

Theoretische Grundlagen der Informatik

Komplexitätsklassen - Teil 1

INSTITUT FÜR THEORETISCHE INFORMATIK



Fragestellung bisher:

- Ist eine Sprache L entscheidbar oder nicht?
- Ist eine Funktion berechenbar oder nicht?
- Benutzung von deterministischen Turing-Maschinen.

In diesem Kapitel:

- Wie effizient kann ein Problem gelöst werden?
- Betrachtung von nichtdeterministischen Turing-Maschinen.

Frage (P vs. NP):

Gibt es einen wesentlichen Effizienzgewinn beim Übergang der deterministischen Turing-Maschine zur nichtdeterministischen Turing-Maschine?

- **Sprachen**
- **Probleme**
- **Zeitkomplexität**

Beispiel: Traveling Salesman Problem (TSP)

Gegeben sei ein vollständiger Graph $G = (V, E, c)$, d.h.

- $V := \{1, \dots, n\}$
- $E := \{\{u, v\} \mid u, v \in V, u \neq v\}$
- $c: E \rightarrow \mathbb{Z}^+$.

Wir betrachten folgende Problemvarianten

- **Optimierungsproblem:**
Gesucht ist eine Tour (Rundreise), die alle Elemente aus V enthält und minimale Gesamtlänge unter allen solchen Touren hat.
- **Optimalwertproblem:**
Gesucht ist die Länge einer minimalen Tour.
- **Entscheidungsproblem:**
Gegeben sei zusätzlich auch ein Parameter $k \in \mathbb{Z}^+$. Die Frage ist nun: Gibt es eine Tour, deren Länge höchstens k ist?

Wir betrachten folgende Problemvarianten

- **Optimierungsproblem:**

Gesucht ist eine Tour (Rundreise), die alle Elemente aus V enthält und minimale Gesamtlänge unter allen solchen Touren hat.

- **Optimalwertproblem:**

Gesucht ist die Länge einer minimalen Tour.

- **Entscheidungsproblem:**

Gegeben sei zusätzlich auch ein Parameter $k \in \mathbb{Z}^+$. Die Frage ist nun: Gibt es eine Tour, deren Länge höchstens k ist?

Bemerkung:

- Mit einer Lösung des Optimierungsproblems kann man leicht auch das Optimalwertproblem und das Entscheidungsproblem lösen.
- Mit einer Lösung des Optimalwertproblems kann man leicht auch das Entscheidungsproblem lösen.

Definition: Problem

Ein **Problem** Π ist gegeben durch:

- eine allgemeine Beschreibung aller vorkommenden Parameter;
- eine genaue Beschreibung der Eigenschaften, die die Lösung haben soll.

Ein **Problembispiel** / (**Instanz**) von Π erhalten wir, indem wir die Parameter von Π festlegen.

Definition: Kodierungsschema

- Wir interessieren uns für die **Laufzeit** von Algorithmen.
- Diese wird in der Größe des Problems gemessen.

Die Größe eines Problems ist abhängig von der Beschreibung oder Kodierung der Problembeispiele

- Ein **Kodierungsschema** s ordnet jedem Problembeispiel eines Problems eine Zeichenkette oder *Kodierung* über einem Alphabet Σ zu.
- Die **Inputlänge** eines Problembeispiels ist die Anzahl der Symbole seiner Kodierung.

Es gibt verschiedene Kodierungsschemata für ein bestimmtes Problem.

Beispiel:

- Zahlen können dezimal, binär, unär, usw. kodiert werden.
- Die Inputlänge von 5127 beträgt dann 4 für dezimal, 13 für binär und 5127 für unär.

Wir werden uns auf vernünftige Schemata festlegen:

- Die Kodierung eines Problembeispiels sollte keine überflüssigen Informationen enthalten.
- Zahlen sollen binär (oder k -är für $k \neq 1$) kodiert sein.

Dies bedeutet, die Kodierungslänge

- einer ganzen Zahl n ist $\lfloor \log_k |n| + 1 \rfloor + 1 =: \langle n \rangle$
(eine 1 benötigt man für das Vorzeichen);
- einer rationalen Zahl $r = \frac{p}{q}$ ist $\langle r \rangle = \langle p \rangle + \langle q \rangle$;
- eines Vektors $X = (x_1, \dots, x_n)$ ist $\langle X \rangle := \sum_{i=1}^n \langle x_i \rangle$;
- einer Matrix $A \in \mathbb{Q}^{m \times n}$ ist $\langle A \rangle := \sum_{i=1}^m \sum_{j=1}^n \langle a_{ij} \rangle$.
- eines Graphen $G = (V, E)$ kann zum Beispiel durch die Kodierung seiner *Adjazenzmatrix*, die eines gewichteten Graphen durch die Kodierung der *Gewichtsmatrix* beschrieben werden.

Äquivalenz von Kodierungsschemata

Zwei Kodierungsschemata s_1, s_2 heißen **äquivalent** bezüglich eines Problems Π , falls es Polynome p_1, p_2 gibt, so dass gilt:

$$(|s_1(I)| = n \Rightarrow |s_2(I)| \leq p_2(n)) \text{ und } (|s_2(I)| = m \Rightarrow |s_1(I)| \leq p_1(m))$$

für alle Problembeispiele I von Π .

- Ein Entscheidungsproblem Π können wir als Klasse von Problembeispielen D_{Π} auffassen.
- Eine Teilmenge dieser Klasse ist $J_{\Pi} \subseteq D_{\Pi}$, die Klasse der **Ja-Beispiele**, d.h. die Problembeispiele deren Antwort Ja ist.
- Der Rest der Klasse $N_{\Pi} \subseteq D_{\Pi}$ ist die Klasse der **Nein-Beispiele**.

Ein Problem Π und ein Kodierungsschema $s: D_{\Pi} \rightarrow \Sigma^*$ zerlegen Σ^* in drei Klassen:

- Wörter aus Σ^* , die *nicht* Kodierung eines Beispiels aus D_{Π} sind,
- Wörter aus Σ^* , die Kodierung eines Beispiels $I \in N_{\Pi}$ sind,
- Wörter aus Σ^* , die Kodierung eines Beispiels $I \in J_{\Pi}$ sind.

Die dritte Klasse ist die Sprache, die zu Π im Kodierungsschema s korrespondiert.

Die zu einem Problem Π und einem Kodierungsschema s **zugehörige Sprache** ist

$$L[\Pi, s] := \left\{ x \in \Sigma^* \mid \begin{array}{l} \Sigma \text{ ist das Alphabet zu } s \text{ und } x \text{ ist Kodierung} \\ \text{eines Ja-Beispiels } I \text{ von } \Pi \text{ unter } s, \text{ d.h. } I \in J_{\Pi} \end{array} \right\}$$

- Wir betrachten im folgenden deterministische Turing-Maschinen mit zwei Endzuständen q_J, q_N , wobei q_J akzeptierender Endzustand ist.
- Dann wird die Sprache $L_{\mathcal{M}}$ akzeptiert von der Turing-Maschine \mathcal{M} , falls

$$L_{\mathcal{M}} = \{x \in \Sigma^* \mid \mathcal{M} \text{ akzeptiert } x\} .$$

- Eine deterministische Turing-Maschine \mathcal{M} **löst** ein Entscheidungsproblem Π unter einem Kodierungsschema s , falls \mathcal{M} bei jeder Eingabe über dem Eingabe-Alphabet in einem Endzustand endet und $L_{\mathcal{M}} = L[\Pi, s]$ ist.

Für eine deterministische Turing-Maschine \mathcal{M} , die für alle Eingaben über dem Eingabe-Alphabet Σ hält, ist die **Zeitkomplexitätsfunktion**

$T_{\mathcal{M}}: \mathbb{Z}^+ \rightarrow \mathbb{Z}^+$ definiert durch

$$T_{\mathcal{M}}(n) = \max \left\{ m \mid \begin{array}{l} \text{es gibt eine Eingabe } x \in \Sigma^* \text{ mit } |x| = n, \text{ so} \\ \text{dass die Berechnung von } \mathcal{M} \text{ bei Eingabe } x \\ m \text{ Berechnungsschritte (Übergänge) benötigt,} \\ \text{bis ein Endzustand erreicht wird} \end{array} \right\}$$

Die Klasse \mathcal{P} ist die Menge aller Sprachen L (Probleme), für die eine deterministische Turing-Maschine existiert, deren Zeitkomplexitätsfunktion polynomial ist, d.h. es existiert ein Polynom p mit

$$T_{\mathcal{M}}(n) \leq p(n).$$

Schwierigkeit von Entscheidungs und Optimierungsproblem

Satz:

Falls es einen Algorithmus \mathcal{A} gibt, der das Entscheidungsproblem des TSP in polynomialer Zeit löst, so gibt es auch einen Algorithmus, der das Optimierungsproblem in polynomialer Zeit löst.

Schwierigkeit von Entscheidungs und Optimierungsproblem

Satz:

Falls es einen Algorithmus \mathcal{A} gibt, der das Entscheidungsproblem des TSP in polynomialer Zeit löst, so gibt es auch einen Algorithmus, der das Optimierungsproblem in polynomialer Zeit löst.

Beweis: Algorithmus, der das Optimierungsproblem löst.

Input: $G = (V, E)$, $c_{ij} = c(\{i, j\})$ für $i, j \in V := \{1, \dots, n\}$,
Algorithmus \mathcal{A}

Output: d_{ij} ($1 \leq i, j \leq n$), so dass alle bis auf n der d_{ij} -Werte den Wert $\left(\max_{i,j} c_{ij}\right) + 1$ haben. Die restlichen n d_{ij} -Werte haben den Wert c_{ij} und geben genau die Kanten einer optimalen Tour an.

Algorithmus OPT-TOUR (als Beweis)

- berechne $m := \max_{1 \leq i, j \leq n} c_{ij}$;
setze $L(\text{ow}) := 0$ und $H(\text{igh}) := n \cdot m$;
- solange $H - L > 1$ gilt, führe aus:
 - falls $\mathcal{A}\left(n, c, \left\lceil \frac{1}{2}(H + L) \right\rceil\right) = \text{nein}$ ist,
setze $L := \left\lceil \frac{1}{2}(H + L) \right\rceil + 1$;
 - sonst
setze $H := \left\lceil \frac{1}{2}(H + L) \right\rceil$;

} (* binäre Suche *)
- falls $\mathcal{A}(n, c, L) = \text{„nein“}$ ist, setze $OPT := H$; sonst setze $OPT := L$;
- für $i = 1 \dots n$ führe aus
 - für $j = 1 \dots n$ führe aus
 - setze $R := c_{ij}$; $c_{ij} := m + 1$;
 - falls $\mathcal{A}(n, c, OPT) = \text{nein}$ ist, setze $c_{ij} := R$;
- setze $d_{ij} = c_{ij}$;

Die Schleife in Schritt 2 bricht ab, und danach ist die Differenz $H - L$ gleich 1 oder 0, denn:

- Solange $H - L > 1$, ändert sich bei jedem Schleifendurchlauf einer der Werte H, L :
 - Für $H - L > 1$ gilt, dass $L \neq \left\lceil \frac{1}{2}(H + L) \right\rceil + 1$ und $H \neq \left\lfloor \frac{1}{2}(H + L) \right\rfloor$ ist.
- Die Differenz verkleinert sich also mit jedem Durchlauf
- Da H und L ganzzahlig sind, tritt der Fall $H - L \leq 1$ ein.

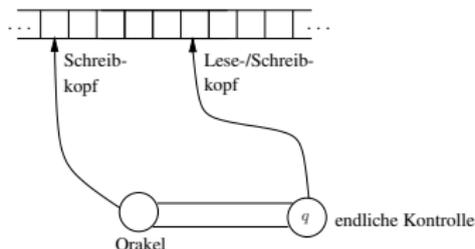
- Nach Abbruch der Schleife gilt $H - L \geq 0$:
 - Eine Differenz zwischen H und L von 0 kann genau durch Erhöhen von L bei einer aktuellen Differenz von 2 bzw. 3 erreicht werden
 - Eine Differenz zwischen H und L von 0 minimal ist (bei einer Differenz von weniger als 2 wird die Schleife nicht mehr betreten).

