Asymptotische Notation

$$O(f(n)) := \{\, g(n)\,;\ \exists c, n_0 > 0\ \forall n \geq n_0\ 0 \leq g(n) \leq c \cdot f(n) \,\}$$

$$\Omega(f(n)) := \{\, g(n)\,;\ \exists c, n_0 > 0\ \forall n \geq n_0\ 0 \leq c \cdot f(n) \leq g(n) \,\}$$

Es gilt: $g(n) \in O(f(n)) \Leftrightarrow f(n) \in \Omega(g(n))$.

$$\Theta(f(n)) := O(f(n)) \cap \Omega(f(n))$$

$$o(f(n)) := \{\, g(n)\,;\ \forall c > 0\ \exists n_0 > 0\ \forall n \geq n_0\ \ 0 \leq g(n) \leq c \cdot f(n) \,\}$$

$$\omega(f(n)) := \{\, g(n)\,;\ \forall c > 0\ \exists n_0 > 0\ \forall n \geq n_0\ \ 0 \leq c \cdot f(n) \leq g(n) \,\}$$

Für $f(n), g(n) > 0$ gilt:

$$f(n) \in o(g(n)) \quad \Leftrightarrow \quad \lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$$

Problemtypen

Erreichbarkeit: Instanz: Graph G, Ecken s, t.
Lösung: Weg von s nach t.

Entscheidungsproblem: Gibt es einen Weg von s nach t ?

Suchproblem: Finde einen Weg von s nach t.

Optimierungsproblem: Finde einen *kürzesten* Weg von s nach t.

Zählproblem: Bestimme die *Anzahl* der Wege von s nach t.

Optimierungsprobleme

Ein *Optimierungsproblem* ist gegeben durch:

- Eine Menge I von Instanzen.
- Für jedes $x \in I$ eine Menge $S(x)$ von potentiellen Lösungen.
- Einer Funktion m, die jedem Paar (x, y) mit $x \in I$ und $y \in S(x)$ ein Maß $m(x, y)$ zuordnet.
- Ein Ziel $\text{goal} \in \{\min, \max\}$.

Eine Lösung $y^* \in S(x)$ heißt optimal, wenn

$$m(x, y^*) = m^*(x) := \text{goal}\{\, m(x, y)\, ;\, y \in S(x)\,\}$$

$$S^*(x) := \{\, y^* \in S(x)\, ;\, m(x, y^*) = m^*(x)\,\}$$

Beispiele

Minimum Path

Instanz:	ungerichteter Graph $G = (V, E)$, Ecken s, t
Lösung:	Weg von s nach t
Maß:	Länge des Weges

Minimum Vertex Cover

Instanz:	ungerichteter Graph $G = (V, E)$
Lösung:	vertex cover $U \subseteq V$
Maß:	$\lvert U \rvert$

Maximum Clique

Instanz:	ungerichteter Graph $G = (V, E)$
Lösung:	Clique $U \subseteq V$
Maß:	$\lvert U \rvert$

Problemvarianten

Zu einem Optimierungsproblem P sind definiert:

konstruktives Problem P_C: Gegeben $x \in I$, finde $y \in S^*(x)$.

Auswertungsproblem P_E: Gegeben $x \in I$, berechne $m^*(x)$.

Entscheidungsproblem P_D: Gegeben $x \in I$ und $k \in \mathbb{N}$, ist $\mathrm{goal}(m^*(x), k) = m^*(x)$?

Beispiele:

P = Minimum Vertex Cover $\quad P_D$ = VC.
P = Maximum Clique $\quad P_D$ = Clique

Die Klassen NPO und PO

Ein Optimierungsproblem ist in der Klasse **NPO**, falls

1. I ist in **P**.
2. für alle $x \in I$ und $y \in S(x)$, $|y| \leq q(|x|)$, und für alle x, y mit $|y| \leq q(|x|)$ ist $y \in S(x)$ in polynomialer Zeit entscheidbar.
3. m ist in polynomialer Zeit berechenbar.

Eigenschaft: P in **NPO** $\Longrightarrow$ P_D in **NP**

Ein Optimierungsproblem P in **NPO** ist in der Klasse **PO**, falls das konstruktive Problem P_C in polynomialer Zeit lösbar ist.

Analog: P in **PO** $\Longrightarrow$ P_D in **P**

NP-harte Optimierungsprobleme

Problem P in **NPO** ist **NP**-hart, falls $Q \leq_T P_C$ für alle $Q \in \mathbf{NP}$.

Eigenschaft: P_D **NP**-vollständig $\Longrightarrow$ P ist **NP**-hart.

$$\mathbf{P} \neq \mathbf{NP} \quad \Longrightarrow \quad \mathbf{PO} \neq \mathbf{NPO}$$

Minimum Traveling Salesperson

Instanz: n Städte $c_1, \ldots, c_n$, Distanzmatrix $D \in \mathbb{N}^{n \times n}$

Lösung: Tour: Permutation $c_{i_1}, \ldots, c_{i_n}$ der Städte

Maß: Kosten $\sum_{k=1}^{n-1} D(i_k, i_{k+1}) + D(i_n, i_1)$

Satz: Minimum Traveling Salesperson ist **NP**-hart.

