

# Grundbegriffe der Theoretischen Informatik

Sommersemester 2018 - Thomas Schwentick

Teil D: Komplexitätstheorie

18: **NP** und **NP**-Vollständigkeit

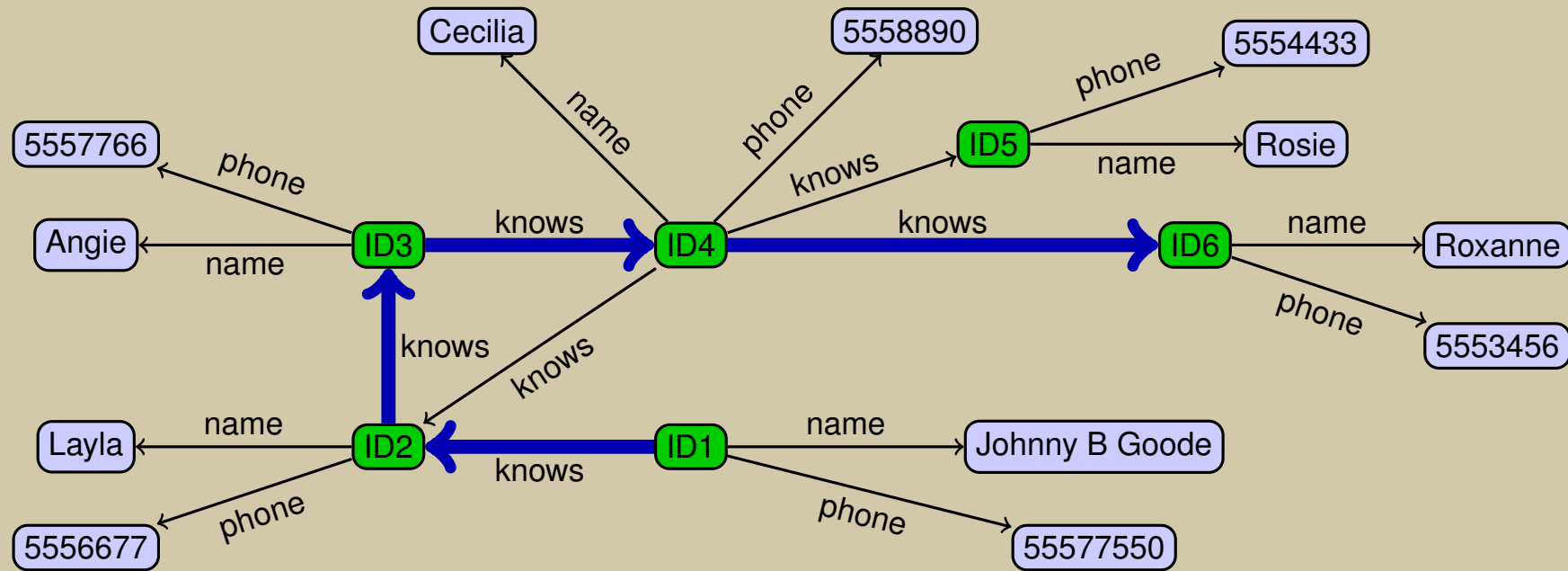
Version von: 26. Juni 2018 (12:13)

# NP-Vollständigkeit: Einleitung (1/6)

- Eine kleine Geschichte
  - ✎ nach Garey&Johnson, abgewandelt von Wim Martens
- Stellen Sie sich vor...
- Sie sollen ein Tool entwickeln, das superschnell Anfragen auf graphstrukturierten Daten auswertet
- Die Anfragen sind reguläre Ausdrücke
  - ☞ SPARQL
- Bei Eingabe eines regulären Ausdrucks  $\alpha$  soll das Tool in der Datenbank Knotenpaare  $(a, b)$  finden, die durch einen Weg verbunden sind, dessen *Label-Folge* in  $L(\alpha)$  liegt
- ...aber der Weg muss wirklich ein Weg sein
  - ☞ kein Knoten doppelt

## NP-Vollständigkeit: Einleitung (2/6)

### Beispiel: Graph-DB und Anfrage



- **Anfrage:** Gibt es einen Weg von ID1 zu ID6, der zum regulären Ausdruck  $\alpha = (\text{knows})^*$  passt?

## NP-Vollständigkeit: Einleitung (3/6)

- Sie versuchen einen Algorithmus zu entwerfen, merken aber, dass Sie kein schnelles Verfahren für dieses Problem finden
- Sie haben schon Effizienzprobleme beim regulären Ausdruck (knows knows)\*
- Sie können das Problem kaum besser lösen, als durch Ausprobieren aller Wege, und das sind sehr, sehr viele...
- Heute ist der Termin für die Vorstellung Ihrer Ergebnisse
- **Was tun?**

## NP-Vollständigkeit: Einleitung (4/6)

- Option 1:

**„Ich kann keine effiziente Methode finden. Ich glaube, ich bin einfach unfähig.“**

- Wäre das klug?

## NP-Vollständigkeit: Einleitung (5/6)

- **Option 2:**  
„Ich kann keine effiziente Methode finden, weil es keine solche Methode gibt.“
- Das wäre ideal. Damit hätten Sie auch gleich eine Million Dollar verdient...

