



Komplexitätstheorie

Kapitel 3: P vs NP

Einleitung

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

- Definitionen der Klassen
- Zusammenhang zwischen NP und Nichtdeterminismus
- Das Konzept der NP-Vollständigkeit
- Beispiele für NP-vollständige Probleme
- Einige Reflektionen zu P vs. NP
- Komplemente und co-NP

Kapitel 3

Definition von P und NP

Die Klasse P

Traditionelle Sicht:

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

Entsprechende Komplexitätsklasse:

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

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

Die Klasse P

Rechtfertigung:

- Tabelle aus Kapitel 1:

Eingabegröße	1	2	3	4	5	6	7	8		20	128
n^2	1	4	9	16	25	36	49	64		400	16.384
2^n	2	4	8	16	32	64	128	256		1.048.576	RIESIG!

- Zeitkomplexität n^k mit sehr großem k ist auch 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. ein vs mehrere Bänder bei TMs)
- P ist abgeschlossen unter den üblichen Kompositionsoperationen auf Algorithmen

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

Der "Beweis" ist 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

Definition Rucksackproblem

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

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 Lösung existiert.

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

Offenbar nicht länger als Eingabe und 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 aus Variablenmenge V , $c_1, \dots, c_n \in \mathbb{N}$ und $\alpha \in V \cup \mathbb{N}$.

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

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

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)

Wenn G Gleichungssystem mit m Gleichungen, n Variablen und Konstanten beschränkt durch a , dann gibt es Lösung gdw. es eine Lösung aus dem Bereich $\{0, \dots, n(ma)^{2m+1}\}$.

Wir verwenden als "kleine Beweise":

- 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 viele tausend 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 $(w, b) \in R$
- $R \in P$ (also: 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, SEQ, IPROG sind in NP.

Kapitel 3

P vs NP

P vs NP

Theorem

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

Sehr unbefriedigend:

Für viele wichtige Probleme in NP (z.B. CLIQUE, SEQ, 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 vs NP

Intuitive Formulierung der P vs 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 $P=NP$ Frage:

- ist eines der wichtigsten offenen Probleme der Informatik
- Lösung wird vom Clay Mathematics Institute mit US\$1.000.000 belohnt (Millenium Prize)

P vs NP

Konsequenzen eines Beweises von

- $P = NP$: dramatisch. Tausende bisher als sehr schwierig eingestufte Probleme wären dann wohl effizient lösbar (z.B. CLIQUE, SEQ, IPROG)
- $P \neq NP$: weniger dramatisch, aber wichtig zu wissen
möglicherweise interessante Konsequenzen in der Kryptographie
(dort: schwierige Probleme nützlich statt ärgerlich)

Kapitel 3

NP und nicht-deterministische TMs

Die Klasse NP

Die Klasse NP hat viele äquivalente Charakterisierungen:

- polynomielle Beweissysteme
- nicht-deterministische Turingmaschinen
- existentielle Logik zweiter Stufe (Fagins Theorem)
- randomisierte Beweissysteme (PCP Theorem)
- etc.

Nicht-deterministische TMs

Nicht-deterministische Turingmaschinen (kurz: 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 **mind. eine** Berechnung auf w akzeptierend



Nicht-deterministische TMs

NTMs

- entsprechen keinem real existierenden Rechner
- sind relevant wegen Ihres Zusammenhanges zu NP und anderen Komplexitätsklassen
- versteht man am besten über die Metapher des "Ratens":
 - NTM kann nicht-det. Wort w auf Band schreiben ("raten")
 - Akzeptiert wird nur, wenn w so ist "wie gewünscht" (**Verifikation**, dass richtig geraten wurde)
 - Die Maschine muss so konstruiert sein, dass falsches Raten immer zum Verwerfen führt
 - Achtung: NTM kann nur etwas raten, dessen (Maximal)-Größe beschränkt ist.

Dadurch kann man NTM als DTM mit zusätzlicher Ratefunktion verstehen!

Nicht-deterministische 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)

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

Nicht-deterministische TMs

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

Konsequenzen der Charakterisierung:

- NP kann als das Äquivalent von P für NTMs angesehen werden
- Die P vs NP Frage ist dann:

Kann eine nicht-deterministische Maschine in polynomieller Zeit mehr erreichen als eine deterministische Maschine?

Kapitel 3

NP-Härte, NP-Vollständigkeit

NP-Härte

Da es bisher nicht gelungen ist, zu zeigen, dass "typische" NP-Probleme wie CLIQUE, SEQ, IPROG **nicht** in P sind, behilft man sich wie folgt:

Anstelle von **absoluten** Resultaten

(z.B. "Problem XYZ ist nicht in P")

betrachten wir **relative** Resultate

(z.B. "CLIQUE und SEQ sind gleich schwierig:
CLIQUE in P gdw SEQ in P").

