

# 9. Polynominalzeitreduktionen

## Einleitung

---

### Effizient lösbare Probleme (Wiederholung)

- **Definition:** Ein Problem gilt als **effizient lösbar**, wenn es durch einen Algorithmus gelöst werden kann, dessen Laufzeit durch ein **Polynom** der Eingabegröße beschränkt ist.
  - Laufzeit:  $O(n^c)$ 
    - $n$ : Eingabegröße (z.B. Anzahl der Bits)
    - $c$ : Konstanter Exponent
- **Synonym:** Effizient lösbare Probleme werden auch als **handhabbar (tractable)** bezeichnet.
- **Cobham-Edmonds-Annahme:**
  - Vorschlag von Alan Cobham und Jack Edmonds in den 1960er-Jahren.
  - **Gleichsetzung von Handhabbarkeit mit Lösbarkeit in Polynomialzeit.**
  - Hat die Informatikforschung der letzten 50 Jahre maßgeblich beeinflusst und sich weitgehend durchgesetzt.

### Diskussion zur Cobham-Edmonds-Annahme

- **Rechtfertigung der Annahme:**
  - **Praxisbezug:** Polynomielle Algorithmen weisen in der Regel kleine Konstanten und niedrige Exponenten auf.
  - **Strukturelle Einsicht:** Der Übergang von exponentiellen (z.B. Brute-Force) zu polynomiellen Algorithmen deutet oft auf das Erkennen einer fundamentalen Struktur des Problems hin.
- **Ausnahmen/Kritik:**
  - **Ineffiziente polynomielle Algorithmen:** Es existieren polynomielle Algorithmen mit sehr großen Konstanten oder hohen Exponenten, die in der praktischen Anwendung unbrauchbar sein können.
  - **Praktische Relevanz exponentieller Algorithmen:** Algorithmen mit exponentieller oder noch schlechterer Laufzeit finden dennoch Anwendung, wenn:
    - Worst-Case-Eingaben extrem selten auftreten.
    - Die Größe der zu lösenden Problemfälle ausreichend klein ist.

---

## Probleme klassifizieren: P or not P?

### Ziel der Klassifizierung von Problemen

- Unterscheidung zwischen Problemen, die **in Polynomialzeit lösbar** sind, und solchen, die **nicht in Polynomialzeit lösbar** sind.
- (NP-Problem)

## Beispiele für Probleme, die nachweislich mehr als polynomielle Zeit erfordern:

- **Halteproblem mit Schranke:** Hält eine gegebene Turingmaschine nach höchstens  $k$  Schritten?
- **Verallgemeinertes Schachgewinnproblem:** Gegeben sei eine Brettbelegung für eine  $n \times n$  Generalisierung von Schach. Kann Schwarz garantiert gewinnen?

## Probleme, deren Klassifizierung (P oder nicht P?) unbekannt ist:

- **Maximum Independent Set:** Gegeben ein Graph  $G$  und eine Zahl  $k$ , enthält  $G$  mindestens  $k$  Knoten, die paarweise nicht adjazent sind?
- **3-Färbbarkeit:** Lassen sich die Knoten eines gegebenen Graphen mit 3 Farben färben, sodass Paare adjazenter Knoten unterschiedliche Farben haben?
- **SAT (Erfüllbarkeitsproblem der Aussagenlogik):** Ist eine gegebene aussagenlogische Formel erfüllbar?

**Anmerkung:** Für viele fundamentale Probleme konnte noch keine eindeutige Klassifizierung (polynomial oder exponentiell) gefunden werden. Dies ist ein unbefriedigender Zustand in der theoretischen Informatik.

## Umgang mit Problemen, die nicht in Polynomialzeit lösbar sind:

- Wie gehen wir damit um, wenn wir ein Problem nicht in Polynomialzeit lösen können?
  - Man soll nicht sagen, dass man zu blöd ist den Algo zu finden, sondern soll sagen, dass „am selbst + die ganzen anderen Personen nichts gefunden haben.“
- 

## Ja/Nein Problem

siehe hier

- **Vereinfachung:** Zur einfacheren Betrachtung konzentrieren wir uns auf **Ja/Nein-Probleme (decision problems)**.
- **Definition Ja/Nein-Problem:** Ein Problem, dessen Lösung entweder **Ja** oder **Nein** ist.
- **Unterscheidung zu anderen Problemtypen:**
  - **Funktionales Problem:** Liefert eine **Lösung** oder eine **Lösungsmenge** als Antwort.
  - **Optimierungsproblem:** Ziel ist es, eine **optimale Lösung** (z.B. Minimum oder Maximum) zu finden.

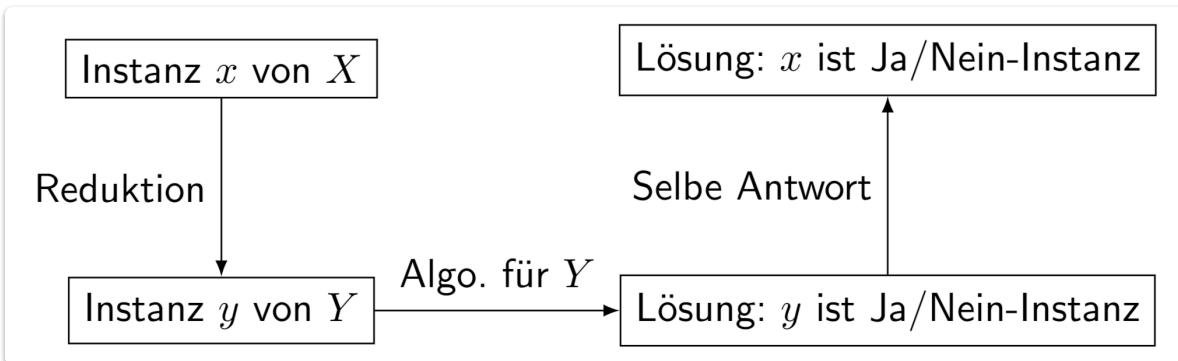
## Beispiel und Unterscheidung:

- **Ja/Nein-Problem:** Gibt es für einen gewichteten Graphen einen Spannbaum mit Kosten  $\leq k$ ? (Antwort: Ja oder Nein)
- **Funktionales Problem:** Finde in einem gewichteten Graphen einen Spannbaum mit Kosten  $\leq k$ . (Antwort: Der Spannbaum selbst, falls er existiert)
- **Optimierungsproblem:** Finde in einem gewichteten Graphen einen Spannbaum mit minimalen Kosten (MST). (Antwort: Der Spannbaum mit den geringsten Gesamtkosten)

## Polynominalzeitreduktionen

Wir haben 2 Probleme und wollen das Problem  $X$  auf das Problem  $Y$  polynomiell reduzieren.

