# Grundlagen Theoretischer Informatik 3

## SoSe 2012 in Trier

Henning Fernau
Universität Trier
fernau@uni-trier.de

## ‘Harte’ Probleme

Sind “harte Probleme” stets NP-hart?
Vermutlich nein. . .
Klassisches Beispiel: Das *Graphenisomorphieproblem*.
Gegeben: Zwei Graphen $G_1, G_2$.
Frage: Sind $G_1 = (V_1, E_1)$ und $G_2 = (V_2, E_2)$ isomorph, d.h.:
Gibt es eine Bijektion $\phi$ zwischen der Knotenmenge $V_1$ und der Knotenmenge $V_2$, sodass $\{u, v\} \in E_1$ gdw. $\{\phi(u), \phi(v)\} \in E_2$?

Das Problem liegt in NP (Warum?), aber die NP-Härte ist unbekannt, und es gibt Hinweise darauf, dass dies nicht der Fall ist (s.u.)

## Grundform einer algorithmischen Aufgabenstellung:

- i.d.R. nicht als Entscheidungsproblem,
- sondern sondern als *funktionales Problem* (oder Suchproblem):
  Gegeben $x$, bestimme eine „Lösung" $y$, falls eine solche existiert.

$y$ ist dabei i.d.R. Resultat der "Guess-Phase" bei Guess-and-Check
und kann schnell überprüft werden
(Bsp: Variaben-Belegung, Knotenüberdeckungseigenschaft, Färbung,...)

Wichtig: Bei nichtdeterministischen Algorithmen ist es nicht gleichgültig, ob man
—die Existenz einer Lösung zeigen will oder
—zeigen will, dass gar keine Lösung existiert.

Zu einem Problem $\mathrm{K}$ aus einer Grundmenge $\mathrm{M}$ sei co-$\mathrm{K} := \mathrm{M} \setminus \mathrm{K}$.

Die Problemklasse co-NP besteht aus allen Problemen $\mathrm{K}$, für die co-$\mathrm{K}$ in NP liegt.
Es ist noch unbekannt, ob co-NP=NP gilt (vermutlich nicht...)

- Unbekannt ist z.B. ob co-SAT in NP liegt
  (Wie soll man durch Raten feststellen, dass z.B. keine erfüllende Belegung existiert?)
- Bei deterministischen Entscheidungsalgorithmen kann man das Resultat einfach negieren, d.h. co-P=P. (vgl. Komplement-Operation bei DEAs)

Weitere Problem-Form: **Optimierungsprobleme**
Zu Eingabe bestimme Lösung, die eine gegebene Kostenfunktion maximiert (oder minimiert)
Bei NP-vollständigen Problemen gilt i.Allg.:

- "entsprechendes" funktionales Problem oder Optimierungsproblem auf Entscheidungsproblem polynomial "reduzierbar";
- gibt es polynomialen Algorithmus für das Entscheidungsproblem, so auch für das funktionale oder Optimierungsproblem.
  An der Tafel: Selbstreduktionseigenschaft am Beispiel SAT.

Entscheidungsprobleme für Theorie ausreichend
(aber nicht für Praxis: hier genauere Betrachtung notwendig...)

*Optimierungsprobleme* werden spezifiziert durch:

- die Menge der zulässigen Eingaben,
- die zulässigen Lösungen $S(x)$ für zulässige Eingaben $x$,
- eine Bewertungsfunktion $v$, die Lösungen $y$ mit $v(y)$ bewertet.

Jede der drei Komponenten sollte polynomiale Komplexität haben:

- Zulässigkeit sollte mit polynomialer Komplexität überprüfbar sein!
- $y \in S(x)$ sollte in polynomialer Komplexität entscheidbar sein.
- $v$ sei in polynomialer Komplexität berechenbar.

Optimierungsprobleme sind gegliedert in

- Minimierungsprobleme:
  suche Lösung $y \in S(x)$ mit $v(y) = \min\{v(s) \mid s \in S(x)\}$.
- Maximierungsprobleme:
  suche analog Lösung mit maximalem $v$-Wert

**Beispiel**: **Traveling Salesman Problem** (TSP) als Minimierungsproblem:

- zulässige Eingaben: $n \times n$ Entfernungsmatrix $M = (m_{i,j})$
  (Abstände/Kosten zuje zwei Städten $i$ und $j$, $(i, j \in \{1, \ldots, n\})$
- Erlaubt: Einträge mit Wert $\infty$
  (keine direkte Straßenverbindung zwischen Ort $i$ und Ort $j$)
- zulässige Lösungen sind Permutationen $\pi$ auf $\{1, 2, \ldots, n\}$
  (repräsentieren Rundreise durch alle Städte)
  zulässige Rundreise müssen ohne $\infty$-Einträge auskommen
- Bewertung $v$ einer zulässigen Permutation $\pi$ für $M$ ist definiert als

$$v(\pi) = \sum_{k=1}^{n-1} m_{\pi(k),\pi(k+1)} + m_{\pi(n),\pi(1)}$$

- O.B.d.A. betrachte nur Permutationen $\pi$ mit $\pi(1) = 1$.

