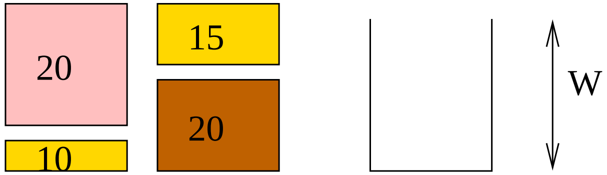
The Knapsack Problem



- n items with weight $w_i \in \mathbb{N}$ and profit $p_i \in \mathbb{N}$
- Choose a subset $\mathbf{x}$ of items
- Capacity constraint $\sum_{i \in \mathbf{x}} w_i \leq W$
  wlog assume $\sum_i w_i > W$, $\forall i : w_i < W$
- Maximize profit $\sum_{i \in \mathbf{x}} p_i$

Reminder?: Linear Programming

Definition 1. *A linear program with n variables and m constraints is specified by the following minimization problem*

- *Cost function* $f(\mathbf{x}) = \mathbf{c} \cdot \mathbf{x}$
  $\mathbf{c}$ *is called the cost vector*

- *m constraints of the form* $\mathbf{a}_i \cdot \mathbf{x} \bowtie_i b_i$ *where* $\bowtie_i \in \{\leq, \geq, =\}$, $\mathbf{a}_i \in \mathbb{R}^n$ *We have*

$$\mathcal{L} = \{\mathbf{x} \in \mathbb{R}^n : \forall 1 \leq i \leq m : x_i \geq 0 \wedge \mathbf{a}_i \cdot \mathbf{x} \bowtie_i b_i\} \quad .$$

Let a_{ij} *denote the* j*-th component of vector* $\mathbf{a}_i$*.*

Complexity

Theorem 1. *A linear program can be solved in polynomial time.*

- Worst case bounds are rather high
- The algorithm used in practice might take exponential worst case time
- Reuse is not only possible but almost necessary because a robust, efficient implementation is quite ComPLEX.

Integer Linear Programming

ILP: Integer Linear Program, A linear program with the additional constraint that all the $x_i \in \mathbb{N}$

Linear Relaxation: Remove the integrality constraints from an ILP

Example: The Knapsack Problem

$$\text{maximize } \mathbf{p} \cdot \mathbf{x}$$

subject to

$$\mathbf{w} \cdot \mathbf{x} \leq W, x_i \in \{0,1\} \text{ for } 1 \leq i \leq n \quad .$$

$x_i = 1$ iff item i is put into the knapsack.

0/1 variables are typical for ILPs

How to Cope with ILPs

– Solving ILPs is NP-hard

+ Powerful modeling language

+ There are generic methods that sometimes work well

+ Many ways to get approximate solutions.

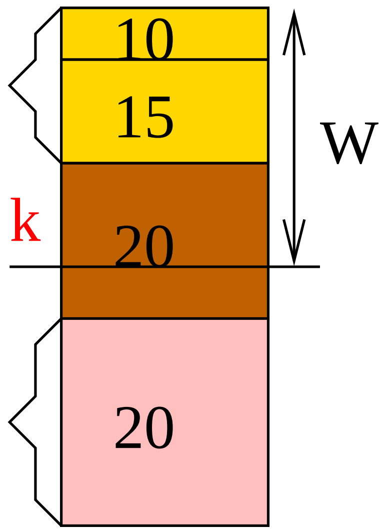
+ The solution of the integer relaxation helps. For example sometimes we can simply round.

Linear Time Algorithm for Linear Relaxation of Knapsack

Classify elements by profit density $\frac{p_i}{w_i}$ into $B, \{k\}, S$ such that

$$\forall i \in B, j \in S : \frac{p_i}{w_i} \geq \frac{p_k}{w_k} \geq \frac{p_j}{w_j}, \text{ and,}$$

$$\sum_{i \in B} w_i \leq W \text{ but } w_k + \sum_{i \in B} w_i > W \ .$$



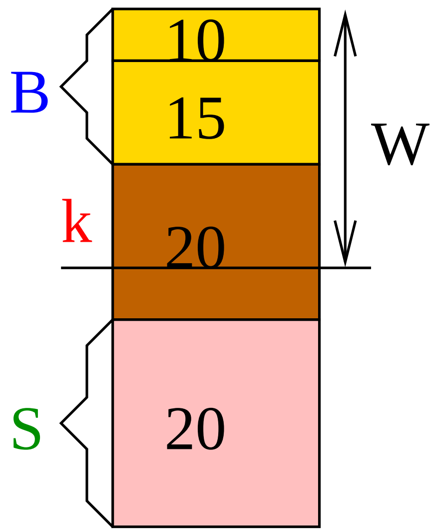
$$\text{Set } x_i = \begin{cases} 1 & \text{if } i \in B \\ \frac{W - \sum_{i \in B} w_i}{w_k} & \text{if } i = k \\ 0 & \text{if } i \in S \end{cases}$$

