5 Approximationsalgorithmen

Optimierungsproblem

Ein **Optimierungsproblem** Π besteht aus den folgenden Komponenten.

- Menge $\mathcal{I}_\Pi$ von **Instanzen** oder **Eingaben**
- für jedes $I \in \mathcal{I}_\Pi$ Menge $\mathcal{S}_I$ von **Lösungen**
- für jedes $I \in \mathcal{I}_\Pi$ **Zielfunktion** $f_I : \mathcal{S}_I \to \mathbb{R}_{\geq 0}$, die jeder Lösung einen reellen Wert zuweist
- Angabe, ob **minimiert** oder **maximiert** werden soll

Für Eingabe I bezeichne $\mathrm{OPT}(I)$ den **Wert einer optimalen Lösung**.

Beispiel: Spannbaumproblem

- **Eingabe I:** ungerichteter Graph $G = (V, E)$, Kantengewichte $c : E \to \mathbb{N}$
- **Lösungsmenge $\mathcal{S}_I$:** Menge aller Spannbäume von G
- **Zielfunktion f_I:** $f_I(T) = \sum_{e \in T} c(e)$ für Spannbaum $T \in \mathcal{S}_I$
- **Minimiere f_I**

Es gilt $\mathrm{OPT}(I) = \min_{T \in \mathcal{S}_I} f_I(T)$.

5 Approximationsalgorithmen

Ein **Approximationsalgorithmus** A für Π ist ein **Polynomialzeitalgorithmus, der zu jeder Instanz I eine Lösung aus $\mathcal{S}_I$ ausgibt**.

Ein **Approximationsalgorithmus** A für Π ist ein **Polynomialzeitalgorithmus, der zu jeder Instanz I eine Lösung aus $\mathcal{S}_I$ ausgibt**.

Es sei $A(I)$ die Lösung, die A bei Eingabe I ausgibt, und $w_A(I) = f_I(A(I))$ ihr Wert.

Ein **Approximationsalgorithmus** A für Π ist ein **Polynomialzeitalgorithmus, der zu jeder Instanz I eine Lösung aus $\mathcal{S}_I$ ausgibt**.

Es sei $A(I)$ die Lösung, die A bei Eingabe I ausgibt, und $w_A(I) = f_I(A(I))$ ihr Wert.

Definition 5.1 (Approximationsfaktor/Approximationsgüte)

Ein Approximationsalgorithmus A für ein Minimierungs- bzw. Maximierungsproblem Π erreicht einen **Approximationsfaktor** oder eine **Approximationsgüte** von $r \geq 1$ bzw. $r \leq 1$, wenn

$$w_A(I) \leq r \cdot \mathrm{OPT}(I) \quad \text{bzw.} \quad w_A(I) \geq r \cdot \mathrm{OPT}(I)$$

für alle Instanzen $I \in \mathcal{I}_\Pi$ gilt. Wir sagen dann, dass A ein **r-Approximationsalgorithmus** ist.

Ein **Approximationsalgorithmus** A für Π ist ein **Polynomialzeitalgorithmus, der zu jeder Instanz I eine Lösung aus $\mathcal{S}_I$ ausgibt**.

Es sei $A(I)$ die Lösung, die A bei Eingabe I ausgibt, und $w_A(I) = f_I(A(I))$ ihr Wert.

Definition 5.1 (Approximationsfaktor/Approximationsgüte)

Ein Approximationsalgorithmus A für ein Minimierungs- bzw. Maximierungsproblem Π erreicht einen **Approximationsfaktor** oder eine **Approximationsgüte** von $r \geq 1$ bzw. $r \leq 1$, wenn

$$w_A(I) \leq r \cdot \mathrm{OPT}(I) \quad \text{bzw.} \quad w_A(I) \geq r \cdot \mathrm{OPT}(I)$$

für alle Instanzen $I \in \mathcal{I}_\Pi$ gilt. Wir sagen dann, dass A ein **r-Approximationsalgorithmus** ist.

Ist Π NP-schwer und gilt P $\neq$ NP, so existiert für Π kein 1-Approximationsalgorithmus.

Scheduling auf identischen Maschinen:

Eingabe: Menge $J = \{1, \ldots, n\}$ von **Jobs**, **Jobgrößen** $p_1, \ldots, p_n \in \mathbb{R}_{>0}$
Menge $M = \{1, \ldots, m\}$ von **Maschinen**

Lösungen: alle **Schedules** $\pi : J \to M$

5.1 Scheduling auf identischen Maschinen

Scheduling auf identischen Maschinen:

Eingabe: Menge $J = \{1, \ldots, n\}$ von **Jobs**, **Jobgrößen** $p_1, \ldots, p_n \in \mathbb{R}_{>0}$
Menge $M = \{1, \ldots, m\}$ von **Maschinen**

Lösungen: alle **Schedules** $\pi : J \to M$

Wir bezeichnen mit $L_i(\pi)$ die **Ausführungszeit** von Maschine $i \in M$ in Schedule π, d. h.

$$L_i(\pi) = \sum_{j \in J : \pi(j) = i} p_j.$$

5.1 Scheduling auf identischen Maschinen

Scheduling auf identischen Maschinen:

Eingabe:	Menge $J = \{1, \ldots, n\}$ von **Jobs**, **Jobgrößen** $p_1, \ldots, p_n \in \mathbb{R}_{>0}$
	Menge $M = \{1, \ldots, m\}$ von **Maschinen**
Lösungen:	alle **Schedules** $\pi : J \to M$

Wir bezeichnen mit $L_i(\pi)$ die **Ausführungszeit** von Maschine $i \in M$ in Schedule π, d. h.