Ziel: **Suche $\pi$ mit minimalen Gesamtkosten** $v(\pi)$

**Noch ein Beispiel**: **MaxSAT** als Maximierungsproblem:

- Gegeben: eine *Menge* $F$ von Klauseln,
  gesucht: eine Belegung, die möglichst viele Klauseln erfüllt.
- Zulässige Eingaben: syntaktisch korrekte Klauselmengen.
- Jede Belegung $\Phi$ der Variablen ist zulässige Lösung!
- Bewertung $\nu(\Phi) =$ Anzahl der durch $\Phi$ erfüllten Klauseln

Beispiel:

$$F = \{(x_1 \vee x_2 \vee x_3), (\neg x_2 \vee x_3), \neg x_1, \neg x_2, \neg x_3\}$$

mit (nicht-eindeutiger aber) optimaler Lösung

$$\Phi : x_1 \mapsto 1, x_2 \mapsto 0, x_3 \mapsto 0$$

und $\nu(\Phi) = 4$.

Umwandlung vom Optimierungs- zum Entscheidungsproblem:

- natürliche Zahl $k$ als weiterer Bestandteil der Eingabe,
  d.h. Eingaben sind

$$\{(x, k) \mid x \text{ ist zulässig}, k \in \mathbb{N}\}$$

- gesucht: Gibt es Lösung der Güte $k$ oder besser?

Bei Minimierungsproblemen lautet das Entscheidungsproblem also:

- Ist zu $(x, k)$ die Menge $\{y \in S(x) \mid v(y) \leq k\}$ nichtleer?

Das liefert auch die *Standard-Parameterisierung* für FPT-Algorithmen.

Weitere Problem-Form: **Zählprobleme**

- Zu Eingabe bestimme genaue Anzahl der Lösungen

Für NP-vollständige Probleme sind die Zählprobleme wohl schwieriger als die Entscheidungsprobleme!

Andererseits beim Graphenisomorphieproblem:

- Aus fiktiven polynomialen Entscheidungsalgorithmus würde polynomialer Algorithmus für das Zählproblem (ohne Beweis...)

- daher Vermutung: Graphenisomorphieproblem ist evtl. nicht NP-vollständig...

Unsere Aufgabe: Algorithmen für NP-vollständige Probleme

- insbesondere für Traveling Salesman Problem und Erfüllbarkeitsproblem als Beispiele

Algorithmen für exakte Lösungen (für funktionales oder Optimierungsproblem) haben (derzeit) exponentielle Laufzeit.

Daher folgende Möglichkeiten:

- Suche nach exakten Algorithmen, die zumindest besser sind als naive Suche
- Suche nach polynomialen Algorithmen, die akzeptable Näherungslösungen berechnen
- Benutze dabei insbesondere Zufallszahlen als Hilfsmittel

**Betrachte Optimierungsprobleme!** Da vermutlich $P \neq NP$:

- Dynamisches Programmieren, Backtracking, Branch-and-Bound sind i.d.R. nicht polynomial!
- Als Alternative: statt 'optimaler' nur 'gute' Lösungen verlangen!
- Suche Lösung $y \in S(x)$, sodass $v(y)$ nicht 'allzu weit' von optimalen $v^*(x) = \mathrm{opt}\{v(z) \mid z \in S(x)\}$ weg ist

Anmerkung: das Finden einer zulässigen Lösung $y \in S(x)$ sollte nicht bereits $NP$-vollständig sein.

Beispiele:

- Finden zulässiger Lösungen bei MaxSAT ist trivial.
- Finden zulässiger Lösungen bei TSP ist trivial,
  wenn $\infty$ in Lösungen zulässig ist
- Finden zulässiger Lösungen bei TSP ist NP-vollständig,
  wenn $\infty$ in Lösungen nicht zuässig ist (da damit die Existenz von Hamilton-kreisen getestet werden kann!)

