text{FEEDBACKVERTEXSET} \in \text{NP}$; die Reduktion $\text{VERTEXCOVER} \leq_p \text{FEEDBACKVERTEXSET}$ macht jede Kante zu zwei antiparallelen Bögen:



Kern. Jedes $L \in \text{NP}$ reduziert auf VERTEXCOVER , und mit $\text{VERTEXCOVER} \leq_p \text{FEEDBACKVERTEXSET}$ folgt per Transitivität $L \leq_p \text{FEEDBACKVERTEXSET}$. Also ist FEEDBACKVERTEXSET NP-schwer und wegen $\text{FEEDBACKVERTEXSET} \in \text{NP}$ NP-vollständig.

Muster Reduktionen verketteten

Meta-Beweise über NP: bekannte Reduktionen einsetzen und verketteten, nie neu konstruieren.

- Schreibe die Voraussetzungen als Reduktionen hin ($L' \leq_p L$, $L \leq_p X$, $L \in P$, ...).
- Verkette per Transitivität von $\leq_p$: $f \circ g$ ist wieder eine Reduktion – Polynom in Polynom bleibt Polynom.
- Folgere die Behauptung Definition für Definition.

Lösung.

Zu zeigen sind die beiden Teile der NP-Vollständigkeit: $L_1 \in \text{NP}$ (gilt nach Voraussetzung) und $L \leq_p L_1$ für jedes $L \in \text{NP}$.

Beweis der NP-Schwere: Sei $L \in \text{NP}$ beliebig.

- Da L_0 NP-vollständig ist, reduziert jedes NP-Problem auf L_0 – insbesondere $L \leq_p L_0$.
- Nach Voraussetzung gilt $L_0 \leq_p L_1$.
- Da $\leq_p$ transitiv ist, folgt $L \leq_p L_1$.

Da L beliebig gewählt war, reduziert jedes NP-Problem auf L_1 – L_1 ist NP-schwer. Zusammen mit $L_1 \in \text{NP}$ ist L_1 NP-vollständig. $\square$

Problem k -Clique:

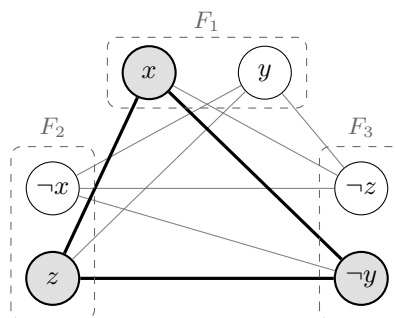
Eingabe: Ein ungerichteter Graph $G = (V, E)$ und eine Zahl $k \geq 1$. Eine Clique ist eine Teilmenge $C \subseteq V$ mit $\{u, v\} \in E$ für alle $u, v \in C$ mit $u \neq v$.

Entscheide: Hat der gegebene Graph G eine Clique $C \subseteq V$ mit mindestens k Knoten (d. h. mit $|C| \geq k$)?

Beweisen Sie: Das Problem k -CLIQUE ist NP-vollständig. *Hinweis: Die Reduktion geht von allgemeinem SAT aus – NICHT von 3-SAT.*

Beispiel k -Clique: Reduktionsgraph und Zertifikat

Instanz. $F_1 = (x \vee y)$, $F_2 = (\neg x \vee z)$, $F_3 = (\neg y \vee \neg z)$, $k = m = 3$.



Kern. Je Literalvorkommen ein Knoten. Kanten verbinden Literale verschiedener Klauseln, die sich nicht widersprechen. Die Belegung $x = z = \text{wahr}$, $y = \text{falsch}$ macht pro Klausel ein Literal wahr – deren Knoten $\{x, z, \neg y\}$ sind die fette 3-Clique und das akzeptierte Zertifikat.

Muster Reduktions-Gerüst (NP-Vollständigkeit)

Das Gerüst jedes NP-Vollständigkeits-Beweises – die Konstruktion wechselt, das Gerüst nie.

- Teil 1: $X \in \text{NP}$ (Muster „ $X \in \text{NP}$ beweisen“, kurz).
- Teil 2: bekanntes NP-vollständiges Problem wählen, Konstruktion $\text{BEKANNT} \leq_p X$ angeben.
- Beweis hin: aus einer Lösung von BEKANNT eine Lösung von X bauen.
- Beweis zurück: aus einer Lösung von X eine Lösung von BEKANNT gewinnen.
- Laufzeit der Transformation: Operation + $O(\cdot)$.
- Schluss: $X \in \text{NP}$ und NP-schwer $\Rightarrow$ NP-vollständig.

Muster Auswahl-Graph

Konstruktion für „wähle je Gruppe einen Vertreter“: Knoten = Möglichkeiten, Kanten = Verträglichkeit.

- Pro Gruppe (Klausel) ein Knoten je Möglichkeit (Literalvorkommen).

- Kante zwischen zwei Knoten, wenn sie gemeinsam wählbar sind – nie innerhalb einer Gruppe.
- Ziel k = Anzahl der Gruppen.
- Lösung $\leftrightarrow$ Clique: je Gruppe ein Vertreter, paarweise verträglich.

Lösung.

∈ **NP**: Eingabe: Graph $G = (V, E)$, Zahl k und Zertifikat $C \subseteq V$.

Der Verifizierer arbeitet wie folgt:

- Falls $|C| < k$, lehne ab.
- Prüfe für jedes Paar $\{u, v\} \subseteq C$, ob $\{u, v\} \in E$ gilt; lehne sonst ab.
- Sonst akzeptiere.

Existiert eine k -Clique, dann gibt es ein Zertifikat dafür. Dieses wird akzeptiert. Sonst verletzt jedes Zertifikat eine Prüfung – abgelehnt.

Laufzeit: Größe zählen $O(|V|)$, alle Paare testen $O(|V|^2)$ – gesamt $O(|V|^2)$. Also gilt k -CLIQUE ∈ NP.

NP-schwer durch SAT $\leq_p k$ -CLIQUE:

Konstruktion: Sei $F = F_1 \wedge \dots \wedge F_m$ eine KNF-Formel. Wir konstruieren daraus eine k -CLIQUE-Instanz durch:

- $V = \{[i, j] \mid j\text{-tes Literal in Klausel } i\}$; y_{ij} bezeichne dieses Literal
- $E = \{\{[i, j], [i', j']\} \mid i \neq i', y_{ij} \neq \neg y_{i'j'}\}$
- $k = m$

Beweis hin:

- Sei ψ eine erfüllende Belegung von F .
- Dann ist in jeder Klausel mindestens ein Literal wahr. Wähle pro Klausel eines.
- C sei die Menge der m zugehörigen Knoten.
- Je zwei Knoten aus C stammen aus verschiedenen Klauseln.
- Ihre Literale widersprechen sich nicht, denn beide sind unter ψ wahr.
- Nach Definition von E sind sie also verbunden.
- Also ist C eine m -Clique und (G, m) eine Ja-Instanz.

Beweis zurück:

- Sei C eine m -Clique in G .
- Knoten derselben Klausel sind nie verbunden. Also enthält C aus jeder der m Klauseln genau einen Knoten.

- Die zugehörigen Literale widersprechen sich paarweise nicht (sonst fehlte eine Kante).
- Setze alle diese Literale wahr – eine widerspruchsfreie Belegung.
- Sie erfüllt in jeder Klausel ein Literal.
- Also ist F erfüllbar.

Laufzeit: Sei L die Zahl der Literalvorkommen. Knoten anlegen $O(L)$, Kanten prüfen $O(L^2)$ –
gesamt $O(L^2)$.

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

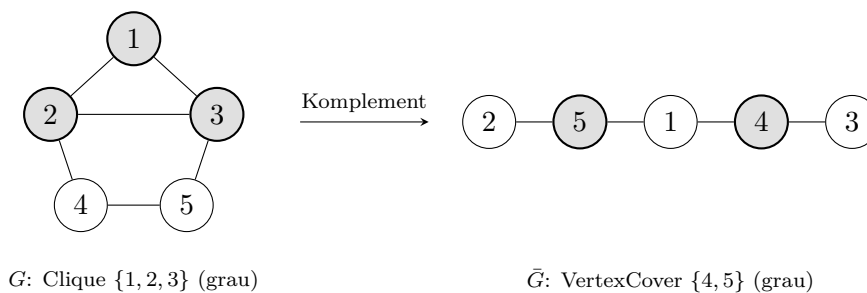
Betrachten Sie die Reduktion $\text{CLIQUE} \leq_p \text{VERTEXCOVER}$ (Komplementgraph, $k' = n - k$).

1. Geben Sie $|V'|$, $|E'|$ und k' in Abhängigkeit der Clique-Instanz an.
2. Zeigen Sie Lower Bounds für VERTEXCOVER bzgl. $|V'|$, $|E'|$ und k' basierend auf der ETH.

Hinweis: Unter der ETH ist CLIQUE nicht in $2^{o(|V|)} \cdot |I|^{O(1)}$ lösbar (Vorlesung).

Beispiel Lower Bounds VertexCover: konkrete Zahlen

Instanz. CLIQUE-Instanz G mit $|V| = 5$, $|E| = 6$, $k = 3$. Die Reduktion bildet den Komplementgraphen $\bar{G}$ mit $k' = |V| - k$.



Kern. Die VC-Instanz hat $|V'| = 5$, $|E'| = \binom{5}{2} - 6 = 4$ und $k' = 5 - 3 = 2$. Wegen $|V'| = |V|$ wäre ein $2^{o(|V'|)}$ -Algorithmus für VERTEXCOVER zugleich ein $2^{o(|V|)}$ -Algorithmus für CLIQUE – Widerspruch zur ETH. Kein $2^{o(|V'|)}$.

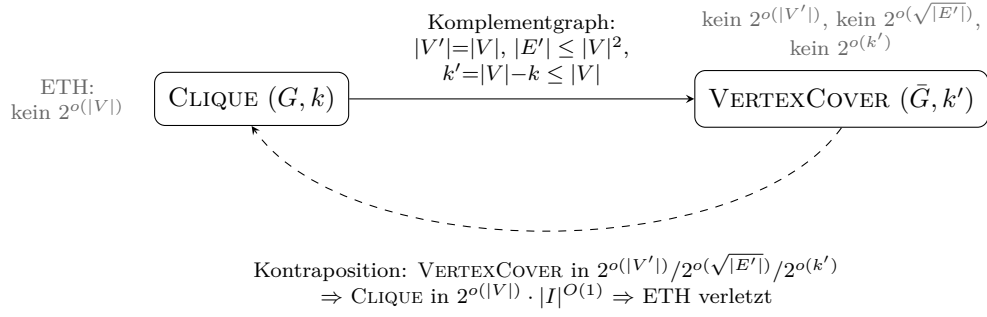
Muster ETH-Kontraposition

Überträgt die ETH-Schranke entlang einer Reduktion: ein schneller Algorithmus fürs Ziel wäre ein schneller Algorithmus für die Quelle.

- Parameter der Zielinstanz zählen – als Funktion der Quellparameter.
- Angenommen, das Ziel wäre in $2^{o(\text{param})} \cdot |I|^{O(1)}$ lösbar.
- Parameter einsetzen: daraus folgt $2^{o(\text{Quellparameter})}$ für die Quelle.
- Bei quadratischem Blowup: Wurzel im Exponenten ansetzen ($\sqrt{\text{param}} = O(\cdot)$).
- Widerspruch zur bekannten Schranke (ETH bzw. Hinweis). Bound notieren.

Lösung.

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



2. Für CLIQUE gilt unter der ETH: kein $2^{o(|V|)} \cdot |I|^{O(1)}$ -Algorithmus.

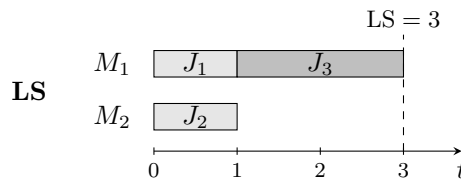
- **Bzgl. $|V'|$:** Angenommen, VERTEXCOVER wäre in $2^{o(|V'|)} \cdot |I|^{O(1)}$ lösbar. Wegen $|V'| = |V|$ ergäbe sich ein $2^{o(|V|)}$ -Algorithmus für CLIQUE. Widerspruch zur ETH.
- **Bzgl. $|E'|$:** Angenommen, VERTEXCOVER wäre in $2^{o(\sqrt{|E'|})} \cdot |I|^{O(1)}$ lösbar. Wegen $|E'| \leq |V|^2$ ist $\sqrt{|E'|} = O(|V|)$ – es ergäbe sich ein $2^{o(|V|)}$ -Algorithmus für CLIQUE. Widerspruch zur ETH.
- **Bzgl. k' :** Angenommen, VERTEXCOVER wäre in $2^{o(k')} \cdot |I|^{O(1)}$ lösbar. Wegen $k' \leq |V|$ ergäbe sich ein $2^{o(|V|)}$ -Algorithmus für CLIQUE. Widerspruch zur ETH.