- **Informelle Beschreibung:** Wenn ein Problem  $X$  **polynomialzeitreduzierbar** auf ein Problem  $Y$  ist, bedeutet das:
  - Falls wir einen **effizienten Algorithmus** (Polynomialzeit) für  $Y$  haben,
  - dann können wir auch  $X$  **effizient lösen**.
- **Vorgehensweise der Reduktion:**
  1. Gegeben eine **Instanz  $x$  von Problem  $X$** .
  2. Transformiere  $x$  in eine **Instanz  $y$  von Problem  $Y$**  in Polynomialzeit.
  3. Löse die Instanz  $y$  von  $Y$  mit dem effizienten Algorithmus für  $Y$ .
  4. **Schlussfolgerung:**
    - Wenn  $y$  eine **Ja-Instanz** von  $Y$  ist, dann ist auch  $x$  eine **Ja-Instanz** von  $X$ .
    - Wenn  $y$  eine **Nein-Instanz** von  $Y$  ist, dann ist auch  $x$  eine **Nein-Instanz** von  $X$ .



## Definition

siehe

Eine **Polynominalzeitreduktion** von Problem  $X$  auf Problem  $Y$  ist ein Algorithmus  $R$ , der für jede Instanz  $x$  von  $X$  eine Instanz  $y$  von  $Y$  berechnet, so dass die folgenden zwei Bedingungen erfüllt sind:

1. **Äquivalenz der Ja-Instanzen:**  $x$  ist eine Ja-Instanz von X genau dann, wenn  $y$  eine Ja-Instanz von Y ist.
2. **Effizienz der Reduktion:** Der Algorithmus  $R$  hat eine **Laufzeit in Polynomialzeit**. Das bedeutet, es existiert eine Konstante  $c$ , sodass  $R$  die Instanz  $y$  in einer Zeit von  $O(n^c)$  berechnet, wobei  $n$  die Eingabegröße der Instanz  $x$  ist.

## Notation

Wir schreiben  $X \leq_P Y$ , um auszudrücken, dass es eine Polynomialzeitreduktion von X auf Y gibt. In diesem Fall sagen wir auch: "**X ist auf Y polynomiell reduzierbar**".

## Hinweis zur Interpretation

Falls  $X \leq_P Y$  gilt, kann man auch sagen, dass "**Y mindestens so schwer ist wie X**"

## Lösung von Problemen durch Reduktion

siehe

- **Idee:** Nutze eine Polynomialzeitreduktion auf ein bereits als handhabbar bekanntes Problem.
- **Grundsatz:** Wenn  $X \leq_P Y$  und Y in Polynomialzeit lösbar ist, dann ist auch X in Polynomialzeit lösbar.
- **Beweis:**
  - Sei  $R$  der Reduktionsalgorithmus von X nach Y mit Laufzeit  $O(n^a)$  für eine Instanz  $x$  von X der Größe  $n$  ( $a \geq 1$ ).
  - Sei  $A$  der Algorithmus zum Lösen von Y mit Laufzeit  $O(n^b)$  für eine Instanz  $y$  von Y der Größe  $n$  ( $b \geq 1$ ).
  - Gegeben eine Instanz  $x$  von X der Größe  $n$ :
    1. Anwenden von  $R$  auf  $x$  erzeugt eine Instanz  $y$  von Y in  $O(n^a)$  Zeit.
    2. Die Größe von  $y$  ist höchstens  $O(n^a)$ , da sie in dieser Zeit erzeugt wurde.
    3. Lösen von  $y$  mit  $A$  benötigt  $O((n^a)^b) = O(n^{ab})$  Zeit.
  - Die Gesamtaufzeit zur Lösung von  $x$  ist  $O(n^a) + O(n^{ab}) = O(n^{ab})$ , was ein Polynom in  $n$  ist.

## Nachweis der Nicht-Handhabbarkeit durch Reduktion

siehe

- **Idee:** Reduziere ein bereits als nicht handhabbar bekanntes Problem auf das zu untersuchende Problem.
- **Grundsatz:** Wenn  $X \leq_P Y$  und X nicht in Polynomialzeit lösbar ist, dann kann auch Y nicht in Polynomialzeit lösbar sein.
- **Beweis (durch Widerspruch):**
  - Angenommen, Y wäre in Polynomialzeit lösbar.

- Da  $X \leq_P Y$ , existiert eine Polynominalzeitreduktion von X auf Y.
- Durch die Kombination der Polynominalzeitreduktion und des Polynomialzeitalgorithmus für Y könnte X ebenfalls in Polynominalzeit gelöst werden.
- Dies widerspricht der Annahme, dass X nicht in Polynominalzeit lösbar ist.

## Polynominalzeitäquivalenz

- **Definition:** Wenn  $X \leq_P Y$  und  $Y \leq_P X$ , dann schreiben wir  $X \equiv_P Y$ . Dies bedeutet, dass X und Y bezüglich ihrer Schwierigkeit in Polynominalzeit äquivalent sind.

## Transitivität der Polynominalzeitreduktion

siehe

- **Satz:** Ist  $X \leq_P Y$  und  $Y \leq_P Z$ , dann folgt daraus  $X \leq_P Z$ .
- **Beweis:**
  - Sei  $R_1$  der Reduktionsalgorithmus von X nach Y mit Laufzeit  $O(n^a)$  ( $a \geq 1$ ).
  - Sei  $R_2$  der Reduktionsalgorithmus von Y nach Z mit Laufzeit  $O(n^b)$  ( $b \geq 1$ ).
  - Sei  $x$  eine Instanz von X der Größe  $n$ .
  - $R_1(x)$  erzeugt eine Instanz  $x'$  von Y in höchstens  $O(n^a)$  Zeit, wobei die Größe von  $x'$   $O(n^a)$  ist.
  - $R_2(x')$  erzeugt eine Instanz  $z$  von Z in  $O((n^a)^b) = O(n^{ab})$  Zeit.
  - Somit existiert eine Polynominalzeitreduktion (die Komposition von  $R_1$  und  $R_2$ ) von X nach Z mit einer Laufzeit von  $O(n^a) + O(n^{ab}) = O(n^{ab})$ , was polynomiell in  $n$  ist.

## Angeben von Polynominalzeitreduktion

siehe

Beim Definieren einer Polynominalzeitreduktion von einem Problem X auf ein Problem Y müssen zwei zentrale Eigenschaften nachgewiesen werden:

### 1. Korrektheit der Reduktion:

- Ja-Instanzen von X müssen auf Ja-Instanzen von Y abgebildet werden.
- Nein-Instanzen von X müssen auf Nein-Instanzen von Y abgebildet werden.

### 2. Polynomialität der Reduktion:

- Der Reduktionsalgorithmus muss in Polynominalzeit ausführbar sein.

Die Schwierigkeit des Nachweises kann variieren: In manchen Fällen ist die Korrektheit offensichtlich, während in anderen die Polynomialität leichter zu zeigen ist.

