

Komplexitätstheorie

SoSe 2019

Jean Christoph Jung, Thomas Schneider

Kapitel 3: P versus NP

Homepage der Vorlesung: <http://tinyurl.com/ss19-kt>

P versus NP

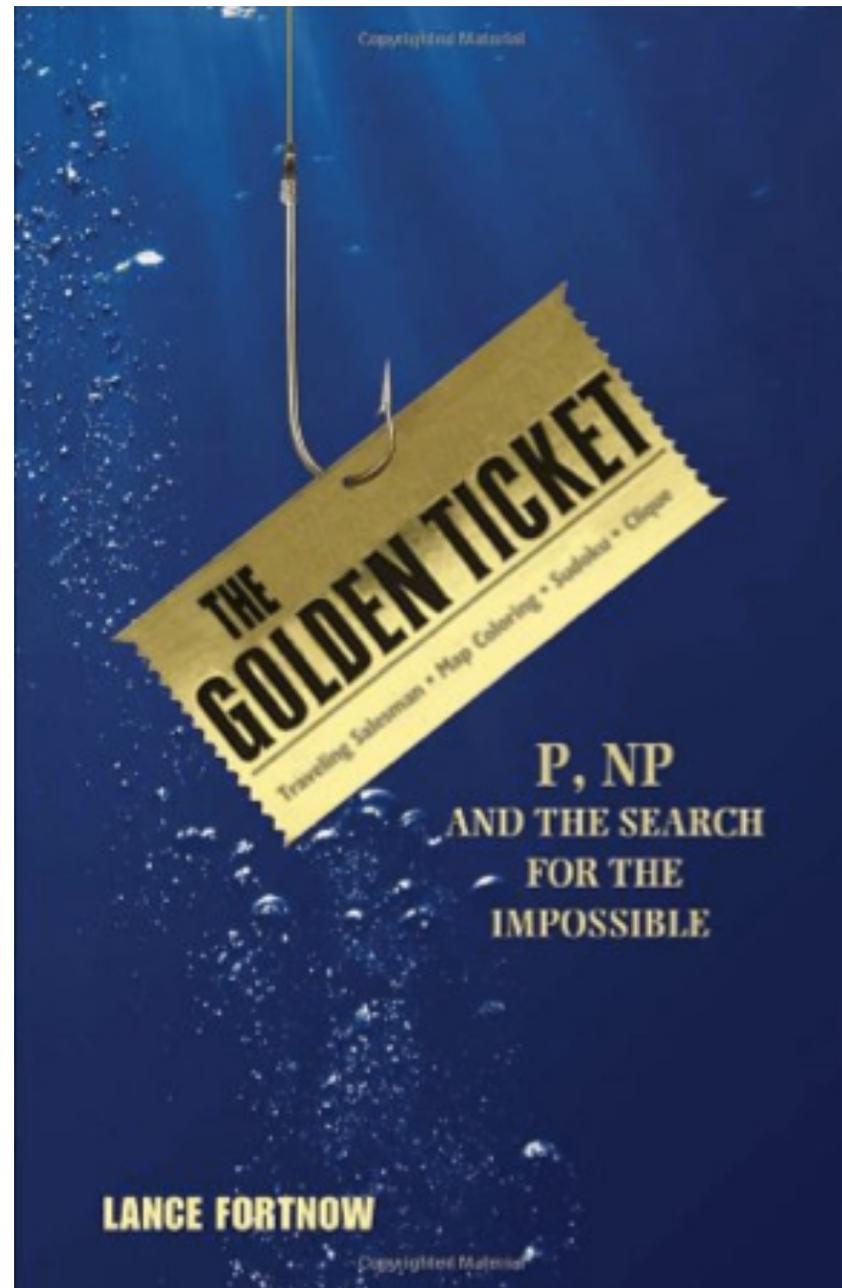
Wir definieren die wichtigen Komplexitätsklassen P und NP und studieren deren Zusammenhang:

- Definitionen der Klassen, polynomielle Beweissysteme
- Zusammenhang zwischen NP und Nichtdeterminismus
- NP-Vollständigkeit
- Beispiele für NP-vollständige Probleme
- Komplemente und coNP

Dies wird Basis für weiterführende Themen sein:

- Isomorphie und spärliche Mengen
- Satz von Ladner und Constraint-Satisfaction-Probleme

Einleitung



Next



3.1 Definition von P und NP

3.2 P versus NP

3.3 NP und nichtdeterministische TMs

3.4 NP-Härte, NP-Vollständigkeit

3.5 Komplemente und coNP

Die Klasse P

Traditionelle Sicht:

Problem ist „effizient lösbar“ gdw. es in polynomieller Zeit lösbar ist
(Polynom von beliebigem Grad!)

Entsprechende Komplexitätsklasse:

$$P := \bigcup_{i \geq 1} \text{DTime}(n^i) \quad (\text{oft auch PTIME})$$

Probleme in P nennt man oft auch **tractable**.

