Universität Bonn

Mitschrift zur Vorlesung

Randomisierte und Approximative Algorithmen

gehalten von

Prof. Dr. Thomas Kesselheim, Prof. Dr. Heiko Röglin

geTEXed von

Daniel Emse

Winter 2025/26

Anmerkungen und Korrekturen

Falls du Anmerkungen/Korrekturen hast, melde dich bei mir (`daniel.emse@uni-bonn.de`).

Inhaltsverzeichnis

Vorlesung

Teil I

Randomisierte und Approximative Algorithmen

Kapitel 1

Einleitung

Definition 1.1: Optimierungsproblem, Approximationsalgorithmus

Ein *Optimierungsproblem* Π besteht aus

- einer Meinge von Instanzen $\mathcal{I}_\Pi$
- für jede Instanz $I \in \mathcal{I}_\Pi$
  - einer Menge $\mathcal{S}_I$ von Lösungen
  - einer Zielfunktion $f_I : \mathcal{S}_I \to \mathbb{R}_{\geq 0}$.

Ein *Approximationsalgorithmus* nimmt I als Eingabe und berechnet $A(I) \in \mathcal{S}_I$ (in polynomieller Zeit).
Sei $w_A(I) = f_I(A(I))$. Ein Approximationsalgorithmus erreicht Approximationsfaktor r, falls

$$w_A(I) \leq r \cdot \mathrm{OPT}(I) \quad \forall I \in \mathcal{I}_\Pi \qquad \text{(für Minimierungsprobleme)},$$

bzw.

$$w_A(I) \geq \frac{1}{r} \cdot \mathrm{OPT}(I) \quad \forall I \in \mathcal{I}_\Pi \qquad \text{(für Maximierungsprobleme)}.$$

Kapitel 2

Greedy-Algorithmen

2.1 Vertex Cover

[14.10.2025, Vorlesung 1]

Vertex CoverVertexCover

Input: Ungerichteter Graph $G = (V, E)$
Output: Alle $V' \subseteq V$, sodass $\forall e = (u, v) \in E$ gilt $u \in V'$ oder $v \in V'$
Objective: Minimiere $|V'|$

Definition 2.1: Matching

Ein *Matching* $M \subseteq E$ eines Graphen $G = (V, E)$ ist eine Teilmenge der Kanten, sodass es keinen Knoten gibt, der zu mehr als einer Kante aus M inzident ist.
Es heißt *inklusionsmaximal*, wenn für alle Kanten $e \in E \setminus M$ gilt: $M \cup \{e\}$ ist kein Matching.

Algorithm 1 MATCHING-VC

Input: ungerichteter Graph $G = (V, E)$
Output: Vertex Cover $V' \subseteq V$ mit $|V'| \leq 2 \cdot \mathrm{OPT}(G)$

1: ▷ *Berechne inklusionsmaximales Matching $M \subseteq E$:* ◁
2: $M := \emptyset$
3: **for all** $e = \{u, v\} \in E$ **do**
4: **if** weder u noch v ist inzident zu einer Kante in M **then**
5: $M := M \cup \{e\}$
6: $V(M) := \{v \in V \mid v \text{ ist inzident zu einer Kante in } M\}$
7: **return** $V(M)$

Theorem 2.2: Matching-VC

Matching-VC hat Laufzeit $\mathcal{O}(|V| + |E|)$ für Graphen in Adjazenzlistendarstellung. Er berechnet ein Vertex Cover der Größe höchstens $2 \cdot \mathrm{OPT}(G)$, wobei $\mathrm{OPT}(G)$ die Größe des kleinsten Vertex Covers ist.

Beweis. **Laufzeit:** In Schritt 1 wird jede Kante genau einmal betrachtet, $\mathcal{O}(|V| + |E|)$.

In Schritt 2 wird jeder Knoten höchstens einmal betrachtet, $\mathcal{O}(|V|)$. Gemeinsam ergibt sich $\mathcal{O}(|V| + |E|)$.

Korrektheit: Angenommen, $e = (u, v) \in E$ wurde nicht von $V(M)$ abgedeckt. Das heißt, es wären weder $u \in V(M)$ noch $v \in V(M)$. Dann wäre $M \cup \{e\}$ ein Matching und e würde in Schritt 1 zu M hinzugefügt werden. Widerspruch zur Inklusionsmaximalität von M. Also ist $V(M)$ ein Vertex Cover.

Approximation: Sei V^* ein minimales Vertex Cover. V^* muss alle Kanten $e \in E$ abdecken, also auch alle $e \in M$.

Außerdem haben in M keine zwei Kanten denselben Endpunkt. Deshalb ist $|M| \leq |V^*|$.

Für unsere Lösung gilt $|V(M)| = 2 \cdot |M| \leq 2 \cdot \mathrm{OPT}(G)$.

□

2.2 Set Cover

Set CoverSetCover

Input: Menge S mit n Elementen, n Teilmengen $S_1, \ldots, S_m \subseteq S$ mit $\bigcup_{i=1}^{m} S_i = S$, Ko
$c_1, \ldots, c_m \in \mathbb{N}$.
Output: Alle $A \subseteq \{1, \ldots, m\}$ mit $\bigcup_{i \in A} S_i = S$
Objective: Minimiere $\sum_{i \in A} c_i$

Algorithm 2 Greedy-SC

Input: Grundmenge S mit n Elementen, n Teilmengen $S_1, \ldots, S_m \subseteq S$ mit $\bigcup_{i=1}^{m} S_i = S$, Kosten $c_1, \ldots, c_m \in \mathbb{N}$.
Output: $A \subseteq \{1, \ldots, m\}$ mit $\bigcup_{i \in A} S_i = S$ und $\sum_{i \in A} c_i$ minimal

1: $A := \emptyset, C := \emptyset$
2: **while** $C \neq S$ **do**
3: Wähle S_i mit minimalen relativen Kosten $r_i(C) = \frac{c_i}{|S_i \setminus C|}$
4: Setze $\mathrm{price}(x) := r_i(C)$ für alle $x \in S_i \setminus C$
5: $A := A \cup \{i\}$
6: $C := C \cup S_i$
7: **return** A

Theorem 2.3: Greedy-SC

Greedy-SC ist ein H_n-Approximationsalgorithmus für Set Cover, wobei

$$H_n = \sum_{i=1}^{n} \frac{1}{i}$$

die n-te harmonische Zahl ist.

