1 Komplexität

1.1 Reduktion revisited

Wie löst man Aufgaben vom Typ: Zeigen Sie, dass $L \ldots$

(a) entscheidbar ist?
⇒ Reduktion $L \leq L_E$ oder expliziten* Algorithmus angeben

(b) r.e. ist?
⇒ Reduktion $L \leq L_{RE}$ oder expliziten* Algorithmus angeben

(c) unentscheidbar ist?
⇒ Reduktion $L_U \leq L$

(d) nicht r.e. ist?
⇒ Reduktion $L_{NRE} \leq L$

*Eine Reduktion ist immer auch ein Algorithmus.

L entscheidbar $\not\Rightarrow$ L in der Praxis lösbar (sowohl Schwierigkeit (hält τ_0 auf w_0?) als auch Komplexität (mehrfach exponentiell/nichtelementar))

L unentscheidbar ⇒ L nicht lösbar; aber:

L unentscheidbar $\not\Rightarrow$ L in der Praxis unlösbar (zunächst keine Aussage darüber, welche realen Instanzen nicht lösbar sind; vielleicht interessieren uns nur „einfache“ Instanzen)

Meistens meint man: L in der Praxis lösbar ⇒ L polynomiell lösbar.

1.2 Polynomielle Probleme

Wie löst man Aufgaben vom Typ: Zeigen Sie, dass L polynomiell lösbar ist?
⇒ expliziten Algorithmus mit Laufzeitabschätzung angeben

Frage Können wir auch wieder eine Reduktion verwenden, z.B. $L \leq L_P$?

Ja, aber sie beweist nichts: Jedes entscheidbare Problem L lässt sich auf L_P reduzieren, indem die Reduktionsfunktion das Problem in L löst und in eine konstante Instanz von L_P übersetzt.

Idee: polynomielle Reduktion $L \leq_p L_P$

⇒ polynomieller Algorithmus für $w \overset{?}{\in} L$: frage $f(w) \overset{?}{\in} L_P$, denn $f(w)$ ist höchstens polynomiell gewachsen und Polynome sind abgeschlossen unter Komposition: $\tau_P(f(w))$ braucht Zeit in Höhe eines Polynoms von $f(w)$.

1.2.1 Probleme

(a) Betrachte folgenden Algorithmus. Gegeben eine natürliche Zahl n, berechne 10^{80}. Ist der Algorithmus polynomiell in der Länge der Eingabe?

Ja! 10^{80} ist konstant und hat nichts mit der Eingabe zu tun.

Liegt die Komplexität für TMs in $\mathcal{O}(1)$?

Nein! Man muss die Eingabe löschen, d.h. eine untere Schranke ist immer $\mathcal{O}(n)$.

Man sieht: Präzise(re) Laufzeitanalysen mit TMs sind nicht realistisch.

(b) Betrachte folgenden Algorithmus. Gegeben eine natürliche Zahl n, berechne ihre Quadratzahl n^2 folgendermaßen: Starte mit 0 und addiere n Mal n darauf. Addition ist linear in der Länge der Zahlen, also $\mathcal{O}(n)$ für jede der n Additionen. Ist der Algorithmus polynomiell in der Länge der Eingabe n?

Nein! Die Eingabe (und Ausgabe) wird logarithmisch kodiert, d.h. die Eingabe w, die n kodiert, hat nur Länge $|w| = \log_2 n$. Daher ist die Anzahl $n = 2^{|w|}$ der Additionen exponentiell in der Eingabe.

1.3 Schwere und Vollständigkeit

Fixiere eine Klasse von Problemen $\mathcal{K} \subseteq \mathcal{P}(\Sigma^*)$ (z.B. alle polynomiell lösbaren Probleme) und eine Reduktionsrelation $\preceq$ (z.B. $\leq_p$). Betrachte ein bestimmtes Problem $X \subseteq \Sigma^*$.

(a) Wenn jedes Problem aus $\mathcal{K}$ mit $\preceq$ auf X reduzierbar ist, also $\forall L \in \mathcal{K} : L \preceq X$, dann nennen wir X *$\mathcal{K}$-schwer* (bzgl. $\preceq$).

