Algorithmen / Algorithms II

Peter Sanders

Exercise:

Daniel Seemaier, Tobias Heuer

Institute of Theoretical Informatics

Web:

`http://algo2.iti.kit.edu/AlgorithmenII_WS20.php`

8 Approximation Algorithms

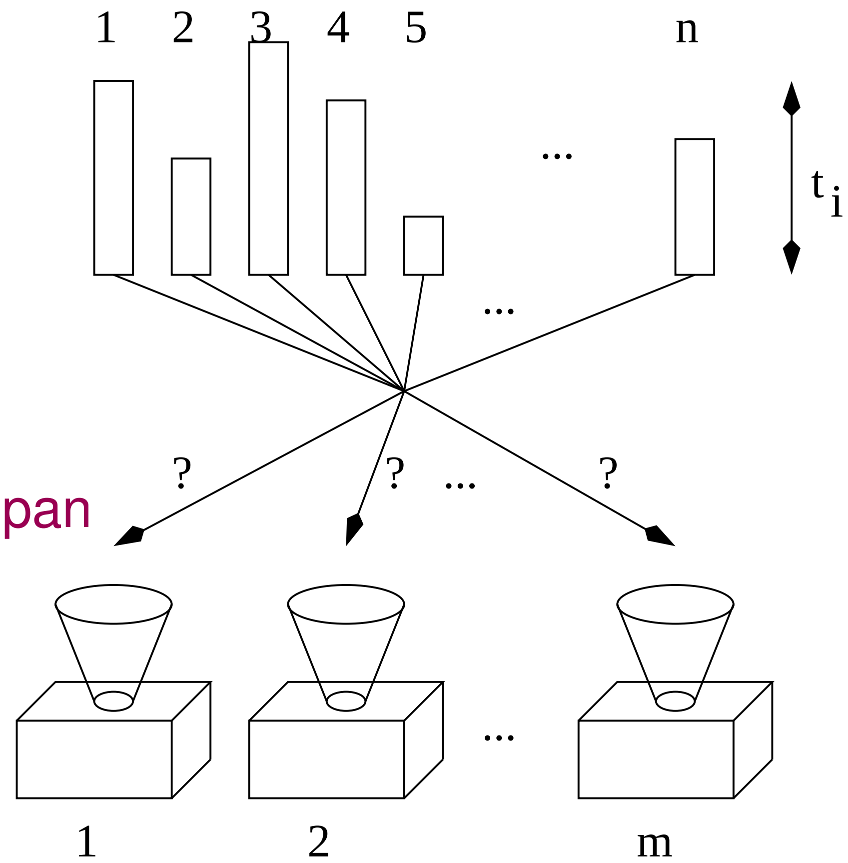
A possibility to tackle NP-hard problems

Observation: Almost all interesting optimization problems are NP-hard

Options:

- Still try to find an optimal solution but risk that the algorithm doesn't finish
- Ad-hoc heuristics. Will find a solution but how good is it?
- Approximation algorithms:
  Polynomial running time.
  Solutions guaranteed to be "close" to optimal.
- Redefine/specialize Problem...

Scheduling of independet weighted jobs on parallel machines



$\mathbf{x}(j)$: machine that runs Job j

L_i: $\sum_{\mathbf{x}(j)=i} t_j$, Load of machine i

Objective function: Minimize makespan

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

Details: identical machines, independent jobs,

known running times, offline

List Scheduling

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

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

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

  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

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

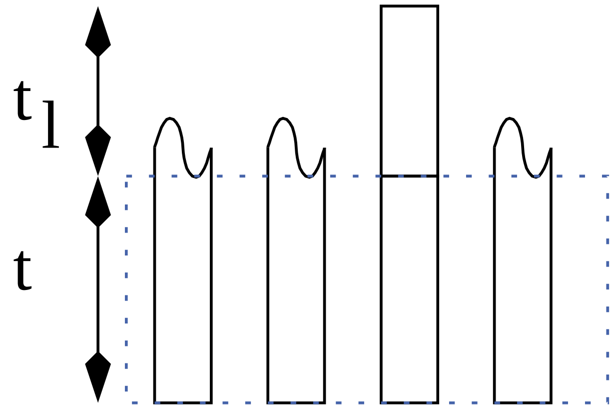
  return $\mathbf{x}$

Many small jobs

Lemma 1. *If ℓ is the job that finishes last, then*

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

Proof



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

= t * m <= all − t l

Lower bounds

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

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

The approximation ratio

Definition:

A minimization algorithms achieves approximation ratio ρ with respect to a objective function f if for all inputs I, it finds a solution $\mathbf{x}(I)$, such that

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

where $\mathbf{x}^*(I)$ is the optimal solution for input I.

Theorem: ListScheduling achieves approximation ratio $2 - \frac{1}{m}$.

Proof:

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

(upper bound Lemma 1)

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

(lower bound Lemma 2)

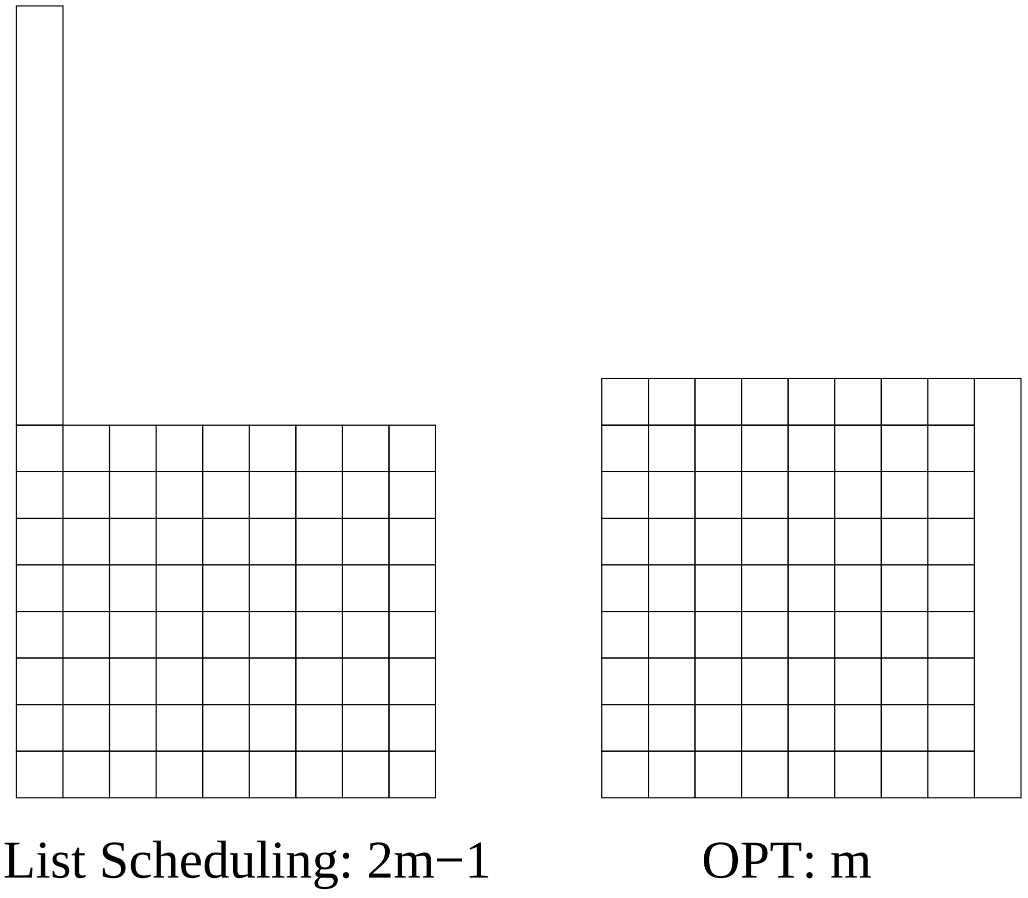
$$\leq 1 + \frac{m-1}{m} \cdot \frac{t_\ell}{f(\mathbf{x}^*)}$$

(lower bound Lemma 3)

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

This bound is optimal

Input: $m(m-1)$ jobs of size 1 and one job of size m.