Die Klasse P

Warum ist P eine „gute“ Komplexitätsklasse?

- Zeitkomplexität n^k mit sehr großem k ist zwar bedenklich, aber Polynome mit großem Grad sind **fast nie notwendig**.
- Nach erweiterter Church-Turing-These ist die Klasse P **für alle Berechnungsmodelle identisch** (z. B. 1- vs. k -Band-TMs)
- P ist **abgeschlossen** unter den üblichen Kompositionsoperationen auf Algorithmen. **T3.1**

Die letzten beiden Eigenschaften werden z. B. von $DTime(n)$ **nicht** erfüllt.

Die Klasse NP

Sehr viele Probleme in der Informatik haben folgende Charakteristik:

Für alle $w \in \Sigma^*$ gilt:

- wenn $w \in L$ (w ist „Ja-Instanz“),
dann gibt es einen **einfach zu verifizierenden Beweis** dafür,
der von **kurzer Länge** ist (polynomiell in der Länge von w)
- wenn $w \notin L$ (w ist „Nein-Instanz“),
dann gibt es **keinen** solchen Beweis.

Diese Beobachtung ist die Grundlage für die Definition der Komplexitätsklasse NP.

Beispiel 1: Cliquesproblem

Beim Cliquesproblem ist der „Beweis“ **die Clique selbst**:

- wenn $(G, k) \in \text{CLIQUE}$, dann gibt es Knotenmenge $\{v_1, \dots, v_k\}$, die Clique in G ist;
- wenn $(G, k) \notin \text{CLIQUE}$, dann gibt es keine solche Menge.

Der Beweis ist **einfach zu verifizieren**: man kann offensichtlich in polynomieller Zeit entscheiden, ob **gegebene** Knotenmenge Clique ist.

Der Beweis ist **kurz**: nicht größer als der Graph.

Beispiel 2: Rucksackproblem

Intuitionen:

- Es soll ein Rucksack mit gegebener Kapazität gepackt werden.
- Es gibt eine Menge von zu packenden Gegenständen mit unterschiedlichem Wert und Gewicht.
- Man möchte Gegenstände von maximalem Gesamtwert mitnehmen.

Beispiel 2: Rucksackproblem

Definition Rucksackproblem

Sei $M = \{a_1, \dots, a_k\}$ eine Menge von *Gegenständen*, wobei Gegenstand a_i *Gewicht* g_i und *Nutzen* n_i hat.

Sei $G \geq 0$ eine *Gewichtsgrenze* und N ein *intendierter Nutzen*.

Lösung für Rucksackproblem (M, G, N) ist Teilmenge $R \subseteq M$, so dass

1. $\sum_{a_i \in R} g_i \leq G$ und
2. $\sum_{a_i \in R} n_i \geq N$.

RP ist die Menge aller Instanzen (M, G, N) , für die eine Lösung existiert.

T3.2

Beweis für Ja-Instanz (M, G, N) ist Teilmenge $R \subseteq M$, die 1.+2. erfüllt

Offenbar nicht länger als Eingabe & leicht zu verifizieren!

Beispiel 3: Integer Programming

Definition Integer Programming

Ein *lineares Gleichungssystem* G ist eine Menge von Gleichungen der Form

$$c_1 \cdot x_1 + \dots + c_n \cdot x_n = \alpha,$$

wobei $x_1, \dots, x_n \in V$, V Variablenmenge, $c_1, \dots, c_n \in \mathbb{N}$ und $\alpha \in V \cup \mathbb{N}$.

Lösung ist Abbildung $\tau : V \rightarrow \mathbb{N}$, so dass alle Gleichungen in G erfüllt sind.

I PROG ist Menge aller Gleichungssysteme G , für die eine Lösung existiert.

T3.3

Gegeben einen Lösungskandidaten kann man offensichtlich in polynomieller Zeit überprüfen, ob er Lösung ist (Addition).

Es ist aber **nicht** offensichtlich, dass es immer **kurze Lösungen/Beweise** für I PROG gibt (\mathbb{N} hat Elemente unbeschränkter Größe).

Beispiel 3: Integer Programming

Ohne Beweis:

Theorem (Papadimitriou)

Sei G Gleichungssystem mit m Gleichungen, n Variablen sowie Konstanten, deren Werte durch a beschränkt sind.

Dann gibt es Lösung **gdw.** es eine Lösung $\tau : V \rightarrow \mathbb{N}$ gibt mit $\tau(x) \leq n(ma)^{2m+1}$ für alle $x \in V$.

→ **Also kann man als „kleine Beweise“ verwenden:**

- Lösungen, deren Zahlenwerte wie im o.g. Theorem beschränkt sind
- Zahlenwerte sind **exponentiell** in der Größe von G , also ist deren **binäre Repräsentation** nur **polynomiell** groß ($\log(2^n)$ ist polynomiell!)

Die Klasse NP

Es gibt Tausende solcher Probleme ...

