

# Algorithmen II

**Peter Sanders**

**Übungen:**

**Moritz Laupichler, Nikolai Maas**

Institut für Theoretische Informatik

Web:

[http://algo2.iti.kit.edu/AlgorithmenII\\_WS23.php](http://algo2.iti.kit.edu/AlgorithmenII_WS23.php)

# 8 Approximationsalgorithmen

Eine Möglichkeit zum **Umgang mit NP-harten Problemen**

Beobachtung: Fast alle interessanten Optimierungsprobleme sind NP-hart

Auswege:

- Trotzdem optimale Lösungen suchen und riskieren, dass der Algorithmus nicht fertig wird
- Ad-hoc Heuristiken. Man kriegt eine Lösung aber wie gut ist die?
- Approximationsalgorithmen:**  
Polynomielle Ausführungszeit.  
Lösungen **garantiert "nah"** am Optimum.
- Problem umdefinieren/spezialisieren

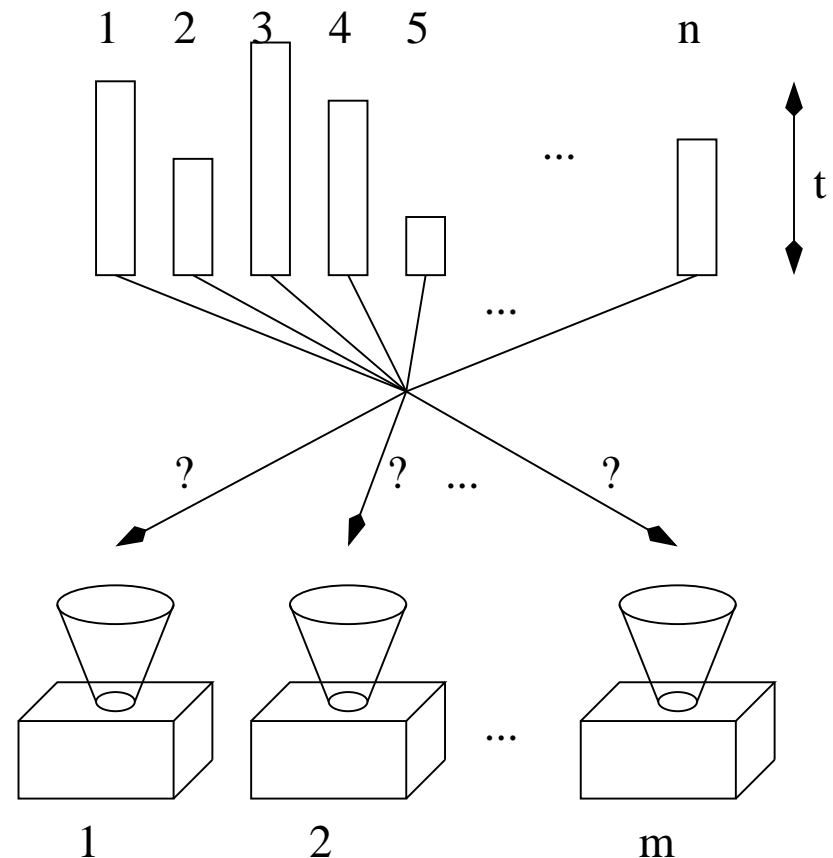
# Scheduling unabhängiger gewichteter Jobs auf parallelen Maschinen

$x(j)$ : Maschine auf der Job  $j$  ausgeführt wird

$L_i$ :  $\sum_{x(j)=i} t_j$ , Last von Maschine  $i$

Zielfunktion: Minimiere Makespan

$$L_{\max} = \max_i L_i$$



Details: Identische Maschinen, unabhängige Jobs, bekannte Ausführungszeiten, offline

## List Scheduling

ListScheduling( $n, m, \mathbf{t}$ )

$J := \{1, \dots, n\}$

array  $L[1..m] = [0, \dots, 0]$

**while**  $J \neq \emptyset$  **do**

    pick **any**  $j \in J$

$J := J \setminus \{j\}$

    // Shortest Queue:

