Technische Universität Dresden

Theoretische Informatik und Logik

9. Vorlesung: NP-Vollständigkeit

Markus Krötzsch

Professur Wissensbasierte Systeme

TU Dresden, 18. Mai 2026

Rückblick

PTime und LogSpace als mathematische Modelle für Effizienz:

- PTime als robuste Verallgemeinerung der in linearer Zeit lösbaren Probleme
- LogSpace als typische (vermutlich) subpolynomielle Klasse

Wichtige Anwendung: Reduktionen, die ein Problem mit wenig Aufwand auf ein anderes zurückführen

- Polynomielle (Many-One-)Reduktionen: verbreitetste Form von effizienter Reduktion; Notation: $\leq_p$
- LogSpace-Reduktionen: häufig anzutreffen, aber selten im Detail definiert

Effizient und doch nicht praktikabel

Es gibt Situationen, in denen ein Problem in PTime liegt und dennoch nicht praktisch algorithmisierbar ist.

Satz: Jede endliche Sprache kann in DTIME(1) erkannt werden und liegt daher insbesondere in PTime und LogSpace.

Beispiel: Sei **L** die Sprache, die alle wahren Aussagen aus der folgenden Menge enthält: {„P = NP“, „P ≠ NP“}. Dann ist **L** ∈ PTime und damit effizient berechenbar.

Erkenntnis: Man kann manchmal die Existenz eines effizienten Algorithmus beweisen, ohne zu wissen, wie er aussehen müsste.
⇝ nichtkonstruktiver Beweis

Auch (möglicherweise) unendliche Sprachen können sich so verhalten; siehe die Sprache $\mathbf{L}_{\pi 7}$ aus Formale Systeme, Vorlesung 20.

Probleme vergleichen mit Reduktionen

Intuition:

$$\mathbf{P} \leq_p \mathbf{Q}$$

bedeutet

„**Q** ist mindestens genauso schwer wie **P**“

- Einordnung in Komplexitätsklassen: obere Schranke
  („**P** ist mit einem bestimmten Aufwand lösbar“)
- Reduktion auf andere Probleme: untere Schranke
  („**Q** benötigt mindestens so viel Aufwand wie **P**“)

Warum sollten Reduktionen effizient sein?

Intuition: Eine aufwändige Reduktion kann jedes Problem indirekt lösen, aber dadurch lernt man nichts interessantes über dessen Komplexität.

Satz: Sei $\mathbf{L} = \{\mathtt{a}\}$ eine Sprache über dem Alphabet $\{\mathtt{a}\}$. Falls **P** entscheidbar ist, dann gibt es eine Many-One-Reduktion $\mathbf{P} \leq_m \mathbf{L}$.

Beweis: Die Reduktion f funktioniert wie folgt:

- Wenn $w \in \mathbf{P}$, dann sei $f(w) = \mathtt{a}$.
- Andernfalls, wenn $w \notin \mathbf{P}$, dann sei $f(w) = \mathtt{aa}$. □

Der Vortrag ohne Worte

Frank Nelson Cole

Der Vortrag ohne Worte

Am 31. Oktober 1903 auf einem Treffen der American Mathematical Society tritt Cole nach vorn und beginnt, an die Tafel zu schreiben . . .

$$
\begin{aligned}
& 2^{67} - 1 \\
&= \ldots \\
&= \ldots \\
&= \ldots \\
&= 147\,573\,952\,589\,676\,412\,927
\end{aligned}
$$

$$
\begin{aligned}
& 193\,707\,721 \times 761\,838\,257\,287 \\
&= \ldots \\
&= \ldots \\
&= \ldots \\
&= 147\,573\,952\,589\,676\,412\,927
\end{aligned}
$$

Damit endet Coles stundenlanger Vortrag, ohne dass ein Wort gesprochen wurde. Das Publikum steht auf und applaudiert.[1]

[1] Sicher ist, dass Cole die Faktoren der Mersenne-Zahl $2^{67} - 1$ als erster ermitteln konnte.
Die historischen Belege für den genauen Ablauf des Vortrags sind dagegen eher schwach.
In Italien würde man sagen: „Si non è vero, è ben trovato“ – Falls es nicht stimmt, so ist es doch gut erfunden!

Rückblick: Polynomielle Verifikatoren

Die Geschichte zu Cole's Vortrag veranschaulicht:
Manche Probleme scheinen schwer zu lösen, aber ihre Lösung ist – wenn sie erst einmal vorliegt – leicht zu überprüfen.

- Formalisierung durch polyomielle Verifikatoren
  $\rightsquigarrow$ nachweispolynomielle Sprachen
- Satz: eine Sprache ist genau nachweispolynomiell wenn sie in NP liegt

Komplemente von Sprachen in NP sind in coNP.