Besonders 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 sehr wichtig in der Komplexitätstheorie:

- zentrales Werkzeug zum Vergleich verschiedener Probleme, nicht nur im Kontext von NP (später mehr)
- 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

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

f heisst *polynomiell* wenn f in polynomieller Zeit berechenbar.

Wenn Reduktion von L auf L' existiert: wir sagen L ist *reduzierbar* auf L' und schreiben $L \leq L'$

Polynomielle Reduzierbarkeit analog mit Notation $L \leq_p L'$.

Reduktionen

Intuitiv: $L \leq_p L'$ formalisiert “ L' ist mindestens so schwer wie L ”

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

Theorem

1. P ist abgeschlossen unter polynomiellen Reduktionen:
 $L' \in P$ und $L \leq_p L'$ impliziert $L \in P$;
2. NP ist abgeschlossen unter polynomiellen Reduktionen:
 $L' \in NP$ und $L \leq_p L'$ impliziert $L \in NP$;
3. “Polynomiell reduzierbar” ist transitive Relation:
 $L_1 \leq_p L_2$ und $L_2 \leq_p L_3$ 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 $P = \text{NP}$ Problem ungelöst ist, wird NP-härte deshalb als "nicht effizient lösbar" gewertet.

NP-Härte

Aber: warum sollte es überhaupt ein Problem geben, das mindestens so schwierig ist wie **jedes** Problem in NP?

Theorem

NP-vollständige Probleme existieren.

Idee: definiere "universelles" Problem in NP, das alle Probleme in NP "enthält".

Noch überraschender: es gibt sogar extrem viele und sehr natürliche NP-vollständige Probleme.

Kapitel 3

Cook's Theorem

NP-Härte

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

1. Zeigen, dass $L' \leq_p L$ für **alle** $L' \in \text{NP}$
Z.B. wie im vorigen Theorem: zeigen, dass das Wortproblem jeder polynomiell zeitbeschränkten NTM reduziert werden kann.
2. Zeigen, dass $L' \leq_p L$ für **ein NP-hartes** Problem L'

Theorem

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



SAT

Definition Formel, Wertzuweisung, Erfüllbar

Sei AV unendlich abzählbare Menge von *Aussagenvariablen*.
Menge der *aussagenlogischen Formeln* (kurz: AL-Formeln) ist kleinste Menge so dass:

- jedes $p \in AV$ ist AL-Formel
- wenn φ, ψ AL-Formeln, so auch $\neg\varphi, \varphi \vee \psi, \varphi \wedge \psi$

Wertzuweisung (kurz: WZ) ist Abbildung $\pi : AV \rightarrow \{0, 1\}$.

WZ π wird wie folgt auf zusammengesetzte AL-Formeln erweitert:

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

AL-Formel φ ist erfüllbar, wenn es WZ π gibt mit $\pi(\varphi) = 1$.

Cook's Theorem

Problem SAT ist Menge aller erfüllbaren AL-Formeln.

Theorem (Cook/Levin aka "Cooks Theorem")

SAT ist NP-vollständig.

In NP: $\{(\varphi, \pi) \mid \pi \text{ WZ für Variablen in } \varphi, \text{ die } \varphi \text{ erfüllt}\}$
ist polynomielles Beweissystem

NP-Härte:

Zeigen, dass Wortproblem **jeder** poly-zeitbeschränkten NTM polynomiell reduziert werden kann.

Cook's Theorem

Sei $L \in \text{NP}$ und M eine $p(n)$ -zeitbeschränkte NTM mit $L(M) = L$.

Berechnung von M auf $w = a_1 \cdots a_n$ ist Matrix:

▷	q_0, a_1	a_2	⋯	a_n	⊥	⋯	⊥	} $0..p(n)$
▷	b	q, a_2	⋯	a_n	⊥	⋯	⊥	
▷	b	q', b'	⋯	a_n	⊥	⋯	⊥	
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	

$0..p(n) + 1$

Ziel: gegeben w , finden von AL-Formel φ_w so dass

1. M akzeptiert w gdw. φ_w erfüllbar und
2. φ_w in Zeit polynomiell in $|w|$ konstruiert werden kann

Cook's Theorem

Repräsentation der Matrix mittels Variablen:

- $B_{a,i,t}$: zum Zeitpunkt t ist Zelle i mit a beschriftet;
- $K_{i,t}$: zum Zeitpunkt t ist der Kopf über Zelle i ;
- $Z_{q,t}$: zum Zeitpunkt t ist q der aktuelle Zustand

für alle $t \leq p(n) + 1$, $i \leq p(n)$, $a \in \Gamma$, $q \in Q$.