$$x_i = \begin{cases} 1 & \text{if } i \in B \\ \frac{W - \sum_{i \in B} w_i}{w_k} & \text{if } i = k \\ 0 & \text{if } i \in S \end{cases}$$

Lemma 2. **x** *is the optimal solution of the linear relaxation.*

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

- $\mathbf{w} \cdot \mathbf{x}^* = W$ otherwise increase some x_i
- $\forall i \in B : x_i^* = 1$ otherwise
  increase x_i^* and decrease some x_j^* for $j \in \{k\} \cup S$
- $\forall j \in S : x_j^* = 0$ otherwise
  decrease x_j^* and increase x_k^*
- This only leaves $x_k = \frac{W - \sum_{i \in B} w_i}{w_k}$

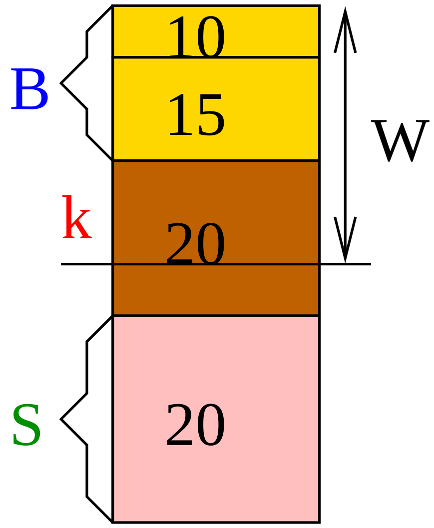


□

$$x_i = \begin{cases} 1 & \text{if } i \in B \\ \frac{W - \sum_{i \in B} w_i}{w_k} & \text{if } i = k \\ 0 & \text{if } i \in S \end{cases}$$

Lemma 3. *For the optimal solution* **x** *of the linear relaxation:*

$$\text{opt} \le \sum_i x_i p_i \le 2\text{opt}$$



Proof. We have

$\sum_{i \in B} p_i \le$ opt. Furthermore, since $w_k < W$,

$p_k \le$ opt.

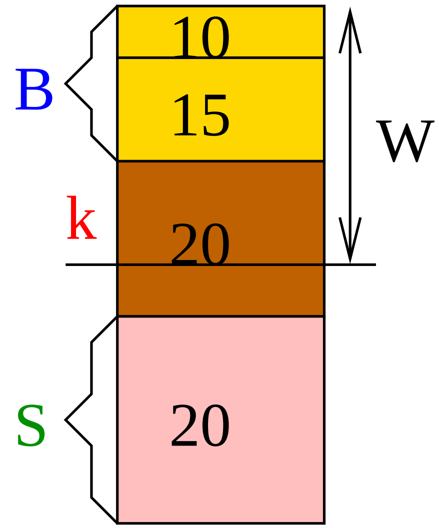
We get

$$\text{opt} \le \sum_i x_i p_i \le \sum_{i \in B} p_i + p_k \le \text{opt} + \text{opt} = 2\text{opt}$$

□

Two-approximation of Knapsack

$$x_i = \begin{cases} 1 & \text{if } i \in B \\ \frac{W - \sum_{i \in B} w_i}{w_k} & \text{if } i = k \\ 0 & \text{if } i \in S \end{cases}$$



Exercise: Prove that either B or $\{k\}$ is a 2-approximation of the (nonrelaxed) knapsack problem.

Dynamic Programming

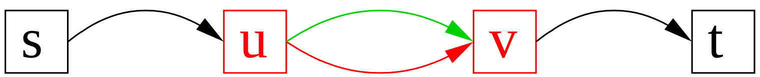
— Building it Piece By Piece

Principle of Optimality

- An optimal solution can be viewed as constructed of optimal solutions for subproblems
- Solutions with the same objective values are interchangeable

Example: Shortest Paths

- Any subpath of a shortest path is a shortest path
- Shortest subpaths are interchangeable



Dynamic Programming by Capacity for the Knapsack Problem

Define

$P(i,C)$ = optimal profit from items $1,\ldots,i$ using capacity $\leq C$.

