Komplexitätstheorie

WiSe 2011/12 in Trier

Henning Fernau
Universität Trier
fernau@uni-trier.de

Komplexitätstheorie Gesamtübersicht

- Organisatorisches / Einführung
  Motivation / Erinnerung / Fragestellungen

- Diskussion verschiedener Komplexitätsklassen:
  Zeitkomplexität
  Platzkomplexität

- zugehörige Reduktionsbegriffe

- vollständige Probleme

- Anpassung von Klassenbegriffen und Reduktionen

Organisatorisches

Vorlesung: Montags 10-12 Uhr, H 6; Vorschlag ab 2. SW: 10.05-11.35
Zusätzlich in der ersten SW:
Freitag 10-12 Uhr, HZ 201

Übungen (Henning Fernau): Freitags 10-12 Uhr, HZ 201;
Beginn 2. Semesterwoche

Meine Sprechstunde: DO, 13-14 Uhr

Kontakt: fernau@uni-trier.de

Hausaufgaben / Schein ?! n.V. (Master ?! $\rightarrow$ mündliche Prüfung)

Motivation

Theoretische Informatik (ThI) versucht ganz allgemein mathematische Grundlagen der Informatik bereitzustellen.

Konkrete Ausprägungen von ThI müssen in diesem Kontext gesehen werden.

Ursprüngliche (Informatik-)Motivation
für die Komplexitätstheorie liegt in der Frage, wie schwierig denn Programme sein müssen, um ein bestimmtes Problem zu lösen.

Einordnung

Formale Sprachen (FS) und Komplexitätstheorie (KT) sind die klassischen Gebiete der Theoretischen Informatik, und zwar unabhängig von nationalen Moden.

Beispiel Deutschland: Algorithmik dominiert stark die Theoretische Informatik
Beispiel Frankreich: Logik und Semantik herrschen vor.

Stets gehören aber FS und KT zum Kanon der Theorie.

Wir können und werden daher hier auch auf Ihre Kenntnisse aus den Grundvorlesungen (AFS bzw. GTI) aufbauen.

Literatur zur Vorlesung: (Auswahl)

- The Design and Analysis of Computer Algorithms;
  Aho, Hopcroft, Ullman; Addison/Wesley; 1974

- Komplexitätstheorie;
  W. Paul; Teubner; 1978

- Introduction to Automata, Languages and Computation;
  Hopcroft, Ullman; Addison/Wesley; 1979

- Computational Complexity;
  K. Wagner & G. Wechsung; Reidel-Verlag; 1986

- Einführung in die Komplexitätstheorie;
  K. Reischuk; Teubner, 1990

Einleitung

— gegeben Problem $\mathcal{P}$:
('Schreibe Algorithmus A mit der Eigenschaft ...')

— Ziel:
Messung der Ressourcen, die 'Lösungsalgorithmen' wie A verwenden

— wichtigste Ressourcen:
Zeit $T_A(E)$ und *Platz* $S_A(E)$ für gegebene Eingaben E

— andere Möglichkeiten:
Prozessor-Zahl bei Parallelrechnern, Schaltkreisgröße/-tiefe von Integrierte Schaltungen...

— gemessene Ressourcen abhängig vom Rechner(-modell) M,
z.B. Einheitskostenmaß, logarithmisches Kostenmaß, ...

Wie wird gemessen ?

— Jeder Eingabe E wird eine 'Größe' $g(E) \in \mathbb{N}_0$ zugeordnet, dann

$$t_A(n) := \max\{T_A(E) \mid g(E) = n\}$$

— Traum-Ziel: 'bester' Algorithmus für $\mathcal{P}$, normalerweise jedoch nur

- untere Schranken, gültig für jeden Algorithmus zur Lösung von $\mathcal{P}$ (i.A. schwer, wenn Schranke nicht trivial!)
- obere Schranken, d.h. Abschätzung von t_A für gegebenen Alg. A

— hier: Probleme bzgl. oberer Schranken in Klassen eingeteilt,

- z.B. $\mathcal{P} \in \mathtt{DTIME}(t)$, falls $t_A \in O(t)$ für einen Alg. A für $\mathcal{P}$
- analog: $\mathcal{P} \in \mathtt{DSPACE}(t)$

Was sind ‘brauchbare’ Algorithmen?

z.B. bei einem Rechenschritt pro μsec:

n	$t(n) = 1000n^2$	$t(n) = 2^n$
5	$0,025\ sec$	$0,000\ 032\ sec$
10	$0,1\ sec$	$0,001\ 024\ sec$
20	$0,4\ sec$	$\approx 1\ sec$
30	$0,9\ sec$	$1000sec$
40	$1,6\ sec$	11 Tage
50	$2,5\ sec$	32 Jahre

Also: Rechenzeit nicht durch Polynom beschränkt

$\Rightarrow$ Alg. höchstens für kleine Eingaben brauchbar!