Untere Schranke für Greedy-SC: Sei $\varepsilon > 0$ sehr klein. Betrachte n Knoten und $m = n + 1$ Mengen, wobei $S_i = \{i\}$ mit Kosten $\frac{1}{n+1-i}$ ist (für $i \in S$) und $S_{n+1} = S$ mit Kosten $1 + \varepsilon$. Dann wählt Greedy-SC wählt alle Einzelnen, OPT die gesamte Menge.

[14.10.2025, Vorlesung 1]
[15.10.2025, Vorlesung 2]

Beweis. Beobachtung: Es gilt

$$c(A) = \sum_{x \in S} \mathrm{price}(x).$$

Sei $x_1, \ldots, x_n$ die Reihenfolge der Elemente in S, wie sie vom Greedy-SC abgedeckt werden (Beim Abdecken von mehreren Elementen in gleicher Iteration wähle beliebige Reihenfolge).

Lemma 2.4: Preis in Greedy-SC

Für alle $k \in \{1, \ldots, n\}$ gilt

$$\mathrm{price}(x_k) \leq \frac{\mathrm{OPT}}{n - k + 1}.$$

Beweis: Lemma **??**. Betrachte ein beliebiges $k \in \{1, \ldots, n\}$. Sei $i \in \{1, \ldots, m\}$ der Index der Menge S_i, mit der x_k erstmalig abgedeckt wird. Bezeichne A die Auswahl der Mengen, *bevor* x_k abgedeckt wird.

Sei $C := \bigcup_{i \in A} S_i$. Es gilt $|C| \leq k-1$. Eine optimale Auswahl A^* von Mengen deckt ganz S ab, insbesondere auch $S \setminus C$, wobei $|S \setminus C| \geq n-k+1$.

Es gilt

$$\begin{aligned}\mathrm{OPT} = \sum_{j \in A^*} c_j &\geq \sum_{\substack{j \in A^* \\ S_j \setminus C \neq \emptyset}} c_j = \sum_{\substack{j \in A^* \\ S_j \setminus C \neq \emptyset}} |S_j \setminus C| \underbrace{\frac{c_j}{|S_j \setminus C|}}_{= r_j(C) \geq r_i(C)} \\ &\geq r_i(C) \underbrace{\sum_{\substack{j \in A^* \\ S_j \setminus C \neq \emptyset}} |S_j \setminus C|}_{\geq |S \setminus C|} \geq \underbrace{r_i(C)}_{= \mathrm{price}(x_k)} \cdot (n-k+1)\end{aligned}$$

wobei $r_j(C) \geq r_i(C)$ wegen der Greedy-Auswahl und

$$\sum_{\substack{j \in A^* \\ S_j \setminus C \neq \emptyset}} |S_j \setminus C| \geq |S \setminus C|,$$

weil A^* ganz $S \setminus C$ abdeckt. □

Nun gilt mit Lemma **??**

$$c(A) = \sum_{k=1}^{n} \mathrm{price}(x_k) \leq \sum_{k=1}^{n} \frac{\mathrm{OPT}}{n-k+1} = \mathrm{OPT} \sum_{i=1}^{n} \frac{1}{i} = \mathrm{OPT} \cdot H_n$$

und Satz **??** ist gezeigt. □

2.3 Scheduling auf identischen Maschinen

Scheduling auf identischen MaschinenSchedulingIdenticalMachines

Input: Menge $J = \{1, \ldots, n\}$ von Jobs
$p_j > 0$ für jeden Job $j \in J$
Output: Schedule $\pi : J \to M$, Last von M

$C(\pi) = \max_{i \in M} L_i(\pi)$
Objective: Finde Schedule π mit kleinstem

Algorithm 3 LeastLoaded
Input: Siehe **??**
Output: Siehe **??**
1: Betrachte Jobs in Reihenfolge $1, \ldots, n$
2: Weise jedem Job der Maschine mit bislang kleinster Last zu

Theorem 2.5: LeastLoaded

LeastLoaded ist ein $(2 - \frac{1}{m})$-Approximationsalgorithmus für Scheduling auf identischen Maschinen.

Beweis. Sei π^* ein optimaler Schedule. Dann ist

$$C(\pi^*) \geq \frac{1}{m} \sum_{j \in J} p_j,$$

denn

$$C(\pi^*) = \max_{i \in M} \sum_{\substack{j \in J \\ \pi(j)=i}} p_j \geq \frac{1}{m} \sum_{i \in M} \sum_{\substack{j \in J \\ \pi(j)=i}} = \frac{1}{m} \sum_{j \in J} p_j.$$

Außerdem gilt $C(\pi^*) \geq \max_{j \in J} p_j$.

Sei π der Schedule, der LeastLoaded berechnet. Sei i die Maschine mit größter Last. Sei j der Job, der zuletzt zu i hinzugefügt wurde. Zu diesem Zeitpunkt war Maschine i diejenige mit der geringsten Last. Ihre Last war höchstens $\frac{1}{m} \sum_{k=1}^{j-1} p_k$. Es gilt

$$C(\pi) = L_i(\pi) \leq \frac{1}{m} \underbrace{\sum_{k=1}^{j-1} p_k}_{\leq \sum_{k \in J \setminus \{j\}} p_k} + p_j \leq \underbrace{\frac{1}{m} \sum_{k \in J} p_k}_{\leq C(\pi^*)} + \left(1 - \frac{1}{m}\right) \underbrace{p_j}_{\leq C(\pi^*)} \leq \left(2 - \frac{1}{m}\right) C(\pi^*)$$

und die Abschätzung ist gezeigt. □

$2 - \frac{1}{m}$ ist eine untere Schranke für LeastLoaded.

Beweis. Sei $n = m(m-1)+1$ und $p_1 = \cdots = p_{m(m-1)} = 1$ und $p_{m(m-1)+1} = m$.

