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 $\ell : \{c_1, \ldots, c_n\} \to \mathbb{Q}$ 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 $T = \{t_1, \ldots, t_n\}$ von Aufträgen,	
	für jedes $t_i \in T$:	Dauer $\ell_i \in \mathbb{N}$
	Anzahl $p \in \mathbb{N}$ von Prozessoren	
Lösung:	schedule:	Funktion $f : T \to \{1, \ldots, p\}$
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*

$k := 1\,;\ B_1 := \emptyset$

$\texttt{for}\ i := 1\ \texttt{to}\ n\ \texttt{do}$

$\quad \texttt{if}\ \Sigma B_k + a_i \leq 1$

$\quad\quad \texttt{then}\ \ B_k := B_k \cup \{a_i\}$

$\quad\quad \texttt{else}\ \ k := k + 1\,;\ B_k := \{a_i\}$

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 $m_{\mathrm{FFD}}(x) \leq \dfrac{3}{2} m^*(x) + 1$.

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}:

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

Optimale Lösung $x_1^*, \ldots, x_n^* \to$ 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}:

$$\begin{aligned} &\text{maximize} && \sum_{\{v_i,v_j\}\in E} y_{i,j} \\ &\text{subject to} && \sum_{j:\{v_i,v_j\}\in E} y_{i,j} \le c_i \quad (C_i) \\ &&& y_{i,j} \ge 0 \end{aligned}$$

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 $m_{PD}/m^* \le 2$.

Dynamic Programming für Maximum Knapsack

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

for $p := 0$ to P

    $S(1,p) := A + 1$

$M(1,0) := \emptyset$; $S(1,0) := 0$;

$M(1,p_1) := \{x_1\}$; $S(1,p_1) := a_1$

for $k := 1$ to n

    for $p := 1$ to P

        if $p_k \le p$ and $M(k-1, p-p_k)$ defined

            and $S(k-1, p-p_k) + a_k \le b$

            and $S(k-1, p-p_k) + a_k \le S(k-1, p)$

        then

            $M(k,p) := M(k-1, p-p_k) \cup \{x_k\}$

            $S(k,p) := S(k-1, p-p_k) + a_k$

        else

            $M(k,p) := M(k-1, p)$

            $S(k,p) := S(k-1, p)$

$p^* := \max\{p ; M(n,p) \text{ defined}\}$

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 $m^*/m_{AS}(r) \leq r$.