TECHNISCHE UNIVERSITÄT DRESDEN

FORMALE SYSTEME

25. Vorlesung: NP-Vollständigkeit

Markus Krötzsch

Professur für Wissensbasierte Systeme

TU Dresden, 22. Januar 2018

Rückblick

Kompexitätsklassen

Komplexitätsklassen sind Mengen von Sprachen, die man (grob) einteilt entsprechend der Ressourcen, die eine TM zur Entscheidung ihres Wortproblems benötigt.

deterministisch		nichtdeterministisch
P = PTime	polynomielle Zeit	NP = NPTime
Exp = ExpTime	exponentielle Zeit	NExp = NExpTime
L = LogSpace	logarithmischer Speicher	NL = NLogSpace
PSpace	polynomieller Speicher	NPSpace

Deterministisch vs. nichtdeterministisch

Welche Beziehungen haben diese Klassen zu anderen?

Deterministisch vs. nichtdeterministisch

Welche Beziehungen haben diese Klassen zu anderen?

- Die Beziehungen zwischen nichtdeterministischen Klassen sind analog zu denen im deterministischen Fall:

$$\mathsf{NL} \subseteq \mathsf{NP} \subseteq \mathsf{NPSpace} \subseteq \mathsf{NExp}$$

Deterministisch vs. nichtdeterministisch

Welche Beziehungen haben diese Klassen zu anderen?

- Die Beziehungen zwischen nichtdeterministischen Klassen sind analog zu denen im deterministischen Fall:

$$\mathsf{NL} \subseteq \mathsf{NP} \subseteq \mathsf{NPSpace} \subseteq \mathsf{NExp}$$

- Eine DTM kann als NTM aufgefasst werden, d.h. die nichtdeterministischen Klassen sind immer stärker:

$$\mathsf{L} \subseteq \mathsf{NL} \qquad \mathsf{P} \subseteq \mathsf{NP} \qquad \mathsf{PSpace} \subseteq \mathsf{NPSpace} \qquad \mathsf{Exp} \subseteq \mathsf{NExp}$$

Deterministisch vs. nichtdeterministisch

Welche Beziehungen haben diese Klassen zu anderen?

- Die Beziehungen zwischen nichtdeterministischen Klassen sind analog zu denen im deterministischen Fall:

$$\mathsf{NL} \subseteq \mathsf{NP} \subseteq \mathsf{NPSpace} \subseteq \mathsf{NExp}$$

- Eine DTM kann als NTM aufgefasst werden, d.h. die nichtdeterministischen Klassen sind immer stärker:

$$\mathsf{L} \subseteq \mathsf{NL} \qquad \mathsf{P} \subseteq \mathsf{NP} \qquad \mathsf{PSpace} \subseteq \mathsf{NPSpace} \qquad \mathsf{Exp} \subseteq \mathsf{NExp}$$

- Man kann NTMs mit DTMs simulieren, aber das ist oft mit exponentiellen Mehrkosten verbunden (Vorlesung 19).

Deterministisch vs. nichtdeterministisch

Welche Beziehungen haben diese Klassen zu anderen?

- Die Beziehungen zwischen nichtdeterministischen Klassen sind analog zu denen im deterministischen Fall:

$$\mathsf{NL} \subseteq \mathsf{NP} \subseteq \mathsf{NPSpace} \subseteq \mathsf{NExp}$$

- Eine DTM kann als NTM aufgefasst werden, d.h. die nichtdeterministischen Klassen sind immer stärker:

$$\mathsf{L} \subseteq \mathsf{NL} \qquad \mathsf{P} \subseteq \mathsf{NP} \qquad \mathsf{PSpace} \subseteq \mathsf{NPSpace} \qquad \mathsf{Exp} \subseteq \mathsf{NExp}$$

- Man kann NTMs mit DTMs simulieren, aber das ist oft mit exponentiellen Mehrkosten verbunden (Vorlesung 19).
- Der berühmte Satz von Savitch besagt, dass speicherbeschränkte NTMs durch DTMs mit nur quadratischen Mehrkosten simuliert werden können. Insbesondere gilt damit $\mathsf{PSpace} = \mathsf{NPSpace}$.

