Methoden zum Entwurf : Greedy

Greedy Algorithmen :

geeignet, wenn Lösungen Mengen von Objekten sind

(z.B. in Graphen: Menge von Ecken / Kanten ...)

Lösung wird schrittweise aufgebaut : in jedem Schritt ein Element aufnehmen oder nicht.

Greedy - Strategie : in jedem Schritt Profit maximieren.

(liefert optimale Lösung, wenn Matroid - Struktur)

in anderen Fällen: gute Approximation

1. Knapsack (Rucksackproblem) Folie

Greedy - Strategie 1. Versuch : immer Objekt mit max. Wert

schlecht wenn: z.B

$b = 11$

i	1	2	3	4	5	6
a_i	10	2	2	2	2	2
p_i	5	3	3	3	3	3

bessere Greedy Strategie: $\frac{\text{Wert}}{\text{Gewicht}}$ (Folie!)

Beispiel zeigt: Performance ratio $= k$ beliebig gross !

Aber: leichte modifikation : $p_{max} := \max_i p_i$

$M_H := \max (p_{max}, m_{Greedy})$

Satz: Für alle Instanzen von MAXIMUM KNAPSACK ist

$$\frac{m^*}{m_H} < 2$$

Bew: Sei x_j das erste Element, das von GREEDY nicht ausgewählt wird.

$$\bar{p}_j := \sum_{i=1}^{j-1} p_i \leq m_{Greedy} \qquad \bar{a}_j := \sum_{i=1}^{j-1} a_i \leq b$$

Behauptung: $m^* \leq \bar{p}_j + p_j$

Denn: f. $\nu \geq 1$ $\frac{p_\nu}{a_\nu} \leq \frac{p_j}{a_j}$, also

$$m^* \leq \bar{p}_j + (b - \bar{a}_j) \frac{p_j}{a_j}$$

nun ist $\bar{a}_j + a_j > b$,

also $a_j > b - \bar{a}_j$

$$< \bar{p}_j + p_j$$

□ Beh.

Nun Fall 1: $p_j \leq \bar{p}_j$

$$m^* < \bar{p}_j + p_j \leq 2\bar{p}_j \leq 2m_{Greedy} \leq 2m_H$$

Fall 2: $p_j > \bar{p}_j$, also $p_{max} > \bar{p}_j$

$$m^* < \bar{p}_j + p_j \leq \bar{p}_j + p_{max} < 2p_{max} \leq 2m_H$$ □

Greedy Algorithmus für TSP

Minimum Traveling Salesperson (Erinnerung an Problem!)
Greedy Methode anwendbar bei schrittweiser Wahl von Kanten der Tour:
Nearest Neighbor Algorithmus (Folie!)
Verhalten des Algorithmus beim allgemeinen Problem: unbekannt.
Spezialfall: Distanz ist eine Metrik (Folie!)
(symmetrisch, dreiecksungl.)

Analyse von Nearest Neighbor für Minimum Metric Traveling Salesperson

Lemma (Folie!)

Beweis: OBdA sei $c_1 \dots c_n$ sortiert $l(c_1) \geq l(c_2) \geq \dots \geq l(c_n)$

Behauptung: $m^*(x) \geq 2 \sum_{i=k+1}^{\min(2k,n)} l(c_i)$ f. alle k

Sei I^* optimale Tour, Kosten m^*

$C_k := \{c_i, 1 \leq i \leq \min(2k,n)\}$

I_k: Tour der Städte in C_k in der Reihenfolge wie in I^*
Länge m_k

(c_r, c_s) in I_k: entweder (c_r, c_s) in I^*
oder in I^* Pfad $c_r \dots c_s$, dann aber wegen Δ
$D(r,s) \leq$ Länge des Pfades.

also: $m^* \geq m_k$

$$m_k = \sum_{(c_i,c_j) \text{ in } I_k} D(i,j) \geq \sum_{(c_i,c_j) \text{ in } I_k} \min(l(c_i), l(c_j)) =: \sum_{c_i \in C_k} \alpha_i l(c_i) \qquad \alpha_i = 0,1,2$$

wähle $\alpha_i = 0$ für $c_1 \dots c_k$ $\alpha_i = 2$ sonst → wird kleiner.