Das Rucksackproblem

Maximum Knapsack

Instanz: Menge $X = \{x_1, \ldots, x_n\}$,
für jedes $x_i \in X$: Gewicht $a_i \in \mathbb{N}$
Wert $p_i \in \mathbb{N}$
Kapazität $b \in \mathbb{N}$

Lösung: Teilmenge $Y \subseteq X$ mit $\sum_{x_i \in Y} a_i \leq b$

Maß: Gesamtwert $\sum_{x_i \in Y} p_i$

Greedy Algorithmus für Maximum Knapsack

Sei $X = \{x_1, \ldots, x_n\}$ mit $\frac{p_1}{a_1} \geq \frac{p_2}{a_2} \geq \cdots \geq \frac{p_n}{a_n}$.

Algorithmus *Greedy Knapsack*:

```
input X, p⃗, a⃗, b
Y := ∅
for i := 1 to n do
   if a_i ≤ b then
      Y := Y ∪ {x_i}
      b := b − a_i
return Y
```

Beispiel: Sei $k \in \mathbb{N}$ beliebig.

- $X = \{x_1, \ldots, x_{n+1}\}$
- $p_i = a_i = 1$ für $i \leq n$
- $p_{n+1} = b - 1$
- $a_{n+1} = b = kn + 1$

$m^* = b - 1 = kn$,
aber $m_{Greedy} = n$.

Greedy Algorithmus für Handlungsreisende

Algorithmus *Nearest Neighbor*:

Wähle c_{i_1} beliebig
Für $1 \leq k \leq n-1$:
wähle als $c_{i_{k+1}}$ ein $c_j \notin \{c_{i_1}, \ldots, c_{i_k}\}$
mit $D(c_{i_k}, c_j)$ minimal.

Minimum Metric Traveling Salesperson:

Spezialfall von Minimum Traveling Salesperson, wo gilt:

Symmetrie: $D(i,j) = D(j,i)$

Dreiecksungleichung: $D(i,j) \leq D(i,k) + D(k,j)$

Performance von *Nearest Neighbor*

Lemma: Sei x eine Instanz von MINIMUM METRIC TRAVELING SALESPERSON, und $\ell : \{c_1, \dots, c_n\} \to \mathbb{Q}$ mit

$$D(i,j) \geq \min(\ell(c_i), \ell(c_j)) \qquad \text{für alle } c_i \neq c_j$$

$$\ell(c_i) \leq \frac{1}{2} m^*(x) \qquad \text{für alle } c_i$$

Dann gilt: $$\sum_{i=1}^{n} \ell(c_i) \leq \frac{1}{2}(\lceil \log n \rceil + 1) m^*(x) \ .$$

Satz: Sei $m_{NN}(x)$ die Länge der vom Algorithmus *Nearest Neighbor* gefundenen Tour. Dann gilt:

$$m_{NN}(x)/m^*(x) \leq \frac{1}{2}(\lceil \log n \rceil + 1) \ .$$

Ein Scheduling-Problem

MINIMUM SCHEDULING ON IDENTICAL MACHINES

Instanz: Menge $T = \{t_1, \dots, t_n\}$ von Aufträgen,
für jedes $t_i \in T$: Dauer $\ell_i \in \mathbb{N}$
Anzahl $p \in \mathbb{N}$ von Prozessoren

Lösung: schedule: Funktion $f : T \to \{1, \dots, p\}$

Maß: makespan: $\max_{1 \leq i \leq p} \sum_{f(t_j)=i} \ell_j$

Finish time: $$A_i(j) := \sum_{\substack{k \leq j \\ f(t_k)=i}} \ell_k$$

List Scheduling

Algorithmus *List Scheduling*:

```
for j := 1 to n do
    assign t_j to processor i with A_i(j-1) minimal
```

Satz: Für jede Instanz x findet *List Scheduling* eine Lösung mit makespan $m_{LS}(x)$, so dass $m_{LS}(x)/m^*(x) \leq (2 - \frac{1}{p})$.

Regel *LPT* (*largest processing time*):

Sortiere T so dass $\ell_1 \geq \ell_2 \geq \ldots \geq \ell_n$, dann *List Scheduling*.

Satz: Für jede Instanz x liefert *LPT* eine Lösung mit makespan $m_{LPT}(x)$ so dass $m_{LPT}(x)/m^*(x) \leq (\frac{4}{3} - \frac{1}{3p})$.

Ein Packungsproblem

MINIMUM BIN PACKING

Instanz: Multimenge $\{a_1, \ldots, a_n\} \subset \mathbb{Q}$ mit $0 < a_i \leq 1$.

Lösung: Partition $B_1, \ldots, B_k$ von A mit $\sum_{a_i \in B_j} a_i \leq 1$ für alle j.

Maß: Anzahl der Blöcke k

Algorithmus *Next Fit*

```
k := 1 ; B_1 := ∅
for i := 1 to n do
    if ΣB_k + a_i ≤ 1
        then B_k := B_k ∪ {a_i}
        else k := k + 1 ; B_k := {a_i}
```

Performance ratio:

Für alle Instanzen x ist

$$m_{NF}(x)/m^*(x) \leq 2$$

Algorithmus *First Fit*

```
k := 1 ; B_1 := ∅
for i := 1 to n do
    flag := true
    for j := 1 to k do
        if ΣB_j + a_i ≤ 1 and flag
            then B_j := B_j ∪ {a_i} ; flag := false
    if flag then k := k + 1 ; B_k := {a_i}
```

Algorithmus *First Fit Decreasing*

Sortiere A so dass $a_1 \geq a_2 \geq \ldots \geq a_n$,
dann wende *First Fit* an.

Performance ratio: Für alle Instanzen x ist $m_{\mathrm{FFD}}(x) \leq \frac{3}{2} m^*(x) + 1$.