## Independent set

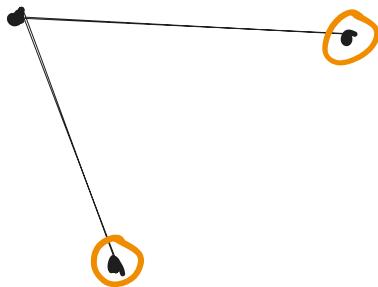
**Independent Set:** Ein Independent Set eines Graphen  $G = (V, E)$  ist eine Teilmenge  $S \subseteq V$ , in der es **keine zwei adjazenten Knoten** gibt.

**INDEPENDENT SET:** Gegeben sei ein Graph  $G = (V, E)$  und eine ganze Zahl  $k$ . Gibt es ein Independent Set  $S$ , sodass  $|S| \geq k$  gilt?

**Wichtig:** Die Zahl  $k$  ist Teil der Eingabe und keine Konstante.

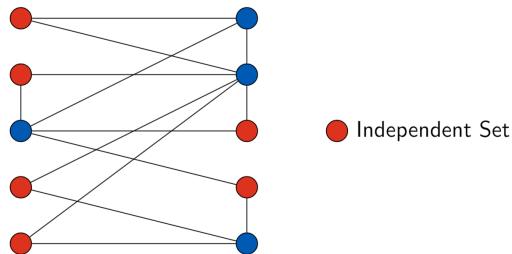
**Hinweis:** Ja/Nein-Probleme werden ab jetzt mit dieser Schreibweise (**INDEPENDENT SET**) gekennzeichnet.

Independent set  $k=2$



## Weiteres Beispiel

**INDEPENDENT SET:** Gegeben sei ein Graph  $G = (V, E)$  und eine ganze Zahl  $k$ . Gibt es ein Independent Set  $S$ , sodass  $|S| \geq k$  gilt?



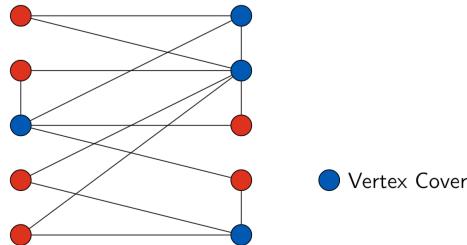
**Beispiel:** Existiert ein Independent Set der Größe  $\geq 6$ ? Ja.

**Beispiel:** Existiert ein Independent Set der Größe  $\geq 7$ ? Nein.

## Vertex Cover

**Vertex Cover:** Ein Vertex Cover eines Graphen  $G = (V, E)$  ist eine Menge  $S \subseteq V$ , sodass jede Kante des Graphen zu mindestens einem Knoten aus  $S$  inzident ist.

**VERTEX COVER:** Gegeben sei ein Graph  $G = (V, E)$  und eine ganze Zahl  $k$ . Gibt es ein Vertex Cover  $S$  von  $G$ , sodass  $|S| \leq k$ ?



Beispiel: Existiert ein Vertex Cover der Größe  $\leq 4$ ? Ja.

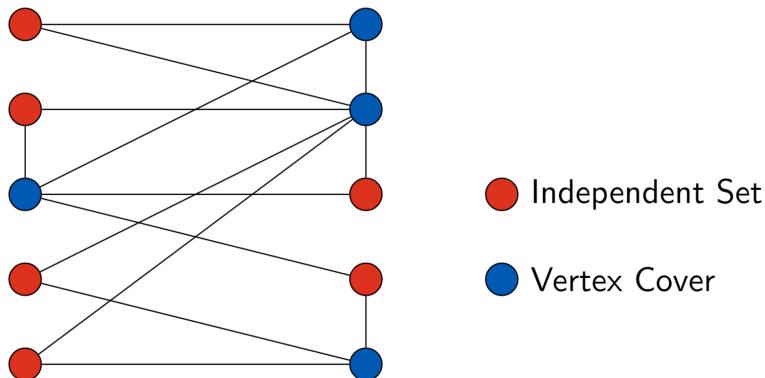
Beispiel: Existiert ein Vertex Cover der Größe  $\leq 3$ ? Nein.

## Konversionslemma: Vertex Cover und Independent Set

Wir wollen zeigen, dass  $\text{Vertex Cover} \equiv_P \text{Independent Set}$ . Dazu zeigen wir zuerst:

Sei  $G = (V, E)$  ein ungerichteter Graph mit Knotenmenge  $V$  und Kantenmenge  $E$ . Sei  $S \subseteq V$  eine Teilmenge der Knoten und  $C = V - S$  das Komplement von  $S$  in  $V$ . Dann gilt:

$S$  ist ein Independent Set von  $G$  genau dann, wenn  $C$  ein Vertex Cover von  $G$  ist.



Beweis:  $\Rightarrow$

- Betrachte eine beliebige Kante  $(u, v) \in E$ . Wir wollen zeigen, dass mindestens einer der beiden Knoten  $u, v$  in  $C$  liegt.
- Weil  $S$  ein Independent Set ist, muss mindestens einer der beiden Knoten  $u, v$  nicht in  $S$  liegen. Also liegt mindestens einer der beiden Knoten in  $C$ .
- Daher ist  $C$  ein Vertex Cover.  $\square$

Beweis:  $\Leftarrow$

- Betrachte zwei Knoten  $u \in S$  und  $v \in S$ . Wir wollen zeigen, dass  $u$  und  $v$  nicht adjazent sind.
- Aus  $u \in S$  und  $v \in S$  folgt  $u \notin C$  und  $v \notin C$ .
- $u$  und  $v$  können nicht adjazent sein, ansonsten wäre  $C$  kein Vertex Cover (weil es die Kante  $(u, v)$  nicht überdeckt).
- Also ist  $S$  ein Independent Set.  $\square$

Also gilt das Lemma.

## Vertex Cover und Independent Set

### Vertex Cover und Independent Set

$$(G, k) \xrightarrow{\text{VC}} (G, n-k)$$

I.S.

Es gilt: VERTEX COVER  $\equiv_P$  INDEPENDENT SET.

Beweis:

Zuerst zeigen wir VERTEX COVER  $\leq_P$  INDEPENDENT SET.

- Sei  $(G, k)$  eine Instanz von VERTEX COVER. Sei  $n$  die Anzahl der Knoten von  $G$ .
- In Polynomialzeit generieren wir  $(G, n - k)$ , eine Instanz von INDEPENDENT SET.
- Die Reduktion ist korrekt, da  $G$  ein Vertex Cover  $\leq k$  hat genau dann wenn  $G$  ein Independent Set der Größe  $\geq n - k$  hat (folgt aus dem Konversionslemma).
- Die Reduktion ist klarerweise polynomiell, da sie ja nur  $n$  durch  $n - k$  ersetzen braucht.  $\square$

Es gilt: VERTEX COVER  $\equiv_P$  INDEPENDENT SET.