Also: $m^* \geq m_k \geq 2 \sum_{i=k+1}^{\min(2k,n)} \ell(c_i)$ □ Behauptung

Aufsummieren der Behauptung für $k = 2^j$, $j = 0, 1, \ldots, \lceil \log n \rceil - 1$:

$$\lceil \log n \rceil \, m^*(x) \geq \sum_{j=0}^{\lceil \log n \rceil - 1} 2 \cdot \sum_{i=2^j+1}^{\min(2^{j+1}, n)} \ell(c_i)$$

$$= 2 \sum_{i=2}^{n} \ell(c_i).$$

Ausserdem $m^*(x) \geq 2\ell(c_1)$, also folgt

$$(\lceil \log n \rceil + 1)\, m^*(x) \geq 2 \sum_{i=1}^{n} \ell(c_i)$$ □

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

Sei $c_{k_1}, \ldots, c_{k_n}$ die von Nearest Neighbor gefundene Tour

Definiere eine Funktion ℓ durch:

$\ell(c_{k_i}) = D(k_i, k_{i+1})$ f. $1 \leq i < n$

$\ell(c_{k_n}) = D(k_n, k_1)$

$m_{NN}(x) = \sum_{i=1}^{n} \ell(c_{k_i}) \leq \frac{1}{2}(\lceil \log n \rceil + 1)\, m^*(x)$ nach Lemma.

Bleibt Voraussetzung des Lemmas zu prüfen:

1) $D(i,j) \geq \min(\ell(c_i), \ell(c_j))$.

Sei $i = k_r$, $j = k_s$.

Fall 1. $r < s$. Dann wäre k_s eine mögliche Wahl für den nächsten Punkt k_{r+1} gewesen, also wegen greediness:

$D(k_r, k_s) \geq D(k_r, k_{r+1}) = \ell(c_{k_r})$

Fall 2 $s < r$ Dann gilt analog

$D(k_s, k_r) \geq D(k_s, k_{s+1}) = \ell(c_{k_s})$

} $D(k_r, k_s) \geq \min(\ell(c_{k_r}), \ell(c_{k_s}))$ □

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

Sei $\ell(c_i) = D(i,j)$. Optimale Tour besteht aus zwei Pfaden von c_i nach c_j und zurück.

$\rightsquigarrow m^*(x) \leq 2\, D(i,j)$ wegen Dreiecksungleichung ■

Sequentielle Algorithmen

Anwendbar für Partitionierungsprobleme:

Lösungen sind Partitionen einer durch die Instanz gegebenen Menge.

Sequentieller Algorithmus:

sortiert Menge (nach irgendeinem Kriterium) $x_1 \dots x_n$

baut Partition der Reihe nach auf:

$P := \{ \{x_1\} \}$

for $i := 2$ to n

if x_i kann zu einem $p \in P$ zugefügt werden

$P := P \setminus \{p\} \cup \{ p \cup \{x_i\} \}$

else

$P := P \cup \{ \{x_i\} \}$

Ein Scheduling Problem

Problem: Scheduling Jobs on Identical Machines (FOLIE)

NP-hart, sogar für $p = 2$

Regel zum Zuweisen des nächsten Jobs:

wähle jeweils den Prozessor, der mit den bisher zugewiesenen am schnellsten fertig wird.

Def. $A_i(j)$ (Folie).

~> List Scheduling (Folie)

Performance ratio $(2 - \frac{1}{p})$ f. jede Reihenfolge !!

Beweis:

$W := \sum_{k=1}^{n} l_k$ gesamte Zeit. offenbar $m^*(x) \geq \frac{W}{p}$

Sei h Prozessor, der als letzter fertig wird.

$A_h(n) = m_{LS}(x)$

sei t_j der letzte Job, der h zugewiesen wurde.

F. alle $i \neq h$ ist $A_i(n) \geq A_h(n) - l_j$,

also $W \geq p(A_h(n) - l_j) + l_j$

~> $m_{LS}(x) = A_h(n) \leq \frac{W}{p} - \frac{(p-1) l_j}{p}$

Grobe Abschätzung: $m^*(x) \geq \frac{W}{p}$ (s.o.) $\quad m^*(x) \geq l_j$

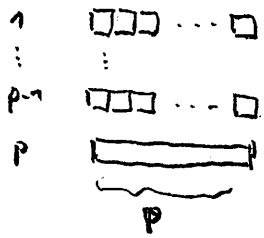
$$m_{LS}(x) \leq \frac{W}{p} + \frac{p-1}{p} l_j \leq m^*(x) + \frac{p-1}{p} m^*(x) = \left(2 - \frac{1}{p}\right) m^*(x)$$