□

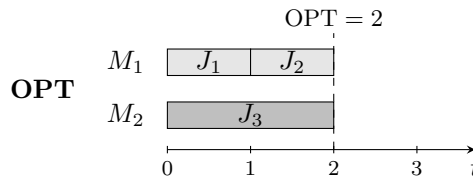
$P \parallel C_{\max}$: n Jobs mit Zeiten $p_1, \dots, p_n$, m Maschinen; minimiere $C_{\max}$. ListScheduling legt jeden Job der Reihe nach auf die Maschine mit minimaler Last. Zeigen Sie: Für alle I gilt $LS(I)/OPT(I) \leq 2 - 1/m$.

Beispiel ListScheduling-Güte $2 - \frac{1}{m}$ scharf: $m = 2$, Jobs $(1, 1, 2)$

Instanz. $m = 2$ Maschinen, Jobs $(1, 1, 2)$. ListScheduling legt jeden Job auf die Maschine kleinster Last: $J_1 \rightarrow M_1$, $J_2 \rightarrow M_2$, $J_3 \rightarrow M_1 \Rightarrow LS = 3$.



Kern. Das Optimum legt J_1, J_2 auf eine Maschine und J_3 auf die andere: $OPT = 2$. Also $LS/OPT = \frac{3}{2} = 2 - \frac{1}{m}$ – die Schranke ist *scharf*:



Muster Güte-Rahmen

Jeder Güte-Beweis klemmt den Algorithmuswert zwischen eine Schranke und $c \cdot OPT$.

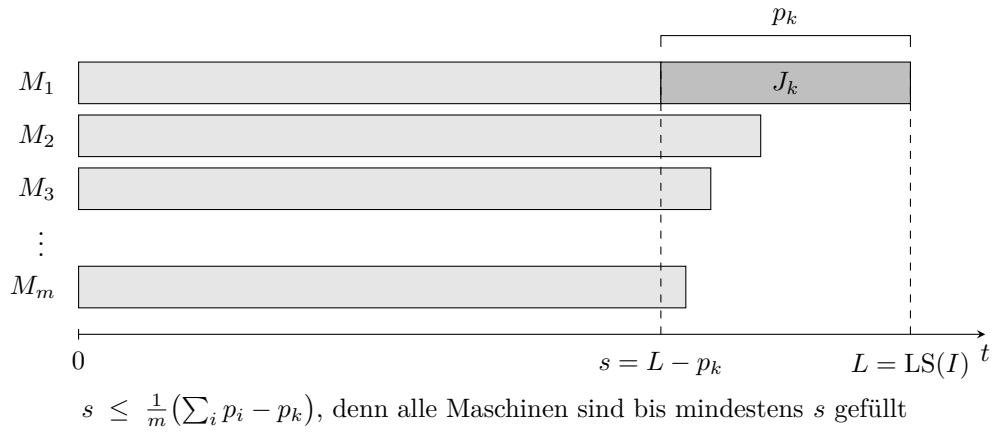
- ALG nach oben abschätzen – am kritischen Moment der Ausführung argumentieren.
- Untere Schranken für OPT sammeln (z.B. $OPT \geq \frac{1}{m} \sum p_j$, $OPT \geq p_{\max}$).
- Die ALG-Schranke durch die OPT -Schranken ausdrücken.
- Kombinieren: $ALG \leq c \cdot OPT$; c ist die Güte.

Lösung.

O.B.d.A. habe Maschine M_1 am Ende die höchste Last $L = LS(I)$. Sei J_k der letzte Job, der M_1 zugeordnet wurde.

Kernbeobachtung: Bei der Zuordnung von J_k hatte M_1 die *kleinste* Last $L - p_k$ – also hatten alle m Maschinen mindestens Last $L - p_k$. Daraus folgt

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



Zwei Schranken für das Optimum: $\text{OPT}(I) \geq \frac{1}{m} \sum_i p_i$ (Gesamtlast auf m Maschinen) und $\text{OPT}(I) \geq p_k$ (jeder Job muss irgendwo laufen).

Kombination:

$$\text{OPT}(I) \geq \frac{1}{m} \sum_i p_i \geq \frac{m(L - p_k) + p_k}{m} = L - \left(1 - \frac{1}{m}\right) p_k.$$

Mit $p_k \leq \text{OPT}(I)$: $\text{OPT}(I) \geq \text{LS}(I) - \left(1 - \frac{1}{m}\right) \text{OPT}(I)$, also $\text{LS}(I) \leq \left(2 - \frac{1}{m}\right) \text{OPT}(I)$. □

ListScheduling hat Güte $2 - \frac{1}{m}$.

- Geben Sie eine Instanz für $m = 2$ an, bei der ListScheduling die Güte $2 - \frac{1}{m}$ erreicht (mit OPT und LS-Wert).
- Dasselbe für $m = 3$.

Hinweis: ListScheduling legt jeden Job in der gegebenen Reihenfolge auf die aktuell am wenigsten belastete Maschine.

Muster Scharfe Instanz bauen

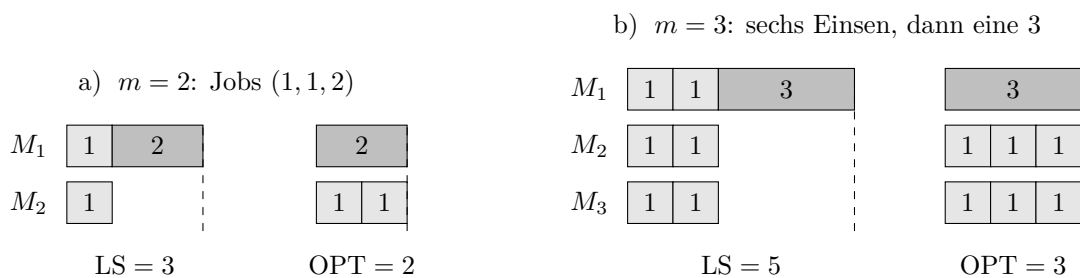
Baut eine Instanz, die die Güte-Schranke exakt erreicht.

- Zwingen den Algorithmus, alle Maschinen gleich zu füllen.
- Setze zum Schluss einen großen Brocken obendrauf.
- OPT verteilt anders: Brocken allein, Kleinteile gleichmäßig.
- Rate ALG/OPT ausrechnen – sie trifft die Schranke exakt.

Lösung.

a) $m = 2$: Jobs $(1, 1, 2)$ in dieser Reihenfolge. LS verteilt die Einsen auf beide Maschinen, die 2 kommt obendrauf: $\text{LS} = 3$. Optimal: $\{2\} / \{1, 1\}$ mit $\text{OPT} = 2$. Rate $3/2 = 2 - \frac{1}{2}$.

b) $m = 3$: sechs Einser-Jobs, dann ein Job der Größe 3. LS verteilt die Einsen gleichmäßig (je 2), die 3 kommt obendrauf: $\text{LS} = 5$. Optimal: 3 allein, die Einsen auf zwei Maschinen: $\text{OPT} = 3$. Rate $5/3 = 2 - \frac{1}{3}$.



□

Wahr oder falsch?

- a) Für $L \in \text{NP}$ gilt: aus $L \leq_p 3\text{-SAT}$ folgt, dass L NP-vollständig ist.
- b) Gilt $L \leq_p 3\text{-SAT}$ und $3\text{-SAT} \leq_p L$, dann ist L NP-vollständig.
- c) Sei L NP-vollständig. Dann gilt $L \in P \iff P = \text{NP}$.
- d) Es ist möglich, dass $3\text{-SAT} \in P$ und $\text{CLIQUE} \notin P$.

Muster Implikationslogik

Wahr/Falsch über P, NP und Reduktionen: Definitionen einsetzen, Richtungen prüfen.

- Aussage in Definitionen übersetzen ($\leq_p$, NP-schwer, NP-vollständig).
- Richtung der Reduktion prüfen: wer reduziert auf wen?
- Für „falsch“: Gegenbeispiel (triviale Sprache $\emptyset/\Sigma^*$, offene $P \neq \text{NP}$ -Frage) oder Annahme per bekannter Reduktion zum Widerspruch führen.
- Für „wahr“: mit dem Verkettungs-Muster in 2–3 Sätzen beweisen.

Lösung.

- a) Falsch. Gegenbeispiel $L = \emptyset \in \text{NP}$: $L \leq_p 3\text{-SAT}$ sagt nur, dass sich L auf 3-SAT reduzieren lässt – nicht umgekehrt, also keine Schwere.
- b) Wahr. $3\text{-SAT} \leq_p L$ gibt die NP-Schwere: 3-SAT ist NP-vollständig und $\leq_p$ ist transitiv. $L \leq_p 3\text{-SAT}$ gibt $L \in \text{NP}$. Das ist die Definition von NP-vollständig.
- c) Wahr. Jedes $L' \in \text{NP}$ reduziert auf L ; liegt L in P , dann auch jedes L' – also $P = \text{NP}$. Umgekehrt folgt aus $P = \text{NP}$ sofort $L \in P$, da $L \in \text{NP}$.
- d) Falsch. $\text{CLIQUE} \in \text{NP}$ reduziert auf das NP-vollständige 3-SAT; aus $3\text{-SAT} \in P$ folgt also $\text{CLIQUE} \in P$. $\square$

2 Konstruktions-Familien: an einem Beispiel lernen, dann ableiten

A9 FeedbackVertexSet ist NP-vollständig

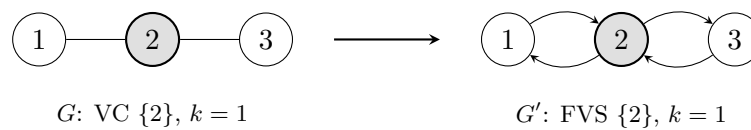
Hausaufgaben 9.2 und 10.1, je 5 P.

Problem FeedbackVertexSet: 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, dass FEEDBACKVERTEXSET NP-vollständig ist. *Hinweis:* Reduzieren Sie VERTEXCOVER auf FEEDBACKVERTEXSET.

Beispiel FeedbackVertexSet NP-vollständig

Instanz. VERTEXCOVER am Pfad 1-2-3 mit $k = 1$.



Kern. Jede Kante $\{u, v\}$ wird zu zwei antiparallelen Bögen $(u, v), (v, u)$; k bleibt. In G' entstehen die 2-Kreise $1 \rightleftarrows 2$ und $2 \rightleftarrows 3$; das Vertex Cover $\{2\}$ trifft beide, ist also ein Feedback Vertex Set mit $|X| = 1 \leq k$.

Muster Kanten-Gadget

Reduktion durch lokalen Ersatz: jede Kante wird durch ein kleines Bauteil ersetzt.

- Knoten übernehmen; pro Kante ein festes Gadget einsetzen (zwei Bögen, ein Dreieck, ...).
- Parameter k unverändert oder einfach angepasst.
- Beweis hin: die Lösung deckt alle Gadgets, weil sie alle Kanten deckt.
- Beweis zurück: das Gadget erzwingt einen Kantenendpunkt – ggf. Austauschargument (Neuknoten durch Altknoten ersetzen).
- Laufzeit: pro Kante konstant viel – $O(|E|)$.

Lösung.

∈ **NP**: Eingabe: gerichteter Graph $G = (V, E)$, Zahl k und Zertifikat $X \subseteq V$. Der Verifizierer arbeitet wie folgt:

- Falls $|X| > k$, lehne ab.
- Prüfe per topologischer Sortierung, ob $G \setminus X$ kreisfrei ist.
- Lehne ab, falls $G \setminus X$ einen Kreis enthält.
- Sonst akzeptiere.

