

Aufgabenkatalog

Alle Klausur-, Haus- und Präsenzaufgaben mit minimalen Lösungen
Analyse von Algorithmen und Komplexität – CAU Kiel

Inhalt

1	NP-Zugehörigkeit	2
2	Vorlesungsbeweise	6
3	Reduktionen: NP-Schwere und NP-Vollständigkeit	23
4	ETH-Schranken	56
5	Approximation: anwenden, Güte beweisen, Worst Case	75
6	Wahr oder falsch?	92
7	Turingmaschinen	95

1 NP-Zugehörigkeit

A1 Longest Path $\in$ NP (Dreischritt)

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

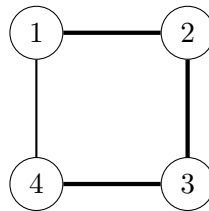
Problem Longest Path: Gegeben ein ungerichteter Graph $G = (V, E)$ und eine Zahl k . Gibt es einen *einfachen* Pfad in G (kein Knoten wird doppelt besucht) mit Länge $\geq k$?

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.

Lösung.

- Eine Folge von $k + 1$ paarweise verschiedenen Knoten $(v_0, \dots, v_k)$ – der behauptete Pfad.
- 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 $O(|V|^2)$.

- 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. □

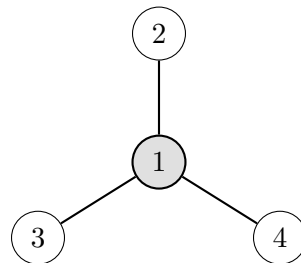
Problem Dominating Set: Gegeben ein ungerichteter Graph $G = (V, E)$ und $k \in \mathbb{N}_0$. Gibt es $D \subseteq V$ mit $|D| \leq k$, sodass jeder Knoten in D liegt oder zu einem Knoten in D benachbart ist?

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

- Geben Sie ein Zertifikat für DOMINATING SET an.
- Beschreiben Sie einen polynomiellen Verifizierer für DOMINATING SET und schätzen Sie die Laufzeit in O -Notation konkret ab.
- Beschreiben Sie (in Worten) eine polynomielle, nichtdeterministische Turing-Maschine, die DOMINATING SET löst. Schätzen Sie auch hier die Laufzeit in O -Notation konkret ab.

Beispiel Dominating Set $\in$ NP

Instanz. Der folgende Stern mit Zentrum 1 und Schranke $k = 1$.



Kern. Zertifikat $D = \{1\}$ (grau markiert): $|D| = 1 \leq k$, das Zentrum liegt in D und jedes Blatt ist zu ihm benachbart – der Verifizierer akzeptiert. Der Rechenweg der NTM, der genau diese Menge wählt, akzeptiert.

Lösung.

- Die Knotenmenge $D \subseteq V$.
- Eingabe: Graph $G = (V, E)$, Zahl k und Zertifikat $D \subseteq V$.

Der Verifizierer arbeitet wie folgt:

- Prüfe $|D| \leq k$; lehne sonst ab.
- Prüfe für jeden Knoten $v \in V$, ob $v \in D$ gilt oder ein Nachbar von v in D liegt. Durchlaufe dazu die Zeile von v in der Adjazenzmatrix.
- Lehne ab, falls ein Knoten weder in D liegt noch einen Nachbarn in D hat.
- Sonst akzeptiere.

Korrektheit: Existiert ein Dominating Set der Größe $\leq k$, dann ist genau dieses ein Zertifikat, das akzeptiert wird. Existiert keines, dann verletzt jedes Zertifikat eine der beiden Prüfungen – der Verifizierer lehnt ab.

Laufzeit: Der Größentest läuft in $O(|V|)$, der Dominanztest in $O(|V|)$ pro Knoten. Gesamt $O(|V|^2)$, polynomiell.

- Die NTM arbeitet wie folgt:

- Entscheide für jeden Knoten $v \in V$ nichtdeterministisch, ob er in D aufgenommen wird.
- Prüfe $|D| \leq k$; lehne sonst ab.
- Prüfe für jeden Knoten $v \in V$, ob $v \in D$ gilt oder ein Nachbar von v in D liegt; lehne sonst ab.
- Sonst akzeptiere.

Korrektheit: Existiert ein Dominating Set der Größe $\leq k$, dann wählt eine Folge von Entscheidungen genau dieses aus. Dieser Rechenweg akzeptiert. Existiert keines, dann lehnt jeder Rechenweg ab.

Laufzeit: $O(|V|)$ Rateschritte plus $O(|V|^2)$ für die Prüfungen, also $O(|V|^2)$, polynomiell.

Also gilt DOMINATING SET $\in$ NP. □

2 Vorlesungsbeweise

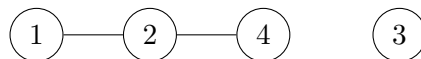
A3 $P \subseteq NP$

Vorlesung (Skript)

Zeigen Sie: $P \subseteq NP$.

Beispiel $P \subseteq NP$: ein P-Problem als NP-Problem

Instanz. $L = \text{ERREICHBARKEIT}$: Gegeben ein Graph G und Knoten s, t – gibt es einen Weg von s nach t ? Das liegt in P (Breitensuche, $O(|V| + |E|)$). Konkret $s = 1, t = 4$:



Kern. Der Verifizierer $V(x, c)$ startet die Breitensuche auf $x = (G, s, t)$ und ignoriert c komplett. Sie findet $1 \rightarrow 2 \rightarrow 4$, also akzeptiert V für jedes Zertifikat – etwa das leere Wort $c = \varepsilon$. Ein akzeptiertes Zertifikat existiert.

Lösung.

Sei $L \in P$. Dann gibt es einen polynomiellen Algorithmus A , der L entscheidet.

Konstruktion: Fasse A als Verifizierer V auf: $V(x, c) := A(x)$ – V ignoriert das Zertifikat c und berechnet die Antwort selbst.

Korrektheit: Ist $x \in L$, akzeptiert $V(x, c)$ für jedes c (etwa das leere Wort) – ein akzeptiertes Zertifikat existiert. Ist $x \notin L$, verwirft $V(x, c)$ für jedes c – kein Zertifikat wird akzeptiert. Also $L = \{x \mid \exists c : V(x, c) = 1\}$.

Laufzeit: Dieselbe wie die von A , also polynomiell. Damit ist V ein polynomieller Verifizierer und $L \in NP$. $\square$

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 $k \in \mathbb{N}_0$.

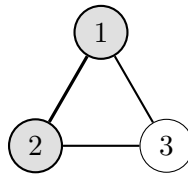
$M(G = (V, E), k)$

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.

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.

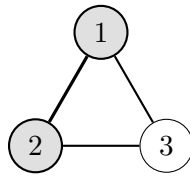
Laufzeit: Höchstens $T(|x|)$ Simulationsschritte mit konstantem Aufwand pro Schritt – $O(T(|x|))$, polynomiell. Das Zertifikat umfasst höchstens $T(|x|)$ Entscheidungen – polynomiell viele. Somit ist V ein polynomieller Verifizierer für L . $\square$

Sei L eine Sprache mit einem polynomiellen Verifizierer M : Für jedes $x \in L$ existiert ein Zertifikat c mit $M(x, c) = 1$ und Laufzeit $O(|x|^d)$; für $x \notin L$ akzeptiert M kein Zertifikat.

Zeigen Sie: Es gibt eine NDTM N , die L in Polynomialzeit entscheidet.

Beispiel Verifizierer $\subseteq$ NDTM: VC am Dreieck

Instanz. VERTEXCOVER am folgenden Dreieck G mit $k = 2$. Gegeben ist der Verifizierer: prüfe $|C| \leq k$ und jede Kante gedeckt.



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

Kern. Die NDTM rät einen String u und simuliert den Verifizierer. Der Rechenweg mit $C = \{1, 2\}$ akzeptiert – also akzeptiert die NDTM.

Lösung.

Konstruktion: Wähle p als das Laufzeitpolynom von M , also $p(|x|) = O(|x|^d)$. In $p(|x|)$ Schritten liest M höchstens $p(|x|)$ Zeichen des Zertifikats. Die NDTM N arbeitet bei Eingabe x in zwei Phasen:

- (1) *Raten:* Schreibe mit nichtdeterministischen Entscheidungen einen beliebigen String u mit $|u| \leq p(|x|)$ auf ein Arbeitsband.
- (2) *Verifizieren:* Simuliere M auf (x, u) deterministisch und übernehme dessen Antwort.

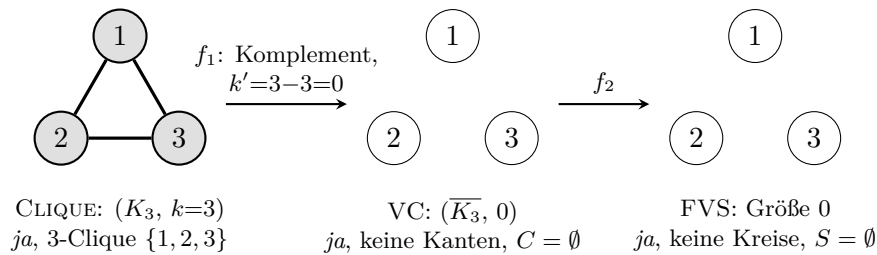
Korrektheit: Ist $x \in L$, dann existiert ein Zertifikat c mit $M(x, c) = 1$. M liest davon höchstens die ersten $p(|x|)$ Zeichen. Der Rechenweg, der genau diesen Anfang als u rät, akzeptiert – also akzeptiert N . Ist $x \notin L$, dann verwirft M jedes geratene u . Alle Rechenwege verwerfen – also verwirft N .

Laufzeit: Phase (1) braucht $p(|x|)$ Rateschritte, Phase (2) läuft in $O(|x|^d)$. Insgesamt polynomiell – N entscheidet L in nichtdeterministischer Polynomialzeit. $\square$

Zeigen Sie: Für alle Entscheidungsprobleme L_1, L_2, L_3 gilt $L_1 \leq_p L_2 \wedge L_2 \leq_p L_3 \implies L_1 \leq_p L_3$.

Beispiel Transitivität an einer Reduktionskette

Instanz. Die Kette $\text{CLIQUE} \leq_p \text{VC} \leq_p \text{FVS}$ an der Ja-Instanz $x = (K_3, 3)$ (Standardreduktionen $f_1(G, k) = (\overline{G}, |V| - k)$, dann f_2 für $\text{VC} \leq_p \text{FVS}$):



Kern. Die Verkettung $f_2 \circ f_1$ ist eine Reduktion $\text{CLIQUE} \leq_p \text{FVS}$: $x \in \text{CLIQUE} \iff x' \in \text{VC} \iff x'' \in \text{FVS}$ – hier dreimal Ja. Laufen f_1 in $O(n^a)$ und f_2 in $O(n^b)$, dann hat x' höchstens Größe $O(n^a)$, und die Verkettung läuft in $O(n^{ab})$ – polynomiell.

Lösung.

Sei R_1 eine Reduktion von L_1 auf L_2 mit Laufzeit $O(n^a)$ und R_2 eine Reduktion von L_2 auf L_3 mit Laufzeit $O(n^b)$.

Korrektheit: Mit $x'' = R_2(R_1(x))$ gilt

$$x \in L_1 \iff x' \in L_2 \quad \text{und} \quad x' \in L_2 \iff x'' \in L_3,$$

also $x \in L_1 \iff x'' \in L_3$. Damit ist $R_2 \circ R_1$ eine Reduktion von L_1 auf L_3 .

Laufzeit: Sei x eine L_1 -Instanz der Größe n .

- $R_1(x)$ braucht $O(n^a)$ Zeit; die Ausgabe x' hat damit Größe $O(n^a)$ (mehr kann in der Zeit nicht geschrieben werden).
- $R_2(x')$ braucht dann $O((n^a)^b) = O(n^{ab})$ Zeit.

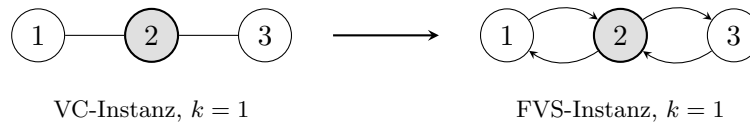
Die Verkettung ist also polynomiell. □

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.

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. $\square$

Beweisen Sie: Sei L_0 eine NP-vollständige Sprache. Falls $L_0 \in P$, dann gilt $P = NP$.

Beispiel L_0 NP-vollständig $\wedge L_0 \in P \Rightarrow P = NP$ konkret

Instanz. $L_0 = 3\text{-SAT}$ ist NP-vollständig, und ein Algorithmus A entscheide 3-SAT in $O(n^3)$.

Kern. Nimm ein beliebiges $L \in \text{NP}$, etwa CLIQUE : Da L_0 NP-vollständig ist, existiert eine Reduktion $f : \text{CLIQUE} \leq_p 3\text{-SAT}$, etwa mit Laufzeit $O(n^2)$. Berechne zur Eingabe x erst $f(x)$ und wende dann A an – das entscheidet CLIQUE korrekt ($x \in \text{CLIQUE} \iff f(x) \in 3\text{-SAT}$) in $O(n^2) + O((n^2)^3) = O(n^6)$, polynomiell. So liegt jedes $L \in \text{NP}$ in P , also $P = NP$.

Lösung.

$P \subseteq NP$ gilt stets. Zu zeigen bleibt $NP \subseteq P$.

Sei $L \in NP$ beliebig. Da L_0 NP-vollständig ist, gibt es eine Reduktion f von L auf L_0 mit Laufzeit $O(n^a)$. Sei A ein Algorithmus, der L_0 in Zeit $O(n^c)$ entscheidet.

So entscheidet man L : Berechne $f(x)$ und wende A darauf an. Das ist korrekt, denn $x \in L \iff f(x) \in L_0$. Und polynomiell: $f(x)$ hat Größe $O(n^a)$, also braucht A darauf $O((n^a)^c) = O(n^{ac})$.

Also gilt $L \in P$ für jedes $L \in NP$. Zusammen mit $P \subseteq NP$ folgt $P = NP$. $\square$

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

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

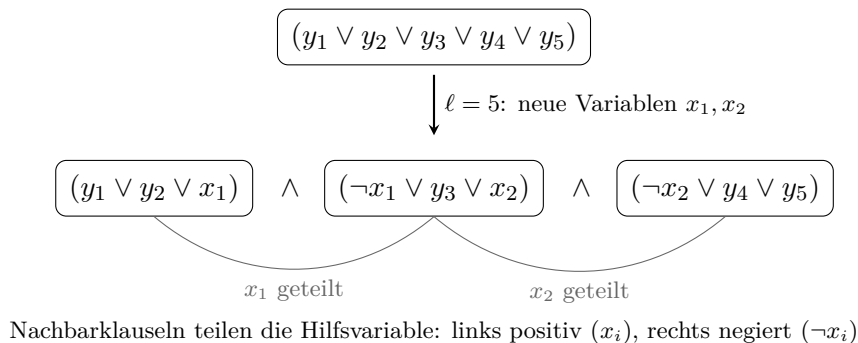
Ist C erfüllt, etwa durch y_3 , dann erfüllt $x_1 = \text{wahr}$, $x_2 = \text{falsch}$ die Kette: K_1 durch x_1 , K_2 durch y_3 , K_3 durch $\neg x_2$.

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 k -Clique:

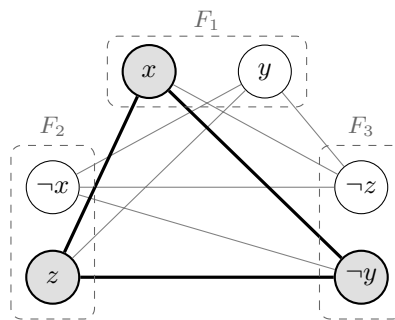
Eingabe: Ein ungerichteter Graph $G = (V, E)$ und eine Zahl $k \geq 1$. Eine Clique ist eine Teilmenge $C \subseteq V$ mit $\{u, v\} \in E$ für alle $u, v \in C$ mit $u \neq v$.

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.

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 $\text{SAT} \leq_p k\text{-CLIQUE}$:

Idee: Ein Knoten pro Literalvorkommen; Kanten nur zwischen verträglichen Literalen verschiedener Klauseln – eine m -Clique wählt dann pro Klausel ein wahres Literal.

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. $\square$

Problem Knapsack: Gegeben n Gegenstände mit Gewichten w_i und Profiten p_i , Kapazität B , Zielprofit P . Gibt es $S \subseteq [n]$ mit $\sum_{i \in S} w_i \leq B$ und $\sum_{i \in S} p_i \geq P$?

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

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

Beispiel SubsetSum $\leq_p$ Knapsack: Zahlen einsetzen

Instanz. $c = (2, 3, 5)$, $K = 5$ mit Lösung $\{1, 2\}$: $2 + 3 = 5$.

Kern. Die Reduktion setzt $w_i = p_i = c_i$ und $B = P = K$ – die beiden Knapsack-Schranken zwingen die gemeinsame Summe auf exakt K . Bildinstanz: Gewichte = Profite = $(2, 3, 5)$, $B = P = 5$; die Auswahl $\{1, 2\}$ hat Gewicht $5 \leq B$ und Profit $5 \geq P$ – die Ja-Instanz bleibt Ja.

Lösung.

∈ **NP**: Eingabe: Gewichte w_i , Profite p_i , Schranken B, P und Zertifikat $S \subseteq [n]$.

Der Verifizierer arbeitet wie folgt:

- Prüfe $\sum_{i \in S} w_i \leq B$; lehne sonst ab.
- Prüfe $\sum_{i \in S} p_i \geq P$; lehne sonst ab.
- Sonst akzeptiere.

Existiert eine gültige Auswahl, dann ist sie ein Zertifikat, das akzeptiert wird. Sonst wird jedes Zertifikat abgelehnt. Zwei Summen bilden und vergleichen – $O(n)$, polynomiell. Also gilt $\text{KNAPSACK} \in \text{NP}$.

Konstruktion: Sei $(c_1, \dots, c_n, K)$ eine SUBSETSUM-Instanz. Setze $w_i := c_i$, $p_i := c_i$ für alle i sowie $B := K$ und $P := K$.

Beweis hin:

- Sei S eine Menge mit $\sum_{i \in S} c_i = K$.
- Dann ist das Gewicht von S gleich $K \leq B$.
- Ebenso ist der Profit von S gleich $K \geq P$.
- Also ist S eine Knapsack-Lösung.

Beweis zurück:

- Sei S eine Knapsack-Lösung.
- Gewicht und Profit von S sind dieselbe Zahl $\sum_{i \in S} c_i$.
- Da S die Kapazität einhält, gilt $\sum_{i \in S} c_i \leq B = K$.
- Da S den Zielprofit erreicht, gilt $\sum_{i \in S} c_i \geq P = K$.
- Somit ist $\sum_{i \in S} c_i = K$ – eine SubsetSum-Lösung.

Laufzeit: Werte kopieren, $O(n)$, polynomiell.

Da SUBSETSUM NP-vollständig ist und $\text{KNAPSACK} \in \text{NP}$, ist KNAPSACK NP-vollständig. $\square$

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.

Lösung.

∈ **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 ∈ 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.

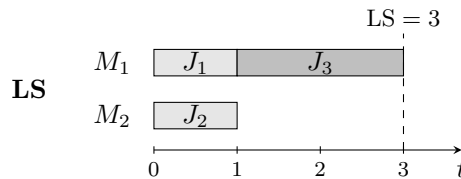
Laufzeit: Gesamtsumme berechnen und zwei Zahlen anhängen – $O(n)$.