    pick  $i$  such that  $L[i]$  is minimized

$\mathbf{x}(j) := i$

$L[i] := L[i] + t_j$

**return**  $\mathbf{x}$

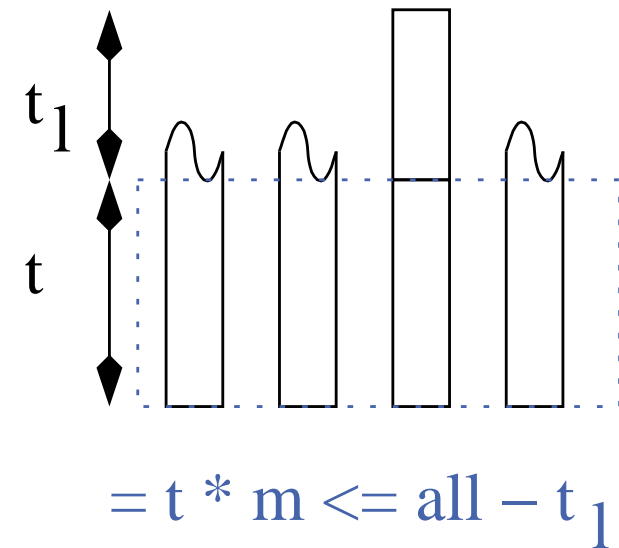
## Viele Kleine Jobs

**Lemma 1.** Falls  $\ell$  der zuletzt beendete Job ist, dann

$$L_{\max} \leq \sum_j \frac{t_j}{m} + \frac{m-1}{m} t_\ell$$

### Beweis

$$L_{\max} = t + t_\ell \leq \sum_{j \neq \ell} \frac{t_j}{m} + t_\ell = \sum_j \frac{t_j}{m} + \frac{m-1}{m} t_\ell$$



## Untere Schranken

**Lemma 2.**  $L_{\max} \geq \sum_j \frac{t_j}{m}$

**Lemma 3.**  $L_{\max} \geq \max_j t_j$

## Der Approximationsfaktor

Definition:

Ein Minimierungsalgorithmus erzielt **Approximationsfaktor**  $\rho$  bezüglich Zielfunktion  $f$  falls er für **alle** Eingaben  $I$ , eine Lösung  $\mathbf{x}(I)$  findet, so dass

$$\frac{f(\mathbf{x}(I))}{f(\mathbf{x}^*(I))} \leq \rho$$

wobei  $\mathbf{x}^*(I)$  die optimale Lösung für Eingabe  $I$  bezeichnet.

**Satz:** ListScheduling erzielt Approximationsfaktor  $2 - \frac{1}{m}$ .

**Beweis:**

$$\frac{f(\mathbf{x})}{f(\mathbf{x}^*)}$$

$$\leq \frac{\sum_j t_j / m}{f(\mathbf{x}^*)} + \frac{m-1}{m} \cdot \frac{t_\ell}{f(\mathbf{x}^*)}$$

$$\leq 1 + \frac{m-1}{m} \cdot \frac{t_\ell}{f(\mathbf{x}^*)}$$

$$\leq 1 + \frac{m-1}{m} = 2 - \frac{1}{m}$$

$$\text{Lemma 1: } L_{\max} \leq \sum_j \frac{t_j}{m} + \frac{m-1}{m} t_\ell$$

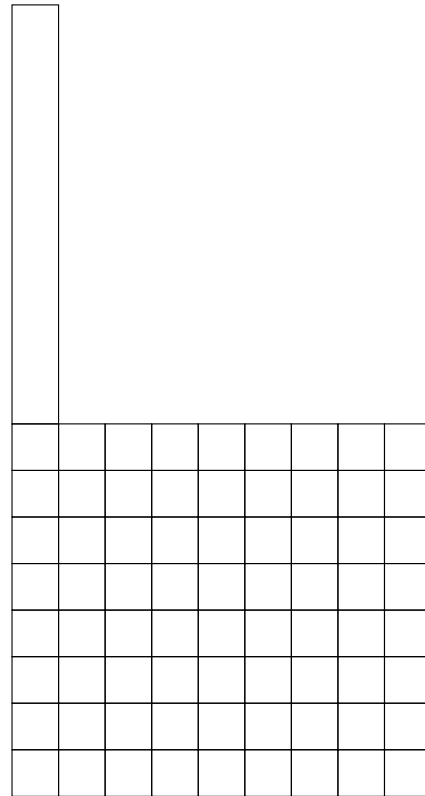
$$\text{Lemma 2: } L_{\max} \geq \sum_j \frac{t_j}{m}$$

$$\text{Lemma 3: } L_{\max} \geq \max_j t_j \geq t_\ell$$

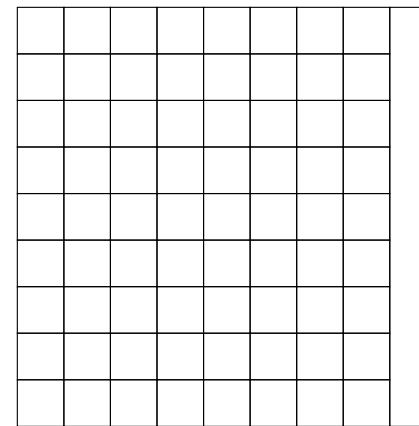


## Diese Schranke ist bestmöglich

Eingabe:  $m(m - 1)$  Jobs der Größe 1 und ein Job der Größe  $m$ .