$$L_i(\pi) = \sum_{j \in J : \pi(j) = i} p_j.$$

Der **Makespan** $C(\pi)$ soll minimiert werden:

$$C(\pi) = \max_{i \in M} L_i(\pi).$$

5.1 Scheduling auf identischen Maschinen

Scheduling auf identischen Maschinen:

Eingabe: Menge $J = \{1, \ldots, n\}$ von **Jobs**, **Jobgrößen** $p_1, \ldots, p_n \in \mathbb{R}_{>0}$
Menge $M = \{1, \ldots, m\}$ von **Maschinen**

Lösungen: alle **Schedules** $\pi : J \to M$

Wir bezeichnen mit $L_i(\pi)$ die **Ausführungszeit** von Maschine $i \in M$ in Schedule π, d. h.

$$L_i(\pi) = \sum_{j \in J : \pi(j) = i} p_j.$$

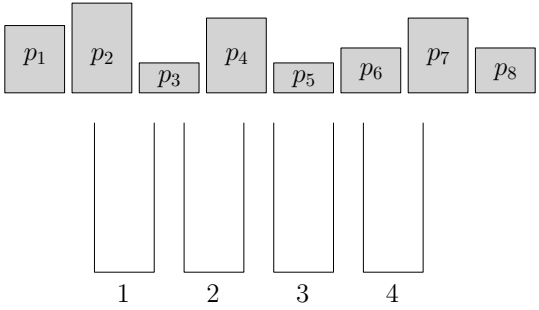
Der **Makespan** $C(\pi)$ soll minimiert werden:

$$C(\pi) = \max_{i \in M} L_i(\pi).$$

Dieses Problem ist **NP-schwer** (Übung).

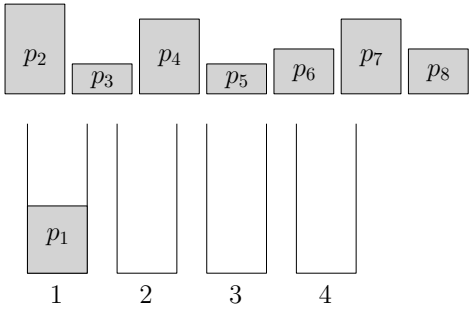
5.1 Scheduling auf identischen Maschinen

Greedy-Algorithmus LEAST-LOADED**:** Betrachte Jobs in der Reihenfolge $1, 2, \ldots, n$ und weise jeden Job einer Maschine zu, die die **kleinste Ausführungszeit bezogen auf die bereits zugewiesenen Jobs** besitzt.



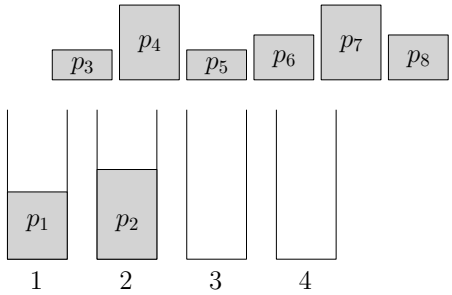
5.1 Scheduling auf identischen Maschinen

Greedy-Algorithmus LEAST-LOADED**:** Betrachte Jobs in der Reihenfolge $1, 2, \ldots, n$ und weise jeden Job einer Maschine zu, die die **kleinste Ausführungszeit bezogen auf die bereits zugewiesenen Jobs** besitzt.



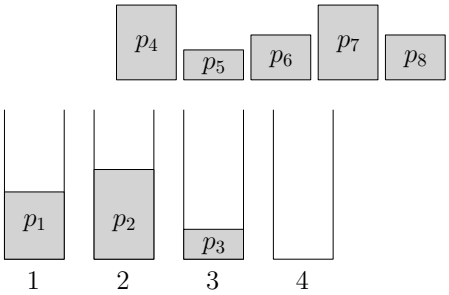
5.1 Scheduling auf identischen Maschinen

Greedy-Algorithmus LEAST-LOADED**:** Betrachte Jobs in der Reihenfolge $1, 2, \ldots, n$ und weise jeden Job einer Maschine zu, die die **kleinste Ausführungszeit bezogen auf die bereits zugewiesenen Jobs** besitzt.



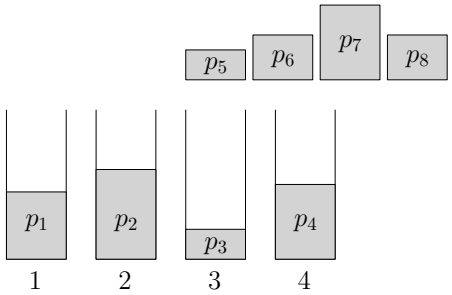
5.1 Scheduling auf identischen Maschinen

Greedy-Algorithmus LEAST-LOADED**:** Betrachte Jobs in der Reihenfolge $1, 2, \ldots, n$ und weise jeden Job einer Maschine zu, die die **kleinste Ausführungszeit bezogen auf die bereits zugewiesenen Jobs** besitzt.



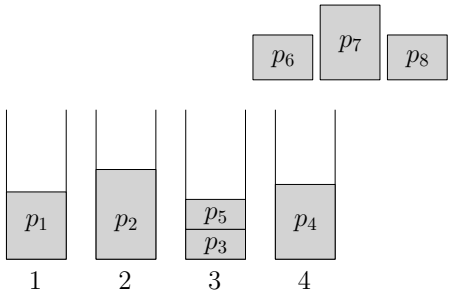
5.1 Scheduling auf identischen Maschinen

Greedy-Algorithmus LEAST-LOADED**:** Betrachte Jobs in der Reihenfolge $1, 2, \ldots, n$ und weise jeden Job einer Maschine zu, die die **kleinste Ausführungszeit bezogen auf die bereits zugewiesenen Jobs** besitzt.