Zusammenfassung der wichtigsten bekannten Beziehungen:

$$\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSpace} = \mathsf{NPSpace} \subseteq \mathsf{Exp} \subseteq \mathsf{NExp}$$

Die Grenzen unseres Wissens

Wir wissen:

$$L \subseteq NL \subseteq P \subseteq NP \subseteq PSpace = NPSpace \subseteq Exp \subseteq NExp$$

- Wir wissen nicht, ob irgendeines dieser $\subseteq$ sogar $\subsetneq$ ist.
- Insbesondere wissen wir nicht, ob $P \subsetneq NP$ oder $P = NP$.
- Wir wissen nicht einmal, ob $L \subsetneq NP$ oder $L = NP$.

Die Grenzen unseres Wissens

Wir wissen:

$$L \subseteq NL \subseteq P \subseteq NP \subseteq PSpace = NPSpace \subseteq Exp \subseteq NExp$$

- Wir wissen nicht, ob irgendeines dieser $\subseteq$ sogar $\subsetneq$ ist.
- Insbesondere wissen wir nicht, ob $P \subsetneq NP$ oder $P = NP$.
- Wir wissen nicht einmal, ob $L \subsetneq NP$ oder $L = NP$.

Es wird aber vermutet, dass alle $\subseteq$ eigentlich $\subsetneq$ sind. Bekannt ist das aber nur bei exponentiell großen Ressourcenunterschieden:

Es gilt:

- $NL \subsetneq PSpace$
- $P \subsetneq Exp$
- $NP \subsetneq NExp$

NP

NP-Probleme effizient lösen?

Wir haben gesehen:

$$\textbf{SAT} \in \text{NP} \subseteq \text{PSpace} \subseteq \text{ExpTime}$$

- Dennoch haben alle unsere Algorithmen für **SAT** bisher exponentielle Laufzeit.
- **SAT** $\in$ NP führt nicht direkt zu einer besseren Lösung, da real existierende Computer nicht wie NTMs arbeiten.
- Andererseits spricht auch nichts dagegen, dass es einen schnelleren Algorithmus gibt.

Kann man **SAT** auch effizienter lösen?

$\rightsquigarrow$ Dazu sollten wir erst einmal NP etwas besser verstehen

Lösungen prüfen

Eine interessante Eigenschaft von SAT:

- Es scheint schwer, die Erfüllbarkeit einer Formel zu ermitteln (z.B. können wir exponentiell viele Wertzuweisungen durchsuchen).
- Es ist einfach, Erfüllbarkeit einer Formel zu überprüfen, wenn ein Modell gegeben ist (dazu muss nur der Wahrheitswert der Formel berechnet werden).

Lösungen prüfen

Eine interessante Eigenschaft von SAT:

- Es scheint schwer, die Erfüllbarkeit einer Formel zu ermitteln (z.B. können wir exponentiell viele Wertzuweisungen durchsuchen).
- Es ist einfach, Erfüllbarkeit einer Formel zu überprüfen, wenn ein Modell gegeben ist (dazu muss nur der Wahrheitswert der Formel berechnet werden).

Ähnlich ist es bei Sudoku:

- Es ist schwer, ein Sudoku korrekt auszufüllen.
- Es ist einfach, zu prüfen, ob ein Sudoku korrekt ausgefüllt wurde.

Lösungen prüfen

Eine interessante Eigenschaft von SAT:

- Es scheint schwer, die Erfüllbarkeit einer Formel zu ermitteln (z.B. können wir exponentiell viele Wertzuweisungen durchsuchen).
- Es ist einfach, Erfüllbarkeit einer Formel zu überprüfen, wenn ein Modell gegeben ist (dazu muss nur der Wahrheitswert der Formel berechnet werden).

Ähnlich ist es bei Sudoku:

- Es ist schwer, ein Sudoku korrekt auszufüllen.
- Es ist einfach, zu prüfen, ob ein Sudoku korrekt ausgefüllt wurde.

Gibt es noch mehr Probleme, die sich so verhalten?

- Es ist schwer, eine Lösung zu finden.
- Es ist einfach, eine gegebene Lösung zu prüfen.

TMs die Lösungen überprüfen

Wir wollen diese Idee formalisieren:

Ein polynomieller Verifikator für eine Sprache $\mathbf{L} \subseteq \Sigma^*$ ist eine polynomiell-zeitbeschränkte, deterministische TM $\mathcal{M}$, für die gilt:

- $\mathcal{M}$ akzeptiert nur Wörter der Form $w\#z$ mit:
  - $w \in \mathbf{L}$
  - $z \in \Sigma^*$ ist ein Zertifikat polynomieller Länge (d.h. für $\mathcal{M}$ gibt es ein Polynom p mit $|z| \leq p(|w|)$)
- Für jedes Wort $w \in \mathbf{L}$ gibt es ein solches Wort $w\#z \in \mathbf{L}(\mathcal{M})$.