Wir setzen:

„einfach zu verifizieren“: **in polynomieller Zeit**

„kurzer Beweis“: **Länge des Beweises polynomiell** in Länge der Instanz

Daraus ergibt sich die Definition der Klasse NP.

Die Klasse NP

Definition Komplexitätsklasse NP

Sei $L \subseteq \Sigma^*$. Relation $R \subseteq \Sigma^* \times \Gamma^*$ ist *Beweissystem* für L , wenn

- $(w, b) \in R$ impliziert $w \in L$ und (Korrektheit)
- $w \in L$ impliziert $(w, b) \in R$ für ein $b \in \Gamma^*$ (Vollständigkeit)
so ein b heißt *Beweis* oder *Zeuge* für w

R ist *polynomiell*, wenn

- es gibt Polynom p , so dass $|b| \leq p(|w|)$ für alle $(w, b) \in R$
- $R \in P$ (d. h.: gegeben $w \in \Sigma^*$ und $b \in \Gamma^*$
kann DTM in polynomieller Zeit entscheiden, ob $(w, b) \in R$)

NP ist Klasse aller Probleme L , für die es polynomielles Beweissystem gibt.

Die Beispiele haben gezeigt:

CLIQUE, RP, IPROG sind in NP.

Next



3.1 Definition von P und NP

3.2 P versus NP

3.3 NP und nichtdeterministische TMs

3.4 NP-Härte, NP-Vollständigkeit

3.5 Komplemente und coNP

P versus NP

Theorem

$P \subseteq NP \subseteq \text{ExpTime}$, wobei $\text{ExpTime} := \bigcup_{i \geq 1} \text{DTime}(2^{\mathcal{O}(n^i)})$.

T3.4

Sehr unbefriedigend:

Für viele wichtige Probleme in NP (z. B. CLIQUE, RP, IPROG) ist **unbekannt**, ob sie in P (also effizient lösbar) sind!

Es ist sogar unbekannt, ob $P \neq NP$ (und $NP \neq \text{ExpTime}$)

P versus NP

Intuitive Formulierung der P-versus-NP-Frage:

Ist das Finden eines Beweises schwieriger als das Überprüfen?

Unsere Intuition sagt **ja**, also $P \neq NP$!

In der Tat glaubt eigentlich niemand, das $P = NP$,
aber ein Beweis konnte seit 50 Jahren nicht geführt werden!

Die P-versus-NP-Frage ...

- ist eines der wichtigsten offenen Probleme der Informatik
- wurde vom *Clay Mathematics Institute* als eins von sieben **Millenium Prize Problems** ausgelobt

Konsequenzen eines Beweises von

- **P = NP: dramatisch.**
Tausende bisher als sehr schwierig eingestufte Probleme wären dann wohl effizient lösbar
(z. B. CLIQUE, RP, IPROG)
- **P ≠ NP: weniger dramatisch** – aber wichtig zu wissen:
Möglicherweise interessante Konsequenzen in Kryptographie
(dort: schwierige Probleme nützlich statt ärgerlich!)

3.1 Definition von P und NP

3.2 P versus NP

Next



3.3 NP und nichtdeterministische TMs

3.4 NP-Härte, NP-Vollständigkeit

3.5 Komplemente und coNP

Die Klasse NP hat viele äquivalente Charakterisierungen:

- polynomielle Beweissysteme
- nichtdeterministische Turingmaschinen
- existentielle Logik zweiter Stufe (Satz von Fagin)
- randomisierte Beweissysteme (PCP-Theorem)
- etc.

Nichtdeterministische TMs

Nichtdeterministische Turingmaschinen (NTMs) generalisieren DTMs:

- Statt Übergangsfunktion δ haben sie Übergangsrelation

$$\Delta \subseteq (Q \setminus \{q_{\text{acc}}, q_{\text{rej}}\}) \times \Gamma \times Q \times \Gamma \times \{L, R, N\}$$

- (q, a, q', a', L) heißt: wenn M in q ist und a liest, so kann sie nach q' gehen, a' schreiben und sich nach links bewegen
- **Relation**: NTM hat u. U. mehrere Möglichkeiten für nächsten Schritt
- Konfigurationen / Berechnungen definiert wie für DTMs
- Es gibt jetzt mehrere Berechnungen auf derselben Eingabe
- NTM akzeptiert Eingabe w , wenn mindestens eine Berechnung auf w akzeptierend

Nichtdeterministische TMs

Definition NTime

Für NTM M und $w \in \Sigma^*$ schreiben wir $\text{time}_M(w) = n$, wenn **alle** Berechnungen von M auf w höchstens n Schritte umfassen.

Sei $t : \mathbb{N} \rightarrow \mathbb{N}$ monoton wachsende Funktion mit $t(n) \geq n$.

M ist *t -zeitbeschränkt*, wenn $\text{time}_M(w) \leq t(n)$ für alle w der Länge n .