Laufzeit des Algorithmus

- In 2. wird $\mathcal{A}(n, c, k)$ etwa $\log(n \cdot m)$ -mal aufgerufen
- In 4. wird $\mathcal{A}(n, c, OPT)$ etwa n^2 -mal aufgerufen.
- Es finden also $\mathcal{O}(n^2 + \log(nm))$ Aufrufe von \mathcal{A} statt.
- Die Inputlänge ist $\mathcal{O}(n^2 \cdot \log(\max c_{ij}))$.
- Da \mathcal{A} polynomial ist, ist dies also auch OPT-TOUR.

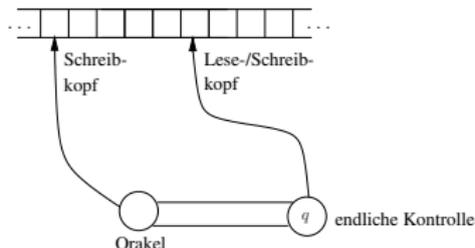
- **Nichtdeterministische Turingmaschinen**
- **Die Klasse NP**

- Bei der nichtdeterministischen Turing-Maschine wird die Übergangsfunktion δ zu einer Relation erweitert.
- Dies ermöglicht Wahlmöglichkeiten und ε -Übergänge (vergleiche endliche Automaten).
- Wir betrachten ein äquivalentes Modell einer nichtdeterministischen Turing-Maschine (NTM), die auf einem **Orakel** basiert
- Dies kommt der Intuition näher.



Nichtdeterministische Turingmaschinen (NTM)

- werden analog zur DTM durch das Oktupel $(Q, \Sigma, \sqcup, \Gamma, s, \delta, q_J, q_N)$ beschrieben.
- haben zusätzlich zu der endlichen Kontrolle mit dem Lese-/Schreibkopf ein **Orakelmodul** mit einem eigenen Schreibkopf
- NTMs haben genau zwei Endzustände q_J und q_N , wobei q_J der akzeptierende Endzustand ist.



Berechnung bei einer nichtdeterministischen Turing-Maschine

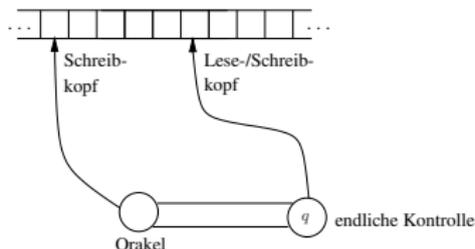
■ 1. Stufe:

- Das Orakelmodul schreibt zunächst ein Trennzeichen in die Zelle direkt vor der Eingabe
- Dann weist es seinen Schreibkopf an, Schritt für Schritt entweder ein Symbol zu schreiben und nach links zu gehen oder anzuhalten.
- Falls der Schreibkopf anhält, wird das Orakelmodul inaktiv, und die endliche Zustandskontrolle wird aktiv.

■ 2. Stufe:

- Ab jetzt genau wie bei DTM.
- Das Orakelmodul und sein Schreibkopf sind nicht weiter beteiligt.

Die Nichtdeterministische Turingmaschine



- Eine nichtdeterministische Turing-Maschine \mathcal{M} **akzeptiert** ein Wort $x \in \Sigma^*$ genau dann, wenn es eine Berechnung gibt, die in q_f endet.
- \mathcal{M} akzeptiert die Sprache $L \subseteq \Sigma^*$ genau dann, wenn sie gerade die Wörter aus L akzeptiert.

Die Eingabe ist ein Wort aus Σ^* , zum Beispiel eine Kodierung eines Problembeispiels $I \in D_{\Pi}$.

- **1. Stufe:** Es wird ein Orakel aus Γ^* berechnet, zum Beispiel ein Lösungsbeispiel für I , also ein Indikator, ob $I \in J_{\Pi}$ oder $I \in N_{\Pi}$ gilt.
- **2. Stufe:** Hier wird nun dieser Lösungsvorschlag überprüft, d.h. es wird geprüft ob $I \in J_{\Pi}$.

Beispiel TSP

- **1. Stufe:** Es wird zum Beispiel eine Permutation σ auf der Knotenmenge V vorgeschlagen. D.h. $(\sigma(1), \dots, \sigma(n))$, $G = (V, E)$, c und k bilden die Eingabe.
- **2. Stufe:** Es wird nun überprüft, ob $\sigma(V)$ eine Tour in G enthält, deren Länge bezüglich c nicht größer als k ist.

- Das Orakel kann ein beliebiges Wort aus Γ^* sein.
- Darum muss in der Überprüfungsphase (2.Stufe) zunächst geprüft werden, ob das Orakel ein zulässiges Lösungsbeispiel ist.
- Ist dies nicht der Fall, so kann die Berechnung zu diesem Zeitpunkt mit der Antwort „Nein“ beendet werden.

- Jede NTM \mathcal{M} hat zu einer gegebenen Eingabe x eine unendliche Anzahl möglicher Berechnungen, eine zu jedem Orakel aus Γ^* .
- Endet mindestens eine in q_J , so wird x akzeptiert.

- Die **Zeit**, die eine nichtdeterministische Turing-Maschine \mathcal{M} benötigt, um ein Wort $x \in L_{\mathcal{M}}$ zu akzeptieren, ist definiert als die minimale Anzahl von Schritten, die \mathcal{M} in den Zustand q_f überführt.
- Die **Zeitkomplexitätsfunktion** $T_{\mathcal{M}}: \mathbb{Z}^+ \rightarrow \mathbb{Z}^+$ einer nichtdeterministischen Turing-Maschine \mathcal{M} ist definiert durch

$$T_{\mathcal{M}}(n) := \max \left(\{1\} \cup \left\{ m \mid \begin{array}{l} \text{es gibt ein } x \in L_{\mathcal{M}} \text{ mit } |x| = n, \text{ so} \\ \text{dass die Zeit, die } \mathcal{M} \text{ benötigt,} \\ \text{um } x \text{ zu akzeptieren, } m \text{ ist} \end{array} \right\} \right)$$

- Die **Zeit**, die eine nichtdeterministische Turing-Maschine \mathcal{M} benötigt, um ein Wort $x \in L_{\mathcal{M}}$ zu akzeptieren, ist definiert als die minimale Anzahl von Schritten, die \mathcal{M} in den Zustand q_f überführt.
- Die **Zeitkomplexitätsfunktion** $T_{\mathcal{M}}: \mathbb{Z}^+ \rightarrow \mathbb{Z}^+$ einer nichtdeterministischen Turing-Maschine \mathcal{M} ist definiert durch

$$T_{\mathcal{M}}(n) := \max \left(\{1\} \cup \left\{ m \mid \begin{array}{l} \text{es gibt ein } x \in L_{\mathcal{M}} \text{ mit } |x| = n, \text{ so} \\ \text{dass die Zeit, die } \mathcal{M} \text{ benötigt,} \\ \text{um } x \text{ zu akzeptieren, } m \text{ ist} \end{array} \right\} \right)$$

Bemerkung 1

- Zur Berechnung von $T_{\mathcal{M}}(n)$ wird für jedes $x \in L_{\mathcal{M}}$ mit $|x| = n$ die kürzeste akzeptierende Berechnung betrachtet.
- Anschließend wird von diesen kürzesten die längste bestimmt.
- Somit ergibt sich eine *worst-case* Abschätzung.

- Die **Zeit**, die eine nichtdeterministische Turing-Maschine \mathcal{M} benötigt, um ein Wort $x \in L_{\mathcal{M}}$ zu akzeptieren, ist definiert als die minimale Anzahl von Schritten, die \mathcal{M} in den Zustand q_f überführt.
- Die **Zeitkomplexitätsfunktion** $T_{\mathcal{M}}: \mathbb{Z}^+ \rightarrow \mathbb{Z}^+$ einer nichtdeterministischen Turing-Maschine \mathcal{M} ist definiert durch

$$T_{\mathcal{M}}(n) := \max \left(\{1\} \cup \left\{ m \mid \begin{array}{l} \text{es gibt ein } x \in L_{\mathcal{M}} \text{ mit } |x| = n, \text{ so} \\ \text{dass die Zeit, die } \mathcal{M} \text{ benötigt,} \\ \text{um } x \text{ zu akzeptieren, } m \text{ ist} \end{array} \right\} \right)$$

Bemerkung 2

- Die Zeitkomplexität hängt nur von der Anzahl der Schritte ab, die bei einer akzeptierenden Berechnung auftreten.
- Hierbei umfasst die Anzahl der Schritte auch die Schritte der Orakelphase.
- Per Konvention ist $T_{\mathcal{M}}(n) = 1$, falls es keine Eingabe x der Länge n gibt, die von \mathcal{M} akzeptiert wird.

Die Klasse NP

Die Klasse \mathcal{NP} ist die Menge aller Sprachen L , für die es eine nichtdeterministische Turing-Maschine gibt, deren Zeitkomplexitätsfunktion polynomial beschränkt ist.

(\mathcal{NP} steht für **nichtdeterministisch polynomial**.)

Die Klasse \mathcal{NP} ist die Menge aller Sprachen L , für die es eine nichtdeterministische Turing-Maschine gibt, deren Zeitkomplexitätsfunktion polynomial beschränkt ist.

(\mathcal{NP} steht für **nichtdeterministisch polynomial**.)

Bemerkungen

- Alle Sprachen in \mathcal{NP} sind entscheidbar.
- Informell ausgedrückt: Π gehört zu \mathcal{NP} , falls Π folgende Eigenschaft hat: Ist die Antwort bei Eingabe eines Beispiels I von Π Ja, dann kann die Korrektheit der Antwort in polynomialer Zeit überprüft werden.

Die Klasse \mathcal{NP} ist die Menge aller Sprachen L , für die es eine nichtdeterministische Turing-Maschine gibt, deren Zeitkomplexitätsfunktion polynomial beschränkt ist.

(\mathcal{NP} steht für **nichtdeterministisch polynomial**.)

Beispiel: $\text{TSP} \in \mathcal{NP}$:

Denn zu gegebenem $G = (V, E)$, c, k und einer festen Permutation σ auf V kann man in $O(|V| \cdot \log C)$, wobei C die größte vorkommende Zahl ist, überprüft werden, ob

$$\sum_{i=1}^{n-1} c(\{\sigma(i), \sigma(i+1)\}) + c(\{\sigma(n), \sigma(1)\}) \leq k$$

gilt.

■ NP-vollständige Probleme

- Trivialerweise gilt: $\mathcal{P} \subseteq \mathcal{NP}$.
- **Frage:** Gilt $\mathcal{P} \subset \mathcal{NP}$ oder $\mathcal{P} = \mathcal{NP}$?
- Die Vermutung ist, dass $\mathcal{P} \neq \mathcal{NP}$ gilt.
- Dazu betrachten wir Probleme, die zu den schwersten Problemen in \mathcal{NP} gehören.
- Dabei ist am schwersten im folgenden Sinne gemeint:
- Wenn ein solches Problem trotzdem in \mathcal{P} liegt, so kann man folgern, dass alle Probleme aus \mathcal{NP} in \mathcal{P} liegen, d.h. $\mathcal{P} = \mathcal{NP}$.
- Diese Probleme sind also Kandidaten, um \mathcal{P} und \mathcal{NP} zu trennen.
- Es wird sich zeigen, dass alle diese schwersten Probleme im wesentlichen gleich schwer sind.

Eine **polynomiale Transformation** einer Sprache $L_1 \subseteq \Sigma_1^*$ in eine Sprache $L_2 \subseteq \Sigma_2^*$ ist eine Funktion $f: \Sigma_1^* \rightarrow \Sigma_2^*$ mit den Eigenschaften:

- es existiert eine polynomiale deterministische Turing-Maschine, die f berechnet;
- für alle $x \in \Sigma_1^*$ gilt: $x \in L_1 \Leftrightarrow f(x) \in L_2$.

Wir schreiben dann $L_1 \propto L_2$ (L_1 ist polynomial transformierbar in L_2).

Eine Sprache L heißt **\mathcal{NP} -vollständig**, falls gilt:

- $L \in \mathcal{NP}$ und
- für alle $L' \in \mathcal{NP}$ gilt $L' \propto L$.

Wir formulieren nun die Begriffe polynomial transformierbar und \mathcal{NP} -vollständig für Entscheidungsprobleme.