Für ein Optimierungsproblem $P$ mit zulässiger Eingabe $x$ und Lösungsmenge $S(x)$ ist die *(Approximations-)Güte* $g(x, y)$ einer Lösung $y \in S(x)$ definiert durch $g(x, y) := \frac{v(y)}{v^*(x)}$, wobei $v^*(x) :=$ Optimum aus $\{v(z) \mid z \in S(x)\}$.

Anmerkungen:

- Bei Minimierungsproblemen: $v^*(x) := \min\{v(z) \mid z \in S(x)\}$
- Bei Maximierungsproblemen: $v^*(x) := \max\{v(z) \mid z \in S(x)\}$
- Stets $g(x, y) \geq 1$ bei Min. und $g(x, y) \leq 1$ bei Max., gute Approximationen haben $g(x, y) \approx 1$
- in Literatur auch Definition über *Performanz*

$$r(x, y) := \min\left\{\frac{v(y)}{v^*(x)}, \frac{v^*(x)}{v(y)}\right\}$$

Ein Optimierungsproblem liegt in ***APX***, wenn es **eine Zahl** $\delta \in (0, 1)$ und einen polynomialen Algorithmus gibt, der bei jeder zulässigen Eingabe $x$ eine Lösung $y \in S(x)$ liefert mit einer zugehörigen Performanz $r(x, y) \geq 1 - \delta$.

Alternativ bei Max.: Zahl $r \leq 1$ mit $g(x, y) \geq r$; bei Min.: $r \geq 1$ mit $g(x, y) \leq r$.

Die Abkürzung **APX** deutet an, dass das Optimierungsproblem — bis zu einem gewissen Grad — approximierbar ist.

$\delta$ darf beliebig nahe an 1 sein, d.h. schlechte Performanz erlaubt.

Stärkere Einschränkung / bessere Approximationen:

Ein Optimierungsproblem liegt in ***PTAS***, wenn es **für jede Zahl** $\delta \in (0, 1)$ einen polynomialen Algorithmus gibt, der bei jeder zulässigen Eingabe $x$ eine Lösung $y \in S(x)$ liefert mit einer zugehörigen Performanz $r(x, y) \geq 1 - \delta$.

Die Abkürzung **PTAS** steht für *polynomial time approximation scheme*.

Hier ist jede beliebig gute Performanzschranke $\delta$ erreichbar.

Ein Optimierungsproblem liegt in ***FPTAS***, wenn es einen Algorithmus gibt, der bei jedem zulässigen $x$ und $k \in \mathbb{N}$ als Eingabe eine Lösung $y \in S(x)$ liefert mit einer zugehörigen Performanz $r(x, y) \geq 1 - 1/k$. Die Komplexität des Algorithmus muss polynomial in $x$ und $k$ sein.

Die Abkürzung **FPTAS** steht für *fully polynomial-time approximation scheme*.

Bei FPTAS ist selbst die Abhängigkeit von der Performanzschranke noch polynomiell!

Damit ergibt sich folgende Inklusionskette

**Menge exakt lösbarer Optimierungsprobleme aus** $\mathrm{P}$
$\subseteq$ **FPTAS** $\subseteq$ **PTAS** $\subseteq$ **APX** $\subseteq$
**Menge aller** $\mathrm{NP}$**-Optimierungsprobleme**

Falls $\mathrm{P} \neq \mathrm{NP}$, sind alle Inklusionen echt.

Typische Beispiele einiger Klassen sind:

- **Menge exakt lösbarer Optimierungsprobleme aus** $\mathrm{P}$:
  Maximale Flüsse in Graphen ($\rightsquigarrow$ Vorlesung Netzwerkalgorithmen)
- **FPTAS**: $0/1$-Rucksackproblem
  ($\rightsquigarrow$ über dynamisches Programmieren, s.u.)
- **APX**: VertexCover oder MaxSAT
  - probabilistisches und deterministische Approximationsverfahren
  - $\mathrm{P} \neq \mathrm{NP} \Rightarrow$ MaxSAT $\notin$ PTAS: 'PCP-Theorem' (probabilistically checkable proofs)
- **Menge aller** $\mathrm{NP}$**-Optimierungsprobleme**: TSP (vermutlich nicht in **APX**)

## Zur Approximierbarkeit von TSP

Annahme: Es gibt polynomialen Algorithmus für TSP und $\delta \in (0,1)$ mit Performanz von mindestens $1-\delta$.