LeastLoaded führt auf jeder Maschine $m-1$ Jobs der Größe 1 durch. Auf einer Maschine wird zusätzlich ein Job der Größe m ausgeführt. Somit ist der Makespan $m - 1 + m = 2m - 1$.

Eine optimale Belegung führt auf $m-1$ Maschinen je m Jobs der Größe 1 aus und auf der letzten Maschine einen Job der Größe m, wir haben also einen Makespan von m, und der Faktor stimmt. □

Algorithm 4 LongestProcessingTime (LPT)

Input: Siehe **??**
Output: Siehe **??**

1: Sortiere Jobs so, dass $p_1 \geq p_2 \geq \cdots \geq p_n$.
2: Führe LeastLoaded auf den sortierten Jobs aus.

Theorem 2.6: LPT

LongestProcessingTime ist ein $\frac{4}{3}$-Approximationsalgorithmus.

Beweis. Angenommen, es gäbe eine Instanz, sodass LPT einen Makespan von mehr als $\frac{4}{3} \cdot \mathrm{OPT}$ erreicht. Betrachte unter diesen Instanzen eine mit geringster Anzahl Jobs.

Sei i eine Maschine mit höchster Last in LPT-Schedule π. Sei j der letzte Job, der Maschine i zugewiesen wird. Es ist $j = n$, da sonst $p_1, \ldots, p_j$ ein Gegenbeispiel mit weniger Jobs wäre. Es gilt

$$C(\pi) = L_i(\pi) \leq \frac{1}{m} \sum_{k=1}^{n-1} p_k + p_n \leq \mathrm{OPT} + p_n.$$

Aus $C(\pi) > \frac{4}{3}\,\mathrm{OPT}$ folgt $p_n > \frac{1}{3}\,\mathrm{OPT}$. Da $p_1 \geq \cdots \geq p_n$ folgt $p_j > \frac{1}{3}\,\mathrm{OPT}$ für alle $j \in J$. Also erhält im optimalen Schedule jede Maschine höchstens zwei Jobs.

Sei k die Anzahl Jobs mit Größe $\frac{2}{3}\,\mathrm{OPT}$. Im optimalen Schedule sind k Jobs alleine auf einer Maschine und $n - k$ Jobs höchstens zu zweit.

Fall 1: $k = n$**.** Also ist $n \leq m$, Job n wird also einer leeren Maschine zugewiesen.

Fall 2: $k < n$**.** Zum Zeitpunkt bevor Job n hinzugefügt wird, hat LPT mindestens eine Maschine mit höchstens einem Job und dieser hat Größe höchstens $\frac{2}{3}$ OPT. Wegen $p_n \leq \frac{2}{3}$ OPT ist die Last dieser Maschine im Anschluss höchstens $\frac{4}{3}$ OPT.

Beides ist ein Widerspruch zur Annahme. ↯ □

$\frac{4}{3}$ ist eine untere Schranke für LongestProcessingTime.

Beweis. Übungsaufgabe. □

Wir werden später sehen, dass für alle $\varepsilon > 0$ ein $(1+\varepsilon)$-Approximationsalgorithmus existiert.

[15.10.2025, Vorlesung 2]
[21.10.2025, Vorlesung 3]

2.4 Rucksackproblem

Rucksackproblem (Knapsack Problem, KP)knapsackProblem

Input: Kapazität $t \in \mathbb{N}$, Nutzen $p_1, \ldots, p_n \in \mathbb{N}$
Output: Teilmenge $A \subseteq \{1, \ldots, n\}$ mit $w(A) :=$
Objective: Maximiere $p(A) := \sum_{i \in A} p_i$

Algorithm 5 Greedy-KP

Input: Siehe **??**
Output: Siehe **??**

1: Sortiere Objekte nach Effizienz, sodass $\frac{p_1}{w_1} \geq \cdots \geq \frac{p_n}{w_n}$
2: $B := \emptyset$, $i := 1$
3: **while** $i \leq n$ **and** $w(B) + w_i \leq t$ **do**
4: $\quad B := B \cup \{i\}$
5: $\quad i := i + 1$
6: Sei i^* ein Objekt mit dem größten Nutzen
7: **if** $p(B) < p_{i^*}$ **then**
8: $\quad A := \{i^*\}$
9: **else** $A := B$
10: **return** A

Theorem 2.7: Greedy-KP

Greedy-KP ist ein $\frac{1}{2}$-Approximationsalgorithmus.

Beweis. Sei $B := \{1, \ldots, i\}$ die im Algorithmus berechnete Menge. Falls $i = n$, gibt es nichts zu zeigen.

Falls $i < n$, gilt $w(B) + w_{i+1} > t$.

Behauptung: Es gilt $p(B) + p_{i+1} \geq$ OPT:

Angenommen, OPT $> p(B) + p_{i+1}$. Dann gibt es eine Menge $A' \subseteq \{1, \ldots, n\}$, für die gilt, dass $w(A') \leq t < w(B) + w_{i+1}$ und $p(A') =$ OPT $> p(B) + p_{i+1}$.

Betrachte $J := \{1, \ldots, i+1\} \setminus A'$ und $J' := A' \setminus \{1, \ldots, i+1\}$. Nun gilt

$$p(B) + p_{i+1} < p(A') = (p(B) + p_{i+1}) + p(J') - p(J) \implies p(J') > p(J)$$

und analog

$$w(B) + w_{i+1} > w(A') = (w(B) + w_{i+1}) + w(J') - w(J) \implies w(J') < w(J).$$

Sei $r = \frac{p_{i+1}}{w_{i+1}}$, dann

$$p(J) \geq r \cdot w(J), \qquad p(J') \leq r \cdot w(J').$$

Zusammen ergibt sich

$$p(J) \geq r \cdot w(J) > r \cdot w(J') \geq p(J') > p(J). \qquad \lightning$$

Damit ist die Behauptung gezeigt und es gilt $p(B) + p_{i+1} \geq \mathrm{OPT}$.