## NP-Vollständigkeit: Einleitung (6/6)

- **Option 3:**  
„Ich kann keine effiziente Methode finden, aber alle diese berühmten Informatiker können es auch nicht.“
- Besser als nichts. Damit das klappt, brauchen wir: **NP**-Vollständigkeit

# Übersicht

- In diesem Kapitel werden wir
  - eine Reihe schwieriger Berechnungsprobleme kennen lernen,
  - ihre Ähnlichkeit näher unter die Lupe nehmen,
  - einen Teil ihrer Ähnlichkeit durch die Definition der Komplexitätsklasse **NP** formalisieren, und
  - ein noch stärkeres Maß ihrer Ähnlichkeit durch den Begriff der **NP**-Vollständigkeit formalisieren



# Inhalt

## ▷ 18.1 Beispiele schwieriger Berechnungsprobleme

18.2 **NP**

18.3 **NP**-Vollständigkeit

# Schwierige Berechnungsprobleme: Rucksack

## Beispiel

- In zwei Monaten startet die nächste Rakete zur Raumstation
- Die Weltraumagentur ist etwas knapp bei Kasse und bietet deshalb kommerziellen Forschungsinstituten an, wissenschaftliche Experimente in der Raumstation durchzuführen
- Die Rakete kann noch maximal 645 kg zusätzliche Last für Experimente mitnehmen
- Die Agentur erhält von den Instituten verschiedene Angebote, in denen steht,
  - wieviel sie für Transport und Durchführung des Experiments zu zahlen bereit sind und
  - wie schwer die Geräte für ihr Experiment sind
- Welche Experimente soll die Weltraumagentur auswählen, um den Gewinn zu maximieren?

Objekt-Nr.	1	2	3	4	5	6	7	8
Gewicht	153	54	191	66	239	137	148	249
Gewinn (1000)	232	73	201	50	141	79	48	38

☞ Rene Beier, Saarbrücken, Berthold Vöcking, Aachen

- Dieses Beispiel führt zu einer Eingabe für das folgende **Rucksackproblem**:

## Definition (KNAPSACKO)

**Gegeben:** Gewichtsschranke  $G$  und  $m$  Gegenstände repräsentiert durch

- Werte  $w_1, \dots, w_m$  und
- Gewichte  $g_1, \dots, g_m$ ,


☞ alle Zahlen aus  $\mathbb{N}$

**Gesucht:**  $I \subseteq \{1, \dots, m\}$ , so dass  $\sum_{i \in I} w_i$  maximal ist und

$$\sum_{i \in I} g_i \leq G \text{ gilt}$$

- Informell: gesucht ist eine Menge von Gegenständen mit maximalem Gesamtwert und Gewicht  $\leq G$

# Schwierige Berechnungsprobleme: Graphfärbung

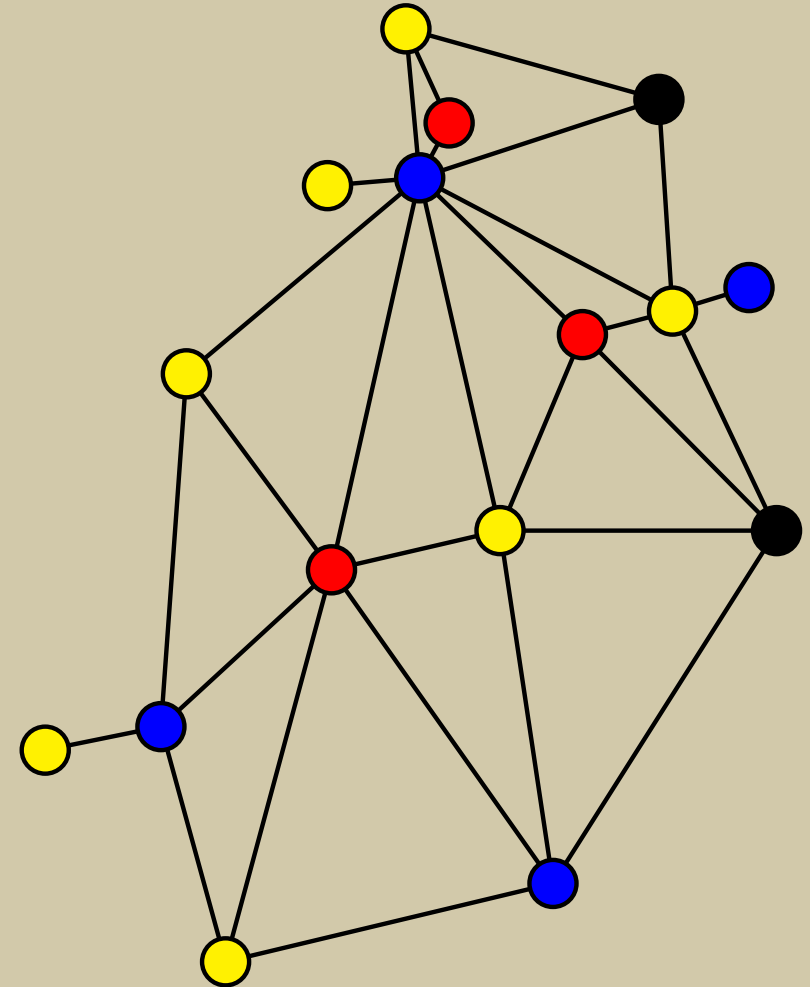
- **Landkartenfärbung:** Lassen sich die Länder einer gegebenen Landkarte mit einer gegebenen Anzahl von Farben so färben, dass benachbarte Länder verschiedene Farben haben?
- Beispiel: lässt sich die Karte der deutschen Bundesländer in dieser Art mit 3 Farben färben? Nein!
- 4 Farben genügen immer (wenn alle Länder zusammenhängen)  Vierfarbensatz
- Das Problem der Landkartenfärbung lässt sich zurückführen auf das allgemeinere Problem, die Knoten eines Graphen zu färben

## Definition (COL)

**Gegeben:** Ungerichteter Graph  $G$ , Zahl  $k$

**Frage:** Lassen sich die Knoten von  $G$  mit  $k$  Farben **zulässig färben**, also so, dass durch Kanten verbundene Knoten verschiedene Farben haben?