Modellierung von Rechnern / Algorithmen

— i.A. Komplexitätsklassen vom Rechner(-modell) abhängig,
aber für ‘realistische’ Modelle z.B. invariant (robust)

$$\mathbf{P} = \bigcup_{k \in \mathbb{N}} \mathtt{DTIME}(n^k)$$

= Menge der in Polynomzeit lösbaren Probleme
(= ‘praktisch lösbare Probleme’)
sowie

$$\mathtt{DSPACE}(s)$$

(Speicherplatz auf verschiedenen Modellen ‘nur’ um konstanten Faktor verschieden)

Erinnerung: Algorithmen+Datenstrukturen:
spezielle Probleme (Sortieren, Suchen, ...) genau untersucht,
mit guter Übereinstimmung unterer und oberer Schranken
— Oft jedoch: bekannte untere und obere Schranken weit auseinander
— Geradezu **typisch**: nichttrivialen unteren Schranken unbekannt und

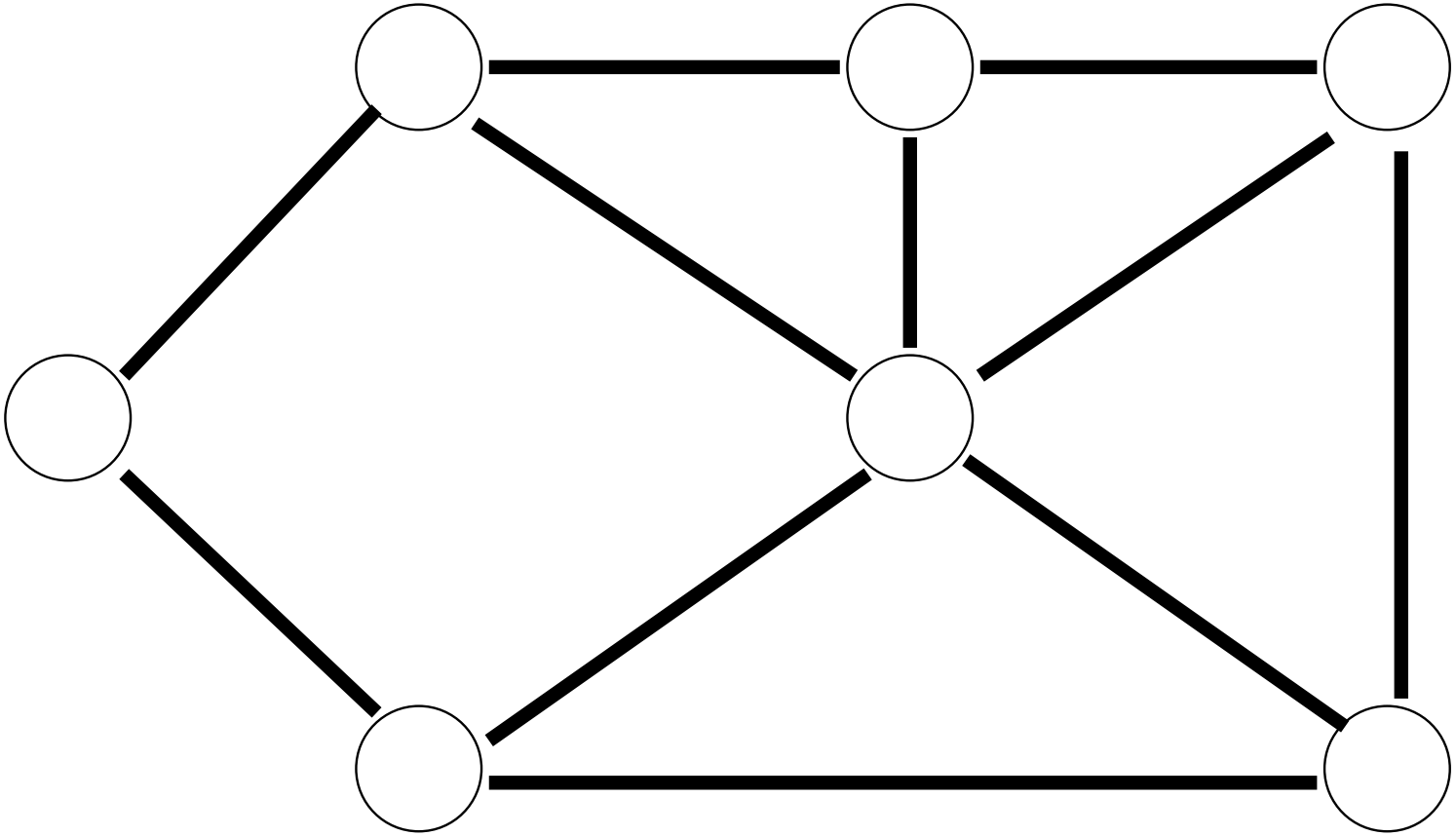
- beste Algorithmen benötigen mindestens exponentielle Zeit