Somit gilt

$$p(A) = \max\{p(B), p_{i^*}\} \geq \frac{1}{2}(p(B) + p_{i^*}) \geq \frac{1}{2}(p(B) + p_{i+1}) \geq \frac{1}{2}\,\mathrm{OPT}$$

und die Aussage ist gezeigt. □

Kapitel 3

Runden und dynamische Programmierung

Definition 3.1: Approximationsschema, PTAS, FPATS

Ein *Approximationsschema* A für ein Optimierungsproblem Π ist ein Algorithmus, der zu jeder Eingabe (I, ε) mit $I \in \mathcal{I}_\Pi, \varepsilon > 0$ eine Lösung $A(I, \varepsilon) \in \mathcal{S}_I$ berechnet und es gilt

$$f(A(I,\varepsilon)) \leq (1+\varepsilon)\,\mathrm{OPT}(I) \qquad \text{bei Minimierungsproblemen, bzw.}$$
$$f(A(I,\varepsilon)) \geq (1-\varepsilon)\,\mathrm{OPT}(I) \qquad \text{bei Maximierungsproblemen.}$$

A heißt *polynomielles Approximationsschema* (PTAS), wenn die Laufzeit für jedes ε polynomiell in $|I|$ beschränkt ist.
A heißt *voll-polynomielles Approximationsschema* (FPTAS), wenn die Laufzeit polynomiell in $\frac{1}{\varepsilon}$ und $|I|$ ist.

- $\Theta(|I|^{1/\varepsilon})$ ist PTAS, aber kein FPTAS
- $\Theta\left(\frac{|I|^{100}}{\varepsilon^{10}}\right)$ ist ein FPTAS

3.1 Rucksackproblem

3.1.1 Dynamische Programmierung

Sei $W(I,p)$ *geringste* Gewicht unter allen Teilmengen von $\{1,\dots,i\}$, die Nutzen mindestens p erreichen, bzw. ∞, wenn das nicht möglich ist.

Falls $I \subseteq \{1,\dots,i\}$ gilt: Falls $\sum_{i\in I} p_i \geq p$, dann ist $\sum_{i\in I} w_i \geq W(I,p)$.

Es gilt

$$W(0,p) = \begin{cases} 0 & p \leq 0 \\ \infty & p > 0, \end{cases}$$

$$W(1,p) = \begin{cases} 0 & p \leq 0 \\ w_1 & 0 < p \leq p_1 \\ \infty & p > p_1. \end{cases}$$

Sei $P := \max_{\{1,\dots,n\}} p_i$. Uns interessieren $W(i,p)$ für $i = 0,\dots,n$ und $p = 0,\dots,nP$.

Rekursionsformel für $W(i,p)$:

Sei $I \subseteq \{1,\dots,i\}$ diejenige Teilmenge mit minimalem Gewicht unter allen mit $\sum_{j\in I} p_j \geq p$.

Fall 1: $i \notin I$. Dann gilt $W(i,p) = W(i-1,p)$.

Fall 2: $i \in I$. Dann betrachte $I \setminus \{i\}$. Es ist $W(I\setminus\{i\}) = W(i,p) - w(i)$ und $p(I\setminus\{i\}) \geq p - p_i$. Daraus folgt

$$W(i-1, p-p_i) \leq W(i,p) - w_i.$$

Umgekehrt sei $I' \subseteq \{1, \dots, i-1\}$ diejenige Menge, die $W(i-1, p-p_i)$ liefert. Nun gilt

$$W(I' \cup \{i\}) = W(i-1, p-p_i) + w_i, \qquad p(I' \cup \{i\}) = p(I') + p_i \geq p.$$

Daraus folgt, dass $W(i,p) \leq W(i-1, p-p_i) + w_i$.

Umstellen liefert

$$W(i,p) = W(i-1, p-p_i) + w_i.$$

Insgesamt ergibt sich die Rekursion

$$W(i,p) = \min\{W(i-1,p), W(i-1,p-p_i) + w_i\}, \qquad W(0,p) = \begin{cases} \infty & p > 0 \\ 0 & p \leq 0. \end{cases}$$

Algorithm 6 DynKP

Input: Siehe **??**
Output: Siehe **??**

1: $P := \max_{i \in \{1,\dots,n\}} p_i$
2: **for** $i = 1$ **to** n **do**
3: **for** $p = 1$ **to** nP **do**
4: $W(i,p) = \min\{W(i-1,p), W(i-1,p-p_i) + w_i\}$
5: **return** maximales $p \in \{1, \dots, nP\}$ mit $W(n,p) \leq t$.

Theorem 3.2: DynKP

DynKP liefert in Zeit $\Theta(n^2 P)$ den maximal erreichbaren Nutzen.

3.1.2 Approximationsschema

Algorithm 7 FPTAS-KP

Input: Siehe **??**, $\varepsilon > 0$
Output: Siehe **??**

1: $P := \max_{i \in 1,\dots,n} p_i$
2: $K := \frac{\varepsilon P}{n}$
3: **for** $i = 1$ **to** n **do**
4: $p'_i = \lfloor \frac{p_i}{K} \rfloor$
5: Benutze DynKP auf $p'_1, \dots, p'_n, w_1, \dots, w_n, t$

Theorem 3.3: FPTAS-KP

FPTAS-KP hat eine Laufzeit von $\mathcal{O}(\frac{n^3}{\varepsilon})$ und berechnet eine $1-\varepsilon$-Approximation.

Beweis.

Laufzeit: DynKP braucht Zeit $\Theta(n^2 P')$, wobei $P' = \max_{i \in \{1,\dots,n\}} p'_i$. Es gilt für alle i:

$$p'_i = \left\lfloor \frac{p_i}{K} \right\rfloor \leq \left\lfloor \frac{P}{K} \right\rfloor = \left\lfloor \frac{n}{\varepsilon} \right\rfloor \leq \frac{n}{\varepsilon},$$

also ist auch $P' \leq \frac{n}{\varepsilon}$.

Approximationsfaktor: Sei I' eine optimale Lösung auf $p'_1, \dots, p'_n$ und I eine optimale Lösung auf $p_1, \dots, p_n$.