(b) Wenn X $\mathcal{K}$-schwer ist und $X \in \mathcal{K}$ gilt, dann nennen wir X *$\mathcal{K}$-vollständig* (bzgl. $\preceq$).

Der Name *schwer* ist nur gerechtfertigt, wenn die Reduktionsfunktion die Klasse $\mathcal{K}$ nicht verlässt. Im Englischen lautet die Bezeichnung *hard*, weshalb auch im Deutschen manchmal *hart* gesagt wird.
Der Name *vollständig* ist gerechtfertigt, weil X zu den schwersten Problemen in $\mathcal{K}$ gehört. Wir können jedes Problem in $\mathcal{K}$ mit dem Aufwand der Reduktion mit einem Algorithmus für X lösen (vgl. Abb. 2).

Frage Können wir für jedes polynomielle Problem mit polynomieller Reduktion in jedes andere polynomielle Problem überführen, also $L_{P_1} \leq_p L_{P_2}$?
Ja, löse einfach die Instanz für P_1 und überführe in eine konstante Instanz von P_2.

1.3.1 Beispiel

Sei P die Klasse aller polynomiell berechenbaren Funktionen. Wir definieren P-schwer/-vollständig bzgl. $\leq_{\log S}$, d.h. die Reduktionsfunktion muss in logarithmischem Platzverbrauch durchgeführt werden (wobei die Eingabe nicht gezählt wird).
Intuitiv sind P-vollständige Probleme die schwersten Probleme in P, die sich vermutlich (ein offenes Problem) nicht parallelisieren lassen und nicht mit logarithmischem Speicher auskommen.
Bekannte P-vollständige Probleme sind:

(a) Wort-/Leerheits-/Endlichkeitsproblem für kontextfreie Grammatiken

(b) lineare Programmierung/Optimierung über reellen Zahlen:

$$\max\{c^T x \mid Ax \leq b, x \geq 0\}$$

(c) Circuit Value Problem: Gegeben ein Schaltkreis und eine Eingabe, ist die Ausgabe 1? Sehr ähnlich zu SAT, aber Eingabe vorgegeben.

(d) Horn-SAT (CNF, Klauseln mit höchstens einem positiven Literal)

1.4 Berechnungskomplexität

Def. 1.1 Seien $f : \mathbb{N} \to \mathbb{N}$ eine Funktion und τ eine (nicht-)deterministische Mehrband-TM mit dem Eingabealphabet Σ.

- τ hat die *Zeitkomplexität* $f(n)$, falls für jedes Wort $w \in \Sigma^*$ der Länge n gilt: τ angesetzt auf die Eingabe w hält bei jeder möglichen Berechnung in höchstens $f(n)$ Schritten an.
- τ hat die *Platzkomplexität* $f(n)$, falls für jedes Wort $w \in \Sigma^*$ der Länge n gilt: τ angesetzt auf die Eingabe w benutzt bei jeder möglichen Berechnung auf jedem Band höchstens $f(n)$ Felder.

Für deterministische TMs gibt es natürlich genau eine Berechnung.

Def. 1.2 Sei $f : \mathbb{N} \to \mathbb{N}$.

$$\texttt{DTIME}\,(f(n)) = \{L \mid \text{es gibt eine } \textit{deterministische} \text{ Mehrband-TM der Zeitkomplexität } f(n), \text{ die } L \text{ akzeptiert } \}$$

analog `DTIME, DSPACE, NSPACE`

Klar: Zeitkomplexität $f(n)$ impliziert Platzkomplexität $f(n)$. Also gilt:

$$\begin{array}{ccccc} & & \texttt{DSPACE}(f(n)) & & \\ & \subseteq & & \subseteq & \\ \texttt{DTIME}(f(n)) & & & & \texttt{NSPACE}(f(n)) \\ & \subseteq & & \subseteq & \\ & & \texttt{NTIME}(f(n)) & & \end{array}$$

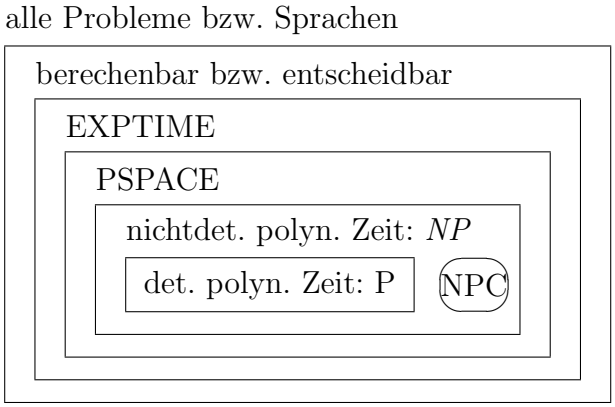


Abbildung 1: Wichtige Problemklassen der Vorlesung.