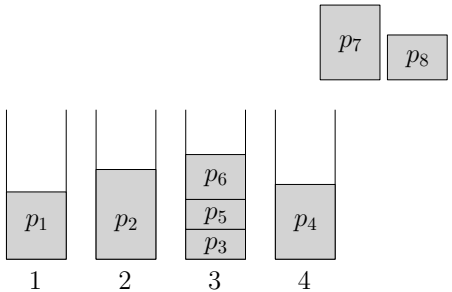
5.1 Scheduling auf identischen Maschinen

Greedy-Algorithmus LEAST-LOADED**:** Betrachte Jobs in der Reihenfolge $1, 2, \ldots, n$ und weise jeden Job einer Maschine zu, die die **kleinste Ausführungszeit bezogen auf die bereits zugewiesenen Jobs** besitzt.



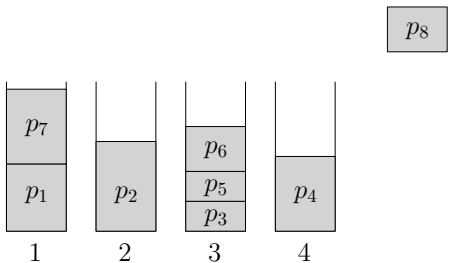
5.1 Scheduling auf identischen Maschinen

Greedy-Algorithmus LEAST-LOADED**:** Betrachte Jobs in der Reihenfolge $1, 2, \ldots, n$ und weise jeden Job einer Maschine zu, die die **kleinste Ausführungszeit bezogen auf die bereits zugewiesenen Jobs** besitzt.



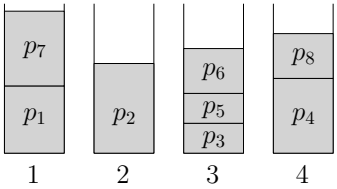
5.1 Scheduling auf identischen Maschinen

Greedy-Algorithmus LEAST-LOADED**:** Betrachte Jobs in der Reihenfolge $1, 2, \ldots, n$ und weise jeden Job einer Maschine zu, die die **kleinste Ausführungszeit bezogen auf die bereits zugewiesenen Jobs** besitzt.



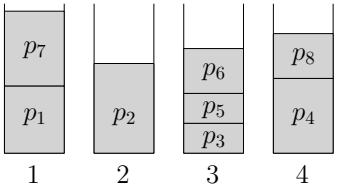
5.1 Scheduling auf identischen Maschinen

Greedy-Algorithmus LEAST-LOADED**:** Betrachte Jobs in der Reihenfolge $1, 2, \ldots, n$ und weise jeden Job einer Maschine zu, die die **kleinste Ausführungszeit bezogen auf die bereits zugewiesenen Jobs** besitzt.



5.1 Scheduling auf identischen Maschinen

Greedy-Algorithmus LEAST-LOADED**:** Betrachte Jobs in der Reihenfolge $1, 2, \ldots, n$ und weise jeden Job einer Maschine zu, die die **kleinste Ausführungszeit bezogen auf die bereits zugewiesenen Jobs** besitzt.



Theorem 5.2

Der LEAST-LOADED-Algorithmus ist ein $(2 - 1/m)$-Approximationsalgorithmus für das Problem Scheduling auf identischen Maschinen.

Beweis:

Untere Schranken für OPT:

Sei π^* ein optimaler Schedule. Es gilt

$$C(\pi^*) \geq \frac{1}{m} \sum_{j \in J} p_j \qquad \text{und} \qquad C(\pi^*) \geq \max_{j \in J} p_j.$$

Obere Schranke für den Makespan von LEAST-LOADED:

Sei π der Schedule, den der LEAST-LOADED-Algorithmus berechnet.

Sei $i \in M$ eine **Maschine mit größter Ausführungszeit**, d. h. $C(\pi) = L_i(\pi)$.

Es sei $j \in J$ der Job, der **als letztes Maschine i hinzugefügt** wurde.

Obere Schranke für den Makespan von LEAST-LOADED:

Sei π der Schedule, den der LEAST-LOADED-Algorithmus berechnet.

Sei $i \in M$ eine **Maschine mit größter Ausführungszeit**, d. h. $C(\pi) = L_i(\pi)$.

Es sei $j \in J$ der Job, der **als letztes Maschine i hinzugefügt** wurde.

$$C(\pi) = L_i(\pi) \leq \frac{1}{m} \left(\sum_{k=1}^{j-1} p_k \right) + p_j$$

Obere Schranke für den Makespan von LEAST-LOADED:

Sei π der Schedule, den der LEAST-LOADED-Algorithmus berechnet.

Sei $i \in M$ eine **Maschine mit größter Ausführungszeit**, d. h. $C(\pi) = L_i(\pi)$.

Es sei $j \in J$ der Job, der **als letztes Maschine i hinzugefügt** wurde.

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

Obere Schranke für den Makespan von LEAST-LOADED:

Sei π der Schedule, den der LEAST-LOADED-Algorithmus berechnet.

Sei $i \in M$ eine **Maschine mit größter Ausführungszeit**, d. h. $C(\pi) = L_i(\pi)$.

Es sei $j \in J$ der Job, der **als letztes Maschine i hinzugefügt** wurde.

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

$$= \frac{1}{m}\left(\sum_{k \in J} p_k\right) + \left(1 - \frac{1}{m}\right) p_j$$

Obere Schranke für den Makespan von LEAST-LOADED:

Sei π der Schedule, den der LEAST-LOADED-Algorithmus berechnet.