Gesuchte Formel φ_w ist Konjunktion folgender Formeln:

Berechnung beginnt mit Startkonfiguration für $w = a_1 \cdots a_n$:

$$\psi_{\text{ini}} := Z_{q_0,0} \wedge K_{1,0} \wedge B_{\triangleright,0,0} \wedge \bigwedge_{1 \leq i \leq n} B_{a_i,i,0} \wedge \bigwedge_{n < i \leq p(n)} B_{\perp,i,0}.$$

Cook's Theorem

Sei $R(i) = i + 1$, $N(i) = i$, $L(i) = i - 1$ wenn $i > 1$, $L(0) = 0$.

Schritte folgen der Übergangsrelation:

$$\psi_{\text{move}} := \bigwedge_{t < p(n), q \in Q \setminus \{q_{\text{acc}}, q_{\text{rej}}\}, a \in \Gamma, i \leq p(n)+1} \left((Z_{q,t} \wedge K_{i,t} \wedge B_{a,i,t}) \rightarrow \bigvee_{(q,a,q',a',M) \in \Delta, M(i) \leq p(n)+1} (Z_{q',t+1} \wedge K_{M(i),t+1} \wedge B_{a',i,t+1}) \right)$$

Zellen ohne Kopf ändern sich nicht:

$$\psi_{\text{keep}} := \bigwedge_{t < p(n), i \leq p(n)+1, a \in \Gamma} \left(\neg K_{i,t} \wedge B_{a,i,t} \rightarrow B_{a,i,t+1} \right)$$

Eingabe wird akzeptiert:

$$\psi_{\text{acc}} := \bigvee_{t \leq p(n)} \left(Z_{q_{\text{acc}},t} \wedge \bigwedge_{t' < t} \neg Z_{q_{\text{rej}},t'} \right)$$

Cook's Theorem

Bandbeschriftung, Kopfposition, Zustand sind eindeutig und definiert:

$$\begin{aligned} \psi_{\text{aux}} := & \bigwedge_{t,q,q',q \neq q'} \neg(Z_{q,t} \wedge Z_{q',t}) \wedge \bigwedge_{t,i,a,a',a \neq a'} \neg(B_{a,i,t} \wedge B_{a',i,t}) \wedge \bigwedge_{t,i,j,i \neq j} \neg(K_{i,t} \wedge K_{j,t}) \\ & \bigwedge_{t \leq p(n)} \bigvee Q \wedge \bigwedge_{t \leq p(n)} \bigvee_{i \leq p(n)+1} K_{i,t} \wedge \bigwedge_{t \leq p(n), i \leq p(n)+1} \bigvee \Gamma \end{aligned}$$

Wir setzen nun

$$\varphi_w := \psi_{\text{ini}} \wedge \psi_{\text{move}} \wedge \psi_{\text{keep}} \wedge \psi_{\text{acc}} \wedge \psi_{\text{aux}}.$$

Leicht zu sehen:

φ hat Länge $\mathcal{O}(p(n)^2)$, kann in Zeit $\mathcal{O}(p(n)^2)$ generiert werden.

Lemma

φ_w erfüllbar gdw. M akzeptiert w



Kapitel 3

Weitere NP-vollständige Problem

3SAT

Für NP-Härte eines Problems L :

es reicht, zu zeigen, dass $\text{SAT} \leq_p L$

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

Definition 3SAT

Literal ist Aussagenvariable oder deren Negation.

Klausel ist Disjunktion von Literalen

Formel in *Klauselform* ist Konjunktion von Klauseln

3-Klausel ist Klausel mit genau 3 Literalen

3-Formel ist Konjunktion von 3-Klauseln

3SAT ist die Menge aller erfüllbaren 3-Formeln.

3SAT ist offensichtlich in NP, da SAT in NP!

3SAT

Theorem

3SAT ist NP-vollständig.

Beweis Härte: polynomielle Reduktion von SAT

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

Idee:

- zuerst wandeln wir φ in Negationsnormalform
- 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

Schritt 1: NNF

Definition NNF

AL-Formel φ ist in *Negations-Normalform (NNF)* wenn Negation in φ nur direkt vor Variablen vorkommt (nicht vor zusammengesetzten Formeln!)

Zwei AL-Formeln φ und ψ sind *äquivalent* wenn für alle WZ π gilt:
 π erfüllt φ gdw. π erfüllt ψ

Jede AL-Formel kann in polynomieller Zeit in äquivalente Formel in NNF gewandelt werden.