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

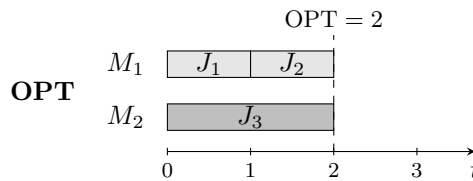
$P \parallel C_{\max}$: n Jobs mit Zeiten $p_1, \dots, p_n$, m Maschinen; minimiere $C_{\max}$. ListScheduling legt jeden Job der Reihe nach auf die Maschine mit minimaler Last. Zeigen Sie: Für alle I gilt $LS(I)/OPT(I) \leq 2 - 1/m$.

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*:

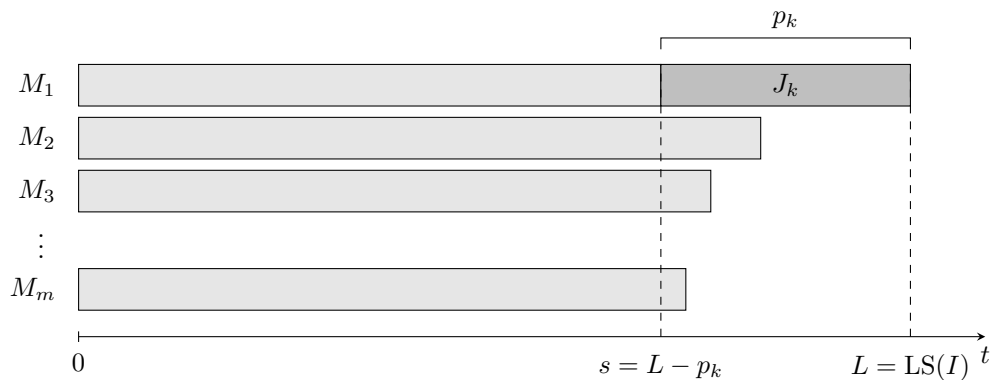


Lösung.

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

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



$$s \leq \frac{1}{m} (\sum_i p_i - p_k), \text{ denn alle Maschinen sind bis mindestens } s \text{ gefüllt}$$

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 $\text{LS}(I) \leq \left(2 - \frac{1}{m}\right) \text{OPT}(I)$. □

KNAPSACK: Gegenstände mit Gewichten w_i , Profiten p_i , Kapazität B ; maximiere den Profit. Greedy sortiert absteigend nach Profitdichte p_i/w_i und packt in dieser Reihenfolge jeden Gegenstand ein, der noch passt. ModifiedGreedy (MGA) gibt das Bessere aus Greedy-Lösung und profitreichstem Einzel-Gegenstand aus.

Zeigen Sie: $\text{OPT}(I) \leq 2 \cdot \text{MGA}(I)$ für alle Eingaben I .

Beispiel MGA-Güte 2: an einer konkreten Instanz

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

Kern. Greedy geht die Liste nach Dichte durch und packt jeweils, was noch passt: erst $(1, 1)$ und $(3, 3)$, dann ist $(11, 13)$ zu schwer und wird übersprungen, danach noch $(5, 6)$ und $(1, 3)$. Greedy-Wert $\text{GA} = 10$. Das übersprungene $(11, 13)$ ist zugleich der beste Einzel-Gegenstand, Profit $p_{\max} = 11$; genau diesen Rückstand fängt $\text{MGA} = \max\{\text{GA}, p_{\max}\} = 11$ ab. Das Optimum $\{(11, 13), (3, 3)\}$ liefert $\text{OPT} = 14$, und es gilt

$$\text{OPT} = 14 \leq \text{GA} + p_{\max} = 21 \leq 2 \text{MGA} = 22.$$

Lösung.

Idee: Greedy geht die Items nach Dichte durch und packt jedes, das noch passt; passt eines nicht, überspringt es und macht weiter. Sein Rückstand zum Optimum ist höchstens der Profit *eines* Items – des ersten übersprungenen. Genau dieses fängt der beste Einzel-Gegenstand ab.

Nummeriere nach Dichte: $p_1/w_1 \geq \dots \geq p_n/w_n$. Sei $k + 1$ der erste Gegenstand, den Greedy nicht einpacken kann.

Schritt 1 (Relaxierung): Erlaubt man, Gegenstände anteilig einzupacken (fraktionales Knapsack mit Optimum OPT_f), wird der Lösungsraum größer – bei einem Maximierungsproblem gilt $\text{OPT}(I) \leq \text{OPT}_f(I)$.

Schritt 2 (fraktionales Optimum): Die beste fraktionale Lösung füllt den Rucksack in Dichte-Reihenfolge: die Gegenstände $1, \dots, k$ vollständig, vom Gegenstand $k + 1$ einen Bruchteil. (Jede andere Lösung ließe sich verbessern, indem man Anteile geringerer Dichte durch Anteile höherer Dichte ersetzt.) Der Bruchteil ist höchstens 1, also $\text{OPT}_f(I) \leq p_1 + \dots + p_k + p_{k+1}$. Und Greedy packt die Gegenstände $1, \dots, k$ ebenfalls ein, somit $p_1 + \dots + p_k \leq \text{GA}(I)$. Zusammen:

$$\text{OPT}_f(I) \leq p_1 + \dots + p_k + p_{k+1} \leq \text{GA}(I) + p_{k+1}.$$

Schritt 3 (Kombination): Mit $p_{k+1} \leq p_{\max}$ und $\text{MGA}(I) = \max\{\text{GA}(I), p_{\max}\}$ folgt

$$\text{OPT}(I) \leq \text{GA}(I) + p_{\max} \leq 2 \max\{\text{GA}(I), p_{\max}\} = 2 \cdot \text{MGA}(I). \quad \square$$

3 Reduktionen: NP-Schwere und NP-Vollständigkeit

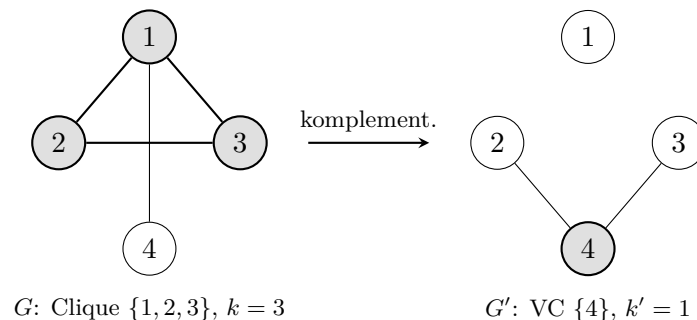
A15 VertexCover ist NP-vollständig

Präsenz 10.2 / Hausaufgabe 9.1

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' .

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 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.

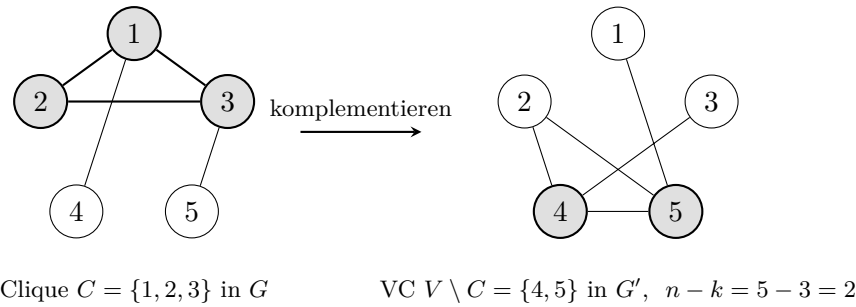
Existiert ein Vertex Cover C mit $|C| \leq k$, dann gibt es ein Zertifikat dafür. Dieses wird vom Verifizierer akzeptiert und sonst abgelehnt. Größen- und Kantenprüfung laufen in $O(|V| + |E|)$. Also gilt VERTEXCOVER ∈ NP. (NDTM-Variante: statt das Zertifikat C als Eingabe zu erhalten, rate es im ersten Schritt nichtdeterministisch.)

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 – $O(|V|^2)$.

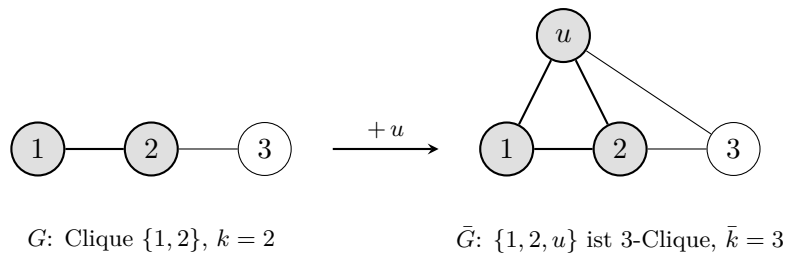
Da VERTEXCOVER $\in$ NP und NP-schwer ist, folgt: VERTEXCOVER ist NP-vollständig. □

Problem k -CliqueUniversal: Gegeben ein Graph $G = (V, E)$ und $k \geq 1$, wobei ein $u \in V$ existiert, das mit allen anderen Knoten verbunden ist. **Entscheide:** Existiert eine Clique C mit $|C| \geq k$?

Zeigen Sie, dass k -CLIQUEUNIVERSAL NP-vollständig ist.

Beispiel k -CliqueUniversal NP-vollständig

Instanz. CLIQUE am folgenden Pfad G mit $k = 2$.



Kern. Ein neuer Knoten u wird mit allen Knoten verbunden, $\bar{k} = k + 1 = 3$. Damit ist u universell, und die Clique $\{1, 2\}$ wächst zur Clique $\{1, 2, u\}$ der Größe $3 \geq \bar{k}$ in $\bar{G}$.

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 $\{x, y\} \subseteq C$, ob $\{x, y\} \in E$ gilt.
- Lehne ab, falls für ein Paar $\{x, y\} \notin E$ gilt.
- Sonst akzeptiere.

Existiert eine Clique C mit $|C| \geq k$, dann gibt es ein Zertifikat dafür. Dieses wird vom Verifizierer akzeptiert und sonst abgelehnt. Größenprüfung und Prüfung aller Paare laufen in $O(|V|^2)$, somit polynomiell. Also gilt k -CLIQUEUNIVERSAL ∈ NP.

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

Konstruktion: Sei $(G = (V, E), k)$ eine CLIQUE-Instanz. Wir konstruieren daraus eine k -CLIQUEUNIVERSAL-Instanz $(\bar{G} = (\bar{V}, \bar{E}), \bar{k})$ durch:

- $\bar{V} = V \cup \{u\}$ mit einem neuen Knoten u
- $\bar{E} = E \cup \{\{u, v\} \mid v \in V\}$
- $\bar{k} = k + 1$

Der neue Knoten u ist mit allen anderen verbunden, also ist $(\bar{G}, \bar{k})$ eine gültige Instanz.

Beweis Clique nach k -CliqueUniversal:

- Sei C eine Clique von G mit $|C| \geq k$.
- Da u mit allen Knoten verbunden ist, ist u auch mit allen Knoten aus C verbunden.
- Damit ist $C \cup \{u\}$ eine Clique in $\bar{G}$.
- Dann gilt auch $|C \cup \{u\}| \geq k + 1 = \bar{k}$.

Beweis k -CliqueUniversal nach Clique:

- Sei $\bar{C}$ eine Clique von $\bar{G}$ mit $|\bar{C}| \geq \bar{k}$.
- Setze $C = \bar{C} \setminus \{u\}$, dann gilt $|C| \geq \bar{k} - 1 = k$.
- Alle Knoten in C liegen in V und alle Kanten zwischen ihnen liegen in E , da nur Kanten zu u neu hinzugefügt wurden.
- Damit ist C eine Clique in G mit $|C| \geq k$.

Laufzeit: Ein Knoten und $|V|$ Kanten werden hinzugefügt – $O(|V|)$, also bleibt die Transformation polynomiell.

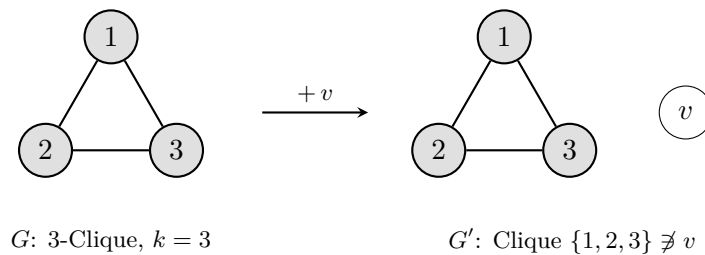
Da k -CLIQUEUNIVERSAL $\in$ NP und NP-schwer ist, folgt: k -CLIQUEUNIVERSAL ist NP-vollständig. $\square$

Problem Clique-Nomember: Gegeben $G = (V, E)$, ein Knoten $v \in V$ und k . Gibt es eine k -Clique, die v *nicht* enthält?

Zeigen Sie NP-Vollständigkeit durch Reduktion von CLIQUE; CLIQUE-NOMEMBER $\in$ NP dürfen Sie annehmen.

Beispiel Clique-Nomember NP-vollständig

Instanz. CLIQUE am folgenden Dreieck G mit $k = 3$.



Kern. Ein neuer isolierter Knoten v kommt hinzu; Ausgabe (G', v, k) . Die 3-Clique $\{1, 2, 3\}$ liegt ganz in G und enthält v nicht – sie ist eine k -Clique in G' ohne v .

Lösung.

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

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

- $G' = (V \cup \{v\}, E)$ mit einem neuen, isolierten Knoten $v \notin V$
- k bleibt unverändert

Beweis Clique nach Clique-Nomember:

- Sei C eine k -Clique in G .
- Da keine Kanten entfernt wurden, ist C auch eine k -Clique in G' .
- Da $v \notin V$, gilt $v \notin C$.
- Also ist C eine k -Clique in G' , die v nicht enthält.

Beweis Clique-Nomember nach Clique:

- Sei C eine k -Clique in G' mit $v \notin C$.
- Dann gilt $C \subseteq V$, denn v ist der einzige neue Knoten.
- Alle Kanten zwischen Knoten aus C liegen in E , denn es wurden keine Kanten hinzugefügt.
- Also ist C eine k -Clique in G .

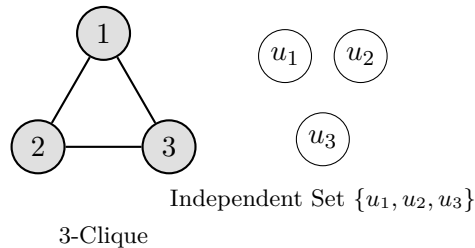
Laufzeit: Graph kopieren und einen isolierten Knoten anhängen – $O(|V| + |E|)$, also bleibt die Transformation polynomiell.

Da $\text{CLIQUE-NOMEMBER} \in \text{NP}$ (nach Annahme) und NP-schwer ist, folgt: CLIQUE-NOMEMBER ist 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.

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 – $O(k) \subseteq O(|V|)$.

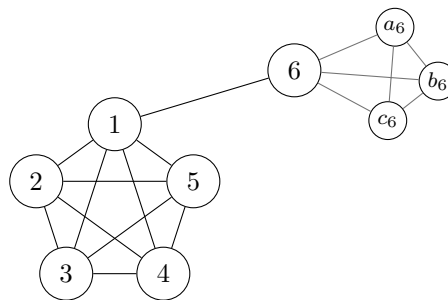
Da CLIQUE NP-schwer ist und $\text{CLIQUE} \leq_p \text{CLIQUEANDINDEPENDENTSET}$ gilt, folgt: CLIQUE-ANDINDEPENDENTSET ist NP-schwer. $\square$

Problem 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.

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.

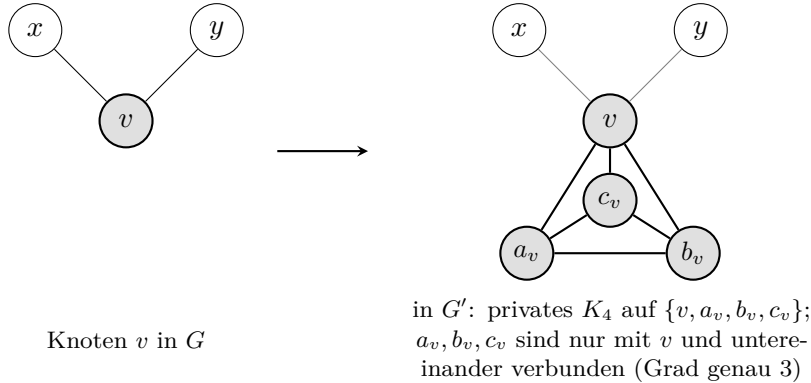
Laufzeit: Gradprüfung $O(|V| + |E|)$, Größen- und Paar-Prüfung $O(|V|^2)$ – gesamt $O(|V|^2)$. Also gilt $k\text{-CLIQUE-DEG-3} \in \text{NP}$.

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

Konstruktion: Sei $(G = (V, E), k)$ eine $k\text{-CLIQUE}$ -Instanz, o.B.d.A. $k \geq 5$. Wir konstruieren daraus eine $k\text{-CLIQUE-DEG-3}$ -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 $k\text{-Clique}$ nach $k\text{-CLIQUE-DEG-3}$ (Fall $k \geq 5$):

- Sei C eine $k\text{-Clique}$ in G .
- Da keine Kanten entfernt wurden, ist C auch eine $k\text{-Clique}$ in G' .

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

- Sei C' eine $k\text{-Clique}$ 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 $k\text{-Clique}$ 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 – $O(|V|^4)$.

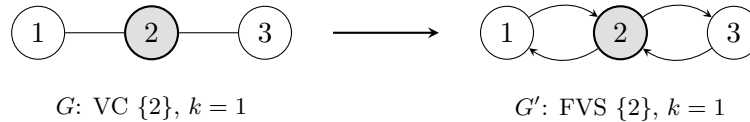
Da k -CLIQUE-DEG-3 $\in$ NP und NP-schwer ist, folgt: k -CLIQUE-DEG-3 ist NP-vollständig.
 $\square$

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 $|X| = 1 \leq k$.

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.

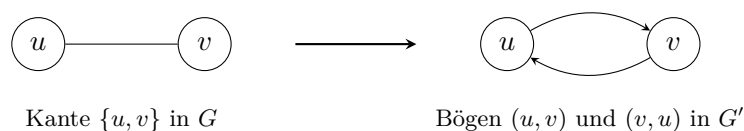
Existiert ein Feedback Vertex Set X mit $|X| \leq k$, dann gibt es ein Zertifikat dafür. Dieses wird vom Verifizierer akzeptiert und sonst abgelehnt. Größenprüfung und topologische Sortierung laufen in $O(|V| + |E|)$. Also gilt FEEDBACKVERTEXSET ∈ NP.

NP-schwer: Zeige, dass FEEDBACKVERTEXSET NP-schwer ist durch die Reduktion VERTEXCOVER $\leq_p$ 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 – $O(|E|)$.

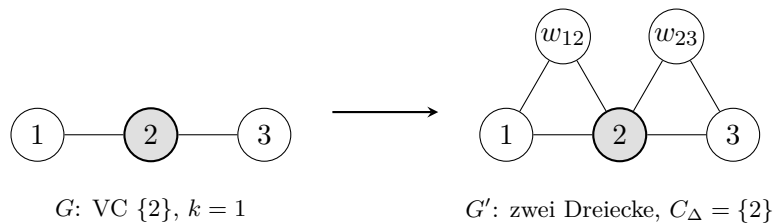
Da $\text{FEEDBACKVERTEXSET} \in \text{NP}$ und NP-schwer ist, folgt: FEEDBACKVERTEXSET ist NP-vollständig. $\square$

Problem Δ -COVER: Gegeben $G = (V, E)$ und k . Gibt es $C_\Delta \subseteq V$ mit $|C_\Delta| \leq k$, sodass jedes Dreieck (3-Clique) von G mindestens einen Knoten in C_Δ hat?

Zeigen Sie, dass Δ -COVER NP-vollständig ist.

Beispiel Δ -Cover NP-vollständig

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



Kern. Jede Kante $e = \{u, v\}$ wird durch einen neuen Knoten w_e zum Dreieck $\{u, v, w_e\}$ ergänzt; k bleibt. Beide Dreiecke von G' enthalten den Knoten 2, also ist das Vertex Cover $\{2\}$ auch eine Dreiecksüberdeckung mit $|C_\Delta| = 1 \leq k$.