- Sei $G$ beliebiger Graph mit $n$ Knoten, definiere dazu TSP $x$ durch

$$m_{i,j} = \begin{cases} 1, \text{ falls } (i,j) \text{ Kante in } G \\ 1 + \lceil \frac{n}{1-\delta} \rceil, \text{ sonst} \end{cases}$$

- TSP-Approximation liefert zu $x$ Lösung $y$ mit Performanz $r(x,y) = \frac{\nu^*(x)}{\nu(y)} \geq 1-\delta$
- Existiert Hamilton-Kreis in $G$, dann $\nu^*(x) = n$, also $\nu(y) < \frac{n}{1-\delta}$, d.h. $y$ nutzt nur Kanten aus $G$ (d.h. findet Hamilton-Kreis in $G$)
- Über $\nu(y)$ kann also entschieden werden, ob in $G$ ein Hamilton-Kreis existiert ...

also

$$\textbf{TSP} \in \textbf{APX} \Longrightarrow P = NP$$

d.h. **TSP** liegt vermutlich nicht in **APX**!

## 0/1-Rucksack liegt in FPTAS

Löse 0/1-Rucksack über 'minimales Gewicht für gesuchten Gewinn':

- Gegeben: $n \in \mathbb{N}$, $G \in \mathbb{N}$, Vektoren $(v_1, ..., v_n), (g_1, ..., g_n) \in \mathbb{N}^n$
- definiere $g(i, w) :=$ minimales Gewicht für Teilmengen von Objekten aus $\{1, ..., i\}$, die zum Gewinn $\geq w$ führen
- formal: berechne

$$g(i, w) = \min\{\sum_{j \in I} g_j \mid I \subseteq \{1, ..., i\} \wedge \sum_{j \in I} v_j \geq w\}$$

- Gesucht: größter Gewinn $w$ mit $g(n, w) \leq G$
- rekursive Formulierung von $g(i, w)$:

$$g(i, w) = \begin{cases} 0 & w \leq 0 \\ \infty & w > 0 \wedge i = 0 \\ \min \left\{ \begin{array}{l} g(i-1, w), \\ g(i-1, w-v_i) + g_i \end{array} \right\} & \text{sonst} \end{cases}$$

Dynamisches Programm dazu (für exakte Lösung):

- statt Funktion $g(i, w)$ verwende Tabelle $g[i, w]$ mit $0 \leq i < n$
  aber nur für $w$ mit $g[i, w] \leq G$

```
FOR i:=0 TO n DO
    w:=0;
    REPEAT
      w = w + 1;
      berechne g[i,w] nach Formel
    UNTIL g[i,w]> G
ENDFOR
Ausgabe: w-1
```

- Ausgabe also: größtes $W$ mit $g[n, W] \leq G$
- $I$ sei zu $W$ gehörige optimale Teilmenge von $\{1, ..., n\}$
- zur Tabellengröße: Zugriffe nur auf $g[i, w]$ mit $w \leq W + 1$
- Aufwand: $\mathcal{O}(n \cdot W)$ (Funktionsauswertungen ignorierbar!)

## 0/1-Rucksack in FPTAS (Forts.)

jetzt approximative Lösung durch modifizierte Probleminstanz:

- Wähle $m$ geeignet (s.u.), setze $v'_i := \lfloor v_i/m \rfloor$
- Löse $(0,1)$-Rucksack für $(v'_1, ..., v'_n)$
  (aber mit unveränderten $(g_1, ..., g_n)$ und $G$)
- $J$ sei die dabei gefundene optimale Menge, mit $V' := \sum_{i \in J} v'_i$
- Setze $V := \sum_{i \in J} v_i$ (d.h. neue Menge, aber alte Kosten!)

**0/1-Rucksack in FPTAS (Forts.)** jetzt approximative Lösung:

Vergleiche nun $J$ mit $I$ sowie die drei Werte $W$, $V'$ und $V$:

- Zwischen $J$ und $I$ lässt sich kein direkter Zusammenhang finden, Objekte der Lösungen $I$ und $J$ können komplett verschieden sein!
- $J$ erfüllt $\sum_{i \in J} g_i \leq G$, d.h. $J$ ist auch (i.d.R. nicht-optimale) Lösung für das Ursprungsproblem, d.h $V \leq W$, zudem

$$V' = \sum_{i \in J} v'_i \leq \sum_{i \in J} v_i/m = V/m \leq W/m$$

- damit sofort: Aufwand für die Lösung des modifizierten Problems mit $(v'_1, ..., v'_n)$ ist $\mathcal{O}(n \cdot V') = \mathcal{O}(n \cdot W/m)$

Betrachte folgendes Beispielproblem:

$$n = 3, (v_1, v_2, v_3) = (3, 5, 6), (g_1, g_2, g_3) = (1, 2, 3), G = 5$$

Resultate beim dynamischen Programm:

| $W:$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | ... |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $i = 1$ | 1 | 1 | 1 | $\infty$ | | | | | | | | | |
| $i = 2$ | 1 | 1 | 1 | 2 | 2 | 3 | 3 | 3 | $\infty$ | | | | |
| $i = 3$ | 1 | 1 | 1 | 2 | 2 | 3 | 3 | 3 | 4 | 5 | 5 | 6 | |