Erschöpfendes Anwenden folgender Regeln:

- $\neg\neg\varphi \longrightarrow \varphi$ (doppelte Negation)
- $\neg(\varphi \vee \psi) \longrightarrow \neg\varphi \wedge \neg\psi$ (de Morgan)
- $\neg(\varphi \wedge \psi) \longrightarrow \neg\varphi \vee \neg\psi$ (de Morgan)

Schritt 2: Teilformeln

Definition Teilformel

Die Menge $TF(\varphi)$ aller *Teilformeln* einer AL-Formel φ ist induktiv wie folgt definiert:

- $TF(p) = \{p\}$;
- $TF(\neg\varphi) = \{\neg\varphi\} \cup TF(\varphi)$;
- $TF(\varphi \sim \psi) = \{\varphi \sim \psi\} \cup TF(\varphi) \cup TF(\psi)$ für $\sim \in \{\wedge, \vee\}$;

Leicht zu sehen:

die Anzahl Formeln in $TF(\varphi)$ ist polynomiell in der Länge von φ .

Noch
zu tun:

- führe Aussagenvariable für jede Teilformel von φ ein
- beschreibe das Verhalten von Teilformeln in AL
- Konvertiere entstehende Formel in 3-Formel



CLIQUE

Theorem

CLIQUE ist NP-vollständig.

Beweis Härte: polynomielle Reduktion von 3SAT

Zunächst eine vorbereitende Beobachtung

Zwei Literale l_1 und l_2 sind *komplementär* wenn sie die Form $l_1 = p$ und $l_2 = \neg p$ haben oder umgekehrt.

Lemma

Eine 3-Formel $c_0 \wedge \dots \wedge c_n$ ist erfüllbar gdw. es Literale l_0, \dots, l_n gibt, so dass für alle $i, j \leq n$ mit $i \neq j$:

1. l_i kommt in c_i vor und
2. l_i und l_j sind nicht komplementär.



CLIQUE

Theorem

CLIQUE ist NP-vollständig.

Beweis Härte: polynomielle Reduktion von 3SAT

Gegeben 3-Formel φ , konstruiere in polynomieller Zeit Graph G und wähle $k \geq 1$ so dass φ erfüllbar gdw. G k -Clique enthält.

Idee:

- G hat Knoten für jedes Vorkommen eines Literals in φ
- Kante zwischen zwei Literalen gdw. sie in unterschiedlichen Klauseln vorkommen und nicht komplementär sind
- Als k wähle Anzahl der Klauseln



3-Färbbarkeit

Viele Probleme auf Graphen sind NP-Vollständig, siehe VL
“Algorithmen auf Graphen” von H-J Kreowski

Wir betrachten ein weiteres Beispiel

Definition 3-Färbbarkeit

Ungerichteter Graph $G = (V, E)$ ist *3-färbbar* gdw. 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.

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 einen Knoten für jedes Literal ein
- verwende die Farben “wahr”, “falsch” und “hilf”
- verbinde komplementäre Literale mit Kanten, um konsistente WZ zu erzwingen
- verwende Teilgraphen, um Erfülltsein jeder Klausel zu erzwingen



NP-Vollständigkeit

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

1972 veröffentlichte Richard Karp in einem berühmten Aufsatz 21 solche (und sehr verschiedene) 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
- Sequenzierung

I PROG

Zur Erinnerung:

Definition Integer Programming

Sei V eine Menge von Variablen und G eine Menge von linearen Gleichungen

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

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

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

I PROG ist Menge aller Gleichungssysteme G , für die 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
- Gleichung stellt sicher, dass diese Variablen konsistente Werte aus dem Bereich $\{0, 1\}$ erhalten
- Zusätzliche Gleichungen und numerische “Auffangvariablen” stellen sicher, dass jede Klausel wahres Literal enthält



Sequenzierung ist NP-Vollständig

Zur Erinnerung:

Definition (Single-Prozessor) Sequenzierung

Sei A eine Menge von *Aufgaben*, die durch \prec partiell geordnet ist und $d : A \rightarrow \mathbb{N}$ Funktion, die Deadlines beschreibt.

Sequenzierung für A mit $k \in \mathbb{N}$ Verspätungen ist injektive Abbildung

$\tau : A \rightarrow \{0, \dots, |A| - 1\}$ so dass

1. $a \prec a'$ impliziert $\tau(a) < \tau(a')$;
2. für höchstens k Aufgaben $a \in A$ gilt $\tau(a) > d(a)$.

Theorem

Sequenzierung ist NP-vollständig.

Enthaltensein in NP schon gezeigt.