## Beispiel



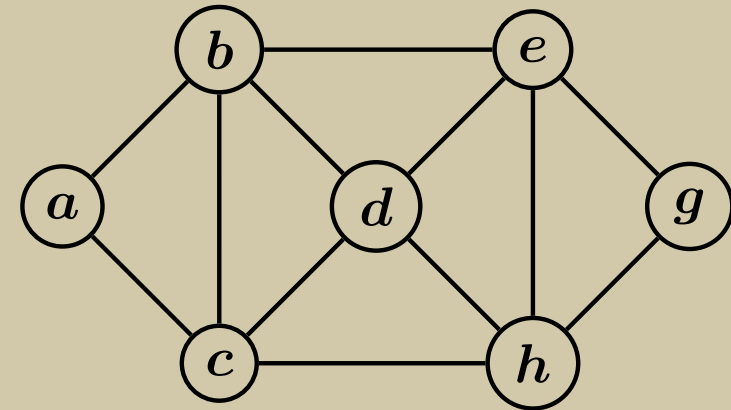
# Graphentheorie: Wiederholung

- Wir wiederholen sicherheitshalber einige Grundbegriffe aus der Graphentheorie:

## Definition (Kantenfolge, Weg)

- Sei  $G = (V, E)$  ein Graph
- Sei  $v_0, \dots, v_n \in V$  eine Folge von Knoten von  $G$  mit der Eigenschaft, dass  $e_i \stackrel{\text{def}}{=} (v_{i-1}, v_i)$  für jedes  $i \in \{1, \dots, n\}$  eine Kante von  $G$  ist
- Dann heißt  $e_1, \dots, e_n$  eine Kantenfolge von  $G$
- Ist  $v_0 = v_n$ , so heißt die Kantenfolge geschlossen
- Eine Kantenfolge ist ein Weg oder Pfad, wenn die Knoten  $v_0, \dots, v_n$  paarweise verschieden sind  
(es darf allerdings  $v_0 = v_n$  gelten)
- Einen geschlossenen Weg nennen wir einen Kreis

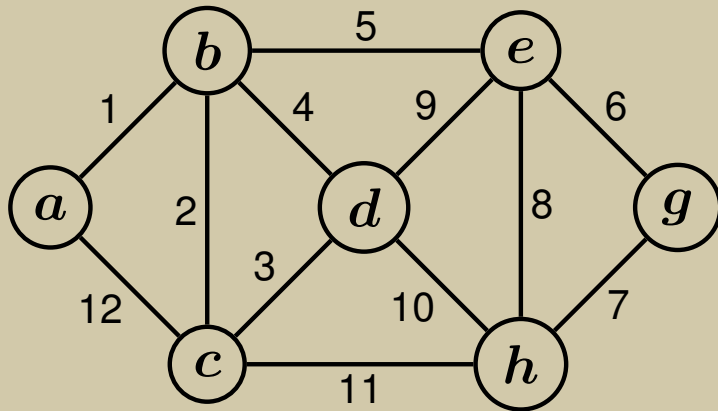
## Beispiel



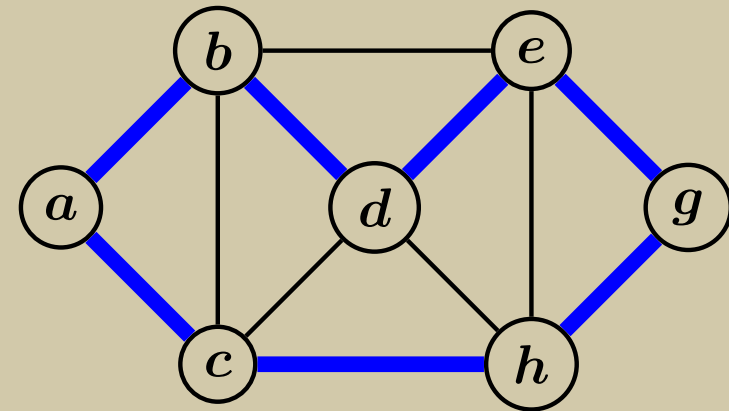
- $(a, b), (b, e), (e, d), (d, b), (b, c)$  ist eine Kantenfolge von  $G$
- $(a, b), (b, e), (e, d), (d, b), (b, c), (c, a)$  ist eine geschlossene Kantenfolge von  $G$
- $(a, b), (b, e), (e, d), (d, c)$  ist ein Weg von  $G$
- $(a, b), (b, e), (e, d), (d, c), (c, a)$  ist ein Kreis von  $G$

# Schwierige Berechnungsprobleme: Hamilton-Kreise

## Beispiel



## Beispiel



## Definition (EULERCYCLE)

**Gegeben:** Ungerichteter Graph  $G$

**Frage:** Gibt es eine geschlossene Kantenfolge in  $G$ , die *jede Kante* genau einmal besucht?

## Fakt

- Ein zusammenhängender Graph  $G$  hat genau dann einen Euler-Kreis, wenn jeder Knoten geraden Grad hat