Sequentielles Färben von Graphen

Satz: Sei k die Anzahl der Farben, die der sequentielle Algorithmus bei Reihenfolge $v_1, v_2, \ldots, v_n$ verwendet. Dann ist

$$k \leq \max_{1 \leq i \leq n} \min(\deg v_i, i - 1) \, .$$

Decreasing Degree: $\deg v_1 \geq \deg v_2 \geq \ldots \geq \deg v_n$.

Smallest Last: v_i hat minimalen Grad im von $\{v_1 \ldots v_i\}$ induzierten Subgraphen.

Satz: *Smallest Last* färbt jeden *planaren* Graphen mit ≤ 6 Farben.

Lokale Suche

Anfangslösung	Für jedes $x \in I$:	$y_0(x) \in S(x)$
Nachbarschaftsstruktur	Für jedes $y \in S(x)$	$N(x, y) \subseteq S(x)$

Algorithmus *Local Search*

$y := y_0(x)$
solange es $y' \in N(x, y)$ mit $m(x, y')$ besser als $m(x, y)$ gibt
setze $y := y'$

Probleme:

- Wie findet man bessere Nachbarlösungen effizient?
- Wie gut ist ein lokales Optimum?

Beispiel für Lokale Suche

Problem MAXIMUM CUT

Instanz:	Graph $G = (V, E)$
Lösung:	Partition $V = V_0 \cup V_1$ mit $V_0 \cap V_1 = \emptyset$
Maß:	Anzahl der Kanten $\{x, y\}$ mit $x \in V_0$ und $y \in V_1$.

Anfangslösung: $(\emptyset, V)$

Nachbarn von (V_0, V_1) sind (V_0', V_1') mit $\big||V_0| - |V_0'|\big| = 1$

Satz: Für ein lokales Maximum (V_0, V_1) in dieser Nachbarschafts-struktur mit Maß m_N gilt $m^*/m_N \leq 2$.

Runden der Lösung eines linearen Programms

MINIMUM WEIGHTED VERTEX COVER

Instanz:	ungerichteter Graph $G = (V, E)$, Gewicht c_i für $v_i \in V$.
Lösung:	vertex cover $U \subseteq V$
Maß:	$\sum_{v_i \in U} c_i$

Lineares Programm P_{VC}:

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^{n} c_i x_i \\ \text{subject to} & x_i + x_j \geq 1 \ \text{ für alle } \{v_i, v_j\} \in E \\ & x_i \geq 0 \end{array}$$

Optimale Lösung $x_1^*, \ldots, x_n^* \rightarrow$ vertex cover $U := \{v_i \, ; \, x_i^* \geq \frac{1}{2}\}$.

Satz: Für das Maß m_{LP} dieses vertex cover U gilt $m_{LP}/m^* \leq 2$

Primal-Dual Algorithmus für VERTEX COVER

Duales Programm D_{VC}:

$$\begin{array}{lll} \text{maximize} & \displaystyle\sum_{\{v_i, v_j\} \in E} y_{i,j} & \\ \text{subject to} & \displaystyle\sum_{j : \{v_i, v_j\} \in E} y_{i,j} \leq c_i & (C_i) \\ & y_{i,j} \geq 0 & \end{array}$$

Algorithmus:

$\vec{y} := 0$; $U := \emptyset$
while U not a vertex cover
  find $\{v_i, v_j\}$ not covered
  increase $y_{i,j}$ until (C_i) or (C_j) tight
  if (C_i) is tight then
    $U := U \cup \{v_i\}$
  else
    $U := U \cup \{v_j\}$

Satz: Der obige Algorithmus findet ein vertex cover mit Maß m_{PD} derart dass $m_{PD}/m^* \leq 2$.

Dynamic Programming für MAXIMUM KNAPSACK

$P = \sum_{i=1}^{n} p_i$, $A = \sum_{i=1}^{n} a_i$

```
for p := 0 to P
    S(1,p) := A + 1
M(1,0) := ∅ ;          S(1,0) := 0;
M(1,p_1) := {x_1} ;    S(1,p_1) := a_1
for k := 1 to n
    for p := 1 to P
        if   p_k ≤ p and M(k−1, p−p_k) defined
             and S(k−1, p−p_k) + a_k ≤ b
             and S(k−1, p−p_k) + a_k ≤ S(k−1, p)
        then
                M(k,p) := M(k−1, p−p_k) ∪ {x_k}
                S(k,p) := S(k−1, p−p_k) + a_k
        else
                M(k,p) := M(k−1, p)
                S(k,p) := S(k−1, p)
p* := max{ p ; M(n,p) defined }
return M(n,p*)
```

Approximations-Schema für MAXIMUM KNAPSACK

Satz: Der *Dynamic Programming* Algorithmus für MAXIMUM KNAPSACK findet eine optimale Lösung in Zeit $O(n \cdot P)$.

Approximations-Schema:

$p_{max} := \max_i p_i$
$t := \lfloor \log(\frac{r-1}{r} \frac{p_{max}}{n}) \rfloor$
Ersetze $p_i' := \lfloor \frac{p_i}{2^t} \rfloor$ für $i = 1, \ldots, n$
Wende den *Dynamic Programming* Algorithmus an

Satz: Das obige Approximations-Schema findet bei Eingabe von $1 < r \in \mathbb{Q}$ in Zeit $O(rn^3/(r-1))$ eine Lösung mit Maß $m_{AS}(r)$ so dass $m^*/m_{AS}(r) \leq r$.

Relative Approximation

Definition: Optimierungsproblem P, $x \in I_P$ und $y \in S_P(x)$.