— typisches Beispiel *3-Färbbarkeit*:

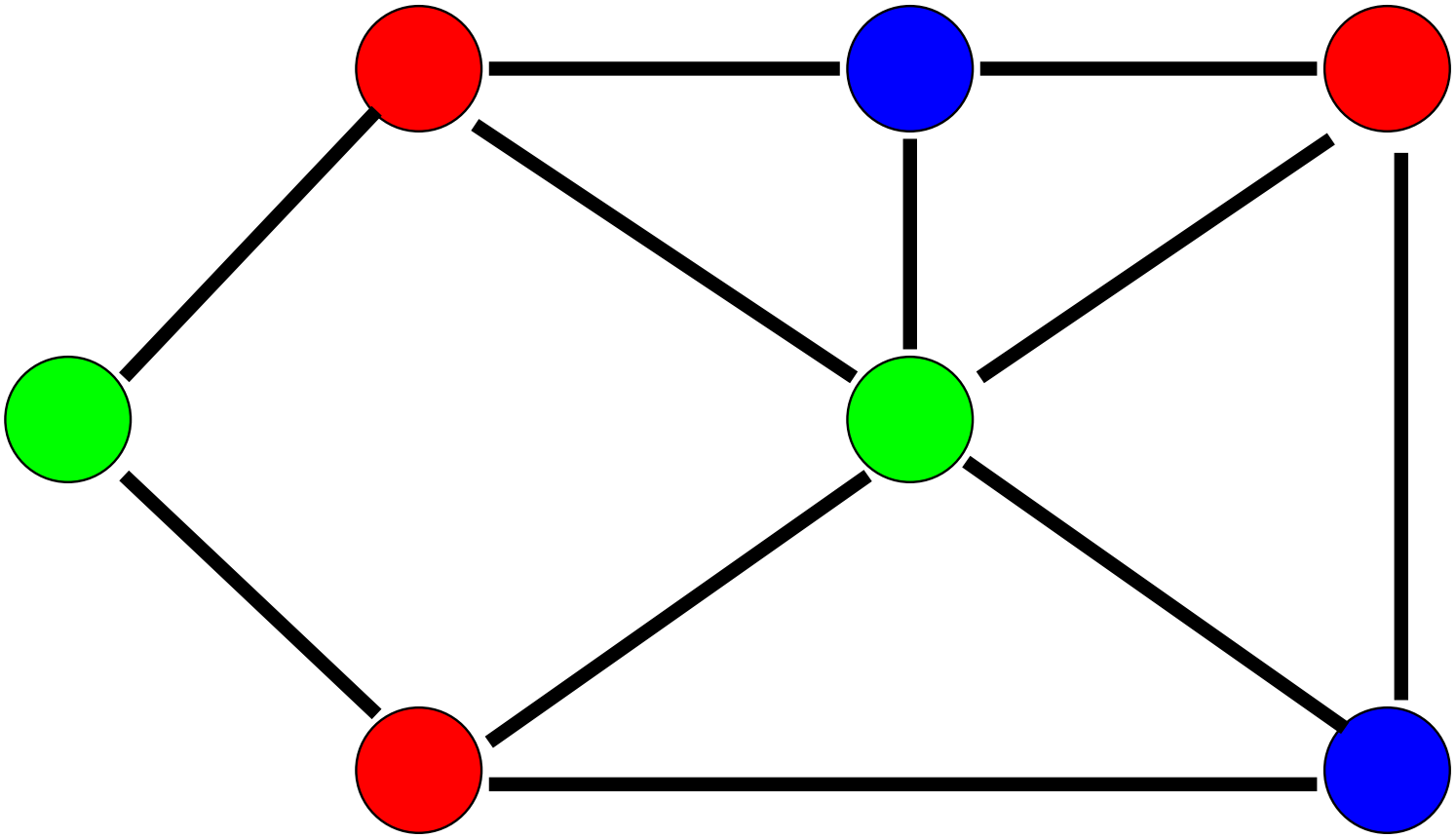
Gegeben sei ein ungerichteter Graph G.

Frage: Kann man die Knoten (Ecken) so mit nur 3 Farben färben,
dass keine benachbarten Ecken gleiche Farbe haben ?