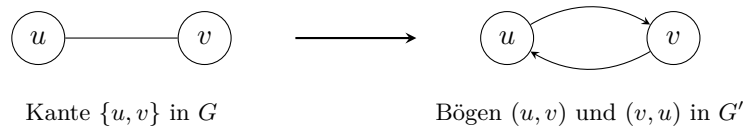
Existiert ein Feedback Vertex Set X mit $|X| \leq k$, dann gibt es ein Zertifikat dafür. Dieses wird vom Verifizierer akzeptiert und sonst abgelehnt. Größenprüfung und topologische Sortierung laufen in $O(|V| + |E|)$. Also gilt $\text{FEEDBACKVERTEXSET} \in \text{NP}$.

NP-schwer: Zeige, dass FEEDBACKVERTEXSET NP-schwer ist durch die Reduktion $\text{VERTEXCOVER} \leq_p \text{FEEDBACKVERTEXSET}$. VERTEXCOVER ist bereits als NP-vollständig bekannt.

Konstruktion: Sei $(G = (V, E), k)$ eine VERTEXCOVER -Instanz. Wir konstruieren daraus eine FEEDBACKVERTEXSET -Instanz $(G' = (V', E'), k')$ durch:

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

So wird jede ungerichtete Kante zu zwei antiparallelen Bögen.



Beweis VertexCover nach FeedbackVertexSet:

- Sei C ein Vertex Cover von G mit $|C| \leq k$.
- Somit hat jede Kante aus G einen Endpunkt in C .
- Also hat $G \setminus C$ keine Kanten mehr.
- Dann hat auch $G' \setminus C$ keine Bögen und damit keine Kreise.
- Also ist C ein Feedback Vertex Set für G' mit $|C| \leq k'$.

Beweis FeedbackVertexSet nach VertexCover:

- Sei X ein Feedback Vertex Set von G' mit $|X| \leq k'$.
- Dann ist $G' \setminus X$ kreisfrei.
- Angenommen, X wäre kein Vertex Cover.
- Dann gäbe es eine Kante $\{u, v\} \in E$ mit $u, v \notin X$.
- Somit wären u und v noch in $G' \setminus X$ vorhanden.
- Die Bögen (u, v) und (v, u) würden einen Kreis bilden.
- Das wäre ein Widerspruch dazu, dass $G' \setminus X$ kreisfrei ist.
- Also ist X ein Vertex Cover von G mit $|X| \leq k$.

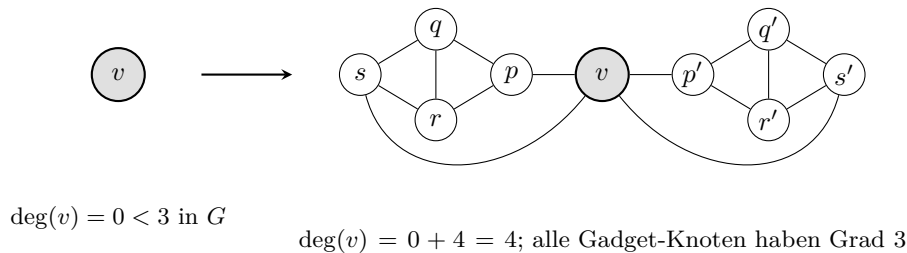
Laufzeit: Pro Kante zwei antiparallele Bögen anlegen – $O(|E|)$.

Da $\text{FEEDBACKVERTEXSET} \in \text{NP}$ und NP-schwer ist, folgt: FEEDBACKVERTEXSET ist NP-vollständig. □

Beweisen Sie die NP-Vollständigkeit von 3-COLOR, wobei jeder Knoten im Graphen mindestens Grad 3 hat.

Beispiel 3-COLOR mit Minimalgrad 3

Instanz. 3-COLOR am Graphen G aus dem einzelnen Knoten v , also $\deg(v) = 0 < 3$.



Kern. An jeden Knoten mit $\text{Grad} < 3$ werden zwei Diamant-Gadgets (K_4 ohne die Kante $\{p, s\}$) über $\{v, p\}$ und $\{v, s\}$ angehängt. Eine 3-Färbung überträgt sich auf G' : $f(v) = 1$, je Gadget $p = s = 2, q = r = 3$.

Muster Knoten-Gadget

Reduktion durch lokalen Ersatz am Knoten: jeder Knoten bekommt ein Bauteil, das eine Zusatzeigenschaft erzwingt.

- Pro Knoten ein festes Gadget anhängen oder ersetzen (Diamant, Zwilling, $K_{4,3}$).
- Das Gadget erzwingt lokal die Zusatzbedingung (Minimalgrad, Portale, Pfadenden).
- Beweis hin: die Lösung setzt sich aufs Gadget fort – Gadget separat lösen.
- Beweis zurück: die Gadget-Eigenschaft zwingt die Lösung aufs Original zurück.
- Laufzeit: pro Knoten konstant viel – $O(|V|)$.

Lösung.

∈ **NP**: Eingabe: Graph $G = (V, E)$ und Zertifikat $f : V \rightarrow \{1, 2, 3\}$. Der Verifizierer arbeitet wie folgt:

- Prüfe für jede Kante $\{u, v\} \in E$, ob $f(u) \neq f(v)$ gilt.
- Lehne ab, falls eine Kante gleich gefärbte Endpunkte hat.
- Sonst akzeptiere.

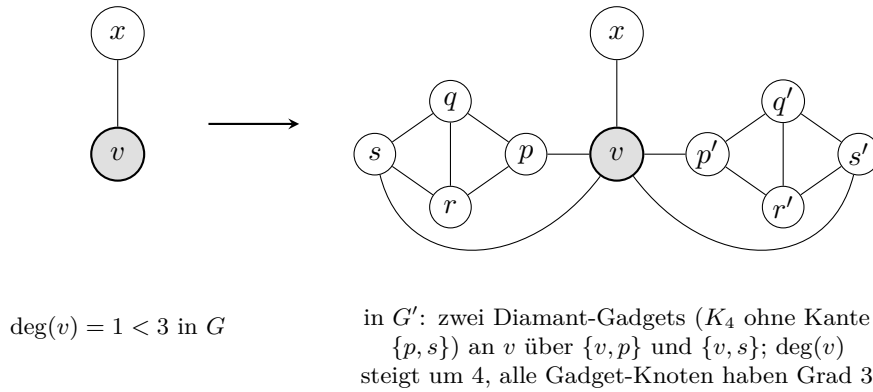
Existiert eine 3-Färbung, dann gibt es ein Zertifikat dafür. Dieses wird vom Verifizierer akzeptiert und sonst abgelehnt. Prüfung aller Kanten – $O(|E|)$. Also liegt die Grad-3-Variante in **NP**.

NP-schwer: Zeige die NP-Schwere durch Reduktion von 3-COLOR auf die Grad-3-Variante. 3-COLOR ist bereits als NP-vollständig bekannt.

Konstruktion: Sei $G = (V, E)$ eine 3-COLOR-Instanz. Wir konstruieren daraus einen Graphen G' mit Minimalgrad 3 durch:

- Benutze das Diamant-Gadget D : Knoten p, q, r, s mit den Kanten pq, pr, qr, qs, rs , also K_4 ohne die Kante $\{p, s\}$.
- Hänge an jeden Knoten v mit $\deg(v) < 3$ zwei eigene Gadgets an.
- Verbinde v je Gadget über die Kanten $\{v, p\}$ und $\{v, s\}$.

Dann steigt $\deg(v)$ um 4. Alle Gadget-Knoten haben Grad 3: q und r über drei Gadget-Kanten, p und s über zwei Gadget-Kanten plus die Kante zu v .



Beweis 3-COLOR nach Grad-3-Variante:

- Sei f eine 3-Färbung von G .
- Erweitere f auf jedes Gadget an v : Setze $p = s = c$ für eine Farbe $c \neq f(v)$.
- Das ist erlaubt, denn p und s sind nicht benachbart.
- q und r sind zu p und s adjazent, aber p und s tragen dieselbe Farbe.
- Somit bleiben für q und r zwei Farben frei.
- Färbe q und r mit diesen beiden Farben.
- Also sind alle Kanten von G' gültig gefärbt, und G' ist 3-färbbar.

Beweis Grad-3-Variante nach 3-COLOR:

- Sei f' eine 3-Färbung von G' .
- Die angehängten Gadgets lassen die Originalkanten unverändert, also gilt $E \subseteq E'$.
- Somit ist die Einschränkung von f' auf V eine 3-Färbung von G .

Laufzeit: Pro untergradigem Knoten zwei Gadgets konstanter Größe anlegen – $O(|V|)$.

Da die Grad-3-Variante in NP liegt und NP-schwer ist, folgt: Sie ist NP-vollständig. $\square$

Beweisen Sie: 3-SAT ist NP-vollständig. Zeigen Sie dafür $\text{SAT} \leq_p \text{3-SAT}$; $\text{3-SAT} \in \text{NP}$ dürfen Sie annehmen.

Beispiel SAT $\leq_p$ 3-SAT: eine 5er-Klausel zerlegt

Instanz. Die Klausel $C = (y_1 \vee y_2 \vee y_3 \vee y_4 \vee y_5)$, also $\ell = 5$ mit neuen Variablen x_1, x_2 .

Kern. C wird ersetzt durch die Kette

$$\underbrace{(y_1 \vee y_2 \vee x_1)}_{K_1} \wedge \underbrace{(\neg x_1 \vee y_3 \vee x_2)}_{K_2} \wedge \underbrace{(\neg x_2 \vee y_4 \vee y_5)}_{K_3}.$$

Ist C erfüllt, etwa durch y_3 , dann erfüllt $x_1 = \text{wahr}$, $x_2 = \text{falsch}$ die Kette: K_1 durch x_1 , K_2 durch y_3 , K_3 durch $\neg x_2$.

Muster Klausel-Splitting

Lange Klauseln in Dreier-Klauseln zerlegen: Hilfsvariablen reichen die Wahrheit weiter.

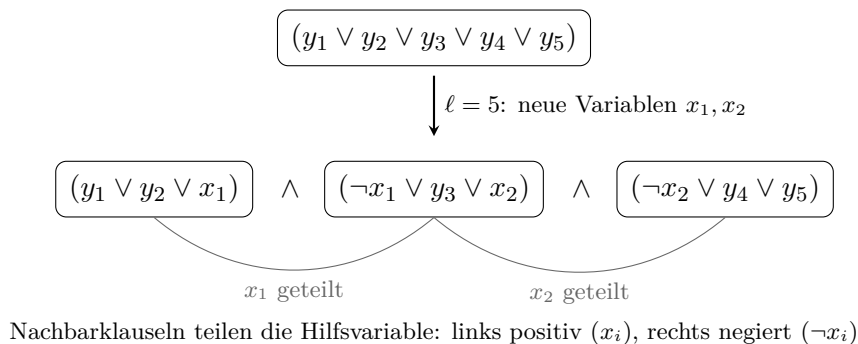
- Lange Klauseln in Ketten zerlegen; kurze Klauseln (≤ 3 Literale) bleiben unverändert.
- Frische Hilfsvariablen verbinden die Kettenglieder.
- Beweis hin: ein wahres Originalliteral macht die Kette erfüllbar – Hilfsvariablen passend setzen.
- Beweis zurück: Hilfsvariablen allein erfüllen nicht alle Glieder – ein Originalliteral ist wahr.

Lösung.

Konstruktion: Ersetze jede Klausel $(y_1 \vee \dots \vee y_\ell)$ mit $\ell > 3$ durch die Kette

$$(y_1 \vee y_2 \vee x_1) \wedge (\neg x_1 \vee y_3 \vee x_2) \wedge \dots \wedge (\neg x_{\ell-3} \vee y_{\ell-1} \vee y_\ell)$$

mit neuen Hilfsvariablen $x_1, \dots, x_{\ell-3}$ (pro Klausel eigene). Kurze Klauseln bleiben unverändert.



Beweis hin:

- Sei die Originalklausel erfüllt, etwa y_i wahr.