Intuition:

- Das Zertifikat z kodiert die Lösung des Problems w, die der Verifikator lediglich nachprüft.
- Zertifikate sollten kurz sein, damit die Prüfung selbst nicht länger dauert als die Lösung des Problems.

Zertifikate werden auch Nachweis, Beweis oder Zeuge genannt

Nachweis-polynomielle Sprachen

Eine Sprache **L** ist nachweis-polynomiell wenn es für sie einen polynomiellen Verifikator gibt.

Nachweis-polynomielle Sprachen

Eine Sprache **L** ist nachweis-polynomiell wenn es für sie einen polynomiellen Verifikator gibt.

Beispiel: **SAT** ist nachweis-polynomiell. Ein möglicher polynomiellen Verifikator für **SAT** akzeptiert Wörter $F\#w$, wobei F die Kodierung einer erfüllbaren Formel ist und w eine Kodierung einer Wertzuweisung, die F erfüllt.

Nachweis-polynomielle Sprachen

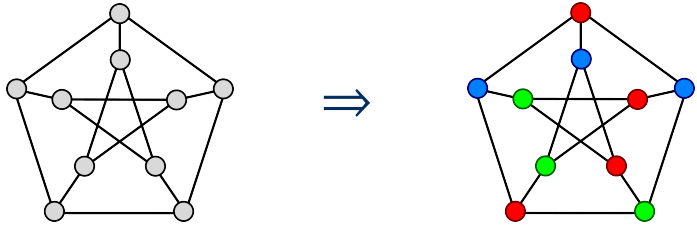
Eine Sprache **L** ist nachweis-polynomiell wenn es für sie einen polynomiellen Verifikator gibt.

Beispiel: **SAT** ist nachweis-polynomiell. Ein möglicher polynomiellen Verifikator für **SAT** akzeptiert Wörter $F\#w$, wobei F die Kodierung einer erfüllbaren Formel ist und w eine Kodierung einer Wertzuweisung, die F erfüllt.

Beispiel: Jede Sprache **L** $\in$ P ist nachweis-polynomiell. Als Verifikator verwenden wir einfach einen polynomiell-zeitbeschränkten Entscheider für **L**. Das Zertifikat kann leer sein.

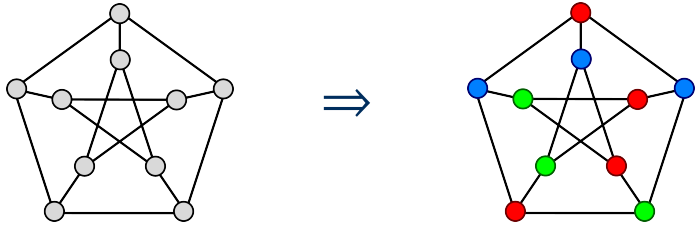
Beispiel: Drei-Farben-Problem

Das Drei-Farben-Problem besteht darin, die Knoten eines ungerichteten Graphen so mit den drei Farben rot, grün und blau zu färben, dass keine zwei benachbarten Knoten die gleiche Farbe haben.



Beispiel: Drei-Farben-Problem

Das Drei-Farben-Problem besteht darin, die Knoten eines ungerichteten Graphen so mit den drei Farben rot, grün und blau zu färben, dass keine zwei benachbarten Knoten die gleiche Farbe haben.



Das Problem kann als Wortproblem aufgefasst werden, wenn man Graphen als Wörter kodiert (z.B. als Adjazenzliste oder als zeilenweise kodierte Adjazenzmatrix).

Das Drei-Farben-Problem ist nachweis-polynomiell
(Zertifikat: Farbzuweisung).

Beispiel: Faktorisierung

Die Faktorisierung einer natürlichen Zahl n ist die Darstellung der Zahl als Produkt von natürlichen Zahlen $f_1 \cdot f_2 = n$.

Folgende Sprache ist nachweis-polynomiell:

$$\textbf{COMPOSITE} = \{\text{bin}(n) \mid \text{es gibt } f_1, f_2 > 1 \text{ mit } n = f_1 \cdot f_2\}$$

(Zertifikat: Kodierung von f_1 und f_2)

Beispiel: Faktorisierung

Die Faktorisierung einer natürlichen Zahl n ist die Darstellung der Zahl als Produkt von natürlichen Zahlen $f_1 \cdot f_2 = n$.