Performance ratio: $R(x,y) := \max\left(\frac{m(x,y)}{m^*(x)}, \frac{m^*(x)}{m(x,y)}\right)$

Relativer Fehler: $E(x,y) := \frac{|m^*(x) - m(x,y)|}{\max(m^*(x), m(x,y))} = 1 - \frac{1}{R(x,y)}$

Für eine Konstante $r \geq 1$ ist ein r-Approximationsverfahren ein Algorithmus A, der für jedes $x \in I_P$ ein $A(x) \in S_P(x)$ mit $R(x, A(x)) \leq r$ liefert.

Ein Optimierungsproblem P heißt r-approximierbar, falls es ein Polynomialzeit-r-Approximationsverfahren für P gibt.

Die Klasse **APX**

APX ist die Klasse aller Optimierungsprobleme $P \in$ **NPO**, die r-approximierbar für ein $r \geq 1$ sind.

$$\mathbf{PO} \subseteq \mathbf{APX} \subseteq \mathbf{NPO}$$

APX enthält **NP**-harte Probleme, z.B. Maximum Knapsack, also

$$\mathbf{P} \neq \mathbf{NP} \implies \mathbf{PO} \subsetneq \mathbf{APX}$$

Maximum Satisfiability

Instanz:	CNF $D_1 \wedge \ldots \wedge D_m$ in Variablen $x_1, \ldots, x_n$
Lösung:	Bewertung $\alpha : \{x_1, \ldots, x_n\} \to \{0, 1\}$
Maß:	Anzahl der erfüllten Klauseln $\#\{i; \alpha \models D_i\}$

Maximum Satisfiability $\in$ **APX**.

Traveling Salespersons revisited

Satz: MINIMUM TRAVELING SALESPERSON ist nicht in **APX**, außer wenn $\mathbf{P} = \mathbf{NP}$.

$$\mathbf{P} \neq \mathbf{NP} \implies \mathbf{PO} \subsetneq \mathbf{APX} \subsetneq \mathbf{NPO}$$

Dagegen ist MINIMUM METRIC TRAVELING SALESPERSON $\frac{3}{2}$-approximierbar durch *Christofides Algorithmus*:

Berechne einen Minimalen Spannbaum T
Erweitere T zu einem Eulerschen Graphen E:
Finde ein minimales Matching M von $\{c_i\ ;\ \deg_T(c_i) \text{ ungerade}\}$
Setze $E := T \cup M$
Berechne eine Euler-Tour in E
Extrahiere aus dieser eine TSP-Tour

Approximationsschemata

Definition: Ein Polynomialzeit-Approximationsschema für P ist ein Algorithmus A mit

- Eingabe: $x \in I_P$ und $1 < r \in \mathbb{Q}$, Ausgabe: $A(x, r) \in S_P(x)$
- Performance ratio $R(x, A(x, r)) \leq r$
- Laufzeit $|x|^{O(1)}$

PTAS ist die Klasse aller Optimierungsprobleme in **NPO**, für die es ein Polynomialzeit-Approximationsschema gibt.

$$\mathbf{PO} \subseteq \mathbf{PTAS} \subseteq \mathbf{APX} \subseteq \mathbf{NPO}$$

PTAS enthält **NP**-harte Probleme (z.B. MAXIMUM KNAPSACK):

$$\mathbf{P} \neq \mathbf{NP} \implies \mathbf{PO} \subsetneq \mathbf{PTAS}$$

Beispiel: ein Approximationsschema

MINIMUM PARTITION

Instanz:	Menge $X = \{x_1, \ldots, x_n\}$, für $x_i \in X$ Gewicht $a_i \in \mathbb{N}$
Lösung:	Partition $X = Y_1 \uplus Y_2$
Maß:	$\max\left(\sum_{x_i \in Y_1} a_i \,, \sum_{x_i \in Y_2} a_i \right)$

Approximationsschema:

sortiere X so dass $a_1 \geq a_2 \geq \ldots \geq a_n$
$k := \lceil (2-r)/(r-1) \rceil$
finde eine optimale Partition $Y_1 \uplus Y_2 = \{x_1, \ldots, x_k\}$
for $j := k+1$ to n
  if $\sum Y_1 \leq \sum Y_2$
    then $Y_1 := Y_1 \cup \{x_j\}$
    else $Y_2 := Y_2 \cup \{x_j\}$

Ein Problem in PTAS

MAXIMUM INTEGER KNAPSACK

Instanz:	Menge $X = \{x_1, \ldots, x_n\}$, für jedes $x_i \in X$: Gewicht $a_i \in \mathbb{N}$ Wert $p_i \in \mathbb{N}$ Kapazität $b \in \mathbb{N}$
Lösung:	Funktion $c : X \to \mathbb{N}$ mit $\sum_{x_i \in Y} c(x_i) a_i \leq b$
Maß:	Gesamtwert $\sum_{x_i \in Y} c(x_i) p_i$

Approximationsschema für Integer Knapsack

```
sortiere X so dass p_1 ≥ p_2 ≥ ... ≥ p_n
c := 0⃗
δ := ⌈1/(r−1)⌉
for all f with ∑ f(x_i) ≤ δ and ∑ f(x_i)a_i ≤ b
    k := max_i f(x_i) > 0
    b_k := b − ∑ f(x_i)a_i
    sei x_m ∈ {x_k, ..., x_n} mit p_m/a_m maximal
    f(x_m) := f(x_m) + ⌊b_k/a_m⌋
    if ∑ f(x_i)p_i ≥ ∑ c(x_i)p_i then
        c := f
```

APX versus PTAS

Lückentechnik: Sei Q **NP**-vollständig und $P \in$ **NPO**. Wenn es Funktionen $f : I_Q \to I_P$ und $c : I_Q \to \mathbb{N}$ in **FP** gibt mit

$$\begin{array}{ll} m^*(f(x)) \leq c(x) & \text{für } x \in Y_Q \\ m^*(f(x)) \geq c(x)(1+G) & \text{für } x \in N_Q \end{array}$$

für ein $G > 0$, dann ist P nicht r-approximierbar für $r < 1 + G$.