Beweis:

Weiters zeigen wir INDEPENDENT SET  $\leq_P$  VERTEX COVER

- Sei  $(G, k)$  eine Instanz von INDEPENDENT SET. Sei  $n$  die Anzahl der Knoten von  $G$ .
- In Polynomialzeit generieren wir  $(G, n - k)$  eine Instanz von VERTEX COVER.
- Die Reduktion ist korrekt, da  $G$  ein Independent Set  $\geq k$  hat genau dann wenn  $G$  ein Vertex Cover Set der Größe  $\leq n - k$  hat (folgt aus dem Konversionslemma).
- Die Reduktion ist klarerweise polynomiell, da sie ja nur  $n$  durch  $n - k$  ersetzen braucht.  $\square$

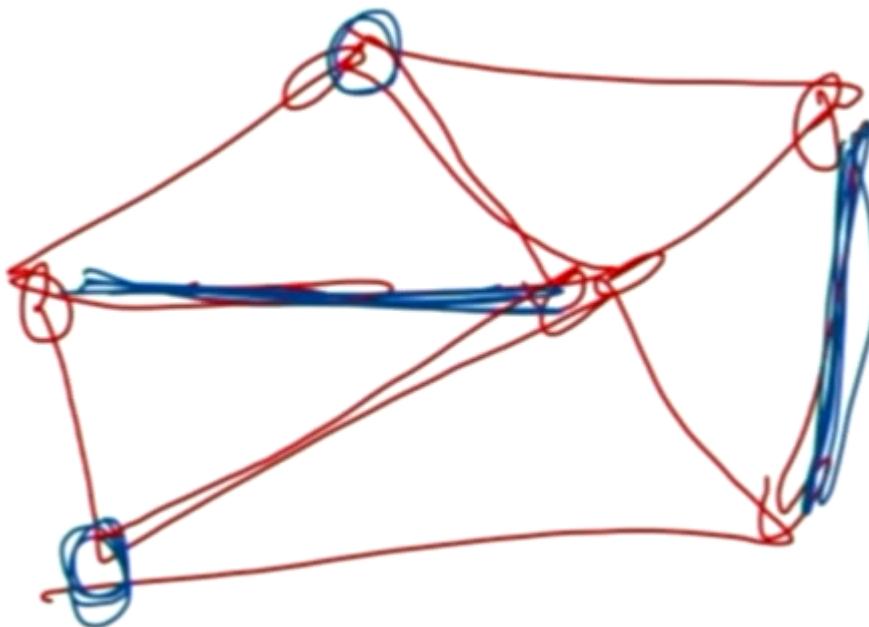
Wir haben also beide Richtungen gezeigt, und es folgt die Äquivalenz.

## Beispiel zu Polynomial-Zeit: Nicht Blokierer

### AD\_09\_PolynomialzeitreduktionUndNP, p.25

Wenn man jeden Knoten von jedem Knoten immer noch erreichen kann auch wenn man eine

Kante weglassen würde. Beispielsweise bei Straßennetz und Baustelle:



Ein maximaler Blockierer ist das Kompliment zu einem minimalen Spannbaum. Wir können daher das Problem ganz einfach in Polynomial-Zeit lösen indem wirs mit dem Minimalen Spannbaum Problem Lösen.

### Die einfache Idee:

- Finde zuerst die kleinstmögliche Menge an "wichtigen" Straßen (den minimalen Spannbaum).
- Alle anderen Straßen, die *nicht* zu dieser kleinen Menge gehören, sind die "nicht so wichtigen" Straßen. Diese "nicht so wichtigen" Straßen bilden den maximalen Blockierer. Wenn du alle diese sperrst, bleibt das Netz trotzdem durch die "wichtigen" Straßen verbunden.

### Warum ist das einfach zu lösen?

Es gibt clevere und schnelle Methoden (in "Polynomialzeit"), um diesen minimalen Satz an "wichtigen" Straßen (den minimalen Spannbaum) zu finden. Da wir den maximalen Blockierer einfach dadurch bekommen, dass wir alle anderen Straßen nehmen, ist auch das Finden des maximalen Blockierers einfach und schnell.

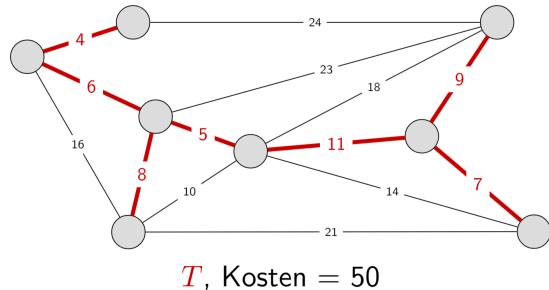
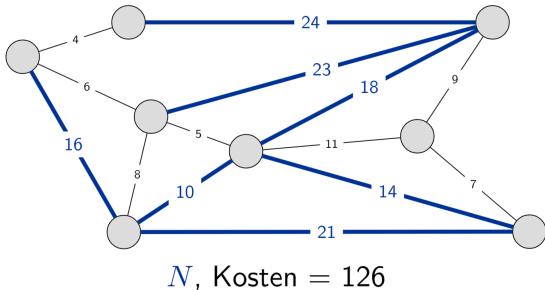
### Kurz gesagt:

Maximaler Blockierer = Alle Verbindungen (Straßen) **außer** den absolut notwendigen Verbindungen (minimaler Spannbaum).

## Beispiel Spannbäume und Nicht-Blockierer

**Konversionslemma:** Sei  $G = (V, E)$  ein gewichteter Graph,  $N \subseteq E$  und  $T = E - N$ . Dann ist  $N$  ein maximaler Nicht-Blockierer genau dann wenn  $T$  ein minimaler Spannbaum ist. Die Kosten von  $N$  sind genau  $K := \sum_{e \in E} c_e$  minus der Kosten von  $T$ .

Wir lassen den sehr einfachen Beweis als Übung.



**Konversionslemma:** Sei  $G = (V, E)$  ein gewichteter Graph,  $N \subseteq E$  und  $T = E - N$ . Dann ist  $N$  ein maximaler Nicht-Blockierer genau dann wenn  $T$  ein minimaler Spannbaum ist. Die Kosten von  $N$  sind genau  $K := \sum_{e \in E} c_e$  minus der Kosten von  $T$ .

Wir lassen den sehr einfachen Beweis als Übung.