Analyse ist trotz grober Abschätzung scharf:

Beispiel:

- p Prozessoren
- $p(p-1)$ Jobs $\quad t_1 \dots t_{p(p-1)}$ mit $l_i = 1 \quad i = 1 \dots p(p-1)$
- 1 Job $\quad t_{p(p-1)+1}$ mit $l_i = p$

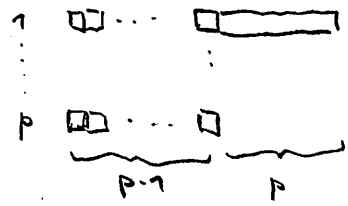
- Optimales Schedule:



$m^*(x) = p$

$$m_{LS}/m^* = 2 - \frac{1}{p}$$ ✓

- List Scheduling mit dieser Reihenfolge



$m_{LS}(x) = 2p-1$

Bessere performance ist möglich für bestimmte Ordnungen.

Regel LPT: Sortiere $t_1 \dots t_n$ sodass $l_1 \geq l_2 \geq \dots \geq l_n$.

$$m_{LPT}(x)/m^*(x) \leq \left(\frac{4}{3} - \frac{1}{3p}\right)$$

Beweis:

Fall 1: $l_n > \frac{m^*(x)}{3}$:

- jeder Prozessor bekommt höchstens zwei Aufträge zugewiesen
  → $n \leq 2p$
  a) $n \leq p$: trivial
  b) $p+1 \leq n \leq 2p$: von LPT gefundene Lösung ist optimal, Vertauschung von zwei Jobs $t_{p+1} \dots t_n$ macht Lösung höchstens schlechter.

In jedem Fall $m^*(x) = m_{LPT}(x)$.

Fall 2: $l_n \leq \frac{m^*(x)}{3}$

wie oben: h sei Prozessor, dem t_n zugewiesen.

$$W \geq p \cdot (A_h(n) - l_n) + l_n$$

$$\rightarrow \quad m_{LPT}(x) \leq \frac{W}{p} + \frac{p-1}{p} l_n$$

$\frac{W}{p} \leq m^*(x)$, $l_n \leq \frac{m^*(x)}{3}$

$$m_{LPT}(x) \leq m^*(x) + \frac{p-1}{3p} m^*(x) = \left(\frac{4}{3} - \frac{1}{3p}\right) m^*(x)$$

Packungsproblem: MINIMUM BIN PACKING (Folie)

- modelliert Speicherzuteilungsprobleme

einfachster sequentieller Algorithmus: Next Fit (Folie)

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

Sei $S := \sum_{i=1}^{n} a_i$.

- $m_{NF}(x) \leq 2\lceil S \rceil$

  denn f. alle i ist $\sum B_i + \sum B_{i+1} > 1$.

- $m^*(x) \geq \lceil S \rceil$ klar

$\rightsquigarrow \quad m_{NF}(x) \leq 2 m^*(x)$ □

Schranke kann nicht verbessert werden: Instanz x gegeben durch

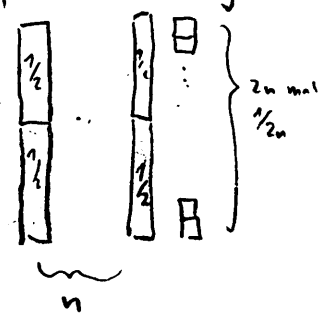
$A = a_1, \ldots, a_{4n}$

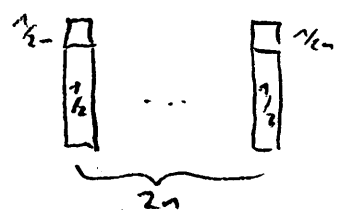
$a_i = \frac{1}{2}$ f. i ungerade

$a_i = \frac{1}{2n}$ f. i gerade.

Optimale Lösung: $m^*(x) = n+1$ $\rightsquigarrow$ Next Fit: $m_{NF}(x) = 2n$

$\frac{m_{NF}(x)}{m^*(x)} = 2 - \frac{2}{n+1}$





Etwas besser: First Fit (Folie)

Performance ratio: $m_{FF}(x) \leq 1.7\, m^*(x) + 2$

(ohne Beweis!)