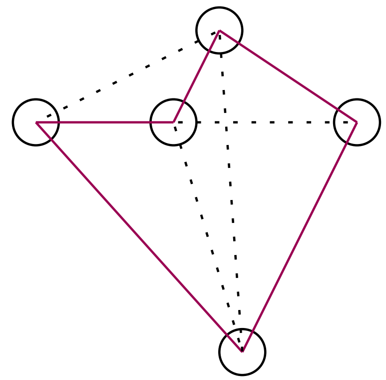


Therefore, the approximation ratio is $\geq 2-1/m$.

More About Scheduling)

- $4/3$ approximation: Sort jobs decreasing by size.
  Then list scheduling. Time $\mathrm{O}(n \log n)$.
- Fast $7/6$ approximation: Guess makespan (binary search). then
  Best Fit Decreasing.
- PTAS ... later ...
- Uniform machines: Machine i has speed v_i, job j needs time t_j/v_i
  on machine j. $\rightsquigarrow$ relatively easy generalization
- Unrelated machines: Job j needs time t_{ji} on machine j.
  2 approximation. Very different algorithm.
- And many more: different objective functions, order restrictions, ...

Inapproximability of the Traveling Salesman Problem (TSP)



Given a graph $G = (V, V \times V)$, find a simple cycle $C = (v_1, v_2, \ldots, v_n, v_1)$ such that $n = |V|$ and $\sum_{(u,v)\in C} d(u,v)$ is minimized.

Theorem: Approximate TSP to any ratio a is NP-hard.

Proof idea: It is sufficient to show that

HamiltonCycle$\leq_p$ a-approximation of TSP

a-Approximation of TSP

Given:

Graph $G = (V, V \times V)$ with edge weights $d(u,v)$,
parameter W.

We need an algorithm with the following properties:

$[G,W]$ is accepted $\longrightarrow \exists$ tour with weight $\leq aW$.

$[G,W]$ is rejected $\longrightarrow \nexists$ tour with weight $\leq W$.

HamiltonCycle$\leq_p$ a Approximation of TSP

Let $G=(V,E)$ an arbitrary undirected graph.

Define $d(u,v) = \begin{cases} 1 & \text{if } (u,v) \in E \\ 1+an & \text{else} \end{cases}$

$\exists$ TSP tour with cost n

If and only if G has a Hamiltonian cycle

(otherwise: optimal cost $\geq (n-1)\cdot 1 + (an+1) = an+n > an$)

Decision algorithms for Hamiltonian cycle:

Run a approx TSP on $[G,n]$.

Is accepted

$\longrightarrow \exists$ tour with weight $\leq an$

$\longrightarrow \exists$ tour with weight $n \longrightarrow \exists$ Hamiltonian path

otherwise $\nexists$ Hamiltonian path

TSP with Triangle Inequality

G (undirected) satisfies triangle inequality

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

Metric completion

Consider arbitrary undirected graph $G = (V,E)$ with weight function $c : E \to \mathbb{R}_+$. Define

$d(u,v) :=$ Length of shortest path from u to v

Example: (undirected) road graph $\longrightarrow$ distance table

Eulerian Path/Cycle

Consider arbitrary connected undirected (multi-)graph $G = (V, E)$ with $|E| = m$.

A path $P = \langle e_1, \ldots, e_m \rangle$ is called a Eulerian cycle if $\{e_1, \ldots, e_m\} = E$. (every edge is visited exactly once)

Theorem: G has Eulerian cycle iff G is connected and $\forall v \in V$:degree(v) is even.

Eulerian cycles can be found in time $\mathrm{O}(|E| + |V|)$.