Lemma 4.

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

Proof

$P(i,C) \geq P(i-1,C)$: Set $x_i = 0$, use optimal subsolution.

$P(i,C) \geq P(i-1,C-c_i)+p_i$: Set $x_i = 1$ ...

$P(i,C) \leq \max(P(i-1,C), P(i-1,C-c_i)+p_i)$

Assume the contrary :

$\exists \mathbf{x}$ that is optimal for the subproblem such that

$$\begin{aligned} P(i-1,C) &< \mathbf{p}\cdot\mathbf{x} \wedge \\ P(i-1,C-c_i)+p_i &< \mathbf{p}\cdot\mathbf{x} \end{aligned}$$

Case $x_i = 0$: $\mathbf{x}$ is also feasible for $P(i-1,C)$. Hence, $P(i-1,C) \geq \mathbf{p}\cdot\mathbf{x}$. Contradiction

Case $x_i = 1$: Setting $x_i = 0$ we get a feasible solution $\mathbf{x}'$ for $P(i-1,C-c_i)$ with profit $\mathbf{p}\cdot\mathbf{x}' = \mathbf{p}\cdot\mathbf{x} - p_i$. Hence, $P(i-1,C-c_i)+p_i \geq \mathbf{p}\cdot\mathbf{x}$. Contradiction

Computing $P(i,C)$ bottom up:

Procedure knapsack(**p**, **c**, n, W)
  array $\mathrm{P}[0\ldots W] = [0,\ldots,0]$
  bitarray $\mathrm{decision}[1\ldots n, 0\ldots W] = [(0,\ldots,0),\ldots,(0,\ldots,0)]$
  for i := 1 **to** n **do**
    // invariant: $\forall C \in \{1,\ldots,W\} : \mathrm{P}[C] = P(i-1,C)$
    for C := W **downto** w_i **do**
      if $\mathrm{P}[C-c_i] + p_i > \mathrm{P}[C]$ **then**
        $\mathrm{P}[C] := \mathrm{P}[C-c_i] + p_i$
        $\mathrm{decision}[i,C] := 1$

Recovering a Solution

$C := W$

array $\mathbf{x}[1 \ldots n]$

for $i := n$ **downto** 1 **do**

  $\mathbf{x}[i] := \text{decision}[i, C]$

  if $\mathbf{x}[i] = 1$ **then** $C := C - w_i$

endfor

return x

Analysis:

Time: $O(nW)$ pseudo-polynomial

Space: $W + O(n)$ words plus Wn bits.

Example: A Knapsack Instance

maximize $(10,20,15,20)\cdot\mathbf{x}$

subject to $(1,3,2,4)\cdot\mathbf{x}\leq 5$

$P(i,C),(\text{decision}[i,C])$

$i\setminus C$	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0, (0)	10, (1)	10, (1)	10, (1)	10, (1)	10, (1)
2						
3						
4						

Example: A Knapsack Instance

maximize $(10, 20, 15, 20) \cdot \mathbf{x}$

subject to $(1, 3, 2, 4) \cdot \mathbf{x} \leq 5$

$P(i, C), (\text{decision}[i, C])$

$i \setminus C$	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0, (0)	10, (1)	10, (1)	10, (1)	10, (1)	10, (1)
2	0, (0)	10, (0)	10, (0)	20, (1)	30, (1)	30, (1)
3						
4						

Example: A Knapsack Instance

maximize $(10,20,15,20)\cdot\mathbf{x}$

subject to $(1,3,2,4)\cdot\mathbf{x}\leq 5$

$P(i,C),(\text{decision}[i,C])$

$i\setminus C$	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0, (0)	10, (1)	10, (1)	10, (1)	10, (1)	10, (1)
2	0, (0)	10, (0)	10, (0)	20, (1)	30, (1)	30, (1)
3	0, (0)	10, (0)	15, (1)	25, (1)	30, (0)	35, (1)
4						

Example: A Knapsack Instance

maximize $(10, 20, 15, 20) \cdot \mathbf{x}$

subject to $(1, 3, 2, 4) \cdot \mathbf{x} \leq 5$

$P(i, C), (\text{decision}[i, C])$

$i \setminus C$	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0, (0)	10, (1)	10, (1)	10, (1)	10, (1)	10, (1)
2	0, (0)	10, (0)	10, (0)	20, (1)	30, (1)	30, (1)
3	0, (0)	10, (0)	15, (1)	25, (1)	30, (0)	35, (1)
4	0, (0)	10, (0)	15, (0)	25, (0)	30, (0)	35, (0)

Dynamic Programming by Profit for the Knapsack Problem

Define

$C(i,P)$ = smallest capacity from items $1,\ldots,i$ giving profit $\geq P$.