Lösung: $W = 11$ bei der Menge $I = \{2, 3\}$

Teste Approximation mit $m = 3$, also $(v'_1, v'_2, v'_3) = (1, 1, 2)$

| $V':$ | 1 | 2 | 3 | 4 | ... |
|---|---|---|---|---|---|
| $i = 1$ | 1 | $\infty$ | | | |
| $i = 2$ | 1 | 3 | $\infty$ | | |
| $i = 3$ | 1 | 3 | 4 | 6 | |

Lösung: $V' = 3$ bei $J = \{1, 3\}$,
d.h. Approximation an $W$ ist $V = 9$.

Zur Performanz $V/W$ der Lösung $J$ (für das Originalproblem):

- Da $J$ optimal für $(v'_1, ..., v'_n)$ gilt (1) $\sum_{i \in I} v'_i \leq V'$.
- Setze $v_{max} := \max\{v_1, ..., v_n\}$, damit sofort (2) $W \leq n \cdot v_{max}$
- o.B.d.A. stets $g_i \leq G$ (sonst Objekt $i$ uninteressant...),
  damit andererseits (3) $v_{max} \leq W$
- Mit Setzung $v'_i := \lfloor v_i/m \rfloor$ ist $v'_i \geq v_i/m - 1$, also

$$\begin{aligned} V &= \sum_{i \in J} v_i \geq \sum_{i \in J} v'_i \cdot m = V' \cdot m \overset{(1)}{\geq} \sum_{i \in I} v'_i \cdot m \\ &\geq \sum_{i \in I} (v_i/m - 1) \cdot m \geq \sum_{i \in I} (v_i - m) \geq W - n \cdot m \end{aligned}$$

- für die Performanz ergibt sich:

$$\frac{V}{W} \geq \frac{W - nm}{W} \geq 1 - \frac{n \cdot m}{W} \overset{(3)}{\geq} 1 - \frac{n \cdot m}{v_{max}}$$

- Für Ziel-Performanz $1 - 1/k$ wähle $m$ so, dass $\frac{n \cdot m}{\nu_{max}} \leq 1/k$, z.B.

$$m = \frac{\nu_{max}}{k \cdot n}$$

mit Aufwand

$$\mathcal{O}(n \cdot W/m) = \mathcal{O}(k \cdot n^2 \cdot W/\nu_{max}) \overset{(2)}{=} \mathcal{O}(k \cdot n^3)$$

- Die Komplexität ist also polynomial in $n$ und $k$.

## Anmerkungen:

- Ein Problem heißt *stark NP-vollständig*, wenn es auch noch NP-vollständig ist, wenn alle involvierten Zahlen in unärer Schreibweise (statt binär) eingegeben werden.
- Bei unärer Notation wird die Eingabe wesentlich länger,
  ein polynomialer Algorithmus darf länger laufen...
- Gibt man bei $0/1$-Rucksack die Schranke $G$ unär an,
  so gibt es eine polynomiale Lösung.
  (Bsp: bekanntes dynamisches Programm mit Laufzeit $\mathcal{O}(n \cdot G)$ )
  $0/1$-Rucksack ist damit nur 'schwach NP-vollständig'!
- Stark NP-vollständig sind z.B. TSP, 3-KNF-SAT, Färbbarkeit,...
- Es gilt sogar:

  **Falls P$\neq$ NP, so liegt kein stark NP-vollständiges Problem in FPTAS!**

## TSP-Varianten

Wichtige Modifikationen des TSP sind:

- Metrisches TSP: Kosten $m_{i,j}$ der Kanten sind symmetrisch und erfüllen die Dreiecksungleichung $m_{i,j} \leq m_{i,k} + m_{k,j}$
- Euklidisches TSP: Punkte liegen auf der Ebene,
  Entfernungen der Punkte entsprechen dem Euklidischen Abstand

Hier gilt:

- Metrisches TSP liegt in **APX**, wobei die Performanz $2/3$ beträgt (d.h. die gefundene Route ist 50% maximal länger als die optimale Route) (Christofides, 1976)
- Metrisches TSP liegt nicht in **PTAS** (wenn P$\neq$NP) (Arora, 1992)
- Euklidisches TSP ist NP-vollständig
- Euklidisches TSP liegt in **PTAS** (Arora, 1996)