👉 gerade viele Nachbarknoten

→ Das ist in polynomieller Zeit testbar

## Definition (HAMILTONCYCLE)

**Gegeben:** Ungerichteter Graph  $G$

**Frage:** Gibt es einen geschlossenen Weg in  $G$ , der **jeden Knoten** genau einmal besucht?

- Für HAMILTONCYCLE ist kein Algorithmus mit polynomieller Laufzeit bekannt
- ✎ Ein Euler-Kreis ist meistens kein Kreis sondern nur eine geschlossene Kantenfolge, die Bezeichnung Euler-Kreis ist aber allgemein üblich

# Schwierige Berechnungsprobleme: Das Cliques-Problem

- Zwei Knoten  $u, v$  eines ungerichteten Graphen  $G = (V, E)$  heißen benachbart, wenn  $(u, v) \in E$
- Eine  $k$ -Clique ist eine Menge  $C$  von  $k$  Knoten, die paarweise benachbart sind

- Das Cliques-Problem hat viele Anwendungen, z.B.
  - im Data Mining
  - in der Bioinformatik

## Definition (CLIQUEO)

**Gegeben:** Graph  $G = (V, E)$

**Gesucht:** Maximale Clique in  $G$ ,  
d.h.:  
maximale Menge  $C$  von Knoten, die paarweise benachbart sind

## Definition (CLIQUE)

**Gegeben:** Graph  $G = (V, E)$ ,  
Zahl  $k$

**Frage:** Gibt es in  $G$  eine Clique mit  $k$  Knoten?

# Schwierige Berechnungsprobleme: AL-Erfüllbarkeit

- **Aussagenlogische Formeln:**

- $x_1, x_2, x_3, \dots$  seien Variablen
- Jedes  $x_i$  ist eine aussagenlogische Formel
- Ist  $\varphi$  eine aussagenlogische Formel, so auch  $\neg\varphi$
- Sind  $\varphi_1, \varphi_2$  aussagenlogische Formeln, so auch  $\varphi_1 \wedge \varphi_2$  und  $\varphi_1 \vee \varphi_2$

- Eine Wahrheitsbelegung  $\alpha : \{x_1, x_2, \dots\} \rightarrow \{0, 1\}$  ordnet jeder Variablen einen Wert zu
- $\alpha \models \varphi$ : Die Formel  $\varphi$  erhält durch die Wahrheitsbelegung  $\alpha$  den Wert 1
- Eine Formel  $\varphi$  ist **erfüllbar**, wenn es ein  $\alpha$  gibt mit  $\alpha \models \varphi$

## Beispiel

- $(x_3 \vee x_1) \wedge (\neg x_2 \vee ((x_3 \wedge \neg x_1) \vee (x_2 \vee \neg x_1)))$

- Wir beschränken uns auf Formeln in **konjunktiver Normalform** (KNF):

$$(x_3 \vee x_1) \wedge (\neg x_2 \vee x_3 \vee x_2) \wedge (\neg x_2 \vee \neg x_3 \vee x_2) \wedge (\neg x_2 \vee x_3) \wedge (\neg x_2 \vee \neg x_3)$$

- Die  $x_i$  und  $\neg x_i$  heißen **Literale**
- Die disjunktiven Teilformeln heißen **Klauseln**
- Eine KNF-Formel ist in **3-KNF**, wenn jede Klausel genau drei Literale enthält
  - Das selbe Literal darf mehrfach in einer Klausel vorkommen

## Definition (SAT)

**Gegeben:** Aussagenlogische Formel  $\varphi$  in KNF

**Frage:** Ist  $\varphi$  erfüllbar?

## Definition (3-SAT)

**Gegeben:** Aussagenlogische Formel  $\varphi$  in 3-KNF

**Frage:** Ist  $\varphi$  erfüllbar?

# Inhalt

18.1 Beispiele schwieriger Berechnungsprobleme

▷ **18.2 NP**

18.3 **NP**-Vollständigkeit



# Eigenschaften der betrachteten Probleme (1/2)

- Nur für zwei der in diesem und im letztem Kapitel erwähnten Probleme ist ein Algorithmus mit polynomieller Laufzeit bekannt:
  - EULERCYCLE
  - MINSPANNINGTREEO
- Für die folgenden Probleme ist kein solcher Algorithmus bekannt:
  - TSP
  - HAMILTONCYCLE
  - SAT
  - COL
  - KNAPSACK
  - 3-SAT
  - CLIQUE
- Wir werden jetzt untersuchen, wie „ähnlich“ diese Probleme zueinander sind

## Eigenschaften der betrachteten Probleme (2/2)

- Bei allen betrachteten Problemen gibt es für jede Eingabe eine Menge von *Lösungskandidaten* und es geht um die Frage, ob einer dieser Lösungskandidaten eine *Lösung* ist
- *Lösungskandidaten* sind für:
  - TSP, HAMILTONCYCLE:  
alle Permutationen der Knotenmenge
  - SAT, 3-SAT: alle Wahrheitsbelegungen
  - COL: alle Färbungen mit  $k$  Farben
  - KNAPSACK: alle Teilmengen der Gegenstandsmenge
  - CLIQUE: alle Mengen mit  $k$  Knoten
- *Lösungen* sind für:
  - SAT: Wahrheitsbelegungen, die alle Klauseln wahr machen
  - COL: Färbungen, die benachbarte Knoten verschieden färben
  - CLIQUE: Mengen von  $k$  Knoten, die alle paarweise miteinander durch Kanten verbunden sind

## Polynomielle Lösungskandidaten: NP (1/2)

- Die betrachteten Entscheidungsprobleme haben folgende Eigenschaften:
  - (1) Sie haben einen Suchraum von Lösungskandidaten
  - (2) Die Lösungskandidaten sind polynomiell groß in der Eingabe
  - (3) Jeder einzelne Lösungskandidat kann in polynomieller Zeit überprüft werden
- Verständnisfrage: Hat PCP auch die Eigenschaften (1)-(3)?
- Wir verwenden die Eigenschaften (1) - (3) zur Definition der Komplexitätsklasse **NP**