Satz: Falls **P** $\neq$ **NP**, dann ist Minimum Bin Packing nicht in **PTAS**.

$$\mathbf{P} \neq \mathbf{NP} \implies \mathbf{PO} \subsetneq \mathbf{PTAS} \subsetneq \mathbf{APX} \subsetneq \mathbf{NPO}$$

Volle Approximationsschemata

Definition: Ein volles Polynomialzeit-Approximationsschema für P ist ein Algorithmus A mit

- Eingabe: $x \in I_P$ und $1 < r \in \mathbb{Q}$, Ausgabe: $A(x, r) \in S_P(x)$
- Performance ratio $R(x, A(x, r)) \leq r$
- Laufzeit polynomial in x und $\frac{1}{r-1}$

FPTAS ist die Klasse aller Optimierungsprobleme in **NPO**, für die es ein volles Polynomialzeit-Approximationsschema gibt.

$$\mathbf{PO} \subseteq \mathbf{FPTAS} \subseteq \mathbf{PTAS} \subseteq \mathbf{APX} \subseteq \mathbf{NPO}$$

FPTAS enthält **NP**-harte Probleme (z.B. MAXIMUM KNAPSACK):

$$\mathbf{P} \neq \mathbf{NP} \implies \mathbf{PO} \subsetneq \mathbf{FPTAS}$$

Asymptotische Approximationsschemata

Definition: Ein asymptotisches Polynomialzeit-Approximationsschema für P ist ein Algorithmus A mit

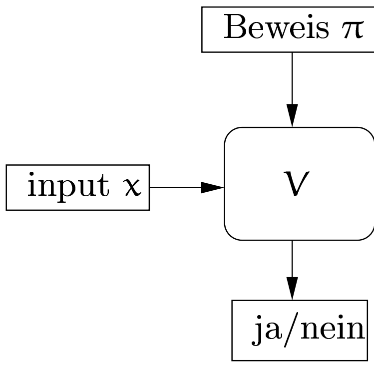
- Eingabe: $x \in I_P$ und $1 < r \in \mathbb{Q}$, Ausgabe: $A(x, r) \in S_P(x)$
- Performance ratio $R(x, A(x, r)) \leq r + \frac{k}{m^*(x)}$ für k konstant
- Laufzeit $|x|^{O(1)}$

PTAS$^\infty$ ist die Klasse aller Optimierungsprobleme in **NPO**, für die es ein asymptotisches Polynomialzeit-Approximationsschema gibt.

$$\mathbf{PTAS} \subseteq \mathbf{PTAS}^\infty \subseteq \mathbf{APX}$$

$\mathbf{NP}$ = Prüfen von Beweisen

Nichtdeterministische TM (Verifikator):



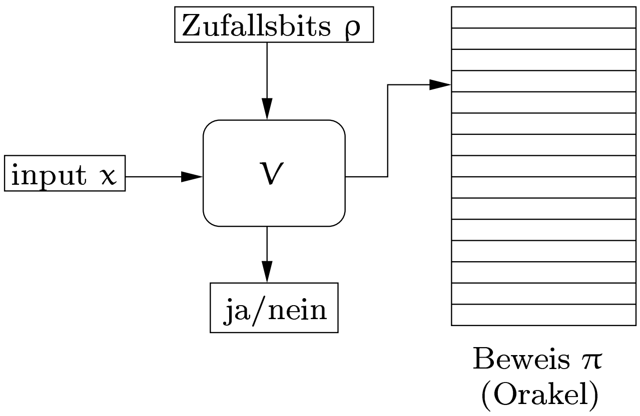
Entscheidungsproblem P ist in $\mathbf{NP}$ gdw. es gibt Verifikator V mit

$$x \in Y_P \quad \rightarrow \quad \exists \pi \quad V(x,p) = \text{ja}$$

$$x \in N_P \quad \rightarrow \quad \forall \pi \quad V(x,p) = \text{nein}$$

Probabilistisch überprüfbare Beweise

Verifikator:



Verifikator V ist $(r(n), q(n))$-beschränkt, falls bei $|x| = n$

- V liest $\leq r(n)$ Zufallsbits,
- berechnet daraus $\leq q(n)$ Orakelfragen
- entscheidet anhand der Antworten.

Die Klasse **PCP**

Ein Entscheidungsproblem P ist in $\mathbf{PCP}[r(n), q(n)]$ gdw. es einen $(r(n), q(n))$-beschränkten Verifikator V gibt mit

$$x \in Y_P \quad \to \quad \exists \pi \;\; \mathbb{P}_\rho[V(x,\rho,\pi) = \text{ja}] = 1$$

$$x \in N_P \quad \to \quad \forall \pi \;\; \mathbb{P}_\rho[V(x,\rho,\pi) = \text{nein}] \geq \frac{1}{2}$$

Korollar: $\mathbf{NP} = \mathbf{PCP}[0, n^{O(1)}]$

sogar: $\mathbf{NP} = \mathbf{PCP}[O(\log n), n^{O(1)}]$

PCP-Theorem (Arora, Lund, Motwani, Safra, Sudan, Szegedy 92):

$$\mathbf{NP} = \mathbf{PCP}[O(\log n), O(1)]$$

Folgerungen aus dem **PCP**-Theorem

Satz: Falls $\mathbf{P} \neq \mathbf{NP}$, so ist Maximum Satisfiability $\notin$ **PTAS**.