Folgende Sprache ist nachweis-polynomiell:

$$\textbf{COMPOSITE} = \{\text{bin}(n) \mid \text{es gibt } f_1, f_2 > 1 \text{ mit } n = f_1 \cdot f_2\}$$

(Zertifikat: Kodierung von f_1 und f_2)

Überraschender Weise ist auch folgende Sprache nachweis-polynomiell:

$$\textbf{PRIMES} = \{\text{bin}(n) \mid \text{es gibt keine } f_1, f_2 > 1 \text{ mit } n = f_1 \cdot f_2\}$$

(Zertifikat: „Primality certificate“, bekannt seit 1975)

Beispiel: Faktorisierung

Die Faktorisierung einer natürlichen Zahl n ist die Darstellung der Zahl als Produkt von natürlichen Zahlen $f_1 \cdot f_2 = n$.

Folgende Sprache ist nachweis-polynomiell:

$$\textbf{COMPOSITE} = \{\text{bin}(n) \mid \text{es gibt } f_1, f_2 > 1 \text{ mit } n = f_1 \cdot f_2\}$$

(Zertifikat: Kodierung von f_1 und f_2)

Überraschender Weise ist auch folgende Sprache nachweis-polynomiell:

$$\textbf{PRIMES} = \{\text{bin}(n) \mid \text{es gibt keine } f_1, f_2 > 1 \text{ mit } n = f_1 \cdot f_2\}$$

(Zertifikat: „Primality certificate“, bekannt seit 1975)

(Seit 2002 wissen wir: **PRIMES** $\in$ P $\rightsquigarrow$ Primzahlentest nach Agrawal, Kayal und Saxena)

NP steht für „nachweis-polynomiell“ (1)

Satz: Eine Sprache **L** ist genau dann nachweis-polynomiell wenn $\mathbf{L} \in \mathrm{NP}$.

NP steht für „nachweis-polynomiell“ (1)

Satz: Eine Sprache **L** ist genau dann nachweis-polynomiell wenn $\mathbf{L} \in \mathrm{NP}$.

Beweis: ($\Rightarrow$) Sei **L** nachweis-polynomiell mit Verifikator $\mathcal{V}$. Wir erhalten eine polynomiell-zeitbeschränkte NTM $\mathcal{M}$ für **L** wie folgt:

NP steht für „nachweis-polynomiell“ (1)

Satz: Eine Sprache **L** ist genau dann nachweis-polynomiell wenn $\mathbf{L} \in \mathrm{NP}$.

Beweis: ($\Rightarrow$) Sei **L** nachweis-polynomiell mit Verifikator $\mathcal{V}$. Wir erhalten eine polynomiell-zeitbeschränkte NTM $\mathcal{M}$ für **L** wie folgt:

- für eine Eingabe w rät $\mathcal{M}$ nichtdeterministisch ein polynomielles Zertifikat z (dabei muss $\mathcal{M}$ die Länge von z selbst beschränken)
- anschließend führt $\mathcal{M}$ die Berechnungen wie $\mathcal{V}$ aus um das geratene Zertifikat in polynomieller Zeit zu prüfen

NP steht für „nachweis-polynomiell“ (2)

Satz: Eine Sprache **L** ist genau dann nachweis-polynomiell wenn $\mathbf{L} \in \mathrm{NP}$.

Beweis: ($\Leftarrow$) Sei $\mathcal{M}$ eine polynomiell-zeitbeschränkte NTM für **L**. Wir erhalten einen polynomiellen Verifikator $\mathcal{V}$ für **L** wie folgt.

NP steht für „nachweis-polynomiell“ (2)

Satz: Eine Sprache **L** ist genau dann nachweis-polynomiell wenn $\mathbf{L} \in$ NP.

Beweis: ($\Leftarrow$) Sei $\mathcal{M}$ eine polynomiell-zeitbeschränkte NTM für **L**. Wir erhalten einen polynomiellen Verifikator $\mathcal{V}$ für **L** wie folgt.