## Polynomielle Lösungskandidaten: NP (2/2)

- Wir betrachten im Folgenden Turingmaschinen, die Paare  $(w, y)$  von Strings über  $\Sigma$  als Eingabe verarbeiten
  - Zur Erinnerung: formal erhält die TM also eine Eingabe der Form  $w\#y$  mit  $w, y \in \{0, 1\}^*$
- Die Rechenzeit von  $M$  bei Eingabe  $(w, y)$  bezeichnen wir mit  $\underline{t_M(w, y)}$

### Definition (Nichtdet. Akzeptieren/Entscheiden)

- Sei  $\Sigma = \{0, 1\}$  und sei  $M$  eine Turingmaschine
- Wir sagen, dass  $M$  einen String  $w \in \Sigma^*$  nicht-deterministisch akzeptiert, wenn es einen String  $y \in \Sigma^*$  gibt, so dass  $M$  bei Eingabe  $(w, y)$  akzeptiert  
*☞  $y$  ist die **Zusatzeingabe***
- $M$  entscheidet eine Sprache  $L \subseteq \Sigma^*$  nicht-deterministisch, falls für alle Strings  $w \in \Sigma^*$  äquivalent sind:
  - $w \in L$
  - $M$  akzeptiert  $w$  nichtdeterministisch

### Definition ( $\mathbf{NTIME}(T)$ , $\mathbf{NP}$ )

- Sei  $\Sigma = \{0, 1\}$ ,  $T : \mathbb{N} \rightarrow \mathbb{R}$
- $\mathbf{NTIME}(T) \stackrel{\text{def}}{=} \text{Klasse aller } \underline{L \subseteq \Sigma^*}$ , für die es eine TM  $M$  gibt, die  $L$  nichtdeterministisch entscheidet

$$\bullet \quad \underline{\mathbf{NP}} \stackrel{\text{def}}{=} \bigcup_{p \text{ Polynom}} \mathbf{NTIME}(p)$$

## Bemerkungen: NP (1/2)

- Die in der Definition von **NP** verwendete Zusatzeingabe  $y$  entspricht gerade den *Lösungskandidaten* in den betrachteten schwierigen Entscheidungsproblemen

### ➡ SAT, 3-SAT $\in$ **NP**

- Zusatzeingabe: Wahrheitsbelegung der in der gegebenen Formel vorkommenden Variablen

### ➡ 3-COL, COL $\in$ **NP**

- Zusatzeingabe: Knotenfärbung

### ➡ HAMILTONCYCLE, TSP $\in$ **NP**

- Zusatzeingabe: Knotenfolge bzw. „Reisefunktion“

### ➡ KNAPSACK $\in$ **NP**

- Zusatzeingabe: Menge von Gegenständen

### ➡ CLIQUE $\in$ **NP**

- Zusatzeingabe: Menge von Knoten

- Zu beachten: bei der Definition von **NTIME**( $T$ ) hängt die Zeitschranke nur von der (Länge der) *Eingabe*  $w$ , aber *nicht* von der *Zusatzeingabe*  $y$  ab

- ➡ Es genügt also, Zusatzeingaben der Länge  $\leq T(|w|)$  zu betrachten, da die TM mehr Zeichen der Zusatzeingabe gar nicht lesen kann

## Bemerkungen: NP (2/2)

- Oft wird zur Definition der Klasse **NP** das Berechnungsmodell *nichtdeterministischer Turingmaschinen* verwendet
- Nichtdeterministische TMs können, wie NFAs, in derselben Situation (Zustand, gelesenes Zeichen) mehrere Transitionen haben
- Die nichtdeterministische Vorgehensweise solcher NTMs lässt sich intuitiv als „Raten“ auffassen:
  - Wenn  $w \in L$ , dann gibt es eine Berechnung der NTM, die „richtig rät“ und akzeptiert
  - Wenn  $w \notin L$ , dann lehnt die NTM ab, unabhängig davon, was sie rät
- Die Zusatzeingabe in unserer Definition entspricht den „geratene Bits“ in der NTM-Definition
- Die hier gegebene Definition von **NP** betont stärker den Aspekt des Überprüfens von Lösungskandidaten polynomieller Größe und vermeidet „ratende Algorithmen“

## Bemerkungen: P vs. NP

- Lösungen finden:

	6		2			7		
					4			9
2		3				5		
	2			3				8
			4		7			
1				8			6	
		5				8		4
9			1					
		4			2		9	

- Damit ein Problem in **P** ist, muss es einen Algorithmus geben, der effizient eine Lösung *findet*
- Zum Beispiel ist HORNSAT  $\in$  **P**: der Markierungsalgorithmus findet effizient eine erfüllende Belegung, wenn es eine gibt

- Lösungskandidaten überprüfen:

4	6	8	2	9	5	7	3	1
7	5	1	3	6	4	2	8	9
2	9	3	7	1	8	5	4	6
5	2	9	6	3	1	4	7	8
8	3	6	4	2	7	9	1	5
1	4	7	5	8	9	3	6	2
3	1	5	9	7	6	8	2	4
9	8	2	1	4	3	6	5	7
6	7	4	8	5	2	1	9	3

- Damit ein Problem in **NP** ist, genügt ein Algorithmus, der effizient *überprüft*, ob ein Lösungskandidat eine Lösung ist
- SAT  $\in$  **NP**: es kann effizient getestet werden, ob eine gegebene Belegung die gegebene Formel wahr macht