Lösung.

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

- Falls $|C_\Delta| > k$, lehne ab.
- Prüfe für jedes Knotentripel $\{a, b, c\} \subseteq V$, ob es ein Dreieck bildet, also ob $\{a, b\}, \{b, c\}, \{a, c\} \in E$ gilt.
- Lehne ab, falls ein solches Dreieck keinen Knoten in C_Δ hat.
- Sonst akzeptiere.

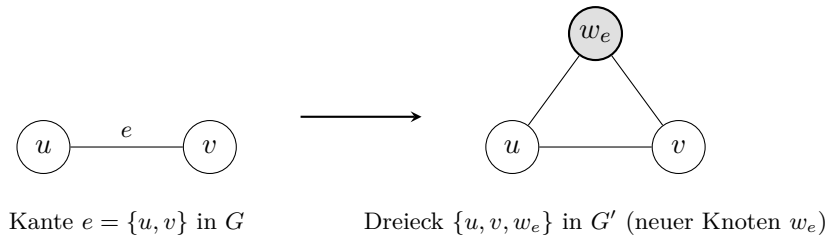
Existiert eine Dreiecksüberdeckung C_Δ mit $|C_\Delta| \leq k$, dann gibt es ein Zertifikat dafür. Dieses wird vom Verifizierer akzeptiert und sonst abgelehnt. Die Größenprüfung läuft in $O(|V|)$ und das Prüfen aller Tripel in $O(|V|^3)$, somit polynomiell. Also gilt Δ -COVER $\in$ NP.

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

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

- Für jede Kante $e = \{u, v\} \in E$ füge einen neuen Knoten w_e hinzu.
- $V' = V \cup \{w_e \mid e \in E\}$
- $E' = E \cup \{\{w_e, u\}, \{w_e, v\} \mid e = \{u, v\} \in E\}$
- $k' = k$

So wird aus jeder Kante $e = \{u, v\}$ ein Dreieck $\{u, v, w_e\}$.



Beweis VertexCover nach Δ -Cover:

- Sei C ein Vertex Cover von G mit $|C| \leq k$.
- Jeder neue Knoten w_e hat genau die zwei Nachbarn u und v .
- Somit ist jedes Dreieck in G' entweder ein Gadget-Dreieck $\{u, v, w_e\}$ oder ein aus G geerbtes Dreieck $\{a, b, c\}$ mit $a, b, c \in V$ (wegen $E \subseteq E'$).
- Falls Gadget-Dreieck $\{u, v, w_e\}$: C deckt die Kante $e = \{u, v\}$, also gilt $u \in C$ oder $v \in C$.
- Falls geerbtes Dreieck $\{a, b, c\}$: C deckt die Kante $\{a, b\} \in E$, also liegt ein Eckknoten in C .
- Somit hat jedes Dreieck in G' mindestens einen Knoten in C .
- Also ist C eine Dreiecksüberdeckung für G' mit $|C| \leq k'$.

Beweis Δ -Cover nach VertexCover:

- Sei C_Δ eine Dreiecksüberdeckung von G' mit $|C_\Delta| \leq k'$.
- Falls C_Δ ein w_e mit $e = \{u, v\}$ enthält, ersetze w_e durch den Endpunkt u .
- Das ist erlaubt, denn w_e liegt in genau einem Dreieck $\{u, v, w_e\}$, und das wird auch durch u gedeckt.
- Die Menge wird dabei nicht größer.
- So entsteht eine Dreiecksüberdeckung C von G' mit $C \subseteq V$ und $|C| \leq k$.
- Jede Kante $e = \{u, v\} \in E$ bildet mit w_e das Gadget-Dreieck $\{u, v, w_e\}$.
- C trifft dieses Dreieck.
- Da $w_e \notin C$, gilt $u \in C$ oder $v \in C$.
- Also ist C ein Vertex Cover für G mit $|C| \leq k$.

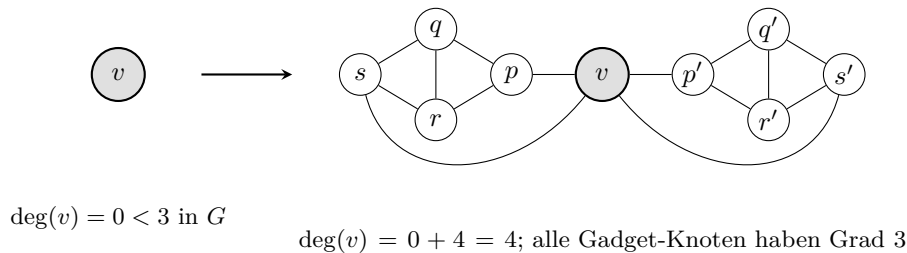
Laufzeit: Pro Kante ein Knoten und zwei Kanten anlegen – $O(|E|)$, also bleibt die Transformation polynomiell.

Da Δ -COVER $\in$ NP und NP-schwer ist, folgt: Δ -COVER ist NP-vollständig. $\square$

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 $p = s = 2, q = 1, r = 3$.

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.

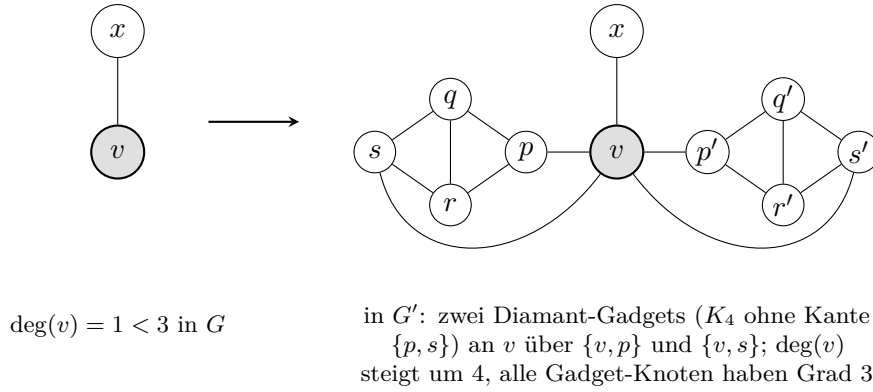
NP-schwer: Zeige die NP-Schwere durch Reduktion von 3-COLOR auf die Grad-3-Variante. 3-COLOR ist bereits als NP-vollständig bekannt.

Idee: Angehängte Gadgets heben den Grad untergradiger Knoten, ändern aber die 3-Färbbarkeit nicht.

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 – $O(|V|)$.

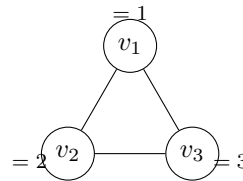
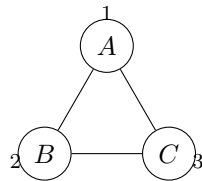
Da die Grad-3-Variante in NP liegt und NP-schwer ist, folgt: Sie ist NP-vollständig. $\square$

Problem k -COLOR-PRECOLORING: Gegeben $G = (V, E)$, $k \in \mathbb{N}_{>0}$ und paarweise verschiedene Knoten $v_1, \dots, v_k \in V$. Existiert eine Färbung $f : V \rightarrow [k]$ mit $f(v_i) = i$ für alle i und $f(u) \neq f(v)$ für alle $\{u, v\} \in E$?

Zeigen Sie, dass k -COLOR-PRECOLORING NP-vollständig ist.

Beispiel k -COLOR-PRECOLORING

Instanz. k -COLOR mit $k = 3$ am Dreieck $\{A, B, C\}$.



G : 3-Färbung $A=1, B=2, C=3$ disjunktes K_3 , vorgefärbt $f(v_i) = i$

Kern. Zu G kommt ein disjunktes K_3 mit vorgefärbten Knoten v_1, v_2, v_3 hinzu (keine Kanten nach G); die Vorfärbung $f(v_i) = i$ erzwingt nichts in G . Die 3-Färbung von G plus $f(v_i) = i$ ist eine gültige Färbung von $G' = G \dot{\cup} K_3$ mit Vorgabe.

Lösung.

∈ **NP**: Eingabe: Graph $G = (V, E)$, Zahl k , Knoten $v_1, \dots, v_k$ und Zertifikat $f : V \rightarrow [k]$. Der Verifizierer arbeitet wie folgt:

- Prüfe für jedes $i \in [k]$, ob $f(v_i) = i$ gilt.
- Prüfe für jede Kante $\{u, v\} \in E$, ob $f(u) \neq f(v)$ gilt.
- Lehne ab, falls eine Prüfung fehlschlägt.
- Sonst akzeptiere.

Existiert eine gültige Färbung mit Vorgabe, dann gibt es ein Zertifikat dafür. Dieses wird vom Verifizierer akzeptiert und sonst abgelehnt. Vorgaben- und Kantenprüfung laufen in $O(k + |E|)$, somit polynomiell. Also gilt k -COLOR-PRECOLORING ∈ NP.

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

Konstruktion: Sei (G, k) eine k -COLOR-Instanz. Wir konstruieren daraus eine k -COLOR-PRECOLORING-Instanz durch:

- Füge k neue Knoten $v_1, \dots, v_k$ hinzu.
- Verbinde die v_i untereinander vollständig zum K_k .
- Kanten zwischen K_k und G gibt es nicht.
- Gib $(G' = G \dot{\cup} K_k, k, v_1, \dots, v_k)$ aus.

Beweis k -Color nach k -COLOR-PRECOLORING:

- Sei f eine k -Färbung von G .
- Erweitere f durch $f(v_i) = i$ für alle $i \in [k]$.
- Die v_i sind nur untereinander verbunden und paarweise verschieden gefärbt.
- Somit ist f eine gültige Färbung von G' und erfüllt die Vorgabe.

Beweis k -COLOR-PRECOLORING nach k -Color:

- Sei f eine gültige Färbung von G' mit $f(v_i) = i$.
- Da alle G -Kanten in G' liegen, ist die Einschränkung von f auf V eine k -Färbung von G .

Laufzeit: Die neue Clique K_k anlegen – $O(k^2)$, also bleibt die Transformation polynomiell.

Da k -COLOR-PRECOLORING $\in$ NP und NP-schwer ist, folgt: k -COLOR-PRECOLORING ist NP-vollständig. $\square$

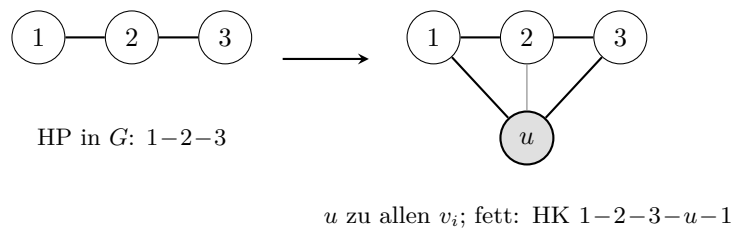
Problem HamiltonianPath: Gegeben ein ungerichteter Graph $G = (V, E)$. **Entscheide:** Existiert ein Pfad in G , der jeden Knoten genau einmal besucht?

Problem HamiltonianCycle: Gegeben ein ungerichteter Graph $G = (V, E)$. **Entscheide:** Existiert ein Kreis in G , der jeden Knoten genau einmal besucht?

Geben Sie eine polynomielle Reduktion von HAMILTONIANPATH auf HAMILTONIANCYCLE an.

Beispiel HamiltonianPath $\leq_p$ HamiltonianCycle

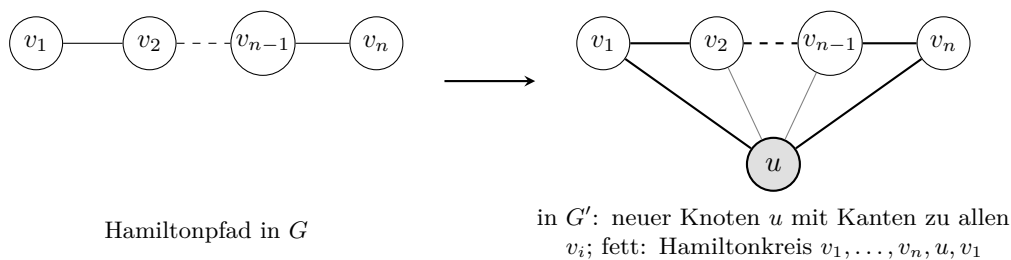
Instanz. HAMILTONIANPATH am folgenden Pfad G .



Kern. Ein neuer universeller Knoten u wird mit allen Knoten verbunden. Der Hamiltonpfad 1-2-3 schließt sich über die Kanten $\{3, u\}$ und $\{u, 1\}$ zum Hamiltonkreis 1-2-3- u -1 in G' .

Lösung.

Konstruktion: Sei $G = (V, E)$ gegeben. Füge einen neuen universellen Knoten u hinzu: $G' = (V \cup \{u\}, E \cup \{\{u, v\} \mid v \in V\})$.



Beweis HamiltonianPath nach HamiltonianCycle:

- Sei $v_1, \dots, v_n$ ein Hamiltonpfad in G .
- Da u universell ist, existieren die Kanten $\{v_n, u\}$ und $\{u, v_1\}$.
- Also ist $v_1, \dots, v_n, u, v_1$ ein Hamiltonkreis in G' .

Beweis HamiltonianCycle nach HamiltonianPath:

- Sei C ein Hamiltonkreis in G' .
- C besucht u genau einmal, zwischen zwei Nachbarn $v_i, v_j \in V$.
- Entferne u aus dem Kreis.

- Dann bleibt ein Pfad von v_i nach v_j , der alle Knoten aus V genau einmal besucht.
- Dieser Pfad benutzt nur Kanten aus E , denn neu sind nur die Kanten an u .
- Also ist er ein Hamiltonpfad in G .

Laufzeit: Einen Knoten und $|V|$ neue Kanten anlegen – $O(|V|)$, polynomiell.

□

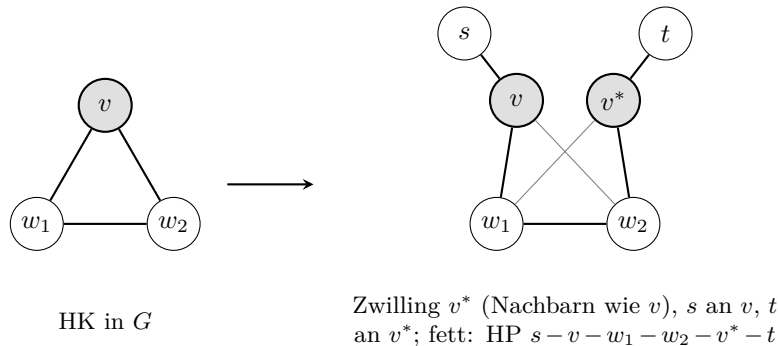
Problem HamiltonianCycle: Gegeben ein ungerichteter Graph $G = (V, E)$. **Entscheide:** Existiert ein Kreis in G , der jeden Knoten genau einmal besucht?

Problem HamiltonianPath: Gegeben ein ungerichteter Graph $G = (V, E)$. **Entscheide:** Existiert ein Pfad in G , der jeden Knoten genau einmal besucht?

Geben Sie eine polynomielle Reduktion von HAMILTONIANCYCLE auf HAMILTONIANPATH an.

Beispiel HamiltonianCycle $\leq_p$ HamiltonianPath

Instanz. HAMILTONIANCYCLE am Dreieck auf $\{v, w_1, w_2\}$.



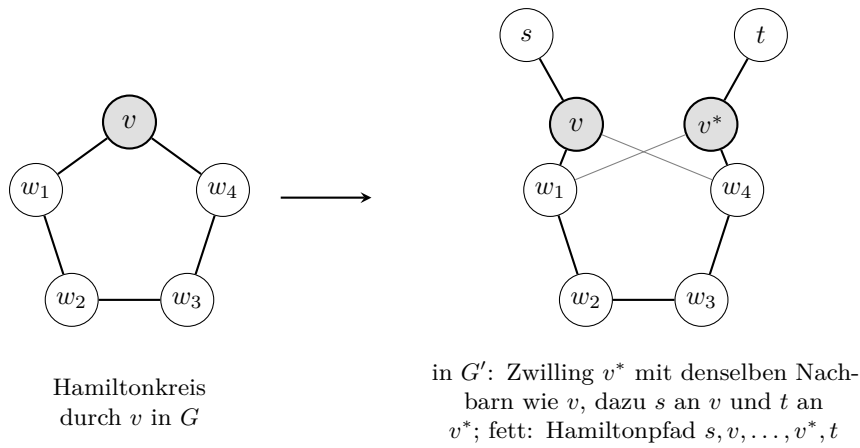
Kern. Ein Zwilling v^* erbt die Nachbarn von v , dazu kommen die Grad-1-Knoten s an v und t an v^* . Der Hamiltonkreis wird bei v aufgetrennt und zum Hamiltonpfad $s-v-w_1-w_2-v^*-t$ verlängert.

Lösung.

Idee: Den Kreis bei v aufschneiden: Ein Zwilling v^* erbt v s Nachbarn, zwei Pendants s, t erzwingen die Pfadenden.

Konstruktion: Sei $G = (V, E)$ gegeben, o.B.d.A. $|V| \geq 3$. Wähle einen beliebigen Knoten v . Wir konstruieren G' durch:

- Füge einen Zwilling v^* mit denselben Nachbarn wie v hinzu.
- Füge einen Knoten s mit der einzigen Kante $\{s, v\}$ hinzu.
- Füge einen Knoten t mit der einzigen Kante $\{t, v^*\}$ hinzu.



Beweis HamiltonianCycle nach HamiltonianPath:

- Sei $v, w_1, \dots, w_{n-1}, v$ ein Hamiltonkreis in G .
- Da v^* dieselben Nachbarn wie v hat, existiert die Kante $\{w_{n-1}, v^*\}$.
- Also ist $s, v, w_1, \dots, w_{n-1}, v^*, t$ ein Hamiltonpfad in G' .

Beweis HamiltonianPath nach HamiltonianCycle:

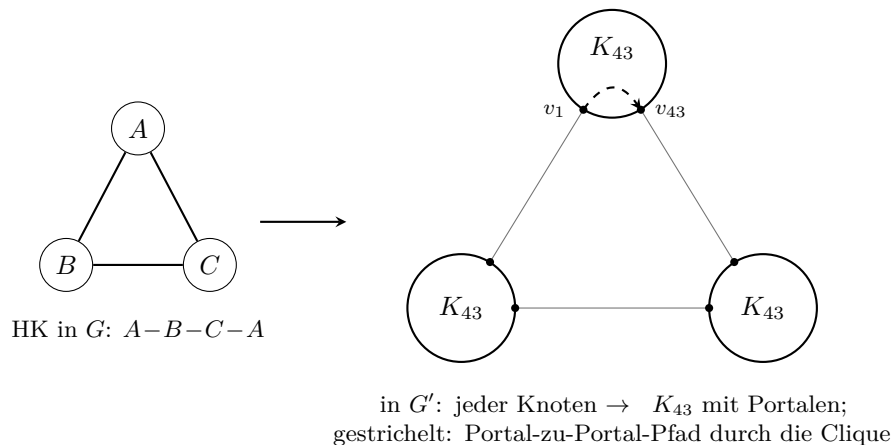
- Sei P ein Hamiltonpfad in G' .
- Die Knoten s und t haben Grad 1, also sind sie die beiden Endpunkte von P .
- Da s nur mit v und t nur mit v^* verbunden ist, hat P die Form $s, v, x_1, \dots, x_{n-1}, v^*, t$.
- Dabei sind $x_1, \dots, x_{n-1}$ alle Knoten aus $V \setminus \{v\}$.
- Da $\{x_{n-1}, v^*\} \in E'$ und v^* dieselben Nachbarn wie v hat, ist x_{n-1} auch ein Nachbar von v .
- Ersetze v^* durch v : Aus dem Mittelstück wird der Kreis $v, x_1, \dots, x_{n-1}, v$.
- Dieser Kreis besucht jeden Knoten von G genau einmal und benutzt nur Kanten aus E .
- Also ist er ein Hamiltonkreis in G .