I' ist auch eine optimale Lösung auf $\{p_i^*\} := \{K \cdot p_i'\}$. Für $n = 4, P = 50, \varepsilon = \frac{4}{5}, K = \frac{\varepsilon P}{n} = 10$. Für $p_1 = 33$ erhalten wir $p_1' = 3$ bzw. $p_1^* = 30$.

Es gilt $p_i^* = K \cdot \lfloor \frac{p_i}{K} \rfloor \geq K \cdot (\frac{p_i}{K} - 1) = p_i - K$ und $p_i^* \leq K \cdot \frac{p_i}{K} = p_i$, also $p_i^* \in [p_i - K, p_i]$.

Für alle $J \subseteq \{1, \ldots, n\}$ gilt

$$p^*(J) := \sum_{i \in J} p_i^* \leq \sum_{i \in J} p_i = p(J)$$

und

$$p^*(J) = \sum_{i \in J} p_i^* \geq \sum_{i \in J} (p_i - K) = \sum_{i \in J} p_i - |J| \cdot K \geq p(J) - nK.$$

Damit ist $p(I') \geq p^*(I') \geq p^*(I) \geq p(I) - nK$, da I' optimal hinsichtlich p^* ist.

Weiterhin ist $p(I) \geq P$, also gilt

$$\frac{p(I')}{\mathrm{OPT}} \geq \frac{p(I) - nK}{p(I)} \geq 1 - \frac{nK}{P} = 1 - \varepsilon$$

und wir erhalten $1 - \varepsilon$ als Approximationsfaktor.

□

[21.10.2025, Vorlesung 3]
[22.10.2025, Vorlesung 4]

3.2 Scheduling auf identischen Maschinen

Erster Ansatz für Approximationsschema: Wir nennen einen Job $j \in J$ *klein*, falls

$$p_j \leq \varepsilon \cdot \frac{1}{m} \sum_{k=1}^{n} p_k,$$

sonst heißt er *groß*. Wir betrachten folgenden Algorithmus:

Algorithm 8 GrossKleinScheduling

Input: Siehe **??**, $\varepsilon > 0$
Output: Siehe **??**

1: Berechne optimalen Schedule auf großen Jobs ▷ *Bspw. durch Ausprobieren aller Schedules*
2: Ergänze kleine Jobs mit LeastLoaded

Theorem 3.4: GroßKleinScheduling

GrossKleinScheduling ist ein PTAS mit Laufzeit $m^{\frac{m}{\varepsilon}}$.

Beweis. **Laufzeit:** Für Algorithmus **??** gibt es höchstens $\frac{m}{\varepsilon}$ große Jobs, denn jeder ist mindestens $\varepsilon \frac{1}{m} \sum_{k=1}^{n} p_k$ groß. Damit gibt es maximal $m^{\frac{m}{\varepsilon}}$ mögliche Schedules. Damit hat Algorithmus **??** polynomielle Laufzeit für jedes $\varepsilon > 0$, wenn m konstant ist.

Approximationsfaktor: Betrachte die Maschine mit der größten Last.

Fall 1: Diese Maschine erhält nur große Jobs in unserem Schedule. Dann ist der Makespan durch die kleinen Jobs unverändert. Unser Schedule ist optimal auf den großen Jobs, und somit auch optimal auf allen Jobs.

Fall 2: Diese Maschine erhält auch einen kleinen Job. Im Moment der Zuweisung des letzten solchen Jobs durch LeastLoaded ist die Last höchstens die $\frac{1}{m}\sum_{k=1}^{n} p_k$. Der letzte Job ist ein kleiner Job, somit haben wir am Ende eine Gesamtlast von höchstens

$$(1+\varepsilon)\frac{1}{m}\sum_{k=1}^{n} p_k \leq (1+\varepsilon)\,\mathrm{OPT}\,.$$

□

3.2.1 Bin-Packing mit konstant vielen Objektgrößen

Bin-PackingbinPacking

Input: Objektgrößen $s_1,\ldots,s_n \in \mathbb{N}$, Kapazität $T \in \mathbb{N}$
Output: Partitionen $C_1,\ldots,C_\ell$ der Menge $\{1,\ldots,n\}$ mit $\sum_{j\in C_i} s_j \leq T$ für alle i
Objective: Minimiere ℓ

Spezialfall: Wir wählen als Eingabe $(s_1,\ldots,s_z) \in \mathbb{N}^z, (n_1,\ldots,n_z) \in \mathbb{N}^z, T \in \mathbb{N}$. Dabei haben wir jeweils n_i Objekte der Größe s_i.

Wir bezeichnen mit $\mathrm{BINS}(i_1,\ldots,i_z)$ den optimalen Zielfunktionswert, wenn es $i_j \leq n_j$ Objekte der Größe s_j gibt. Uns interessiert $\mathrm{BINS}(n_1,\ldots,n_z)$. Berechne aller Werte $\mathrm{BINS}(i_1,\ldots,i_z)$ für $(i_1,\ldots,i_z) \in \mathbb{N}_0^z$ mit $i_j \leq n_j$. Wenn z konstant ist, gibt es höchstens $(n+1)^z \in \mathcal{O}(n^z)$ viele solche Vektoren.

Definiere

$$C := \left\{ (i_1,\ldots,i_z) \in \mathbb{N}_0^z \;\middle|\; \sum_{j=1}^{z} i_j s_j \leq T \right\} \setminus \{(0,\ldots,0)\}.$$

Wir bemerken:

- $\mathrm{BINS}(0,\ldots,0) = 0$
- $\mathrm{BINS}(i_1,\ldots,i_z) = 1$ für $(i_1,\ldots,i_z \in C)$

Für die allgemeine Größe von $\mathrm{BINS}(i_1,\ldots,i_z)$ betrachte ein optimales Packing. Der erste Bin erhalte $(c_1,\ldots,c_z) \in C$ viele Objekte. So erhalten die übrigen Bins insgesamt $(i_1-c_1,\ldots,i_z-c_z)$ Objekte und es gilt

$$\mathrm{BINS}(i_1,\ldots,i_z) = 1 + \mathrm{BINS}(i_1-c_1,\ldots,i_z-c_z).$$

Als allgemeine Rekursionsformel ergibt sich

$$\mathrm{BINS}(i_1,\ldots,i_n) = 1 + \min_{(c_1,\ldots,c_z)\in C} \mathrm{BINS}(i_1-c_1,\ldots,i_z-c_z) \qquad \text{für alle } (i_1,\ldots,i_z) \neq (0,\ldots,0)$$

und $\mathrm{BINS}(0,\ldots,0) = 0$.