Zeitkomplexitätsklasse $\text{NTime}_i(t)$ ist definiert als Menge der Probleme

$$\{L \subseteq \Sigma^* \mid \exists \mathcal{O}(t)\text{-zeitbeschr. } i\text{-Band-NTM } M \text{ mit } L(M) = L\}.$$

Wir setzen $\text{NTime}(t) := \bigcup_{i \geq 1} \text{NTime}_i(t)$.

Theorem (NTM-Charakterisierung von NP)

$$\text{NP} = \bigcup_{i \geq 1} \text{NTime}(n^i)$$

Nichtdeterministische TMs

Klassischerweise ist diese Charakterisierung sogar die **eigentliche Definition** von NP.

Die P-versus-NP-Frage kann also auch wie folgt verstanden werden:

Kann eine nichtdeterministische Maschine in polynomieller Zeit mehr erreichen als eine deterministische Maschine?

3.1 Definition von P und NP

3.2 P versus NP

3.3 NP und nichtdeterministische TMs

Next



3.4 NP-Härte, NP-Vollständigkeit

3.5 Komplemente und coNP

NP-Härte

Bisher ist es für sehr viele „typische“ NP-Probleme nicht gelungen zu zeigen, ob sie in P (effizient lösbar) sind oder nicht.

Der Begriff der NP-Härte liefert ein Mittel, solche Probleme trotzdem als schwierig klassifizieren zu können.

Starke Indikation dafür, dass Problem L **nicht** in P ist:

1. L ist „mindestens so schwierig“ wie **alle** anderen Probleme in NP
2. Daraus folgt, dass L nur in P ist, wenn $P = NP$ (also unwahrscheinlich!)

Reduktionen

Reduktionen sind zentral in der Komplexitätstheorie:

- Werkzeug um Probleme zu vergleichen, nicht nur im Kontext von NP
- existieren in zahllosen Variationen (für verschiedene Zwecke)

Definition (polynomielle) Reduktion

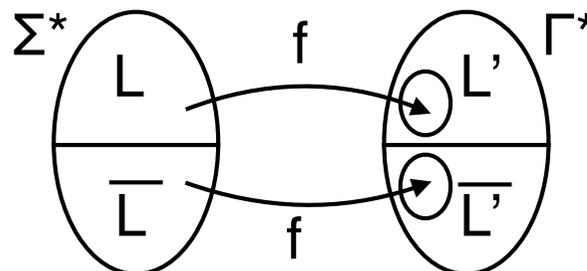
Sei $L \subseteq \Sigma^*$, $L' \subseteq \Gamma^*$. Eine *Reduktion* von L auf L' ist berechenbare Funktion $f : \Sigma^* \rightarrow \Gamma^*$, so dass

$$w \in L \text{ gdw. } f(w) \in L' \quad \text{für alle } w \in \Sigma^*.$$

f heißt *polynomiell*, wenn f in polynomieller Zeit berechenbar ist.

Wir schreiben $L \leq_p L'$, wenn polynomielle Reduktion von L auf L' existiert.

Illustration:



Reduktionen

Intuitiv: $L \leq_p L'$ bedeutet „ L' ist **mindestens so schwer wie** L “.

Denn: Algorithmus für L' kann mit nur **polynomiell**em Mehraufwand zum Lösen von L verwendet werden!

Theorem

1. **P ist abgeschlossen** unter polynomiellen Reduktionen:

$$L' \in P \text{ und } L \leq_p L' \text{ impliziert } L \in P.$$

2. **NP ist abgeschlossen** unter polynomiellen Reduktionen:

$$L' \in NP \text{ und } L \leq_p L' \text{ impliziert } L \in NP.$$

3. „Polynomiell reduzierbar“ ist **transitive Relation**:

$$L_1 \leq_p L_2 \text{ und } L_2 \leq_p L_3 \text{ impliziert } L_1 \leq_p L_3.$$

NP-Härte

Definition NP-Härte, NP-Vollständigkeit

Problem L ist

- *NP-hart*, wenn $L' \leq_p L$ für alle $L' \in \text{NP}$;
- *NP-vollständig*, wenn L sowohl NP-hart als auch in NP.

Ein NP-hartes Problem ist also **mindestens so schwer** wie jedes andere Problem in NP.

Beobachtung

Wenn L NP-hart und $L \in P$, dann $P = \text{NP}$.

Solange das P-versus-NP-Problem ungelöst ist, wird NP-Härte deshalb **als „nicht effizient lösbar“ gewertet.**

NP-Härte

Zwei Ansätze, NP-Härte von Problem L zu zeigen:

1. Zeigen, dass $L' \leq_p L$ für **alle** $L' \in \text{NP}$
2. Zeigen, dass $L' \leq_p L$ für **ein NP-hartes** Problem L'

Lemma

Wenn L NP-hart ist und $L \leq_p L'$, dann ist L' NP-hart.

