# Berechenbarkeit und Komplexität

## Optimierungsprobleme und polynomielle Reduktion

Prof. Berthold Vöcking
präsentiert von Prof. Joost-Pieter Katoen

6. Januar 2009

# Wiederholung

## Definition (Komplexitätsklasse P)

P *ist die Klasse der Probleme, für die es einen Polynomialzeitalgorithmus gibt.*

## Definition (Komplexitätsklasse NP)

NP *ist die Klasse der Entscheidungsprobleme, die durch eine NTM* $M$ *erkannt werden, deren worst case Laufzeit* $t_M(n)$ *polynomiell beschränkt ist.*

# Laufzeit einer NTM

## Definition (Laufzeit der NTM)

*Sei $M$ eine NTM. Die Laufzeit von $M$ auf einer Eingabe $x \in L(M)$ ist definiert als*

$$T_M(x) := \text{Länge des kürzesten akzeptierenden Rechenweges von } M \text{ auf } x \ .$$

*Für $x \notin L(M)$ definieren wir $T_M(x) = 0$.*

*Die worst case Laufzeit $t_M(n)$ für $M$ auf Eingaben der Länge $n \in \mathbb{N}$ ist definiert durch*

$$t_M(n) := \max\{T_M(x) \mid x \in \Sigma^n\} \ .$$

# Alternative Charakterisierung der Klasse NP

## Satz

*Eine Sprache* $L \subseteq \Sigma^*$ *ist genau dann in* NP, *wenn es einen Polynomialzeitalgorithmus* $V$ *(einen sogenannten* Verifizierer*) und ein Polynom* $p$ *mit der folgenden Eigenschaft gibt:*

$$x \in L \quad \Leftrightarrow \quad \exists y \in \{0,1\}^*, |y| \le p(|x|) : V \textit{ akzeptiert } y\#x.$$

# Die große offene Frage der Informatik lautet

$$\mathrm{P} = \mathrm{NP}?$$

Hierbei ist $P$ natürlich auf Entscheidungsprobleme eingeschränkt. Das machen wir implizit immer dann, wenn $P$ zu $NP$ in Bezug gesetzt wird.

# Offensichtlich gilt

$$\mathsf{P} \subseteq \mathsf{NP}.$$

Klar: Denn eine (deterministische) TM ist eine spezielle NTM.

**Arbeitshypothese:** $\mathsf{P} \neq \mathsf{NP}$

# Exponentielle Laufzeitschranke für Probleme aus NP

## Satz

*Für jedes Entscheidungsproblem $L \in \mathrm{NP}$ gibt es einen Algorithmus $A$, der $L$ entscheidet, und dessen worst case Laufzeit durch $2^{q(n)}$ nach oben beschränkt ist, wobei $q$ ein geeignetes Polynom ist.*

**Fazit:** $\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{EXPTIME}$

# Exponentielle Laufzeitschranke für Probleme aus NP

**Beweis:**

Um die Eingabe $x \in \{0,1\}^n$ zu entscheiden,

- starte Verifiz. $V$ mit $y\#x$ für jedes Zertifikat $y \in \{0,1\}^{p(n)}$;
- akzeptiere, falls $V$ eines der generierten Zertifikate akzeptiert.

*Laufzeitanalyse:*

- Sei $p'$ eine polynomielle Laufzeitschranke für $V$.
- Die Laufzeit unseres Algorithmus ist dann höchstens

$$\begin{aligned} 2^{p(n)} \cdot p'(p(n)+1+n) &\leq 2^{p(n)} \cdot 2^{p'(p(n)+1+n)} \\ &\leq 2^{p(n)+p'(p(n)+1+n)} = 2^{q(n)} . \end{aligned}$$

für das Polynom $q(n) = p(n) + p'(p(n)+1+n)$. $\square$

# Optimierungsprobleme und ihre Entscheidungsvariante

Beim Rucksackproblem (KP) suchen wir eine Teilmenge $K$ von $N$ gegebenen Objekten mit Gewichten $w_1, \ldots, w_N$ und Nutzenwerten $p_1, \ldots, p_N$, so dass die Objekte aus $K$ in einen Rucksack mit Gewichtsschranke $b$ passen und dabei der Nutzen maximiert wird.