Laufzeit: Zwillung v^* mit den bis zu $|V|$ Nachbarn von v anlegen, dazu die zwei Pendants s, t – $O(|V|)$, polynomiell. $\square$

Problem Hitchhiker's-HamiltonianCycle: Gegeben $G = (V, E)$ mit $\deg(v) \geq 42$ für alle $v \in V$. Gibt es einen Hamiltonkreis? Zeigen Sie die NP-Schwere durch Reduktion eines bekannten NP-schweren Problems.

Beispiel Hitchhiker's-HamiltonianCycle

Instanz. HAMILTONIANCYCLE am Dreieck G mit Hamiltonkreis $A-B-C-A$.

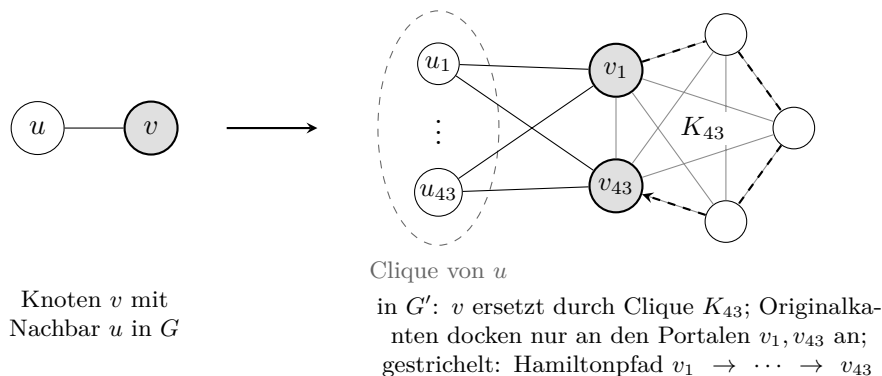


Kern. Jeder Knoten wird zur Clique K_{43} mit den Portalen v_1, v_{43} ; jede Originalkante liefert vier Portalkanten. Damit sind alle Grade mindestens 42, und der Hamiltonkreis überträgt sich: jede Clique wird als ein Portal-zu-Portal-Pfad durchlaufen.

Lösung.

Idee: Jeder Knoten wird zu einer K_{43} -Clique mit zwei Portalen – das erzwingt überall Grad ≥ 42 , ohne die Hamiltonizität zu ändern.

Konstruktion (Reduktion von HAMILTONIANCYCLE): Ersetze jeden Knoten v durch eine Clique K_{43} mit Knoten $v_1, \dots, v_{43}$. Für jede Originalkante $\{u, v\} \in E$ füge die vier Kanten $\{u_1, v_1\}, \{u_1, v_{43}\}, \{u_{43}, v_1\}, \{u_{43}, v_{43}\}$ hinzu (Portale sind v_1 und v_{43}).



Grade: innere Clique-Knoten haben Grad 42; die Portale v_1, v_{43} haben Grad ≥ 42 . Die Instanz ist also gültig.

Beweis HamiltonianCycle nach Hitchhiker's-HamiltonianCycle:

- Sei $v^{(1)}, \dots, v^{(n)}, v^{(1)}$ ein Hamiltonkreis in G .
- Durchlaufe die Clique jedes besuchten Knotens $v^{(j)}$ auf dem Hamiltonpfad $v_1^{(j)} \rightarrow \dots \rightarrow v_{43}^{(j)}$.
- Denn in einer Clique existiert dieser Pfad immer.
- Wechsle danach über die Portalkante $\{v_{43}^{(j)}, v_1^{(j+1)}\}$ zur nächsten Clique.
- Diese Portalkante existiert, denn $\{v^{(j)}, v^{(j+1)}\}$ ist eine Kante in G .
- Der letzte Wechsel führt über $\{v_{43}^{(n)}, v_1^{(1)}\}$ zurück zum Start.
- Somit wird jeder Knoten von G' genau einmal besucht.
- Also ist das ein Hamiltonkreis in G' .

Beweis Hitchhiker's-HamiltonianCycle nach HamiltonianCycle:

- Sei C' ein Hamiltonkreis in G' .
- Dann besucht C' in jeder Knoten-Clique alle 43 Knoten.
- Die inneren Clique-Knoten haben nur Nachbarn in der eigenen Clique.
- Kanten nach außen gibt es nur an den beiden Portalen v_1 und v_{43} .
- Somit durchläuft C' jede Clique als *ein* Segment von Portal zu Portal.
- Ziehe nun jede Clique zu ihrem Originalknoten zusammen.
- Dann wird aus C' ein Kreis in G , der jeden Knoten genau einmal besucht.
- Denn jede benutzte Portalkante stammt von einer Originalkante $\{u, v\} \in E$.
- Also ist das ein Hamiltonkreis in G .

Laufzeit: pro Knoten eine K_{43} -Clique aufbauen (43^2 Kanten), pro Originalkante vier Portalkanten – $O(43^2 \cdot |V| + |E|)$, polynomiell.

Da HAMILTONIANCYCLE NP-schwer ist, ist HITCHHIKER'S-HAMILTONIANCYCLE NP-schwer. $\square$

Problem SubsetSumCardinality: Gegeben $c_1, \dots, c_n \in \mathbb{N}_{>0}$ (n gerade) und K . Gibt es S mit $|S| = n/2$ und $\sum_{i \in S} c_i = K$? Zeigen Sie NP-Vollständigkeit; $\in \text{NP}$ dürfen Sie annehmen.

Beispiel SubsetSumCardinality NP-vollständig

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

Kern. Shift $c'_i = c_i + 1$ plus n Padding-Items der Größe 1:

$$c' = (3, 4, 6, 1, 1, 1), \quad \text{Zielwert } K + n = 8, \quad \text{Kardinalität } n = 3.$$

Die Bildmenge $\{1, 2\}$ mit einem Padding-Item aufgefüllt ergibt $S' = \{1, 2, 4\}$: Summe $3+4+1 = 8$ bei genau 3 Elementen.

Lösung.

Konstruktion: Sei $(c_1, \dots, c_n, K)$ eine SUBSETSUM-Instanz. Setze $c'_i := c_i + 1$ für $i \leq n$ (Shift) und $c'_i := 1$ für $i \in \{n+1, \dots, 2n\}$ (n Padding-Items). Gib $(c'_1, \dots, c'_{2n}, K+n)$ aus ($2n$ Items, verlangt $|S| = n$).

Beweis SubsetSum nach SubsetSumCardinality:

- Sei $S \subseteq [n]$ eine Lösung mit $\sum_{i \in S} c_i = K$.
- Dann gilt $\sum_{i \in S} c'_i = K + |S|$, denn jedes gewählte Item wurde um 1 geshiftet.
- Setze $S' := S \cup \{n+1, \dots, 2n - |S|\}$, fülle also mit $n - |S|$ Padding-Items auf.
- Dann gilt $|S'| = n$.
- Da jedes Padding-Item genau 1 beiträgt, folgt

$$\sum_{i \in S'} c'_i = K + |S| + (n - |S|) = K + n.$$

- Also ist S' eine Lösung der konstruierten Instanz.

Beweis SubsetSumCardinality nach SubsetSum:

- Sei S' eine Lösung mit $|S'| = n$ und $\sum_{i \in S'} c'_i = K + n$.
- Setze $S := S' \cap \{1, \dots, n\}$.
- Dann enthält S' genau $n - |S|$ Padding-Items.
- Da jedes Padding-Item genau 1 beiträgt, folgt

$$\sum_{i \in S} (c_i + 1) = (K + n) - (n - |S|) = K + |S|.$$

- Somit gilt $\sum_{i \in S} c_i = K$.
- Also ist S eine Lösung der SUBSETSUM-Instanz.

Laufzeit: n Zahlen um 1 shiften und n Padding-Items anhängen – $O(n)$, polynomiell.

Da SUBSETSUM NP-vollständig ist und SUBSETSUMCARDINALITY $\in \text{NP}$, ist SUBSETSUMCARDINALITY NP-vollständig. $\square$

Problem $(a_1=1)$ -SubsetSum: n Items mit Größen $a_i \in \mathbb{N}_{>0}$, wobei $a_1 = 1$, und Zielwert T . Gibt es $I \subseteq [n]$ mit $\sum_{i \in I} a_i = T$? Zeigen Sie die NP-Schwere durch Reduktion eines bekannten NP-schweren Problems.

Beispiel $(a_1=1)$ -SubsetSum

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

Kern. Verdopple alle Größen und stelle das Ballast-Item $a_1 = 1$ voran:

$$a = (1, 4, 6, 10), \quad T = 2K = 10 \quad (a_{i+1} = 2c_i).$$

Aus $S = \{1, 2\}$ wird $I = \{2, 3\}$ mit $4 + 6 = 10 = T$; das einzige ungerade Item a_1 bleibt ungenutzt.

Lösung.

Konstruktion: Sei $(c_1, \dots, c_n, K)$ eine SUBSETSUM-Instanz. Erzeuge die Items $a_1 = 1$ und $a_{i+1} = 2c_i$ für $i \in [n]$, mit Zielwert $T = 2K$. ($a_1 = 1$ ist erfüllt, die Instanz ist gültig.)

Beweis SubsetSum nach $(a_1=1)$ -SubsetSum:

- Sei S eine Lösung mit $\sum_{i \in S} c_i = K$.
- Setze $I := \{i + 1 \mid i \in S\}$.
- Dann gilt

$$\sum_{j \in I} a_j = \sum_{i \in S} 2c_i = 2K = T.$$

- Also ist I eine Lösung der konstruierten Instanz.

Beweis $(a_1=1)$ -SubsetSum nach SubsetSum:

- Sei I eine Lösung mit $\sum_{i \in I} a_i = T = 2K$.
- Alle Items außer a_1 sind gerade.
- Die Zielsumme $T = 2K$ ist ebenfalls gerade.
- Angenommen, I enthielte das Item $a_1 = 1$.
- Dann wäre die Summe $\sum_{i \in I} a_i$ ungerade, denn a_1 ist das einzige ungerade Item.
- Das wäre ein Widerspruch zur geraden Zielsumme T .
- Somit gilt $1 \notin I$.
- Setze $S := \{i - 1 \mid i \in I\}$.
- Dann gilt $\sum_{i \in S} 2c_i = 2K$, also $\sum_{i \in S} c_i = K$.
- Also ist S eine Lösung der SUBSETSUM-Instanz.

Laufzeit: n Zahlen verdoppeln und das Ballast-Item $a_1 = 1$ voranstellen – $O(n)$, polynomiell.

Da SUBSETSUM NP-schwer ist, ist $(a_1=1)$ -SUBSETSUM NP-schwer. $\square$

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 $c_1 + c_2 = 21 + 1344 = 1365 = K$.

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 ATMOSTTWOPEPERSIZE-SUBSETSUM NP-schwer. □

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 $6 + 9 = 15 = K'$.

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. □

Sei L die Menge der SUBSETSUM-Instanzen, bei denen keine Itemgröße eine Zweierpotenz ist. Zeigen Sie, dass L NP-vollständig ist.

Beispiel SubsetSum ohne Zweierpotenzen

Instanz. SUBSETSUM-Instanz $c = (1, 4, 6)$, $K = 5$; die Größen $1 = 2^0$ und $4 = 2^2$ sind Zweierpotenzen. Lösung $S = \{1, 2\}$ mit $1 + 4 = 5$.

Kern. Multipliziere alle Größen und K mit 3: $c' = (3, 12, 18)$, $K' = 15$. Jede Größe hat den Primfaktor 3 und ist damit keine Zweierpotenz. Dieselbe Auswahl liefert $3 + 12 = 15 = K'$.

Lösung.

∈ **NP**: Zertifikat S ; summiere und vergleiche mit K . Laufzeit $O(n)$, polynomiell.

Konstruktion: Sei $(c_1, \dots, c_n, K)$ eine SUBSETSUM-Instanz. Setze $c'_i := 3c_i$ und $K' := 3K$. Jede Größe $c'_i \geq 3$ ist durch 3 teilbar und damit keine Zweierpotenz (Zweierpotenzen haben nur den Primfaktor 2) – die Instanz liegt in der eingeschränkten Menge.

Beweis hin: $\sum_{i \in S} c_i = K \Rightarrow \sum_{i \in S} c'_i = 3K = K'$.

Beweis zurück: $\sum_{i \in S} c'_i = K' \Rightarrow$ Division durch 3: $\sum_{i \in S} c_i = K$.

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

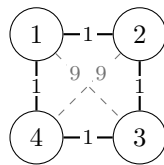
Da SUBSETSUM NP-vollständig ist und $L \in \text{NP}$, ist L NP-vollständig. □

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.

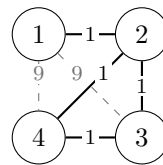
Zeigen Sie: Für kein $\alpha > 1$ gibt es einen Approximationsalgorithmus mit Güte α für TSP, außer $P = NP$.

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.

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 $P = NP$. $\square$

$\text{HALT}_{\text{TM}} := \{\langle M, w \rangle \mid M \text{ ist DTM und hält auf } w\}$. Zeigen Sie: HALT_{TM} ist NP-schwer.

Beispiel HALT_{TM} NP-schwer

Instanz. SAT-Formel $\varphi = (x_1 \vee x_2) \wedge (\neg x_1)$, erfüllbar durch $x_1 x_2 = 01$.

Kern. Die Reduktion gibt $\langle M, \langle \varphi \rangle \rangle$ aus: die feste DTM M probiert nacheinander alle Belegungen von φ , hält bei einer erfüllenden und geht sonst in eine Endlosschleife. Hier findet M die erfüllende Belegung 01 und hält – also $\langle M, \langle \varphi \rangle \rangle \in \text{HALT}_{\text{TM}}$.

Lösung.

Zeige, dass HALT_{TM} NP-schwer ist durch die Reduktion: $\text{SAT} \leq_p \text{HALT}_{\text{TM}}$. SAT ist bereits als NP-vollständig bekannt, also insbesondere NP-schwer.

Konstruktion: Sei φ eine SAT-Instanz mit den Variablen $x_1, \dots, x_n$. Wir konstruieren daraus die HALT_{TM} -Instanz $\langle M, w \rangle$:

- $w = \langle \varphi \rangle$, also die Kodierung der Formel φ .
- M ist eine feste DTM, die auf Eingabe $\langle \varphi \rangle$ wie folgt arbeitet:
 1. Lese φ und probiere nacheinander alle 2^n Belegungen β der Variablen $x_1, \dots, x_n$.
 2. Falls eine Belegung β die Formel φ erfüllt, halte.
 3. Falls alle 2^n Belegungen geprüft wurden und keine φ erfüllt, gehe in eine Endlosschleife.

Beweis SAT nach HALT_{TM} :

- Sei $\varphi \in \text{SAT}$.
- Dann existiert eine erfüllende Belegung β .
- M probiert alle Belegungen und findet β , also hält M auf w .
- Damit gilt $\langle M, w \rangle \in \text{HALT}_{\text{TM}}$.

Beweis HALT_{TM} nach SAT:

- Sei $\langle M, w \rangle \in \text{HALT}_{\text{TM}}$, also hält M auf w .
- M hält nur, wenn es eine erfüllende Belegung β findet.
- Denn ohne erfüllende Belegung geht M in die Endlosschleife.
- Also existiert eine erfüllende Belegung für φ .
- Damit gilt $\varphi \in \text{SAT}$.

Laufzeit: Es wird nur φ kodiert und die feste Maschine M ausgegeben – $O(|\varphi|)$, also bleibt die Transformation polynomiell. Ob M auf w hält, spielt für die Laufzeit der Reduktion keine Rolle. Da SAT NP-schwer ist und $\text{SAT} \leq_p \text{HALT}_{\text{TM}}$ gilt, ist HALT_{TM} NP-schwer. $\square$

4 ETH-Schranken

A34 Lower Bounds: VertexCover

Hausaufgabe 12.1, 5 P.

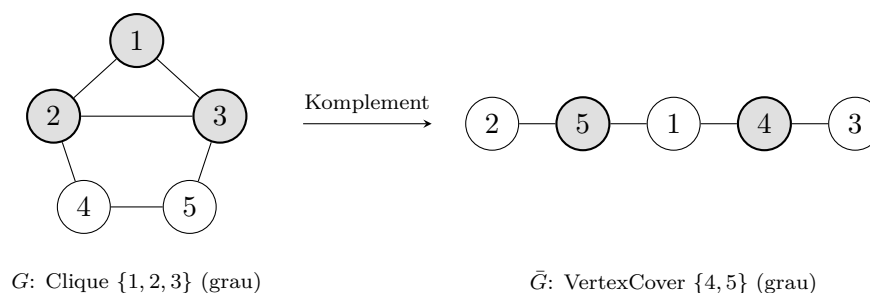
Betrachten Sie die Reduktion $\text{CLIQUE} \leq_p \text{VERTEXCOVER}$ (Komplementgraph, $k' = n - k$).

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

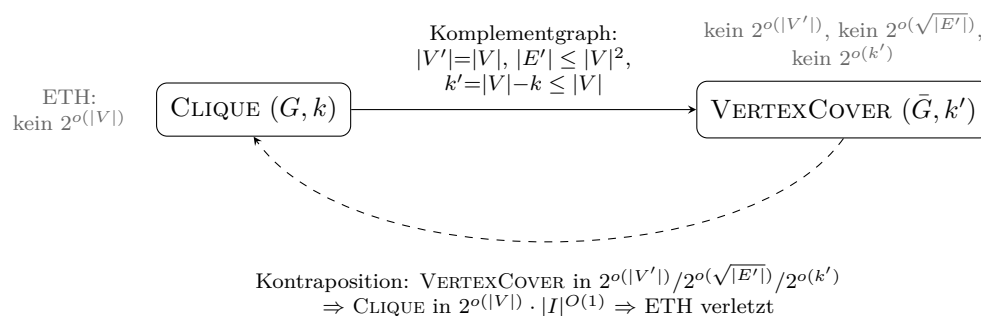
Instanz. CLIQUE -Instanz G mit $|V| = 5$, $|E| = 6$, $k = 3$. Die Reduktion bildet den Komplementgraphen $\bar{G}$ mit $k' = |V| - k$.



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 $2^{o(|V'|)}$.

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.

□

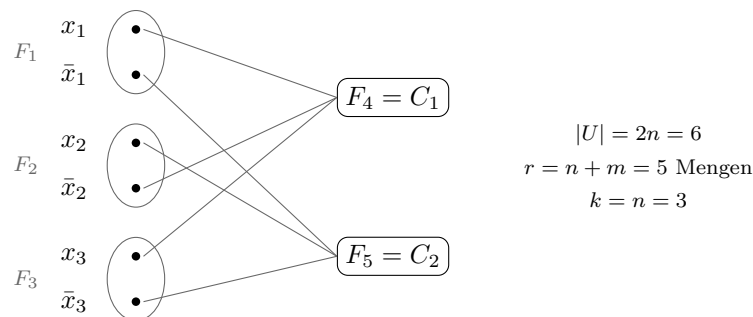
Reduktion $3\text{-SAT} \leq_p \text{HITTINGSET}$: $U = \{x_1, \bar{x}_1, \dots, x_n, \bar{x}_n\}$; $F_i = \{x_i, \bar{x}_i\}$ pro Variable; $F_{n+j} = C_j$ pro Klausel; $k = n$.

1. Geben Sie $|U|$, r und k in Abhängigkeit von n und m an.
2. Beweisen Sie Lower Bounds bzgl. $|U|$, r und k unter der ETH.