1.5 Die Klassen P und NP

Def. 2.1

$$
\begin{aligned}
P &= \bigcup_{p \text{ Polynom in } n} \texttt{DTIME}(p(n)) \\
NP &= \bigcup_{p \text{ Polynom in } n} \texttt{NTIME}(p(n)) \\
PSPACE &= \bigcup_{p \text{ Polynom in } n} \texttt{DSPACE}(p(n)) \\
NPSPACE &= \bigcup_{p \text{ Polynom in } n} \texttt{NSPACE}(p(n))
\end{aligned}
$$

Es gilt: $LOGSPACEP \subseteq NP \subseteq PSPACE \subseteq NPSPACE \subseteq EXPTIME$

Satz 2.2 (Savitch) $PSPACE = NPSPACE$

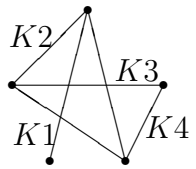
Andere Beziehungen unbekannt: \$1 Mio. für $P \stackrel{?}{=} NP$

Eine alternative Charakterisierung für Probleme in NP ist: Für das entsprechende Suchproblem lässt sich, gegeben eine Lösung, diese in polynomieller Zeit überprüfen. Eine NTM kann also nichtdeterministisch eine Lösung raten („raten und verifizieren / guess and check“).

1.5.1 Beispiele für Probleme aus der Klasse NP

(1) **Problem des Hamiltonschen Pfades**
Gegeben: Ein endlicher Graph mit n Knoten.
Frage: Gibt es einen Hamiltonschen Pfad in dem Graphen, d.h. einen Kantenzug, der jeden Knoten genau einmal trifft?

Betrachte z.B. den folgenden Graphen:



Dann ist der Kantenzug K1-K2-K3-K4 ein Hamiltonscher Pfad. Es ist leicht zu sehen, dass das Problem des Hamiltonschen Pfades in *NP* liegt: Man rate zunächst einen Pfad und prüfe dann, ob jeder Knoten genau einmal getroffen wird. Da es $n!$ Pfade im Graphen gibt, wäre dieses Verfahren nur mit exponentiellem Aufwand in deterministischer Weise anzuwenden.

(2) **Problem des Handlungsreisenden**
Gegeben: Ein endlicher vollständiger Graph mit n Knoten und Längenangaben $\in \mathbb{N}$ für jede Kante sowie eine Zahl $k \in \mathbb{N}$.
Frage: Gibt es für den Handlungsreisenden eine Rundreise der Länge $\leq k$ oder mathematischer: gibt es einen Kreis, d.h. einen geschlossenen Kantenzug, der Länge $\leq k$ der jeden Knoten mindestens einmal trifft?

Auch dieses Problem liegt in *NP*: Man rate zunächst einen Kreis und berechne dann dessen Länge.

(3) **Erfüllbarkeitsproblem für Boolesche Ausdrücke (SAT)**
Gegeben: Ein Boolescher Ausdruck B, also ein Ausdruck, der nur aus den Variablen $x_1, x_2, \ldots, x_n$ besteht, die durch Operatoren $\neg$ (*not*), $\wedge$ (*and*) und $\vee$ (*or*), sowie durch Klammern miteinander verknüpft sind.
Frage: Ist B erfüllbar, d.h. gibt es eine Belegung der Booleschen Variablen $x_1, x_2, \ldots, x_n$ in B mit 0 und 1, so dass B insgesamt den Wert 1 liefert?

Zum Beispiel ist $B = (x_1 \wedge x_2) \vee \neg x_3$ erfüllbar durch die Werte $x_1 = x_2 = 1$ oder $x_3 = 0$.

1.5.2 NP-vollständige Probleme

Def. 2.4 Eine Sprache L_0 heißt *NP-vollständig*, falls $L_0 \in NP$ gilt und $\forall L \in NP : L \leq_p L_0$.