T3.6

Wir beginnen mit Ansatz 1 für das klassische SAT-Problem.

Definition Formel, Wertzuweisung, Erfüllbarkeit

Sei VAR unendlich abzählbare Menge von *Aussagenvariablen*.

Menge der *aussagenlogischen Formeln* (kurz: *AL-Formeln*)

ist die kleinste Menge, so dass:

- Jedes $x \in \text{VAR}$ ist AL-Formel.
- Wenn φ, ψ AL-Formeln, so auch $\neg\varphi, \varphi \vee \psi, \varphi \wedge \psi$.

Eine *Belegung* ist eine Abbildung $V : \text{VAR} \rightarrow \{0, 1\}$.

Belegung V wird wie folgt auf zusammengesetzte AL-Formeln erweitert:

- $V(\neg\varphi) = 1 - V(\varphi)$
- $V(\varphi \wedge \psi) = \min\{V(\varphi), V(\psi)\}$
- $V(\varphi \vee \psi) = \max\{V(\varphi), V(\psi)\}$

AL-Formel φ ist *erfüllbar*, wenn es Belegung V gibt mit $V(\varphi) = 1$.

Der Satz von Cook

Problem **SAT**: die Menge aller erfüllbaren AL-Formeln

Theorem (Cook/Levin aka „Satz von Cook“ aka „Cook's Theorem“)

SAT ist NP-vollständig.

SAT \in **NP**: $R = \{(\varphi, V) \mid V \text{ ist Belegung für Variablen in } \varphi, \text{ die } \varphi \text{ erfüllt}\}$
ist polynomielles Beweissystem

NP-Härte: Zeigen, dass Wortproblem **jeder** polyzeit-beschränkten NTM polynomiell auf SAT reduziert werden kann.

(Bekannt aus „Theoretische Informatik 2“, Satz 20.4 im Skript)

Zur Erinnerung:

Zwei Ansätze, NP-Härte von Problem L zu zeigen:

1. Zeigen, dass $L' \leq_p L$ für **alle** $L' \in \text{NP}$
2. Zeigen, dass $L' \leq_p L$ für **ein NP-hartes** Problem L'

Wir verwenden jetzt Ansatz 2 für weitere Probleme.

3SAT

Oft ist es jedoch bequemer, nur **Formeln in bestimmter Form** zu betrachten.

Definition 3SAT

Eine *3-Formel* hat die Form

$$\bigwedge_i (\ell_{1,i} \vee \ell_{2,i} \vee \ell_{3,i})$$

wobei die $\ell_{j,i}$ *Literale* sind, also Variable oder deren Negation.

Jede Disjunktion $(\ell_{1,i} \vee \ell_{2,i} \vee \ell_{3,i})$ heißt *Klausel*.

3SAT ist die Menge aller erfüllbaren 3-Formeln.

T3.8

Bekannt aus „Theoretische Informatik 2“:

3SAT ist NP-vollständig.

Hier nur kurze Erinnerung.

3SAT

Theorem

3SAT ist NP-vollständig.

3SAT \in NP: klar (Spezialfall von SAT)

Beweis NP-Härte: polynomielle Reduktion von SAT

Gegeben AL-Formel φ , konstruiere in polynomieller Zeit 3-Formel ψ so dass φ erfüllbar gdw. ψ erfüllbar.

Idee:

- führe Aussagenvariable für jede Teilformel von φ ein
- beschreibe das Verhalten von Teilformeln in AL
- Konvertiere entstehende Formel in 3-Formel
- Resultierende Formel ist nicht äquivalent, aber äqui-erfüllbar

T3.9

CLIQUE

Ebenfalls bekannt aus „Theoretische Informatik 2“:

Theorem

CLIQUE ist NP-vollständig.

Beweis Härte: polynomielle Reduktion von 3SAT

3-Färbbarkeit

Wie CLIQUE sind **viele interessante Probleme auf Graphen NP-vollständig.**

Wir betrachten ein **weiteres Beispiel.**

Definition 3-Färbbarkeit

Ungerichteter Graph $G = (V, E)$ ist **3-färbbar**, wenn es Abbildung

$$f : V \rightarrow \{R, G, B\}$$

gibt, so dass $\{v, v'\} \in E$ impliziert $f(v) \neq f(v')$ (**3-Färbung**).

3F ist die Menge aller Graphen G , die 3-färbbar sind.

T3.10

Leicht zu sehen: $3F \in NP$

3-Färbbarkeit

Theorem

3F ist NP-vollständig.

Beweis Härte: polynomielle Reduktion von 3SAT

Gegeben 3-Formel φ , konstruiere in polynomieller Zeit Graph G so dass φ erfüllbar gdw. G 3-färbbar

Idee:

- Führe pro Literal einen Knoten ein.
- Verwende die Farben „wahr“, „falsch“ und „hilf“.
- Verbinde komplementäre Literale mit Kanten, um konsistente Belegungen zu erzwingen.
- Verwende Teilgraphen, um Erfülltsein jeder Klausel zu erzwingen.