Hinweis: Unter der ETH ist 3-SAT nicht in $2^{o(n)} \cdot |I|^{O(1)}$ lösbar; mit dem Sparsification-Lemma auch nicht in $2^{o(m)} \cdot |I|^{O(1)}$. Für 3-SAT-Formeln gilt $n \leq 3m$.

Beispiel Lower Bounds HittingSet: konkrete Instanz

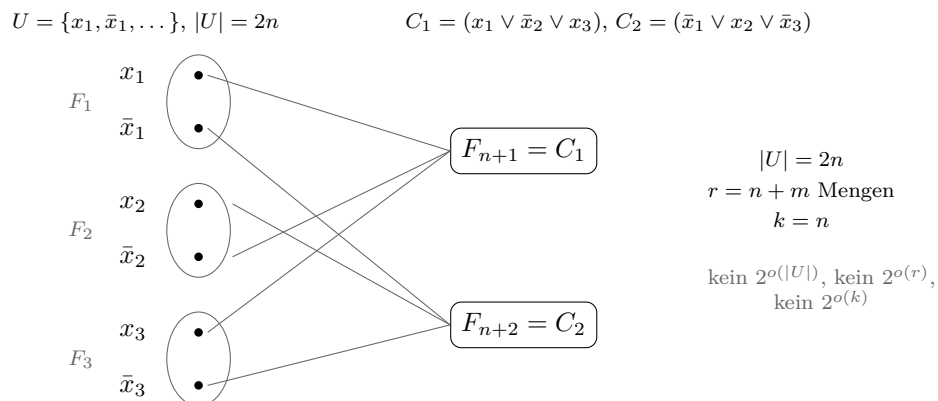
Instanz. 3-SAT-Formel mit $n = 3$, $m = 2$: $C_1 = (x_1 \vee \bar{x}_2 \vee x_3)$, $C_2 = (\bar{x}_1 \vee x_2 \vee \bar{x}_3)$. Die Reduktion liefert $F_i = \{x_i, \bar{x}_i\}$ pro Variable, $F_{n+j} = C_j$ pro Klausel und $k = n$.



Kern. Wegen $|U| = 2n = 6$ wäre ein $2^{o(|U|)}$ -Algorithmus für HITTINGSET zugleich ein $2^{o(n)}$ -Algorithmus für 3-SAT – direkter Widerspruch zur ETH. Kein $2^{o(|U|)}$.

Lösung.

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



- 2.

- **Bzgl. $|U|$:** Angenommen, HITTINGSET wäre in $2^{o(|U|)} \cdot |I|^{O(1)}$ lösbar. Wegen $|U| = 2n$ ergäbe Reduktion + Algorithmus einen $2^{o(n)}$ -Algorithmus für 3-SAT – direkter Widerspruch zur ETH.

- **Bzgl. r :** Angenommen, HITTINGSET wäre in $2^{o(r)} \cdot |I|^{O(1)}$ lösbar. Wegen $r = n + m$ und $n \leq 3m$ ist $r = O(m)$ – es ergäbe sich ein $2^{o(m)}$ -Algorithmus für 3-SAT. Nach dem **Sparsification-Lemma** geht das nur, wenn die ETH falsch ist.
- **Bzgl. k :** Angenommen, HITTINGSET wäre in $2^{o(k)} \cdot |I|^{O(1)}$ lösbar. Wegen $k = n$ ergäbe sich ein $2^{o(n)}$ -Algorithmus für 3-SAT – direkter Widerspruch zur ETH.

□

Welche Lower Bounds ergeben sich unter der ETH (mit Sparsification-Lemma) aus den Reduktionen (1) $3\text{-SAT} \leq_p k\text{-CLIQUE}$, (2) $k\text{-CLIQUE} \leq_p k\text{-IS}$ (Komplementgraph), (3) $\text{SAT} \leq_p 3\text{-DM}$, (4) $3\text{-DM} \leq_p 3\text{-EC}$, (5) $3\text{-EC} \leq_p \text{SUBSETSUM}$? Schätzen Sie zuerst die Parameter der jeweils erzeugten Instanz ab.

Beispiel Lower Bounds entlang der Kette: $m = 4$ durchpropagiert

Instanz. Eine 3-SAT-Formel mit $m = 4$ Klauseln; die O -Schranken der Reduktionen setzen wir zur Illustration als konkrete Werte an ($|V| = 3m$ usw.):

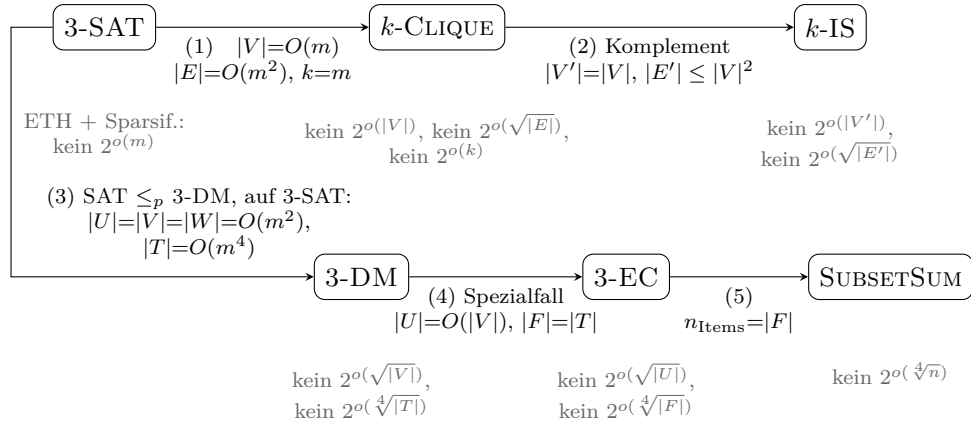
Schritt	erzeugte Instanz	Größen bei $m = 4$
(1) $3\text{-SAT} \leq_p k\text{-CLIQUE}$	$ V = O(m)$, $ E = O(m^2)$, $k = m$	$ V = 12$, $ E = 16$, $k = 4$
(2) $k\text{-CLIQUE} \leq_p k\text{-IS}$	$ V' = V $, $k' = k$ (Komplementgraph)	$ V' = 12$, $k' = 4$
(3) $\text{SAT} \leq_p 3\text{-DM}$	$ U = O(m^2)$, $ T = O(m^4)$	$ U = 16$, $ T = 256$
(4) $3\text{-DM} \leq_p 3\text{-EC}$	$ U = O(V)$, $ F = T $	$ U = 16$, $ F = 256$
(5) $3\text{-EC} \leq_p \text{SUBSETSUM}$	$n_{\text{Items}} = F $	$n = 256$

Kern. Ein $2^{o(\sqrt[4]{n})}$ -Algorithmus für SUBSETSUM hätte bei $n = 256$ den Exponenten $\sqrt[4]{256} = 4 = m$ – rückwärts durch die Kette ergäbe er einen $2^{o(m)}$ -Algorithmus für 3-SAT, den es unter der ETH (Sparsification-Lemma) nicht gibt. Kein $2^{o(\sqrt[4]{n})}$.

Lösung.

Unter der ETH ist 3-SAT nicht in $2^{o(m)} \cdot |I|^{O(1)}$ lösbar (Sparsification-Lemma).

- (1) Erzeugt $|V| = O(m)$, $|E| = O(m^2)$, $k = m$. Ein $2^{o(|V|)}$ -, $2^{o(\sqrt{|E|})}$ - oder $2^{o(k)}$ -Algorithmus für CLIQUE ergäbe jeweils $2^{o(m)}$ für 3-SAT – Widerspruch. Bounds: kein $2^{o(|V|)}$, kein $2^{o(\sqrt{|E|})}$, kein $2^{o(k)}$.
- (2) Erzeugt $|V'| = |V|$, $|E'| \leq |V|^2$. Ein zu schneller IS-Algorithmus ergäbe via Komplementbildung denselben für CLIQUE. Bounds: kein $2^{o(|V'|)}$, kein $2^{o(\sqrt{|E'|})}$.
- (3) Erzeugt $|U| = |V| = |W| = O(mn) \subseteq O(m^2)$ und $|T| = O(m^2n^2) \subseteq O(m^4)$ (angewandt auf 3-SAT-Instanzen). Ein $2^{o(\sqrt{|V|})}$ -Algorithmus ergäbe $2^{o(m)}$ für 3-SAT; ebenso ein $2^{o(\sqrt[4]{|T|})}$ -Algorithmus. Bounds: kein $2^{o(\sqrt{|V|})}$, kein $2^{o(\sqrt[4]{|T|})}$.
- (4) 3-DM ist Spezialfall von 3-EC mit $|U| = O(|V|)$ und $|F| = |T|$. Die Bounds übertragen sich: kein $2^{o(\sqrt{|U|})}$, kein $2^{o(\sqrt[4]{|F|})}$.
- (5) Erzeugt $n_{\text{Items}} = |F|$. Ein $2^{o(\sqrt[4]{n})}$ -Algorithmus für SUBSETSUM ergäbe $2^{o(\sqrt[4]{|F|})}$ für 3-EC – nach (4) verboten. Bound: kein $2^{o(\sqrt[4]{n})}$. $\square$

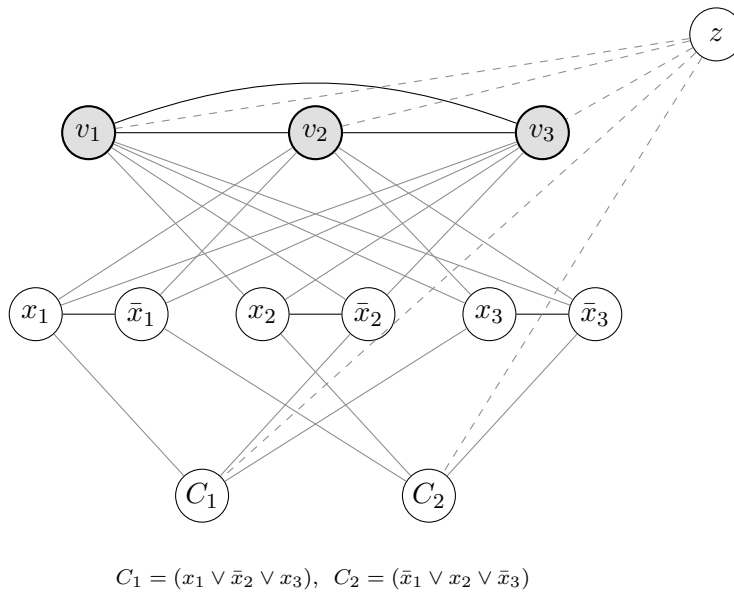


Die Reduktion $3\text{-SAT} \leq_p k\text{-COLOR}$ erzeugt $V = \{x_i, \bar{x}_i, v_i \mid i \in [n]\} \cup \{C_j \mid j \in [m]\} \cup \{z\}$ mit den Kanten des Vorlesungs-Gadgets. Welche Lower Bounds ergeben sich unter der ETH bzgl. (i) $|V|$ und (ii) $|E|$?

Hinweis: Mit dem Sparsification-Lemma ist 3-SAT unter der ETH nicht in $2^{o(m)} \cdot |I|^{O(1)}$ lösbar; für 3-SAT-Formeln gilt $n \leq 3m$.

Beispiel Lower Bounds k -Color: konkrete Zahlen

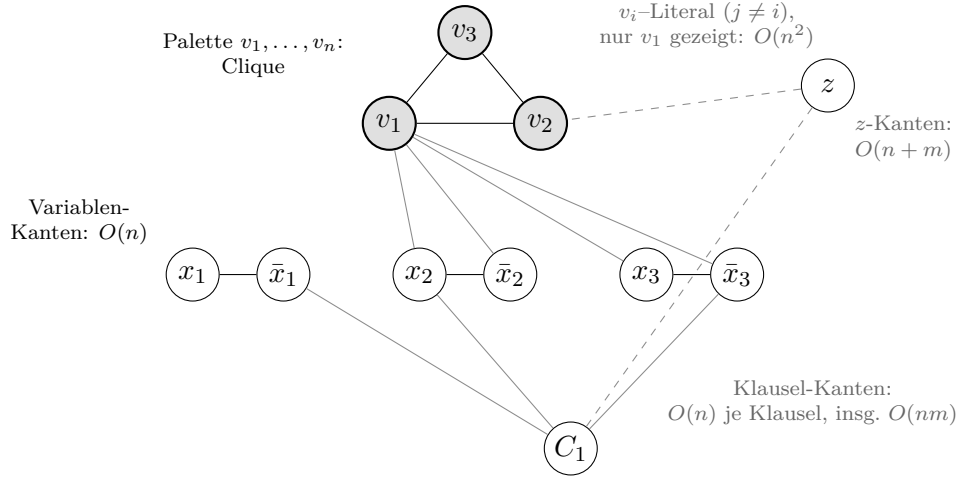
Instanz. 3-SAT-Formel mit $n = 3$, $m = 2$. Die Reduktion erzeugt die Knotenmenge $V = \{x_i, \bar{x}_i, v_i \mid i \in [3]\} \cup \{C_1, C_2\} \cup \{z\}$:



Kern. Hier ist $|V| = 3n + m + 1 = 12$; wegen $n \leq 3m$ gilt allgemein $|V| = O(m)$. Ein $2^{o(|V|)}$ -Algorithmus für k -COLOR wäre damit ein $2^{o(m)}$ -Algorithmus für 3-SAT – nach dem Sparsification-Lemma Widerspruch zur ETH. Kein $2^{o(|V|)}$.

Lösung.

Parameter: $|V| = 3n + m + 1$; wegen $n \leq 3m$ gilt $|V| = O(m)$. Für die Kanten: v_i -Clique und v_i -Literal-Kanten $O(n^2)$, Variablen-Kanten $O(n)$, Klausel-Kanten $O(nm)$, z -Kanten $O(n + m)$ – insgesamt $|E| = O(n^2 + nm + m) = O(m^2)$.



$$|V| = 3n + m + 1 = O(m), \quad |E| = O(n^2 + nm + m) = O(m^2) \Rightarrow \text{kein } 2^{o(|V|)}, \text{ kein } 2^{o(\sqrt{|E|})}$$

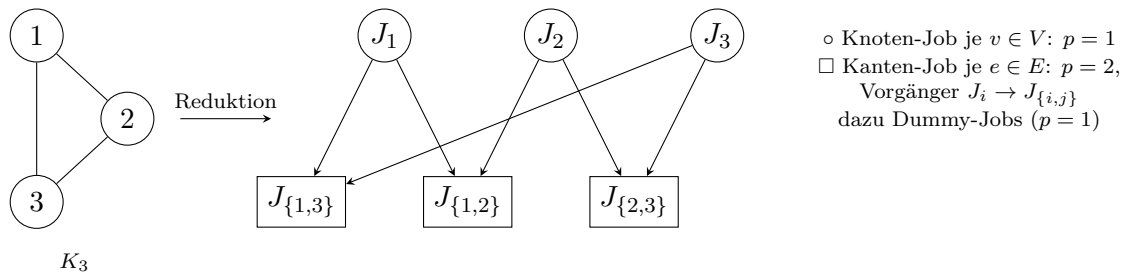
(i) Angenommen, k -COLOR wäre in $2^{o(|V|)} \cdot |I|^{O(1)}$ lösbar. Wegen $|V| = O(m)$ ergäbe sich ein $2^{o(m)}$ -Algorithmus für 3-SAT – nach dem Sparsification-Lemma nur möglich, wenn die ETH falsch ist. Bound: kein $2^{o(|V|)}$.

(ii) Angenommen, k -COLOR wäre in $2^{o(\sqrt{|E|})} \cdot |I|^{O(1)}$ lösbar. Wegen $|E| = O(m^2)$ ist $\sqrt{|E|} = O(m)$ – es ergäbe sich ein $2^{o(m)}$ -Algorithmus für 3-SAT. Widerspruch (Sparsification-Lemma). Bound: kein $2^{o(\sqrt{|E|})}$. $\square$

Reduktion von k -CLIQUE: $n = 4|V| + 3|E|$ Jobs (Knoten-Jobs $p = 1$, Kanten-Jobs $p = 2$, Dummy-Jobs $p = 1$), Präzedenzen $(J_i, J_{\{i,j\}})$, Makespan $T = 2|V| + 2|E|$. Nutzen Sie die bekannten Clique-Bounds (zusammenhängender Graph): kein $2^{o(|V|)}$, kein $2^{o(\sqrt{|E|})}$. Welche Lower Bounds ergeben sich bzgl. (i) n und (ii) T ?

Beispiel Lower Bounds Scheduling: Dreieck einsetzen

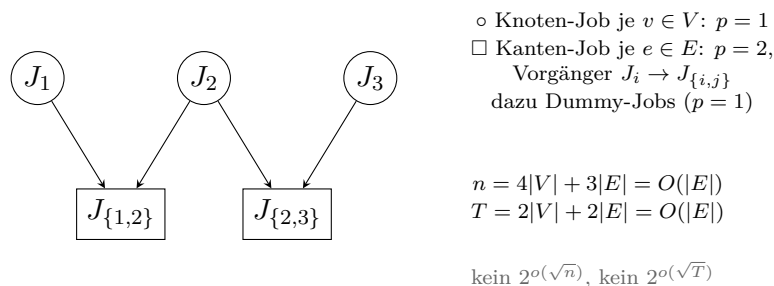
Instanz. CLIQUE-Instanz K_3 (zusammenhängend): $|V| = 3$, $|E| = 3$. Die Reduktion erzeugt $n = 4|V| + 3|E| = 21$ Jobs und den Makespan $T = 2|V| + 2|E| = 12$.



Kern. Wegen $|V| \leq |E| + 1$ ist $n = O(|E|)$, also $\sqrt{n} = O(\sqrt{|E|})$. Ein $2^{o(\sqrt{n})}$ -Algorithmus für das Scheduling-Problem wäre damit ein $2^{o(\sqrt{|E|})}$ -Algorithmus für CLIQUE – Widerspruch zur ETH. Kein $2^{o(\sqrt{n})}$.

Lösung.

Parameter: Da der Clique-Graph zusammenhängend ist, gilt $|V| \leq |E| + 1$, also $n = 4|V| + 3|E| = O(|E|)$ und ebenso $T = 2|V| + 2|E| = O(|E|)$.



(i) Angenommen, das Scheduling-Problem wäre in $2^{o(\sqrt{n})} \cdot |I|^{O(1)}$ lösbar. Wegen $n = O(|E|)$ ist $\sqrt{n} = O(\sqrt{|E|})$ – Reduktion + Algorithmus ergäben einen $2^{o(\sqrt{|E|})}$ -Algorithmus für CLIQUE, den es unter der ETH nicht gibt. Bound: kein $2^{o(\sqrt{n})}$.

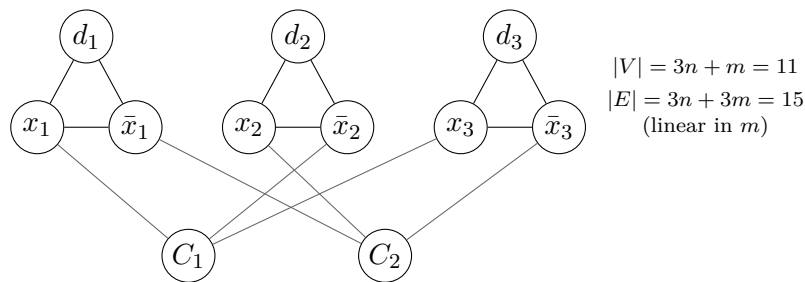
(ii) Analog mit $T = O(|E|)$: Ein $2^{o(\sqrt{T})}$ -Algorithmus ergäbe $2^{o(\sqrt{|E|})}$ für CLIQUE. Bound: kein $2^{o(\sqrt{T})}$. □

Reduktion $3\text{-SAT} \leq_p \text{DOMINATINGSET}$: pro Variable die Knoten $x_i, \bar{x}_i, d_i$ mit Dreieckskanten; pro Klausel ein Knoten C_j mit Kanten zu seinen Literalen. Welche Lower Bounds ergeben sich unter der ETH bzgl. (i) $|V|$ und (ii) $|E|$?