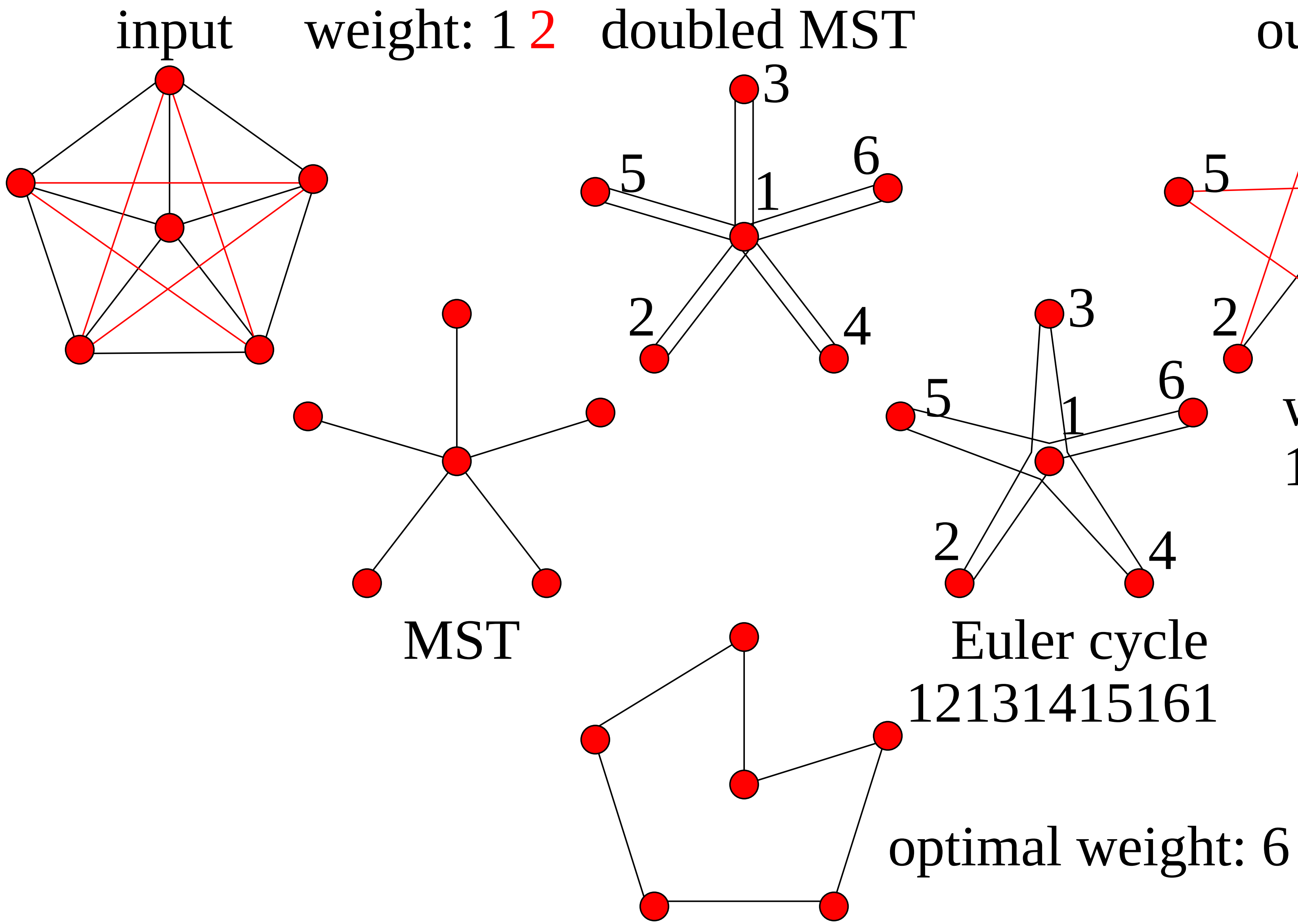
2 Approximation by Minimum Spanning Tree

Lemma 4.
Total weight of an MST ≤
Total weight of every TSP tour

Algorithm:

T:= MST(G) // weight(T) $\leq$opt
T':= T with every edge doubled // weight(T') $\leq$2opt
T'':= EulerianCycle(T') // weight(T'') $\leq$ 2opt
output removeDuplicates(T'') // shortcutting

Example


input
weight: 1 2
doubled MST
output
3
5
6
1
2
4
MST
Euler cycle
12131415161
weight
10
optimal weight: 6

Proof of Weight MST$\leq$ Weight TSP tour

Let T be the optimal TSP tour.