Wir berechnen Werte von $\mathrm{BINS}(i_1,\ldots,i_z)$ aufsteigend nach $\sum_{j=1}^{z} i_j$. Insgesamt werden $\mathcal{O}(n^z)$ viele Vektoren mit jeweils Laufzeit $\mathcal{O}(n^z)$ pro Vektor, damit ergibt sich eine Gesamtlaufzeit von $\mathcal{O}(n^{2z})$, also folgender Satz:

Theorem 3.5: Bin-Packing

Das Bin-Packing-Problem mit z verschiedenen Objektgrößen kann für Instanzen mit n Objekten in Zeit $\mathcal{O}(n^{2z})$ gelöst werden (wobei die Konstante in der $\mathcal{O}$-Notation von z abhängt.)

3.2.2 Approximationsschema

Zusätzliche Eingabe: T. Der Algorithmus liefert entweder einen Schedule mit Makespan höchstens $(1+\varepsilon)T$ oder er beweist, dass $\mathrm{OPT} > T$.

Wir bezeichnen einen Job j als *groß*, falls $p_j > \varepsilon T$. Wir runden die Größen aller großen Jobs auf Vielfache von $\varepsilon^2 T$ ab, somit gibt es höchstens $\frac{1}{\varepsilon^2}$ viele Jobgrößen. Auf diesen bestimmen wir eine optimale Bin-Packing-Lösung. Wir erhalten als Antwort entweder eine Packung der großen Jobs auf m Maschinen mit Makespan $\leq T$ (bzgl. der gerundeten Jobgrößen), oder dass es nicht möglich ist (dann brechen wir ab). Für die kleinen Jobs verwenden wir anschließend LeastLoaded.

Für den Makespan gilt insgesamt:

- Erhält eine Maschine einen kleinen Job, betrachte den letzten derartigen Job. Dann gilt

$$L_i(\pi) \leq \underbrace{\left(\frac{1}{m}\sum_{k=1}^{n} p_k\right)}_{\leq T} + p_j \leq (1+\varepsilon)T.$$

- Erhält eine Maschine keinen kleinen Job, dann ist die Last durch die gerundeten Jobgrößen höchstens T. Sei p'_j die abgerundete Jobgröße und p_j die tatsächliche Jobgröße. Dann gilt

$$p_j \geq \varepsilon T, \qquad p'_j \geq p_j - \varepsilon^2 T \geq p_j - \varepsilon p'_j \implies p_j \leq (1+\varepsilon)p'_j,$$

  unter der Voraussetzung, dass $\frac{1}{\varepsilon} \in \mathbb{N}$.

Somit sind nicht gerundete Jobgrößen höchstens um den Faktor $(1+\varepsilon)$ größer und somit ist die Last der Maschine $\leq (1+\varepsilon)T$ bezüglich der tatsächlichen Jobgrößen.

Hier der Algorithmus nochmal in Pseudocode:

Algorithm 9 $B_\varepsilon(T)$

1: Sei $J' = \{j \in J \mid p_j > \varepsilon T\}$ die Menge der großen Jobs
2: Für $j \in J$ sei p'_j der Wert p_j auf das nächste Vielfache von $\varepsilon^2 T$ abgerundet. Sei I' die Instanz, die nur aus Jobs mit den Größen p'_j mit $j \in J'$ besteht.
3: Konstruiere eine Instanz für Bin-Packing mit Objektgrößen $\{p'_j \mid j \in J'\}$ und Kapazität T. Bezeichne a die minimale Anzahl an Bins, die für diese Instanz benötigt werden. Berechne a und die dazugehörige Verteilung π' der Objekte mit Hilfe des dynamischen Programms aus Satz **??**.
4: **if** $a > m$ **then**
5: **return** $\mathrm{OPT}(I) \geq \mathrm{OPT}(I') > T$
6: **else**
7: Erweitere π' auf alle Jobs aus J durch Hinzufügen der Jobs aus $J \setminus J'$ mittels LeastLoaded.
8: **return** resultierenden Schedule π

Somit haben wir gezeigt:

Lemma 3.6: $B_\varepsilon(T)$

Es gibt einen Algorithmus, der in Zeit $\mathcal{O}(n^{\frac{2}{\varepsilon^2}})$ entweder beweist, dass es keinen Schedule mit Makespan T gibt oder einen Schedule mit Makespan $(1+\varepsilon)T$ berechnet.

Für einen Optimalen Makespan gilt num

$$\begin{aligned}\mathrm{OPT} &\geq \left\lceil \frac{1}{m} \sum_{k=1}^{n} p_k \right\rceil \\ \mathrm{OPT} &\geq p_{\max} \\ \mathrm{OPT} &\leq \left\lceil \frac{1}{m} \sum_{k=1}^{n} p_k \right\rceil + p_{\max}.\end{aligned}$$

Setze initial $L := \max\left\{\left\lceil \frac{1}{m}\sum_{k=1}^{n} p_k \right\rceil, p_{\max}\right\}$ und $U := \left\lceil \frac{1}{m}\sum_{k=1}^{n} p_k \right\rceil + p_{\max}$. Dabei haben wir folgende Invarianten:

1. $L \leq \mathrm{OPT}$
2. Wir können einen Schedule mit Makespan höchstens $(1+\varepsilon)U$ in Zeit $\mathcal{O}(n^{\frac{2}{\varepsilon^2}})$ berechnen.

Nun berechne $T := \left\lfloor \frac{L+U}{2} \right\rfloor$ und prüfe, ob ein Schedule mit Makespan T berechnet werden kann. Falls ja, setze $U := T$, sonst setze $L := T + 1$. Es gilt $U - L \leq \frac{p_{\max}}{2^i}$ nach i Iterationen, somit gilt $U - L \leq \frac{1}{2}$ nach $\lfloor \log_2(p_{\max}) \rfloor + 1$ Iterationen. Dann muss $L = U$ gelten und wir erhalten als Resultat folgenden Satz:

Theorem 3.7: PTAS Scheduling

Für Scheduling auf identischen Maschinen existiert ein PTAS mit einer Laufzeit von $\mathcal{O}(m^{\frac{2}{\varepsilon^2}} \log(p_{\max}))$.