Sei $i \in M$ eine **Maschine mit größter Ausführungszeit**, d. h. $C(\pi) = L_i(\pi)$.

Es sei $j \in J$ der Job, der **als letztes Maschine i hinzugefügt** wurde.

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

$$= \frac{1}{m}\left(\sum_{k \in J} p_k\right) + \left(1 - \frac{1}{m}\right) p_j \leq \frac{1}{m}\left(\sum_{k \in J} p_k\right) + \left(1 - \frac{1}{m}\right) \cdot \max_{k \in J} p_k$$

Obere Schranke für den Makespan von LEAST-LOADED:

Sei π der Schedule, den der LEAST-LOADED-Algorithmus berechnet.

Sei $i \in M$ eine **Maschine mit größter Ausführungszeit**, d. h. $C(\pi) = L_i(\pi)$.

Es sei $j \in J$ der Job, der **als letztes Maschine i hinzugefügt** wurde.

$$\begin{aligned} C(\pi) = L_i(\pi) &\leq \frac{1}{m}\left(\sum_{k=1}^{j-1} p_k\right) + p_j \leq \frac{1}{m}\left(\sum_{k \in J \setminus \{j\}} p_k\right) + p_j \\ &= \frac{1}{m}\left(\sum_{k \in J} p_k\right) + \left(1 - \frac{1}{m}\right) p_j \leq \frac{1}{m}\left(\sum_{k \in J} p_k\right) + \left(1 - \frac{1}{m}\right) \cdot \max_{k \in J} p_k \\ &\leq C(\pi^*) + \left(1 - \frac{1}{m}\right) \cdot C(\pi^*) = \left(2 - \frac{1}{m}\right) \cdot C(\pi^*), \end{aligned}$$

wobei wir die beiden unteren Schranken für $C(\pi^*)$ benutzt haben. □

Untere Schranke für den Approximationsfaktor von LEAST-LOADED:

Sei m beliebig.

Setze $n = m(m-1)+1$ mit $p_1 = \ldots = p_{n-1} = 1$ und $p_n = m$.

Untere Schranke für den Approximationsfaktor von LEAST-LOADED:

Sei m beliebig.

Setze $n = m(m-1)+1$ mit $p_1 = \ldots = p_{n-1} = 1$ und $p_n = m$.

Dann gilt: $\mathrm{OPT} = m$.

Untere Schranke für den Approximationsfaktor von LEAST-LOADED:

Sei m beliebig.

Setze $n = m(m-1)+1$ mit $p_1 = \ldots = p_{n-1} = 1$ und $p_n = m$.

Dann gilt: $\mathrm{OPT} = m$.

LEAST-LOADED verteilt die ersten $m(m-1)$ Jobs gleichmäßig auf den Maschinen und platziert den letzten Job auf einer beliebigen Maschine i. Diese Maschine hat dann eine Ausführungszeit von $(m-1)+m = 2m-1$.

Untere Schranke für den Approximationsfaktor von LEAST-LOADED:

Sei m beliebig.

Setze $n = m(m-1)+1$ mit $p_1 = \ldots = p_{n-1} = 1$ und $p_n = m$.

Dann gilt: $\mathrm{OPT} = m$.

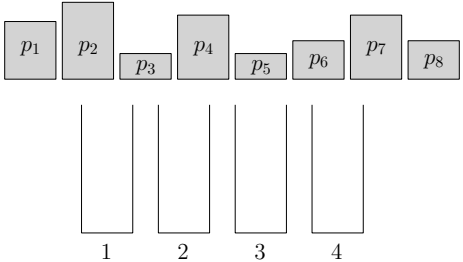
LEAST-LOADED verteilt die ersten $m(m-1)$ Jobs gleichmäßig auf den Maschinen und platziert den letzten Job auf einer beliebigen Maschine i. Diese Maschine hat dann eine Ausführungszeit von $(m-1)+m = 2m-1$.

Für den **Approximationsfaktor von LEAST-LOADED** gilt auf dieser Eingabe:

$$\frac{2m-1}{m} = 2 - \frac{1}{m}.$$

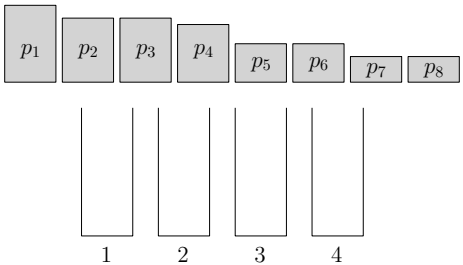
LONGEST-PROCESSING-TIME (LPT)

1. Sortiere die Jobs so, dass $p_1 \geq p_2 \geq \ldots \geq p_n$ gilt.
2. Führe den LEAST-LOADED-Algorithmus auf den so sortierten Jobs aus.



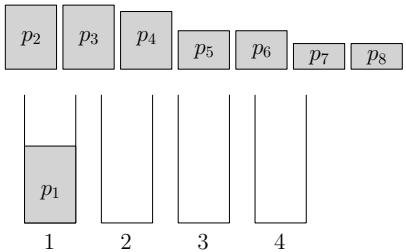
LONGEST-PROCESSING-TIME (LPT)

1. Sortiere die Jobs so, dass $p_1 \geq p_2 \geq \ldots \geq p_n$ gilt.
2. Führe den LEAST-LOADED-Algorithmus auf den so sortierten Jobs aus.