Ein Entscheidungsproblem Π_1 ist **polynomial transformierbar** in das Entscheidungsproblem Π_2 , wenn eine Funktion $f: D_{\Pi_1} \rightarrow D_{\Pi_2}$ existiert mit folgenden Eigenschaften:

- f ist durch einen polynomialen Algorithmus berechenbar;
- $\forall I \in D_{\Pi_1}: I \in J_{\Pi_1} \iff f(I) \in J_{\Pi_2}$.

Wir schreiben dann $\Pi_1 \propto \Pi_2$.

Ein Entscheidungsproblem Π heißt **\mathcal{NP} -vollständig**, falls gilt:

- $\Pi \in \mathcal{NP}$ und
- für alle $\Pi' \in \mathcal{NP}$ gilt $\Pi' \propto \Pi$.

Eine **polynomiale Transformation** einer Sprache $L_1 \subseteq \Sigma_1^*$ in eine Sprache $L_2 \subseteq \Sigma_2^*$ ist eine Funktion $f: \Sigma_1^* \rightarrow \Sigma_2^*$ mit den Eigenschaften:

- es existiert eine polynomiale deterministische Turing-Maschine, die f berechnet;
- für alle $x \in \Sigma_1^*$ gilt: $x \in L_1 \Leftrightarrow f(x) \in L_2$.

Wir schreiben dann $L_1 \propto L_2$ (L_1 ist polynomial transformierbar in L_2).

Lemma. \propto ist transitiv, d.h. aus $L_1 \propto L_2$ und $L_2 \propto L_3$ folgt $L_1 \propto L_3$.

Beweis. Die Hintereinanderausführung zweier polynomialer Transformationen ist wieder eine polynomiale Transformation.

Korollar. Falls $L_1, L_2 \in \mathcal{NP}$, $L_1 \propto L_2$ und L_1 \mathcal{NP} -vollständig, dann ist auch L_2 \mathcal{NP} -vollständig.

Bedeutung.

Um also zu zeigen, dass ein Entscheidungsproblem Π \mathcal{NP} -vollständig ist, gehen wir folgendermaßen vor. Wir beweisen:

- $\Pi \in \mathcal{NP}$
- für ein bekanntes \mathcal{NP} -vollständiges Problem Π' gilt: $\Pi' \propto \Pi$.

Problem.

- Wir wissen noch für kein einziges Problem, dass es \mathcal{NP} -vollständig ist.
- Das erste \mathcal{NP} -vollständige Problem ist das Erfüllbarkeitsproblem SAT (satisfiability).

Das Problem SAT (satisfiability)

Sei $U = \{u_1, \dots, u_m\}$ eine Menge von booleschen Variablen
Es heißen u_j, \bar{u}_j Literale.

Eine Wahrheitsbelegung für U ist eine Funktion $t: U \rightarrow \{\text{wahr}, \text{falsch}\}$.

Eine **Klausel** ist ein Boole'scher Ausdruck der Form

$$y_1 \vee \dots \vee y_s \quad \text{mit} \quad y_i \in \underbrace{\{u_1, \dots, u_m\} \cup \{\bar{u}_1, \dots, \bar{u}_m\}}_{\text{Literalmenge}} \cup \{\text{wahr}, \text{falsch}\}$$

Problem SAT

Gegeben: Menge U von Variablen, Menge C von Klauseln über U .

Frage: Existiert eine Wahrheitsbelegung von U , so dass C erfüllt wird, d.h. dass alle Klauseln aus C den Wahrheitswert `wahr` annehmen?

Beispiel: $U = \{u_1, u_2\}$ mit $C = \{u_1 \vee \bar{u}_2, \bar{u}_1 \vee u_2\}$ ist Ja-Beispiel von SAT. Mit der Wahrheitsbelegung $t(u_1) = t(u_2) = \text{wahr}$ wird C erfüllt.

Lösbar:

$$U = \{a, b, c, d, e\}, C = \{c \vee \bar{d}, \bar{a} \vee b \vee \bar{c} \vee d \vee e, \bar{c} \vee d\}$$

Nicht lösbar:

$$U = \{a, b, c\}, C = \{a \vee b, \bar{a}, \bar{b} \vee c, \bar{c}\}$$

Der Satz von Cook (Steven Cook, 1971)

SAT ist \mathcal{NP} -vollständig.

SAT ist \mathcal{NP} -vollständig.

Beweis:

- SAT $\in \mathcal{NP}$ ist erfüllt:
Für ein Beispiel I von SAT (mit n Klauseln und m Variablen) und einer Wahrheitsbelegung t kann in $O(m \cdot n)$ überprüft werden, ob t alle Klauseln erfüllt, d.h. ob I ein Ja-Beispiel ist.
- Wir müssen zeigen, dass für jede Sprache $L \in \mathcal{NP}$ gilt: $L \propto L_{\text{SAT}}$, wobei $L_{\text{SAT}} = L[\text{SAT}, s]$ für ein geeignetes Kodierungsschema s ist.

Wir müssen zeigen, dass für jede Sprache $L \in \mathcal{NP}$ gilt: $L \propto L_{\text{SAT}}$, wobei $L_{\text{SAT}} = L[\text{SAT}, s]$ für ein geeignetes Kodierungsschema s ist.

- Dazu muss für alle Sprachen $L \in \mathcal{NP}$ eine polynomiale Transformation f_L angegeben werden, für die gilt, dass für alle $x \in \Sigma^*$ (Σ Alphabet zu L) gilt

$$x \in L \iff f_L(x) \in L_{\text{SAT}}.$$

- Wir benutzen, dass es eine NDTM \mathcal{M} zu L gibt, die L in polynomialer Laufzeit erkennt.
- \mathcal{M} sei gegeben durch $(Q, \Sigma, \sqcup, \Gamma, q_0, \delta, q_J, q_N)$ und akzeptiere die Sprache $L = L_{\mathcal{M}}$ in der Laufzeit $T_{\mathcal{M}} \leq p(n)$, wobei p ein Polynom ist. O.B.d.A. gilt $p(n) \geq n$.

- Sei x eine Instanz und $n := |x|$
- Bei einer akzeptierenden Berechnung von \mathcal{M} für $x \in \Sigma^*$ ist die Anzahl der Berechnungsschritte beschränkt durch $p(n)$.
- An einer so beschränkten Berechnung können höchstens die Zellen $-p(n)$ bis $p(n) + 1$ des Bandes beteiligt sein.

Der Zustand der deterministischen Stufe ist zu jedem Zeitpunkt eindeutig festgelegt durch:

- den jeweiligen Bandinhalt dieser $-p(n)$ bis $p(n) + 1$ Plätze,
- den Zustand der endlichen Kontrolle
- und der Position des Lese-/Schreibkopfs.

Im folgenden beschreiben wir eine Berechnung vollständig durch Variablen

Bezeichne

- die Zustände aus Q durch $q_0, q_1 = q_J, q_2 = q_N, q_3, \dots, q_r$
- die Symbole aus Γ durch $s_0 = \sqcup, s_1, \dots, s_\ell$ mit $|\Gamma| = \ell + 1$.

Es gibt drei Typen von Variablen in dem zugehörigen Problem SAT

Variable	Gültigkeitsbereich	Bedeutung
$Q[i, k]$	$0 \leq i \leq p(n)$ $0 \leq k \leq r$	zum Zeitpunkt i der Überprüfungsphase ist \mathcal{M} in Zustand q_k
$H[i, j]$	$0 \leq i \leq p(n)$ $-p(n) \leq j \leq p(n) + 1$	zum Zeitpunkt i der Überprüfungsphase ist der Lese-/Schreibkopf an Position j des Bandes
$S[i, j, k]$	$0 \leq i \leq p(n)$ $-p(n) \leq j \leq p(n) + 1$ $0 \leq k \leq \ell$	zum Zeitpunkt i der Überprüfungsphase ist der Bandinhalt an Position j das Symbol s_k

Variable	Gültigkeitsbereich	Bedeutung
$Q[i, k]$	$0 \leq i \leq p(n)$ $0 \leq k \leq r$	zum Zeitpunkt i der Überprüfungsphase ist \mathcal{M} in Zustand q_k
$H[i, j]$	$0 \leq i \leq p(n)$ $-p(n) \leq j \leq p(n) + 1$	zum Zeitpunkt i der Überprüfungsphase ist der Lese-/Schreibkopf an Position j des Bandes
$S[i, j, k]$	$0 \leq i \leq p(n)$ $-p(n) \leq j \leq p(n) + 1$ $0 \leq k \leq \ell$	zum Zeitpunkt i der Überprüfungsphase ist der Bandinhalt an Position j das Symbol s_k

- Eine Berechnung von \mathcal{M} induziert in kanonischer Weise eine Wahrheitsbelegung dieser Variablen.
- Wir benutzen folgende Konvention:
- Falls \mathcal{M} vor dem Zeitpunkt $p(n)$ stoppt, bleibt \mathcal{M} in allen folgenden Zuständen in demselben Zustand und der Bandinhalt unverändert.

Beweis: Konstruktion der Variablen

Variable	Gültigkeitsbereich	Bedeutung
$Q[i, k]$	$0 \leq i \leq p(n)$ $0 \leq k \leq r$	zum Zeitpunkt i der Überprüfungsphase ist \mathcal{M} in Zustand q_k
$H[i, j]$	$0 \leq i \leq p(n)$ $-p(n) \leq j \leq p(n) + 1$	zum Zeitpunkt i der Überprüfungsphase ist der Lese-/Schreibkopf an Position j des Bandes
$S[i, j, k]$	$0 \leq i \leq p(n)$ $-p(n) \leq j \leq p(n) + 1$ $0 \leq k \leq \ell$	zum Zeitpunkt i der Überprüfungsphase ist der Bandinhalt an Position j das Symbol s_k

Der Bandinhalt zum Zeitpunkt 0 der Überprüfungsphase sei

- Eingabe x auf Platz 1 bis n
- Orakel w auf Platz -1 bis $-|w|$
- ansonsten Blanks.

- Eine beliebige Wahrheitsbelegung muss nicht notwendigerweise eine Berechnung induzieren (zum Beispiel $Q[i, k] = Q[i, \ell]$ für $k \neq \ell$).

Also konstruiere Transformation f_L die Klauseln einführt, so dass äquivalent ist:

- Für Eingabe x gibt es eine akzeptierende Berechnung, deren Überprüfungsphase höchstens $p(n)$ Zeit benötigt, und deren Orakel höchstens Länge $p(n)$ hat.
- Es gibt eine erfüllende Belegung für die SAT-Instanz $f_L(x)$.

Also konstruiere Transformation f_L die Klauseln einführt, so dass äquivalent ist:

- Für Eingabe x gibt es eine akzeptierende Berechnung, deren Überprüfungsphase höchstens $p(n)$ Zeit benötigt, und deren Orakel höchstens Länge $p(n)$ hat.
- Es gibt eine erfüllende Belegung für die SAT-Instanz $f_L(x)$.

Damit können wir dann schließen:

- $x \in L \Leftrightarrow$ es existiert eine akzeptierende Berechnung von \mathcal{M} bei Eingabe x
- \Leftrightarrow es existiert eine akzeptierende Berechnung von \mathcal{M} bei Eingabe x mit höchstens $p(n)$ Schritten in der Überprüfungsphase und einem Orakel w der Länge $|w| = p(n)$
- \Leftrightarrow es existiert eine erfüllende Wahrheitsbelegung für die Klauselmenge $f_L(x)$

Konvention:

- Die Bewegungsrichtung des Kopfes sei $d \in \{-1, 0, 1\}$.

Klausel-
gruppe

Einschränkung / Bedeutung

- G_1 Zum Zeitpunkt i ist \mathcal{M} in genau einem Zustand.
- G_2 Zum Zeitpunkt i hat der Lese-/Schreibkopf genau eine Position.
- G_3 Zum Zeitpunkt i enthält jede Bandstelle genau ein Symbol aus Γ .
- G_4 Festlegung der Anfangskonfiguration zum Zeitpunkt 0: \mathcal{M} ist im Zustand q_0 , der Lese-/Schreibkopf steht an Position 1 des Bandes; in den Zellen 1 bis n steht das Wort $x = s_{k_1} \dots s_{k_n}$
- G_5 Bis zum Zeitpunkt $p(n)$ hat \mathcal{M} den Zustand q_J erreicht.
- G_6 Zu jedem Zeitpunkt i folgt die Konfiguration von \mathcal{M} zum Zeitpunkt $i + 1$ aus einer einzigen Anwendung von δ aus der Konfiguration von \mathcal{M} zum Zeitpunkt i .