Hinweis: Mit dem Sparsification-Lemma ist 3-SAT unter der ETH nicht in $2^{o(m)} \cdot |I|^{O(1)}$ lösbar; für 3-SAT-Formeln gilt $n \leq 3m$.

Beispiel Lower Bounds DominatingSet: konkrete Instanz

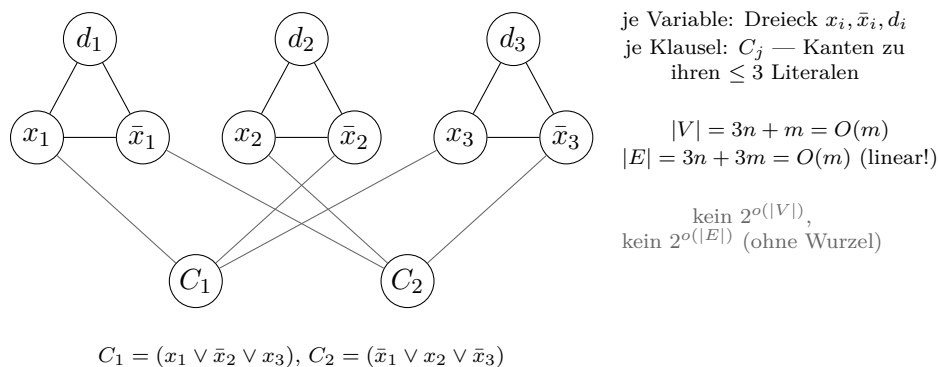
Instanz. 3-SAT-Formel mit $n = 3, m = 2$. Pro Variable ein Dreieck $x_i, \bar{x}_i, d_i$; pro Klausel ein Knoten C_j mit Kanten zu seinen Literalen.



Kern. Die Kantenanzahl $|E| = 3n + 3m = 15$ wächst wegen $n \leq 3m$ nur *linear* in m . Ein $2^{o(|E|)}$ -Algorithmus (ohne Wurzel) für DOMINATINGSET wäre damit ein $2^{o(m)}$ -Algorithmus für 3-SAT – nach dem Sparsification-Lemma Widerspruch zur ETH. Kein $2^{o(|E|)}$.

Lösung.

Parameter: $|V| = 3n + m$; wegen $n \leq 3m$ gilt $|V| = O(m)$. Kanten: $3n$ Dreieckskanten plus höchstens $3m$ Klausel-Literal-Kanten (je Klausel ≤ 3 Literale), also $|E| = 3n + 3m = O(m)$ – *linear* in m .



(i) Angenommen, DOMINATINGSET wäre in $2^{o(|V|)} \cdot |I|^{O(1)}$ lösbar. Wegen $|V| = O(m)$ ergäbe sich ein $2^{o(m)}$ -Algorithmus für 3-SAT – nach dem Sparsification-Lemma Widerspruch zur ETH. Bound: kein $2^{o(|V|)}$.

(ii) Angenommen, DOMINATINGSET wäre in $2^{o(|E|)} \cdot |I|^{O(1)}$ lösbar. Wegen $|E| = O(m)$ ergäbe sich ebenfalls ein $2^{o(m)}$ -Algorithmus für 3-SAT – Widerspruch (Sparsification-Lemma). Bound: kein $2^{o(|E|)}$ (hier ohne Wurzel, da die Kantenanzahl nur linear wächst). $\square$

Reduktion von CLIQUE (mit $c \cdot n \leq k \leq n$ erlaubt): $S_{i,i} = \{(a, a) \mid a \in V\}$, $S_{i,j} = \{(a, b) \mid a \neq b, \{a, b\} \in E\}$ für $i \neq j$; k übernommen. Welche Lower Bounds ergeben sich bzgl. (i) $X = \sum_{(i,j)} |S_{i,j}|$ und (ii) $Y = \max_{(i,j)} |S_{i,j}|$?

Beispiel Lower Bounds GridTiling: 3×3 -Instanz

Instanz. CLIQUE-Instanz mit $N = |V| = 4$, $|E| = 5$, $k = 3$. Die Reduktion baut ein $k \times k = 3 \times 3$ -Gitter aus Mengen $S_{i,j}$:

	$j = 1$	$j = 2$	$j = 3$	
$i = 1$	4	10	10	
$i = 2$	10	4	10	
$i = 3$	10	10	4	

grau (Diagonale): $|S_{i,i}| = N = 4$
 weiß: $|S_{i,j}| = 2|E| = 10$
 3×3 -Gitter, $k = 3 = \Theta(N)$

Kern. Hier ist $X = \sum_{(i,j)} |S_{i,j}| = 3 \cdot 4 + 6 \cdot 10 = 72$; allgemein $X = O(N^4)$, also $\sqrt[4]{X} = O(N)$. Ein $2^{o(\sqrt[4]{X})}$ -Algorithmus für GRIDTILING wäre damit ein $2^{o(N)}$ -Algorithmus für CLIQUE – Widerspruch zur ETH. Kein $2^{o(\sqrt[4]{X})}$.

Lösung.

Parameter (mit $N := |V|$, $k = \Theta(N)$): Diagonale Mengen haben N Tupel, Nebendiagonal-Mengen $2|E| \leq N^2$ Tupel. Also $X \leq k \cdot N + k^2 \cdot N^2 = O(N^4)$ und $Y \leq \max(N, N^2) = O(N^2)$.

$k \times k$ Zellen (hier $k = 3$), k von CLIQUE übernommen, $k = \Theta(N)$, $N = |V|$

	$j = 1$	$j = 2$	$j = 3$
$i = 1$	$\{(a, a) \mid a \in V\}$	$\{(a, b) \mid \{a, b\} \in E\}$	$\{(a, b) \mid \{a, b\} \in E\}$
$i = 2$	$\{(a, b) \mid \{a, b\} \in E\}$	$\{(a, a) \mid a \in V\}$	$\{(a, b) \mid \{a, b\} \in E\}$
$i = 3$	$\{(a, b) \mid \{a, b\} \in E\}$	$\{(a, b) \mid \{a, b\} \in E\}$	$\{(a, a) \mid a \in V\}$

$$|S_{i,i}| = N, \quad |S_{i,j}| = 2|E| \leq N^2 \Rightarrow X = \sum_{(i,j)} |S_{i,j}| = O(N^4), \quad Y = \max_{(i,j)} |S_{i,j}| = O(N^2)$$

kein $2^{o(\sqrt[4]{X})}$, kein $2^{o(\sqrt{Y})}$

(i) Angenommen, GRIDTILING wäre in $2^{o(\sqrt[4]{X})} \cdot |I|^{O(1)}$ lösbar. Wegen $X = O(N^4)$ ist $\sqrt[4]{X} = O(N)$ – es ergäbe sich ein $2^{o(N)}$ -Algorithmus für CLIQUE, den es unter der ETH nicht gibt. Bound: kein $2^{o(\sqrt[4]{X})}$.

(ii) Angenommen, GRIDTILING wäre in $2^{o(\sqrt{Y})} \cdot |I|^{O(1)}$ lösbar. Wegen $Y = O(N^2)$ ist $\sqrt{Y} = O(N)$ – wieder ein $2^{o(N)}$ -Algorithmus für CLIQUE. Widerspruch. Bound: kein $2^{o(\sqrt{Y})}$. $\square$

Reduktion $3\text{-SAT} \leq_p \text{SUBSETSUM}$ mit Dezimalziffern: pro Variable zwei Items a_i, b_i ($s = 10^{i-1} + \sum 10^{n+j-1}$ über die Klauseln mit x_i bzw. $\bar{x}_i$), pro Klausel zwei Items c_j, d_j mit $s = 10^{n+j-1}$; Ziel $B = \sum_j 3 \cdot 10^{n+j-1} + \sum_i 10^{i-1}$. Welche Lower Bounds ergeben sich bzgl. (i) der Anzahl unterschiedlicher Itemgrößen d und (ii) der Kodierungslänge $L = \log \Delta$ der größten Itemgröße?

Hinweis: Mit dem Sparsification-Lemma ist 3-SAT unter der ETH nicht in $2^{o(m)} \cdot |I|^{O(1)}$ lösbar; für 3-SAT-Formeln gilt $n \leq 3m$.

Beispiel Lower Bounds SubsetSum: Parameter der $n=3, m=2$ -Instanz

Instanz. 3-SAT-Formel mit $n = 3, m = 2$: $\varphi = (x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3)$. Die Reduktion erzeugt $2n + 2m = 10$ Items.

Kern. Die Klausel-Items c_j, d_j teilen sich je eine Größe, also gibt es nur $d \leq 2n + m = 8$ verschiedene Itemgrößen. Wegen $n \leq 3m$ ist $d = O(m)$. Ein $2^{o(d)}$ -Algorithmus für SUBSETSUM wäre damit ein $2^{o(m)}$ -Algorithmus für 3-SAT – nach dem Sparsification-Lemma Widerspruch zur ETH. Kein $2^{o(d)}$.

Lösung.

Parameter: Es gibt $2n + 2m$ Items; die Klausel-Items c_j, d_j teilen sich je eine Größe, also $d \leq 2n + m$. Wegen $n \leq 3m$ gilt $d = O(m)$. Die größte Größe ist $\Delta < 10^{n+m+1}$, also $L = \log \Delta = O(n + m) = O(m)$.

$$\text{Beispiel } n = 3, m = 2: \quad \varphi = \underbrace{(x_1 \vee x_2 \vee x_3)}_{C_1} \wedge \underbrace{(\bar{x}_1 \vee \bar{x}_2 \vee x_3)}_{C_2}$$

Item	10^4 (C_2)	10^3 (C_1)	10^2 (x_3)	10^1 (x_2)	10^0 (x_1)	
a_1 (x_1)	0	1	0	0	1	$10^0 + 10^3$
b_1 ($\bar{x}_1$)	1	0	0	0	1	$10^0 + 10^4$
a_2 (x_2)	0	1	0	1	0	$10^1 + 10^3$
b_2 ($\bar{x}_2$)	1	0	0	1	0	$10^1 + 10^4$
a_3 (x_3)	1	1	1	0	0	$10^2 + 10^3 + 10^4$
b_3 ($\bar{x}_3$)	0	0	1	0	0	10^2
$c_1 = d_1$	0	1	0	0	0	10^3 (je zweimal)
$c_2 = d_2$	1	0	0	0	0	10^4 (je zweimal)
B	3	3	1	1	1	Ziel

$2n + 2m$ Items, aber $d \leq 2n + m = O(m)$ Größen; $\Delta < 10^{n+m+1}$, also $L = \log \Delta = O(m) \Rightarrow$ kein $2^{o(d)}$, kein $2^{o(L)}$

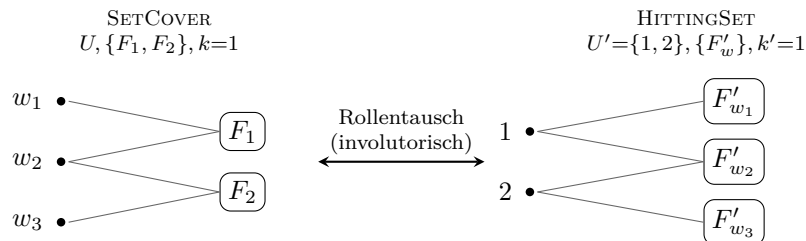
(i) Angenommen, SUBSETSUM wäre in $2^{o(d)} \cdot |I|^{O(1)}$ lösbar. Wegen $d = O(m)$ ergäbe Reduktion + Algorithmus einen $2^{o(m)}$ -Algorithmus für 3-SAT – nach dem Sparsification-Lemma Widerspruch zur ETH. Bound: kein $2^{o(d)}$.

(ii) Angenommen, SUBSETSUM wäre in $2^{o(L)} \cdot |I|^{O(1)}$ lösbar. Wegen $L = O(m)$ ergäbe sich ebenso ein $2^{o(m)}$ -Algorithmus für 3-SAT – Widerspruch (Sparsification-Lemma). Bound: kein $2^{o(L)}$. $\square$

Gegeben: HITTINGSET ist unter der ETH nicht in $2^{o(r')} \langle I \rangle^{O(1)}$ und nicht in $2^{o(|U'|)} \langle I \rangle^{O(1)}$ lösbar. Die Reduktion $\text{SETCOVER} \leq_p \text{HITTINGSET}$ vertauscht die Rollen: $U' = \{1, \dots, r\}$ und pro $w \in U$ die Menge $F'_w = \{v \mid w \in F_v\}$, $k' = k$. Welche Lower Bounds ergeben sich für SETCOVER bzgl. (i) $|U|$ und (ii) r ?

Beispiel Lower Bounds SetCover: Rollentausch konkret

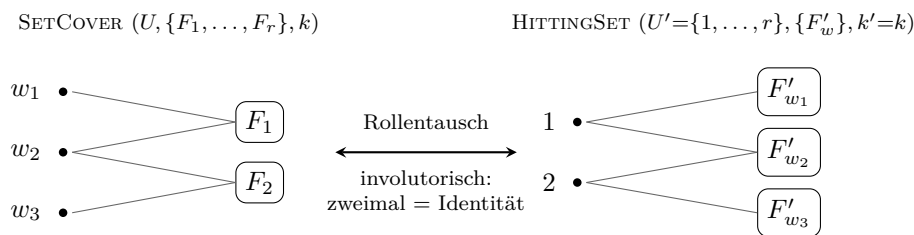
Instanz. SETCOVER-Instanz mit $|U| = 3$ Elementen, $r = 2$ Mengen und $k = 1$. Der involutorische Rollentausch $F'_w = \{v \mid w \in F_v\}$, $U' = \{1, \dots, r\}$ liefert die HITTINGSET-Instanz:



Kern. Es entsteht eine HITTINGSET-Instanz mit $r' = |U| = 3$ Mengen und $|U'| = r = 2$ Elementen. Ein $2^{o(|U|)}$ -Algorithmus für SETCOVER wäre via Rollentausch ein $2^{o(r')}$ -Algorithmus für HITTINGSET – den es unter der ETH nicht gibt. Kein $2^{o(|U|)}$.

Lösung.

Die Abbildung ist involutorisch (Elemente und Mengen tauschen die Plätze; zweimal angewandt entsteht die Ausgangsinstanz). Sie ist damit zugleich eine Reduktion $\text{HITTINGSET} \leq_p \text{SETCOVER}$, bei der aus einer HittingSet-Instanz mit Universum U' und r' Mengen eine SetCover-Instanz mit $|U| = r'$ und $r = |U'|$ entsteht.



$$F'_w = \{v \mid w \in F_v\}: \text{ aus } (U', r') \text{ wird } |U| = r', r = |U'| \Rightarrow \text{kein } 2^{o(|U|)}, \text{kein } 2^{o(r)}$$

(i) Angenommen, SETCOVER wäre in $2^{o(|U|)} \langle I \rangle^{O(1)}$ lösbar. Über die Reduktion entstünde ein Algorithmus für HITTINGSET in $2^{o(r')} \langle I \rangle^{O(1)}$ – den es unter der ETH nicht gibt. Bound: kein $2^{o(|U|)}$.

(ii) Angenommen, SETCOVER wäre in $2^{o(r)} \langle I \rangle^{O(1)}$ lösbar. Über die Reduktion entstünde ein $2^{o(|U'|)}$ -Algorithmus für HITTINGSET – auch den gibt es unter der ETH nicht. Bound: kein $2^{o(r)}$.

□

Reduktion 3-SAT $\leq_p$ ILP-FEASIBILITY: $N = 2n$ ILP-Variablen (eine pro Literal); pro Klausel C_i die Ungleichung $-\sum_{\ell \in C_i} x_\ell \leq -1$; pro Variable v die Ungleichungen $x_v + x_{\bar{v}} \leq 1$ und $-x_v - x_{\bar{v}} \leq -1$. Welche Lower Bounds ergeben sich bzgl. (i) M (Zeilenzahl) und (ii) N (Spaltenzahl)?

Hinweis: Unter der ETH ist 3-SAT nicht in $2^{o(n)} \cdot |I|^{O(1)}$ lösbar; mit dem Sparsification-Lemma auch nicht in $2^{o(m)} \cdot |I|^{O(1)}$. Für 3-SAT-Formeln gilt $n \leq 3m$.

Beispiel Lower Bounds ILP-Feasibility: Ungleichungen einsetzen

Instanz. 3-SAT-Formel mit $n = 3$, $m = 2$: $C_1 = (x_1 \vee x_2 \vee x_3)$, $C_2 = (\bar{x}_1 \vee x_2 \vee \bar{x}_3)$. Pro Literal eine 0/1-Variable, also $N = 2n = 6$ Spalten; die Reduktion erzeugt die Ungleichungen

$$\begin{aligned} \text{je Klausel:} \quad & - (x_{x_1} + x_{x_2} + x_{x_3}) \leq -1, \quad - (x_{\bar{x}_1} + x_{x_2} + x_{\bar{x}_3}) \leq -1; \\ \text{je Variable } v: \quad & x_{x_v} + x_{\bar{x}_v} \leq 1, \quad -x_{x_v} - x_{\bar{x}_v} \leq -1 \quad (v = 1, 2, 3). \end{aligned}$$

Kern. Das sind $M = m + 2n = 8$ Zeilen; wegen $n \leq 3m$ ist $M = O(m)$. Ein $2^{o(M)}$ -Algorithmus für ILP-FEASIBILITY wäre damit ein $2^{o(m)}$ -Algorithmus für 3-SAT – nach dem Sparsification-Lemma Widerspruch zur ETH. Kein $2^{o(M)}$.

Lösung.

Parameter: $M = m + 2n$; wegen $n \leq 3m$ gilt $M = O(m)$. Und $N = 2n$.

$$N = 2n \text{ Spalten: eine 0/1-Variable je Literal}$$

$$M = m + 2n \text{ Zeilen} \left[\begin{array}{c} \boxed{\begin{array}{c} m \text{ Klausel-Zeilen:} \\ -\sum_{\ell \in C_j} x_\ell \leq -1 \\ \hline 2n \text{ Variablen-Zeilen:} \\ x_v + x_{\bar{v}} \leq 1 \text{ und } -x_v - x_{\bar{v}} \leq -1 \end{array}} \end{array} \right]$$

(i) Angenommen, ILP-FEASIBILITY wäre in $2^{o(M)} \langle I \rangle^{O(1)}$ lösbar. Wegen $M = O(m)$ ergäbe Reduktion + Algorithmus einen $2^{o(m)}$ -Algorithmus für 3-SAT – nach dem Sparsification-Lemma Widerspruch zur ETH. Bound: kein $2^{o(M)}$.

(ii) Angenommen, ILP-FEASIBILITY wäre in $2^{o(N)} \langle I \rangle^{O(1)}$ lösbar. Wegen $N = 2n$ ergäbe sich ein $2^{o(n)}$ -Algorithmus für 3-SAT – direkter Widerspruch zur ETH. Bound: kein $2^{o(N)}$. $\square$

NAE-4-SAT. Gegeben: n Variablen $x_1, \dots, x_n$ und m Klauseln $C_1, \dots, C_m$ mit $|C_i| \leq 4$.
Entscheide: Gibt es eine Belegung, unter der jede Klausel zwei *verschieden* belegte Literale enthält?

Reduktion von 3-SAT: Die Variablen sind $x_1, \dots, x_n$ und eine frische Variable w . Jede Klausel C_i wird zu $C_i \cup \{w\}$.

- Geben Sie die Anzahl der Variablen und die Anzahl der Klauseln der NAE-4-SAT-Instanz in Abhängigkeit von der 3-SAT-Instanz an.
- Beweisen Sie Lower Bounds für NAE-4-SAT bezüglich der Anzahl der Variablen und der Anzahl der Klauseln basierend auf der ETH und der oben beschriebenen Reduktion.