List Scheduling:  $2m-1$



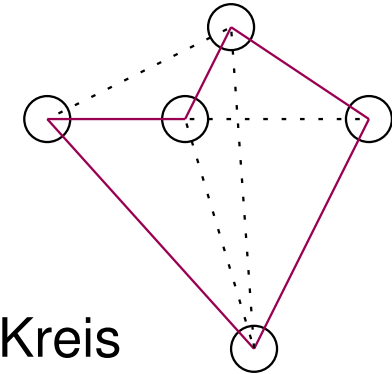
OPT:  $m$

Also ist der Approximationsfaktor  $\geq 2 - 1/m$ .

## Mehr zu Scheduling (siehe Approx-Vorlesung)

- 4/3-Approximation: Sortiere die Jobs nach **absteigender Größe**.  
Dann List-Scheduling. Zeit  $O(n \log n)$ .
- Schnelle 7/6 Approximation: Rate Makespan (binäre Suche).  
Dann **Best Fit Decreasing**.
- PTAS** ... später ...
- Uniform machines: Maschine  $i$  hat **Geschwindigkeit  $v_i$**  job  $j$   
braucht Zeit  $t_j/v_i$  auf Maschine  $j$ .  $\rightsquigarrow$  relative einfache  
Verallgemeinerung
- Unrelated Machines** Job  $j$  braucht Zeit  $t_{ji}$  auf Maschine  $j$ .  
2-Approximation. Ganz anderer Algorithmus.
- uvam: Andere Zielfunktionen, Reihenfolgebeschränkungen, ...

# Nichtapproximierbarkeit des Handlungsreisendenproblems (TSP)



Gegeben ein Graph  $G = (V, V \times V)$ , finde einen einfachen Kreis  $C = (v_1, v_2, \dots, v_n, v_1)$  so dass  $n = |V|$  und  $\sum_{(u,v) \in C} d(u, v)$  minimiert wird.

**Satz:** Es ist NP-hart das TSP innerhalb irgendeines Faktors  $a$  zu approximieren.

**Beweisansatz:** Es genügt zu zeigen, dass

HamiltonCycle  $\leq_p$   $a$ -Approximation von TSP

## $\alpha$ -Approximation von TSP

### Gegeben:

Graph  $G = (V, V \times V)$  mit Kantengewichten  $d(u, v)$ ,  
Parameter  $W$ .

Gesucht ist ein Algorithmus, mit folgenden Eigenschaften:

$[G, W]$  wird akzeptiert  $\longrightarrow \exists$  Tour mit Gewicht  $\leq \alpha W$ .

$[G, W]$  wird abgelehnt  $\longrightarrow \nexists$  Tour mit Gewicht  $\leq W$ .

# HamiltonCycle $\leq_p$ $a$ -Approximation von TSP

Sei  $G = (V, E)$  beliebiger ungerichteter Graph.

Definiere  $d(u, v) = \begin{cases} 1 & \text{falls } (u, v) \in E \\ 1 + an & \text{sonst} \end{cases}$

Dann und nur dann, wenn  $G$  einen Hamiltonkreis hat gilt

$\exists$  TSP Tour mit **Kosten  $n$**

(sonst optimale **Kosten  $\geq (n - 1) \cdot 1 + (an + 1) = an + n > an$** )

Entscheidungsalgorithmus für Hamiltonkreis:

Führe  $a$ -approx TSP auf  $[G, n]$  aus.