Zum Zeitpunkt i ist \mathcal{M} in genau einem Zustand.

Konstruktion:

- Zu jedem Zeitpunkt i ist \mathcal{M} in mindestens einem Zustand

$$Q[i, 0] \vee \dots \vee Q[i, r] \quad \text{für } 0 \leq i \leq p(n)$$

- Zu jedem Zeitpunkt i ist \mathcal{M} in nicht mehr als einem Zustand

$$\overline{Q[i, j]} \vee \overline{Q[i, j']} \quad \text{für } 0 \leq i \leq p(n), 0 \leq j < j' \leq r$$

,

Zum Zeitpunkt i hat der Lese-/Schreibkopf genau eine Position

Konstruktion:

- Zum Zeitpunkt i hat der Lese-/Schreibkopf mindestens eine Position

$$H[i, -p(n)] \vee \dots \vee H[i, p(n) + 1] \quad \text{für } 0 \leq i \leq p(n)$$

- Zum Zeitpunkt i hat der Lese-/Schreibkopf höchstens eine Position

$$\overline{H[i, j]} \vee \overline{H[i, j']} \quad \text{für } 0 \leq i \leq p(n) \text{ und } -p(n) \leq j < j' \leq p(n) + 1$$

Zum Zeitpunkt i enthält jede Bandstelle genau ein Symbol

Konstruktion:

- Zum Zeitpunkt i enthält jede Bandstelle mindestens ein Symbol

$$S[i, j, 0] \vee S[i, j, 1] \vee \dots \vee S[i, j, \ell] \quad \text{für} \quad \begin{cases} 0 \leq i \leq p(n) \\ -p(n) \leq j \leq p(n) + 1 \end{cases}$$

- Zum Zeitpunkt i enthält jede Bandstelle höchstens ein Symbol

$$\overline{S[i, j, k]} \vee \overline{S[i, j, k']} \quad \text{für} \quad \begin{cases} 0 \leq i \leq p(n) \\ -p(n) \leq j \leq p(n) + 1 \\ 0 \leq k < k' \leq \ell \end{cases}$$

Festlegung der Anfangskonfiguration zum Zeitpunkt 0

Konstruktion:

- \mathcal{M} ist im Zustand q_0

$$Q[0, 0]$$

- der Lese-/Schreibkopf steht an Position 1 des Bandes

$$H[0, 1]$$

- in den Zellen 1 bis n steht das Wort $x = s_{k_1} \dots s_{k_n}$

$$\begin{cases} S[0, 0, 0], S[0, 1, k_1], \dots, S[0, n, k_n] & , \text{ für Eingabe } x = s_{k_1} \dots s_{k_n} \\ S[0, n+1, 0], \dots, S[0, p(n)+1, 0] & , \text{ alle anderen Positionen} \end{cases}$$

Klauselgruppe 5:

Bis zum Zeitpunkt $p(n)$ hat \mathcal{M} den Zustand q_J erreicht.

Konstruktion:

$$Q[p(n), 1]$$

Zu jedem Zeitpunkt i folgt die Konfiguration von \mathcal{M} zum Zeitpunkt $i + 1$ aus einer einzigen Anwendung von δ aus der Konfiguration von \mathcal{M} zum Zeitpunkt i .

Wir unterteilen Klauselgruppe G_6 in zwei Teilgruppen $G_{6,1}$, $G_{6,2}$.

- $G_{6,1}$: Falls \mathcal{M} zum Zeitpunkt i an der Position j das Symbol s_k hat und der Lese-/Schreibkopf nicht an der Position j steht, dann hat \mathcal{M} auch zum Zeitpunkt $i + 1$ an Position j das Symbol s_k für $0 \leq i < p(n)$.
- $G_{6,2}$: Der Wechsel von einer Konfiguration zur nächsten entspricht tatsächlich δ .

Falls \mathcal{M} zum Zeitpunkt i an der Position j das Symbol s_k hat und der Lese-/Schreibkopf nicht an der Position j steht, dann hat \mathcal{M} auch zum Zeitpunkt $i + 1$ an Position j das Symbol s_k für $0 \leq i < p(n)$.

Konstruktion:

$$\left(\left(S[i, j, k] \wedge \overline{H[i, j]} \right) \implies S[i + 1, j, k] \right)$$

Dies ergibt die Klausel

$$\left(\overline{S[i, j, k]} \vee H[i, j] \vee S[i + 1, j, k] \right)$$

Der Wechsel von einer Konfiguration zur nächsten entspricht tatsächlich δ .

- Sei $\delta(q_k, s_m) = (q_\kappa, s_\mu, d)$.
- \exists sei q_k aus $Q \setminus \{q_J, q_N\}$ sonst gilt $q_\kappa = q_k, s_\mu = s_m$ und $d = 0$.
 $(H[i, j] \wedge Q[i, k] \wedge S[i, j, m]) \Rightarrow H[i + 1, j + d]$
und $(H[i, j] \wedge Q[i, k] \wedge S[i, j, m]) \Rightarrow Q[i + 1, \kappa]$
und $(H[i, j] \wedge Q[i, k] \wedge S[i, j, m]) \Rightarrow S[i + 1, j, \mu]$

Dies ergibt folgende Klauseln

$$\overline{H[i, j]} \vee \overline{Q[i, k]} \vee \overline{S[i, j, m]} \vee H[i + 1, j + d]$$

$$\overline{H[i, j]} \vee \overline{Q[i, k]} \vee \overline{S[i, j, m]} \vee Q[i + 1, \kappa]$$

$$\overline{H[i, j]} \vee \overline{Q[i, k]} \vee \overline{S[i, j, m]} \vee S[i + 1, j, \mu]$$

für $0 \leq i < p(n), -p(n) \leq j \leq p(n) + 1, 0 \leq k \leq r, 0 \leq m \leq \ell$

- Durch f_L wird nun ein Input (\mathcal{M}, x) auf die Klauselmenge

$$G := G_1 \wedge G_2 \wedge \dots \wedge G_6$$

abgebildet.

- Wenn $x \in L$, dann ist G erfüllbar.
- Eine erfüllende Wahrheitsbelegung der Variablen aus G induziert eine akzeptierende Berechnung von \mathcal{M} für die Eingabe $x \in L$.

Polynomialität der Transformation

Wir schätzen die Anzahl der Literale in den Klauselgruppen ab.

Zum Zeitpunkt i ist \mathcal{M} in genau einem Zustand.

Konstruktion:

- Zu jedem Zeitpunkt i ist \mathcal{M} in mindestens einem Zustand

$$Q[i, 0] \vee \dots \vee Q[i, r] \quad \text{für } 0 \leq i \leq p(n)$$

- Zu jedem Zeitpunkt i ist \mathcal{M} in nicht mehr als einem Zustand

$$\overline{Q[i, j]} \vee \overline{Q[i, j']} \quad \text{für } 0 \leq i \leq p(n), 0 \leq j < j' \leq r$$

,

Abschätzung:

$$(p(n) + 1)(r + 1) + (p(n) + 1) \frac{1}{2}(r(r + 1))$$

Zum Zeitpunkt i hat der Lese-/Schreibkopf genau eine Position

Konstruktion:

- Zum Zeitpunkt i hat der Lese-/Schreibkopf mindestens eine Position

$$H[i, -p(n)] \vee \dots \vee H[i, p(n) + 1] \quad \text{für } 0 \leq i \leq p(n)$$

- Zum Zeitpunkt i hat der Lese-/Schreibkopf höchstens eine Position

$$\overline{H[i, j]} \vee \overline{H[i, j']} \quad \text{für } 0 \leq i \leq p(n) \text{ und } -p(n) \leq j < j' \leq p(n) + 1$$

Abschätzung:

$$(p(n) + 1)(2p(n) + 1) + (p(n) + 1) \frac{1}{2} (2p(n) \cdot (2p(n) + 1))$$

Zum Zeitpunkt i enthält jede Bandstelle genau ein Symbol

Konstruktion:

- Zum Zeitpunkt i enthält jede Bandstelle mindestens ein Symbol

$$S[i, j, 0] \vee S[i, j, 1] \vee \dots \vee S[i, j, \ell] \quad \text{für} \quad \begin{cases} 0 \leq i \leq p(n) \\ -p(n) \leq j \leq p(n) + 1 \end{cases}$$

- Zum Zeitpunkt i enthält jede Bandstelle höchstens ein Symbol

$$\overline{S[i, j, k]} \vee \overline{S[i, j, k']} \quad \text{für} \quad \begin{cases} 0 \leq i \leq p(n) \\ -p(n) \leq j \leq p(n) + 1 \\ 0 \leq k < k' \leq \ell \end{cases}$$

Abschätzung:

$$(p(n) + 1)(2p(n) + 1)(\ell + 1) + (p(n) + 1)(2p(n) + 1)\frac{1}{2}(\ell(\ell + 1))$$

Festlegung der Anfangskonfiguration zum Zeitpunkt 0

- \mathcal{M} ist im Zustand q_0

$$Q[0, 0]$$

- der Lese-/Schreibkopf steht an Position 1 des Bandes

$$H[0, 1]$$

- in den Zellen 1 bis n steht das Wort $x = s_{k_1} \dots s_{k_n}$

$$\begin{cases} S[0, 0, 0], S[0, 1, k_1], \dots, S[0, n, k_n] & , \text{ für Eingabe } x = s_{k_1} \dots s_{k_n} \\ S[0, n+1, 0], \dots, S[0, p(n)+1, 0] & , \text{ alle anderen Positionen} \end{cases}$$

Abschätzung:

$$2 + (n + 1) + (p(n) + 2 - (n + 1)) = p(n) + 4$$

Polynomialität - Klauselgruppe 5:

Bis zum Zeitpunkt $p(n)$ hat \mathcal{M} den Zustand q_J erreicht.

Konstruktion:

$$Q[p(n), 1]$$

Abschätzung:

1

Falls \mathcal{M} zum Zeitpunkt i an der Position j das Symbol s_k hat und der Lese-/Schreibkopf nicht an der Position j steht, dann hat \mathcal{M} auch zum Zeitpunkt $i + 1$ an Position j das Symbol s_k für $0 \leq i < p(n)$.

Konstruktion:

$$\left(\left(S[i, j, k] \wedge \overline{H[i, j]} \right) \implies S[i + 1, j, k] \right)$$

Dies ergibt die Klausel

$$\left(\overline{S[i, j, k]} \vee H[i, j] \vee S[i + 1, j, k] \right)$$

Abschätzung:

$$p(n)(\ell + 1)(2p(n) + 2) \cdot 3$$

Der Wechsel von einer Konfiguration zur nächsten entspricht tatsächlich δ .

- Sei $\delta(q_k, s_m) = (q_\kappa, s_\mu, d)$.
- \exists sei q_κ aus $Q \setminus \{q_J, q_N\}$ sonst gilt $q_\kappa = q_k, s_\mu = s_m$ und $d = 0$.

$$\overline{H[i, j]} \vee \overline{Q[i, k]} \vee \overline{S[i, j, m]} \vee H[i + 1, j + d]$$

$$\overline{H[i, j]} \vee \overline{Q[i, k]} \vee \overline{S[i, j, m]} \vee Q[i + 1, \kappa]$$

$$\overline{H[i, j]} \vee \overline{Q[i, k]} \vee \overline{S[i, j, m]} \vee S[i + 1, j, \mu]$$

für $0 \leq i < p(n), -p(n) \leq j \leq p(n) + 1, 0 \leq k \leq r, 0 \leq m \leq \ell$

Abschätzung:

$$p(n)(p(n) + 2)(r + 1)(\ell + 1) \cdot 3 \cdot 4$$

Wir schätzen die Anzahl der Literale in den Klauselgruppen ab.