# P vs NP vs. EXPTIME

## Proposition 18.1

$$\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{EXPTIME}$$

## Beweisskizze

- $\mathbf{P} \subseteq \mathbf{NP}$ : die Zusatzeingabe kann einfach ignoriert werden...
  - $\mathbf{NP} \subseteq \mathbf{EXPTIME}$ :
    - Sei  $M$  eine TM, die  $L$  nichtdeterministisch mit polynomieller Zeitschranke  $p$  entscheidet
    - **EXPTIME**-Algorithmus:
      - \* Simuliere  $M$  für alle möglichen Zusatzeingaben  $y$  der Länge  $\leq p(|w|)$
- ➔ maximal  $2^{p(|w|)}$  Teilberechnungen zu je  $\leq p(|w|)$  Schritten

- Ob auch die Umkehrung ( $\mathbf{NP} \subseteq \mathbf{P}$ ) der ersten Aussage gilt, ist das größte offene Problem der Theoretischen Informatik



# Inhalt

18.1 Beispiele schwieriger Berechnungsprobleme

18.2 **NP**

▷ **18.3 NP-Vollständigkeit**

# Ähnlichkeit schwieriger Optimierungsprobleme

- Wir werden sehen: die genannten Probleme sind sich noch viel ähnlicher:
  - entweder lassen sie sich alle in polynomieller Zeit lösen oder keines
- Wir betrachten diesen Zusammenhang zunächst anhand von SAT und der folgenden Einschränkung von COL:

## Definition (3-COL)

**Gegeben:** Ungerichteter Graph  $G$

**Frage:** Lassen sich die Knoten von  $G$  mit 3 Farben zulässig färben?

## Proposition 18.2

- Die folgenden Aussagen sind äquivalent:
  - (a) SAT lässt sich in polynomieller Zeit lösen
  - (b) 3-COL lässt sich in polynomieller Zeit lösen

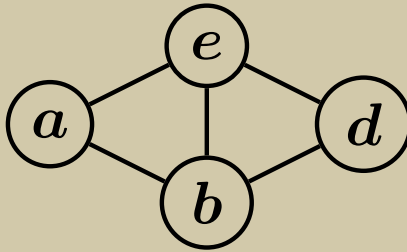
## Beweisskizze

- Wir zeigen hier nur: „(a)  $\Rightarrow$  (b)“
  - „(b)  $\Rightarrow$  (a)“ zeigen wir dann in Kapitel 19
- Wir nehmen an,  $A$  ist ein Algorithmus, der in Zeit  $|\varphi|^k$  entscheidet, ob eine gegebene KNF-Formel  $\varphi$  erfüllbar ist
- Wir beschreiben ein Verfahren, das unter Verwendung von  $A$  in polynomieller Zeit entscheidet, ob ein gegebener Graph  $G$  3-färbbar ist

## Beweis von Proposition 18.2: „(a) $\Rightarrow$ (b)“ (1/4)

### Illustration des Beweises

- Sei  $G$  der folgende Graph:



- $\varphi_G =$   
 $(x_{a1} \vee x_{a2} \vee x_{a3}) \wedge$   
 $(x_{b1} \vee x_{b2} \vee x_{b3}) \wedge$   
 $(x_{d1} \vee x_{d2} \vee x_{d3}) \wedge$   
 $(x_{e1} \vee x_{e2} \vee x_{e3}) \wedge$   
 $(\neg x_{a1} \vee \neg x_{b1}) \wedge (\neg x_{a2} \vee \neg x_{b2}) \wedge (\neg x_{a3} \vee \neg x_{b3}) \wedge$   
 $(\neg x_{a1} \vee \neg x_{e1}) \wedge (\neg x_{a2} \vee \neg x_{e2}) \wedge (\neg x_{a3} \vee \neg x_{e3}) \wedge$   
 $(\neg x_{b1} \vee \neg x_{d1}) \wedge (\neg x_{b2} \vee \neg x_{d2}) \wedge (\neg x_{b3} \vee \neg x_{d3}) \wedge$   
 $(\neg x_{b1} \vee \neg x_{e1}) \wedge (\neg x_{b2} \vee \neg x_{e2}) \wedge (\neg x_{b3} \vee \neg x_{e3}) \wedge$   
 $(\neg x_{d1} \vee \neg x_{e1}) \wedge (\neg x_{d2} \vee \neg x_{e2}) \wedge (\neg x_{d3} \vee \neg x_{e3})$

- Zulässige Färbung:  $c(a) = c(d) = 1, c(b) = 2, c(e) = 3$
- Korrespondierende Wahrheitsbelegung:
  - $\alpha(x_{a1}) = \alpha(x_{b2}) = \alpha(x_{d1}) = \alpha(x_{e3}) = 1$
  - $\alpha(x) = 0$  für alle übrigen Variablen  $x$

## Beweis von Proposition 18.2: „(a) $\Rightarrow$ (b)“ (2/4)

### Beweisskizze (Forts.)

- Sei also  $G = (V, E)$  eine Eingabe für 3-COL
- Wir konstruieren eine Formel  $\varphi_G$ , so dass gilt:  
 $\varphi_G$  erfüllbar  $\iff G$  3-färbbar  $(*)$
- $\varphi_G$  hat für jeden Knoten  $v \in V$  und jedes  $j \in \{1, 2, 3\}$  eine Variable  $x_{vj}$
- Intention:
  - Die Farben heißen **1, 2, 3**
  - $\alpha(x_{vj}) = 1 \iff$   
 $\alpha$  repräsentiert eine Färbung, in der Knoten  $v$  die Farbe  $j$  hat