Durch Reduktion erhält man:

Satz: Falls $\mathbf{P} \neq \mathbf{NP}$, so ist Maximum Clique $\notin$ **PTAS**.

Korollar: Falls $\mathbf{P} \neq \mathbf{NP}$, so ist Maximum Clique $\notin$ **APX**

Beweis des PCP-Theorems

1. $\mathrm{NP} \subseteq \mathbf{PCP}[n^{O(1)}, O(1)]$

2. $\mathrm{NP} \subseteq \mathbf{PCP}[O(\log n), \log^{O(1)} n]$

3. Komposition:

$$\mathrm{NP} \subseteq \mathbf{PCP}[r_1(n), q_1(n)] \qquad \text{und} \qquad \mathrm{NP} \subseteq \mathbf{PCP}[r_2(n), q_2(n)]$$

$$\Longrightarrow \quad \mathrm{NP} \subseteq \mathbf{PCP}[r_1(n) + r_2(c \cdot q_1(n)), q_2(c \cdot q_1(n))]$$

Arithmetisierung

$$\begin{aligned}
\text{Klauseln} &\mapsto \text{Polynome über } \mathbb{Z}_2 \\
x &\mapsto p_x := (1 - x) \\
\bar{x} &\mapsto p_{\bar{x}} := x \\
D = \ell_1 \vee \ldots \vee \ell_k &\mapsto p_D := p_{\ell_1} \cdot \ldots \cdot p_{\ell_k} \\
\vec{r} \in \mathbb{Z}_2^m \;,\; C = D_1 \wedge \ldots \wedge D_m &\mapsto p_C^{\vec{r}} := r_1 \cdot p_{D_1} + \ldots + r_m \cdot p_{D_m}
\end{aligned}$$

Lemma: Für $0 \neq \vec{v} \in \mathbb{Z}_2^m$ gilt: $\mathbb{P}_{\vec{r} \in \mathbb{Z}_2^m}[\sum_i r_i v_i = 1] = \frac{1}{2}$.

Für jede Bewertung α gilt:

$$\alpha \models C \quad \rightarrow \quad \mathbb{P}_{\vec{r} \in \mathbb{Z}_2^m}[p_C^{\vec{r}}(\alpha) = 0] = 1$$

$$\alpha \not\models C \quad \rightarrow \quad \mathbb{P}_{\vec{r} \in \mathbb{Z}_2^m}[p_C^{\vec{r}}(\alpha) = 0] = \frac{1}{2}$$

Lineare Funktionen

$f, g : D \to R$ sind δ-*nah*, falls $\mathbb{P}_{x \in D}[f(x) \neq g(x)] \leq \delta$.

Linearitätstest:

Für $g : \mathbb{Z}_2^m \to \mathbb{Z}_2$ sei

$$\mathbb{P}_{x,y \in \mathbb{Z}_2^m}[g(x+y) \neq g(x) + g(y)] \leq \frac{\delta}{2} .$$

Dann ist g δ-nah zu einem *linearen* $f : \mathbb{Z}_2^m \to \mathbb{Z}_2$.

Korrektureigenschaft:

Sei $f : \mathbb{Z}_2^m \to \mathbb{Z}_2$ linear, und $g : \mathbb{Z}_2^m \to \mathbb{Z}_2$ δ-nah zu f. Dann gilt

$$\forall x \in \mathbb{Z}_2^m \qquad \mathbb{P}_{y \in \mathbb{Z}_2^m}[g(x+y) - g(x) = f(x)] \geq 1 - 2\delta .$$

Satz: Sei $a \in \mathbb{Z}_2^n$. Dann gibt es lineare Funktionen

$$A_a : \mathbb{Z}_2^n \to \mathbb{Z}_2 \qquad B_a : \mathbb{Z}_2^{n^2} \to \mathbb{Z}_2 \qquad C_a : \mathbb{Z}_2^{n^3} \to \mathbb{Z}_2$$

so dass $\forall$ Polynome $p \in \mathbb{Z}_2[x_1, \ldots, x_n]$ vom Grad 3 gilt

$$p(a) = \alpha_p + A_a(q_{p,1}) + B_a(q_{p,2}) + C_a(q_{p,3})$$

für geeignete $\alpha_p \in \mathbb{Z}_2$, $q_{p,1} \in \mathbb{Z}_2^n$, $q_{p,2} \in \mathbb{Z}_2^{n^2}$ und $q_{p,3} \in \mathbb{Z}_2^{n^3}$, die nur von p abhängen.

Definition: Für $x \in \mathbb{Z}_2^m$, $y \in \mathbb{Z}_2^k$ sei $x \circ y := z \in \mathbb{Z}_2^{mk}$ mit $z_{im+j} := x_i y_j$.

Aus Beweis: $\qquad A_a = a$, $B_a = a \circ a$, $C_a = a \circ (a \circ a)$.

A, B, C heissen *konsistent*, falls $\exists a$ mit $A = A_a,\ B = B_a,\ C = C_a$.