- $G_1: (p(n) + 1)(r + 1) + (p(n) + 1)\frac{1}{2}(r(r + 1))$
- $G_2: (p(n) + 1)(2p(n) + 1) + (p(n) + 1)\frac{1}{2}(2p(n) \cdot (2p(n) + 1))$
- $G_3:$
 $(p(n) + 1)(2p(n) + 1)(\ell + 1) + (p(n) + 1)(2p(n) + 1)\frac{1}{2}(\ell(\ell + 1))$
- $G_4: 2 + (n + 1) + (p(n) + 2 - (n + 1)) = p(n) + 4$
- $G_5: 1$
- $G_6: \underbrace{p(n)(\ell + 1)(2p(n) + 2) \cdot 3}_{G_{6,1}} + \underbrace{p(n)(p(n) + 2)(r + 1)(\ell + 1) \cdot 3 \cdot 4}_{G_{6,2}}$

Wir schätzen die Anzahl der Literale in den Klauselgruppen ab.

- $G_1: (p(n) + 1)(r + 1) + (p(n) + 1)\frac{1}{2}(r(r + 1))$
- $G_2: (p(n) + 1)(2p(n) + 1) + (p(n) + 1)\frac{1}{2}(2p(n) \cdot (2p(n) + 1))$
- $G_3:$
 $(p(n) + 1)(2p(n) + 1)(\ell + 1) + (p(n) + 1)(2p(n) + 1)\frac{1}{2}(\ell(\ell + 1))$
- $G_4: 2 + (n + 1) + (p(n) + 2 - (n + 1)) = p(n) + 4$
- $G_5: 1$
- $G_6: \underbrace{p(n)(\ell + 1)(2p(n) + 2) \cdot 3}_{G_{6,1}} + \underbrace{p(n)(p(n) + 2)(r + 1)(\ell + 1) \cdot 3 \cdot 4}_{G_{6,2}}$

- r und ℓ sind Konstanten, die durch \mathcal{M} (und damit durch L) induziert werden
- $p(n)$ ist ein Polynom in n

Wir schätzen die Anzahl der Literale in den Klauselgruppen ab.

- $G_1: (p(n) + 1)(r + 1) + (p(n) + 1)\frac{1}{2}(r(r + 1))$
- $G_2: (p(n) + 1)(2p(n) + 1) + (p(n) + 1)\frac{1}{2}(2p(n) \cdot (2p(n) + 1))$
- $G_3:$
 $(p(n) + 1)(2p(n) + 1)(\ell + 1) + (p(n) + 1)(2p(n) + 1)\frac{1}{2}(\ell(\ell + 1))$
- $G_4: 2 + (n + 1) + (p(n) + 2 - (n + 1)) = p(n) + 4$
- $G_5: 1$
- $G_6: \underbrace{p(n)(\ell + 1)(2p(n) + 2) \cdot 3}_{G_{6,1}} + \underbrace{p(n)(p(n) + 2)(r + 1)(\ell + 1) \cdot 3 \cdot 4}_{G_{6,2}}$

- Also sind alle Größen polynomial in n .
- Die angegebene Funktion f_L ist damit eine polynomiale Transformation von L nach L_{SAT} .

Problem 3-SAT

Gegeben: Menge U von Variablen
Menge C von Klauseln über U
jede Klausel enthält genau *drei* Literale

Frage: Existiert eine erfüllende Wahrheitsbelegung für C ?

Problem 3-SAT

Gegeben: Menge U von Variablen
Menge C von Klauseln über U
jede Klausel enthält genau *drei* Literale

Frage: Existiert eine erfüllende Wahrheitsbelegung für C ?

Satz:
Das Problem 3SAT ist \mathcal{NP} -vollständig.

Beweis: NP-Vollständigkeit von 3-SAT

$3\text{SAT} \in \mathcal{NP}$:

- Für eine feste Wahrheitsbelegung t kann in polynomialer Zeit $O(|C|)$ überprüft werden, ob t alle Klauseln aus C erfüllt.

SAT \propto 3SAT:

- Wir geben eine polynomiale Transformation f von SAT zu 3SAT an.
- Gegeben sei eine SAT-Instanz I

Wir konstruieren eine 3SAT-Instanz $f(I)$ indem wir jede Klausel c in I einzeln auf Klausel(n) $f(c)$ in $f(I)$ abbilden:

- Besteht die Klausel $c = x$ aus **einem** Literal, so wird c auf $x \vee x \vee x$ abgebildet.
- Besteht die Klausel $c = x \vee y$ aus **zwei** Literalen, so wird c auf $x \vee y \vee x$ abgebildet.
- Besteht die Klausel c aus **drei** Literalen, so wird c auf sich selbst abgebildet.

Beweis: NP-Vollständigkeit von 3-SAT

Wir konstruieren eine 3SAT-Instanz $f(I)$ indem wir jede Klausel c in I einzeln auf Klausel(n) $f(c)$ in $f(I)$ abbilden:

- Besteht die Klausel $c = x_1 \vee \dots \vee x_k$ aus $k > 3$ Literalen, bilde c wie folgt ab:
 - Führe $k - 3$ neue Variablen $y_{c,1}, \dots, y_{c,k-3}$ ein.
 - Bilde c auf die folgenden $k - 2$ Klauseln ab:

$$\begin{aligned} & x_1 \vee x_2 \vee y_{c,1} \\ & \overline{y_{c,1}} \vee x_3 \vee y_{c,2} \\ & \vdots \\ & \overline{y_{c,k-4}} \vee x_{k-2} \vee y_{c,k-3} \\ & \overline{y_{c,k-3}} \vee x_{k-1} \vee x_k \end{aligned}$$

- Diese Klauseln lassen sich in Zeit $\mathcal{O}(|C| \cdot |U|)$ konstruieren.

Beweis: NP-Vollständigkeit von 3-SAT

Noch zu zeigen:

- I ist erfüllbar $\Leftrightarrow f(I)$ ist erfüllbar

I ist erfüllbar $\Rightarrow f(I)$ ist erfüllbar

- Sei die SAT-Instanz I erfüllbar
- Wir setzen eine erfüllende Wahrheitsbelegung von I auf $f(I)$ fort
- Wir untersuchen jede Klausel $c = x_1 \vee \dots \vee x_k$ in I einzeln
- Es ist mindestens ein x_j wahr
- Fall $k \leq 3$: Damit ist auch $f(c)$ wahr.
- Fall $k > 3$. Falls $x_1 = \text{wahr}$ oder $x_2 = \text{wahr}$ ist, setze

$$y_{c,j} \equiv \text{falsch}$$

sonst setze, für ein $i > 2$ mit $x_i = \text{wahr}$,

$$y_{c,j} = \begin{cases} \text{wahr} & \text{falls } 1 \leq j \leq i-2 \\ \text{falsch} & \text{falls } i-1 \leq j \leq k-3 \end{cases}$$

- Diese Erweiterung erfüllt alle Klauseln in $f(c)$

Beweis: NP-Vollständigkeit von 3-SAT

I ist erfüllbar $\Leftrightarrow f(I)$ ist erfüllbar

- Wir zeigen: I ist nicht erfüllbar $\Rightarrow f(I)$ ist nicht erfüllbar.
- Sei also die SAT-Instanz I nicht erfüllbar.
- Wir betrachten eine beliebige Belegung der Variablen von $f(I)$
- Da I nicht erfüllbar ist, gibt es eine Klausel $c = x_1 \vee \dots \vee x_k$ in I bei der alle Literale x_i auf falsch gesetzt sind.

■ c wird abgebildet auf

$$\begin{aligned} & x_1 \vee x_2 \vee y_{c,1} \\ & \overline{y_{c,1}} \vee x_3 \vee y_{c,2} \\ & \quad \vdots \\ & \overline{y_{c,k-4}} \vee x_{k-2} \vee y_{c,k-3} \\ & \overline{y_{c,k-3}} \vee x_{k-1} \vee x_k \end{aligned}$$

- Um $f(c)$ zu erfüllen, müßten alle $y_{c,j}$ wahr sein
- Dann ist die letzte Klausel $\overline{y_{c,k-3}} \vee x_{k-1} \vee x_k$ nicht erfüllt.
- Also ist die 3SAT-Instanz $f(I)$ nicht erfüllbar.

Problem 2SAT

Gegeben: Menge U von Variablen
Menge C von Klauseln über U
wobei jede Klausel genau zwei Literale enthält

Frage: Existiert eine erfüllende Wahrheitsbelegung für C ?

Satz:
Das Problem 2SAT liegt in \mathcal{P} .

Beweis: Übung

Problem MAX2SAT

Gegeben: Menge U von Variablen
Menge C von Klauseln über U
wobei jede Klausel genau zwei Literale enthält
Zahl $K \in \mathbb{N}$

Frage: Existiert eine Wahrheitsbelegung, die mindestens K Klauseln erfüllt?

Satz:
Das Problem MAX2SAT ist \mathcal{NP} -vollständig.

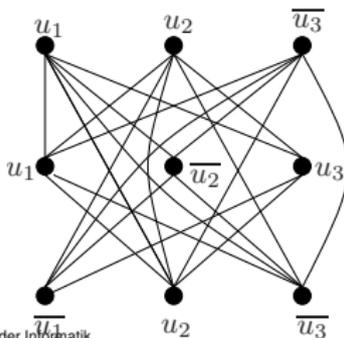
Beweis: Übung

Eine **Clique** in einem Graphen $G = (V, E)$ ist eine Menge $V' \subseteq V$ so, dass für alle $i, j \in V', i \neq j$, gilt: $\{i, j\} \in E$.

Problem CLIQUE

Gegeben: Graph $G = (V, E)$ und ein Parameter $K \leq |V|$

Frage: Gibt es in G eine Clique der Größe mindestens K ?



Beweis: NP-Vollständigkeit von CLIQUE

Satz:

Das Problem CLIQUE ist \mathcal{NP} -vollständig.

CLIQUE $\in \mathcal{NP}$

Beweis: Übung.

Beweis: NP-Vollständigkeit von CLIQUE

3SAT \propto CLIQUE

- Sei $C = \{c_1, \dots, c_n\}$ eine 3SAT-Instanz mit
 $c_i = x_{i1} \vee x_{i2} \vee x_{i3}$ und $x_{ij} \in \{u_1, \dots, u_m, \overline{u_1}, \dots, \overline{u_m}\}$.

Wir transformieren C in eine CLIQUE-Instanz $(G = (V, E), K)$

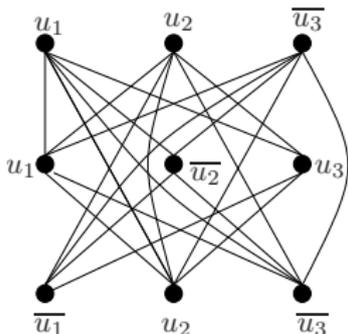
- V enthält $3n$ Knoten v_{ij} für $1 \leq i \leq n$, $1 \leq j \leq 3$
- v_{ij} und v_{kl} sind durch Kanten aus E verbunden genau dann, wenn:
 - $i \neq k$ (Literale sind in verschiedenen Klauseln)
 - $x_{ij} \neq \overline{x_{kl}}$ (Literale sind gleichzeitig erfüllbar)
- Wir setzen $K := n$

Beweis: NP-Vollständigkeit von CLIQUE

- V enthält $3n$ Knoten v_{ij} für $1 \leq i \leq n$, $1 \leq j \leq 3$
- v_{ij} und v_{kl} sind durch Kanten aus E verbunden genau dann, wenn:
 - $i \neq k$ (Literale sind in verschiedenen Klauseln)
 - $x_{ij} \neq \overline{x_{kl}}$ (Literale sind gleichzeitig erfüllbar)

Beispiel: Sei $C = \{u_1 \vee u_2 \vee \overline{u_3}, u_1 \vee \overline{u_2} \vee u_3, \overline{u_1} \vee u_2 \vee \overline{u_3}\}$.

Knotennummer	v_{11}	v_{12}	v_{13}	v_{21}	v_{22}	v_{23}	v_{31}	v_{32}	v_{33}
Literal	u_1	u_2	$\overline{u_3}$	u_1	$\overline{u_2}$	u_3	$\overline{u_1}$	u_2	$\overline{u_3}$.



Beweis: NP-Vollständigkeit von CLIQUE

- Die Transformation kann in polynomieller Zeit berechnet werden.