- Setze $x_1 = \dots = x_{i-2} = \text{wahr}$ und $x_{i-1} = \dots = x_{\ell-3} = \text{falsch}$.
- Dann sind die Kettenklauseln vor der Klausel mit y_i durch ihr x -Literal erfüllt.
- Die Klausel mit y_i ist durch y_i erfüllt.
- Die restlichen Kettenklauseln sind durch ihr $\neg x$ -Literal erfüllt.
- Also ist jede Kettenklausel erfüllt.

Beweis zurück:

- Sei die Kette erfüllt.
- Angenommen, alle y_i wären falsch.
- Dann erzwingt die erste Klausel $x_1 = \text{wahr}$.
- Damit erzwingt die zweite Klausel $x_2 = \text{wahr}$, induktiv folgt $x_{\ell-3} = \text{wahr}$.
- Dann ist die letzte Klausel $(\neg x_{\ell-3} \vee y_{\ell-1} \vee y_{\ell})$ falsch – Widerspruch.
- Also ist ein y_i wahr und die Originalklausel erfüllt.

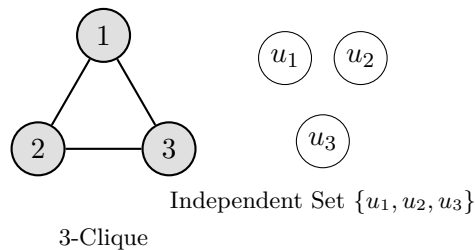
Laufzeit: Jede Klausel der Länge $\ell \geq 3$ wird zu $\ell - 2$ Dreier-Klauseln, kurze bleiben unverändert – gesamt $O(n)$ bei Eingabelänge n .

Da SAT NP-vollständig ist, $\text{SAT} \leq_p \text{3-SAT}$ gilt und $\text{3-SAT} \in \text{NP}$, ist 3-SAT NP-vollständig. $\square$

Problem CliqueAndIndependentSet: Gegeben $G = (V, E)$ und $k \geq 1$. Gibt es in G sowohl ein Independent Set (eine Menge von Knoten, die paarweise nicht adjazent sind, d. h. zwischen denen es keine Kanten gibt) der Größe k als auch eine Clique der Größe k ? Zeigen Sie die NP-Schwere durch Reduktion eines bekannten NP-schweren Problems.

Beispiel CliqueAndIndependentSet NP-schwer

Instanz. CLIQUE am Dreieck $\{1, 2, 3\}$ mit $k = 3$.



Kern. Es kommen $k = 3$ neue isolierte Knoten u_1, u_2, u_3 hinzu; k bleibt. In G' ist $\{1, 2, 3\}$ weiterhin eine 3-Clique, und $\{u_1, u_2, u_3\}$ ist ein Independent Set der Größe 3.

Muster Knoten anhängen (Padding)

Neue Knoten oder Teile anhängen, die die Zusatzforderung gratis erfüllen.

- Originalinstanz unverändert lassen; passende Teile anhängen (isolierte Knoten, universeller Knoten, K_k).
- Parameter anpassen (k bleibt, $k + 1, \dots$).
- Beweis hin: Originallösung + angehängte Teile = neue Lösung.
- Beweis zurück: die angehängten Teile leisten nicht mehr als geplant – die Lösung steckt im Original. Ggf. Austauschargument.

Lösung.

NP-schwer: Zeige, dass CLIQUEANDINDEPENDENTSET NP-schwer ist durch die Reduktion $\text{CLIQUE} \leq_p \text{CLIQUEANDINDEPENDENTSET}$. CLIQUE ist bereits als NP-vollständig bekannt, also insbesondere NP-schwer.

Konstruktion: Sei $(G = (V, E), k)$ eine CLIQUE-Instanz, o.B.d.A. $k \geq 2$. (Bei $k = 1$ ist ein einzelner Knoten zugleich Clique und Independent Set.) Wir konstruieren daraus eine CLIQUEANDINDEPENDENTSET-Instanz (G', k) durch:

- $G' = (V \cup \{u_1, \dots, u_k\}, E)$ mit k neuen isolierten Knoten $u_1, \dots, u_k$
- k bleibt unverändert

Beweis Clique nach CliqueAndIndependentSet:

- Sei C eine k -Clique in G .
- Da keine Kanten entfernt wurden, ist C auch eine k -Clique in G' .
- Die Knoten $u_1, \dots, u_k$ sind isoliert, also paarweise nicht adjazent.
- Somit ist $\{u_1, \dots, u_k\}$ ein Independent Set der Größe k in G' .
- Also gibt es in G' sowohl eine k -Clique als auch ein Independent Set der Größe k .

Beweis CliqueAndIndependentSet nach Clique:

- Sei (G', k) eine Ja-Instanz, dann enthält G' insbesondere eine k -Clique C' .
- Da $k \geq 2$, hat jeder Knoten in C' mindestens einen Nachbarn in C' .
- Die Knoten $u_1, \dots, u_k$ sind isoliert, haben also keine Nachbarn.
- Somit gilt $C' \subseteq V$.
- Also ist C' eine k -Clique in G .

Laufzeit: k isolierte Knoten $u_1, \dots, u_k$ anhängen – $O(k) \subseteq O(|V|)$.

Da CLIQUE NP-schwer ist und $\text{CLIQUE} \leq_p \text{CLIQUEANDINDEPENDENTSET}$ gilt, folgt: CLIQUE-ANDINDEPENDENTSET ist NP-schwer. $\square$

Problem Partition: Gegeben Zahlen $c_1, \dots, c_n$. Gibt es S mit $\sum_{j \in S} c_j = \frac{1}{2} \sum_{j=1}^n c_j$?

Problem SubsetSum: Gegeben Zahlen $c_1, \dots, c_n$ und K . Gibt es S mit $\sum_{j \in S} c_j = K$?

Zeigen Sie, dass PARTITION NP-vollständig ist. *Hinweis:* Reduzieren Sie SUBSETSUM auf PARTITION.

Beispiel SubsetSum $\leq_p$ Partition: Zahlen einsetzen

Instanz. $c = (3, 5, 8, 4)$, $K = 12$ mit Lösung $\{3, 5, 4\}$: $3 + 5 + 4 = 12$.

Kern. Die Reduktion hängt mit $N = \sum c_j + 1 = 21$ die Zusatzzahlen $a = N - K = 9$ und $b = K + 1 = 13$ an; wegen $a + b = N + 1 > N$ trennen sich a und b in jeder Lösung. Neue Zahlen $(3, 5, 8, 4, 9, 13)$ mit Hälfte $N = 21$: Die Seite $\{3, 5, 4\} \cup \{a\}$ summiert $3 + 5 + 4 + 9 = 21$ – die Ja-Instanz bleibt Ja.

Muster Zahlen-Padding

Zahlen hinzufügen, die das Zahlenproblem in die gewünschte Form zwingen.

- Ausgleichszahlen so wählen, dass sie nicht auf dieselbe Seite passen (ihre Summe übersteigt die Zielhälfte).
- Sie erzwingen die Zielstruktur (gleiche Hälften, feste Kardinalität, festes erstes Element).
- Beweis hin: Lösung + Ausgleichszahlen aufteilen, Summen nachrechnen.
- Beweis zurück: die Größe erzwingt die Trennung der Ausgleichszahlen – der Rest ist die Originallösung.

Lösung.

$\in \mathbf{NP}$: Eingabe: Zahlen $c_1, \dots, c_n$ und Zertifikat $S \subseteq [n]$.

Der Verifizierer arbeitet wie folgt:

- Berechne die Gesamtsumme $\Sigma = \sum_{j=1}^n c_j$.
- Prüfe $\sum_{j \in S} c_j = \Sigma/2$; lehne sonst ab.
- Sonst akzeptiere.

Existiert eine Partitionslösung, dann ist eine ihrer Seiten ein Zertifikat, das akzeptiert wird. Sonst wird jedes Zertifikat abgelehnt. Zwei Summen bilden und vergleichen – $O(n)$. Also gilt PARTITION $\in \mathbf{NP}$.

Konstruktion: Sei $(c_1, \dots, c_n, K)$ eine SUBSETSUM-Instanz. Setze $N := \sum_{j=1}^n c_j + 1$ und hänge zwei Zahlen als Items $n+1$ und $n+2$ an:

$$c_{n+1} := N - K, \quad c_{n+2} := K + 1.$$

Die neue Gesamtsumme ist $(N - 1) + (N - K) + (K + 1) = 2N$, die Hälfte also N .

Schlüsselbeobachtung: $c_{n+1} + c_{n+2} = N + 1 > N$ – in jeder Partitionslösung liegen die beiden Zusatzzahlen auf *verschiedenen* Seiten.

Beweis hin:

- Sei $S \subseteq [n]$ eine Menge mit $\sum_{j \in S} c_j = K$.
- Setze $S' = S \cup \{n+1\}$.
- Dann gilt $\sum_{j \in S'} c_j = K + (N - K) = N$.
- Also ist S' eine Seite mit halber Gesamtsumme – eine Partitionslösung.

Beweis zurück:

- Sei $S' \subseteq [n+2]$ eine Seite mit $\sum_{j \in S'} c_j = N$.
- Nach der Schlüsselbeobachtung liegen $n+1$ und $n+2$ auf verschiedenen Seiten.
- O.B.d.A. gilt $n+1 \in S'$ und $n+2 \notin S'$.
- Setze $S = S' \setminus \{n+1\}$, also $S \subseteq [n]$.
- Dann gilt $\sum_{j \in S} c_j = N - (N - K) = K$ – also ist S eine SubsetSum-Lösung.

Laufzeit: Gesamtsumme berechnen und zwei Zahlen anhängen – $O(n)$.

Da SUBSETSUM NP-vollständig ist und PARTITION $\in$ NP, ist PARTITION NP-vollständig. □

Beweisen Sie die NP-Vollständigkeit von SUBSETSUM, bei dem jede Itemgröße durch 3 oder durch 7 teilbar ist.

Beispiel SubsetSum mit Teilbarkeit

Instanz. SUBSETSUM-Instanz $c = (2, 3, 5)$, $K = 5$; Lösung $S = \{1, 2\}$ mit $2 + 3 = 5$.

Kern. Multipliziere alle Größen und K mit 3: $c' = (6, 9, 15)$, $K' = 15$ – jede Größe ist durch 3 teilbar. Dieselbe Auswahl liefert $6 + 9 = 15 = K'$.

Muster Skalieren/Einbetten

Alle Zahlen mit c multiplizieren oder Parameter gleichsetzen – die Lösungen bleiben exakt dieselben.

- Transformation: $c'_i := c \cdot c_i$ und $K' := c \cdot K$ (bzw. Parameter des Zielproblems gleichsetzen).
- Die Zusatzeigenschaft folgt direkt aus dem Faktor (teilbar, keine Zweierpotenz).
- Beweis hin und zurück: Summen skalieren mit – Division durch c führt zurück.
- Laufzeit: eine Multiplikation pro Zahl – $O(n)$.

Lösung.

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

- Prüfe für jede Größe c_i , ob sie durch 3 oder durch 7 teilbar ist; lehne sonst ab.
- Prüfe, ob $\sum_{i \in S} c_i = K$ gilt; lehne sonst ab.
- Sonst akzeptiere.

Korrektheit: Ist die Instanz eine gültige Variante mit Lösung S , dann gibt es ein Zertifikat dafür. Dieses wird vom Verifizierer akzeptiert und sonst abgelehnt.

Laufzeit: Teilbarkeitstests, Summieren und Vergleichen – $O(n)$. Also liegt die Teilbarkeits-Variante in NP.

NP-schwer: Zeige, dass die Variante NP-schwer ist durch die Reduktion $\text{SUBSETSUM} \leq_p$ Teilbarkeits-Variante. SUBSETSUM ist bereits als NP-vollständig bekannt.

Konstruktion: Sei $(c_1, \dots, c_n, K)$ eine SUBSETSUM-Instanz. Setze $c'_i := 3c_i$ und $K' := 3K$. Alle Größen sind durch 3 teilbar – die Instanz ist gültig.

Beweis SubsetSum nach Variante:

- Sei S eine Lösung mit $\sum_{i \in S} c_i = K$.
- Dann gilt $\sum_{i \in S} c'_i = \sum_{i \in S} 3c_i = 3 \sum_{i \in S} c_i = 3K = K'$.
- Also ist S eine Lösung der neuen Instanz.