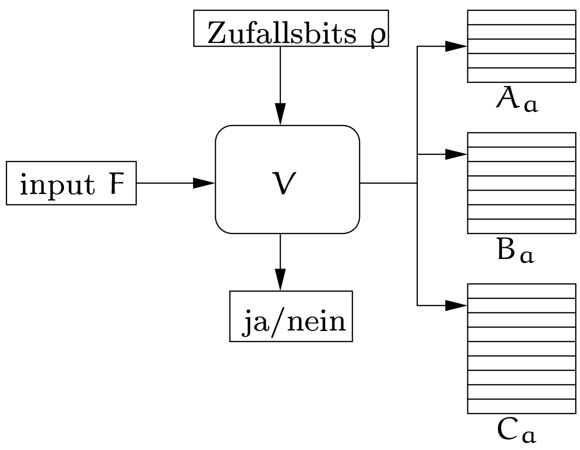
Konsistenztest:

Gegeben: A', B', C' sind δ-nah zu linearen Funktionen A, B, C

Wähle zufällig $x, x' \in \mathbb{Z}_2^n$;
$a := A(x)$; $a' := A(x')$; $b := B(x \circ x')$;
if $aa' \neq b$ then reject;
Wähle zufällig $x \in \mathbb{Z}_2^n$, $y \in \mathbb{Z}_2^{n^2}$;
$a := A(x)$; $b := B(y)$; $c := C(x \circ y)$;
if $ab \neq c$ then reject else accept;

(Berechne A, B, C jeweils mittels Korrektureigenschaft)

Falls A, B, C konsistent sind, akzeptiert der Konsistenztest, andernfalls weist er mit hinreichender Wahrscheinlichkeit zurück.



Verifikator V:

Gegeben: Orakel A, B, C, 3CNF F
Linearitätstest: sind A, B, C δ-nah zu linearen Funktion A', B', C'
Konsistenztest: gibt es a mit $A' = A_a,\ B' = B_a,\ C' = C_a$
Wähle zufällig $\vec{r}$, setze $p := p_F^{\vec{r}}$
Berechne $\alpha_p,\ q_{p,1},\ q_{p,2},\ q_{p,3}$
Berechne $v := \alpha_p + A_a(q_{p,1}) + B_a(q_{p,2}) + C_a(q_{p,3})$
if $v = 0$ then accept else reject

Größe des Beweises: $|A_a| + |B_a| + |C_a| = 2^{n^3} + 2^{n^2} + 2^n$

Also: Anzahl der Zufallsbits $= O(n^3)$.

Anzahl der Orakelfragen ist konstant.

AP-Reduktion

Definition: P ist AP-reduzierbar auf Q, $P \leq_{AP} Q$, falls es Funktionen f und g und $\alpha \geq 1$ gibt mit:

- für $x \in I_P$ und $1 < r \in \mathbb{Q}$ ist $f(x,r) \in I_Q$,
- ist $S_P(x) \neq \emptyset$, dann ist $S_Q(f(x,r)) \neq \emptyset$,
- für $y \in S_Q(f(x,r))$ ist $g(x,y,r) \in S_P(x)$,
- für alle $r > 1$ sind $\lambda x.f(x,r)$ und $\lambda x,y\,.\,g(x,y,r)$ in **FP**,
- für alle $y \in S_Q(f(x,r))$ gilt

$$R_Q(f(x,r),y) \leq r \quad \Longrightarrow \quad R_P(x,g(x,y,r)) \leq 1 + \alpha(r-1)\ .$$

Abschluss unter AP-Reduktion

Lemma:

$$Q \in \textbf{APX} \quad \text{und} \quad P \leq_{AP} Q \quad \Longrightarrow \quad P \in \textbf{APX}$$
$$Q \in \textbf{PTAS} \quad \text{und} \quad P \leq_{AP} Q \quad \Longrightarrow \quad P \in \textbf{PTAS}$$

Beispiele:

Maximum Satisfiability $\leq_{AP}$ Maximum Clique
Maximum Clique $\leq_{AP}$ Maximum Independent Set

Korollar: $\textbf{P} \neq \textbf{NP} \quad \Longrightarrow$ Maximum Independent Set $\notin$ **APX**

L-Reduktion

Definition: P ist L-reduzierbar auf Q, $P \leq_L Q$, falls es Funktionen $f : I_P \to I_Q$ und g in **FP** und $\beta, \gamma \geq 0$ gibt mit:

- ist $S_P(x) \neq \emptyset$, dann ist $S_Q(f(x)) \neq \emptyset$,
- für $y \in S_Q(f(x))$ ist $g(x, y) \in S_P(x)$,
- für alle $x \in I_P$ ist $m_Q^*(f(x)) \leq \beta m_P^*(x)$
- für alle $y \in S_Q(f(x))$ gilt

$$|m_P^*(x) - m_P(x, g(x, y))| \leq \gamma |m_Q^*(f(x)) - m_Q(f(x), y)| \, .$$

Satz: Ist $P \leq_L Q$ und $P \in$ **APX**, dann auch $P \leq_{AP} Q$.

Ein NPO-vollständiges Problem

MAXIMUM (MINIMUM) WEIGHTED SATISFIABILITY:

Instanz:	Aussagenlogische Formel φ in Variablen $x_1, \ldots x_n$, für jede Variable x_i Gewicht $w_i \geq 0$
Lösung:	Bewertung α mit $\alpha \models \varphi$
Maß:	$\max(1, \sum_{i=1}^n w_i \alpha(x_i))$

Satz: MAXIMUM (MINIMUM) WEIGHTED SATISFIABILITY ist vollständig für die Klasse der Maximierungs- (Minimierungs-) probleme in **NPO**.