- Für jedes Wort $w \in \mathbf{L}$ gibt es einen akzeptierenden Lauf von $\mathcal{M}$.
- In jedem Schritt dieses Laufs wählt $\mathcal{M}$ nichtdeterministisch einen Übergang aus.
- Der Lauf ist von polynomieller Länge, da $\mathcal{M}$ polynomiell-zeitbeschränkt ist.

NP steht für „nachweis-polynomiell“ (2)

Satz: Eine Sprache **L** ist genau dann nachweis-polynomiell wenn $\mathbf{L} \in$ NP.

Beweis: ($\Leftarrow$) Sei $\mathcal{M}$ eine polynomiell-zeitbeschränkte NTM für **L**. Wir erhalten einen polynomiellen Verifikator $\mathcal{V}$ für **L** wie folgt.

- Für jedes Wort $w \in \mathbf{L}$ gibt es einen akzeptierenden Lauf von $\mathcal{M}$.
- In jedem Schritt dieses Laufs wählt $\mathcal{M}$ nichtdeterministisch einen Übergang aus.
- Der Lauf ist von polynomieller Länge, da $\mathcal{M}$ polynomiell-zeitbeschränkt ist.

$\rightsquigarrow$ Die Liste der Übergänge eines akzeptierenden Laufs kann als Zertifikat dienen

$\mathcal{V}$ prüft, ob die durch z gegebenen Entscheidungen für die Eingabe w zu einem akzeptierenden Lauf von $\mathcal{M}$ führen. □

NP ist nicht symmetrisch

Es ist leicht zu sehen:

Satz: Die Klasse P ist unter Komplement abgeschlossen.

Beweis: Wenn es für **L** eine polynomiell-zeitbeschränkte TM $\mathcal{M}$ gibt, dann erhält man eine TM für $\overline{\mathbf{L}}$ indem man akzeptierende und nicht-akzeptierende Zustände von $\mathcal{M}$ vertauscht. □

NP ist nicht symmetrisch

Es ist leicht zu sehen:

Satz: Die Klasse P ist unter Komplement abgeschlossen.

Beweis: Wenn es für **L** eine polynomiell-zeitbeschränkte TM $\mathcal{M}$ gibt, dann erhält man eine TM für $\overline{\mathbf{L}}$ indem man akzeptierende und nicht-akzeptierende Zustände von $\mathcal{M}$ vertauscht. □

Für NP ist das nicht so einfach möglich. Wir haben das schon für NFAs beobachtet (eine spezielle Art von NTMs).*

Beispiele: Es scheint kein einfaches Zertifikat dafür zu geben, dass eine Formel nicht erfüllbar ist oder dass ein Graph keine Dreifärbung zulässt.

* Grund: für ein akzeptiertes Wort kann es bei einer NTM akzeptierende **und** verwerfende Läufe geben. Nach dem Austausch der Endzustände gäbe es dann aber immer noch akzeptierende Läufe.

NP ist nicht symmetrisch

Es ist leicht zu sehen:

Satz: Die Klasse P ist unter Komplement abgeschlossen.

Beweis: Wenn es für **L** eine polynomiell-zeitbeschränkte TM $\mathcal{M}$ gibt, dann erhält man eine TM für $\overline{\mathbf{L}}$ indem man akzeptierende und nicht-akzeptierende Zustände von $\mathcal{M}$ vertauscht. □

Für NP ist das nicht so einfach möglich. Wir haben das schon für NFAs beobachtet (eine spezielle Art von NTMs).*

Beispiele: Es scheint kein einfaches Zertifikat dafür zu geben, dass eine Formel nicht erfüllbar ist oder dass ein Graph keine Dreifärbung zulässt.

Die Klasse aller Sprachen **L**, für die $\overline{\mathbf{L}} \in$ NP gilt, heißt coNP.

* Grund: für ein akzeptiertes Wort kann es bei einer NTM akzeptierende **und** verwerfende Läufe geben. Nach dem Austausch der Endzustände gäbe es dann aber immer noch akzeptierende Läufe.

NP-Vollständigkeit

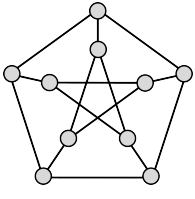
Reduktionen

Manche Probleme lassen sich auf andere zurückführen

Reduktionen

Manche Probleme lassen sich auf andere zurückführen

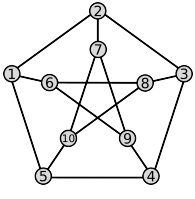
Beispiel: Drei-Farben-Problem ist auf **SAT** reduzierbar.