Beweis Variante nach SubsetSum:

- Sei S eine Lösung mit $\sum_{i \in S} c'_i = K'$.
- Wegen $c'_i = 3c_i$ heißt das $3 \sum_{i \in S} c_i = 3K$.
- Division durch 3 liefert $\sum_{i \in S} c_i = K$.
- Also ist S eine Lösung der Original-Instanz.

Laufzeit: jede der n Zahlen und K mit 3 multiplizieren – $O(n)$.

Da die Variante in **NP** liegt und NP-schwer ist, folgt: Sie ist NP-vollständig.

□

Problem AtMostTwoPerSize-SubsetSum: Zielwert $T > 0$, n Items mit Größen $a_i \in \mathbb{N}_{>0}$; jede Größe tritt höchstens zweimal auf. Gibt es S mit $\sum_{i \in S} a_i = T$? Zeigen Sie die NP-Schwere durch Reduktion eines bekannten NP-schweren Problems.

Hinweis: Bei 3-EXACTCOVER sind ein Universum U mit $|U| = 3m$ und Dreiermengen $S_1, \dots, S_n \subseteq U$ gegeben; gefragt ist eine Auswahl von Mengen, die jedes Element *genau einmal* überdeckt. Die Vorlesung reduziert 3-EC auf SUBSETSUM über die Kodierung $c_j = \sum_{u_i \in S_j} (n+1)^{i-1}$ mit Ziel $K = \sum_{j=0}^{3m-1} (n+1)^j$.

Beispiel AtMostTwoPerSize-SubsetSum

Instanz. 3-EXACTCOVER mit $U = \{u_1, \dots, u_6\}$ und den Mengen $S_1 = \{u_1, u_2, u_3\}$, $S_2 = \{u_4, u_5, u_6\}$, $S_3 = \{u_1, u_2, u_4\}$; exakte Überdeckung $\{S_1, S_2\}$.

Kern. Kodiere jede Menge in Basis $n+1=4$: $c_j = \sum_{u_i \in S_j} 4^{i-1}$, Ziel $K = \sum_{j=0}^5 4^j = 1365$.

	Ziffer zu u_i (Basis 4)						c_j
	u_6	u_5	u_4	u_3	u_2	u_1	
$S_1 = \{u_1, u_2, u_3\}$	0	0	0	1	1	1	21
$S_2 = \{u_4, u_5, u_6\}$	1	1	1	0	0	0	1344
$S_3 = \{u_1, u_2, u_4\}$	0	0	1	0	1	1	69
Ziel K	1	1	1	1	1	1	1365

Verschiedene Mengen wählen verschiedene Potenzen, also sind c_1, c_2, c_3 paarweise verschieden – jede Größe tritt nur einmal auf. Die Überdeckung $\{S_1, S_2\}$ liefert $c_1 + c_2 = 21 + 1344 = 1365 = K$.

Muster Basis-Kodierung

Kombinatorik in Ziffern: jedes Element bekommt eine Stelle in einer großen Basis.

- Basis größer als jede mögliche Ziffernsumme wählen – kein Übertrag.
- Objekt $\rightarrow$ Zahl: Ziffer 1 an den Stellen seiner Elemente.
- Ziel K : an jeder Stelle die geforderte Ziffer.
- Kein Übertrag $\Rightarrow$ die Auswahl erfüllt jede Stelle einzeln $\leftrightarrow$ exakte Überdeckung.

Lösung.

Konstruktion: Sei $(U, S_1, \dots, S_n)$ eine 3-EXACTCOVER-Instanz mit $U = \{u_1, \dots, u_{3m}\}$, o.B.d.A. S_j paarweise verschieden (Duplikate vorab löschen). Wir konstruieren daraus in Basis $n+1$ eine SubsetSum-Instanz durch:

- $c_j = \sum_{u_i \in S_j} (n+1)^{i-1}$ für $j = 1, \dots, n$ – Ziffer 1 an den Stellen der drei Elemente von S_j
- $K = \sum_{j=0}^{3m-1} (n+1)^j$ – an jeder Stelle eine 1

Beweis 3-ExactCover nach AtMostTwoPerSize-SubsetSum:

- Sei $\mathcal{S}$ eine exakte Überdeckung.
- Dann wird jedes Element u_i von genau einer Menge in $\mathcal{S}$ überdeckt.
- Somit kommt in der Summe $\sum_{S_j \in \mathcal{S}} c_j$ jede Potenz $(n+1)^{i-1}$ genau einmal vor.
- Also gilt

$$\sum_{S_j \in \mathcal{S}} c_j = \sum_{j=0}^{3m-1} (n+1)^j = K.$$

Beweis AtMostTwoPerSize-SubsetSum nach 3-ExactCover:

- Sei S eine Auswahl mit $\sum_{j \in S} c_j = K$.
- An jeder Ziffernstelle addieren sich höchstens n Einsen, denn es gibt nur n Mengen.
- Da die Basis $n+1$ ist, entsteht beim Addieren kein Übertrag.
- In K hat jede der $3m$ Ziffernstellen den Wert 1.
- Somit trägt zu jeder Ziffernstelle genau eine gewählte Menge bei.
- Also überdecken die gewählten Mengen jedes Element genau einmal.
- Damit ist die Auswahl eine exakte Überdeckung.

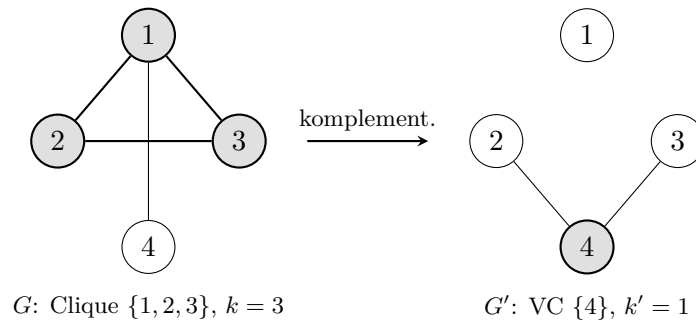
Laufzeit: Duplikate durch paarweisen Vergleich der n Mengen entfernen – $O(n^2)$ Vergleiche.
 Pro Menge drei Potenzen von $n+1$ addieren und K aus $3m$ Potenzen aufsummieren – $O(n+m)$
 Additionen polynomiell langer Zahlen.

Da 3-EXACTCOVER NP-schwer ist, ist ATMOSTTWOPEERSIZE-SUBSETSUM NP-schwer. □

Zeigen Sie, dass VERTEXCOVER NP-vollständig ist. *Hinweis:* Reduzieren Sie CLIQUE auf VERTEXCOVER.

Beispiel VertexCover NP-vollständig

Instanz. CLIQUE am folgenden Graphen G mit $n = 4$ Knoten.



Kern. Komplementieren liefert G' mit genau den Nicht-Kanten von G und $k' = n - k = 1$. Die Clique $\{1, 2, 3\}$ wird zum Vertex Cover $V \setminus C = \{4\}$: Knoten 4 deckt beide Kanten von G' .

Muster Umdeuten/Komplement

Gleiche Instanz, neue Brille: Komplement von Graph oder Lösung.

- Instanz komplementieren (Kanten $\leftrightarrow$ Nicht-Kanten) oder die Lösung als Rest $V \setminus C$ lesen.
- Parameter umrechnen ($k' = n - k$).
- Beweis hin und zurück: Definitionen ineinander übersetzen – Clique $\leftrightarrow$ unabhängig $\leftrightarrow$ überdeckend.
- Laufzeit: Komplement bilden – $O(|V|^2)$.

Lösung.

$\in \mathbf{NP}$: Eingabe: Graph $G = (V, E)$, Zahl k und Zertifikat $C \subseteq V$. Der Verifizierer arbeitet wie folgt:

- Falls $|C| > k$, lehne ab.
- Prüfe für jede Kante $\{u, v\} \in E$, ob $u \in C$ oder $v \in C$ gilt.
- Lehne ab, falls das für eine Kante fehlschlägt.
- Sonst akzeptiere.

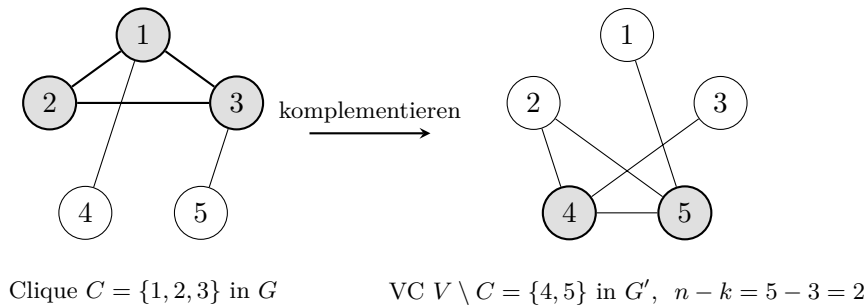
Existiert ein Vertex Cover C mit $|C| \leq k$, dann gibt es ein Zertifikat dafür. Dieses wird vom Verifizierer akzeptiert und sonst abgelehnt. Größen- und Kantenprüfung laufen in $O(|V| + |E|)$. Also gilt VERTEXCOVER $\in \mathbf{NP}$.

NP-schwer: Zeige, dass VERTEXCOVER NP-schwer ist durch die Reduktion $\text{CLIQUE} \leq_p \text{VERTEXCOVER}$. CLIQUE ist bereits als NP-vollständig bekannt.

Konstruktion: Sei $(G = (V, E), k)$ eine CLIQUE-Instanz mit $n = |V|$ (o.B.d.A. $k \leq n$). Wir konstruieren daraus eine VERTEXCOVER-Instanz $(G' = (V, E'), k')$ durch:

- $E' = \binom{V}{2} \setminus E$, also genau die Nicht-Kanten von G
- $k' = n - k$

So wird G zum Komplementgraphen G' invertiert.



Beweis Clique nach VertexCover:

- Sei C eine Clique in G mit $|C| \geq k$.
- Setze $W = V \setminus C$, dann gilt $|W| \leq n - k = k'$.
- In G sind alle Kanten innerhalb von C vorhanden.
- Somit hat G' keine Kante mit beiden Endpunkten in C – also ist C unabhängig in G' .
- Also hat jede Kante von G' einen Endpunkt in W .
- Damit ist W ein Vertex Cover von G' mit $|W| \leq k'$.

Beweis VertexCover nach Clique:

- Sei W ein Vertex Cover von G' mit $|W| \leq k'$.
- Setze $C = V \setminus W$, dann gilt $|C| \geq n - k' = k$.
- Sei $\{u, v\} \subseteq C$ ein beliebiges Knotenpaar.
- Da $u, v \notin W$ und W jede Kante von G' deckt, gilt $\{u, v\} \notin E'$.
- Somit gilt $\{u, v\} \in E$, denn E' enthält genau die Nicht-Kanten von G .
- Also ist C eine Clique in G mit $|C| \geq k$.

Laufzeit: Alle Knotenpaare durchgehen und die Kanten invertieren – $O(|V|^2)$.

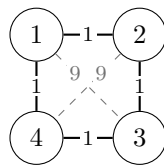
Da VERTEXCOVER $\in$ NP und NP-schwer ist, folgt: VERTEXCOVER ist NP-vollständig. □

Problem TSP: Gegeben n Städte mit beliebigen Distanzen $d(u, v) > 0$ – *allgemeines* TSP, die Dreiecksungleichung wird *nicht* vorausgesetzt. Gesucht ist eine Rundtour minimaler Länge durch alle Städte. Ein Algorithmus A hat Güte α , wenn $A(I) \leq \alpha \cdot \text{OPT}(I)$ für alle Instanzen I gilt.

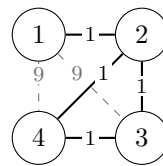
Zeigen Sie: Für kein $\alpha > 1$ gibt es einen Approximationsalgorithmus mit Güte α für TSP, außer $P = NP$.

Beispiel TSP ist nicht approximierbar

Instanz. HAMILTONIANCYCLE-Graphen mit $n = 4$ Knoten, angenommene Güte $\alpha = 2$: Kante $\rightarrow$ Distanz 1, Nicht-Kante $\rightarrow \alpha n + 1 = 9$ (gestrichelt).