Noch besser: First Fit Decreasing (off-line!)

$$m_{FFD}(x) \leq \tfrac{3}{2}\, m^*(x) + 1$$

Beweis:

$A = \{ a_i ; a_i > \frac{2}{3} \}$

$B = \{ a_i ; \frac{2}{3} \geq a_i > \frac{1}{2} \}$

$C = \{ a_i ; \frac{1}{2} \geq a_i > \frac{1}{3} \}$

$D = \{ a_i ; a_i \leq \frac{1}{3} \}$

Lösung von First Fit Decreasing:

Fall 1: mindestens ein bin enthält nur Elemente aus D

$\rightsquigarrow$ höchstens ein bin B_k (der letzte) mit $\sum B_k < \frac{2}{3}$

$\rightarrow$ $m_{FFD}(x) \leq \frac{3}{2}\lceil S \rceil + 1$, wie oben $m^* \geq \lceil S \rceil$

$\rightarrow$ $m_{FFD}(x) \leq \frac{3}{2} m^*(x) + 1$ □

Fall 2: jedes bin enthält items aus A, B, C.

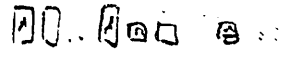
$\rightsquigarrow$ Lösung ist optimal:

Elemente aus D spielen keine Rolle (nicht zugewiesen)

$\rightarrow$ Betrachte Instanz $x' = \{a_1, \ldots, a_n\} \setminus D$

Behauptung: FFD optimal für x'.

- Elemente aus A allein in einem bin
- jedes andere bin höchstens zwei Elemente, nur eines davon aus B



Optimale Lösung:

1 bin f. jedes A
1 bin f. jedes B
C's auf B' bins verteilt
restliche C's je zwei in eigene bin's

} dies ist genau die FFD-Lösung

□

Durch feinere Analyse: (mehr Fälle unterscheiden...)

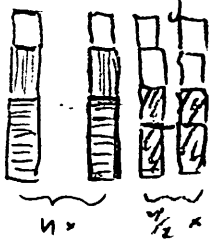
$$m_{FFD}(x) \leq \frac{11}{9} m^*(x) + 4$$

Dies ist optimal. Beispiel:

$5n$ Elemente: (n durch 6 teilbar)

n Elemente	$1/2 + \epsilon$	
n Elemente	$1/4 + 2\epsilon$	
n Elemente	$1/4 + \epsilon$	
$2n$ Elemente	$1/4 - 2\epsilon$	

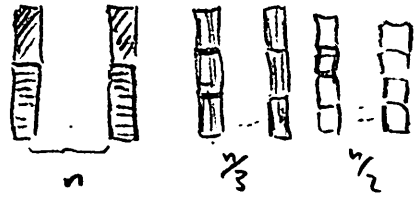
Optimale Lösung:



$m^* = \frac{3}{2}n$

$m^*(x) = \frac{3}{2}n$

FFD:



$m_{FFD}(x) = \frac{11}{6}n$

Sequentieller Algorithmus zum Graphen Färben:

$G = (V, E)$ $V = \{v_1, \dots, v_n\}$ V angeordnet $v_1, \dots, v_n$

Betrachte Knoten in der Reihenfolge $v_1 \dots v_n$:

Färbe v_i mit der minimalen Farbe, die noch kein Nachbar unter den $v_1, \dots, v_{i-1}$ hat

SATZ: $k \leq 1 + \max_i \min(\deg v_i, i-1)$ (Folie)

Definiere G_i = induzierter Subgraph auf $\{v_1, \dots, v_i\}$

k_i = Farben, die der Algo zum Färben von G_i braucht

$\deg_i v$ = Grad von v in G_i

Durch Induktion nach v_i:

$$k_i \leq 1 + \max_{1 \leq j \leq i} \deg_j v_j$$

$i = 1$: klar.

$i+1 \leftarrow i$ zwei Fälle $k_{i+1} = k_i$ · $k_{i+1} = k_i + 1$

obwohl jeder planare Graph 4-färbbar ist, ist es schwer, optimal zu färben

3 färbbar + planar immer noch NP-hart!

→ Algorithmus: A

input: planarer Graph G

versuche, G mit 2 Farben zu färben

Falls nicht möglich, verwende 5 mal so viele Farben.