Reduktionen

Manche Probleme lassen sich auf andere zurückführen

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



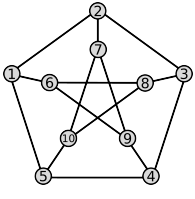
Darstellung von Farben mit Atomen:

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

Reduktionen

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)

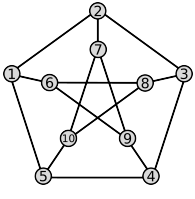
Färbe-Bedingungen für Kanten:

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

Reduktionen

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:

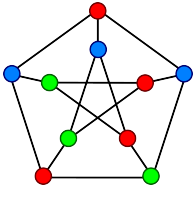
$\neg(r_1 \wedge r_2) \wedge \neg(g_1 \wedge g_2) \wedge \neg(b_1 \wedge b_2)$ (usw. für alle Kanten)

Erfüllende Wertzuweisung $\Leftrightarrow$ zulässige Färbung

Reduktionen

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

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:

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

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:

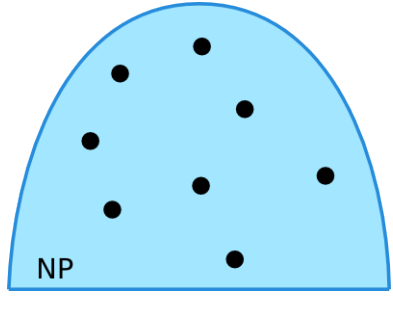
$$w \in \mathbf{L} \quad \text{genau dann wenn} \quad f(w) \in \mathbf{G}$$

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

Beispiel: Wir haben das Äquivalenzproblem endlicher Automaten in Vorlesung 10 polynomiell auf das Inklusionsproblem reduziert.

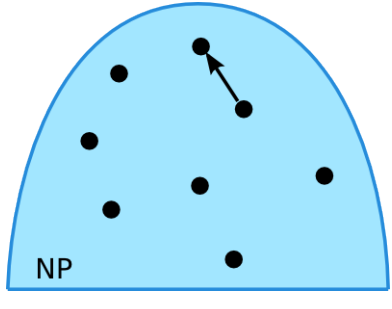
Die Struktur von NP

Idee: polynomielle Reduzierbarikeit definiert eine Art Ordnung auf Problemen



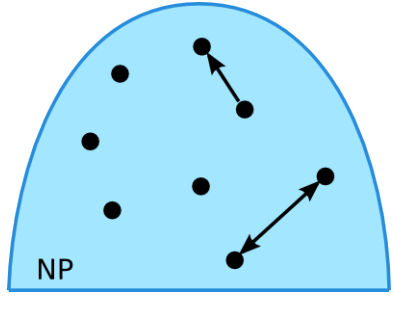
Die Struktur von NP

Idee: polynomielle Reduzierbarikeit definiert eine Art Ordnung auf Problemen



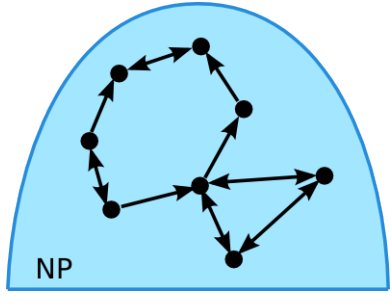
Die Struktur von NP

Idee: polynomielle Reduzierbarikeit definiert eine Art Ordnung auf Problemen



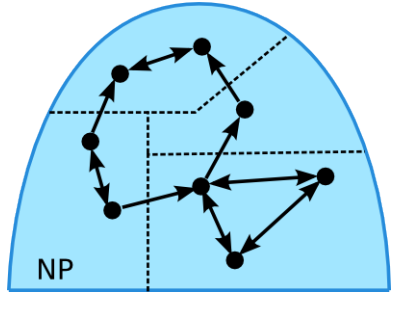
Die Struktur von NP

Idee: polynomielle Reduzierbarikeit definiert eine Art Ordnung auf Problemen



Die Struktur von NP

Idee: polynomielle Reduzierbarikeit 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.

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

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

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-Härte und NP-Vollständigkeit

Eine Sprache ist

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

NP-Härte und NP-Vollständigkeit

Eine Sprache ist

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

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

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

Beispiel: Das Halteproblem ist NP-hart aber sicher nicht in NP. Gleiches gilt für jedes unentscheidbare Problem.