### Beweisskizze (Forts.)

- Wir verwenden folgende Teilformeln:
  - $\psi_v \stackrel{\text{def}}{=} (x_{v1} \vee x_{v2} \vee x_{v3})$   
☞  $v$  hat (mindestens) eine Farbe
  - $\chi_{uv} \stackrel{\text{def}}{=} (\neg x_{u1} \vee \neg x_{v1}) \wedge (\neg x_{u2} \vee \neg x_{v2}) \wedge (\neg x_{u3} \vee \neg x_{v3})$   
☞  $u$  und  $v$  haben verschiedene Farben
- Sei schließlich
$$\varphi_G \stackrel{\text{def}}{=} \bigwedge_{v \in V} \psi_v \wedge \bigwedge_{(u,v) \in E} \chi_{uv}$$

✎ Für jede ungerichtete Kante  $(u, v)$  verwenden wir nur die Formel  $\chi_{uv}$  oder die Formel  $\chi_{vu}$ , aber nicht beide

## Beweis von Proposition 18.2: „(a) $\Rightarrow$ (b)“ (3/4)

### Beweisskizze (Forts.)

- Zu zeigen:  $\varphi_G$  erfüllbar  $\iff G$  3-färbbar

- „ $\Leftarrow$ “:

- Sei  $c : V \rightarrow \{1, 2, 3\}$  eine zulässige Färbung und

$$\alpha(x_{vi}) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{falls } c(v) = i \\ 0 & \text{sonst} \end{cases}$$

- Behauptung:  $\alpha \models \varphi_G$
- Denn:
  - \*  $\alpha \models \psi_v$  für jeden Knoten  $v$  und
  - \*  $\alpha \models \chi_{uv}$  für jedes  $(u, v) \in E$ ,  
da  $c(u) \neq c(v)$

### Beweisskizze (Forts.)

- „ $\Rightarrow$ “:

- Sei  $\alpha$  erfüllende Belegung von  $\varphi_G$

➡ Für jedes  $v$  gilt  $\alpha \models \psi_v$

- Sei  $c(v) \stackrel{\text{def}}{=} \text{kleinstes } j \text{ mit}$

$$\alpha(x_{vj}) = 1$$

- Behauptung:  $c$  ist zulässige Färbung
- Denn: wenn  $(u, v) \in E$ ,  
dann  $\alpha \models \chi_{uv}$
- Also muss  $c(u) \neq c(v)$  sein

## Beweis von Proposition 18.2: „(a) $\Rightarrow$ (b)“ (4/4)

### Beweisskizze (Forts.)

- Zu beachten: Zu einem gegebenen  $G = (V, E)$  hat  $\varphi_G$  genau  $|V| + 3|E|$  Klauseln
- Ist  $n$  die Größe der Kodierung von  $G$  (beispielsweise durch Adjazenzlisten), so ist also  $|\varphi_G| \leq cn$ , für eine Konstante  $c$
- Offensichtlich kann  $\varphi_G$  in (polynomieller) Zeit  $n^\ell$ , für ein geeignetes  $\ell$ , berechnet werden
- Also kann 3-COL wie folgt gelöst werden:
  - Bei Eingabe  $G$  berechne  $\varphi_G$
  - Teste mit Hilfe von Algorithmus  $A$ , ob  $\varphi_G$  erfüllbar ist
- Die Gesamtlaufzeit ist  $\leq n^\ell + (cn)^k = \mathcal{O}(n^{\max(\ell, k)})$

➡ Behauptung

- Um Proposition 18.2 zu beweisen, hätte es genügt zu zeigen, dass gilt:  
$$\varphi_G \text{ erfüllbar} \iff G \text{ 3-färbbar}$$
- Der Beweis zeigt mehr als das:
  - er konstruiert aus jeder erfüllenden Belegung von  $\varphi_G$  eine korrekte 3-Färbung von  $G$  und umgekehrt
- Das ist für diese Art von Beweisen typisch

# Polynomielle Reduktionen

- Schauen wir nochmal auf den Beweis von Proposition 18.2 „(a)  $\Rightarrow$  (b)“

- Sei  $f$  die Funktion  $G \mapsto \varphi_G$

- Dann gilt:

$$G \in 3\text{-COL} \iff f(G) \in \text{SAT}$$

- ➔  $f$  ist eine Reduktion von 3-COL auf SAT
  - Also:  $3\text{-COL} \leq \text{SAT}$

- Und:  $f$  ist in polynomieller Zeit berechenbar

## Definition (Polynomielle Reduktion, $\leq_p$ )

- Eine totale Funktion  $f$  heißt polynomielle Reduktion von  $L$  auf  $L'$ , falls
  - (1)  $f$  eine Reduktion ist, also für alle  $w \in \Sigma^*$  gilt:
$$w \in L \iff f(w) \in L', \text{ und}$$
  - (2)  $f$  in polynomieller Zeit berechenbar ist
- $L$  heißt polynomiell reduzierbar auf  $L'$ , falls es eine polynomielle Reduktion von  $L$  auf  $L'$  gibt
- Schreibweise:  $L \leq_p L'$

- Also:  $3\text{-COL} \leq_p \text{SAT}$

# Abschluss unter Reduktionen

## Proposition 18.3

- Seien  $L, L' \subseteq \Sigma^*$  und gelte:  $L \leq_p L'$
- Dann gelten:
  - (a) wenn  $L' \in \mathbf{P}$  dann  $L \in \mathbf{P}$
  - (b) wenn  $L' \in \mathbf{NP}$  dann  $L \in \mathbf{NP}$