Beispiel Lower Bounds NAE-4-SAT: Parameter einsetzen

Instanz. 3-SAT-Formel mit $n = 3$ Variablen und $m = 2$ Klauseln. Die Reduktion hängt die frische Variable w an jede Klausel an: $n' = n + 1 = 4$, $m' = m = 2$.

Kern. Ein $2^{o(n')}$ -Algorithmus für NAE-4-SAT wäre wegen $n' = n + 1$ ein $2^{o(n)}$ -Algorithmus für 3-SAT – direkter Widerspruch zur ETH. Kein $2^{o(n')}$.

Lösung.

a) Variablen $n' = n + 1$, Klauseln $m' = m$.

b) Zu zeigen ist, dass NAE-4-SAT unter der ETH weder in $2^{o(n')} \cdot |I|^{O(1)}$ noch in $2^{o(m')} \cdot |I|^{O(1)}$ lösbar ist.

Bzgl. n' :

- Angenommen, NAE-4-SAT wäre in $2^{o(n')} \cdot |I|^{O(1)}$ lösbar.
- Wegen $n' = n + 1$ ergäben Reduktion und Algorithmus einen $2^{o(n)}$ -Algorithmus für 3-SAT.
- Das ist ein direkter Widerspruch zur ETH.

Bzgl. m' :

- Angenommen, NAE-4-SAT wäre in $2^{o(m')} \cdot |I|^{O(1)}$ lösbar.
- Wegen $m' = m$ ergäbe sich ein $2^{o(m)}$ -Algorithmus für 3-SAT.
- Nach dem Sparsification-Lemma ist das ein Widerspruch zur ETH.

Bounds: kein $2^{o(n')}$, kein $2^{o(m')}$. □

SetSplitting. Gegeben: Eine Grundmenge S und Teilmengen $F_1, \dots, F_r \subseteq S$. **Entscheide:** Gibt es eine Partition $S = S_1 \dot{\cup} S_2$, sodass jede Teilmenge F_i sowohl S_1 als auch S_2 schneidet?

Reduktion von NAE-4-SAT (n Variablen, m Klauseln mit je höchstens 4 Literalen): Grundmenge $S = \{x_i, \bar{x}_i \mid i \in [n]\}$; als Teilmengen pro Variable das Paar $\{x_i, \bar{x}_i\}$ und pro Klausel die Menge C_i .

- Geben Sie die Größe der Grundmenge S und die Anzahl der Teilmengen r der SetSplitting-Instanz in Abhängigkeit von der NAE-4-SAT-Instanz an.
- Beweisen Sie Lower Bounds für SETSPLITTING bezüglich der Größe der Grundmenge und der Anzahl der Teilmengen basierend auf der ETH und der oben beschriebenen Reduktion.

Hinweis: NAE-4-SAT ist unter der ETH weder in $2^{o(n)} \cdot |I|^{O(1)}$ noch in $2^{o(m)} \cdot |I|^{O(1)}$ lösbar.

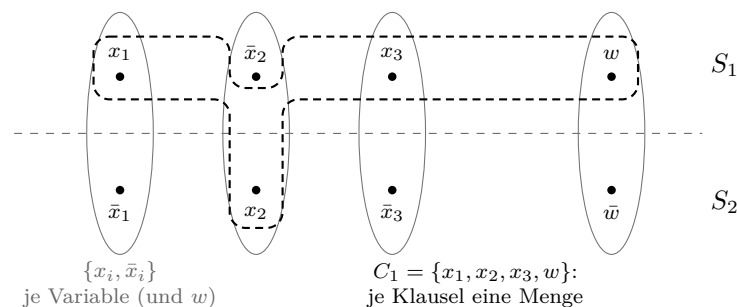
Beispiel Lower Bounds SetSplitting: Parameter einsetzen

Instanz. NAE-4-SAT-Instanz mit $n = 4$ Variablen (x_1, x_2, x_3, w) und $m = 2$ Klauseln: $|S| = 2n = 8$, $r = n + m = 6$.

Kern. Wegen $|S| = 2n$ wäre ein $2^{o(|S|)}$ -Algorithmus für SETSPLITTING ein $2^{o(n)}$ -Algorithmus für NAE-4-SAT – unter der ETH ausgeschlossen. Kein $2^{o(|S|)}$.

Lösung.

- Mit n Variablen und m Klauseln der NAE-Instanz: $|S| = 2n$, $r = n + m$.



$$|S| = 2n, \quad r = n + m \text{ Mengen; jede Menge schneidet } S_1 \text{ und } S_2 \Rightarrow \text{kein } 2^{o(|S|)}, \text{ kein } 2^{o(r)}$$

- Zu zeigen ist, dass SETSPLITTING unter der ETH weder in $2^{o(|S|)} \cdot |I|^{O(1)}$ noch in $2^{o(r)} \cdot |I|^{O(1)}$ lösbar ist.

Bzgl. $|S|$:

- Angenommen, SETSPLITTING wäre in $2^{o(|S|)} \cdot |I|^{O(1)}$ lösbar.
- Wegen $|S| = 2n$ ergäbe sich ein $2^{o(n)}$ -Algorithmus für NAE-4-SAT.
- Nach dem Hinweis ist das unter der ETH ausgeschlossen – Widerspruch.

Bzgl. r :

- Angenommen, SETSPLITTING wäre in $2^{o(r)} \cdot |I|^{O(1)}$ lösbar.

- Jede NAE-Klausel hat höchstens 4 Literale, also $n \leq 4m + 1 = O(m)$.
- Damit ist $r = n + m = O(m)$.
- Es ergäbe sich ein $2^{o(m)}$ -Algorithmus für NAE-4-SAT.
- Nach dem Hinweis ist das unter der ETH ausgeschlossen – Widerspruch.

Bounds: kein $2^{o(|S|)}$, kein $2^{o(r)}$.

□

CoverClique. Gegeben: Ein ungerichteter Graph $G = (V, E)$ und eine Zahl $k \in \mathbb{N}_{\geq 1}$.
Entscheide: Existieren k paarweise disjunkte Cliques $C_1, \dots, C_k \subseteq V$, die alle Knoten überdecken?

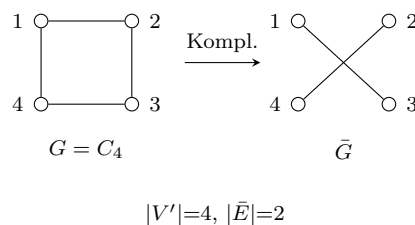
Reduktion von k -COLOR: Aus (G, k) wird $(G' = (V, \bar{E}), k)$ mit dem Komplementgraphen, also $\bar{E} = \{\{v, u\} \mid v \neq u, \{v, u\} \notin E\}$.

- Geben Sie die Anzahl der Knoten und die Anzahl der Kanten der CoverClique-Instanz in Abhängigkeit von der k -Color-Instanz an.
- Beweisen Sie Laufzeit-Lower-Bounds für COVERCLIQUE bezüglich der Anzahl der Knoten und der Anzahl der Kanten basierend auf der ETH und der oben beschriebenen Reduktion.

Hinweis: k -COLOR ist unter der ETH nicht in $2^{o(|V|)} \cdot |I|^{O(1)}$ lösbar.

Beispiel Lower Bounds CoverClique: Mini-Gadget

Instanz. k -COLOR-Instanz $G = C_4$ mit $k = 2$; die Reduktion bildet den Komplementgraphen $\bar{G}$.



Kern. Wegen $|V'| = |V| = 4$ wäre ein $2^{o(|V'|)}$ -Algorithmus für COVERCLIQUE ein $2^{o(|V|)}$ -Algorithmus für k -COLOR – Widerspruch zur ETH. Kein $2^{o(|V'|)}$.

Lösung.

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

b) Zu zeigen ist, dass COVERCLIQUE unter der ETH weder in $2^{o(|V'|)} \cdot |I|^{O(1)}$ noch in $2^{o(\sqrt{|\bar{E}|})} \cdot |I|^{O(1)}$ lösbar ist.

Bzgl. $|V'|$:

- Angenommen, COVERCLIQUE wäre in $2^{o(|V'|)} \cdot |I|^{O(1)}$ lösbar.
- Wegen $|V'| = |V|$ ergäbe sich ein $2^{o(|V|)}$ -Algorithmus für k -COLOR.
- Den gibt es unter der ETH nicht – Widerspruch.

Bzgl. $|\bar{E}|$:

- Angenommen, COVERCLIQUE wäre in $2^{o(\sqrt{|\bar{E}|})} \cdot |I|^{O(1)}$ lösbar.
- Wegen $|\bar{E}| \leq |V|^2$ ist $\sqrt{|\bar{E}|} = O(|V|)$.
- Es ergäbe sich wieder ein $2^{o(|V|)}$ -Algorithmus für k -COLOR – Widerspruch zur ETH.

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

Δ Cover. Gegeben: Ein ungerichteter Graph $G = (V, E)$ und eine Zahl $k \in \mathbb{N}_0$. **Entscheide:** Gibt es eine Dreiecksüberdeckung $C_\Delta \subseteq V$ mit $|C_\Delta| \leq k$, also eine Knotenmenge, die jedes Dreieck in G trifft?

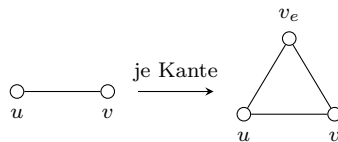
Reduktion von VERTEXCOVER (aus der Übung): $V' = V \cup \{v_e \mid e \in E\}$ und $E' = E \cup \{\{v_e, e_1\}, \{v_e, e_2\} \mid e = \{e_1, e_2\} \in E\}$.

- Geben Sie die Anzahl der Knoten und die Anzahl der Kanten der Δ Cover-Instanz in Abhängigkeit von der VertexCover-Instanz an.
- Beweisen Sie Laufzeit-Lower-Bounds für Δ Cover bezüglich der Anzahl der Knoten und der Anzahl der Kanten basierend auf der ETH und der oben beschriebenen Reduktion.

Hinweis: VERTEXCOVER ist unter der ETH weder in $2^{o(|V|)} \cdot |I|^{O(1)}$ noch in $2^{o(\sqrt{|E|})} \cdot |I|^{O(1)}$ lösbar.

Beispiel Lower Bounds Δ Cover: Mini-Gadget

Instanz. VERTEXCOVER-Instanz mit der einen Kante $\{u, v\}$; die Reduktion setzt auf jede Kante ein Dreieck mit neuem Knoten v_e .



$$|V'|=3, |E'|=3$$

Kern. Wegen $|E'| = 3|E| = 3$ wäre ein $2^{o(\sqrt{|E'|})}$ -Algorithmus für Δ Cover ein $2^{o(\sqrt{|E|})}$ -Algorithmus für VERTEXCOVER – Widerspruch zur ETH. Kein $2^{o(\sqrt{|E'|})}$.

Lösung.

a) $|V'| = |V| + |E|$, $|E'| = |E| + 2|E| = 3|E|$.

b) Zu zeigen ist, dass Δ Cover unter der ETH weder in $2^{o(\sqrt{|V'|})} \cdot |I|^{O(1)}$ noch in $2^{o(\sqrt{|E'|})} \cdot |I|^{O(1)}$ lösbar ist. **Bzgl. $|V'|$:**

- Angenommen, Δ Cover wäre in $2^{o(\sqrt{|V'|})} \cdot |I|^{O(1)}$ lösbar.
- Wegen $|V'| = |V| + |E| \leq |V| + |V|^2 = O(|V|^2)$ ist $\sqrt{|V'|} = O(|V|)$.
- Es ergäbe sich ein $2^{o(|V|)}$ -Algorithmus für VERTEXCOVER – Widerspruch zur ETH.

Bzgl. $|E'|$:

- Angenommen, Δ Cover wäre in $2^{o(\sqrt{|E'|})} \cdot |I|^{O(1)}$ lösbar.
- Wegen $|E'| = 3|E| = O(|E|)$ ist $\sqrt{|E'|} = O(\sqrt{|E|})$.
- Es ergäbe sich ein $2^{o(\sqrt{|E|})}$ -Algorithmus für VERTEXCOVER – Widerspruch zur ETH.

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

5 Approximation: anwenden, Güte beweisen, Worst Case

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.

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):

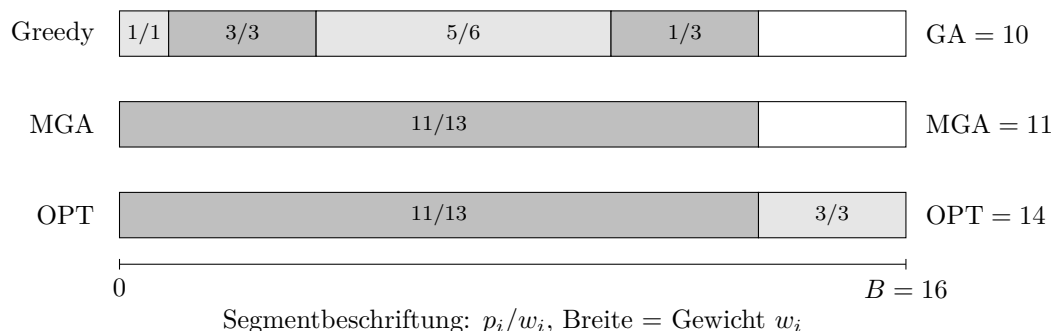
- (1, 1) einpacken – Restkapazität 15.
- (3, 3) einpacken – Restkapazität 12.
- (11, 13) passt nicht ($13 > 12$).
- (5, 6) einpacken – Restkapazität 6.
- (1, 3) einpacken – Restkapazität 3.
- (1, 4) und (1, 5) passen nicht.

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

b) ModifiedGreedy vergleicht zwei Kandidaten:

- Greedy-Wert: $GA = 10$.
- Profitreichstes Einzel-Item: (11, 13) mit Profit 11.
- $MGA = \max\{10, 11\} = 11$.

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

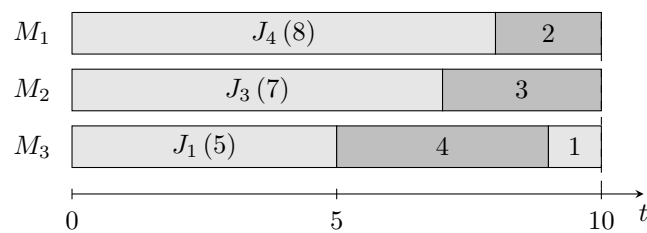
- Wenden Sie LPT an (Zwischenschritte, Reihenfolge, Schedule grafisch).
- Approximative Güte von LPT?

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

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

□

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

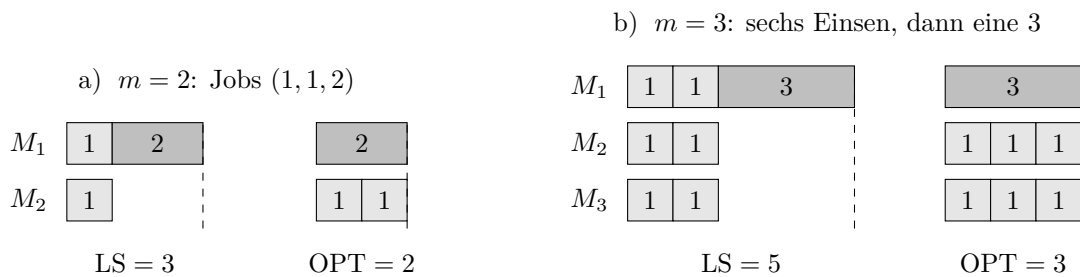
- a) Geben Sie eine Instanz für $m = 2$ an, bei der ListScheduling die Güte $2 - \frac{1}{m}$ erreicht (mit OPT und LS-Wert).
- b) Dasselbe für $m = 3$.

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

Lösung.

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

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



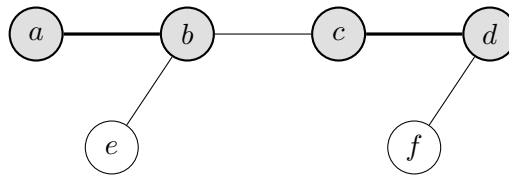
□

Algorithmus **2ApproxCV**: Gehe alle Kanten durch; sind beide Endpunkte noch nicht in C , füge beide zu C hinzu. Beweisen Sie die Güte 2.

Lösung.

Sei A die Menge der Kanten, für die beide Endpunkte aufgenommen wurden. Dann gilt $|C| = 2|A|$.

Die Kanten in A sind paarweise knotendisjunkt: Hätten zwei Kanten aus A einen gemeinsamen Endpunkt, wäre bei der später betrachteten dieser Endpunkt schon in C gewesen – sie wäre nicht aufgenommen worden. A ist also ein Matching.



dick = A , gefüllt = C : $|C| = 2|A| = 4$

Sei C^* ein minimales Vertex Cover. C^* enthält von jeder Kante aus A mindestens einen Endpunkt, und da die Kanten aus A knotendisjunkt sind, braucht es für jede einen *eigenen* Knoten: $|C^*| \geq |A|$.

Zusammen: $|C| = 2|A| \leq 2|C^*|$. □

MAX-3-SAT: Gegeben eine Formel φ in KNF mit drei Literalen pro Klausel; gesucht eine Belegung β , die die Anzahl $v(\beta)$ der erfüllten Klauseln maximiert.

Algorithmus A : Werte β_0 (alle false) und β_1 (alle true) aus, gib die bessere Belegung zurück.

- Zeigen Sie $v(A(\varphi)) \geq \frac{1}{2}v(\text{OPT}(\varphi))$ für alle φ .
- Geben Sie eine Formel an, bei der A genau die Hälfte der maximal erfüllbaren Klauseln erfüllt (Begründung).

Lösung.

a) Zu zeigen ist, dass A die Güte 2 hat, also

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

Sei φ eine Formel mit m Klauseln. Betrachte die beiden Belegungen β_0 (alle false) und β_1 (alle true):

- Unter β_0 ist jedes negative Literal wahr.
- Unter β_1 ist jedes positive Literal wahr.

Jedes Literal ist positiv oder negativ, und jede Klausel enthält mindestens ein Literal. Also wird jede Klausel von β_0 oder von β_1 (oder von beiden) erfüllt. Da damit jede Klausel zu $v(\beta_0)$ oder zu $v(\beta_1)$ zählt, gilt

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

Der Algorithmus wählt die bessere der beiden Belegungen. Das Optimum erfüllt höchstens alle m Klauseln, also $v(\text{OPT}(\varphi)) \leq m$. Damit folgt

$$v(A(\varphi)) = \max\{v(\beta_0), v(\beta_1)\} \geq \frac{v(\beta_0) + v(\beta_1)}{2} \geq \frac{m}{2} \geq \frac{v(\text{OPT}(\varphi))}{2}.$$

Somit hat A die Güte 2.

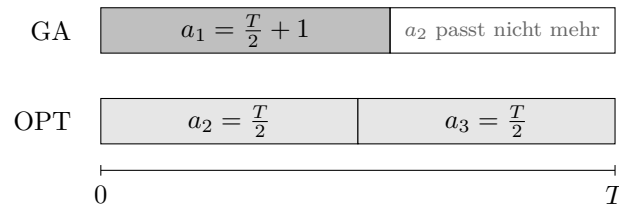
b) $\varphi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3)$. Mit $x_1 = \text{wahr}$, $x_2 = x_3 = \text{falsch}$ sind beide Klauseln erfüllt: $\text{OPT} = 2$. Aber β_0 erfüllt nur die zweite Klausel, β_1 nur die erste: $v(A(\varphi)) = 1 = \frac{1}{2} \cdot \text{OPT}$.
 $\square$

APPROXIMATESUBSETSUM: Zahlen $a_1, \dots, a_n \leq T$ mit $\sum a_i > T$; maximiere $\sum_{j \in S} a_j \leq T$.
 Algorithmus GA: sortiere absteigend, nimm der Reihe nach, *stoppe* beim ersten nicht mehr passenden Element.