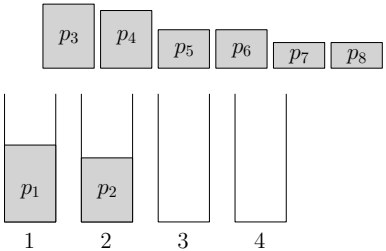
LONGEST-PROCESSING-TIME (LPT)

1. Sortiere die Jobs so, dass $p_1 \geq p_2 \geq \ldots \geq p_n$ gilt.
2. Führe den LEAST-LOADED-Algorithmus auf den so sortierten Jobs aus.



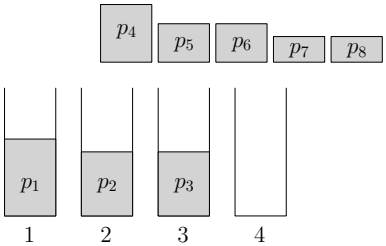
LONGEST-PROCESSING-TIME (LPT)

1. Sortiere die Jobs so, dass $p_1 \geq p_2 \geq \ldots \geq p_n$ gilt.
2. Führe den LEAST-LOADED-Algorithmus auf den so sortierten Jobs aus.



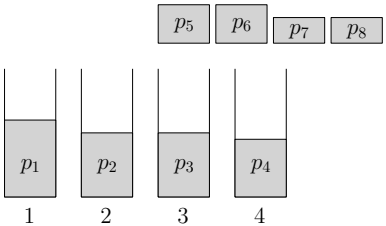
LONGEST-PROCESSING-TIME (LPT)

1. Sortiere die Jobs so, dass $p_1 \geq p_2 \geq \ldots \geq p_n$ gilt.
2. Führe den LEAST-LOADED-Algorithmus auf den so sortierten Jobs aus.



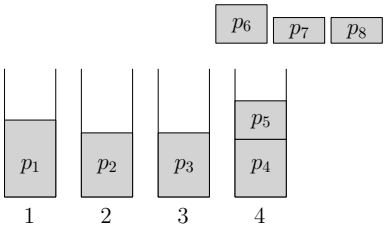
LONGEST-PROCESSING-TIME (LPT)

1. Sortiere die Jobs so, dass $p_1 \geq p_2 \geq \ldots \geq p_n$ gilt.
2. Führe den LEAST-LOADED-Algorithmus auf den so sortierten Jobs aus.



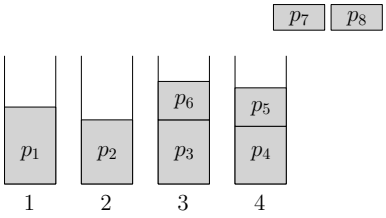
LONGEST-PROCESSING-TIME (LPT)

1. Sortiere die Jobs so, dass $p_1 \geq p_2 \geq \ldots \geq p_n$ gilt.
2. Führe den LEAST-LOADED-Algorithmus auf den so sortierten Jobs aus.



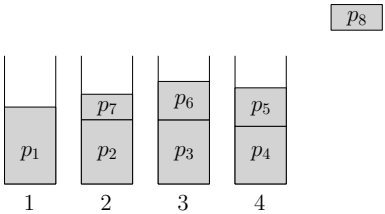
LONGEST-PROCESSING-TIME (LPT)

1. Sortiere die Jobs so, dass $p_1 \geq p_2 \geq \ldots \geq p_n$ gilt.
2. Führe den LEAST-LOADED-Algorithmus auf den so sortierten Jobs aus.



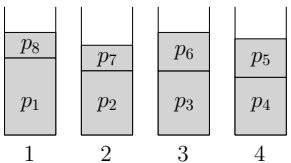
LONGEST-PROCESSING-TIME (LPT)

1. Sortiere die Jobs so, dass $p_1 \geq p_2 \geq \ldots \geq p_n$ gilt.
2. Führe den LEAST-LOADED-Algorithmus auf den so sortierten Jobs aus.



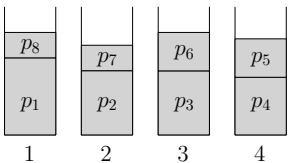
LONGEST-PROCESSING-TIME (LPT)

1. Sortiere die Jobs so, dass $p_1 \geq p_2 \geq \ldots \geq p_n$ gilt.
2. Führe den LEAST-LOADED-Algorithmus auf den so sortierten Jobs aus.



LONGEST-PROCESSING-TIME (LPT)

1. Sortiere die Jobs so, dass $p_1 \geq p_2 \geq \ldots \geq p_n$ gilt.
2. Führe den LEAST-LOADED-Algorithmus auf den so sortierten Jobs aus.



Theorem 5.3

Der LONGEST-PROCESSING-TIME-Algorithmus ist ein $\frac{4}{3}$-Approximationsalgorithmus für Scheduling auf identischen Maschinen.

Beweis durch Widerspruch:

Sei Eingabe $p_1 \geq \ldots \geq p_n$ mit m Maschinen gegeben, auf der LPT einen Schedule π mit $C(\pi) > \frac{4}{3} \cdot \mathrm{OPT}$ berechnet. Außerdem sei **n kleinstmöglich gewählt**.

Beweis durch Widerspruch:

Sei Eingabe $p_1 \geq \ldots \geq p_n$ mit m Maschinen gegeben, auf der LPT einen Schedule π mit $C(\pi) > \frac{4}{3} \cdot \mathrm{OPT}$ berechnet. Außerdem sei **n kleinstmöglich gewählt**.

Es sei nun $i \in M$ eine **Maschine mit größter Ausführungszeit** und $j \in J$ der **letzte Job, der Maschine i zugewiesen** wird. Dann gilt $j = n$ und

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

Beweis durch Widerspruch:

Sei Eingabe $p_1 \geq \ldots \geq p_n$ mit m Maschinen gegeben, auf der LPT einen Schedule π mit $C(\pi) > \frac{4}{3} \cdot \mathrm{OPT}$ berechnet. Außerdem sei **n kleinstmöglich gewählt**.