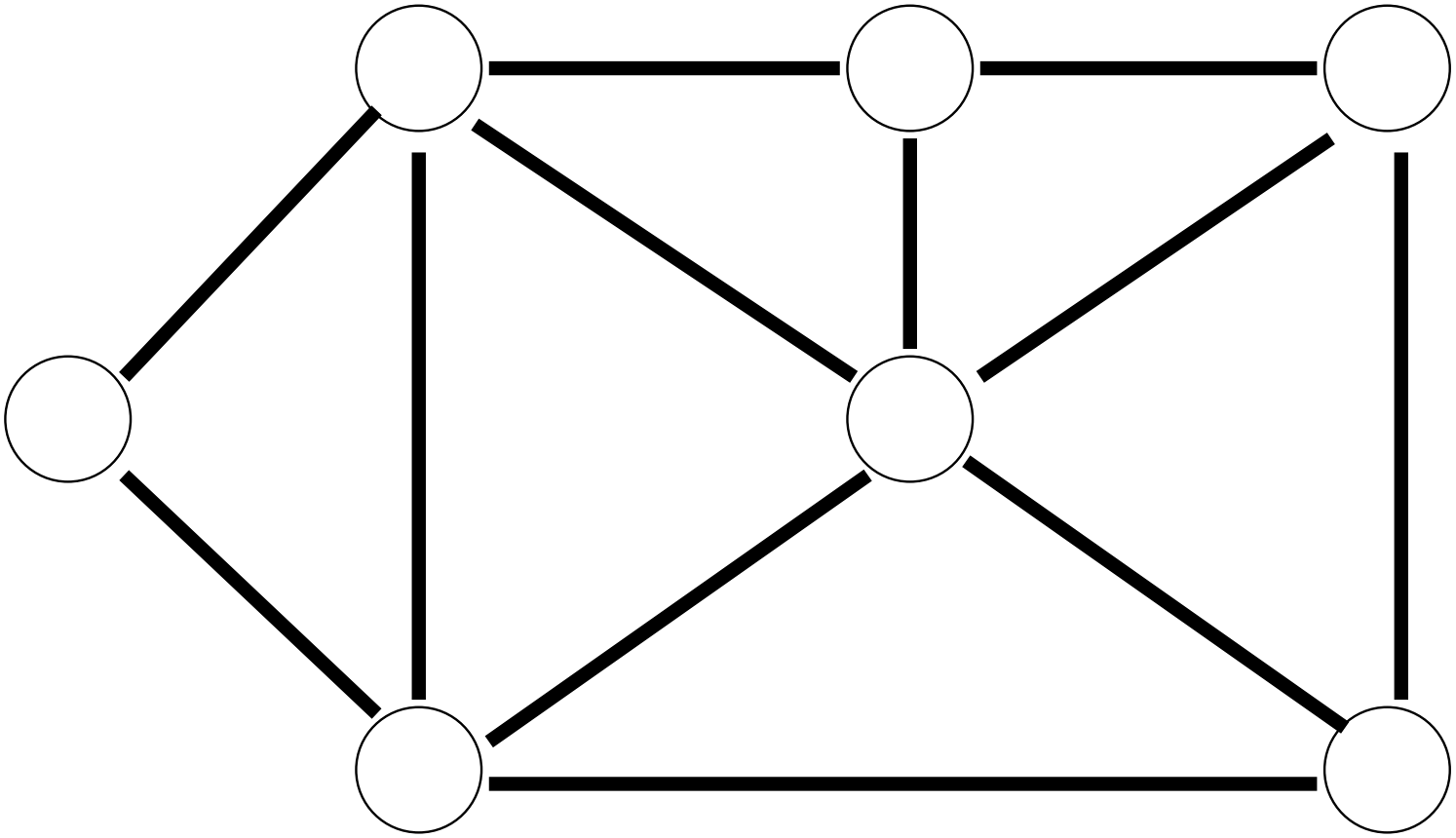
Probleminstanz



Probleminstanz mit Lösung



nicht-3-färbbarer Graph



Algorithmus für 3-Färbbarkeit:

Teste *jede* mögliche Farbkombination darauf, ob sie Lösung ist!
Gibt es eine Lösung $\Rightarrow$ Graph ist 3-färbbar!
Gibt es keine Lösung $\Rightarrow$ Graph ist nicht 3-färbbar!

Nachteil:

Bei n Knoten gibt es 3^n Farbkombinationen,
bei nicht-3-färbbaren Graphen müssen alle getestet werden!
Also: Exponentieller Zeitaufwand !

Leider: Kein Algorithmus bekannt,
der nur n^c Schritte statt 3^n Schritten braucht.
Starke Vermutung: Solch einen "schnellen" Alg. gibt es nicht!

Übertragung der Komplexität von 3-Färbbarkeit auf praktisch relevantere Probleme über *Reduzierbarkeit*:

> Ein Problem A ist '*mindestens so leicht lösbar*' wie Problem B (in Zeichen: $A \leq B$), wenn man aus jedem Algorithmus für B einen (ungefähr) gleich guten Algorithmus für A konstruieren kann.