Removing an edge makes T lighter.

Now T is a spanning tree

that cannot be lighter than the MST

$\blacksquare$

General technique: Relaxation

here: a TSP path is a special case of a spanning tree

More TSP

- In practice better 2 approximations, e.g. lightest edge first
- Relatively easy but inpractical 3/2 approximation (MST+min. weight perfect matching+Eulerian cycle)
- PTAS for Euclidean TSP
- Guinea pig for virtually every optimization heuristic
- Optimal solutions for practical inputs. Rule of thumb: If it fits into memory, you can solve it. [http://www.tsp.gatech.edu/concorde.html] Six-figure number of code lines.
- TSP-like applications are usually more complicated

Pseudo- Polynomial Time Algorithms

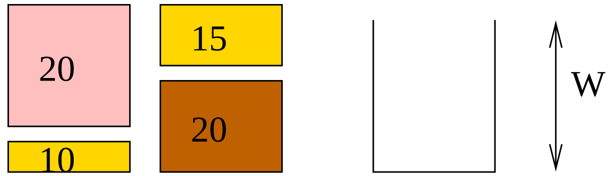
$\mathscr{A}$ is pseudo-polynomial time algorithms if

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

where n is the number of input bits,

if all numbers are in unary coding ($k \equiv 1^k$).

Example: Knapsack Problem



- n items with weight $w_i \in \mathbb{N}$ and value p_i.
  Wlog: $\forall i \in 1..n : w_i \leq W$
- Choose a subset $\mathbf{x}$ of items
- Such that $\sum_{i \in \mathbf{x}} w_i \leq W$ and
- Maximize the value $\sum_{i \in \mathbf{x}} p_i$

Dynamic Programming by Value

$C(i,P)$:= smallest capacity for items $1,\ldots,i$ that add up to value $\geq P$.

Lemma 5.

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

Dynamic programming by value

Let $\hat{P}$ be an upper bound for the value (e.g. $\sum_i p_i$).

Time: $\mathrm{O}\left(n\hat{P}\right)$ pseudo-polynomial

e.g. fill $0..n \times 0..\hat{P}$ table $C(i,P)$ column-wise

Space: $\hat{P} + \mathrm{O}(n)$ machine words plus $\hat{P}n$ bits.

Fully Polynomial Time Approximation Scheme

Algorithm $\mathscr{A}$ is a

(Fully) Polynomial Time Approximation Scheme

for $\begin{matrix}\text{minimization}\\ \text{maximization}\end{matrix}$ problem Π if:

Input: Instance I, error parameter ε

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

Time: Polynomial in $|I|$ (and $1/\varepsilon$)

Examples for bounds

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/ε
n^{42/ε^3}	$\vdots$
$n + 2^{2^{1000/\varepsilon}}$	$\vdots$
$\vdots$	$\vdots$

FPTAS for Knapsack

$P := \max_i p_i$ // maximum single value

$K := \dfrac{\varepsilon P}{n}$ // scaling factor

$p'_i := \left\lfloor \frac{p_i}{K} \right\rfloor$ // scaled values

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

return $\mathbf{x}'$

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

Proof. Consider the optimal solution $\mathbf{x}^*$.

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

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

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

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

□

Lemma 7. *Running time* $\mathrm{O}\left(n^3/\varepsilon\right)$.

Proof. The running time $\mathrm{O}\left(n\hat{P}'\right)$ of dynamic programming dominates:

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

$\square$

The Best Known FPTAS

[Kellerer, Pferschy 04]

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

- Fewer buckets C_j (non-uniform)
- Sophisticated dynamic programming

Optimal Algorithms for the Knapsack Problem

Near linear running time for almost all inputs! In theory and practice.

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

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