Es gilt: MNB  $\equiv_P$  MST\*

Beweis:

- MNB  $\leq_P$  MST\*: Wir reduzieren eine Instanz  $(G, k)$  von MNB auf die Instanz  $(G, K - k)$  von MST\*.
- MST\*  $\leq_P$  MNB: Wir reduzieren eine Instanz  $(G, k)$  von MST\* auf die Instanz  $(G, K - k)$  von MNB.  $\square$

Es folgt daher: MNB ist in Polynomialzeit lösbar.

ab hier erst so richtig Mitschrift weil davor Folien nicht online waren

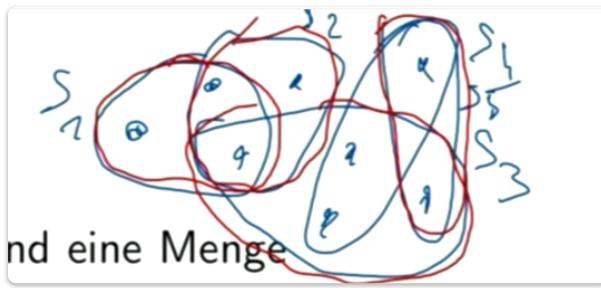
## Set Cover (Mengenüberdeckungsproblem)

**Set Cover:** Gegeben sei eine Menge  $U$  von Elementen und eine Menge  $\mathcal{S} = \{S_1, S_2, \dots, S_m\}$  von Teilmengen von  $U$ . Ein Set Cover ist eine Teilmenge  $\mathcal{C} \subseteq \mathcal{S}$ , also eine Menge von Mengen, deren Vereinigung  $U$  entspricht.  $\mathcal{C}$  ist ein Set Cover von  $\mathcal{S}$ .

**SET COVER:** Gegeben sei eine Menge  $U$  von Elementen, eine Menge  $\mathcal{S} = \{S_1, S_2, \dots, S_m\}$  von Teilmengen von  $U$  und eine ganze Zahl  $k$ . Existiert eine Teilmenge  $\mathcal{C} \subseteq \mathcal{S}$  mit  $|\mathcal{C}| \leq k$ , sodass die Vereinigung von  $\mathcal{C}$  gleich  $U$  ist?

### Beispielhafte Anwendung:

- $m$  verfügbare Softwarekomponenten.
- Menge  $U$  von  $n$  Eigenschaften, die unser Softwaresystem haben sollte.
- Die  $i$ -te Softwarekomponente bietet eine Menge  $S_i \subseteq U$  von Eigenschaften an.
- Ziel: Erreiche alle  $n$  Eigenschaften mit maximal  $k$  Komponenten.



Ziel: die gesammte Menge mit möglichst wenig Elementen überdecken

Beispiel möglichst wenige Pizzen aus Speisekarten aussuchen, dass möglichst viele Zutaten drauf sind.

### Beispiel:

$$\begin{aligned} U &= \{1, 2, 3, 4, 5, 6, 7\} \\ k &= 2 \\ S_1 &= \{3, 7\} & S_4 &= \{2, 4\} \\ S_2 &= \{3, 4, 5, 6\} & S_5 &= \{5\} \\ S_3 &= \{1\} & S_6 &= \{1, 2, 6, 7\} \end{aligned}$$

### UE5

Bei der ue5 haben wir eine spezielle Art von Set Cover kennengelernt das 2 Set Cover:

### Set Cover (klassisch):

- **Eingabe:** Eine Menge  $U = \{u_1, u_2, \dots, u_n\}$  von Elementen, eine Sammlung  $S = \{S_1, S_2, \dots, S_m\}$  von Teilmengen von  $U$ , und eine Zahl  $k$ .

- **Frage:** Gibt es eine Teilmenge  $C \subseteq S$ , so dass  $|C| \leq k$  und jedes Element  $u_i \in U$  von mindestens einer Menge in  $C$  abgedeckt wird?

## 2-Set Cover:

- **Eingabe:** Eine Menge  $U = \{u_1, u_2, \dots, u_n\}$  von Elementen, eine Sammlung  $S = \{S_1, S_2, \dots, S_m\}$  von Teilmengen von  $U$ , und eine Zahl  $k$ .
- **Frage:** Gibt es eine Teilmenge  $C \subseteq S$ , so dass  $|C| \leq k$  und jedes Element  $u_i \in U$  von **mindestens 2** verschiedenen Mengen in  $C$  abgedeckt wird?

## Vertex Cover auf Set Cover reduzieren

**Behauptung:** VERTEX COVER  $\leq_P$  SET COVER.

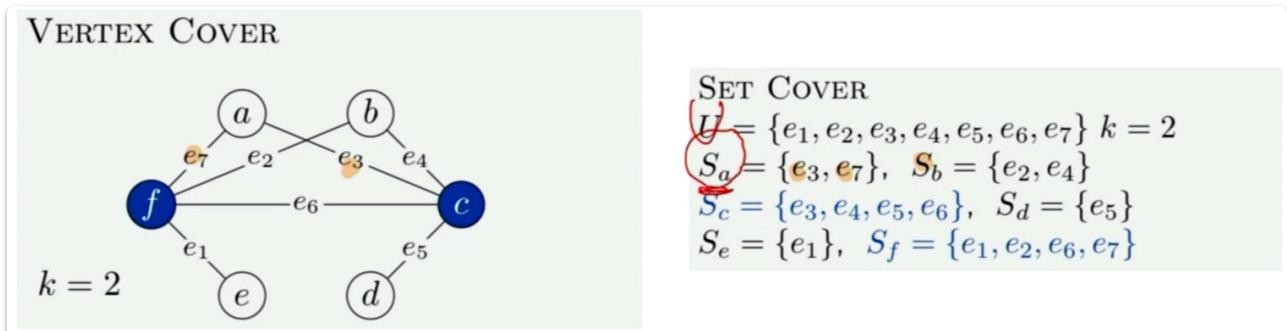
**Beweis:** Gegeben sei eine Vertex Cover Instanz  $(G, k)$  mit  $G = (V, E)$ .

Wir konstruieren eine Instanz von SET COVER.

- $k = k$ ,
- $U = E$ ,
- Für jedes  $v \in V$  erzeugen wir eine Menge  $S_v = \{e \in E : e \text{ inzident zu } v\}$
- $S$  ist die Menge aller  $S_v$  für  $v \in V$ .
- Die Reduktion ist klarerweise *polynomiell*.

Wir wollen hier das **Vertex Cover als Set Cover darstellen**.

Für jeden Knoten nehmen wir alle Kanten die zu dem Knoten inzident sind. Beispiel Kante  $e_3$  und  $e_7$



und das machen wir jetzt für alle Knoten. Und das  $k$  übernehmen wir einfach und lassen wir gleich.