[22.10.2025, Vorlesung 4]
[28.10.2025, Vorlesung 5]

3.2.3 Untere Schranke für Approximierbarkeit

Definition 3.8: Stark NP-schwer

Sei Π ein Optimierungsproblem, in dessen Instanzen ganze Zahlen enthalten sind. Π heißt *stark NP-schwer*, wenn es ein Polynom q gibt, sodass Π eingeschränkt auf Instanzen I, in denen alle Zahlen durch $q(|I|)$ beschränkt sind, bereits NP-schwer ist.

Scheduling auf identischen Maschinen ist tatsächlich stark NP-schwer.

Theorem 3.9: Scheduling hat keinen FPTAS

Unter der Annahme P $\neq$ NP *existiert kein FPTAS für Scheduling auf identischen Maschinen.*

Beweis. Angenommen, es gäbe einen FPTAS. Weil Scheduling auf identischen Maschinen stark NP-schwer ist, gibt es ein Polynom q, sodass es bereits NP-schwer ist, das Problem mit Jobgrößen $p_1 \leq \cdots \leq p_n \in \{1, \ldots, q(n)\}$ zu lösen.

Berechne mit dem FPTAS eine $(1+\varepsilon)$-Approximation, wobei $\varepsilon = \frac{1}{2nq(n)}$. Die Laufzeit ist polynomiell in n und $\frac{1}{\varepsilon} = 2nq(n)$, also beides polynomiell in n. Für den Approximationsfaktor erhalten wir $w_A(I, \varepsilon) \leq (1+\varepsilon)\,\mathrm{OPT}$. Es gilt

$$\mathrm{OPT}(I) \leq \sum_{i=1}^{n} p_i \leq n \cdot q(n)$$

und folglich

$$(1+\varepsilon)\,\mathrm{OPT} = \mathrm{OPT} + \underbrace{\varepsilon\,\mathrm{OPT}}_{\leq \frac{1}{2}} \leq \mathrm{OPT}(I) + \varepsilon n q(n) = \mathrm{OPT} + \frac{1}{2}.$$

Weil alle Jobgrößen ganzzahlig sind, sind auch $w_A(I,\varepsilon)$ und $\mathrm{OPT}(I)$ ganze Zahlen. Da die Differenz höchstens $\frac{1}{2}$ ist, ist $w_A(I,\varepsilon) = \mathrm{OPT}(I)$. Also löst der FPTAS das Problem exakt in polynomieller Zeit. Also muss P = NP sein. $\lightning$ zur Annahme □

Kapitel 4

Randomisierte Approximationsalgorithmen

Maximum-Cut Problem MaxCut	*Input:*	Ungerichteter Graph $G = (V, E)$, Kantengewichte $w : E \to \mathbb{R}_{>0}$
	Output:	Menge $U \subseteq V$. Wir nennen $(U, V \setminus U)$ *Schnitt* des Graphen G
	Objective:	Maximiere $\sum_{x \in U} \sum_{y \in V \setminus U} w(x, y)$ mit $w(x, y) = 0$, falls $\{x, y\} \notin E$.

Algorithm 10 RandMaxCut

Input: Wie in **??**
Output: Wie in **??**

```
1: U := ∅
2: for v ∈ V do
3:     Wirf eine Münze
4:     if Münze zeigt Kopf then
5:         Füge v zu U hinzu
6: return U
```

4.1 Grundlagen der Wahrscheinlichkeitsrechnung

4.1.1 Diskrete Wahrscheinlichkeitsräume

Definition 4.1: Wahrscheinlichkeitsraum

Sei Ω eine endliche oder abzählbare Menge (*Ergebnismenge*).
Elemente von Ω heißen *Elementarereignisse*, Teilmengen von Ω heißen *Ereignisse*.
Für $A \subseteq \Omega$ bezeichne $\overline{A} := \Omega \setminus A$ das *Gegenereignis*.
Ein(-e) *Wahrscheinlichkeit(-smaß)* ist eine Funktion $\mathbb{P} : 2^\Omega \to [0, 1]$ mit $\mathbb{P}[\Omega] = 1$ und $\mathbb{P}$ *σ-additiv*, d.h. für $A_1, A_2, \ldots \subseteq \Omega$ mit $A_i \cap A_j = \emptyset$ für $i \neq j$ gilt

$$\mathbb{P}\left[\bigcup_{i=1}^{\infty} A_i\right] = \sum_{i=1}^{\infty} \mathbb{P}[A_i].$$

Das Paar $(\Omega, \mathbb{P})$ heißt (diskreter) *Wahrscheinlichkeitsraum*.

Es reicht, $\mathbb{P}[a]$ für alle $a \in \Omega$ zu definieren, denn

$$\mathbb{P}[\{a_1, \ldots, a_n\}] = \sum_{i=1}^{n} \mathbb{P}[\{a_i\}].$$

Bei einem Würfelwurf setze $\Omega = \{1, \ldots, 6\}$. Dann ist $\mathbb{P}[i] = \frac{1}{6}$ für alle $i \in \Omega$. Es ist auch $\mathbb{P}[\text{"gerade Zahl"}] = \mathbb{P}[\{2, 4, 6\}] = 3 \cdot \frac{1}{6} = \frac{1}{2}$.