T3.11

NP-vollständige Probleme aus anderen Bereichen

NP-vollständige Probleme gibt es nicht nur in der Logik und Graphentheorie, sondern in **vielen weiteren Bereichen der Informatik**.

1972 veröffentlichte Richard Karp in einem berühmten Aufsatz **21** solche (und **sehr unterschiedliche**) Probleme.

Im Buch „Computers and Intractability“ von Garey und Johnson finden sich **ca. 300 NP-vollständige Probleme**; heute kennt man **Tausende!**

Wir betrachten **zwei weitere Beispiele**:

- Integer Programming
- Rucksackproblem

Zur Erinnerung:

Definition Integer Programming

Ein *lineares Gleichungssystem* G ist eine Menge von Gleichungen der Form

$$c_1 \cdot x_1 + \cdots + c_n \cdot x_n = \alpha,$$

wobei $x_1, \dots, x_n \in V$, V Variablenmenge, $c_1, \dots, c_n \in \mathbb{N}$ und $\alpha \in V \cup \mathbb{N}$.

Lösung ist Abbildung $\tau : V \rightarrow \mathbb{N}$, so dass alle Gleichungen in G erfüllt sind.

IPROG ist Menge aller Gleichungssysteme G , für die eine Lösung existiert.

Integer Programming

Theorem

IPROG ist NP-vollständig.

Beweis Härte: polynomielle Reduktion von 3SAT

Gegeben 3-Formel φ , konstruiere in polynomieller Zeit Gleichungssystem G so dass φ erfüllbar gdw. G Lösung hat.

Idee:

- Verwende für jedes Literal eine numerische Variable.
- Stelle durch Gleichungen sicher, dass diese Variablen konsistente Werte aus dem Bereich $\{0, 1\}$ erhalten.
- Stelle durch zusätzliche Gleichungen sicher, dass jede Klausel mindestens ein wahres Literal enthält.

Rucksackproblem (Wdhlg.)

- Es soll ein Rucksack mit gegebener Kapazität gepackt werden.
- Es gibt eine Menge von zu packenden Gegenständen mit unterschiedlichem Wert und Gewicht.
- Man möchte Gegenstände von maximalem Gesamtwert mitnehmen.

Definition Rucksackproblem

Sei $M = \{a_1, \dots, a_k\}$ eine Menge von *Gegenständen*, wobei Gegenstand a_i *Gewicht* g_i und *Nutzen* n_i hat.

Sei $G \geq 0$ eine *Gewichtsgrenze* und N ein *intendierter Nutzen*.

Lösung für Rucksackproblem (M, G, N) ist Teilmenge $R \subseteq M$, so dass

1. $\sum_{a_i \in R} g_i \leq G$ und

2. $\sum_{a_i \in R} n_i \geq N$.

RP ist die Menge aller Instanzen (M, G, N) , für die eine Lösung existiert.

Rucksackproblem

Theorem

RP ist NP-vollständig.

Beweis Härte: polynomielle Reduktion von 3SAT

Gegeben 3-Formel φ , konstruiere in polynomieller Zeit Rucksackproblem (M, G, N) so dass φ erfüllbar gdw. (M, G, N) Lösung hat.

Idee:

- Verwende für jede Variable x_i zwei Gegenstände a_i und \bar{a}_i .
- Wenn x_i wahr ist, dann ist a_i im Rucksack, aber \bar{a}_i nicht; wenn x_i falsch ist, dann umgekehrt.
- Für jeden Gegenstand a_i ist Gewicht g_i gleich Nutzen n_i und $G = N$.
- Zusätzlich werden Gegenstände b_i und c_i für jede **Klausel** benötigt.

Rucksackproblem

Beweisdetails.

Sei φ 3-Formel mit ℓ Variablen x_1, \dots, x_ℓ und k Klauseln.

Konstruiere Instanz (M, G, N) von RP wie folgt.

- $M = \{a_1, \bar{a}_1, \dots, a_\ell, \bar{a}_\ell, b_1, c_1, \dots, b_k, c_k\}$
- Für alle Gegenstände ist Nutzen = Gewicht; außerdem $N = G$.
- Alle Nutzenwerte sind $(\ell + k)$ -stellige Dezimalzahlen wie folgt:

Nutzen von a_i (\bar{a}_i):

- Dezimalstelle i ist 1; Rest der ersten ℓ Dezimalstellen ist 0.
- Dezimalstelle $\ell + j$ ist 1, wenn Literal x_i ($\neg x_i$) in j -ter Klausel auftritt, sonst 0.

Nutzen von b_i, c_i :

- Dezimalstelle $\ell + i$ ist 1, alle anderen 0.

$G (= N)$ hat Dezimaldarstellung $\underbrace{11 \dots 1}_\ell \underbrace{33 \dots 3}_k$

Achtung: Binäre Kodierung der Eingabe wichtig!