- **Korrektheit:**  $G$  hat ein Vertex Cover der Größe  $\leq k$  genau dann wenn  $S$  ein Set Cover der Größe  $\leq k$  hat.
- $\Rightarrow$ : Sei  $C = \{v_1, \dots, v_k\} \subseteq V$  ein Vertex Cover von  $G$ . Dann ist  $\mathcal{C} = \{S_{v_1}, \dots, S_{v_k}\}$  ein Set Cover von  $S$ .
- $\Leftarrow$ : Sei  $\mathcal{C} = \{S_{v_1}, \dots, S_{v_k}\}$  ein Set Cover von  $S$ . Dann ist  $C = \{v_1, \dots, v_k\} \subseteq V$  ein Vertex Cover von  $G$ .  $\square$

## Bemerkung

Jede Setcover Instanz die wir durch Reduktion bekommen, hat bestimmte Eigenschaften. Nicht jede Instanz wird diese Eigenschaften haben. Eine die wir bei unserem Beispiel herauslesen können ist, dass jede Kante in genau 2 Mengen vorkommt. Weil wir für jeden Knoten die Menge aller inzidenten Kanten finden und jede ist zu genau 2 inzident also wird das stimmen. Wenn wir aber im allgemeinen das machen, muss das nicht stimmen.

- Nicht jede Set Cover Instanz kann durch die Reduktion von Vertex Cover entstehen.
  - Daher sprechen wir hier von einer "**Reduktion eines Spezialfalls auf den allgemeinen Fall**".
- 

## Reduktion mit Gadgets

Mit Gadgets sind in diesem Fall diese Dreiecke gemeint und ist ein allgemeiner Begriff um kleine Bausteine zu beschreiben, die dann eine ganze Instanz implementieren. Aber es gibt keine genau Definition von Gadgets.

### Erfüllbarkeitsproblem (satisfiability)

siehe [GDS](#)

- **Literal:** Eine boolesche Variable ( $x_i$ ) oder ihre Negation ( $\neg x_i$ ).
  - Beispiele:  $x_i, \neg x_i$
- **Klausel:** Eine Disjunktion (logisches ODER,  $\vee$ ) von Literalen.
  - Beispiel:  $C_j = x_1 \vee x_2 \vee x_3$
- **Konjunktive Normalform (KNF):** Eine aussagenlogische Formel  $\Phi$ , bei der Klauseln konjunktiv (logisches UND,  $\wedge$ ) verknüpft werden.
  - Beispiel:  $\Phi = C_1 \wedge C_2 \wedge C_3 \wedge C_4$
- **Wahrheitsbelegung (truth assignment):** Eine Wahrheitsbelegung ist eine Funktion  $f$ , die jeder Variable einen Wahrheitswert `true` oder `false` zuordnet.
- **Erfüllen einer Formel:** Eine Wahrheitsbelegung  $f$  erfüllt eine KNF-Formel  $\Phi$ , falls jede Klausel von  $\Phi$  mindestens eine Variable  $x$  mit  $f(x) = \text{true}$  oder eine negierte Variable  $\neg x$  mit  $f(x) = \text{false}$  enthält.

### Das Erfüllbarkeitsproblem (SAT)

- **SAT (satisfiability):** Gegeben ist eine KNF-Formel  $\Phi$ . Gibt es eine Wahrheitsbelegung, die  $\Phi$  erfüllt?
- **3-SAT:** SAT, bei dem jede Klausel genau 3 Literale enthält.
  - Jedes Literal muss sich auf eine unterschiedliche Variable beziehen.
  - Beispiel:  $(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$
  - **Erfüllende Wahrheitsbelegung:**  $f(x_1) = \text{true}, f(x_2) = \text{true}, f(x_3) = \text{false}$ .

## Bedeutung von SAT

Das Erfüllbarkeitsproblem (SAT) ist in zweierlei Hinsicht von großer Bedeutung:

1. **Mächtige heuristische Algorithmen (SAT-Solver):** Für SAT existieren leistungsstarke heuristische Algorithmen (SAT-Solver), die große, "strukturierte" Instanzen lösen können.
  - → Daher ist SAT ein beliebtes Problem, um andere Probleme darauf zu reduzieren (Lösung durch Reduktion).
2. **Nicht-Handhabbarkeit für allgemeine Instanzen:** Andererseits wird SAT für allgemeine Instanzen als nicht-handhabbar angesehen (SAT ist "NP-vollständig", was auf den nächsten Folien erläutert wird).
  - → Daher ist SAT ein beliebtes Problem, das auf andere Probleme reduziert wird, um deren Nicht-Handhabbarkeit zu zeigen.

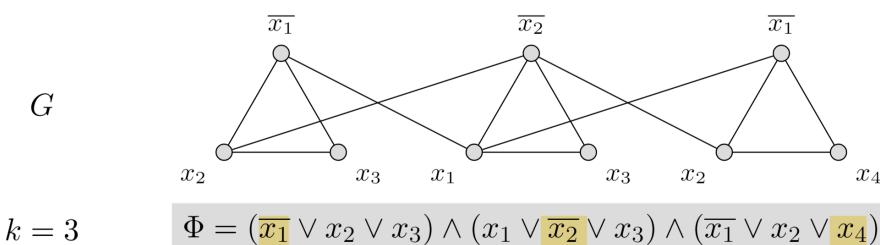
## 3-Sat auf Independent Set reduzieren

**Behauptung:** 3-SAT  $\leq_P$  INDEPENDENT SET.

**Beweis:** Gegeben sei eine Instanz  $\Phi$  von 3-SAT mit  $k$  Klauseln. Wir konstruieren eine Instanz  $(G, k)$  von INDEPENDENT SET.

- $G$  enthält 3 Knoten für jede Klausel (einen für jedes Literal).
- Verbinde 3 Literale in einer Klausel zu einem Dreieck.
- Verbinde ein Literal mit jeder seiner Negationen.

Die Reduktion ist klarerweise **polynomiel**.



- Als  $k$  definieren wir immer die **Anzahl der Klauseln**

**Korrektheit:**  $G$  enthält ein Independent Set der Größe  $k$  genau dann wenn  $\phi$  erfüllbar ist.

**Beweis:**

( $\Rightarrow$ ) Sei  $S$  ein Independent Set der Größe  $k$ .

- $S$  enthält **genau einen Knoten pro Dreieck** ( $S$  kann höchstens einen Knoten pro Dreieck enthalten, ansonsten ist  $S$  nicht unabhängig;  $S$  muss mindestens einen Knoten pro Dreieck enthalten, da  $|S| = k$  und es  $k$  Dreiecke gibt).
- Wir konstruieren eine Wahrheitsbelegung der Variablen, indem wir `x = true` setzen, falls  $x \in S$ , und `x = false` setzen, falls  $\neg x \in S$ .

- Wegen der Kanten zwischen den Dreiecken kann es zu *keinen widersprüchlichen Belegungen ein und derselben Variable* kommen.
- Wir setzen die *übrigen Variablen* beliebig auf `true` oder `false`.
- Diese *Wahrheitsbelegung erfüllt alle Klauseln*.

( $\Leftarrow$ ) Gegeben sei eine Wahrheitsbelegung  $f$ , die  $\Phi$  erfüllt.