(I.d.R.: durch Übersetzung der Probleminstanzen von A in Probleminstanzen von B)

Anmerkungen:

— $A \leq B$ ist nur eine 'Vorordnung',
d.h. $A \leq B$ und $B \leq A$ ist auch bei $B \neq A$ möglich
$\Rightarrow$ Äquivalenzrelation $A \equiv B$ mit Klassen gleich schwerer Probleme
— $A \leq B$ ist keine totale Ordnung, d.h. möglich: $A \not\leq B$ und $B \not\leq A$

Zu 3-Färbbarkeit äquivalente Probleme:

Erfüllbarkeitsproblem der Logik (SATisfiability)

Gegeben sei eine Formel aus boolesche Variablen und den logischen Operatoren $\wedge, \vee, \neg$.

Frage: Kann man die Variablen so mit den Wert $\mathtt{t}$ und $\mathtt{f}$ belegen, dass die Formel wahr wird?

Zu 3-Färbbarkeit äquivalente Probleme:

Traveling Salesman-Problem (TSP)

Gegeben: Städte, die durch Straßen verbunden sind.
Jeder Straße werden Kosten zugeordnet (Zeit, Entfernung, Fahrtkosten...)

Frage: Kann man mit gegebenem Budget alle Städte einmal aufsuchen?

Ähnliche Struktur äquivalenter Probleme:

— Lösbar durch Suche in einer Vielzahl von Lösungsansätzen

— Einzelne Lösungsansätze sind leicht erzeugbar

— Test eines Lösungsansatzes ist leicht möglich

Modellierung dieser Eigenschaft:

‘nicht-deterministischer Algorithmus’

— Nichtdet. Algorithmen dürfen beim Ablauf aus Alternativen wählen

— Exakter Ablauf eines nichtdet. Alg. im Voraus nicht bekannt!

Beispiel-Algorithmus für 3-Färbbarkeit:

(1) Eingabe sei Graph G mit Knoten $1, \ldots, n$.
(2) Wiederhole für $i = 1, \ldots, n$:

  (3) Wähle beliebig eine der Farben rot/grün/blau (nichtdet.!)
  (4) Färbe Knoten i mit dieser Farbe

(5) Teste für alle Paare (i, j) von Knoten:
  (6) Sind i, j durch Kante verbunden und mit der gleichen Farbe gefärbt, so beende den Algorithmus ohne Ausgabe

(7) Ausgabe von '*Der Graph ist 3-färbbar*'

Eigenschaften des Algorithmus:

— Wegen (3) verschiedene Abläufe bei gleicher Eingabe möglich

Nichtdeterministische Auswahl / Raten

— Sogar verschiedene Endresultate sind möglich!