Falls $m^* \leq 2$: $m_A = m^*$

$m^* \geq 3$ $m_A = 6$, also $m_A \leq 2m^*$

→ performance ratio 2. □

Lokale Suche

- Starte mit einer Anfangslösung,
verbessere diese iterativ, indem schrittweise zu einer besseren, "nahegelegenen" Lösung übergegangen wird.

Nachbarschaftsstruktur:

$\forall$ jedes $y \in S(x)$: $N(x,y) \subseteq S(x)$ (Folie)

Lokale Suche:

$y :=$ initial solution
solange es $y' \in N(x,y)$ mit $m(x,y') > m(x,y)$ gibt (max; oder < bei min)
setze $y := y'$
Gib y zurück (* "lokales Optimum" *)

Anfangslösung: meist trivial, oder von einfachem Algorithmus.

problem: - bessere Nachbarlösung effizient finden!
- wie gut ist lokales Optimum?

MAXIMUM CUT (Folie)

Nachbarschaftsstruktur:

$||V_0| - |V_0'|| = 1$, also auch $||V_1| - |V_1'|| = 1$

→ ein Knoten von V_0 nach V_1 oder umgekehrt.

Fall 1: $k_{i+1} = k_i \leq 1 + \max_{j \leq i} \deg_j v_j \leq 1 + \max_{j \leq i+1} \deg_j v_j$ ✓

Fall 2: $k_{i+1} = k_i + 1$ Dann ist $k_i \leq \deg_{i+1} v_{i+1}$, also

$k_{i+1} = 1 + k_i \leq 1 + \deg_{i+1} v_{i+1} \leq 1 + \max_{j \leq i+1} \deg_j v_j$ ✓

Insbesondere $\boxed{k \leq 1 + \max_i \deg_i v_i}$. ⊗

Offenbar ist $\deg_i v_i \leq \deg v_i$

und $\deg_i v_i \leq i-1$,

also $k \leq 1 + \max_i \min(\deg v_i, i-1)$ □

Korollar: $k \leq \Delta + 1$, wo $\Delta = \max_{v \in V} \deg v$.

Schranke aus Satz wird minimal, wenn $\deg v_1 \geq \deg v_2 \geq \ldots \geq \deg v_n$

→ Decreasing Degree! (Folie)

Beispiel: $V = \{x_1, y_1, \ldots, x_n, y_n\}$ (in dieser Reihenfolge)

$E = \{\{x_i, y_j\} : i \neq j\}$

$\deg v = n-1$ f. alle v ; sortieren ändert Reihenfolge nicht!

⇝ Decreasing Degree braucht n Farben, obwohl 2-Färbbar!

Bessere Reihenfolge: minimiere ⊗ direkt. Smallest Last (Folie)

Satz: smallest Last färbt jeden planaren Graphen mit ≤ 6 Farben

Beweis: Euler: in planarem Graphen ist $|E| \leq 3n - 6$

⇝ es gibt Knoten mit $\deg \leq 5$

induzierter Teilgraph planar:

$\max_i \deg_i v_i \leq 5$ □

Satz $m_N \geq \frac{1}{2} m^*$ (Folie)

Beweis: Sei $m := |E|$. Da $m^* \leq m$, reicht es zu zeigen

$$m_N \geq \frac{1}{2} m.$$

Sei m_0 = # Kanten innerhalb von V_0

m_1 = # Kanten innerhalb von V_1

also $$m = m_0 + m_1 + m_N \qquad (*)$$

Für jeden Knoten v_i

$$m_{0i} := \#\{v \in V_0 \quad \{v, v_i\} \in E\}$$

$$m_{1i} := \#\{v \in V_1 \quad \{v, v_i\} \in E\}$$

Also: $$2m_0 = \sum_{i \in V_0} m_{0i} \qquad 2m_1 = \sum_{i \in V_1} m_{1i}$$

$$m_N = \sum_{i \in V_0} m_{1i} = \sum_{i \in V_1} m_{0i}$$

Jede Nachbarlösung ist von kleinerem Maß als (V_0, V_1)