Lemma 2.5 Sei $L_1 \leq_p L_2$. Dann gilt:

(i) Falls $L_2 \in P$ gilt, so auch $L_1 \in P$.

(ii) Falls $L_2 \in NP$ gilt, so auch $L_1 \in NP$.

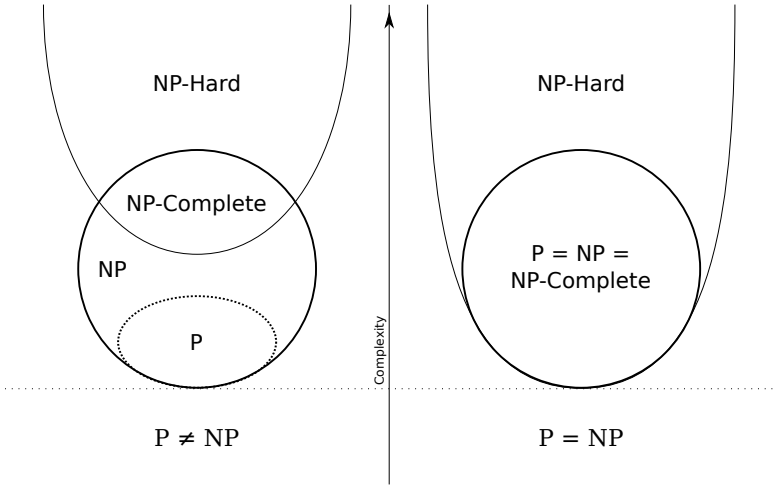


Abbildung 2: Venn-Diagramm zur Klasse der *NP*-vollständigen Probleme.

(iii) Falls L_1 *NP*-vollständig ist und $L_2 \in NP$ gilt, so ist auch L_2 *NP*-vollständig.

Beweis (iii): Sei $L \in NP$. Da L_1 *NP*-vollständig ist, gilt $L \leq_p L_1$. Ferner gilt $L_1 \leq_p L_2$. Aus der Transitivität von $\leq_p$ folgt $L \leq_p L_2$. Also ist L_2 *NP*-schwer. Zusammen mit $L_2 \in NP$ folgt die Aussage.

Frage Was ist, wenn wir ein Problem L_0 haben, das in P liegt und *NP*-schwer ist?
Dann gilt $P = NP$, denn wir können beliebiges Problem $L \in NP$ so lösen: Wende erst Reduktionsfunktion f an und dann Lösungsalgorithmus für L_0.

Korollar 2.6 (Karp) Sei L_0 eine *NP*-vollständige Sprache. Dann gilt: $L_0 \in P \Leftrightarrow P = NP$.

Gibt es *NP*-vollständige Probleme überhaupt? Ja, z.B. SAT.
Viele dieser Probleme sind hochgradig praktisch relevant. Weit über 1000 *NP*-vollständige Probleme sind bekannt. Für viele Probleme gibt es Heuristiken, Näherungsverfahren und probabilistische Verfahren, sodass man in der Praxis auch große Instanzen verarbeiten kann. Zum Beispiel kann man SAT-Instanzen mit mehreren 100k Klauseln und Millionen Variablen behandeln. Wie immer gilt: Nicht alle Instanzen müssen die worst case-Komplexität erreichen.
Vorsicht mit der Intuition: Für viele *NP*-vollständige Probleme muss man nur eine kleine Definition ändern und sie sind trivial polynomiell. Z.B. ist das Eulerkreisproblem (Kreis, der alle Kanten genau einmal besucht) linear lösbar (gdw. Graph zusammenhängend und Knotengrad gerade).

1.6 Beispiel

Wir reduzieren das Hamiltonpfadproblem (HPP) polynomiell auf das Problem des Handlungsreisenden (TSP). Dazu nehmen wir den „Umweg“ über ein drittes Problem, das Hamiltonkreisproblem (HCP). HCP ist ähnlich zu HPP, nur dass wir hier einen Kreis suchen, d.h. einen Pfad, der am Startknoten endet. Darum müssen wir hier verlangen, dass jeder Knoten außer dem Startknoten genau einmal besucht wird – der Startknoten hingegen soll genau zweimal besucht werden.
Im Folgenden bezeichnen wir mit n immer die Anzahl der Knoten in einem Graphen, also $n := |V|$ für $G = (V, E)$.
Offensichtlich ist HCP in NP: Man rät eine Abfolge von $n + 1$ Knoten und testet in linearer Zeit in n, ob es sich um einen Hamiltonkreis handelt (d.h. ob jeder Knoten außer dem Startknoten genau einmal besucht wird).