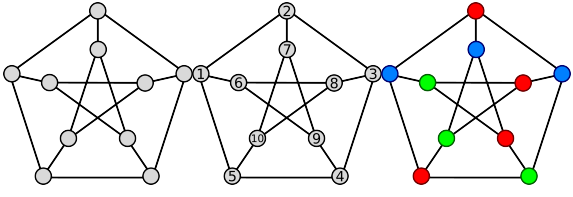
- Vermutlich sind diese Klassen unterschiedlich
- $P \subseteq coNP \cap NP$
- Es gibt auch (scheinbar) schwere Probleme in $coNP \cap NP$,
  z.B. Faktorisierung (siehe Cole)

NP-Vollständigkeit

Noch eine Reduktion

Manche Probleme lassen sich auf andere zurückführen

Beispiel: Drei-Farben-Problem ist auf **SAT** reduzierbar.



Darstellung von Farben mit Atomen:

- r_i bedeutet „Knoten i ist rot“
- g_i bedeutet „Knoten i ist grün“
- b_i bedeutet „Knoten i ist blau“

Färbe-Bedingungen für Knoten:

$(r_1 \wedge \neg g_1 \wedge \neg b_1) \vee (\neg r_1 \wedge g_1 \wedge \neg b_1) \vee (\neg r_1 \wedge \neg g_1 \wedge b_1)$ (usw. für alle Knoten)

Färbe-Bedingungen für Kanten:

$\neg(r_1 \wedge r_2) \wedge \neg(g_1 \wedge g_2) \wedge \neg(b_1 \wedge b_2)$ (usw. für alle Kanten)

Erfüllende Wertzuweisung $\Leftrightarrow$ zulässige Färbung

Rückblick: Polynomielle Reduktionen

Eine Funktion $f : \Sigma^* \to \Sigma^*$ ist polynomiell berechenbar wenn es eine polynomiell-zeitbeschränkte TM gibt, die bei einer Eingabe w die Ausgabe $f(w)$ auf das Band schreibt und anschließend anhält.

Eine polynomiell berechenbare Funktion $f : \Sigma^* \to \Sigma^*$ ist eine polynomielle Reduktion von einer Sprache **L** auf eine Sprache **G**, wenn für alle Wörter $w \in \Sigma^*$ gilt:

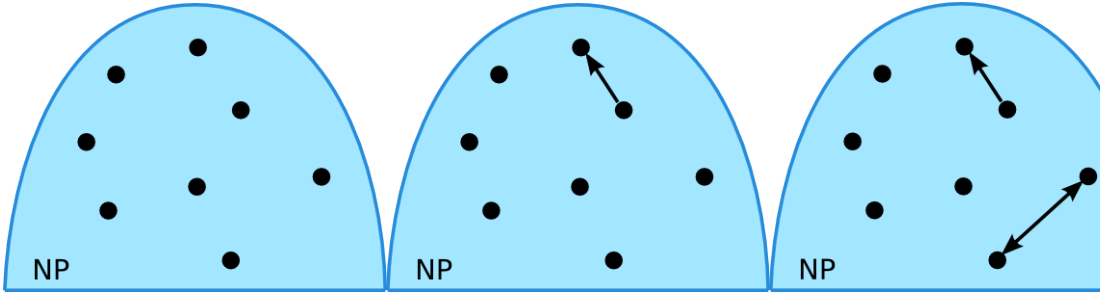
$$w \in \mathbf{L} \quad \text{genau dann wenn} \quad f(w) \in \mathbf{G}$$

Beispiel: Die soeben skizzierte Funktion von Graphen auf logische Formeln ist eine polynomielle Reduktion des Drei-Farben-Problems auf **SAT**.

Beispiel: In Vorlesung 7 haben wir **2Col** auf **2SAT** reduziert. Diese Reduktion war einfacher, da wir Wahrheitswerte direkt als Farben verwenden konnten.

Die Struktur von NP

Idee: polynomielle Reduzierbarkeit definiert eine Art Ordnung auf Problemen



Das schwerste Problem in NP?

Stephen Cook

Leonid Levin

Richard Karp

Satz von Cook [1971] & Levin [1973]: Alle Probleme in NP können polynomiell auf **SAT** reduziert werden.

- Wer **SAT** polynomiell löst, der hat alle Probleme in NP polynomiell gelöst
- Es gibt in NP eine maximale Klasse, die ein praktisch interessantes Problem enthält
- Karp findet kurz darauf 21 weitere solcher Probleme (1972)
- Seitdem wurden tausende weitere entdeckt ...