**Problem (Rucksackproblem, Knapsack Problem – KP)**

**Eingabe:** $b \in \mathbb{N}$, $w_1, \ldots, w_N \in \{1, \ldots, b\}$, $p_1, \ldots, p_N \in \mathbb{N}$

**zulässige Lösungen:** $K \subseteq \{1, \ldots, N\}$, *so dass* $\sum_{i \in K} w_i \leq b$

**Zielfunktion:** *Maximiere* $\sum_{i \in K} p_i$

**Entscheidungsvariante:** $p \in \mathbb{N}$ sei gegeben. Gibt es eine zulässige Lösung mit Nutzen mindestens $p$?

# Optimierungsprobleme und ihre Entscheidungsvariante

Beim Bin Packing Problem suchen wir eine Verteilung von $N$ Objekten mit Gewichten $w_1, \ldots, w_N$ auf eine möglichst kleine Anzahl von Behältern mit Gewichtskapazität jeweils $b$.

## Problem (Bin Packing Problem – BPP)

**Eingabe:** $b \in \mathbb{N}$, $w_1, \ldots, w_N \in \{1, \ldots, b\}$

**zulässige Lösungen:** $k \in \mathbb{N}$ *und Fkt* $f : \{1, \ldots, N\} \to \{1, \ldots, k\}$,

$$\textit{so dass } \forall i \in \{1, \ldots, k\} : \sum_{j \in f^{-1}(i)} w_j \leq b$$

**Zielfunktion:** *Minimiere* $k$ *(= Anzahl Behälter)*

**Entscheidungsvariante:** $k \in \mathbb{N}$ ist gegeben. Passen die Objekte in $k$ Behälter?

# Optimierungsprobleme und ihre Entscheidungsvariante

Beim TSP ist ein vollständiger Graph aus $N$ Knoten (Orten) mit Kantengewichten (Kosten) gegeben. Gesucht ist eine Rundreise (ein *Hamiltonkreis*, eine *Tour*) mit kleinstmöglichen Kosten.

**Problem (Traveling Salesperson Problem – TSP)**

**Eingabe:** $c(i,j) \in \mathbb{N}$ *für* $i,j \in \{1,\ldots,N\}$ *mit* $c(j,i) = c(i,j)$

**zulässige Lösungen:** *Permutationen* $\pi$ *auf* $\{1,\ldots,N\}$

**Zielfunktion:** *Minimiere* $\sum_{i=1}^{N-1} c(\pi(i),\pi(i+1)) + c(\pi(N),\pi(1))$

**Entscheidungsvariante:** $b \in \mathbb{N}$ ist gegeben. Gibt es eine Tour der Länge höchstens $b$?

# Zertifikat & Verifizierer für Optimierungsprobleme

**Satz**

*Die Entscheidungsvarianten von KP, BPP und TSP sind in* NP.

**Beweis:**

Entscheidungsvarianten von Opt.problemen haben einen natürlichen Kandidaten für ein Zertifikat, nämlich zulässige Lösungen.

Es muss allerdings gezeigt werden, dass

- diese Lösungen eine polynomiell in der Eingabelänge beschränkte Kodierungslänge haben, und
- ihre Zulässigkeit durch einen Polynomialzeitalgorithmus überprüft werden kann.

# Zertifikat & Verifizierer für Optimierungsprobleme

- KP: Die Teilmenge $K \subseteq \{1, \ldots, N\}$ kann mit $N$ Bits kodiert werden. Gegeben $K$ kann die Einhaltung von Gewichts- und Nutzenwertschranke in polynomieller Zeit überprüft werden.
- BPP: Die Abbildung $f : \{1, \ldots, N\} \to \{1, \ldots, k\}$ kann mit $O(N \log k)$ Bits kodiert werden. Gegeben $f$ kann die Einhaltung der Gewichtsschranken in polynomieller Zeit überprüft werden.
- TSP: Für die Kodierung einer Permutation $\pi$ werden $O(N \log N)$ Bits benötigt. Es kann in polynomieller Zeit überprüft werden, ob die durch $\pi$ beschriebene Rundreise die vorgegebene Kostenschranke $b$ einhält.

$\square$

# Optimierungsproblem versus Entscheidungsproblem

- Mit Hilfe eines Algorithmus, der ein Optimierungsproblem löst, kann man offensichtlich auch die Entscheidungsvariante lösen. (Wie?)
- Häufig funktioniert auch der umgekehrte Weg. Wir illustrieren dies am Beispiel von KP.
- In den Übungen zeigen wir dasselbe für TSP und BPP.