Sequenzierung ist NP-Vollständig

Härtebeweis durch Reduktion des Cliquesproblems:

Gegeben Graph G und Cliquesgröße c , konstruiere in polynomieller Zeit Menge von Aufgaben A mit assoziiertem d und \prec und Verspätungszahl k so dass G eine c -Clique hat gdw. es Sequenzierung für A mit k Verspätungen gibt.

Idee:

- verwende sowohl Knoten als auch Kanten als Aufgaben
- bestimme "globale" deadline, vor der **alle** nicht verspäteten Kanten sequenziert werden müssen
- erzwingen mittels \prec , dass alle Kanten nach ihren Endpunkten sequenziert werden
- wähle globale Deadline und Anzahl Verspätungen so, dass nur die Knoten und Kanten einer Clique vor die deadline "passen"



NP Vollständigkeit

NP-vollständige Probleme sind überall:

- Minesweeper. Gegeben den momentanen Stand eines Minesweeper Spieles (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?
- etc.

Kapitel 3

Reflektion zu NP-Härte

Kombinatorik

NP-harte Probleme sind oft "kombinatorisch", wie logische Puzzles

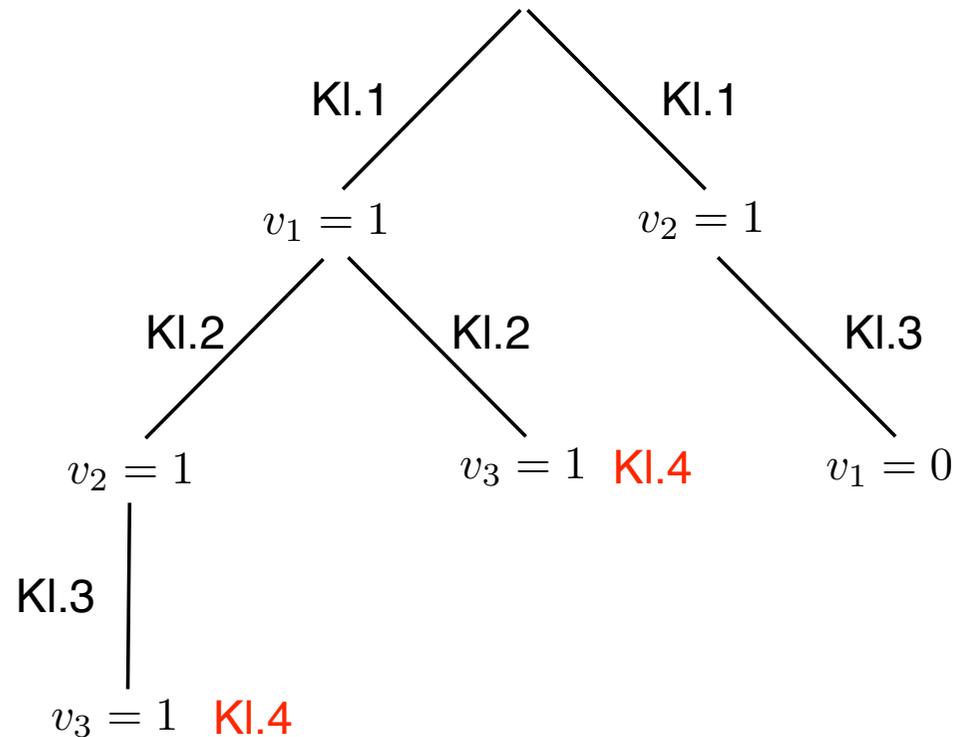
z.B. 3SAT:

$$(v_1 \vee v_2)$$

$$\wedge (\neg v_1 \vee v_2 \vee v_3)$$

$$\wedge (\neg v_1 \vee \neg v_2 \vee v_3)$$

$$\wedge (\neg v_1 \vee \neg v_3)$$



Aber nicht alle "kombinatorisch aussehenden" Probleme sind auch wirklich NP-hart!!

Bipartites Matching

Beispiel für ein Problem, das erstmal NP-hart "aussieht",
sich aber als polynomiell herausstellt

Definition Bipartiter Graph

Ein ungerichteter Graph $G = (V, E)$ ist *bipartit* wenn V in zwei Mengen V_1 und V_2 partitioniert ist und $\{v, v'\} \in E$ impliziert dass (i) $v \in V_1$ und $v' \in V_2$ oder (ii) $v' \in V_1$ und $v \in V_2$.

Definition Matching

Matching für Bipartiten Graphen $G = (V, E)$ ist Teilmenge $M \subseteq E$ so dass es für jeden Knoten $v \in V$ höchstens eine Kante $K \in E$ gibt mit $v \in K$.