Noch zu zeigen:

- 3SAT-Instanz C ist erfüllbar \Leftrightarrow CLIQUE-Instanz (G, K) ist erfüllbar

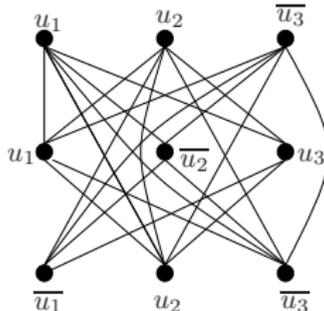
Beweis: NP-Vollständigkeit von CLIQUE

3SAT-Instanz C ist erfüllbar \Rightarrow CLIQUE-Instanz (G, K) ist erfüllbar

- Wähle eine beliebige erfüllende Wahrheitsbelegung von C
- Wähle in jeder Klausel ein wahres Literal
- Die entsprechenden Knoten in G bilden eine Clique der Größe n

Beispiel: Sei $C = \{u_1 \vee u_2 \vee \overline{u_3}, u_1 \vee \overline{u_2} \vee u_3, \overline{u_1} \vee u_2 \vee \overline{u_3}\}$.

Knotennummer	v_{11}	v_{12}	v_{13}	v_{21}	v_{22}	v_{23}	v_{31}	v_{32}	v_{33}
Literal	u_1	u_2	$\overline{u_3}$	u_1	$\overline{u_2}$	u_3	$\overline{u_1}$	u_2	$\overline{u_3}$.



Beweis: NP-Vollständigkeit von CLIQUE

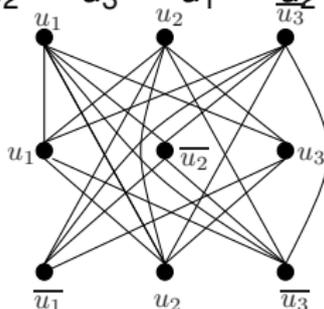
3SAT-Instanz C ist erfüllbar \Leftrightarrow CLIQUE-Instanz (G, K) ist erfüllbar

- Wähle eine Clique V' der Größe n in G
- Die entsprechenden Literale sind
 - gleichzeitig erfüllbar
 - decken alle Klauseln ab

und induzieren deswegen eine erfüllende Wahrheitsbelegung von C .

Beispiel: Sei $C = \{u_1 \vee u_2 \vee \overline{u_3}, u_1 \vee \overline{u_2} \vee u_3, \overline{u_1} \vee u_2 \vee \overline{u_3}\}$.

Knotennummer	V_{11}	V_{12}	V_{13}	V_{21}	V_{22}	V_{23}	V_{31}	V_{32}	V_{33}
Literal	u_1	u_2	$\overline{u_3}$	u_1	$\overline{u_2}$	u_3	$\overline{u_1}$	u_2	$\overline{u_3}$.



Problem COLOR

Gegeben: Graph $G = (V, E)$ und ein Parameter $K \in \mathbb{N}$.

Frage: Gibt es eine Knotenfärbung von G mit höchstens K Farben, so dass je zwei adjazente Knoten verschiedene Farben besitzen?

3COLOR bezeichnet das Problem COLOR mit festem Parameter $k = 3$.

Satz:

Das Problem 3COLOR ist \mathcal{NP} -vollständig.

Beweis: NP-Vollständigkeit von 3COLOR

$3\text{COLOR} \in \mathcal{NP}$

- Es kann in Zeit $\mathcal{O}(|E|)$ überprüft werden, ob eine Färbung von Graph $G = (V, E)$ mit drei Farben zulässig ist.

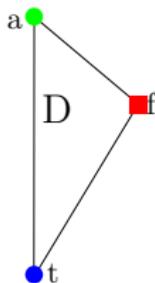
Beweis: NP-Vollständigkeit von 3COLOR

3SAT \propto 3COLOR

- Sei I eine 3SAT-Instanz mit Variablen $U = \{u_1, \dots, u_m\}$ und Klauseln $\{c_1, \dots, c_n\}$
- Wir konstruieren in Polynomialzeit eine 3COLOR-Instanz G
- Es soll gelten: I ist erfüllbar $\Leftrightarrow G$ ist 3-färbbar

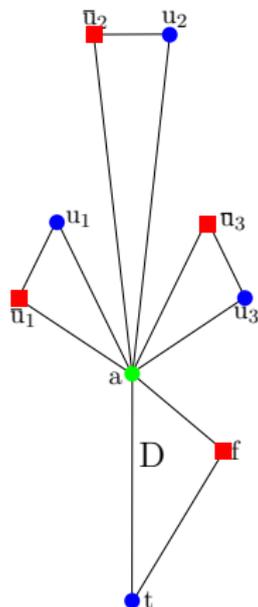
Der Graph G enthält

- Ein 'Hauptdreieck' aus Knoten $\{t,f,a\}$ und Kanten $\{\{t,f\}, \{f,a\}, \{t,a\}\}$
- Interpretation: t,f,a sind die drei Farben mit denen G gefärbt wird
- Interpretation: t : wahr, f : falsch



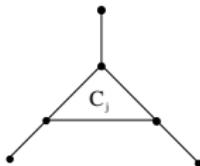
Der Graph G enthält

- Für jede Variable $u \in U$ ein Dreieck D_u mit Eckknoten u, \bar{u}, a
- Interpretation: Falls u mit t gefärbt ist, muss \bar{u} mit f gefärbt sein



Der Graph G enthält

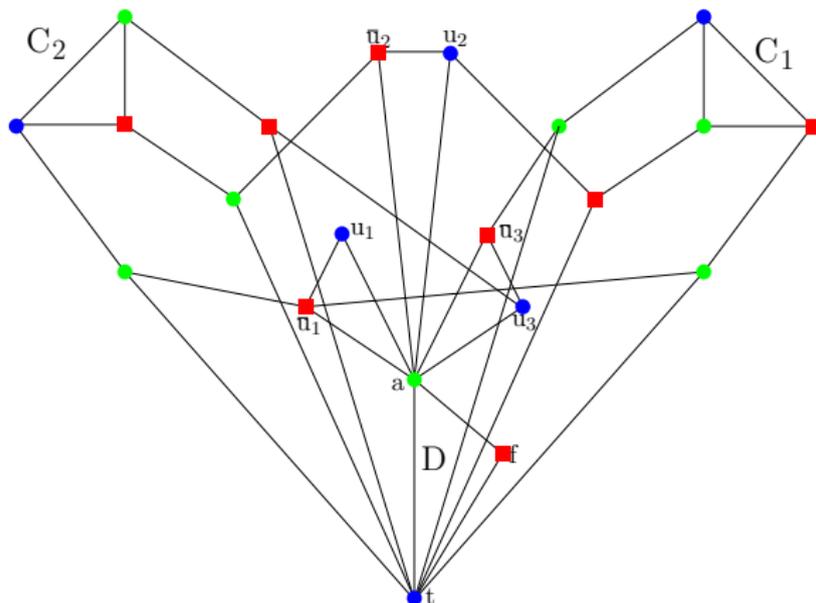
- für jede Klausel $c_j = x \vee y \vee z$ eine Komponente C_j wie folgt
- C_j besteht aus sechs Knoten: einem „inneren Dreieck“ und drei „Satelliten“



- Jeder der drei Satelliten wird mit einem der Literale x, y, z verbunden
- Alle drei Satelliten werden mit dem Eckknoten t in D verbunden.

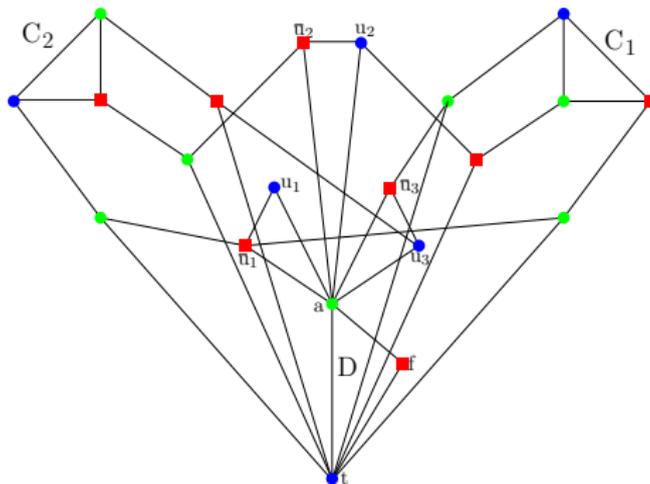
Beispielgraph zur Reduktion

$$c_1 = \bar{u}_1 \vee u_2 \vee \bar{u}_3, c_2 = \bar{u}_1 \vee \bar{u}_2 \vee u_3$$



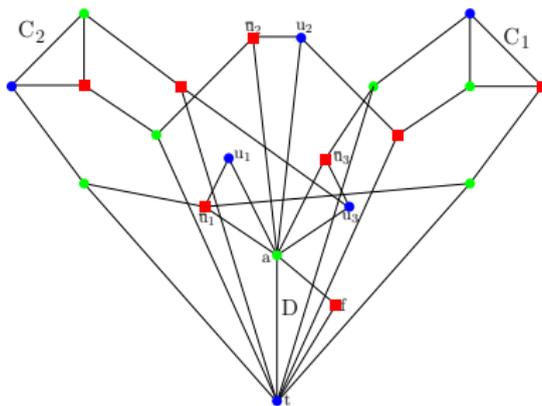
Polynomialität der Reduktion

- Die Knotenanzahl von G liegt in $\mathcal{O}(n + m)$.
- Deswegen ist die Transformation polynomial.



Instanz I erfüllbar \Leftrightarrow Instanz G erfüllbar

- Betrachte Dreifärbung von G
- Färbung von Literal-Knoten induziert eine gültige Wahrheitsbelegung von I



Problem EXACT COVER

Gegeben: Eine endliche Menge X und eine Familie \mathcal{S} von Teilmengen von X .

Frage: Existiert eine Menge $\mathcal{S}' \subseteq \mathcal{S}$, so dass jedes Element aus X in genau einer Menge aus \mathcal{S}' liegt?

Problem EXACT COVER

Gegeben: Eine endliche Menge X und eine Familie \mathcal{S} von Teilmengen von X .

Frage: Existiert eine Menge $\mathcal{S}' \subseteq \mathcal{S}$, so dass jedes Element aus X in genau einer Menge aus \mathcal{S}' liegt?

Beispiel:

$$X = \{1, 2, \dots, 7\}$$

$$\mathcal{S} = \{\{1, 2, 3\}, \{1, 2, 4\}, \{2, 3, 4\}, \{1, 3, 4\}, \{1, 5\}, \{3, 5\}, \{1, 3\}, \\ \{5, 6, 7\}, \{4, 5, 6\}, \{4, 5, 7\}, \{4, 6, 7\}, \{5, 6\}, \{5, 7\}, \{6, 7\}\}$$

Ist (X, \mathcal{S}) eine Ja-Instanz?

Problem EXACT COVER

Gegeben: Eine endliche Menge X und eine Familie \mathcal{S} von Teilmengen von X .

Frage: Existiert eine Menge $\mathcal{S}' \subseteq \mathcal{S}$, so dass jedes Element aus X in genau einer Menge aus \mathcal{S}' liegt?

Beispiel:

$$X = \{1, 2, \dots, 7\}$$

$$\mathcal{S} = \{\{1, 2, 3\}, \{1, 2, 4\}, \{2, 3, 4\}, \{1, 3, 4\}, \{1, 5\}, \{3, 5\}, \{1, 3\}, \\ \{5, 6, 7\}, \{4, 5, 6\}, \{4, 5, 7\}, \{4, 6, 7\}, \{5, 6\}, \{5, 7\}, \{6, 7\}\}$$

$$\mathcal{S}' = \{\{1, 5\}, \{2, 3, 4\}, \{6, 7\}\}$$

Ja

Problem EXACT COVER

Gegeben: Eine endliche Menge X und eine Familie \mathcal{S} von Teilmengen von X .

Frage: Existiert eine Menge $\mathcal{S}' \subseteq \mathcal{S}$, so dass jedes Element aus X in genau einer Menge aus \mathcal{S}' liegt?

Satz:

Problem EXACT COVER ist \mathcal{NP} -vollständig.