Wird akzeptiert

$\longrightarrow \exists$  Tour mit Gewicht  $\leq an$

$\longrightarrow \exists$  Tour mit Gewicht  $n \longrightarrow \exists$  Hamiltonpfad

sonst  $\nexists$  Hamiltonpfad

## TSP mit Dreiecksungleichung

$G$  (ungerichtet) erfüllt die **Dreiecksungleichung**

$$\forall u, v, w \in V : d(u, w) \leq d(u, v) + d(v, w)$$

Satz: für solche Graphen gibt es einen 2-Approximationsalgorithmus

## Metrische Vervollständigung

Betrachte beliebigen unger. Graph  $G = (V, E)$  mit Gewichtsfunktion

$c : E \rightarrow \mathbb{R}_+$ . Definiere

$d(u, v) :=$  Länge des kürzesten Pfades von  $u$  nach  $v$

Beispiel: (ungerichteter) Strassengraph  $\longrightarrow$  Abstandstabelle

## Euler-Touren/-Kreise

Betrachte beliebigen zusammenhängenden ungerichteten (Multi-)Graph  $G = (V, E)$  mit  $|E| = m$ .

Ein Pfad  $P = \langle e_1, \dots, e_m \rangle$  ist ein **Euler-Kreis** falls  $\{e_1, \dots, e_m\} = E$ .  
(Jede **Kante** wird genau einmal besucht)

Satz:  $G$  hat Euler-Kreis gdw.  $G$  ist zusammenhängend und  $\forall v \in V : \text{Grad}(v)$  ist gerade.

Euler-Kreise lassen sich in Zeit  $O(|E| + |V|)$  finden.

## 2-Approximation durch minimalen Spannbaum

### Lemma 4.

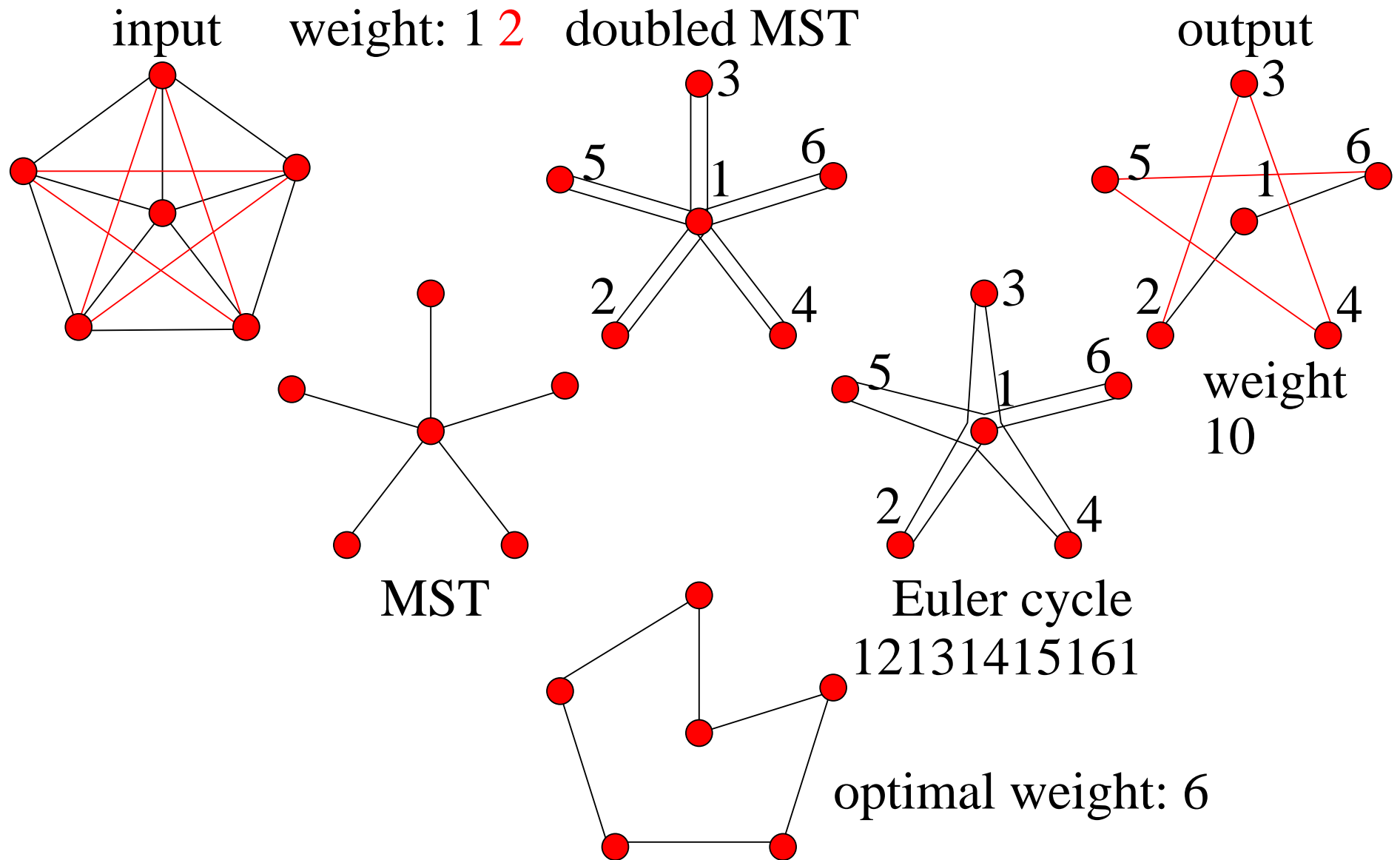
Gesamtgewicht eines *MST*  $\leq$   
Gesamtgewicht jeder *TSP-Tour*