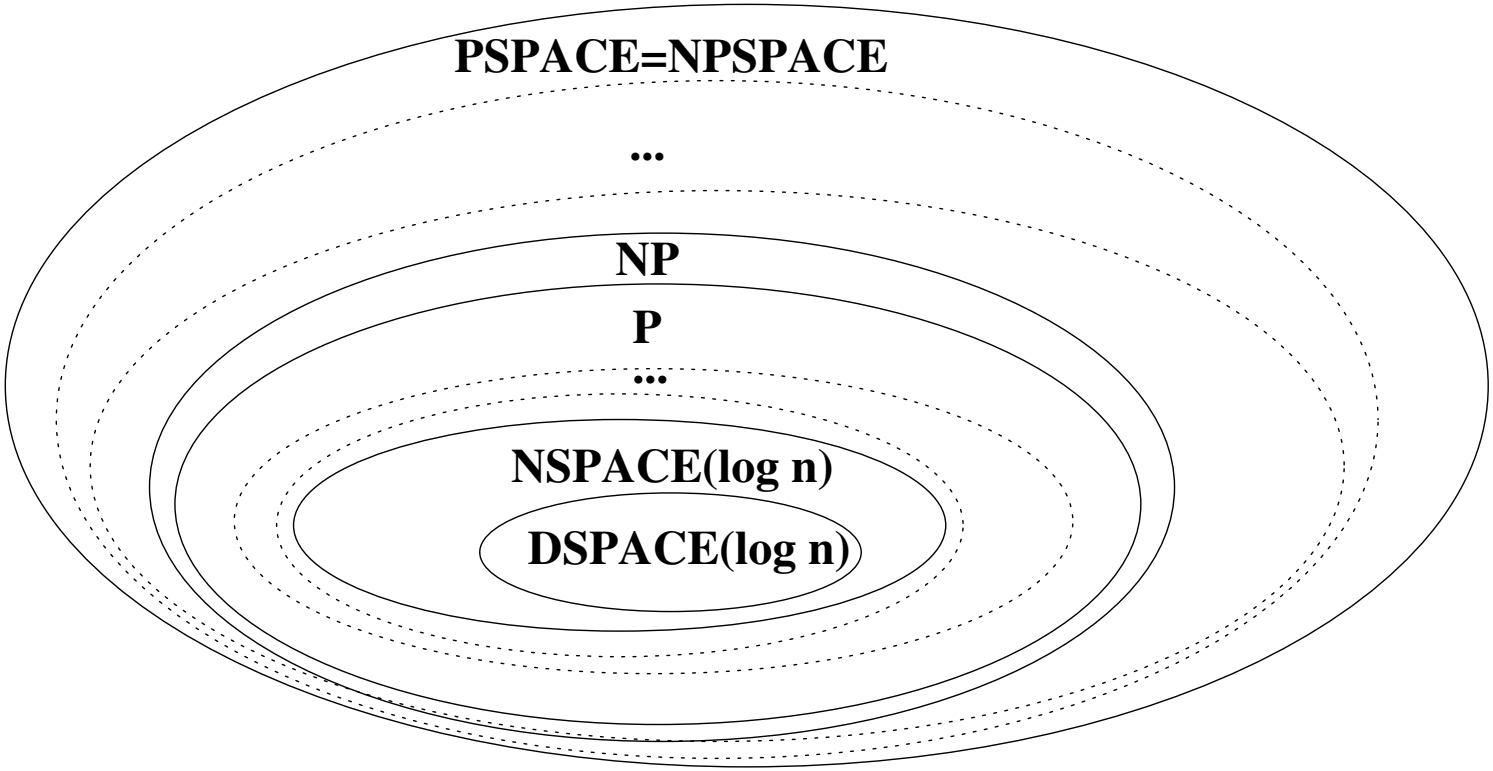
in Abhängigkeit von der erfolgten Wahl

— Bei nicht 3-färbbaren Graphen kann (7) **nie** erreicht werden!

— Ist G 3-färbbar, so *kann* (7) erreicht werden.

— Der Alg. arbeitet in Zeit $O(n^2)$ (bei geeigneten Graph-Datenstrukturen)

Ein wesentliches **Ziel der Vorlesung** ist die Untersuchung folgender Zusammenhänge zwischen Determinismus und Nichtdeterminismus:



Einige wichtige Begriffe aus dem **Bereich der formalen Sprachen**:

- *Alphabet* Σ : nichtleere (i.A. endliche) Menge

- Wort w über Σ : endliche Folge von Zeichen aus Σ
  Länge des Wortes w : $\mathrm{lg}(w)$ oder $|w|$

- leeres Wort: λ (unabhängig vom Alphabet)

- $W(\Sigma)$ oder Σ^*: Menge aller Wörter über Σ
  $W_k(\Sigma)$ oder Σ^k: Menge der Wörter der Länge $k \in \mathbb{N}$.

- Konkatenation zweier Worte w, w' : $w \circ w'$ oder kurz ww'
- w^{r}: Spiegelung von w (z.B. *Hallo*$^{\mathrm{r}}$ $=$*ollaH*)
- Formale Sprache L über Σ : beliebige Teilmenge von $W(\Sigma)$.
- Problem P : ebenfalls Teilmenge von $W(\Sigma)$, d.h. formale Sprache.

Beispiel: *Erfüllbarkeitsproblem der Logik* SAT

Menge aller Worte über Alphabet

$$\{\wedge, \vee, \neg, (,), V, 0, 1\}$$

die erfüllbare Formel darstellen.

dabei z.B. $V101$ als die Variable v_5 interpretiert

z.B.

$$V1 \wedge (V10 \vee \neg V1) \in \text{ SAT}$$

(entsprechend $v_1 \wedge (v_2 \vee \neg v_1)$)

Noch mehr Motivation

Das Gute: P

Erinnerung: **P** ist die Klasse von (Entscheidungs-)Problemen, die von *deterministischen* Turing-Maschinen in Polynomzeit entschieden werden können.

Beispiele:

- Sortieren von n Gegenständen: $\mathcal{O}(n \log(n))$.
- Wortproblem für kontextfreie Sprachen: $\mathcal{O}(n^3)$.
- Matrixmultiplikation: $\mathcal{O}(n^3)$.

Das Böse: NP-Härte

Erinnerung: NP ist die Klasse von (Entscheidungs-)Problemen, die von *nichtdeterministischen* Turing-Maschinen in Polynomzeit entschieden werden können.

Meist wird von solch einem NP-Entscheidungsalgorithmus eine Lösung mitgeliefert. M.a.W.: Es gibt (dann) einen *deterministischen* Algorithmus, der die Gültigkeit der gefundenen Lösung *überprüft*.

NP-vollständige Probleme sind die “schwierigsten” Probleme in NP. NP-harte Probleme sind keineswegs einfacher, liegen aber nicht unbedingt in NP.