Also: f. alle $v_i \in V_0$ $\quad m_{0i} \leq m_{1i}$ $\quad m_{0i} - m_{1i} \leq 0$

f. alle $v_i \in V_1$ $\quad m_{1i} \leq m_{0i}$ $\quad m_{1i} - m_{0i} \leq 0$

$$\sum_{v_i \in V_0} m_{0i} - m_{1i} = 2m_0 - m_N \leq 0$$

$$\sum_{v_i \in V_1} m_{1i} - m_{0i} = 2m_1 - m_N \leq 0$$

$+, :2$

$$m_0 + m_1 - m_N \leq 0 \qquad (-)$$

$$m_0 + m_1 + m_N = m \qquad (+)$$

$$2m_N \geq m$$

□

Trade off:

Nachbarschaften gross: lokale Optima wahrscheinlich gut, aber Suche nach besseren Nachbarn teuer

(extrem: $N(y) = S(x)$: lokal opt = global opt. lokale Suche = exhaustive Suche)

Dynamische Programmierung

Programmiertechnik zur Lösung von Optimierungsproblemen.
Anwendbar, wenn zwei Bedingungen erfüllt sind

- Optimale Lösung enthält optimale Lösungen von Teilproblemen
  ~> Problem rekursiv lösbar, unter umständen ineffizient, da Teilprobleme vielfach gelöst werden
- Teilprobleme überlappen sich!

Standardbeispiel: Matrix-Kettenmultiplikation

Aufgabe Multiplikation von Matrizen $A_1, \ldots, A_n$
wo $A_i \in R^{n_{i-1} \times n_i}$

Effizienz kann sehr unterschiedlich sein:

$A: 50 \times 10$, $B: 10 \times 20$ $C: 20 \times 5$

$(AB)C$: $50 \cdot 10 \cdot 20 + 50 \cdot 20 \cdot 5 = 15.000$

$A(BC)$: $10 \cdot 20 \cdot 5 + 50 \cdot 10 \cdot 5 = 3.500$

Definiere $A_{i..j} := A_i \cdot \ldots \cdot A_j$

Optimale Lösung hat letzte Multiplikation $A_{1..k} \cdot A_{k+1..n}$
→ Teillösung für $A_1, \ldots, A_k$ $A_{k+1}, \ldots, A_n$ optimal.

$m(i,j) :=$ min. # Multiplikationen zum Ausrechnen von $A_{i..j}$

$$m(i,i) = 0$$

$$m(i,j) = \min_{i \le k < j} \{ m(i,k) + m(k+1,j) + n_{i-1} n_k n_j \}$$

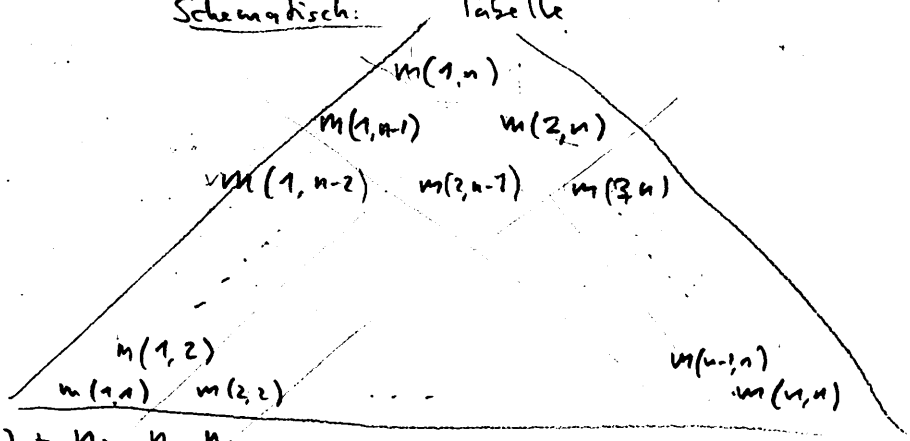
$s(i,j) :=$ ein k sodass $m(i,k) + m(k+1,j) + n_{i-1} n_k n_j = m(i,j)$.

Berechnung mit Hilfe der rekursiven Definition
~> exponentielle Zeit

aber: nur $O(n^2)$ viele Teilprobleme
→ bottom-up Berechnung mit Tabelle

```
for i := 1 to n
    m(i,i) := 0
for l := 1 to n-1
  for i := 1 to n-l+1
    j := i+l-1
    m(i,j) := ∞
    for k := i to j-1
      q := m(i,k) + m(k+1,j) + n_{i-1} n_k n_j
      if q < m(i,j) then
          m(i,j) := q
          s(i,j) := k
```

Schematisch: Tabelle



von unten nach oben gefüllt

→ Laufzeit nur $O(n^3)$!