Algorithmus:

$T := \text{MST}(G)$	// $\text{weight}(T) \leq \text{opt}$
$T' := T$ with every edge doubled	// $\text{weight}(T') \leq 2\text{opt}$
$T'' := \text{EulerKreis}(T')$	// $\text{weight}(T'') \leq 2\text{opt}$
output $\text{removeDuplicates}(T'')$	// <b>shortcutting</b>



# Beispiel



## **Beweis von $\text{Gewicht MST} \leq \text{Gewicht TSP-Tour}$**

Sei  $T$  die optimale TSP tour

entferne eine Kante

macht  $T$  leichter

nun ist  $T$  ein Spannbaum

der nicht leichter sein kann als der

MST



## **Allgemeine Technik: Relaxation**

hier: ein TSP-Pfad ist ein Spezialfall eines Spannbaums

## Mehr TSP I

- Praktisch bessere 2-Approximationen, z.B. lightest edge first
- Relativ einfache aber unpraktische  $3/2$ -Approximation  
(MST + min. weight perfect matching + Euler-Kreis) Christofides76
- $3/2 - 10^{36}$  approximation KarlinKleinGharan21
- $117/116$  Approximation ist NP-hart
- 1.4 Approximation wenn Abstände kürzeste Wege in einem ungewichteten Graphen sind.
- PTAS für **Euklidisches TSP**

## Mehr TSP II

- Versuchskanichen für praktisch jede Optimierungsheuristik
- Optimale Lösungen für praktische Eingaben. Faustregel:  
Falls es in den Speicher passt, läßt sich lösen.  
`[http://www.tsp.gatech.edu/concorde.html]`  
sechsstellige Anzahl Codezeilen.
- TSP-artige Anwendungen sind meist komplizierter

# Pseudopolynomielle Algorithmen

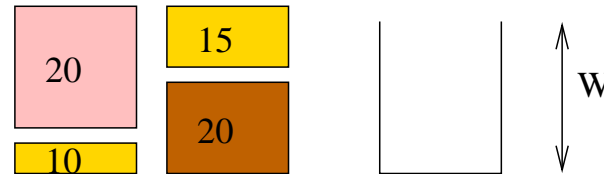
$\mathcal{A}$  ist **pseudopolynomieller** Algorithmus falls

$$\text{Time}_{\mathcal{A}}(n) \in \mathbf{P}(n)$$

wobei  $n$  die Anzahl Eingabebits ist,

wenn alle Zahlen **unär** codiert werden ( $k \equiv 1^k$ ).

# Beispiel Rucksackproblem



- $n$  Gegenstände mit **Gewicht**  $w_i \in \mathbb{N}$  und **profit**  $p_i$   
oBdA:  $\forall i \in 1..n : w_i \leq W$
- Wähle eine Teilmenge **x** von Gegenständen
- so dass  $\sum_{i \in \mathbf{x}} w_i \leq W$  und
- maximiere den Profit**  $\sum_{i \in \mathbf{x}} p_i$

# Dynamische Programmierung **nach Profit**

$C(i, P)$  := kleinste Kapazität für Gegenstände  $1, \dots, i$  die Profit  $\geq P$  ergeben.

## Lemma 5.

$$\forall 1 \leq i \leq n : C(i, P) = \min(C(i-1, P), \\ C(i-1, P - p_i) + w_i)$$

# Dynamische Programmierung **nach Profit**

Sei  $\hat{P}$  obere Schranke für den Profit (z.B.  $\sum_i p_i$ ).