- Konstruktionsvorschrift für Instanzen, mit denen GA beliebig nah an Güte 2 kommt.
- Zeigen Sie $\text{GA}(I) \geq \text{OPT}(I)/2$.

Lösung.

a) Für gerades T : $a_1 = \frac{T}{2} + 1$, $a_2 = a_3 = \frac{T}{2}$. GA nimmt a_1 und stoppt, denn $a_1 + a_2 > T$:
 $\text{GA} = \frac{T}{2} + 1$. Optimal ist $a_2 + a_3 = T$. Rate $\frac{T}{T/2+1} \rightarrow 2$ für $T \rightarrow \infty$.



b) Sei a_ℓ das erste Element, das nicht mehr passte (existiert wegen $\sum a_i > T$). Dann gilt

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

Wegen der absteigenden Sortierung enthält die GA-Auswahl 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 $\text{GA}(I) \geq \text{OPT}(I)/2$. $\square$

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

- a) Zeigen Sie per Widerspruch, dass RoundRobin Güte 2 hat.
- b) **Bonus:** Konstruktionsvorschrift für Instanzen mit $\text{Rate} \rightarrow 2 - \frac{1}{m}$ bei beliebigem festen m (mit OPT- und RR-Wert).

Lösung.

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

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

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

b) Ein Job der Größe m und $m(m-1)$ Jobs der Größe 1 (absteigend sortiert steht der große zuerst). RoundRobin gibt Maschine 1 den großen Job und danach reihum jede m -te Eins – $m-1$ weitere: $\text{RR} = m + (m-1) = 2m-1$. Optimal: großer Job allein, die $m(m-1)$ Einsen auf die übrigen $m-1$ Maschinen (je m): $\text{OPT} = m$. $\text{Rate} \frac{2m-1}{m} = 2 - \frac{1}{m}$. (Für $m=3$: $\text{RR} = 5$, $\text{OPT} = 3$, $\text{Rate} \frac{5}{3}$.)

M_1	3	1	1
M_2	1	1	
M_3	1	1	

RoundRobin: $\text{RR} = 5$

M_1	3		
M_2	1	1	1
M_3	1	1	1

Optimum: $\text{OPT} = 3$

□

MIN-EDGE-COVER: Finde eine kleinste Kantenmenge C , sodass jeder Knoten zu einer Kante aus C inzident ist (G zusammenhängend). Algorithmus A : Gehe die Knoten in beliebiger Reihenfolge durch; ist der aktuelle Knoten v unbedeckt, füge eine beliebige zu v inzidente Kante hinzu.

- Zeigen Sie $A(G) \leq 2 \text{OPT}(G)$.
- Kreisgraph G_n ($n \geq 4$ gerade): Größe eines Min-Edge-Covers? Schlechteste Approximationsrate von A ? Geben Sie Reihenfolge und Kantenwahl an.

Lösung.

a) Der Algorithmus fügt höchstens eine Kante pro Knoten hinzu, also $A(G) \leq |V|$. Jede Kante deckt höchstens zwei Knoten, also braucht jedes Edge Cover mindestens $|V|/2$ Kanten: $\text{OPT}(G) \geq |V|/2$. Zusammen: $A(G) \leq |V| \leq 2 \text{OPT}(G)$.

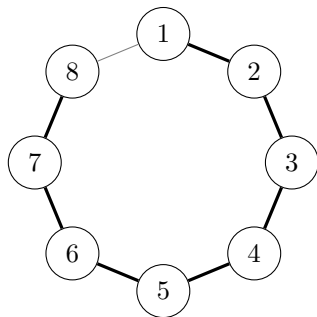
b) Im Kreis G_n bilden $n/2$ paarweise disjunkte Kanten (jede zweite Kreiskante) ein Edge Cover – weniger geht nicht, also $\text{OPT} = n/2$.

Schlechteste Ausführung: Reihenfolge $1, 3, 4, 5, \dots, n$.

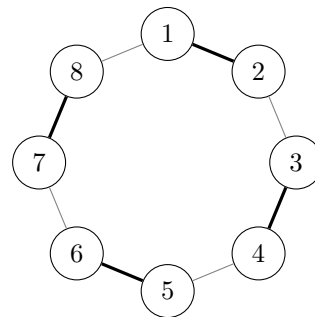
- Knoten 1 wählt $\{1, 2\}$ – deckt die zwei neuen Knoten 1 und 2.
- Knoten 3 wählt $\{3, 2\}$ – 2 ist schon gedeckt, die Kante deckt nur den neuen Knoten 3.
- Knoten 4 wählt $\{4, 3\}$ – deckt nur den neuen Knoten 4.
- Knoten 5 wählt $\{5, 4\}$ – deckt nur den neuen Knoten 5.
- So weiter bis Knoten n : Er wählt $\{n, n-1\}$ – deckt nur den neuen Knoten n .

Nach der ersten Kante deckt jede weitere nur einen neuen Knoten: insgesamt $1 + (n-2) = n-1$ Kanten. Rate

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



A : $n-1 = 7$ Kanten



OPT : $n/2 = 4$ Kanten

□

Leitergraph: K_n ($n \bmod 4 = 2$), Kanten $E_2 = \{\{i, i+2\}\} \cup \{\{2i+1, 2i+2\}\}$ mit Gewicht 1; alle übrigen Distanzen = kürzester Weg. Zeigen Sie: (a) $\text{OPT} = n$; (b) es gibt eine Ausführung von ΔTSP2 mit Tourlänge $(n-1) + n/2$.

Hinweis (ΔTSP2 , Christofides): (1) MST T ; (2) X = Knoten mit ungeradem Grad in T ; (3) minimales perfektes Matching K auf X ; (4) Eulerkreis in $T+K$; (5) Abkürzen. Bei Gleichständen (mehrere MSTs, mehrere Eulerkreise) darf die Ausführung beliebig – auch ungünstigst – gewählt werden.

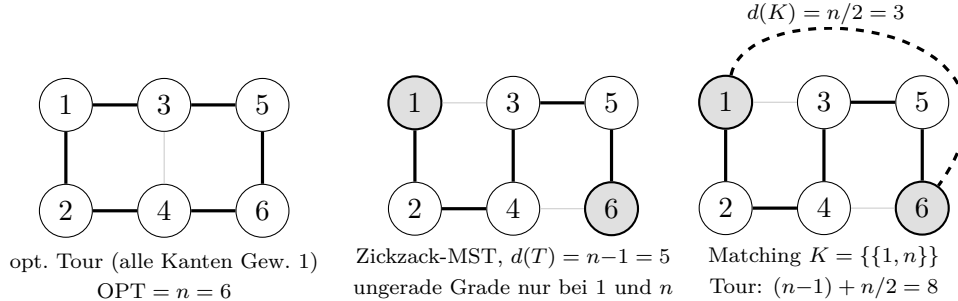
Lösung.

a) Die Tour „oben hin, unten zurück“ – $1, 3, 5, \dots, n-1, n, n-2, \dots, 2, 1$ – benutzt n Kanten vom Gewicht 1, also $\text{OPT} \leq n$. Weniger geht nicht (n Knoten brauchen n Tourkanten vom Gewicht ≥ 1): $\text{OPT} = n$.

b) Bei Gleichständen darf die Ausführung (MST-Wahl, Knotenreihenfolge) adversariell gewählt werden. Wähle den Zickzack-MST

$$1-2, 2-4, 4-3, 3-5, 5-6, 6-8, 8-7, \dots, \{n-1, n\}$$

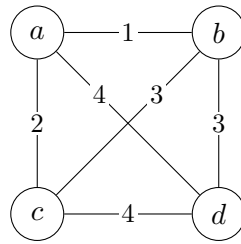
mit $n-1$ Kanten vom Gewicht 1: $d(T) = n-1$. In diesem Baum haben nur die Knoten 1 und n ungeraden Grad, also ist das Matching $K = \{\{1, n\}\}$ – Kosten = kürzester Weg von 1 nach $n = n/2$. Die Tour aus Eulerkreis und Abkürzen hat damit Länge $d(T) + d(K) = (n-1) + n/2$.



Rate: $\frac{n-1+n/2}{n} = \frac{3}{2} - \frac{1}{n} \rightarrow \frac{3}{2}$.

□

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



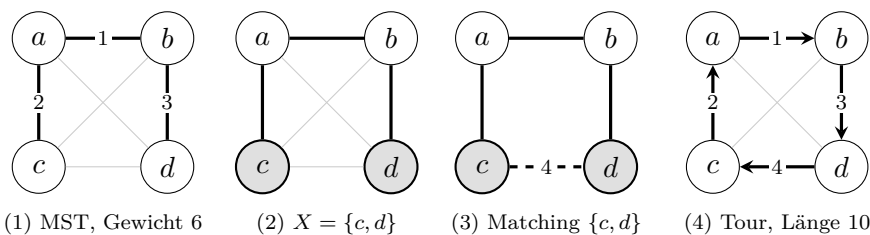
- Wenden Sie Christofides (Δ TSP2) auf den Graphen G mit Startknoten a an; geben Sie alle Zwischengraphen an, auch Schritte ohne Änderung.
- Was berechnet Christofides? Nennen Sie die zwei Zusatzeigenschaften der Eingabe, die er braucht, mit je einer verletzenden Eingabe. Geben Sie die Güte an.

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.

Lösung.

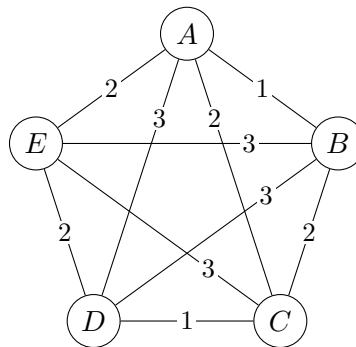
a)

- 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: $\frac{3}{2}$. $\square$

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



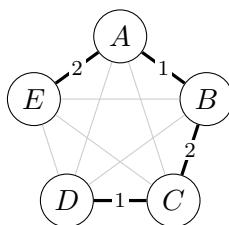
- Wenden Sie Δ TSP1 auf den Graphen G ab Startknoten C an; geben Sie alle Zwischengraphen und die Länge der Rundreise an.
- Was berechnet Δ TSP1, und welche Bedingungen muss der Graph erfüllen?
- Begründen Sie die approximative Güte 2.
- Ist das Entscheidungsproblem zum TSP in P ? Kurze Begründung.
- 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.

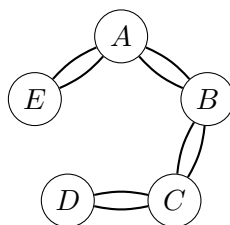
Lösung.

a)

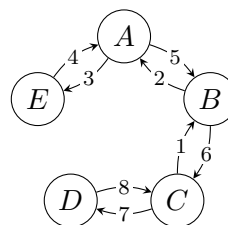
- MST (Kruskal): $A-B$ (1), $C-D$ (1), $B-C$ (2), $A-E$ (2); Gewicht 6.
- Verdoppeln: Multigraph, Gewicht 12 – alle Grade gerade.
- Eulerkreis ab C : $[C, B, A, E, A, B, C, D, C]$.
- Abkürzen: Tour $[C, B, A, E, D, C]$, Länge 8.



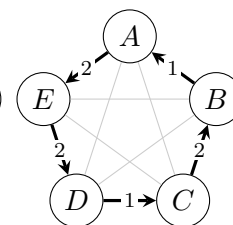
(1) MST, Gewicht 6



(2) verdoppelt, Gewicht 12



(3) Eulerkreis ab C



(4) Tour, 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. $\square$

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.

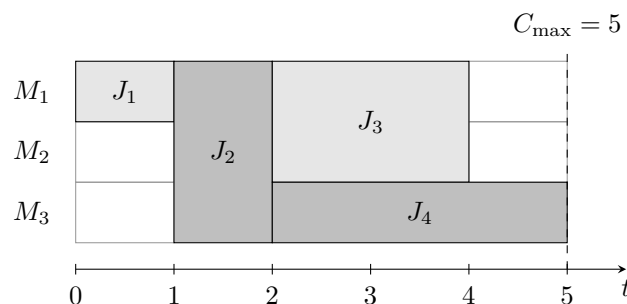
- Anwendung: $m = 3$; Jobs (p, q) : $(1, 1), (1, 3), (2, 2), (3, 1)$. Geben Sie den Schedule an.
- Worst-Case-Güte für $q_j = 1$?
- Zeigen Sie $LS(I) \leq OPT(I) + \frac{1}{2}p_{\max}$ für alle I mit $\frac{m}{3} < q_j \leq \frac{m}{2}$.
- Bonus:** Güte für $\frac{m}{k+1} < q_j \leq \frac{m}{k}$ ($k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

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

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

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

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

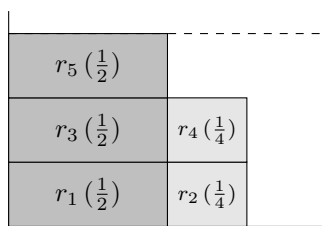
□

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.

- a) 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.
- b) Zeigen Sie: Es gibt L_n mit $\lim_{n \rightarrow \infty} \text{NFDH}(L_n)/\text{OPT}(L_n) = 2$.
- c) Zeigen Sie für Instanzen mit $h(r_i) = 1$: $\text{NFDH}(L) \leq 2 \text{OPT}(L) + 1$.
- d) **Bonus:** Zeigen Sie dieselbe Schranke für $h(r_i) \leq 1$.

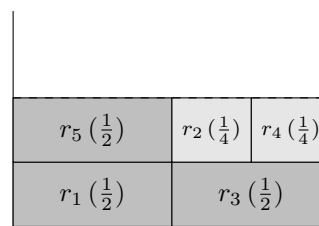
Lösung.

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: $\frac{3}{2}$.



Breite 1

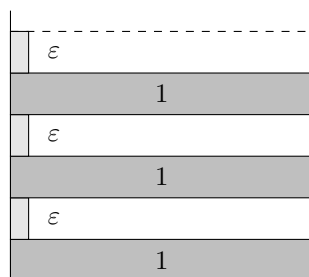
NFDH = 3



Breite 1

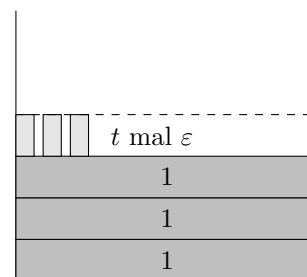
OPT = 2

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

NFDH = 2t

OPT: alle ε auf eine StufeOPT $\leq t + 1$

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

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

□

6 Wahr oder falsch?

Diskutieren Sie:

- (i) Für jede Sprache, die eine NDTM in polynomieller Zeit akzeptiert, existiert eine DTM, die sie in polynomieller Zeit akzeptiert.
- (ii) A ist NP-schwer $\implies A \in P$.

Lösung.

- (i) Gilt genau dann, wenn $P = NP$: Dann liegen alle NDTM-polynomiell lösbaren Probleme ($= NP$) auch in P . Falls $P \neq NP$, gilt sie nicht – ein Problem aus $NP \setminus P$ wäre sonst per DTM-Simulation in P .
- (ii) Falsch. Es gibt NP-schwere Probleme, die nicht einmal in NP liegen, z. B. das Halteproblem – diese liegen erst recht nicht in P . (NP-schwer ist eine untere Schranke, $\in NP$ eine obere; beides zusammen ist NP-vollständig.) $\square$

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 $\text{CLIQUE} \notin P$.

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 $\text{CLIQUE} \in P$. $\square$

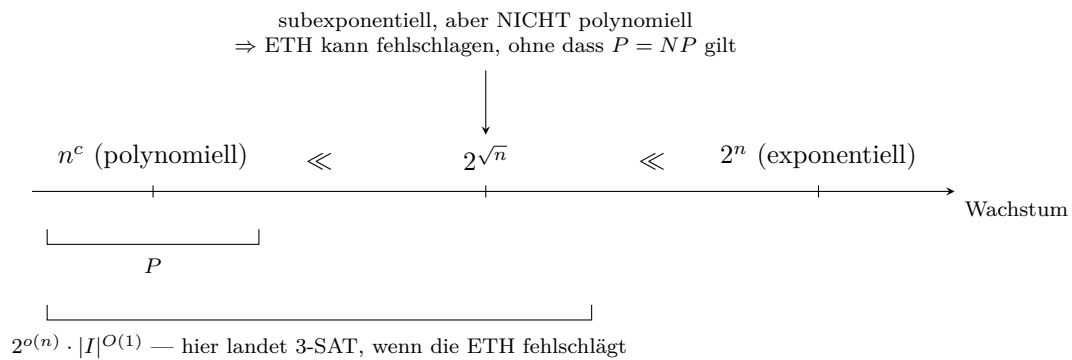
Die *Exponentialzeithypothese* (ETH) besagt: 3-SAT mit n Variablen ist nicht in Zeit $2^{o(n)} \cdot |I|^{O(1)}$ lösbar.

Beweisen oder widerlegen Sie: Wenn die ETH fehlschlägt, gilt $P = NP$.

Lösung.

Die Aussage ist falsch (nicht beweisbar; die Implikation gilt nicht).

„ETH fehlschlägt“ bedeutet nur: 3-SAT ist in $2^{o(n)} \cdot |I|^{O(1)}$ lösbar – also subexponentiell mal ein Polynom. Subexponentiell ist nicht polynomiell: Eine Laufzeit wie $2^{\sqrt{n}}$ erfüllt $2^{o(n)}$, ist aber superpolynomiell. Ein Fehlschlagen der ETH wäre also damit verträglich, dass 3-SAT (und damit alle NP-Probleme) zwar in $2^{\sqrt{n}}$, aber nicht in Polynomialzeit lösbar sind – $P \neq NP$ bliebe möglich.



Die Umkehrung gilt dagegen: Aus $P = NP$ folgt $3\text{-SAT} \in P \subseteq 2^{o(n)}$, also wäre die ETH falsch. Die behauptete Richtung folgt daraus nicht. $\square$

7 Turingmaschinen

A64 TM für Palindrome

Hausaufgabe 10.2, 5 P.

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.

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 $O(n^2)$. $\square$

Entwerfen Sie eine Turingmaschine (Einband-TM) für $L = \{0^{2^n} \mid n \in \mathbb{N}\}$ über $\Sigma = \{0\}$; begründen Sie Korrektheit und Laufzeit. Diskutieren Sie die Laufzeit im Vergleich zu einem RAM-Algorithmus.

Beispiel TM für $L = \{0^{2^n}\}$

Instanz. Eingabe $w = 0000 (= 0^{2^2})$, also in L .

Kern. Jeder Durchlauf ersetzt jede zweite 0 durch x und halbiert so die Anzahl der Nullen: $0000 \rightarrow 0x0x \rightarrow 0xxx$. Nie tritt eine ungerade Anzahl > 1 auf, am Ende bleibt genau eine 0 – die Maschine akzeptiert.

Lösung.**Die Maschine:**

- (1) Falls genau eine 0 auf dem Band steht, akzeptiere.
- (2) Laufe über das Band und ersetze jede zweite 0 durch x .
- (3) Falls das letzte Symbol eine 0 war (ungerade Anzahl), 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: Jeder Durchlauf halbiert die Anzahl der Nullen; eine ungerade Anzahl > 1 führt zum Verwerfen.

Laufzeit: Jeder Durchlauf kostet $O(n)$; die Anzahl halbiert sich pro Durchlauf, also $O(\log n)$ Durchläufe. Insgesamt $O(n \log n)$.

RAM-Vergleich: Ein RAM-Algorithmus zählt die Nullen ($O(n)$) und prüft, ob die Anzahl eine Zweierpotenz ist ($O(\log n)$) – Laufzeit $O(n)$. Der wahlfreie Speicherzugriff erlaubt es, die Aufgabe schneller zu lösen als die kopfbewegungsgebundene Turingmaschine. $\square$