Es sei nun $i \in M$ eine **Maschine mit größter Ausführungszeit** und $j \in J$ der **letzte Job, der Maschine i zugewiesen** wird. Dann gilt $j = n$ und

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

Aus $C(\pi) > \frac{4}{3} \cdot \mathrm{OPT}$ folgt demnach $p_n > \mathrm{OPT}/3$.

Beweis durch Widerspruch:

Sei Eingabe $p_1 \geq \ldots \geq p_n$ mit m Maschinen gegeben, auf der LPT einen Schedule π mit $C(\pi) > \frac{4}{3} \cdot \mathrm{OPT}$ berechnet. Außerdem sei **n kleinstmöglich gewählt**.

Es sei nun $i \in M$ eine **Maschine mit größter Ausführungszeit** und $j \in J$ der **letzte Job, der Maschine i zugewiesen** wird. Dann gilt $j = n$ und

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

Aus $C(\pi) > \frac{4}{3} \cdot \mathrm{OPT}$ folgt demnach $p_n > \mathrm{OPT}/3$.
Dies bedeutet, dass $p_j > \mathrm{OPT}/3$ **für alle** $j \in J$ **gilt**.

$\forall j \in J : p_j > \mathrm{OPT}/3$

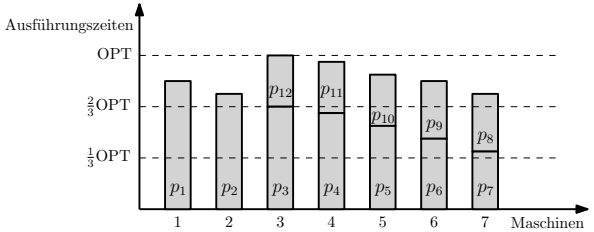
$\Rightarrow$ In opt. Schedule π^* **erhält jede Maschine maximal zwei Jobs** (insbesondere $n \leq 2m$).

$\forall j \in J : p_j > \mathrm{OPT}/3$

$\Rightarrow$ In opt. Schedule π^* **erhält jede Maschine maximal zwei Jobs** (insbesondere $n \leq 2m$).

Optimaler Schedule:

- Jeder Job $j \in \{1, \ldots, \min\{n, m\}\}$ wird Maschine j zugewiesen.
- Jeder Job $j \in \{m+1, \ldots, n\}$ wird Maschine $2m - j + 1$ zugewiesen.



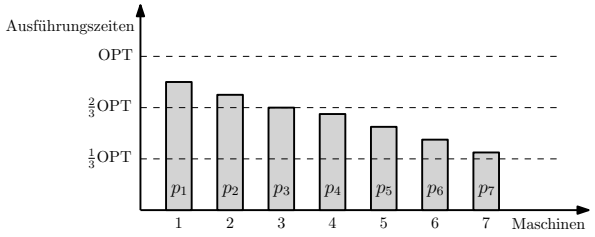
$\forall j \in J : p_j > \mathrm{OPT}/3$

$\Rightarrow$ In opt. Schedule π^* **erhält jede Maschine maximal zwei Jobs** (insbesondere $n \leq 2m$).

Optimaler Schedule:

- Jeder Job $j \in \{1, \ldots, \min\{n, m\}\}$ wird Maschine j zugewiesen.
- Jeder Job $j \in \{m+1, \ldots, n\}$ wird Maschine $2m - j + 1$ zugewiesen.

LPT berechnet opt. Schedule π^*



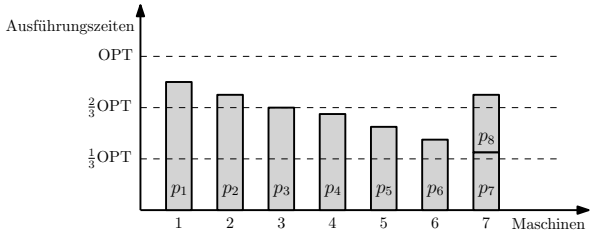
$\forall j \in J : p_j > \mathrm{OPT}/3$

$\Rightarrow$ In opt. Schedule π^* **erhält jede Maschine maximal zwei Jobs** (insbesondere $n \leq 2m$).

Optimaler Schedule:

- Jeder Job $j \in \{1, \ldots, \min\{n, m\}\}$ wird Maschine j zugewiesen.
- Jeder Job $j \in \{m+1, \ldots, n\}$ wird Maschine $2m - j + 1$ zugewiesen.

LPT berechnet opt. Schedule π^*



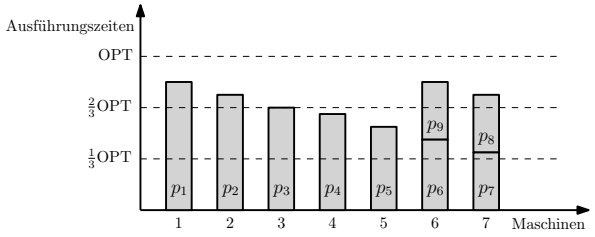
$\forall j \in J : p_j > \mathrm{OPT}/3$

$\Rightarrow$ In opt. Schedule π^* **erhält jede Maschine maximal zwei Jobs** (insbesondere $n \leq 2m$).

Optimaler Schedule:

- Jeder Job $j \in \{1, \ldots, \min\{n, m\}\}$ wird Maschine j zugewiesen.
- Jeder Job $j \in \{m+1, \ldots, n\}$ wird Maschine $2m - j + 1$ zugewiesen.