T3.12

NP-vollständige Probleme finden sich auch im täglichen Leben:

- **Minesweeper:** Gegeben den momentanen Stand eines Spiels (mit Brett beliebiger Größe), ist an einer bestimmten (verdeckten) Stelle mit Sicherheit eine Mine versteckt?
- **Sudoku:** Gegeben ein partiell gelöstes Sudoku, kann es zu einem vollständig gelösten erweitert werden?
- **Bundesliga:** Gegeben den momentanen Punktestand der Bundesliga (mit beliebig vielen Mannschaften), kann eine bestimmte Mannschaft noch Meister werden?
- ...

Kapitel 3

3.1 Definition von P und NP

3.2 P versus NP

3.3 NP und nichtdeterministische TMs

3.4 NP-Härte, NP-Vollständigkeit

Next



3.5 Komplemente und coNP

Motivation

Sei $\overline{3F}$ die Menge aller **nicht** 3-färbbaren Graphen.

In welcher Komplexitätsklasse ist $\overline{3F}$?

- Wenn $\overline{3F}$ in NP ist, müsste es polynomielles Beweissystem geben.
Aber was ist polynomieller Beweis dafür, dass Graph **nicht** 3-färbbar?
Menge **aller** möglichen 3-Färbungen? Exponentiell viele!
- Offensichtlich ist $\overline{3F}$ ein Problem, bei dem es kurze und einfach zu verifizierende Beweise für **nein**-Instanzen gibt statt für ja-Instanzen.

Komplemente von Problemen

Definition Komplement (Problem)

Sei $L \subseteq \Sigma^*$ Problem. Das *Komplement* von L ist $\bar{L} := \Sigma^* \setminus L$.

Beispiele:

$\overline{3F}$

$\overline{\text{SAT}}$ (Menge aller **unerfüllbaren** AL-Formeln)

$\overline{\text{CLIQUE}}$ (Menge aller Paare (G, k) , so dass G **keine** k -Clique hat)

Deterministische Zeitkomplexitätsklassen wie z. B. P sind **unter Komplement abgeschlossen**:

Lemma

Sei $\mathcal{C} = \text{DTime}(t)$ für beliebiges t . Dann gilt: $L \in \mathcal{C}$ impliziert $\bar{L} \in \mathcal{C}$.

T3.13

Also z. B. für **GAP** (Erreichbarkeitsproblem auf gerichteten Graphen):

$\overline{\text{GAP}} \in P$

Komplement-Komplexitätsklassen

Argument aus dem letzten Beweis **schlägt fehl** für **nichtdeterministische** TMs, funktioniert also nicht für die **alternative Charakterisierung von NP!**

Definition Komplement-Komplexitätsklasse

Sei \mathcal{C} Komplexitätsklasse. $\text{co}\mathcal{C} := \{L \mid \bar{L} \in \mathcal{C}\}$

Wir interessieren uns hier **insbesondere** für die **Klasse coNP**.

Z. B. sind $\overline{3F}$, $\overline{\text{SAT}}$, $\overline{\text{CLIQUE}}$ per Definition in coNP.

Weiteres natürliches coNP-Problem:

Definition Gültigkeit in Aussagenlogik

AL-Formel φ ist **gültig** gdw. jede Belegung φ erfüllt.

Lemma

Gültigkeit ist in coNP.

coNP-Vollständigkeit

Härte und Vollständigkeit sind definiert wie für NP:

Definition coNP-Härte, coNP-Vollständigkeit

Ein Problem L ist

- *coNP-hart*, wenn $L' \leq_p L$ für alle $L' \in \text{coNP}$;
- *coNP-vollständig*, wenn L coNP-hart und in coNP.

Lemma

L ist NP-hart gdw. \bar{L} coNP-hart.

$\overline{3F}$, $\overline{\text{SAT}}$, $\overline{\text{CLIQUE}}$ etc. sind also **coNP-vollständig**.

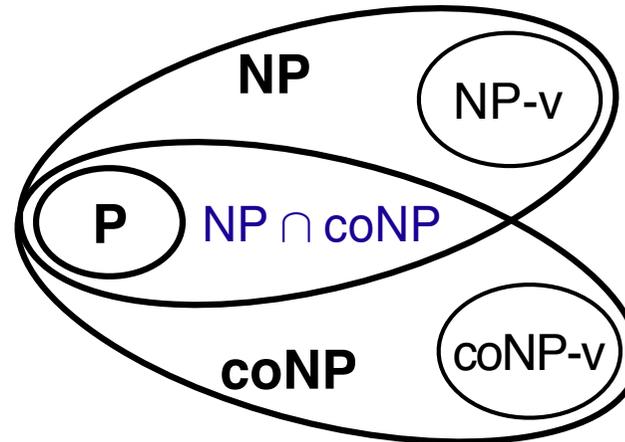
T3.15

Lemma

Gültigkeit ist coNP-vollständig.