Bipartites Matching

Beispiel Anwendung:

Sei V_1 Menge von Aufgaben, V_2 Menge von Maschinen und $E \subseteq V_1 \times V_2$
Zuordnung von Aufgaben zu Maschinen, die diese bearbeiten können

Matching **mit maximaler Kardinalität** beschreibt, wie man maximal viele
Aufgaben gleichzeitig bearbeiten kann

”Heiratsproblem”: V_1 Menge Frauen, V_2 Menge Männer, $E \subseteq V_1 \times V_2$
Relation “würde heiraten”, möglichst viele Hochzeiten gewünscht

Eigentlich also Optimierungsproblem; hier Entscheidungsvariante:

Problem MATCH: Paare (G, k) mit G bipartiter Graph und $k \geq 0$,
so dass es Matching M für G gibt mit $|M| \geq k$.

Bipartites Matching

"Sieht kombinatorisch aus":

- man kann gegebenes Matching nicht in jedem Fall zu einem mit maximaler Kardinalität erweitern
- muss ggf. Kanten wieder "wegnehmen"

Ein gieriger ("greedy") Ansatz funktioniert also nicht.

Dennoch:

Theorem

MATCH ist in P

Im folgenden: *Pfad* in $G = (V, E)$ ist Knotenfolge v_1, \dots, v_k so dass

- $(v_i, v_{i+1}) \in E$ für $1 \leq i < k$
- alle v_i verschieden sind.

Bipartites Matching

Definition Alternierender Pfad; erweiternder Pfad

Sei M ein Matching. Pfad v_1, \dots, v_k ist M -alternierend wenn

$$(v_i, v_{i+1}) \in M \text{ gdw. } (v_{i+1}, v_{i+2}) \notin M \text{ f\u00fcr alle } i < k - 2$$

Alternierender Pfad v_1, \dots, v_k ist M -erweiternd wenn

weder v_1 noch v_k eine adjazente M -Kante hat.

Offensichtlich f\u00fcr jeden erweiternden Pfad P :

- P hat k Kanten in $M \implies P$ hat $k + 1$ Kanten ausserhalb M
- $M' = M \Delta P = (M \setminus P) \cup (P \setminus M)$ ist Matching mit $|M'| > |M|$

Lemma

Matching M hat maximale Kardinalit\u00e4t gdw. es keinen M -erweiternden Pfad gibt.

Bipartites Matching

Folgender Algorithmus entscheidet also Matching:

```
procedure match( $G, k$ )  
  if  $|E| = 0$  then  
    if  $k = 0$  then return "true" else return "false"  
  endif  
   $M := \{(v, v')\}$  für beliebiges  $(v, v') \in E$   
  while es gibt  $M$ -erweiternden Pfad  $P$  do  
     $M := M \Delta P$   
  endwhile  
  if  $k \leq |M|$  then return "true" else return "false"
```

Terminiert offensichtlich nach max. $|E|$ Iterationen

Noch zu zeigen (Aufgabe): man kann in polynomieller Zeit die Existenz eines erweiternden Pfades entscheiden und diesen Ggf. ausrechnen

Kapitel 3

Einige weitere Themen rund um P und NP:

- Probleme, die weder NP-hart noch in P sind
- Isomorphie von NP-Problemen
- Komplemente und co-NP

Kapitel 3

NP-Intermediate

NP Intermediate

Die Resultate mögen den Eindruck vermitteln, dass wenn $P \neq NP$ ist, dann für jedes Problem $L \in NP \setminus P$ gilt: L ist NP-vollständig.

Dieser Eindruck ist nicht korrekt!

Definiere Komplexitätsklasse NPI (NP-Intermediate):

Menge aller Probleme $L \in NP \setminus P$, die nicht NP-vollständig sind.

Ladner's Theorem

Wenn $P \neq NP$, dann $NPI \neq \emptyset$.

Zeigt, dass NP-Vollständigkeit noch wichtig ist wenn $P \neq NP$ bewiesen!