LPT berechnet opt. Schedule π^*



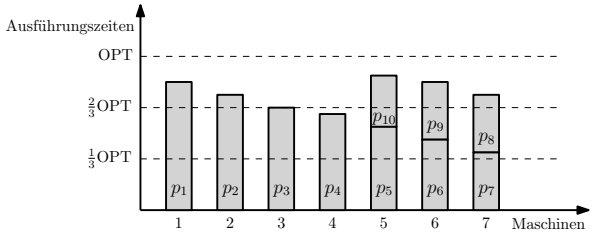
$\forall j \in J : p_j > \mathrm{OPT}/3$

$\Rightarrow$ In opt. Schedule π^* **erhält jede Maschine maximal zwei Jobs** (insbesondere $n \leq 2m$).

Optimaler Schedule:

- Jeder Job $j \in \{1, \ldots, \min\{n, m\}\}$ wird Maschine j zugewiesen.
- Jeder Job $j \in \{m+1, \ldots, n\}$ wird Maschine $2m - j + 1$ zugewiesen.

LPT berechnet opt. Schedule π^*



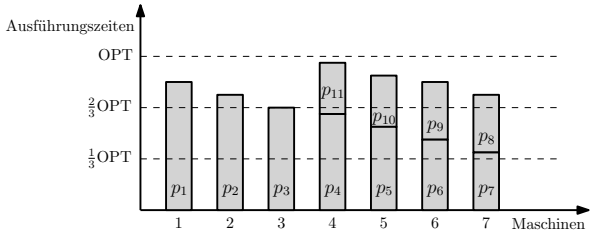
$\forall j \in J : p_j > \mathrm{OPT}/3$

$\Rightarrow$ In opt. Schedule π^* **erhält jede Maschine maximal zwei Jobs** (insbesondere $n \leq 2m$).

Optimaler Schedule:

- Jeder Job $j \in \{1, \ldots, \min\{n, m\}\}$ wird Maschine j zugewiesen.
- Jeder Job $j \in \{m+1, \ldots, n\}$ wird Maschine $2m - j + 1$ zugewiesen.

LPT berechnet opt. Schedule π^*



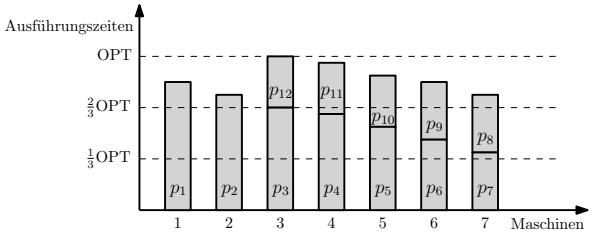
$\forall j \in J : p_j > \mathrm{OPT}/3$

$\Rightarrow$ In opt. Schedule π^* **erhält jede Maschine maximal zwei Jobs** (insbesondere $n \leq 2m$).

Optimaler Schedule:

- Jeder Job $j \in \{1, \ldots, \min\{n, m\}\}$ wird Maschine j zugewiesen.
- Jeder Job $j \in \{m+1, \ldots, n\}$ wird Maschine $2m - j + 1$ zugewiesen.

LPT berechnet opt. Schedule π^*



□

Traveling Salesman Problem (TSP)

Eingabe:	Menge $V = \{v_1, \ldots, v_n\}$ von Knoten
	symmetrische Distanzfunktion $d : V \times V \to \mathbb{R}_{\geq 0}$
	(d. h. $\forall u, v \in V : d(u, v) = d(v, u) \geq 0$)
Lösungen:	alle Permutationen $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$
	eine solche Permutation nennen wir auch *Tour*
Zielfunktion:	minimiere $\sum_{i=1}^{n-1} d\big(v_{\pi(i)}, v_{\pi(i+1)}\big) + d\big(v_{\pi(n)}, v_{\pi(1)}\big)$

5.2 Traveling Salesman Problem

Traveling Salesman Problem (TSP)

Eingabe: Menge $V = \{v_1, \ldots, v_n\}$ von Knoten
symmetrische Distanzfunktion $d : V \times V \to \mathbb{R}_{\geq 0}$
(d. h. $\forall u, v \in V : d(u, v) = d(v, u) \geq 0$)

Lösungen: alle Permutationen $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$
eine solche Permutation nennen wir auch *Tour*

Zielfunktion: minimiere $\sum_{i=1}^{n-1} d\left(v_{\pi(i)}, v_{\pi(i+1)}\right) + d\left(v_{\pi(n)}, v_{\pi(1)}\right)$

Theorem 5.4

Falls P $\neq$ NP, so existiert kein 2^n-Approximationsalgorithmus für das TSP.

Beweis:

Hamiltonkreis-Problem (HC): Existiert in einem ungerichteten Graph ein Kreis, der **jeden Knoten genau einmal** enthält?

HC ist **NP-vollständig** (das folgt aus einer Reduktion von 3-SAT).

Beweis:

Hamiltonkreis-Problem (HC): Existiert in einem ungerichteten Graph ein Kreis, der **jeden Knoten genau einmal** enthält?

HC ist **NP-vollständig** (das folgt aus einer Reduktion von 3-SAT).

Wir konstruieren polynomielle Reduktion von HC auf TSP, die folgenden Schluss zulässt:
Falls ein **2^n-Approximationsalgorithmus A für das TSP** existiert, so kann **HC in polynomieller Zeit gelöst** werden.

Beweis:

Hamiltonkreis-Problem (HC): Existiert in einem ungerichteten Graph ein Kreis, der **jeden Knoten genau einmal** enthält?

HC ist **NP-vollständig** (das folgt aus einer Reduktion von 3-SAT).