**Satz**

*Wenn die Entscheidungsvariante von KP in polynomieller Zeit lösbar ist, dann auch die Optimierungsvariante.*

# Beweis: Entscheidungsvariante A $\longrightarrow$ Zwischenvariante B

**Zwischenvariante:** Gesucht ist nicht eine optimale Lösung sondern nur der optimale Zielfunktionswert.

## Polynomialzeitalgorithmus B für die Zwischenvariante

Wir verwenden eine Binärsuche mit folgenden Parametern:

- Der minimale Profit ist 0. Der maximale Profit ist $P := \sum_{i=1}^{N} p_i$.
- Wir finden den optimalen Zielfunktionswert durch Binärsuche auf dem Wertebereich $\{0, P\}$.
- Sei $A$ ein Polynomialzeitalgorithmus für die Entscheidungsvariante von $KP$.
- In jeder Iteration verwenden wir Algorithmus $A$, der uns sagt in welche Richtung wir weitersuchen müssen.

# Beweis: Entscheidungsvariante A $\rightarrow$ Zwischenvariante B

Die Anzahl der Iterationen der Binärsuche ist $\lceil \log(P+1) \rceil$.

Diese Anzahl müssen wir in Beziehung zur Eingabelänge $n$ setzen.

## Untere Schranke für die Eingabelänge:

- Die Kodierungslänge von $a \in \mathbb{N}$ ist $\kappa(a) := \lceil \log(a+1) \rceil$.
- Die Funktion $\kappa$ ist subadditiv, d.h. für alle $a, b \in \mathbb{N}$ gilt $\kappa(a+b) \leq \kappa(a) + \kappa(b)$.
- Die Eingabelänge $n$ ist somit mindestens

$$\sum_{i=1}^{N} \kappa(p_i) \;\geq\; \kappa\left(\sum_{i=1}^{N} p_i\right) \;=\; \kappa(P) \;=\; \lceil \log(P+1) \rceil \;.$$

Also reichen $n$ Aufrufe von $A$ um den optimalen Zielfunktionswert zu bestimmen.

# Beweis: Zwischenvariante B $\rightarrow$ Optimierungsvariante C

Aus einem Algorithmus $B$ für die Zwischenvariante konstruieren wir jetzt einen Algorithmus $C$ für die Optimierungsvariante.

**Algorithmus $C$**

1. $K := \{1, \ldots, N\}$;
2. $p := B(K)$;
3. **for** $i := 1$ **to** $N$ **do**
   **if** $B(K \setminus \{i\}) = p$ **then** $K := K - \{i\}$;
4. **Ausgabe** $K$.

*Laufzeit:* $N + 1$ Aufrufe von Algorithmus $B$, also polynomiell beschränkt, falls die Laufzeit von $B$ polynomiell beschränkt ist.

# Polynomielle Reduktion

**Definition (Polynomielle Reduktion)**

*$L_1$ und $L_2$ seien zwei Sprachen über $\Sigma_1$ bzw. $\Sigma_2$. $L_1$ ist* polynomiell reduzierbar *auf $L_2$, wenn es eine Reduktion von $L_1$ nach $L_2$ gibt, die in polynomieller Zeit berechenbar ist. Wir schreiben $L_1 \leq_p L_2$.*

D.h. $L_1 \leq_p L_2$, genau dann, wenn es eine Funktion $f : \Sigma_1^* \to \Sigma_2^*$ mit folgenden Eigenschaften gibt:

- $f$ ist in polynomieller Zeit berechenbar
- $\forall x \in \Sigma_1^* : x \in L_1 \;\Leftrightarrow\; f(x) \in L_2$

# Polynomielle Reduktion

**Lemma**

$L_1 \leq_p L_2,\ L_2 \in P \Rightarrow L_1 \in P.$

**Beweis:** Die Reduktion $f$ habe die polyn. Laufzeitschranke $p(\cdot)$. Sei $B$ ein Algorithmus für $L_2$ mit polyn. Laufzeitschranke $q(\cdot)$.

**Algorithmus $A$ für $L_1$:**

1. Berechne $f(x)$.
2. Starte Algorithmus $B$ für $L_2$ auf $f(x)$.

Schritt 1 hat Laufzeit höchstens $p(|x|)$. Schritt 2 hat Laufzeit höchstens $q(|f(x)|) \ \leq \ q(p(|x|) + |x|)$. □

# Beispiel einer polyn. Reduktion: COLORING $\leq_p$ SAT

Die eigentliche Stärke des Reduktionsprinzips ist es, dass man Probleme unterschiedlichster Art aufeinander reduzieren kann.