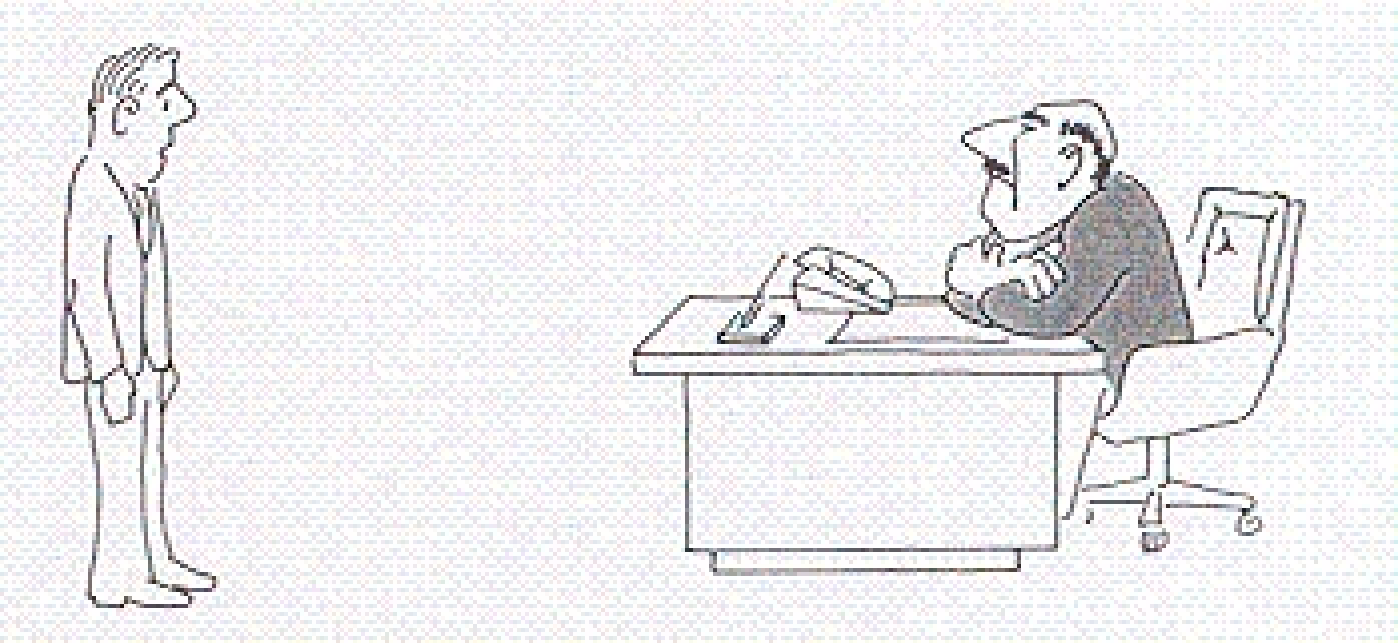
Motivation

Viele interessante Probleme (aus der Praxis!) sind NP-hart

$\Rightarrow$ wohl keine Polynomialzeitalgorithmen sind zu erwarten.