> *Hinweis*: Für die Klausur ist diese Laufzeitanalyse ausreichend. Tatsächlich müsste man aber einen möglichen Algorithmus genauer untersuchen. Eine mögliche Implementierung könnte annehmen, dass die Knoten fortlaufend nummeriert sind und man ein Array der Größe n mit Nullen anlegt. Für jeden besuchten Knoten $(n + 1)$ erhöht man den entsprechenden Eintrag um eins ($\mathcal{O}(1)$). Am Ende testet man, ob alle Einträge (bis auf einen) gleich eins sind ($\mathcal{O}(n)$). Auf Turingmaschinen ist die Analyse komplizierter, aber man darf davon ausgehen, dass es zumindest einen polynomiellen Algorithmus gibt.

(a) Zeige HPP $\leq_p$ HCP.

Konstruktion:

Eingabe: endlicher Graph $G = (V, E)$

Ausgabe: endlicher Graph $G' = (V', E')$ mit

- $V' = V \uplus \{v_0\}, v_0 \notin V$
- $E' = E \cup \{(v, v_0) \mid v \in V\}$

Die Reduktionsfunktion ist:

$$f(w) = \begin{cases} G' & \text{falls } w = G \text{ für einen Graphen } G \\ w_0 & \text{sonst } (w_0 \notin \text{HCP}) \end{cases}$$

Hinweis: Ganz korrekt wäre eigentlich $code(G)$ und $code(G')$ für eine Kodierung *code* zu schreiben.

Die Funktion f ist total, da wir eine volle Fallunterscheidung haben. Sie ist zudem polynomiell berechenbar:

(i) Die Fallunterscheidung an sich ist polynomiell: Man muss nur gültige Graphenkodierungen von ungültigen unterscheiden.

(ii) Die Konstruktion von G' ist polynomiell: Man kopiert den alten Graphen und fügt einen Knoten und $|V|$ Kanten hinzu. Das ist in Zeit in $\mathcal{O}(|V| + |E|)$ möglich.

Hinweis: Tatsächlich hängt die genaue Laufzeit auch von der gewählten Kodierung ab. Man darf aber davon ausgehen, dass sie dennoch polynomiell bleibt.

Noch zu zeigen: $G \in \text{HPP} \Leftrightarrow G' \in \text{HCP}$.

OBdA können wir von einer gültigen Graphenkodierung ausgehen, sonst ist die Reduktion klar.

"$\Rightarrow$"

$G \in \text{HPP} \Rightarrow$ es gibt einen Hamiltonpfad in G, also:

$$\exists v_1, \ldots, v_n \in V : (v_1, v_2), \ldots, (v_{n-1}, v_n) \in E \wedge \bigwedge_{i \neq j} v_i \neq v_j$$

$\Rightarrow$ es gibt diesen Hamiltonpfad auch in G', also:

$$\exists v_1, \ldots, v_n \in V' : (v_1, v_2), \ldots, (v_{n-1}, v_n) \in E' \wedge \bigwedge_{\substack{i \neq 0 \neq j \\ i \neq j}} v_i \neq v_j$$

$\Rightarrow$ wir finden einen Hamiltonkreis, indem wir über v_0 zum Startknoten zurückgehen, also:

$$\exists v_1, \ldots, v_n \in V' : (v_1, v_2), \ldots, (v_{n-1}, v_n) \in E' \wedge \bigwedge_{\substack{i \neq 0 \neq j \\ i \neq j}} v_i \neq v_j$$

$$\wedge (v_n, v_0), (v_0, v_1) \in E'$$

$\Rightarrow G' \in \text{HCP}$

"$\Leftarrow$"