## Problem (Knotenfärbung – COLORING)

*Eingabe: Graph* $G = (V, E)$, *Zahl* $k \in \{1, \ldots, |V|\}$
*Frage: Gibt es eine Färbung* $c : V \to \{1, \ldots, k\}$ *der Knoten von* $G$ *mit* $k$ *Farben, so dass benachbarte Knoten verschiedene Farben haben, d.h.* $\forall \{u, v\} \in E : c(u) \neq c(v)$.

## Problem (Erfüllbarkeitsproblem / Satisfiability — SAT)

*Eingabe: Aussagenlogische Formel* $\phi$ *in KNF*
*Frage: Gibt es eine erfüllende Belegung für* $\phi$*?*

# Beispiel einer polyn. Reduktion: COLORING $\leq_p$ SAT

**Satz**

*COLORING $\leq_p$ SAT.*

**Beweis:**
Wir beschreiben eine polynomiell berechenbare Funktion $f$, die eine Eingabe $(G, k)$ für das COLORING-Problem auf eine Formel $\phi$ für das SAT-Problem abbildet, mit der Eigenschaft

$$G \text{ hat eine } k\text{-Färbung} \iff \phi \text{ ist erfüllbar} .$$

# Beispiel einer polyn. Reduktion: COLORING $\leq_p$ SAT

*Beschreibung der Funktion f :*

Die Formel $\phi$ hat für jede Knoten-Farb-Kombination $(v, i)$, $v \in V$, $i \in \{1, \ldots, k\}$, eine Variable $x_v^i$. Die Formel für $(G, k)$ lautet

$$\phi = \bigwedge_{v \in V} \underbrace{(x_v^1 \vee x_v^2 \vee \ldots \vee x_v^k)}_{\text{Knotenbedingung}} \wedge \bigwedge_{\{u,v\} \in E} \bigwedge_{i \in \{1,\ldots,k\}} \underbrace{(\bar{x}_u^i \vee \bar{x}_v^i)}_{\text{Kantenbedingung}} .$$

Anzahl der Literale $= O(k \cdot |V| + k \cdot |E|) = O(|V|^3)$.

Die Länge der Formel ist somit polynomiell beschränkt und die Formel kann in polynomieller Zeit konstruiert werden.

Aber ist die Konstruktion auch korrekt?

# Beispiel einer polyn. Reduktion: COLORING $\leq_p$ SAT

**Korrektheit:**

**zz: $G$ hat eine $k$ Färbung $\Rightarrow \phi$ ist erfüllbar**

- Sei $c$ eine $k$-Färbung für $G$.
- Für jeden Knoten $v$ mit $c(v) = i$ setzen wir $x_v^i = 1$ und alle anderen Variablen auf 0.
- Knotenbedingung: Offensichtlich erfüllt.
- Kantenbedingung: Für jede Farbe $i$ und jede Kante $\{u, v\}$ gilt $\bar{x}_u^i \vee \bar{x}_v^i$, denn sonst hätten $u$ und $v$ beide die Farbe $i$.
- Damit erfüllt diese Belegung die Formel $\phi$.

# Beispiel einer polyn. Reduktion: COLORING $\leq_p$ SAT

**zz: $\phi$ ist erfüllbar $\Rightarrow$ $G$ hat eine $k$ Färbung**

- Fixiere eine beliebige erfüllende Belegung für $\phi$.
- Wegen der Knotenbedingung gibt es für jeden Knoten $v$ mindestens eine Farbe mit $x_v^i = 1$.
- Für jeden Knoten wähle eine beliebige derartige Farbe aus.
- Sei $\{u, v\} \in E$. Wir behaupten $c(u) \neq c(v)$.
- Zum Widerspruch nehmen wir an, $c(u) = c(v) = i$. Dann wäre $x_u^i = x_v^i = 1$ und die Kantenbedingung $\bar{x}_u^i \vee \bar{x}_v^i$ wäre verletzt.

$\square$

# Beispiel einer polyn. Reduktion: COLORING $\leq_p$ SAT

COLORING $\leq_p$ SAT impliziert die folgenden beiden Aussagen.

**Korollar**

*Wenn SAT einen Polynomialzeitalgorithmus hat, so hat auch COLORING einen Polynomialzeitalgorithmus.*

**Korollar**

*Wenn COLORING keinen Polynomialzeitalgorithmus hat, so hat auch SAT keinen Polynomialzeitalgorithmus.*