Ja: $\text{OPT} = 4$



Nein: $\text{OPT} \geq 12$

Kern. Links gibt es einen Hamiltonkreis: $\text{OPT} = n = 4$, also $A(I) \leq \alpha n = 8$. Rechts hat Knoten 1 Grad 1, also keinen: jede Tour benutzt eine Nicht-Kante, $\text{OPT} \geq 9 + 3 \cdot 1 = 12 > 8$, somit $A(I) > 8$. Der Test „ $A(I) \leq 8$?“ entscheidet HAMILTONIANCYCLE.

Muster Gap strecken

Approximation ausschließen: Ja und Nein so weit auseinanderziehen, dass ein Approximierer sie trennen würde.

- Ja-Fall billig kodieren (Kante $\rightarrow$ Distanz 1, Tour der Länge n).
- Nein-Fall teuer machen (Nicht-Kante $\rightarrow$ Distanz $\alpha n + 1$).
- Ein α -Approximierer bliebe im Ja-Fall unter der Nein-Schwelle.
- Er entschiede das NP-schwere Problem exakt – Widerspruch, außer $P = NP$.

Lösung.

Angenommen, es gäbe einen polynomiellen Algorithmus A mit $A(I) \leq \alpha \cdot \text{OPT}(I)$ für ein festes $\alpha > 1$. Wir lösen damit HAMILTONIANCYCLE in Polynomialzeit:

Konstruktion: Zu $G = (V, E)$ mit $n = |V|$ baue die TSP-Instanz mit Städten V und

$$d(u, v) = \begin{cases} 1 & \{u, v\} \in E, \\ \alpha n + 1 & \text{sonst.} \end{cases}$$

Fall 1: G hat einen Hamiltonkreis. Dann ist $\text{OPT} = n$ (Tour nur über echte Kanten), also $A(I) \leq \alpha n$.

Fall 2: G hat keinen Hamiltonkreis. Dann benutzt jede Tour mindestens eine Nicht-Kante, also $\text{OPT} \geq (\alpha n + 1) + (n - 1) > \alpha n$, und damit auch $A(I) > \alpha n$.

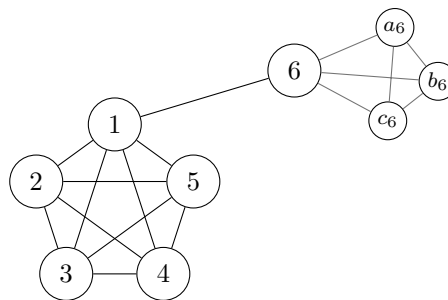
Der Vergleich „ $A(I) \leq \alpha n$?“ entscheidet also HAMILTONIANCYCLE in Polynomialzeit. Da HAMILTONIANCYCLE NP-vollständig ist, folgt $P = NP$. Also existiert ein solcher Algorithmus nur, wenn $P = NP$. $\square$

Problem k -CLIQUE-DEG-3: Gegeben $G = (V, E)$ mit $\deg(v) \geq 3$ für alle $v \in V$, und $k \in \mathbb{N}_{>0}$. Existiert eine Clique C mit $|C| \geq k$?

- Zeigen Sie, dass k -CLIQUE für $k \in \{1, 2, 3, 4\}$ polynomiell lösbar ist.
- Zeigen Sie, dass k -CLIQUE-DEG-3 NP-vollständig ist.

Beispiel k -CLIQUE-DEG-3

Instanz. k -CLIQUE am folgenden G : ein K_5 auf $\{1, \dots, 5\}$ plus Pendant-Knoten 6 mit $\deg(6) = 1 < 3$, gesucht $k = 5$.



gezeichnet: das private K_4 an Knoten 6; an jedem der Knoten $1, \dots, 5$ hängt ebenso eines

Kern. An jeden Knoten v wird ein privates K_4 auf $\{v, a_v, b_v, c_v\}$ angehängt; k bleibt. In G' haben alle Knoten Grad ≥ 3 , und die 5-Clique $\{1, \dots, 5\}$ bleibt erhalten. Gadget-Cliquen haben Größe $\leq 4 < k$, erzeugen also keine neue 5-Clique.

Muster Spezialfall vorab lösen

Kleine Parameter direkt lösen, damit die Reduktion nur den harten Fall behandeln muss.

- Fallunterscheidung: für $k \leq c$ ist das Problem polynomiell lösbar (alle Mengen testen).
- Antwort als feste triviale Ja- bzw. Nein-Instanz ausgeben.
- Für große k : die Instanz (angepasst) durchreichen.
- Korrektheit und Laufzeit in beiden Zweigen begründen.

Lösung.

a) Algorithmus für festes $k \leq 4$:

- Teste jede Knotenmenge $C \subseteq V$ mit $|C| = k$.
- Prüfe per Adjazenzmatrix, ob alle Paare in C verbunden sind.
- Akzeptiere, sobald eine Menge alle Prüfungen besteht; sonst lehne ab.

Korrektheit: Der Algorithmus testet alle Kandidaten. Er akzeptiert also genau dann, wenn eine k -Clique existiert.

Laufzeit: Es gibt $\binom{|V|}{k} = O(|V|^k) \subseteq O(|V|^4)$ Mengen. Pro Menge höchstens $\binom{4}{2} = 6$ Paar-Prüfungen in $O(1)$ – gesamt $O(|V|^4)$, also polynomiell lösbar.

b) $\in$ NP: Eingabe: Graph $G = (V, E)$, Zahl k und Zertifikat $C \subseteq V$. Der Verifizierer arbeitet wie folgt:

- Prüfe für jeden Knoten $v \in V$, ob $\deg(v) \geq 3$ gilt; lehne sonst ab.
- Falls $|C| < k$, lehne ab.
- Prüfe für jedes Paar $\{x, y\} \subseteq C$, ob $\{x, y\} \in E$ gilt.
- Lehne ab, falls für ein Paar $\{x, y\} \notin E$ gilt.
- Sonst akzeptiere.

Korrektheit: Ist die Instanz gültig und hat G eine Clique C mit $|C| \geq k$, dann gibt es ein Zertifikat dafür. Dieses wird vom Verifizierer akzeptiert und sonst abgelehnt.

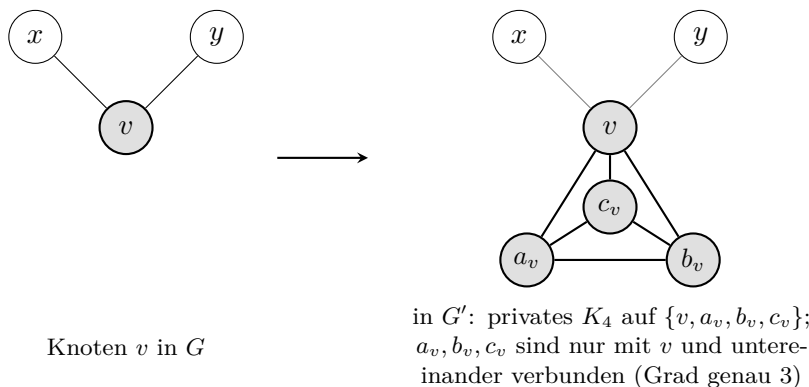
Laufzeit: Gradprüfung $O(|V| + |E|)$, Größen- und Paar-Prüfung $O(|V|^2)$ – gesamt $O(|V|^2)$. Also gilt k -CLIQUE-DEG-3 $\in$ NP.

NP-schwer: Zeige, dass k -CLIQUE-DEG-3 NP-schwer ist durch die Reduktion k -CLIQUE $\leq_p$ k -CLIQUE-DEG-3. k -CLIQUE ist bereits als NP-vollständig bekannt.

Konstruktion: Sei $(G = (V, E), k)$ eine k -CLIQUE-Instanz, o.B.d.A. $k \geq 5$. Wir konstruieren daraus eine k -CLIQUE-DEG-3-Instanz (G', k) durch:

- $V' = V \cup \{a_v, b_v, c_v \mid v \in V\}$
- $E' = E \cup \{\{x, y\} \mid v \in V, x, y \in \{v, a_v, b_v, c_v\}, x \neq y\}$
- $k' = k$

Dann gilt $\deg(v) \geq 3$ für alle Knoten: Jedes $v \in V$ hat die drei Nachbarn a_v, b_v, c_v , und Gadget-Knoten haben Grad genau 3. Die Instanz ist gültig.



Beweis k -Clique nach k -CLIQUE-DEG-3 (Fall $k \geq 5$):

- Sei C eine k -Clique in G .
- Da keine Kanten entfernt wurden, ist C auch eine k -Clique in G' .

Beweis k -CLIQUE-DEG-3 nach k -Clique (Fall $k \geq 5$):

- Sei C' eine k -Clique in G' .
- Jeder Knoten in C' hat mindestens $k - 1 \geq 4$ Nachbarn in C' .
- Gadget-Knoten haben Grad 3. Somit liegt keiner in C' .
- Also gilt $C' \subseteq V$.
- Da zwischen Knoten aus V keine neuen Kanten hinzugekommen sind, ist C' eine k -Clique in G .

Im Fall $k \leq 4$ wird (G, k) direkt gelöst, und die ausgegebene triviale Instanz hat dieselbe Antwort.

Laufzeit: Pro Knoten ein Gadget konstanter Größe anlegen – $O(|V|)$. Im Fall $k \leq 4$ alle Knotenmengen testen – $O(|V|^4)$.

Da k -CLIQUE-DEG-3 $\in \text{NP}$ und NP-schwer ist, folgt: k -CLIQUE-DEG-3 ist NP-vollständig.
 $\square$

3 Verfahren und Einzelmuster: einzeln lernen

A19 Knapsack: Greedy und ModifiedGreedy

Klausur SS23-H, A1, 3+2+5 P.

Rucksack-Instanz mit Kapazität $B = 16$, Items als (p_i, w_i) :

$$I = [(1, 4), (1, 1), (1, 3), (11, 13), (3, 3), (5, 6), (1, 5)]$$

- Wenden Sie Greedy an (mit Begründung der Auswahl). Lösungswert?
- Wenden Sie ModifiedGreedy an. Wie ändert sich die Lösung?
- Güte von ModifiedGreedy? Idee zur Verbesserung auf $\frac{3}{2}$ bzw. $\frac{4}{3}$?

Hinweis: Greedy sortiert die Items absteigend nach Profitdichte p_i/w_i und packt in dieser Reihenfolge jedes Item ein, das noch passt. ModifiedGreedy gibt das Bessere aus Greedy-Lösung und profitreichstem Einzel-Item aus.

Muster Verfahren Greedy/ModifiedGreedy

Knapsack-Heuristik: nach Profitdichte packen; MGA sichert mit dem besten Einzel-Gegenstand ab.

- Nach p_i/w_i absteigend sortieren.
- Der Reihe nach packen, was noch passt; Rest überspringen.
- MGA = $\max\{\text{Greedy-Wert, bester Einzel-Gegenstand}\}$.
- In der Klausur: Schrittfolge mit Restkapazität notieren.

Lösung.

a) Profitdichten p_i/w_i : 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). Einpacken (Restkapazität beachten):

- (1, 1) einpacken – Restkapazität 15.
- (3, 3) einpacken – Restkapazität 12.
- (11, 13) passt nicht ($13 > 12$).
- (5, 6) einpacken – Restkapazität 6.
- (1, 3) einpacken – Restkapazität 3.
- (1, 4) und (1, 5) passen nicht.

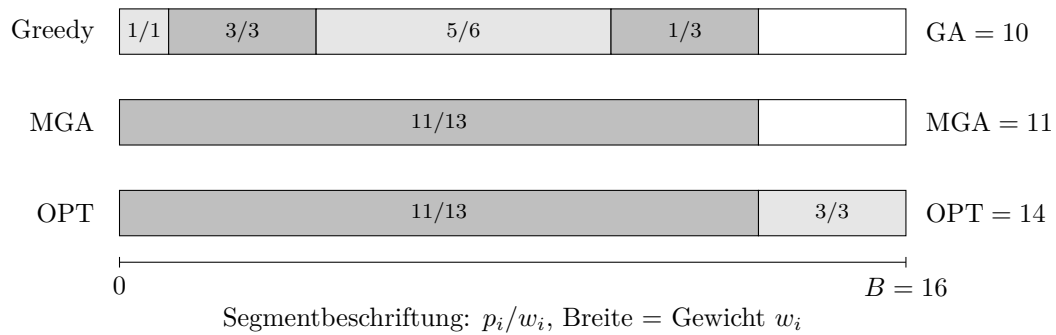
$$GA = 1 + 3 + 5 + 1 = 10.$$

b) ModifiedGreedy vergleicht zwei Kandidaten:

- Greedy-Wert: $GA = 10$.
- Profitreichstes Einzel-Item: (11, 13) mit Profit 11.