Beweis: NP-Vollständigkeit von EXACT COVER

EXACT COVER $\in \mathcal{NP}$

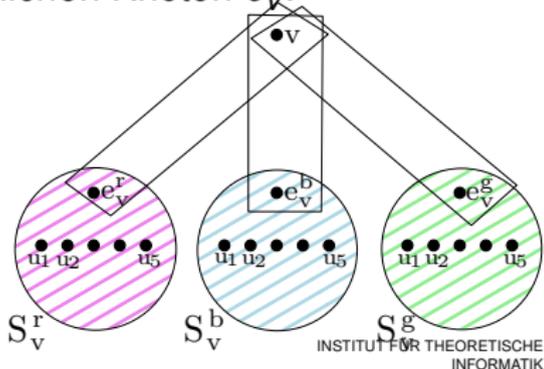
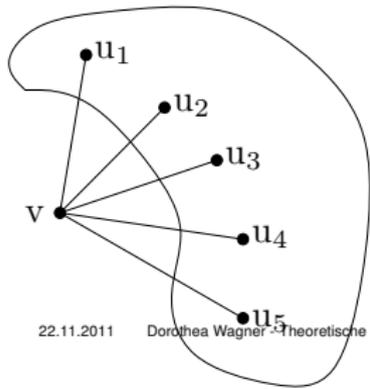
- Es kann in Polynomialzeit überprüft werden, ob eine Teilmenge $\mathcal{S}' \subseteq \mathcal{S}$ aus disjunkten Mengen besteht und X überdeckt.

3COLOR \propto EXACT COVER

- Sei $G = (V, E)$ eine 3COLOR-Instanz
- Wir konstruieren in Polynomialzeit eine EXACT COVER-Instanz (X, S)
- Es soll gelten: G ist 3-färbbar $\Leftrightarrow (X, S)$ ist erfüllbar

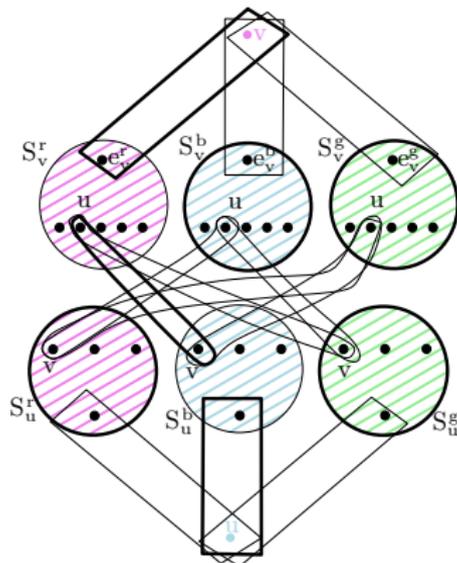
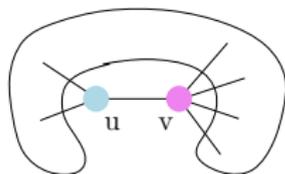
Konstruktion von (X, S)

- Sei $C = \{r(ot), b(lau), g(rün)\}$
- Sei $N(v) := \{u \in V : \{u, v\} \in E\}$ die Nachbarschaft von v .
- Für jedes $v \in V$ enthalte X ein „Element“ v und jeweils $3 \cdot |N(v)| + 3$ zusätzliche Elemente.
- Zu jedem $v \in V$ gebe es in S drei disjunkte Mengen S_v^r, S_v^b, S_v^g mit jeweils $|N(v)| + 1$ Elementen.
- Außerdem enthalte S für jedes v drei zweielementige Mengen $\{v, e_v^r\}, \{v, e_v^b\}$ und $\{v, e_v^g\}$ mit $e_v^r \in S_v^r, e_v^b \in S_v^b$ und $e_v^g \in S_v^g$.
- **Interpretation:** S_v^r entspricht der „Farbe“ r , enthält für jeden Knoten aus $N(v)$ eine Kopie und einen zusätzlichen Knoten e_v^r .



Konstruktion von (X,S)

- Außerdem enthält S für jede Kante $\{u, v\} \in E$ und je zwei $c, c' \in C$, $c \neq c'$, die zweielementigen Mengen $\{u_v^c, v_u^{c'}\}$, $u_v^c \in S_v^c$ „Kopie“ von u , $v_u^{c'} \in S_u^{c'}$ „Kopie“ von v .



Konstruktion von (X, S)

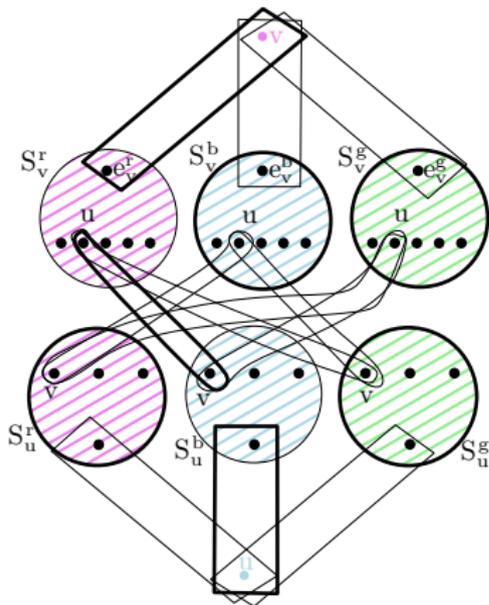
- Die Konstruktion ist polynomial.

Noch zu zeigen:

- G ist 3-färbbar $\Leftrightarrow (X, S)$ ist erfüllbar

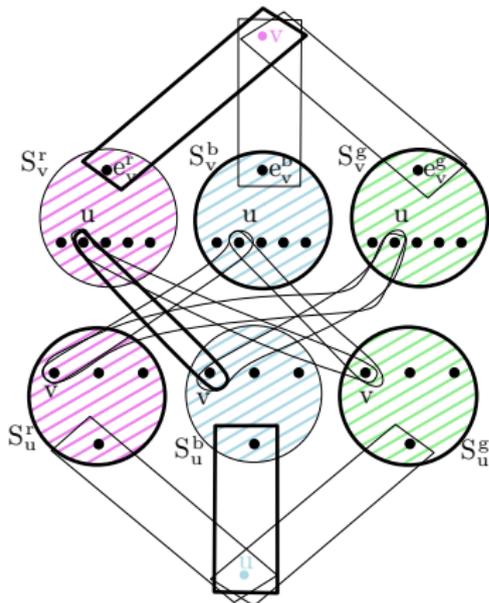
G dreifärbbar $\Rightarrow (X,S)$ hat exakte Überdeckung

- Sei $\chi : V \rightarrow C$ eine zulässige Dreifärbung.
- S' enthalte für jedes $v \in V$ die Mengen $\{v, e_v^{\chi(v)}\}$ und S_v^c mit $c \neq \chi(v)$.
- Diese Mengen überdecken alle Elemente exakt, außer den Elementen der Form $u_v^{\chi(v)}, v_u^{\chi(u)}$ für $\{u, v\} \in E$.
- Daher enthalte S' für jede Kante $\{u, v\} \in E$ die Menge $\{u_v^{\chi(v)}, v_u^{\chi(u)}\}$.
- Diese Menge existiert, da $\chi(u) \neq \chi(v)$, und damit überdeckt S' jedes Element aus X genau einmal.



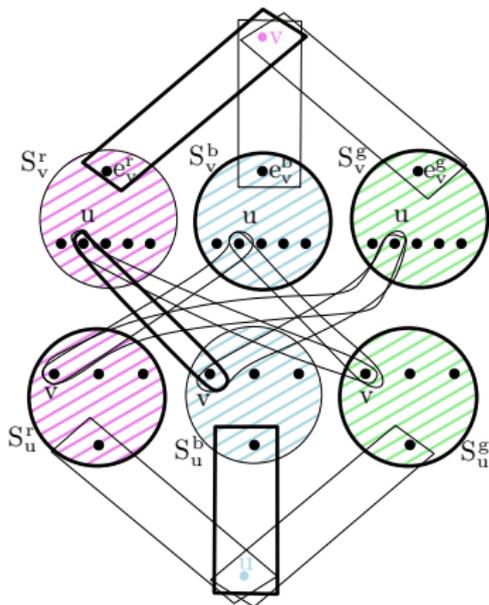
G dreifärbbar $\Leftrightarrow (X, S)$ hat exakte Überdeckung

- Sei also S' eine exakte Überdeckung.
- Jedes Element v muss von genau einer Menge der Form $\{v, e_v^c\}$ überdeckt sein.
- Dies induziert eine Färbung χ von G mit den Farben r, b und g .
- Wir müssen beweisen, dass diese Färbung zulässig ist
- Da für jedes v bereits $\{v, e_v^{\chi(v)}\} \in S'$, kann e_v^c mit $c \neq \chi(v)$ nur durch die Menge S_v^c überdeckt werden.



G dreifärbbar $\Leftrightarrow (X,S)$ hat exakte Überdeckung

- Da für jedes v bereits $\{v, e_v^{\chi(v)}\} \in S'$, kann e_v^c mit $c \neq \chi(v)$ nur durch die Menge S_v^c überdeckt werden.
- Da die Mengen der Form $\{v, e_v^{\chi(v)}\}$ und S_v^c , $c \neq \chi(v)$, alle Elemente außer den $u^{\chi(v)}$ mit $\{u, v\} \in E$ überdecken, müssen auch die Mengen $\{u^{\chi(v)}, v^{\chi(u)}\}$ für $\{u, v\} \in E$ in S' enthalten sein.
- Für diese gilt per Konstruktion $\chi(v) \neq \chi(u)$.



Problem SUBSET SUM

Gegeben: Eine endliche Menge M , eine Gewichtsfunktion
 $w : M \rightarrow \mathbb{N}_0$ und $K \in \mathbb{N}_0$

Frage: Existiert eine Teilmenge $M' \subseteq M$ mit $\sum_{a \in M'} w(a) = K$?

Satz:

Problem SUBSET SUM ist \mathcal{NP} -vollständig.

NP-Vollständigkeit von SUBSET SUM

SUBSET SUM $\in \mathcal{NP}$.

- Es kann für eine gegebene Teilmenge $M' \subseteq M$ in Polynomialzeit der Wert $\sum_{a \in M'} w(a)$ ausgerechnet und mit K verglichen werden.

EXACT COVER α SUBSET SUM

- Sei $(X = \{0, 1, \dots, m-1\}, \mathcal{S})$ EXACT COVER-Instanz.
- Konstruiere SUBSET SUM Instanz (M, w, K)

$$M := \mathcal{S}$$

$$\#x := |\{Y \in \mathcal{S} : x \in Y\}|$$

$$p := \max_{x \in X} \#x + 1$$

$$w(Y) := \sum_{x \in Y} p^x$$

$$K := \sum_{x=0}^{m-1} p^x$$

- Die Konstruktion benötigt nur Polynomialzeit.

Beweis: NP-Vollständigkeit von SUBSET SUM

$$M := \mathcal{S}$$

$$\#x := |\{Y \in \mathcal{S} : x \in Y\}|$$

$$p := \max_{x \in X} \#x + 1$$

$$w(Y) := \sum_{x \in Y} p^x$$

$$K := \sum_{x=0}^{m-1} p^x$$

Veranschaulichung:

- Wir stellen die Mengenzugehörigkeiten als Zahlen zur Basis p dar.
- Kodiere $w(Y)$ für $Y \in \mathcal{S}$ als String aus Nullen und Einsen der Länge m , wobei an i -ter Stelle eine 1 steht genau dann, wenn $i \in Y$;
- entsprechend ist K ein String der Länge m aus Einsen

Beweis: NP-Vollständigkeit von SUBSET SUM

$$M := \mathcal{S}$$

$$\#x := |\{Y \in \mathcal{S} : x \in Y\}|$$

$$p := \max_{x \in X} \#x + 1$$

$$w(Y) := \sum_{x \in Y} p^x$$

$$K := \sum_{x=0}^{m-1} p^x$$

Veranschaulichung:

- Komponentenweise Addition der zu Teilmenge Y_1, \dots, Y_n von \mathcal{S} gehörigen Strings $w(Y_1), \dots, w(Y_n)$ ergibt einen String der Länge m , an dessen i -ter Stelle steht in wievielen der $Y_j (j = 1, \dots, n)$ das Element i vorkommt.
- $\sum_{Y \in \mathcal{S}'} w(Y) = K$ bedeutet also, dass jedes $x \in X$ in genau einem $Y \in \mathcal{S}'$ vorkommt.