- Wir konstruieren ein Independent Set  $S$ , indem wir von jedem Dreieck einen Knoten  $l$  mit  $l = x$  und  $f(x) = \text{true}$  oder einen Knoten  $l$  mit  $l = \neg x$  und  $f(x) = \text{false}$  wählen. (So einen Knoten  $l$  gibt es immer, da  $f$  erfüllend).
- $S$  ist unabhängig, weil die Kanten zwischen den Dreiecken jeweils zwischen einer Variable und ihrer Negation verlaufen.
- $|S| = k$ , weil wir von *jedem Dreieck einen Knoten wählen*.

## Optimierungsprobleme

Ja/Nein-Problem: Existiert ein Vertex Cover der Größe  $\leq k$ ?

Optimierungsproblem: Finde kleinstes Vertex Cover.

Klar:

- Ja/Nein-Problem kann mit dem Optimierungsproblem gelöst werden.
- Berechne kleinstes Vertex Cover  $C$  und antworte „Ja“ falls  $|C| \leq k$  ist.

Umgekehrte Richtung: Löse Optimierungsproblem mittels (mehrmaligem) Lösen des Ja/Nein-Problems.

Können wir die beiden Probleme jetzt aufeinander reduzieren?

Das Ja/Nein Problem kann sehr einfach auf das Optimierungsproblem reduzieren, weil man einfach den Algorithmus ausführen kann von dem OP und dann schauen ob  $k$  kleiner/größer ist, aber umgekehrt ist das schon bisschen schwerer.

## Optimierungsproblem für Vertex Cover lösen

**Ausgangslage:** Es existiert ein Algorithmus  $VC(G, k)$ , der das Ja/Nein-Problem für ein Vertex Cover der Größe  $\leq k$  löst.

**Ziel:** Wir möchten mittels  $VC(G, k)$  ein kleinstes Vertex Cover finden.

**Verwendete Eigenschaften für jeden Graphen  $G = (V, E)$ :**

1. Sei  $v \in V$ . Falls  $C$  ein Vertex Cover von  $G - v$  ist, dann ist  $C \cup \{v\}$  ein Vertex Cover von  $G$ .
2. Falls  $G$  ein Vertex Cover der Größe  $k \geq 1$  besitzt, dann gibt es ein  $v \in V$ , sodass  $G - v$  ein Vertex Cover der Größe  $k - 1$  besitzt.

## Der Algorithmus OptVC(G)

Der Algorithmus `OptVC(G)` berechnet ein kleinstes Vertex Cover von  $G$  mittels mehrmaligen Aufrufs von  $VC(G, k)$ .

```

OptVC(G):
  for k ← 0 bis n - 1
    if VC(G, k)
      return FindVC(G, k)

FindVC(G, k):
  if k = 0
    return ∅
  else
    foreach v ∈ V(G)
      if VC(G - v, k - 1)
        return {v} ∪ FindVC(G-v, k-1)

```

**Komplexität:** Insgesamt wird  $VC(G, k)$  höchstens  $O(n) + O(n^2) = O(n^2)$  mal aufgerufen.

- Analog kann für viele andere Optimierungsprobleme vorgegangen werden.
- Das rechtfertigt unseren Fokus auf Ja/Nein Probleme.

## Definition von NP

### NP-Probleme

**NP-Probleme:** Ein Ja/Nein-Problem ist ein NP-Problem, falls wir Ja-Instanzen mit Hilfe eines **Zertifikats** effizient überprüfen können.

**Zertifikat:** Ein Zertifikat  $t$  für eine Instanz  $x$  ist ein beliebiger Input dessen Größe  $m$  polynomiell in der Größe  $n$  von  $x$  beschränkt ist, das heißt  $m \leq p(n)$  gilt für ein Polynom  $p$ .

**Zertifizierer:** Einen Polynomialzeitalgorithmus  $C(x, t)$ , der Ja-Instanzen  $x$  mit Hilfe von Zertifikaten  $t$  überprüft, nennt man Zertifizierer.

**Anmerkung zur Notation:** NP steht für „nicht-deterministisch polynomielle“ Zeit.

Für ein NP-Problem  $X$  sagen wir auch „ $X$  ist in NP“.

Genauer gesagt, der Zertifizierer  $C(x, t)$  soll folgende Eigenschaften haben:

- Für jede Ja-Instanz  $x$  gibt es ein Zertifikat  $t$  (polynomieller Länge), welches den Zertifizierer zum akzeptieren bringt.  
(„Zertifizierer kann überzeugt werden“)
- Für keine Nein-Instanz  $x$  gibt es ein Zertifikat  $t$ , welches den Zertifizierer zum akzeptieren bringt.  
(„Zertifizierer kann nicht ausgetrickst werden“)

### Beispiel dazu: SAT und HAM-Cycle

**SAT:** Gegeben sei eine Formel  $\Phi$  in konjunktiver Normalform. Ist diese Formel erfüllbar?

**Zertifikat:** Eine Wahrheitsbelegung  $f$  für die  $n$  booleschen Variablen, die  $\Phi$  erfüllt.

**Zertifizierer:** Überprüfe, ob  $f$  die Formel  $\Phi$  erfüllt.

**Beispiel:**

$$\Phi = (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_4); \text{ Instanz } s \\ f(x_1) = \text{true}, f(x_2) = \text{true}, f(x_3) = \text{false}, f(x_4) = \text{true} \text{ Zertifikat } t$$

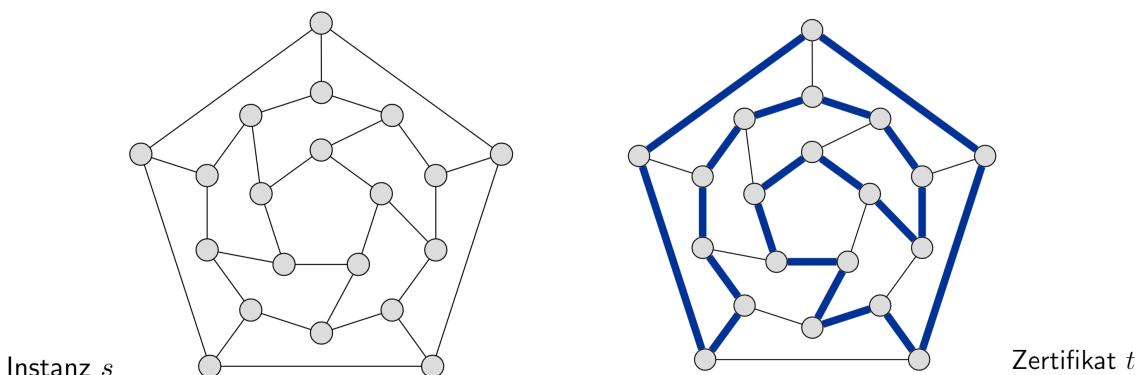
**Schlussfolgerung:** SAT ist in NP.