## Beweisskizze für (a)

- Sei  $M'$  TM, die  $L'$  mit Zeitschranke  $n^k$  entscheidet
- Sei  $M_f$  eine TM, die mit Zeitschranke  $n^\ell$  eine Reduktion  $f$  berechnet, so dass gilt:
$$w \in L \iff f(w) \in L'$$

- Wir konstruieren eine TM  $M$ , die  $L$  entscheidet, indem sie bei Eingabe  $w$  wie folgt vorgeht:  $\oplus$ 
  - Berechne  $f(w)$   $\Rightarrow$  durch TM  $M_f$
  - Teste  $f(w) \in L'$   $\Rightarrow$  durch TM  $M'$
  - Falls ja, akzeptiere, falls nein, lehne ab
- Laufzeit:  $\leq |w|^\ell + |f(w)|^k \leq |w|^\ell + |w|^{k\ell}$

$\Rightarrow L \in \mathbf{P}$

## Beweisskizze für (b)

- Der Beweis für (b) ist eine Erweiterung des Beweises von (a)
- Wir nehmen an, dass  $M'$  die Sprache  $L'$  nichtdeterministisch entscheidet
- Dann hat  $M$  ebenfalls eine Zusatzeingabe  $y$  und testet in der zweiten Phase, ob  $(f(w), y) \in L'$  ist
- Der Rest ist analog

## Folgerung 18.4

- Seien  $L, L' \subseteq \Sigma^*$  und gelte  $L \leq_p L'$
- $\Rightarrow$  Wenn  $L \notin \mathbf{P}$  dann  $L' \notin \mathbf{P}$



# NP-schwierige Probleme

## Definition (NP-schwierig)

- Eine Sprache  $L$  heißt NP-schwierig, falls für alle  $L' \in \mathbf{NP}$  gilt:  $L' \leq_p L$
- Gibt es überhaupt NP-schwierige Probleme?

## Proposition 18.5

- TM-HALT ist NP-schwierig

## Beweisskizze

- Sei  $L \in \mathbf{NP}$
- ➡  $L \in \mathbf{EXPTIME}$
- Sei  $M$  TM, die  $L$  in exponentieller Zeit entscheidet
- Sei  $M'$  eine TM, die  $M$  bei Eingabe  $w$  simuliert und
  - akzeptiert, wenn  $M(w)$  akzeptiert, aber
  - endlos läuft, wenn  $M(w)$  ablehnt
- Sei  $f(w) \stackrel{\text{def}}{=} (M', w)$
- ➡  $w \in L \iff (M', w) \in \text{TM-HALT}$
- Klar:  $f$  ist in polynomieller Zeit berechenbar
- ☞  $M'$  ist fest!
- ➡  $f$  ist eine polynomielle Reduktion von  $L$  auf TM-HALT
- ➡ Behauptung

# NP-vollständige Probleme

- Das zeigt zwar, dass es **NP**-schwierige Probleme gibt, aber das Halteproblem ist für unsere jetzigen Zwecke nicht interessant...
- Interessanter wären **NP**-schwierige Probleme **innerhalb von NP**

## Definition (**NP**-vollständig)

- Eine Sprache  $L$  heißt **NP-vollständig**, falls  $L$  **NP**-schwierig ist *und*  $L \in \mathbf{NP}$  gilt
- Die **NP**-vollständigen Probleme sind also gewissermaßen die schwierigsten Probleme in **NP**


# NP-Vollständigkeit: Bedeutung

## Satz 18.6

- Sei  $L$  eine **NP**-vollständige Sprache
  - (a) Falls  $L \in \mathbf{P}$ , so ist  $\mathbf{P} = \mathbf{NP}$ 
    - insbesondere sind dann auch alle anderen **NP**-vollständigen Sprachen in  $\mathbf{P}$
  - (b) Falls  $L \notin \mathbf{P}$ , so ist  $\mathbf{P} \neq \mathbf{NP}$ 
    - und alle anderen **NP**-vollständigen Sprachen sind auch nicht in  $\mathbf{P}$

## Beweisskizze

- (a)
  - Sei  $L' \in \mathbf{NP}$
  - Da  $L$  **NP**-vollständig ist, gilt  $L' \leq_p L$
  - ➔  $L' \in \mathbf{P}$  (gemäß Proposition 18.3)
- (b)
  - $L \notin \mathbf{P} \Rightarrow \mathbf{P} \neq \mathbf{NP}$  ist trivial
  - Dass dann alle anderen **NP**-vollständigen Sprachen auch nicht in  $\mathbf{P}$  sind, folgt aus (a)

- Das „Schicksal“ der „ $\mathbf{P}$  vs. **NP**“-Frage hängt also an jedem einzelnen **NP**-vollständigen Problem
- Es stellt sich aber die Frage:
  - Gibt es überhaupt **NP**-vollständige Probleme?  Nächstes Kapitel

# Zusammenfassung

- Wir haben einige Berechnungsprobleme kennen gelernt, für die keine Algorithmen mit polynomieller Laufzeit bekannt sind
- **NP** ist die Klasse der Probleme, für die sich in polynomieller Zeit testen lässt, ob ein gegebener Lösungskandidat eine Lösung ist
- Die **NP**-vollständigen Probleme sind die schwierigsten Probleme in **NP**:
  - Jedes **NP**-Problem lässt sich polynomiell auf jedes **NP**-vollständige Problem reduzieren
- Ob  $P = NP$  ist, ist die größte offene Frage der Theoretischen Informatik