- $\text{MGA} = \max\{10, 11\} = 11$.

Die Lösung wechselt von $\{(1, 1), (3, 3), (5, 6), (1, 3)\}$ zum einzelnen Item $(11, 13)$.



c) ModifiedGreedy hat Güte 2.

Verbesserung durch k -Enumeration (Sahni):

- Probiere jede Vorauswahl aus höchstens k Items.
- Fülle jede Vorauswahl mit Greedy auf.
- Gib die beste gefundene Lösung aus.

Das liefert Güte $1 + \frac{1}{k}$: $\frac{3}{2}$ für $k = 2$ und $\frac{4}{3}$ für $k = 3$.

□

Jobs $p = (5, 3, 7, 8, 1, 4, 2)$, $m = 3$.

- Wenden Sie LPT an (Zwischenschritte, Reihenfolge, Schedule grafisch).
- Approximative Güte von LPT?

Hinweis: ListScheduling legt jeden Job in der gegebenen Reihenfolge auf die aktuell am wenigsten belastete Maschine. LPT ist ListScheduling nach absteigender Sortierung der Bearbeitungszeiten.

Muster Verfahren LPT/ListScheduling

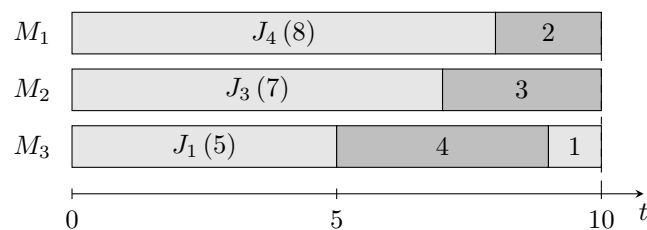
Scheduling-Heuristik: jeden Job auf die aktuell leerste Maschine; LPT sortiert vorher absteigend.

- (LPT) Jobs absteigend sortieren.
- Je Job die Maschine mit kleinster Last wählen.
- Lastvektor nach jedem Schritt notieren; Makespan = größte Last.
- Schedule als Balkendiagramm zeichnen.

Lösung.

a) Absteigend sortiert: 8, 7, 5, 4, 3, 2, 1 (also $J_4, J_3, J_1, J_6, J_2, J_7, J_5$). Platzierung auf die jeweils leerste Maschine:

- $8 \rightarrow M_1: (8, 0, 0)$, $7 \rightarrow M_2: (8, 7, 0)$, $5 \rightarrow M_3: (8, 7, 5)$,
- $4 \rightarrow M_3: (8, 7, 9)$, $3 \rightarrow M_2: (8, 10, 9)$, $2 \rightarrow M_1: (10, 10, 9)$, $1 \rightarrow M_3: (10, 10, 10)$.

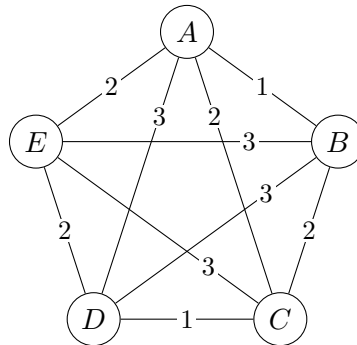


Makespan = 10.

b) LPT hat Güte $\frac{4}{3} - \frac{1}{3m}$.

□

Gegeben sei folgender Graph $G = (V, E)$:



- Wenden Sie Δ TSP1 auf den Graphen G ab Startknoten C an; geben Sie alle Zwischengraphen und die Länge der Rundreise an.
- Was berechnet Δ TSP1, und welche Bedingungen muss der Graph erfüllen?
- Begründen Sie die approximative Güte 2.
- Ist das Entscheidungsproblem zum TSP in P ? Kurze Begründung.
- Wahr oder falsch: „Güte 1,5 bedeutet $\frac{d(R)}{\text{OPT}(I)} \geq 1,5$ für eine Instanz I .“ Kurze Begründung.

Hinweis (Δ TSP1): (1) MST T berechnen; (2) alle MST-Kanten verdoppeln; (3) Eulerkreis ab Startknoten; (4) Abkürzen: bereits besuchte Knoten überspringen.

Muster Verfahren Δ TSP1

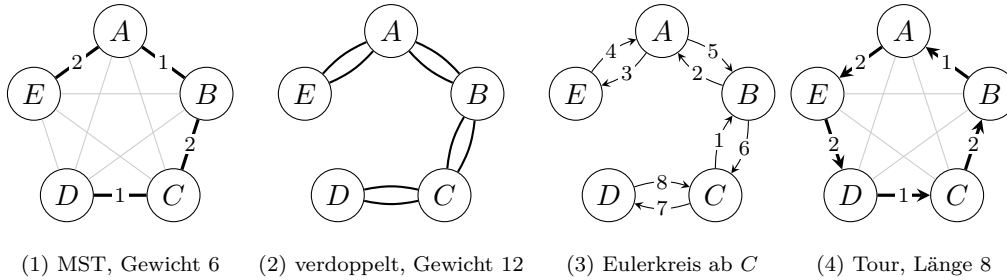
MST-Verdopplung: Baum $\rightarrow$ Eulerkreis $\rightarrow$ Abkürzen; Güte 2.

- MST berechnen (Kruskal: Kanten nach Gewicht sortiert nennen).
- Alle MST-Kanten verdoppeln – alle Grade werden gerade.
- Eulerkreis ab Startknoten notieren.
- Abkürzen: besuchte Knoten überspringen; Tour und Länge angeben.

Lösung.

a)

- MST (Kruskal): $A-B$ (1), $C-D$ (1), $B-C$ (2), $A-E$ (2); Gewicht 6.
- Verdoppeln: Multigraph, Gewicht 12 – alle Grade gerade.
- Eulerkreis ab C : $[C, B, A, E, A, B, C, D, C]$.
- Abkürzen: Tour $[C, B, A, E, D, C]$, Länge 8.



b) Δ TSP1 berechnet eine Rundreise (TSP-Tour). Der Graph muss vollständig sein, die Distanzen symmetrisch und die Dreiecksungleichung $d(i, j) \leq d(i, k) + d(k, j)$ erfüllen – sonst kann das Abkürzen die Tour verlängern.

c) Güte 2, in drei Schritten:

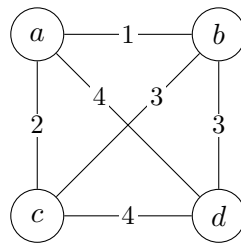
- Entfernt man aus einer optimalen Tour eine Kante, entsteht ein Spannbaum. T ist ein minimaler Spannbaum, also $w(T) \leq \text{OPT}$.
- Der Eulerkreis nutzt jede MST-Kante zweimal. Seine Länge ist $2w(T) \leq 2\text{OPT}$.
- Abkürzen verlängert wegen der Dreiecksungleichung nicht.

Also gilt $d(R) \leq 2\text{OPT}$.

d) Nein (außer $P = NP$): Das TSP-Entscheidungsproblem ist NP-vollständig – HAMILTONIAN-CYCLE reduziert darauf (Kanten Distanz 1, Nicht-Kanten $|V|+1$, $L = |V|$). Läge es in P , folgte $P = NP$.

e) Falsch. Multiplikative Güte α ist eine *obere* Schranke für alle Instanzen: $d(R) \leq \alpha \cdot \text{OPT}(I)$ für *jede* Instanz I – nicht eine untere Schranke $\geq$ für eine Instanz. $\square$

Gegeben sei folgender Graph $G = (V, E)$:



- Wenden Sie Christofides (Δ TSP2) auf den Graphen G mit Startknoten a an; geben Sie alle Zwischengraphen an, auch Schritte ohne Änderung.
- Was berechnet Christofides? Nennen Sie die zwei Zusatzeigenschaften der Eingabe, die er braucht, mit je einer verletzenden Eingabe. Geben Sie die Güte an.

Hinweis (Christofides): (1) MST T berechnen; (2) X = Knoten mit ungeradem Grad in T ; (3) minimales perfektes Matching K auf X ; (4) Eulerkreis im Multigraphen $T + K$; (5) Abkürzen.

Muster Verfahren Christofides

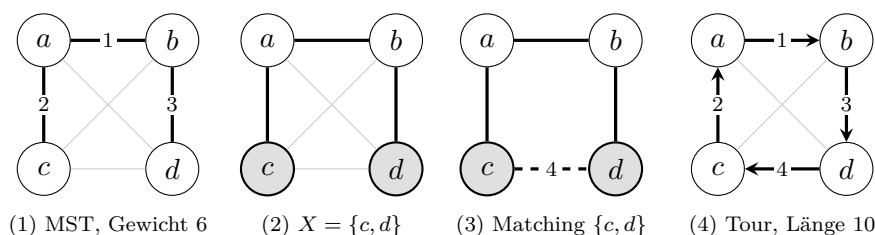
Wie Δ TSP1, aber statt Verdopplung ein Matching auf den ungeraden Knoten; Güte $\frac{3}{2}$.

- MST berechnen.
- X = Knoten mit ungeradem Grad im MST.
- Minimales perfektes Matching auf X hinzufügen.
- Eulerkreis ab Startknoten, dann abkürzen; Tour und Länge angeben.

Lösung.

a)

- MST (Kruskal): $\{a, b\}$ (1), $\{a, c\}$ (2), $\{b, d\}$ (3); Gewicht 6.
- Ungerade Grade im MST: $a: 2, b: 2, c: 1, d: 1 \Rightarrow X = \{c, d\}$.
- Minimales perfektes Matching auf X : einzige Möglichkeit $K = \{\{c, d\}\}$, Kosten 4.
- Multigraph $T + K$: alle Grade gerade. Eulerkreis ab a : $[a, b, d, c, a]$, Kosten $1 + 3 + 4 + 2 = 10$.
- Abkürzen: kein Knoten doppelt – keine Änderung (wird aber geprüft). Tour $R = [a, b, d, c, a]$, Länge 10.



b) Christofides berechnet eine TSP-Rundreise. Benötigt werden *Symmetrie* ($d(u, v) = d(v, u)$) und die *Dreiecksungleichung* ($d(u, v) \leq d(u, w) + d(w, v)$). Verletzende Eingaben: $d(a, b) = 1$, $d(b, a) = 5$ (asymmetrisch); bzw. $d(a, b) = 10$, $d(a, c) = d(c, b) = 1$ (Δ -Ungleichung verletzt). Güte: $\frac{3}{2}$. $\square$

Jobs belegen $q_j \in [m]$ Maschinen gleichzeitig für Zeit p_j . ListScheduling platziert die Jobs der Reihe nach zum frühestmöglichen Zeitpunkt auf den q_j am wenigsten belasteten Maschinen.

- Anwendung: $m = 3$; Jobs (p, q) : $(1, 1), (1, 3), (2, 2), (3, 1)$. Geben Sie den Schedule an.
- Worst-Case-Güte für $q_j = 1$?
- Zeigen Sie $LS(I) \leq OPT(I) + \frac{1}{2}p_{\max}$ für alle I mit $\frac{m}{3} < q_j \leq \frac{m}{2}$.
- Bonus:** Güte für $\frac{m}{k+1} < q_j \leq \frac{m}{k}$ ($k \geq 2$)? Beweis.

Muster Verfahren Parallel-Task-ListScheduling

ListScheduling für Jobs, die mehrere Maschinen gleichzeitig belegen.

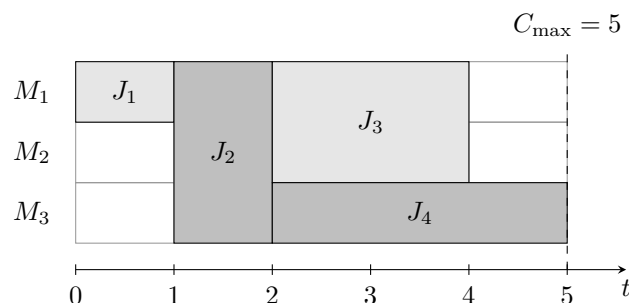
- Je Job den frühesten Zeitpunkt suchen, an dem die q_j leersten Maschinen frei sind.
- Lastvektor nach jedem Job notieren, Schedule als Zeitdiagramm zeichnen.
- Für Schranken: Moment vor dem Makespan-Job betrachten – Auslastungsargument.