$G' \in \text{HCP} \Rightarrow$ es gibt einen Hamiltonkreis in G', also:

$$\exists v_0, \ldots, v_n \in V' : (v_0, v_1), \ldots, (v_{n-1}, v_n), (v_n, v_0) \in E'$$
$$\wedge \bigwedge_{i \neq j} v_i \neq v_j$$

$\Rightarrow$ wir finden einen Hamiltonpfad in G ohne v_0, also:

$$\exists v_1, \ldots, v_n \in V : (v_1, v_2), \ldots, (v_{n-1}, v_n) \in E \wedge \bigwedge_{\substack{i \neq 0 \neq j \\ i \neq j}} v_i \neq v_j$$

$\Rightarrow G \in \text{HHP}$

(b) Zeige HCP $\leq_p$ TSP.

Konstruktion:

Eingabe: endlicher Graph $G = (V, E)$

Ausgabe: Tupel (G', k) mit

- $G' = (V', E')$ ist ein endlicher, vollständiger Graph mit $V' = V$ und $E' = \{(u, v, 1) \mid (u, v) \in E\} \cup \{(u, v, 2) \mid (u, v) \notin E\}$, d.h., E' enthält alle Kanten von E mit Gewicht 1 und alle anderen Kanten haben Gewicht 2,
- $k = |V|$.

Die Reduktionsfunktion ist analog zu oben:

$$f(w) = \begin{cases} G' & \text{falls } w = G \text{ für einen Graphen } G \\ w_0 & \text{sonst } (w_0 \notin \text{TSP}) \end{cases}$$

Die Funktion ist wieder total (gleiches Argument wie oben). Sie ist auch polynomiell berechenbar (es gibt $\mathcal{O}(|V|^2)$ mögliche Kanten in G' und für das Kopieren der Knoten benötigt man Zeit in $\mathcal{O}(|V|)$, insgesamt also $\mathcal{O}(|V|^2)$).

Noch zu zeigen: $G \in \text{HCP} \Leftrightarrow (G', k) \in \text{TSP}$.

OBdA gehen wir wieder von einer gültigen Graphenkodierung aus.

"$\Rightarrow$"

$G \in \text{HCP} \Rightarrow$ es gibt einen Hamiltonkreis in G, also:

$$\exists v_0, \ldots, v_n \in V : (v_0, v_1), \ldots, (v_{n-1}, v_n), (v_n, v_0) \in E$$
$$\wedge \bigwedge_{i \neq j} v_i \neq v_j$$

$\Rightarrow$ es gibt diese Kanten in G' mit Gewicht 1, also:

$$\exists v_0, \ldots, v_n \in V' : (v_0, v_1, 1), \ldots, (v_{n-1}, v_n, 1), (v_n, v_0, 1) \in E'$$
$$\wedge \bigwedge_{i \neq j} v_i \neq v_j$$

$\Rightarrow$ Die Gesamtsumme des Kreises ist gleich $|V|$, also $\leq k$

$\Rightarrow (G', k) \in \text{TSP}$

"$\Leftarrow$"

$(G', k) \in \text{TSP} \Rightarrow$ es gibt eine Rundreise in G' mit Kosten höchstens k;
da es $|V| = k$ Knoten gibt, muss man mindestens
k Kanten nehmen; da jede Kante mindestens
Gewicht 1 hat, muss man genau k Kanten
mit Gewicht 1 nehmen, also:

$$\exists v_0, \ldots, v_n \in V' : (v_1, v_2, 1), \ldots, (v_{n-1}, v_n, 1), (v_n, v_1, 1) \in E'$$
$$\wedge \bigwedge_{i \neq j} v_i \neq v_j$$

$\Rightarrow$ diese Kanten gab es in G, also:

$$\exists v_0, \ldots, v_n \in V : (v_1, v_2), \ldots, (v_{n-1}, v_n), (v_n, v_1) \in E$$
$$\wedge \bigwedge_{i \neq j} v_i \neq v_j$$

$\Rightarrow G \in \text{HCP}$

(c) Wir haben in zwei Schritten gezeigt: HPP $\leq_p$ HCP $\leq_p$ TSP. Weil HPP *NP*-vollständig ist, also insbesondere auch *NP*-schwer, sind HCP und TSP damit auch *NP*-schwer. Weil HCP und TSP in *NP* liegen, sind sie sogar *NP*-vollständig.