Satz: MAXIMUM WEIGHTED SATISFIABILITY $\leq_{AP}$ MINIMUM WEIGHTED SATISFIABILITY, und umgekehrt.

APX-Vollständigkeit

Satz: Maximum 3-Satisfiability ist vollständig für die Klasse der Maximierungsprobleme in **APX**.

Satz: Für jedes Minimierungsproblem P in **APX** gibt es ein Maximierungsproblem $\mathsf{P}' \in$ **APX** mit $\mathsf{P} \leq_{\mathrm{AP}} \mathsf{P}'$.

Korollar: Maximum 3-Satisfiability ist **APX**-vollständig.

Mehr **APX**-vollständige Probleme

Lemma:

1. Maximum 3-Satisfiability $\leq_{\mathrm{L}}$ Maximum 2-Satisfiability
2. Maximum 2-Satisfiability $\leq_{\mathrm{L}}$ Maximum Not-All-Equal 3-Satisfiability
3. Maximum Not-All-Equal 3-Satisfiability $\leq_{\mathrm{L}}$ Maximum Cut

Ausserdem: Maximum Not-All-Equal 3-Satisfiability $\in$ **APX**

Korollar: Maximum 2-Satisfiability, Maximum Not-All-Equal 3-Satisfiability und Maximum Cut sind **APX**-vollständig.

Eingeschränkte Erfüllbarkeitsprobleme:

Maximum 3-Satisfiability($\mathfrak{b}$): Einschränkung von Maximum 3-Satisfiability auf Instanzen, in denen jede Variable höchstens $\mathfrak{b}$ mal vorkommt.

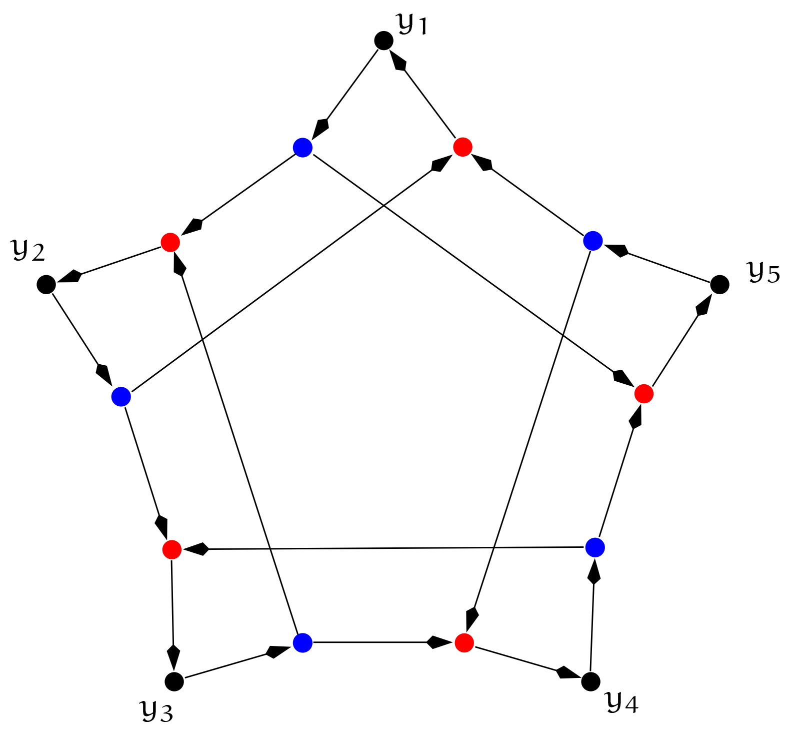
Lemma:

Maximum 3-Satisfiability $\leq_L$ Maximum 3-Satisfiability(29)

Maximum 3-Satisfiability(29) $\leq_L$ Maximum 3-Satisfiability(5) (Übung)

Maximum 3-Satisfiability(5) $\leq_L$ Maximum 3-Satisfiability(3)

Satz: Maximum 3-Satisfiability(3) ist **APX**-volständig.



Ein Problem in log-**APX**

Minimum Set Cover

Instanz:	Menge C von Teilmengen einer Menge S
Lösung:	Teilmenge $C' \subseteq C$ sodass $\forall s \in S\ \exists c \in C'$ mit $s \in c$
Maß:	$\lvert C' \rvert$

Minimum Set Cover ist äquivalent zu Minimum Hitting Set.

Greedy Algorithmus für Minimum Set Cover

Sei $C := \{c_1, \ldots, c_m\}$, $|S| = n$.

```
U := S ; C' := ∅
for i := 1 to m do c'_i := c_i
while U ≠ ∅ do
   choose i with |c'_i| maximal
   C' := C' ∪ {c_i}
   U := U \ c'_i
   for j := 1 to m do c'_j := c'_j \ c_i
return C'
```

Satz: Der obige Algorithmus ist ein $(\lfloor \ln n \rfloor + 1)$-Approximationsverfahren, also Minimum Set Cover $\in$ log-**APX**.

Algorithmus für Minimum Graph Coloring

```
U := V ; i := 0;
while U ≠ ∅ do
   i := i + 1 ; W := U
   while W ≠ ∅ do
      choose v of minimal degree in W
      f(v) := i
      W := W \ ({v} ∪ Γ(v))
   U := U \ f^{-1}[i]
return f
```

Satz: Der obige Algorithmus färbt einen k-färbbaren Graphen mit höchstens $3|V|/\log_k |V|$ Farben.

Korollar: Der Algorithmus ist ein $O(n/\log n)$-Approximationsverfahren, also ist Minimum Graph Coloring $\in$ poly-**APX**.