**Zeit:**  $O(n\hat{P})$  **pseudo**-polynomiell

z.B.  $0..n \times 0..\hat{P}$  Tabelle  $C(i, P)$  spaltenweise ausfüllen

**Platz:**  $\hat{P} + O(n)$  Maschinenworte plus  $\hat{P}n$  bits.



# Fully Polynomial Time Approximation Scheme

Algorithm  $\mathcal{A}$  ist ein

(Fully) Polynomial Time Approximation Scheme

für  $\begin{matrix} \text{minimization} \\ \text{maximization} \end{matrix}$  Problem  $\Pi$  falls:

Eingabe: Instanz  $I$ , Fehlerparameter  $\varepsilon$

Ausgabequalität:  $f(\mathbf{x}) \begin{matrix} \leq \\ \geq \end{matrix} \begin{pmatrix} 1+\varepsilon \\ 1-\varepsilon \end{pmatrix} \text{opt}$

Zeit: Polynomiell in  $|I|$  (und  $1/\varepsilon$ )

# Beispielschranken

PTAS	FPTAS
$n + 2^{1/\varepsilon}$	$n^2 + \frac{1}{\varepsilon}$
$n^{\log \frac{1}{\varepsilon}}$	$n + \frac{1}{\varepsilon^4}$
$n^{\frac{1}{\varepsilon}}$	$n/\varepsilon$
$n^{42/\varepsilon^3}$	$\vdots$
$n + 2^{2^{1000/\varepsilon}}$	$\vdots$
$\vdots$	$\vdots$

# FPTAS für Knapsack

$$P := \max_i p_i$$

// maximaler Einzelprofit

$$K := \frac{\varepsilon P}{n}$$

// Skalierungsfaktor

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

// skaliere Profite

$$\mathbf{x}' := \text{dynamicProgrammingByProfit}(\mathbf{p}', \mathbf{w}, C)$$

gib  $\mathbf{x}'$  aus

**Lemma 6.**  $\mathbf{p} \cdot \mathbf{x}' \geq (1 - \varepsilon)\text{opt}$ .

*Proof.* Betrachte die optimale Lösung  $\mathbf{x}^*$ .

$$\begin{aligned} \mathbf{p} \cdot \mathbf{x}^* - K\mathbf{p}' \cdot \mathbf{x}^* &= \sum_{i \in \mathbf{x}^*} \left( p_i - K \left\lfloor \frac{p_i}{K} \right\rfloor \right) \\ &\leq \sum_{i \in \mathbf{x}^*} \left( p_i - K \left( \frac{p_i}{K} - 1 \right) \right) = |\mathbf{x}^*|K \leq nK, \end{aligned}$$

also,  $K\mathbf{p}' \cdot \mathbf{x}^* \geq \mathbf{p} \cdot \mathbf{x}^* - nK$ . Weiterhin,

$$K\mathbf{p}' \cdot \mathbf{x}^* \leq K\mathbf{p}' \cdot \mathbf{x}' = \sum_{i \in \mathbf{x}'} K \left\lfloor \frac{p_i}{K} \right\rfloor \leq \sum_{i \in \mathbf{x}'} K \frac{p_i}{K} = \mathbf{p} \cdot \mathbf{x}'. \text{ Also,}$$

$$\mathbf{p} \cdot \mathbf{x}' \geq K\mathbf{p}' \cdot \mathbf{x}^* \geq \mathbf{p} \cdot \mathbf{x}^* - nK = \text{opt} - \varepsilon \underbrace{P}_{\leq \text{opt}} \geq (1 - \varepsilon)\text{opt}$$

□

**Lemma 7.** Laufzeit  $O(n^3 / \epsilon)$ .

*Proof.* Die Laufzeit  $O(n\hat{P}')$  der dynamischen Programmierung dominiert:

$$n\hat{P}' \leq n \cdot (n \cdot \max_{i=1}^n p'_i) = n^2 \left\lfloor \frac{P}{K} \right\rfloor = n^2 \left\lfloor \frac{Pn}{\epsilon P} \right\rfloor \leq \frac{n^3}{\epsilon}.$$

□

# Das beste bekannte FPTAS

[Kellerer, Pferschy 04]

$$O\left(\min\left\{n \log \frac{1}{\varepsilon} + \frac{\log^2 \frac{1}{\varepsilon}}{\varepsilon^3}, \dots\right\}\right)$$

- Weniger buckets  $C_j$  (nichtuniform)
- Ausgefeilte dynamische Programmierung

# Optimale Algorithmen für das Rucksackproblem

Annähernd Linearzeit für fast alle Eingaben! In Theorie und Praxis.

[Beier, Vöcking, An Experimental Study of Random Knapsack Problems, European Symposium on Algorithms, 2004.]

[Kellerer, Pferschy, Pisinger, Knapsack Problems, Springer 2004.]