Beweis: NP-Vollständigkeit von SUBSET SUM

$$M := \mathcal{S}$$

$$\#x := |\{Y \in \mathcal{S} : x \in Y\}|$$

$$\rho := \max_{x \in X} \#x + 1$$

$$w(Y) := \sum_{x \in Y} \rho^x$$

$$K := \sum_{x=0}^{m-1} \rho^x$$

Beispiel:

$$X = \{0, 1, 2, 3, 4, 5, 6\},$$

$$\mathcal{S} = \{Y_1 = \{0, 1, 2, 3\}, Y_2 = \{2, 5\}, Y_3 = \{3, 4, 5\}, Y_4 = \{4, 5, 6\}\}$$

$$\#0 = 1, \#1 = 1, \#2 = 2, \#3 = 2, \#4 = 2, \#5 = 3, \#6 = 1, \rho = 4$$

$$w(Y_1) = 0001111_4, w(Y_2) = 0100100_4,$$

$$w(Y_3) = 0111000_4, w(Y_4) = 1110000_4$$

Beweis: NP-Vollständigkeit von SUBSET SUM

$$M := S$$

$$\#x := |\{Y \in \mathcal{S} : x \in Y\}|$$

$$p := \max_{x \in X} \#x + 1$$

$$w(Y) := \sum_{x \in Y} p^x$$

$$K := \sum_{x=0}^{m-1} p^x$$

■ (X, \mathcal{S}) lösbar $\Rightarrow (M, w, K)$ lösbar.

Sei $\mathcal{S}' \subseteq \mathcal{S}$ exakte Überdeckung von (X, \mathcal{S}) . Dann gilt

$$\sum_{Y \in \mathcal{S}'} w(Y) = \sum_{Y \in \mathcal{S}'} \sum_{x \in Y} p^x = \sum_{x=0}^{m-1} p^x = K$$

da jedes $x \in X$ genau einmal überdeckt wird.

\mathcal{S}' erfüllt also die Bedingung für SUBSET SUM.

Beweis: NP-Vollständigkeit von SUBSET SUM

$$M := \mathcal{S}$$

$$\#x := |\{Y \in \mathcal{S} : x \in Y\}|$$

$$\rho := \max_{x \in X} \#x + 1$$

$$w(Y) := \sum_{x \in Y} \rho^x$$

$$K := \sum_{x=0}^{m-1} \rho^x$$

■ (X, \mathcal{S}) lösbar $\Leftrightarrow (M, w, K)$ lösbar.

Ist $\mathcal{S}' \subseteq M = \mathcal{S}$ eine geeignete Menge für SUBSET SUM, so gilt

$$\sum_{Y \in \mathcal{S}'} w(Y) = K = \sum_{x=0}^{m-1} \rho^x.$$

Also kommt jedes $x \in X$ in genau einem $Y \in \mathcal{S}'$ vor.

Damit ist \mathcal{S}' eine exakte Überdeckung.

Problem PARTITION

Gegeben: Eine endliche Menge M und eine Gewichtsfunktion $w : M \rightarrow \mathbb{N}_0$.

Frage: Existiert eine Teilmenge $M' \subseteq M$ mit $\sum_{a \in M'} w(a) = \sum_{a \in M \setminus M'} w(a)$?

Satz:
Problem PARTITION ist \mathcal{NP} -vollständig.

Beweis: NP-Vollständigkeit von PARTITION

PARTITION $\in \mathcal{NP}$.

- Für eine Menge M' können in Polynomialzeit die Werte $\sum_{a \in M'} w(a)$ und $\sum_{a \in M \setminus M'} w(a)$ ausgerechnet und verglichen werden.

Beweis: NP-Vollständigkeit von PARTITION

SUBSET SUM \propto PARTITION.

- Sei (M, w, K) eine SUBSET SUM-Instanz
- Konstruiere PARTITION-Instanz (M^*, w^*)

$$N := \sum_{a \in M} w(a) + 1$$

$$M^* := M \cup \{b, c\}$$

$$w^*(a) = w(a) \quad \text{für } a \in M$$

$$w^*(b) := N - K$$

$$w^*(c) := K + 1$$

- Die Konstruktion benötigt nur Polynomialzeit.

Beweis: NP-Vollständigkeit von PARTITION

$$N := \sum_{a \in M} w(a) + 1$$

$$M^* := M \cup \{b, c\}$$

$$w^*(a) = w(a) \quad \text{für } a \in M$$

$$w^*(b) := N - K$$

$$w^*(c) := K + 1$$

- (M, w, K) Ja-Instanz genau dann, wenn (M^*, w^*) Ja-Instanz:

$$\exists M' \subseteq M^* \text{ mit } \sum_{a \in M'} w^*(a) = \sum_{a \in M^* \setminus M'} w^*(a) \iff \exists M'' \subseteq M \text{ mit } w(M'') = K.$$

- Es können b und c nicht beide in M' bzw. $M^* \setminus M'$ enthalten sein
- o.B.d.A. $b \in M'$

Beweis: NP-Vollständigkeit von PARTITION

$$N := \sum_{a \in M} w(a) + 1$$

$$M^* := M \cup \{b, c\}$$

$$w^*(a) = w(a) \quad \text{für } a \in M$$

$$w^*(b) := N - K$$

$$w^*(c) := K + 1$$

■ (M, w, K) Ja-Instanz genau dann, wenn (M^*, w^*) Ja-Instanz:

$$\exists M' \subseteq M^* \text{ mit } \sum_{a \in M'} w^*(a) = \sum_{a \in M^* \setminus M'} w^*(a) \iff \exists M'' \subseteq M \text{ mit } w(M'') = K.$$

\Rightarrow

■ Sei M' , so dass $\sum_{a \in M'} w^*(a) = \sum_{a \in M^* \setminus M'} w^*(a)$

■ Dann gilt $w(M') = N$, da $w(M^*) = 2N$

■ Damit erfüllt $M'' := M' \setminus \{b\}$ die Bedingung für SUBSET SUM.

Beweis: NP-Vollständigkeit von PARTITION

$$N := \sum_{a \in M} w(a) + 1$$

$$M^* := M \cup \{b, c\}$$

$$w^*(a) = w(a) \quad \text{für } a \in M$$

$$w^*(b) := N - K$$

$$w^*(c) := K + 1$$

- (M, w, K) Ja-Instanz genau dann, wenn (M^*, w^*) Ja-Instanz:

$$\exists M' \subseteq M^* \text{ mit } \sum_{a \in M'} w^*(a) = \sum_{a \in M^* \setminus M'} w^*(a) \iff \exists M'' \subseteq M \text{ mit } w(M'') = K.$$

⇐

- Sei M'' , so dass $w(M'') = K$
- Dann erfüllt $M' := M'' \cup \{b\}$ die Bedingung für PARTITION.

Problem KNAPSACK

Gegeben: Eine endliche Menge M ,
eine Gewichtsfunktion $w : M \rightarrow \mathbb{N}_0$,
eine Kostenfunktion $c : M \rightarrow \mathbb{N}_0$
 $W, C \in \mathbb{N}_0$.

Frage: Existiert eine Teilmenge $M' \subseteq M$ mit $\sum_{a \in M'} w(a) \leq W$
und $\sum_{a \in M'} c(a) \geq C$?

Satz:
Problem KNAPSACK ist \mathcal{NP} -vollständig.

KNAPSACK $\in \mathcal{NP}$.

- Für eine Menge M' kann in Polynomialzeit überprüft werden, ob
 - $\sum_{a \in M'} w(a) \leq W$ und
 - $\sum_{a \in M'} c(a) \geq C$
- gilt.

Beweis: NP-Vollständigkeit von KNAPSACK

PARTITION \propto KNAPSACK.

- Sei (M, w) eine PARTITION-Instanz
- Konstruiere KNAPSACK-Instanz (M, w', c, W, C)

$$\begin{aligned}w' &:= 2w \\c &:= 2w \\W = C &:= \sum_{a \in M} w(a)\end{aligned}$$

- Die Konstruktion benötigt nur Polynomialzeit.
- Es ist (M, w) genau dann eine Ja-Instanz, wenn (M, w', c, W, C) eine Ja-Instanz ist (ohne Beweis)

Auswirkung auf die Frage $\mathcal{P} = \mathcal{NP}$

- Wir haben gesehen, dass es für je zwei \mathcal{NP} -vollständige Probleme eine polynomiale Transformation von einem zum anderen Problem gibt.
- Deshalb sind alle \mathcal{NP} -vollständigen Probleme im wesentlichen gleich schwer
- Dies hat Auswirkungen auf die Frage, ob $\mathcal{P} = \mathcal{NP}$ ist.

Satz:

Sei L \mathcal{NP} -vollständig, dann gilt:

- $L \in \mathcal{P} \implies \mathcal{P} = \mathcal{NP}$
- $L \notin \mathcal{P} \implies$ für jede \mathcal{NP} -vollständigen Sprache L' gilt $L' \notin \mathcal{P}$

Auswirkung auf die Frage $\mathcal{P} = \mathcal{NP}$

Beweis: L \mathcal{NP} -vollständig, $L \in \mathcal{P} \implies \mathcal{P} = \mathcal{NP}$

- Sei $L \in \mathcal{P}$ und L \mathcal{NP} -vollständig.
- Dann existiert eine polynomiale deterministische TM M für L .
- Sei $L' \in \mathcal{NP}$
- Es gibt polynomiale Transformation $L' \propto L$
- Hintereinanderausführung von $L' \propto L$ und M liefert deterministische polynomielle TM-Berechnung für L' .
- Damit ist $L' \in \mathcal{P}$.

Auswirkung auf die Frage $\mathcal{P}=\mathcal{NP}$

Beweis: L \mathcal{NP} -vollständig, $L \notin \mathcal{P}$

\implies für jede \mathcal{NP} -vollständigen Sprache L' gilt $L' \notin \mathcal{P}$

- Sei $L \notin \mathcal{P}$ und L \mathcal{NP} -vollständig.
- Angenommen für eine \mathcal{NP} -vollständige Sprache L' gilt: $L' \in \mathcal{P}$
- Dann folgt aus Teil 1 des Satzes $\mathcal{P} = \mathcal{NP}$.
- Dies ist aber ein Widerspruch zur Voraussetzung $L \notin \mathcal{P}$.

- Die Klasse \mathcal{P} ist die Klasse aller Entscheidungsprobleme/Sprachen die mit einer **deterministischen** Turingmaschine in polynomieller Zeit gelöst werden können
- Die Klasse \mathcal{NP} ist die Klasse aller Entscheidungsprobleme/Sprachen die mit einer **nicht-deterministischen** Turingmaschine in polynomieller Zeit gelöst werden können
- Informell ausgedrückt: Π gehört zu \mathcal{NP} , falls Π folgende Eigenschaft hat: Ist die Antwort bei Eingabe eines Beispiels I von Π Ja, dann kann die Korrektheit eines Beweises (Zeugen) dafür in polynomialer Zeit überprüft werden.

- Eine **polynomiale Transformation** einer Sprache $L_1 \subseteq \Sigma_1^*$ in eine Sprache $L_2 \subseteq \Sigma_2^*$ ist eine Funktion $f: \Sigma_1^* \rightarrow \Sigma_2^*$ mit den Eigenschaften:
 - es existiert eine polynomiale deterministische Turing-Maschine, die f berechnet;
 - für alle $x \in \Sigma_1^*$ gilt: $x \in L_1 \Leftrightarrow f(x) \in L_2$.

- Eine Sprache L heißt **\mathcal{NP} -vollständig**, falls gilt:
 - $L \in \mathcal{NP}$ und
 - für alle $L' \in \mathcal{NP}$ gilt $L' \leq L$ (\mathcal{NP} -Schwere).

- **Bedeutung:** Unter der Annahme $\mathcal{P} \neq \mathcal{NP}$ gibt es kein polynomielles Lösungsverfahren für ein \mathcal{NP} -vollständiges Problem.

- Mit dem Satz von Cook haben wir direkt gezeigt, dass Problem SAT \mathcal{NP} -schwer ist
- Bei allen anderen Problemen haben wir polynomielle Transformationen (Reduktionen) benutzt um die \mathcal{NP} -Schwere nachzuweisen:

$$\text{SAT} \propto \text{3SAT} \propto \text{3COLOR} \propto \text{EXACT COVER} \propto \text{SUBSET SUM} \propto$$
$$\text{PARTITION} \propto \text{KNAPSACK}$$