Ladner's Theorem

Offensichtlich:

Es gibt Aufzählung M_1, M_2, \dots aller DTMs, so dass jede DTM unendlich oft vorkommt

Beweisidee:

- starte mit SAT, fülle aber die Eingaben aber mit zusätzlichen sinnlosen Zeichen auf (Padding)
- dadurch hat die TM mehr Zeit zum Lösen des Problems zur Verfügung
- Wähle die Länge des Paddings so, dass das entstehende Problem nicht mehr NP-hart ist, aber auch noch nicht in P

Ladner's Theorem

Jede Funktion $H : \mathbb{N} \rightarrow \mathbb{N}$ definiert Variante von SAT wie folgt:

$$\text{SAT}_H := \{\varphi 01^{n^{H(n)}} \mid \varphi \in \text{SAT und } n = |\varphi|\}$$

Aus jeder Eingabe der Länge n wird also eine der Länge $\mathcal{O}(n^{P(n)})$

Die Definition der gewünschten Funktion H ist rekursiv:

$$H(n) = \min\{i < \log \log n \mid \text{für alle Eingaben } x \leq \log n \text{ berechnet } M_i \\ \text{SAT}_H(x) \text{ in } i|x|^i \text{ Schritten}\}$$


Rekursion

$$H(n) = \log \log n \text{ wenn kein solches } i \text{ existiert}$$

Intuitiv: $H(n)$ gibt Index der "ersten" DTM unter den "ersten wenigen" DTMs, die ein Anfangsstück von $H(n)$ in polynomieller Zeit berechnet

Ladner's Theorem

$$\text{SAT}_H := \{\varphi 01^{n^{H(n)}} \mid \varphi \in \text{SAT} \text{ und } n = |\varphi|\}$$

$H(n) = \min\{i < \log \log n \mid \text{für alle Eingaben } x \leq \log n \text{ berechnet } M_i$
 $\text{SAT}_H(x) \text{ in } i|x|^i \text{ Schritten}\}$

H ist wohldefiniert, denn

- $H(n)$ **bestimmt** Zugehörigkeit von Strings der Länge $> n$ zu SAT_H
- die **Definition** von $H(n)$ betrachtet Strings der Länge $\max. \log n$

Lemma

1. $\text{SAT}_H \in \text{P}$ gdw. $H(n) \in \mathcal{O}(1)$
2. Wenn $\text{SAT}_H \notin \text{P}$, dann geht $H(n)$ gegen unendlich
3. H ist in Polyzeit berechenbar (Übung)

NP Intermediate

Man kann Ladner's Theorem noch signifikant stärker machen:

Ladner's Theorem, reloaded

Wenn $P \neq NP$, dann gibt es unendliche Folge von Sprachen L_1, L_2, \dots in NPI so dass für alle $i \geq 1$ gilt: $L_{i+1} \leq_p L_i$ und $L_i \not\leq_p L_{i+1}$.

Bisher konnte kein natürliches Problem in NPI identifiziert werden
(nicht mal unter der Annahme $P \neq NP$)

Kandidaten:

- Graph Isomorphismus. Gegeben ungerichtete Graphen G_1, G_2 , kann man die Knoten in G_2 umbenennen so dass $G_1 = G_2$?
- Integer Faktorisierung. Gegeben $i, j \in \mathbb{N}$, hat i einen Faktor kleiner als j ?

Kapitel 3

Isomorphie von NP-Problemen

Isomorphievermutung

NP-vollständige Probleme scheinen sich alle ähnlich zu sein.

Beobachtung

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

Die Reduktionen sind aber nicht unbedingt strukturerhaltend: ●

- Injektivität nicht gewährleistet
z.B. $3SAT \leq_p CLIQUE$: konsistentes Austauschen von Variablen ändert konstruierten Graph/Cliquenzahl nicht.
- Surjektivität nicht gewährleistet
z.B. $3SAT \leq_p 3F$: nur Graphen mit bestimmter Struktur konstruiert
- Reduktionen $L \leq_p L'$ und $L' \leq_p L$ sind unabhängig voneinander

Wie ähnlich sind NP-vollständige Probleme **genau**?

Isomorphievermutung

Wir

- fordern Injektivität und Surjektivität
- verschmelzen die zwei unabhängigen Reduktionen zu einer bidirektionalen

Definition p -Isomorphismus

Polynomielle Reduktion $f : \Sigma^* \rightarrow \Gamma^*$ von L' auf L ist *p -Isomorphismus* wenn

- f *Bijektion* ist (also injektiv und surjektiv)
- nicht nur f , sondern auch f^{-1} in Polyzeit berechenbar ist.

Existiert so ein f sind L' und L *p -isomorph*.

Leicht zu sehen: f^{-1} ist Polyzeit-Reduktion von L auf L'

Isomorphievermutung

Vermutung (Berman/Hartmanis)

Alle NP-vollständigen Probleme sind paarweise p-isomorph.

Intuitiv sagt diese Vermutung: alle NP-vollständigen Probleme sind dasselbe Problem, in unterschiedlicher "Verkleidung"

Sollte die Vermutung wahr sein, so ist sie nicht leicht zu beweisen:

Theorem

Wenn alle NP-vollständigen Probleme paarweise p-isomorph sind, dann $P \neq NP$.