Lösung.

a) Start mit Lasten $C = (0, 0, 0)$:

- Job 1 $(1, 1)$: startet bei 0 auf einer Maschine – $C = (1, 0, 0)$.
- Job 2 $(1, 3)$: braucht alle 3 Maschinen, frühester Start $\max C = 1$, läuft $[1, 2]$ – $C = (2, 2, 2)$.
- Job 3 $(2, 2)$: Start 2 auf zwei Maschinen, läuft $[2, 4]$ – $C = (4, 4, 2)$.
- Job 4 $(3, 1)$: leerste Maschine hat Last 2, Start 2, läuft $[2, 5]$ – $C = (5, 4, 4)$.

Makespan 5.



b) Für $q_j = 1$ ist es das klassische ListScheduling: Güte $2 - \frac{1}{m}$.

c) Sei j ein Job, der den Makespan bestimmt, mit Startzeit $s = LS - p_j$. Zu jedem Zeitpunkt $t < s$ sind weniger als $q_j \leq \frac{m}{2}$ Maschinen frei (sonst wäre j früher platziert worden), also mehr als $\frac{m}{2}$ belegt. Jeder Job belegt höchstens $\frac{m}{2}$ Maschinen, also laufen mindestens zwei Jobs. Jeder

belegt *mehr* als $\frac{m}{3}$, also laufen höchstens zwei. Also laufen in $[0, s)$ stets *genau zwei* Jobs. Damit ist

$$2s = \int_0^s \# \text{laufende Jobs } dt \leq \sum_i p_i - p_j.$$

Im Optimum laufen ebenfalls nie drei Jobs gleichzeitig (drei Jobs belegen zusammen mehr als $3 \cdot \frac{m}{3} = m$ Maschinen – mehr als vorhanden), also $\sum_i p_i \leq 2 \text{OPT}$. Einsetzen: $s \leq \frac{1}{2}(\sum_i p_i - p_j) \leq \text{OPT} - \frac{1}{2}p_j$, folglich

$$\text{LS} = s + p_j \leq \text{OPT} + \frac{1}{2}p_j \leq \text{OPT} + \frac{1}{2}p_{\max}.$$

d) Güte $2 - \frac{1}{k}$, analog zu c). Vor dem Start von j sind mehr als $m - \frac{m}{k} = \frac{(k-1)m}{k}$ Maschinen belegt. Jeder Job belegt $\leq \frac{m}{k}$, also laufen mindestens k Jobs. Jeder belegt $> \frac{m}{k+1}$, also höchstens k . Es laufen also stets genau k Jobs, und $ks \leq \sum_i p_i - p_j$. Im Optimum laufen $\leq k$ Jobs parallel, also $\sum_i p_i \leq k \text{OPT}$. Daraus $s \leq \text{OPT} - \frac{1}{k}p_j$, und mit $p_{\max} \leq \text{OPT}$ folgt

$$\text{LS} \leq \text{OPT} + (1 - \frac{1}{k})p_j \leq \text{OPT} + (1 - \frac{1}{k})p_{\max} \leq (2 - \frac{1}{k}) \text{OPT}.$$

□

STRIPPACKING: Rechtecke (nach Höhe absteigend sortiert) in einen Streifen der Breite 1 packen, Höhe minimieren. NFDH packt stufenweise von links; passt ein Rechteck nicht mehr, beginnt eine neue Stufe.

- Instanz: fünf Rechtecke der Höhe 1 mit Breiten $\frac{1}{2}, \frac{1}{4}, \frac{1}{2}, \frac{1}{4}, \frac{1}{2}$; NFDH braucht Höhe 3. Geben Sie eine optimale Packung und die Approximationsrate an.
- Zeigen Sie: Es gibt L_n mit $\lim_{n \rightarrow \infty} \text{NFDH}(L_n)/\text{OPT}(L_n) = 2$.
- Zeigen Sie für Instanzen mit $h(r_i) = 1$: $\text{NFDH}(L) \leq 2 \text{OPT}(L) + 1$.
- Bonus:** Zeigen Sie dieselbe Schranke für $h(r_i) \leq 1$.

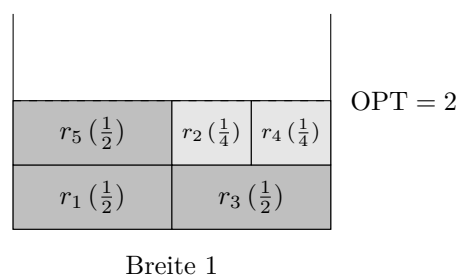
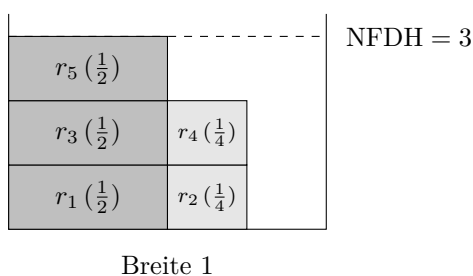
Muster Verfahren NFDH

Shelf-Packing: nach Höhe sortiert stufenweise packen; Schranke über Flächenargument.

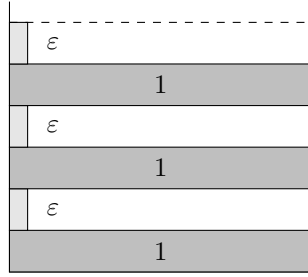
- Rechtecke nach Höhe absteigend sortieren.
- Stufe von links füllen; passt eines nicht mehr, neue Stufe darüber öffnen.
- Gesamthöhe = Summe der Stufenhöhen.
- Schranke: zwei aufeinanderfolgende Stufen füllen zusammen mehr als Breite 1 – daraus die untere Flächenschranke.

Lösung.

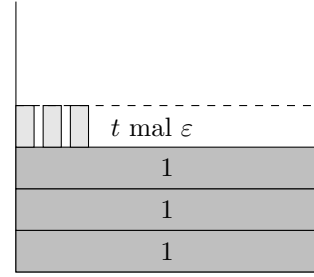
a) Optimal: Stufe 1 mit $\frac{1}{2} + \frac{1}{2} = 1$ (Rechtecke 1 und 3), Stufe 2 mit $\frac{1}{2} + \frac{1}{4} + \frac{1}{4} = 1$ (Rechtecke 5, 2, 4) – Höhe $\text{OPT} = 2$. Rate: $\frac{3}{2}$.



b) L_n ($n = 2t$): alle Höhen 1; Breiten abwechselnd 1 und ε . NFDH: Jedes Breite-1-Rechteck füllt seine Stufe komplett, jedes ε -Rechteck eröffnet eine neue Stufe, auf die das nächste Breite-1-Rechteck nicht mehr passt – $2t$ Stufen, $\text{NFDH} = 2t$. Optimal: die t Breite-1-Rechtecke je eigene Stufe, alle t ε -Rechtecke zusammen auf eine Stufe – $\text{OPT} \leq t + 1$. Rate $\frac{2t}{t+1} \rightarrow 2$.



NFDH: jede Stufe ein Rechteck



OPT: alle ε auf eine Stufe

c) Seien $W_1, \dots, W_S$ die Gesamtbreiten der NFDH-Stufen; $\text{NFDH} = S$. Für $i < S$ gilt $W_i + W_{i+1} > 1$: Das erste Rechteck der Stufe $i+1$ (Breite $\leq W_{i+1}$) passte nicht mehr auf Stufe i . Summieren über $i = 1, \dots, S-1$:

$$2 \sum_i W_i \geq \sum_{i=1}^{S-1} (W_i + W_{i+1}) > S - 1.$$

Die Gesamtfläche ist $\sum_i W_i$ (Höhen 1) und passt in einen Streifen der Breite 1: $\text{OPT} \geq \sum_i W_i$. Also $2 \text{OPT} > S - 1$, d. h. $\text{NFDH} = S \leq 2 \text{OPT} + 1$.

d) Zu zeigen ist $\text{NFDH}(L) \leq 2 \text{OPT}(L) + 1$ auch für Höhen $h(r_i) \leq 1$. Sei H_i die Höhe der Stufe i (= Höhe ihres ersten Rechtecks), W_i ihre Gesamtbreite und A_i ihre Rechteck-Gesamtfläche.

Erstens: Jedes Rechteck der Stufe i hat Höhe $\geq H_{i+1}$ (absteigende Sortierung). Die Rechtecke der Stufe i haben zusammen die Breite W_i , also

$$A_i \geq W_i \cdot H_{i+1}.$$

Zweitens: Das *erste* Rechteck der Stufe $i+1$ passte nicht mehr auf Stufe i . Es hat also Breite $> 1 - W_i$ und Höhe H_{i+1} , folglich

$$\text{Fläche des ersten Rechtecks von Stufe } i+1 > (1 - W_i) H_{i+1}.$$

Kombination: Zerlege H_{i+1} in die beiden Anteile und setze Erstens und Zweitens ein:

$$H_{i+1} = W_i H_{i+1} + (1 - W_i) H_{i+1} < A_i + (\text{Fläche des ersten Rechtecks von Stufe } i+1).$$

Summierung: Summiere die Kombination über $i \geq 1$. Dabei zählt jede Stufe höchstens zweimal: einmal als A_i und einmal über ihr erstes Rechteck. Also

$$\sum_{i \geq 2} H_i < 2 \cdot \text{Gesamtfläche} \leq 2 \text{OPT}.$$

Fazit: Die erste Stufe hat Höhe $H_1 \leq 1$. Damit folgt

$$\text{NFDH} = \sum_i H_i \leq 2 \text{OPT} + 1.$$

□

Entwerfen Sie eine Turingmaschine (Einband-TM) für $L = \{w \in \Sigma^* \mid w \text{ ist ein Palindrom}\}$ und geben Sie die Laufzeit an (mit Begründung von Korrektheit und Laufzeit).

Beispiel TM für Palindrome

Instanz. Eingabe $w = abba$ (ein Palindrom).

Kern. Die Maschine vergleicht wiederholt den ersten und den letzten unmarkierten Buchstaben und ersetzt gleiche Paare durch x . Der Bandinhalt läuft über $abba \rightarrow xbbx \rightarrow xxxx$; kein unmarkierter Buchstabe bleibt übrig – die Maschine akzeptiert.

Muster Maschine bauen

TM konstruieren: Bandbewegungen als Schleife beschreiben, Laufzeit über Durchläufe zählen.

- Invariante festlegen: Was ist nach jedem Durchlauf erledigt?
- Schleife beschreiben: markieren, laufen, vergleichen, zurück.
- Abbruchfälle klar benennen: akzeptieren/verwerfen.
- Laufzeit = Anzahl Durchläufe $\times$ Bandlänge.

Lösung.

Die Maschine:

- (1) Falls kein Wort auf dem Band steht, akzeptiere.
- (2) Falls nur ein Buchstabe auf dem Band steht, akzeptiere.
- (3) Lies den ersten Buchstaben, bewege den Kopf zum letzten. Sind beide verschieden, verwirf. Sonst ersetze beide durch ein neues Symbol x (erst den letzten, dann zurück zum ersten).
- (4) Gehe zu Schritt (1).

Korrektheit: *Invariante:* Nach jedem Durchlauf ist das äußerste unmarkierte Buchstabenpaar als gleich bestätigt und markiert. Der noch unmarkierte Rest ist genau dann ein Palindrom, wenn w eines ist. Bei Ungleichheit eines Paares ist w kein Palindrom – verwerfen. Sonst bleibt am Ende in der Mitte ein oder kein Buchstabe übrig – beides ist ein Palindrom, akzeptieren.

Laufzeit: Die Schritte (1)–(3) kosten je $O(n)$. Pro Durchlauf werden zwei Buchstaben gestrichen, also gibt es höchstens $\lfloor n/2 \rfloor$ Durchläufe. Insgesamt $O(n^2)$. $\square$