Wir modellieren nun einen Wahrscheinlichkeitsraum für RandMaxCut:

Sei $V = \{v_1, \dots, v_n\}$. Setze $\Omega = \{0,1\}^n$. Wir berechnen den Schnitt als

$$U = \{v_i \in V \mid \omega_i = 1\}.$$

Für alle $e = \{v_i, v_j\} \in E$, ist die Wahrscheinlichkeit, dass e über den Schnitt läuft, gerade $\frac{1}{2}$, denn es ist

$$\mathbb{P}[e \text{ läuft über den Schnitt } (U, V \setminus U)] = \mathbb{P}[\{\omega \in \Omega \mid \omega_i \neq \omega_j\}] = \frac{2^{n-1}}{2^n} = \frac{1}{2}.$$

4.1.2 Unabhängigkeit und bedingte Wahrscheinlichkeiten

Definition 4.2: Unabhängige Ereignisse

Sei $(\Omega, \mathbb{P})$ ein diskreter Wahrscheinlichkeitsraum. Zwei Ereignisse $A, B \subseteq \Omega$ heißen *unabhängig*, falls

$$\mathbb{P}[A \cap B] = \mathbb{P}[A] \cdot \mathbb{P}[B].$$

Allgemeiner: Ereignisse $A_1, \dots, A_k \subseteq \Omega$ heißen *unabhängig*, wenn für alle $I \subseteq \{1, \dots, k\}$ gilt

$$\mathbb{P}\left[\bigcap_{i \in I} A_i\right] = \prod_{i \in I} \mathbb{P}[A_i].$$

- Sei $\Omega = \{(i,j) \mid i,j \in \{1,\dots,6\}\}$. Dann ist $\mathbb{P}[(i,j)] = \frac{1}{36}$.

  Seien $A :=$ "Erster Würfel zeigt eine gerade Zahl" und $B :=$ "Zweiter Würfel zeigt ≥ 3". Dann ist $\mathbb{P}[A] = \frac{1}{2}, \mathbb{P}[B] = \frac{2}{3}$ und $\mathbb{P}[A \cap B] = \frac{1}{3} = \mathbb{P}[A] \cdot \mathbb{P}[B]$. Also sind A und B unabhängig.
- Für eine endlihce Folge von Wahrscheinlichkeitsräumen $(\Omega_1, \mathbb{P}_1), \dots, (\Omega_n, \mathbb{P}_n)$ definieren wir den *Produktraum* durch $\Omega := \Omega_1 \times \cdots \times \Omega_n$ und $\mathbb{P}[(x_1, \dots, x_n)] := \mathbb{P}_1[x_1] \cdot \ldots \cdot \mathbb{P}_n[x_n]$.

In RandMaxCut ist der Wahrscheinlichkeitsraum eigentlich ein Produktraum von n Münzwürfen $(\{0,1\}, \mathbb{P})$ mit $\mathbb{P}[0] = \mathbb{P}[1] = \frac{1}{2}$. Seien $e, e' \in E$ ohne gemeinsame Endpunkte. Dann ist

$$\mathbb{P}[\text{"}e \text{ und } e' \text{ laufen über Schnitt"}] = \frac{1}{4} = \mathbb{P}[\text{"}e \text{ läuft über Schnitt"}] \cdot \mathbb{P}[\text{"}e' \text{ läuft über Schnitt"}].$$

Definition 4.3: Bedingte Wahrscheinlichkeit

Sei $(\Omega, \mathbb{P})$ ein diskreter Wahrscheinlichkeitsraum. Seien $A, B \subseteq \Omega$ mit $\mathbb{P}[B] > 0$. Dann bezeichne

$$\mathbb{P}[A \mid B] := \frac{\mathbb{P}[A \cap B]}{\mathbb{P}[B]}$$

die *bedingte Wahrscheinlichkeit* von A gegeben B.

Wir betrachten zwei unanbhängige Würfelwürfe. Sei $A :=$ "Erster Würfel zeigt 6" und $B :=$ "Summe beider Würfe ist 10". Dann sind

$$\mathbb{P}[A] = \frac{1}{6}, \quad \mathbb{P}[B] = \frac{3}{36}, \quad \mathbb{P}[A \cap B] = \frac{1}{36} \quad \text{und} \quad \mathbb{P}[A \mid B] = \frac{1}{3}.$$

4.2 Zufallsvariablen und Erwartungswerte

Definition 4.4: Zufallsvariable

Sei $(\Omega, \mathbb{P})$ ein diskreter Wahrscheinlichkeitsraum. eine Abbildung $X : \Omega \to \mathbb{R}$ heißt (diskrete, reelle) *Zufallsvariable*. Wir schreiben

$$\mathbb{P}[X = a] = \mathbb{P}[\{\omega \in \Omega \mid X(\omega) = a\}]$$

und allgemeiner

$$\mathbb{P}[X \in A] = \mathbb{P}[\{\omega \in \Omega \mid X(\omega) \in A\}].$$

Betrachte zwei unabhängige Würfelwürfe. Sei $X : \Omega := \{1, \dots, 12\}$ definiert durch $X(\omega) := \omega_1 + \omega_2$ für alle $\omega = (\omega_1, \omega_2) \in \Omega$.

Definition 4.5: Unabhängigkeit (Zufallsvariablen)

Seien $X, Y : \Omega \to \mathbb{R}$ Zufallsvariablen. X und Y heißen *unabhängig*, wenn für alle $x, y \in \mathbb{R}$ gilt

$$\mathbb{P}[(X = x) \cap (Y = y)] = \mathbb{P}[X = x] \cdot \mathbb{P}[Y = y].$$

Allgemeiner seien $X_1, \dots, X_n : \Omega \to \mathbb{R}$ Zufallsvariablen. Diese heißen *unabhängig*, wenn für alle $I \subseteq \{1, \dots, n\}, x_i \in \mathbb{R}$ gilt

$$\mathbb{P}\left[\bigcap_{i \in I} (X_i = x)\right] = \prod_{i \in I} \mathbb{P}[X_i = x].$$

Definition 4.6: Erwartungswert

Sei $X : \Omega \to \mathbb{R}$ eine diskrete Zufallsvariable. Falls die Reihe

$$\sum_{\omega \in \Omega} |X(\omega)| \cdot \mathbb{P}[\omega]$$

konvergiert, heißt

$$\mathbb{E}[X] := \sum_{\omega \in \Omega} X(\omega) \cdot \mathbb{P}[\omega]$$

Erwartungswert von X.

[28.10.2025, Vorlesung 5]
[29.10.2025, Vorlesung 6]

Anhang

Liste der Definitionen

Liste der Aussagen

Index