Man nimmt heute eher an, dass die Isomorphievermutung falsch ist.
(andererseits gelten ähnliche Vermutungen in der Rekursionstheorie)

Kapitel 3

Komplemente und co-NP

Motivation

Subtile Frage:

Sei $\overline{3F}$ die Menge aller **nicht** 3-färbbaren ungerichteten Graphen.
In welcher Komplexitätsklasse ist $\overline{3F}$?

Um in NP zu sein, müsste es polynomielles Beweissystem geben.

Aber was ist polynomieller Beweis dafür, dass Graph **nicht** 3-färbbar?

Z.B. Menge aller Färbungen mit 3 Farben; exponentiell viele!

Es ist nicht bekannt, ob es für $\overline{3F}$ polynomielles Beweissystem gibt!

Komplement

Definition Komplement

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

Komplexitätsklasse \mathcal{C} ist *abgeschlossen unter Komplement* wenn:

$$L \in \mathcal{C} \text{ impliziert } \bar{L} \in \mathcal{C}$$

Z.B. $\exists F$

$\overline{\text{SAT}}$ ist die Menge aller unerfüllbaren AL-Formeln

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

Lemma

Deterministische Zeitkomplexitätsklassen wie $\text{DTIME}(f)$ und P sind abgeschlossen unter Komplement.

Also z.B. $\overline{\text{GAP}} \in P$ und $\overline{2\text{SAT}} \in P$

Komplement

Dieses Argument schlägt fehl für nicht-deterministische TMs,
funktioniert also nicht für die alternative Charakterisierung von NP! ●

Definition Komplementklassen

Wenn \mathcal{C} Komplexitätsklasse, dann $\text{co-}\mathcal{C} := \{\bar{L} \mid L \in \mathcal{C}\}$.

Wegen $P = \text{co-}P$ wird $\text{co-}P$ nicht betrachtet, co-NP aber sehr wohl!

Z.B. $\overline{3F}$, $\overline{\text{SAT}}$, $\overline{\text{CLIQUE}}$ $\in \text{co-NP}$ nach Definition.

Definition Gültigkeit in Aussagenlogik

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

Lemma

Gültigkeit ist in co-NP . ●

Vollständigkeit

Härte und Vollständigkeit sind definiert wie für NP:

Definition co-NP-Härte, co-NP-Vollständigkeit

Problem L ist

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

Lemma

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

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

Lemma

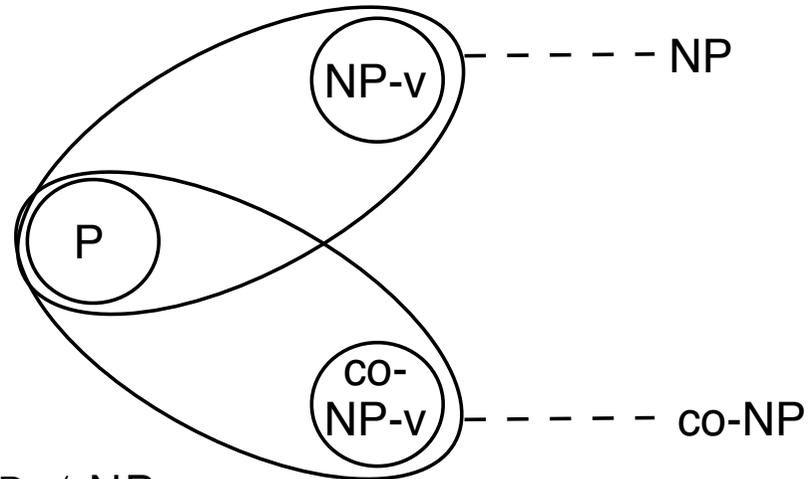
Gültigkeit ist co-NP-vollständig.

P vs NP vs co-NP

Leicht zu sehen: $P \subseteq \text{co-NP}$

Es wird vermutet, dass

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



Beide Vermutungen implizieren $P \neq \text{NP}$
(aber nicht umgekehrt)

Lemma

Wenn $\text{NP} \neq \text{co-NP}$ und $L \in \text{NP} \cap \text{co-NP}$, dann ist L nicht NP-vollständig.

Es sind also alle Probleme in $\text{NP} \cap \text{co-NP}$ in P oder NP-Intermediate!

Kandidat: Integer Faktorisierung ist in NP und co-NP

Übersicht Vorlesung

- Kapitel 1: Einführung
- Kapitel 2: Turingmaschinen
- Kapitel 3: P vs. NP
- Kapitel 4: Mehr Ressourcen, mehr Möglichkeiten?
- Kapitel 5: Platzkomplexität
- Kapitel 6: Zeitkomplexität Reloaded