Beispiele

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

Beispiel: Verallgemeinertes Sudoku (n 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-hart 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-harte Probleme?

Schwer ist, worauf sich schwere Dinge leicht zurückführen lassen:

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

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

Wie findet man NP-harte Probleme?

Schwer ist, worauf sich schwere Dinge leicht zurückführen lassen:

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

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

Technik zum Beweis von NP-Härte:

- Suche ein bekanntes NP-hartes 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-harte Probleme kennt – aber wo fängt man an?

Das erste NP-vollständige Problem

Ein Problem ist per Definition NP-vollständig:

> Das Wortpoblem für polynomiell-zeitbeschränkte nichtdeterministische Turingmaschinen besteht darin zu entscheiden, ob ein gegebenes Wort w von einer gegebenen polynomiell-zeitbeschränkten NTM akzeptiert wird.

Offenbar kann jedes Problem in NP darauf reduziert werden.

Für den Satz von Cook & Levin zeigt man die NP-Vollständigkeit von **SAT** durch Reduktion auf dieses Urproblem.

Cook/Levin: Beweisskizze (1)

Wie stellen wir TM-Berechnungen in aussagenlogisch dar?

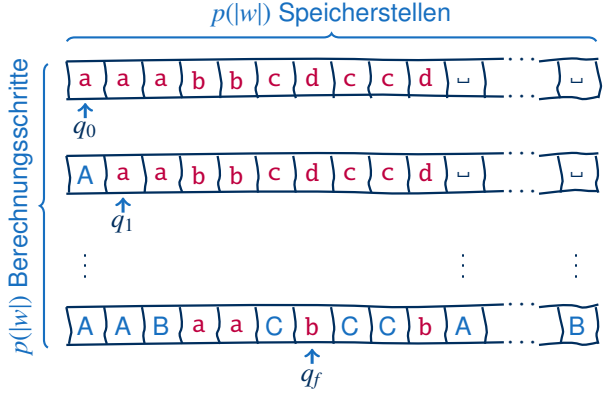
Idee:

- Eine Instanz des Wortproblems ist eine polynomiell-zeitbeschränkte NTM $\mathcal{M}$ und ein Eingabewort
- Die Zeitbeschränkung für $\mathcal{M}$ kann durch ein konkretes Polynom p angegeben werden
- Wir kennen also die maximale Zahl an Schritten $p(|w|)$ für diese 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.

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 $i \in \{1,\ldots,p(|w|)\}$).

Deterministisch vs. Nichtdeterministisch

Wir haben bereits bemerkt, dass P $\subseteq$ NP gilt, da DTMs spezielle NTMs sind.

Bis heute ist nicht bekannt, ob die Umkehrung dieser Inklusion gilt oder nicht.

- Intuitiv gefragt: „Wenn es einfach ist, eine mögliche Lösung für ein Problem zu prüfen, ist es dann auch einfach, eine zu finden?“
- Übertrieben: „Can creativity be automated?“ (Wigderson, 2006)
- Seit über 40 Jahren ungelöst
- Eines der größten offenen Probleme der Informatik und Mathematik unserer Zeit
- 1.000.000 USD Preisgeld für die Lösung („Millenium Problem“)

Status von P vs. NP

Viele Menschen glauben P $\neq$ NP

- Hauptargument: „Wenn NP = P wäre, dann müsste jemand inzwischen einen polynomiellen Algorithmus für ein NP-vollständiges Problem gefunden haben.“
- “This is, in my opinion, a very weak argument. The space of algorithms is very large and we are only at the beginning of its exploration.” (Moshe Vardi, 2002)
- Ein weiterer Grund für die verbreitete Meinung: Menschen finden es schwer, NP-Probleme zu lösen und können sich schwer vorstellen, wie man sie vereinfachen sollte – eventuell “human chauvinistic bravado” (Zeilenberger, 2006)
- Es gibt noch bessere Argumente, aber keines mehr als eine Intuition

Status von P vs. NP (2)

Verschiedene Ergebnisse sind denkbar:

Status von P vs. NP (2)

Verschiedene Ergebnisse sind denkbar:

- P = NP könnte durch einen nicht-konstruktiven Beweis gezeigt werden

Status von P vs. NP (2)

Verschiedene Ergebnisse sind denkbar:

- P = NP könnte durch einen nicht-konstruktiven Beweis gezeigt werden
- Die Antwort könnte unabhängig von den Standardaxiomen der Mathematik (ZFC) sein