Dynamische Programmierung für MAXIMUM KNAPSACK

Instanz: $X = \{x_1, \dots, x_n\}$ f. $x_i \in X$: p_i Wert a_i Gewicht
b Kapazität

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

Maß: $\sum_{x_i \in Y} p_i$

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

Teilproblem mit Parameter $k \leq n$ und $p \leq P$:

- Lösung: $M \subseteq \{x_1, \dots, x_k\}$ mit $\sum_{x_i \in M} p_i = p$ und $\sum_{x_i \in M} a_i \leq b$
- Maß: $\sum_{x_i \in M} a_i$ Ziel: min

$M(k,p)$: eine optimale Lösung dieses Teilproblems!

$S(k,p) := \sum_{x_i \in M(k,p)} a_i$

berechnet durch Dynamische Programmierung

$M(1,0) = \emptyset$

$M(1,p_1) = \{x_1\}$ $\qquad$ $M(1,p)$ undef. für $p \notin \{0, p_1\}$

$$M(k,p) = \begin{cases} M(k-1, p-p_k) \cup \{x_k\} & \text{falls } p_k \le p \text{ und } M(k-1,p-p_k) \text{ def.} \\ & s(k-1,p-p_k) + a_k \le b \\ & s(k-1,p-p_k) + a_k \le s(k-1,p) \\ M(k-1,p) & \text{sonst} \end{cases}$$

Damit: dynamischer Programm (FOLIE)

→ suche maximales p^* so dass $M(n,p^*)$ definiert ist, dann ist $M(n,p^*)$ optimale Lösung.

Laufzeit $O(n \cdot P)$ ist exponentiell in der Eingabegrösse!
aber: polynomial in den Eingabewerten

→ Pseudo-Polynomialzeit (= poly, wenn Zahlen unär repräsentiert werden)

Approximationsschema: Idee:

- Dynamisches Programm ist effizient, falls $|P| = n^{O(1)}$
  d.h. alle $|p_i| = n^{O(1)}$.
  → skaliere Werte p_i, so dass dies erfüllt ist
  (dadurch wird Ergebnis ungenau)

→ Approximationsschema. (FOLIE)

Analyse:

(1) performance ratio:

Sei Y^* optimale Lösung.

F. alle i: $\quad p_i - 2^t p_i' \le 2^t$, alg. ist

$$m(Y^*) - 2^t m'(Y^*) \le n \cdot 2^t \qquad (*)$$

Sei Y_r die vom AS mit inp.t r gefundene Lösung

$$m(Y_r) \geq 2^t \cdot m'(Y_r) \geq 2^t m'(Y^*)$$

Also ist

$$m^* - m_{AS}(r) = m(Y^*) - m(Y_r) \leq m(Y^*) - 2^t m'(Y^*) \leq n 2^t .$$

Ausserdem ist $m^* \geq p_{max}$.

$$\frac{m^* - m_{AS}(r)}{m^*} \leq \frac{n 2^t}{m^*} \leq \frac{n 2^t}{p_{max}}$$

$$1 - \frac{m_{AS}}{m^*} \leq \frac{n 2^t}{p_{max}}$$

$$\frac{m_{AS}}{m^*} \geq 1 - \frac{n 2^t}{p_{max}}$$

$$m^* \leq \frac{p_{max}}{p_{max} - n 2^t} m_{AS}(r)$$

Einsetzen. $2^t \leq \frac{r-1}{r} \cdot \frac{p_{max}}{n}$

$$m^* \leq \frac{p_{max}}{p_{max} - \frac{r-1}{r} p_{max}} m_{AS}(r) = r \, m_{AS}(r)$$ □

(2) Laufzeit.

ist $O\left(n \cdot \sum_{i=1}^{n} p_i'\right)$

$$n \cdot \sum_{i=1}^{n} p_i' \leq n \cdot \sum_{i=1}^{n} \frac{p_i}{2^t} \leq n \cdot \sum_{i=1}^{n} \frac{p_{max}}{2^t}$$

$$\leq n \cdot \sum_{i=1}^{n} \frac{p_{max}}{\left(\frac{r-1}{r}\right) \frac{p_{max}}{n}} \leq n \cdot \frac{r}{r-1} \sum_{i=1}^{n} n$$

$$= O\left(\frac{r}{r-1} n^3\right)$$ □