**HAM-CYCLE:** Gegeben sei ein ungerichteter Graph  $G$ . Existiert ein Kreis  $C$  in  $G$ , der alle Knoten von  $G$  genau einmal enthält? So ein Kreis wird als Hamiltonkreis bezeichnet.

**Zertifikat:** Ein Hamiltonkreis.

**Zertifizierer:** Überprüfe, ob der Hamiltonkreis jeden Knoten in  $V$  genau einmal enthält und dass es eine Kante zwischen jedem Paar von direkt aufeinander folgenden Knoten in dem Hamiltonkreis und auch vom ersten zum letzten Knoten gibt.

**Beispiel:**



**Schlussfolgerung:** HAM-CYCLE ist in NP.

### Quiz zur Definition:

**Frage 6:** Welche der folgenden Probleme sind in NP?

- (A) INDEPENDENT SET
- (B) Gegeben ein Graph  $G$ , finde ein kleinstes Vertex Cover von  $G$ .
- (C) Gegeben ein Graph  $G$  und  $k > 0$ . Hat jedes Vertex Cover von  $G$  mindestens  $k$  Knoten?
- (D) Gegeben die Präferenzen von  $n$  Kindern und  $n$  Gastfamilien, existiert ein Stable Matching?

- B stimmt nicht weil kein ja/nein
- C stimmt nicht, weil man nicht Existenz beweisen muss --> alle anschauen --> nicht zertifizierbar.

**Unterscheidung P und NP**

P: Ja/Nein-Probleme, für die **polynomielle Algorithmen** existieren.

NP: Ja/Nein-Probleme, für die **polynomielle Zertifizierer** existieren.

Es gilt:  $P \subseteq NP$ .

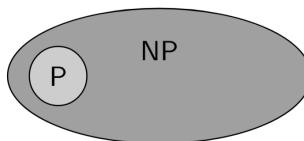
**Beweis:** Wir betrachten ein beliebiges Problem  $X$  in P.

- Nach Definition existiert ein Polynomialzeit-Algorithmus  $A(s)$ , der  $X$  löst.
- Zertifikat:  $t = \emptyset$ , Zertifizierer  $C(s, t) = A(s)$ .  $\square$

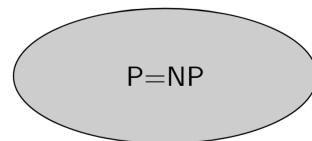
**Gilt  $P = NP$ ? [Cook 1971, Levin 1973]**

Gilt  $P = NP$ ? [Cook 1971, Levin 1973]

- Ist das Ja/Nein-Problem so leicht wie das Zertifizierungsproblem?
- Für die Beantwortung der Frage ist 1 Million US Dollar ausgeschrieben (Clay Mathematics Institute).



Falls  $P \neq NP$



Falls  $P = NP$

- **Falls ja:** Effiziente Algorithmen für Vertex Cover, Ham-Cycle, TSP\*, SAT, ...
- **Falls nein:** Keine effizienten Algorithmen für Vertex Cover, Ham-Cycle, TSP\*, SAT, ...

**Vorherrschende Meinung zu  $P = NP$ :** Wahrscheinlich "nein".

# NP-Vollständigkeit

**Definition:** Ein Ja/Nein-Problem  $Y$  ist **NP-schwer**, falls für jedes Problem  $X$  in NP gilt, dass  $X \leq_p Y$ . Das heißt, jedes NP-Problem  $X$  kann in Polynomialzeit auf  $Y$  reduziert werden. NP-schwere Probleme sind also gewissermaßen "mindestens so schwer" wie alle Probleme in NP.

**Definition:** Ein Problem  $Y$  ist **NP-vollständig**, falls es sowohl in NP liegt als auch NP-schwer ist. Die NP-vollständigen Probleme sind also gewissermaßen die "schwersten" Probleme in NP. Nach dieser Definition können nur Ja/Nein-Probleme NP-vollständig sein.

**Theorem:** Sei  $Y$  ein NP-vollständiges Problem.  $Y$  ist in polynomieller Zeit lösbar genau dann, wenn  $P = NP$ .

**Beweis:**

- ( $\Leftarrow$ ) Wenn  $P = NP$ , dann kann  $Y$  in polynomieller Zeit gelöst werden, da  $Y$  sich in NP befindet.
- ( $\Rightarrow$ ) Angenommen,  $Y$  kann in polynomieller Zeit gelöst werden. Sei  $X$  ein beliebiges Problem in NP. Da  $X \leq_p Y$ , können wir  $X$  in Polynomialzeit lösen (reduziere  $X$  auf  $Y$  in Polynomialzeit, löse  $Y$  in Polynomialzeit). Das impliziert  $NP \subseteq P$ . Wir wissen bereits, dass  $P \subseteq NP$ . Daher  $P = NP$ .

## Gibt es ein NP-vollständiges Problem?

**Allgemeine Überlegung:** Das ist nicht von vornherein klar. Es könnte z.B. mehrere schwerste NP-Probleme geben, die nicht jeweils aufeinander reduzierbar sind.

**Theorem:** SAT ist NP-vollständig. [Cook 1971, Levin 1973]

## NP-Vollständigkeit nachweisen

**Anmerkung:** Sobald wir ein erstes Problem als NP-vollständig nachgewiesen haben, fallen die anderen wie Dominosteine.

**Rezept um die NP-Vollständigkeit eines Problems  $Y$  nachzuweisen:**

- Schritt 1. Zeige dass  $Y$  in NP ist.
- Schritt 2. Wähle ein NP-vollständiges Problem  $X$ .
- Schritt 3. Beweise, dass  $X \leq_p Y$ .

**Rechtfertigung:** Wenn  $X$  ein NP-vollständiges Problem ist und  $Y$  ein Problem in NP mit der Eigenschaft  $X \leq_p Y$ , dann ist  $Y$  NP-vollständig.

**Rechtfertigung:** Wenn  $X$  ein NP-vollständiges Problem ist und  $Y$  ein Problem in NP mit der Eigenschaft  $X \leq_p Y$ , dann ist  $Y$  NP-vollständig.

**Beweis:** Sei  $W$  ein beliebiges Problem in NP.

Dann gilt  $W \leq_p X \leq_p Y$ .

■ durch Definition von NP-vollständig ■ durch Annahme

■ Durch Transitivität,  $W \leq_p Y$ .

■ Daher ist  $Y$  NP-vollständig.  $\square$

[Karp 1972] Karp hat mit einem einflussreichen Artikel wesentlich zur Verbreitung der Theorie der NP-Vollständigkeit beigetragen. Er hat 1985 dafür den Turingpreis erhalten.

**Beobachtung:** Alle Probleme in der Abbildung sind NP-vollständig und lassen sich polynomiell aufeinander reduzieren.