Status von P vs. NP (2)

Verschiedene Ergebnisse sind denkbar:

- P = NP könnte durch einen nicht-konstruktiven Beweis gezeigt werden
- Die Antwort könnte unabhängig von den Standardaxiomen der Mathematik (ZFC) sein
- Selbst wenn P ≠ NP gilt, ist weiterhin unklar, ob NP-harte Probleme wirklich exponentielle Zeit benötigen – es gibt viele andere super-polynomielle Funktionen …

Status von P vs. NP (2)

Verschiedene Ergebnisse sind denkbar:

- P = NP könnte durch einen nicht-konstruktiven Beweis gezeigt werden
- Die Antwort könnte unabhängig von den Standardaxiomen der Mathematik (ZFC) sein
- Selbst wenn P ≠ NP gilt, ist weiterhin unklar, ob NP-harte Probleme wirklich exponentielle Zeit benötigen – es gibt viele andere super-polynomielle Funktionen ...
- Das Problem könnte für immer ungelöst bleiben

Status von P vs. NP (2)

Verschiedene Ergebnisse sind denkbar:

- P = NP könnte durch einen nicht-konstruktiven Beweis gezeigt werden
- Die Antwort könnte unabhängig von den Standardaxiomen der Mathematik (ZFC) sein
- Selbst wenn P ≠ NP gilt, ist weiterhin unklar, ob NP-harte Probleme wirklich exponentielle Zeit benötigen – es gibt viele andere super-polynomielle Funktionen ...
- Das Problem könnte für immer ungelöst bleiben

Über 100 „Beweise“ zeigen, dass P = NP wahr/falsch/beides/keines der beiden ist:
`https://www.win.tue.nl/~gwoegi/P-versus-NP.htm`

P-Härte

Man kann Härte und Vollständigkeit auch für andere Klassen definieren. Bei schwächeren Klassen muss man allerdings sicherstellen, dass die Reduktion selbst nicht schon so viele Ressourcen benötigt wie die Lösung des Problems!

Eine Sprache ist P-hart wenn jede Sprache in P mit logarithmischem Speicherbedarf darauf reduzierbar ist. Sie ist P-vollständig, wenn sie zudem in P liegt.

P-Härte

Man kann Härte und Vollständigkeit auch für andere Klassen definieren. Bei schwächeren Klassen muss man allerdings sicherstellen, dass die Reduktion selbst nicht schon so viele Ressourcen benötigt wie die Lösung des Problems!

Eine Sprache ist P-hart wenn jede Sprache in P mit logarithmischem Speicherbedarf darauf reduzierbar ist. Sie ist P-vollständig, wenn sie zudem in P liegt.

- P-vollständige Probleme sind die schwersten polynomiell lösbaren Probleme.
- Sie gelten als inhärent seriell, d.h. als nicht gut parallelisierbar (bisher nicht bewiesen).

P-Härte

Man kann Härte und Vollständigkeit auch für andere Klassen definieren. Bei schwächeren Klassen muss man allerdings sicherstellen, dass die Reduktion selbst nicht schon so viele Ressourcen benötigt wie die Lösung des Problems!

Eine Sprache ist P-hart wenn jede Sprache in P mit logarithmischem Speicherbedarf darauf reduzierbar ist. Sie ist P-vollständig, wenn sie zudem in P liegt.

- P-vollständige Probleme sind die schwersten polynomiell lösbaren Probleme.
- Sie gelten als inhärent seriell, d.h. als nicht gut parallelisierbar (bisher nicht bewiesen).

Beispiel: Die Erfüllbarkeit von aussagenlogischen Horn-Formeln (**HornSAT**) ist ein P-vollständiges Problem.

Zusammenfassung und Ausblick

NP ist die Klasse der nachweis-polynomiellen Probleme.

NP-vollständige Probleme sind die schwersten Probleme in NP: Wenn man eines davon effizient löst, dann kann alle Probleme in NP effizient lösen.

SAT ist NP-vollständig und **HornSAT** ist P-vollständig.

Offene Fragen:

- Wie genau stehen die Komplexitätsklassen in Beziehung? Insbesondere: $P \neq NP$?
- Haben Sie noch inhaltliche Fragen? ($\leadsto$ Lernräume nutzen!)
- Haben Sie sich ausreichend auf die Prüfung vorbereitet?