Wir konstruieren polynomielle Reduktion von HC auf TSP, die folgenden Schluss zulässt:
Falls ein **2^n-Approximationsalgorithmus A für das TSP** existiert, so kann **HC in polynomieller Zeit gelöst** werden.

Sei $G = (V, E)$ Eingabe für HC. Wir **konstruieren TSP-Instanz** auf V mit:

$$\forall u, v \in V, u \neq v : d(u, v) = d(v, u) = \begin{cases} 1 & \text{falls } \{u, v\} \in E, \\ n2^{n+1} & \text{falls } \{u, v\} \notin E. \end{cases}$$

Beweis:

Hamiltonkreis-Problem (HC): Existiert in einem ungerichteten Graph ein Kreis, der **jeden Knoten genau einmal** enthält?

HC ist **NP-vollständig** (das folgt aus einer Reduktion von 3-SAT).

Wir konstruieren polynomielle Reduktion von HC auf TSP, die folgenden Schluss zulässt: Falls ein **2^n-Approximationsalgorithmus A für das TSP** existiert, so kann **HC in polynomieller Zeit gelöst** werden.

Sei $G = (V, E)$ Eingabe für HC. Wir **konstruieren TSP-Instanz** auf V mit:

$$\forall u, v \in V, u \neq v : d(u, v) = d(v, u) = \begin{cases} 1 & \text{falls } \{u, v\} \in E, \\ n2^{n+1} & \text{falls } \{u, v\} \notin E. \end{cases}$$

G enthält HC. $\Rightarrow$ Es gibt **TSP-Tour C der Länge n**.

Beweis:

Hamiltonkreis-Problem (HC): Existiert in einem ungerichteten Graph ein Kreis, der **jeden Knoten genau einmal** enthält?

HC ist **NP-vollständig** (das folgt aus einer Reduktion von 3-SAT).

Wir konstruieren polynomielle Reduktion von HC auf TSP, die folgenden Schluss zulässt: Falls ein **2^n-Approximationsalgorithmus A für das TSP** existiert, so kann **HC in polynomieller Zeit gelöst** werden.

Sei $G = (V, E)$ Eingabe für HC. Wir **konstruieren TSP-Instanz** auf V mit:

$$\forall u, v \in V, u \neq v : d(u, v) = d(v, u) = \begin{cases} 1 & \text{falls } \{u, v\} \in E, \\ n2^{n+1} & \text{falls } \{u, v\} \notin E. \end{cases}$$

G enthält HC. $\Rightarrow$ Es gibt **TSP-Tour C der Länge n**. $\Rightarrow$ A berechnet Tour C' mit $d(C') \leq 2^n \cdot d(C) \leq n2^n$.

Beweis:

Hamiltonkreis-Problem (HC): Existiert in einem ungerichteten Graph ein Kreis, der **jeden Knoten genau einmal** enthält?

HC ist **NP-vollständig** (das folgt aus einer Reduktion von 3-SAT).

Wir konstruieren polynomielle Reduktion von HC auf TSP, die folgenden Schluss zulässt: Falls ein **2^n-Approximationsalgorithmus A für das TSP** existiert, so kann **HC in polynomieller Zeit gelöst** werden.

Sei $G = (V, E)$ Eingabe für HC. Wir **konstruieren TSP-Instanz** auf V mit:

$$\forall u, v \in V, u \neq v : d(u, v) = d(v, u) = \begin{cases} 1 & \text{falls } \{u, v\} \in E, \\ n2^{n+1} & \text{falls } \{u, v\} \notin E. \end{cases}$$

G enthält HC. $\Rightarrow$ Es gibt **TSP-Tour C der Länge n**. $\Rightarrow$ A berechnet Tour C' mit $d(C') \leq 2^n \cdot d(C) \leq n2^n$. C' **enthält nur Kanten e $\in$ E** $\Rightarrow$ C' ist Hamiltonkreis in G. $\square$

5.2 Traveling Salesman Problem

Beim **metrischen TSP** bilden die Distanzen d eine Metrik auf V.

5.2 Traveling Salesman Problem

Beim **metrischen TSP** bilden die Distanzen d eine Metrik auf V.

Definition 5.5

Sei X eine Menge und $d : X \times X \to \mathbb{R}_{\geq 0}$ eine Funktion. Die Funktion d heißt **Metrik auf X**, wenn die folgenden drei Eigenschaften erfüllt sind.

- $\forall x, y \in X : d(x, y) = 0 \iff x = y$ **(positive Definitheit)**
- $\forall x, y \in X : d(x, y) = d(y, x)$ **(Symmetrie)**
- $\forall x, y, z \in X : d(x, z) \leq d(x, y) + d(y, z)$ **(Dreiecksungleichung)**

Das Paar (X, d) heißt **metrischer Raum**.

5.2 Traveling Salesman Problem

Beim **metrischen TSP** bilden die Distanzen d eine Metrik auf V.

Definition 5.5

Sei X eine Menge und $d : X \times X \to \mathbb{R}_{\geq 0}$ eine Funktion. Die Funktion d heißt **Metrik auf X**, wenn die folgenden drei Eigenschaften erfüllt sind.

- $\forall x, y \in X : d(x, y) = 0 \iff x = y$ **(positive Definitheit)**
- $\forall x, y \in X : d(x, y) = d(y, x)$ **(Symmetrie)**
- $\forall x, y, z \in X : d(x, z) \leq d(x, y) + d(y, z)$ **(Dreiecksungleichung)**

Das Paar (X, d) heißt **metrischer Raum**.

Das metrische TSP ist ein Spezialfall des TSP.
Es ist noch **NP-schwer** denn das TSP ist bereits dann NP-schwer, wenn alle Distanzen entweder 1 oder 2 sind.