T3.16

P versus NP versus coNP



Leicht zu sehen:

$$P \subseteq \text{coNP}$$

T3.17

Es wird vermutet:

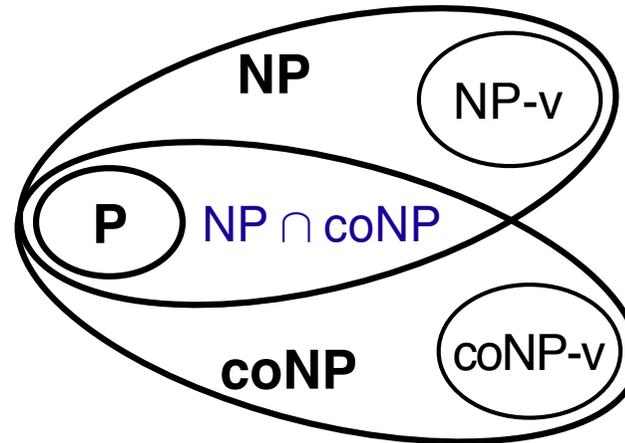
1. $\text{NP} \neq \text{coNP}$
2. $\text{NP} \cap \text{coNP} \neq P$

Beide Vermutungen implizieren $P \neq \text{NP}$:

Wenn $P = \text{NP}$, dann $P = \text{NP} = \text{coNP}$ (Abschluss P unter Komplement)

Die umgekehrte Implikation ist **nicht** bekannt.

NP \cap coNP



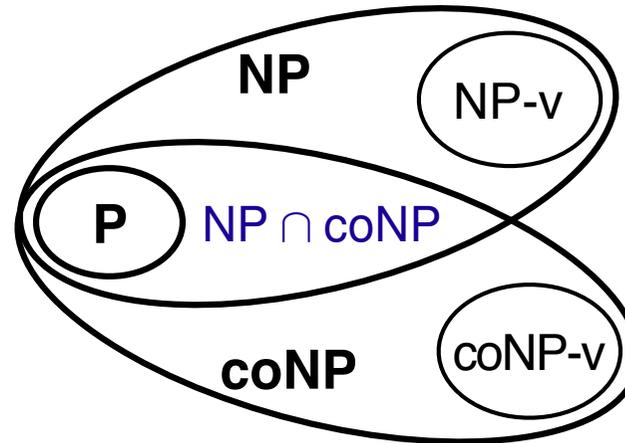
Damit ergibt sich $NP \cap coNP$ als weitere interessante Klasse.

NP: kurze/einfach zu verifizierende Beweise für **ja**-Instanzen

coNP: kurze/einfach zu verifizierende Beweise für **nein**-Instanzen

$NP \cap coNP$: **beides** ist der Fall

NP \cap coNP



Ein bekanntes Problem in NP \cap coNP:

Definition Integer-Faktorisierung

Integer-Faktorisierung (IF) ist das folgende Problem:

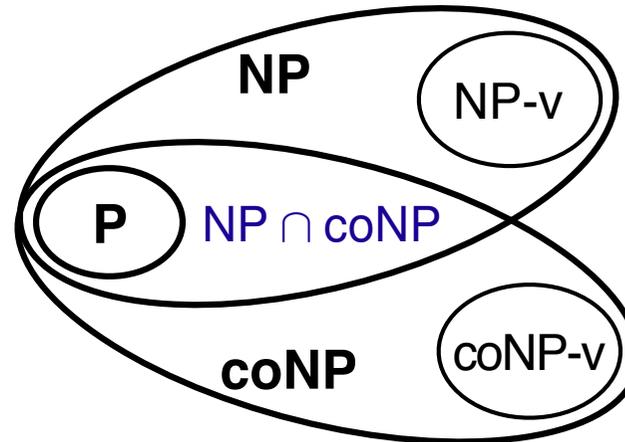
Gegeben $n, k \in \mathbb{N}$, entscheide, ob n einen Faktor $\leq k$ hat.

Theorem

IF \in NP \cap coNP

T3.18

NP \cap coNP und NP-Vollständigkeit



Probleme in NP \cap coNP sind ...

- wahrscheinlich **nicht** NP-vollständig:

Lemma

Wenn es ein NP-vollständiges $L \in \text{NP} \cap \text{coNP}$ gibt, dann $\text{NP} = \text{coNP}$.

T3.19

Kryptographie mit dem RSA-Verfahren ist **nicht sicher**, wenn $\text{IF} \in \text{P}$
(denn mit binärer Suche ließen sich dann effizient Faktoren finden).

- natürliche Kandidaten für **NP-intermediate Probleme**
(siehe Satz von Ladner)

Übersicht Vorlesung

Kapitel 1: Einführung

Kapitel 2: Turingmaschinen

Kapitel 3: P vs. NP (Grundlagen)

Next



Kapitel 4: Platzkomplexität (Grundlagen)

Kapitel 5: Orakel (Grundlagen)