NP-Schwere und NP-Vollständigkeit

Eine Sprache ist

- NP-schwer, wenn jede Sprache in NP polynomiell darauf reduzierbar ist
- NP-vollständig, wenn sie NP-schwer ist und in NP liegt

Beispiel: SAT ist NP-vollständig (Cook & Levin).

Beispiel: PRIMES ist in NP, aber vermutlich nicht NP-schwer. Gleiches gilt für viele Probleme in P (bei einigen ist dagegen sicher, dass sie nicht NP-schwer sind).

Beispiel: Das Halteproblem ist NP-schwer aber sicher nicht in NP. Gleiches gilt für viele unentscheidbare Probleme.

Beispiele

Beispiel: Das Drei-Farben-Problem ist NP-vollständig.

Beispiel: Verallgemeinertes Sudoku (n^2 Zahlen in einem $n^2 \times n^2$-Feld mit $n \times n$-Unterfeldern) ist NP-vollständig. Das Entscheidungsproblem dabei ist, ob das gegebene, teilweise gefüllte Feld eine Lösung zulässt. Gleiches gilt für Verallgemeinerungen vieler anderer Solitärspiele (z.B. Minesweeper).

Beispiel: Das Wortproblem für Typ-1-Sprachen (gegeben als Grammatik) ist NP-schwer aber vermutlich nicht in NP. Gleiches gilt für das Halteproblem von LBAs.

Beispiel: Das Problem des Handelsreisenden ist NP-vollständig. Es besteht darin, in einem Graph mit bewerteten Kanten einen Pfad durch alle Knoten zu finden, so dass die Summe der Kantenwerte unter einem gegebenen Maximum liegt.

Wie findet man NP-schwere Probleme?

Schwer ist, worauf sich schwere Dinge leicht zurückführen lassen:

Satz: Wenn **L** NP-schwer ist und polynomiell auf **G** reduziert werden kann, dann ist **G** auch NP-schwer.

Beweis: Folgt direkt aus der Definition, da die Komposition von zwei polynomiellen Reduktionen ebenfalls eine polynomielle Reduktion ist. □

Technik zum Beweis von NP-Schwere:

- Suche ein bekanntes NP-schweres Problem, welches dem untersuchten Problem möglichst ähnlich ist.
- Gib eine polynomielle Reduktion an und zeige, dass sie korrekt ist.

$\rightsquigarrow$ Oft nicht schwer, wenn man bereits viele NP-schwere Probleme kennt – aber wo fängt man an?

Cook/Levin: Beweisskizze (1)

Wie stellen wir TM-Berechnungen in aussagenlogisch dar?

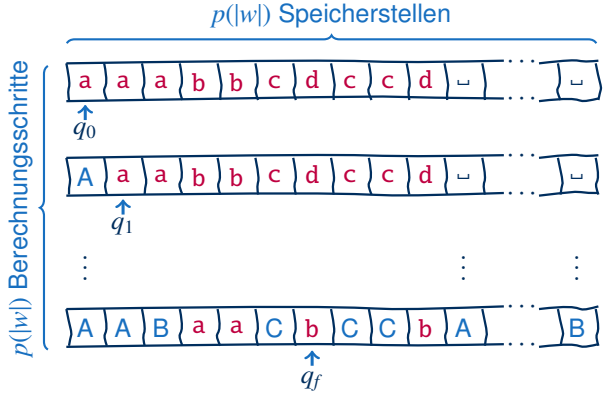
Idee:

- Jedes Problem in NP ist durch eine konkrete NTM $\mathcal{M}$ in einer Zeit lösbar, die durch ein konkretes Polynom p gegeben ist.
- Für eine Eingabe $|w|$ kennen wir also die maximale Zahl an Schritten $p(|w|)$ dieser Berechnung.

$\rightsquigarrow$ Ein Lauf ist eine Folge von maximal $p(|w|)$ Konfigurationen, die jeweils maximal $p(|w|)$ Speicherstellen verwenden

Ansatz: Kodiere $\mathcal{M}$ und w so in Logik, dass die möglichen Läufe genau den erfüllenden Wertzuweisungen entsprechen

Cook/Levin: Beweisskizze (2)



Wir können diesen Lauf mit aussagenlogischen Atomen kodieren:

- $Q_{q,s}$: „TM ist in Schritt s in Zustand q“
- $P_{i,s}$: „TM-Kopf ist in Schritt s an Position i“
- $S_{a,i,s}$: „Das Band enthält in Schritt s Zeichen a an Position i“

Cook/Levin: Beweisskizze (3)

Wir können diesen Lauf mit aussagenlogischen Atomen kodieren:

- $Q_{q,s}$: „TM ist in Schritt s in Zustand q“
- $P_{i,s}$: „TM-Kopf ist in Schritt s an Position i“
- $S_{a,i,s}$: „Das Band enthält in Schritt s Zeichen a an Position i“

$\rightsquigarrow$ Nur polynomiell viele Atome nötig

Mithilfe von Formeln kann man alle Bedingungen ausdrücken, die für einen korrekten Lauf der NTM gelten müssen.