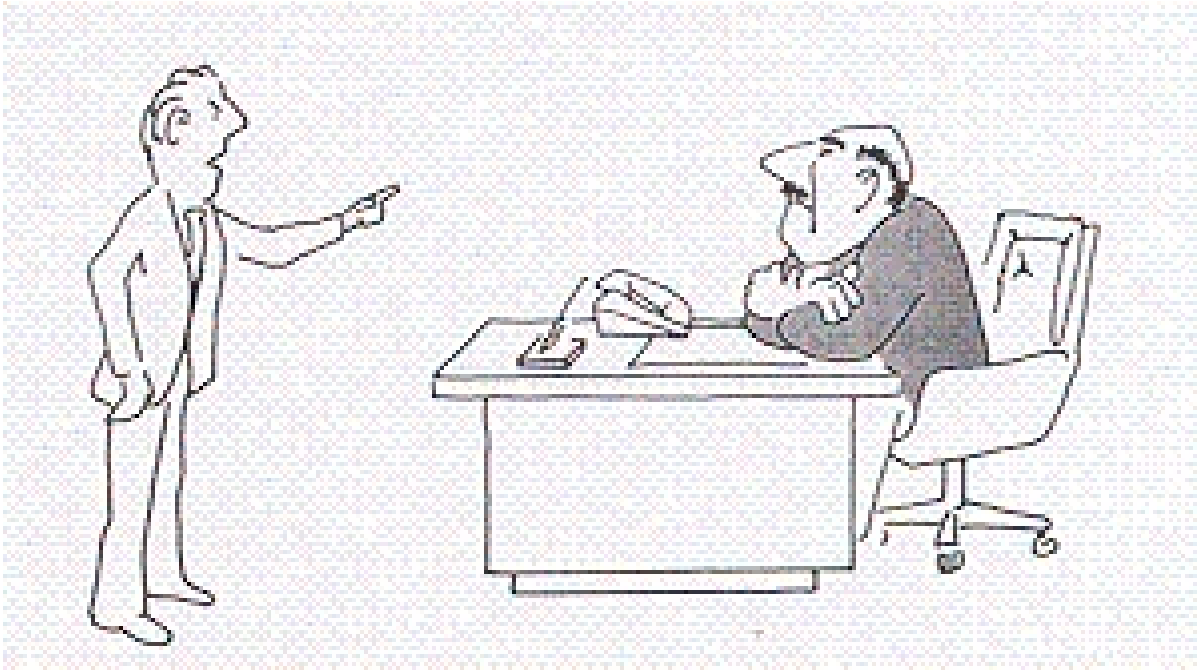
Motivation

siehe `http://max.cs.kzoo.edu/~kschultz/CS510/ClassPresentations/NPCartoons.html` wiederum aus Garey / Johnson



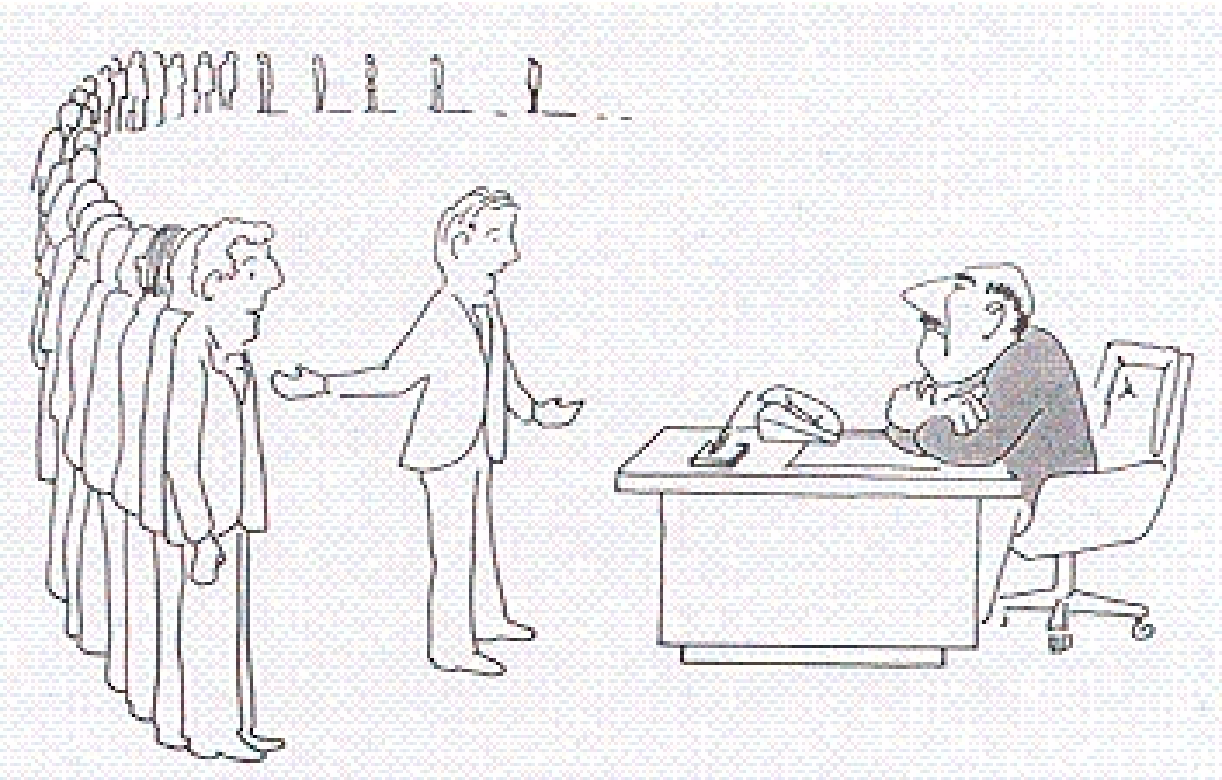
Sorry Chef, aber ich kann für das Problem keinen guten Algorithmus finden. . .

Die beste Antwort wäre hier aber. . .



. . . Ich kann aber beweisen, dass es für das Problem keinen guten Algorithmus geben kann !

Was die Komplexitätstheorie statt dessen liefert. . .



. . . Ich kann aber beweisen, dass das alle anderen auch nicht können !

Das Credo: $\mathbf{P} \subsetneq \mathbf{NP}$

Folgerung: Für NP-harte Probleme “glaubt man” nicht an Polynomzeitalgorithmen zu ihrer Lösung.

“Ergo” (?) Exponentialzeitalgorithmen sind unvermeidlich für exakte Lösungen NP-harter Probleme.

Ähnliche Zusammenhänge und Folgerungen sind in anderen Bereichen der KT möglich und üblich.