Beispiel: Die Übergangsrelation wird z.B. mit Formel der folgenden Form kodiert:

$$(Q_{q,s} \wedge P_{i,s} \wedge S_{a,i,s}) \rightarrow \bigvee_{\substack{\langle q',b,D\rangle \in \delta(q,a) \\ i'=D(i)}} (S_{b,i,s+1} \wedge Q_{q',s+1} \wedge P_{i',s+1})$$

(für alle $s \in \{1,\ldots,p(|w|)\}$ und Positionen $i \in \{1,\ldots,p(|w|)\}$).

Weitere NP-vollständige Probleme

Clique

Eine Clique ist ein Graph, bei dem jeder Knoten mit jedem anderen direkt durch eine Kante verbunden ist

Clique

Gegeben: Ein Graph G und eine Zahl k

Frage: Enthält G eine Clique mit k Knoten?

Satz: Clique ist NP-vollständig.

Beweis:

(1) **Clique** $\in$ NP: Die Clique selbst ist ein geeignetes Zertifikat.

(Kontrollfrage: ist dieses Zertifikat wirklich polynomiell? Ist k in Binärkodierung gegeben, dann ist diese Eingabe schließlich nur $\log(k)$ Zeichen lang ...)

(2) Clique ist NP-schwer. Dazu konstruieren wir eine Reduktion von **SAT**.

Clique: Schwere (1)

Beweis Teil (2): Clique ist NP-schwer.

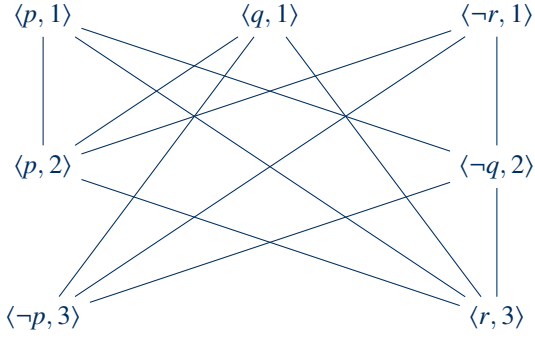
- Sei F eine aussagenlogische Formel in CNF:
  $F = \left((L_1^1 \vee \ldots \vee L_{n_1}^1) \wedge \ldots \wedge (L_1^\ell \vee \ldots \vee L_{n_\ell}^\ell)\right)$
- Wir definieren einen Graphen G_F, so dass gilt:

  G_F hat eine Clique der Größe ℓ gdw. F ist erfüllbar
- Knoten von G_F: die Paare $\langle L_j^i, i\rangle$, für alle $i \in \{1, \ldots, \ell\}$ und $j \in \{1, \ldots, n_i\}$
- Kanten von G_F: alle Paare $\langle L, i\rangle - \langle L', j\rangle$, für die gilt:
  (1) $i \neq j$ und
  (2) $L \wedge L'$ ist erfüllbar, d.h. $L \neq \neg L'$ und $L' \neq \neg L$

Offensichtlich kann man G_F in polynomieller Zeit berechnen

Clique: Schwere (2)

Beispiel: $F = ((p \vee q \vee \neg r) \wedge (p \vee \neg q) \wedge (\neg p \vee r))$



Erfüllende Belegung: $p \mapsto 1, q \mapsto 0, r \mapsto 1$

Man sieht leicht, dass unsere Reduktion korrekt ist. □

Unabhängige Mengen

Eine unabhängige Menge ist eine Teilmenge von Knoten in einem Graph, bei der kein Knoten mit einem anderen direkt verbunden ist

Unabhängige Menge

Gegeben: Ein Graph G und eine Zahl k

Frage: Enthält G eine unabhängige Menge mit k Knoten?

Satz: Unabhängige Menge ist NP-vollständig.

Beweis: Die Reduktion auf **Clique** ist sehr einfach:

- Ein Graph hat eine unabhängige Menge der Größe k genau dann wenn
- sein Komplementgraph eine Clique der Größe k hat

$\rightsquigarrow$ Komplementierung ist polynomiell □

Zusammenfassung und Ausblick

Polynomielle Reduktionen erlauben uns, die Schwere von Problemen zu vergleichen

SAT ist das historisch erste von vielen NP-vollständigen Problemen (Cook & Levin)

SAT $\leq_p$ **CLIQUE** $\leq_p$ **Unabhängige Menge**

Was erwartet uns als nächstes?

- Pseudopolynomielle Probleme
- NL
- Komplexität jenseits von NP