Lemma 5.

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

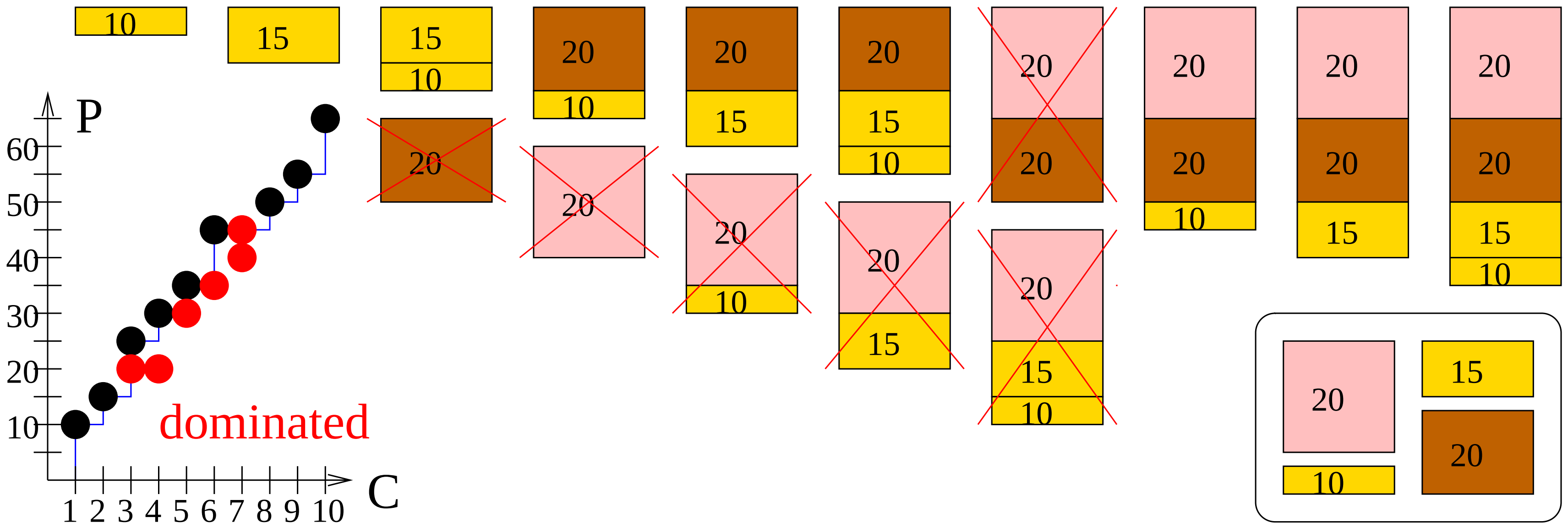
Dynamic Programming by Profit

Let $\hat{P} := \lfloor \mathbf{p} \cdot \mathbf{x}^* \rfloor$ where x^* is the optimal solution of the linear relaxation.

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

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

A Bicriteria View on Knapsack



- n items with weight $w_i \in \mathbb{N}$ and profit $p_i \in \mathbb{N}$
- Choose a subset **x** of items
- Minimize total weight $\sum_{i \in \mathbf{x}} w_i$
- Maximize total profit $\sum_{i \in \mathbf{x}} p_i$

Problem: How should we model the tradeoff?

Pareto Optimal Solutions

[Vilfredo Frederico Pareto (gebürtig Wilfried Fritz)
* 15. Juli 1848 in Paris, † 19. August 1923 in Céligny]

Solution $\mathbf{x}$ dominates solution $\mathbf{x}'$ iff

$$\mathbf{p}\cdot\mathbf{x} \geq \mathbf{p}\cdot\mathbf{x}' \wedge \mathbf{c}\cdot\mathbf{x} \leq \mathbf{c}\cdot\mathbf{x}'$$

and one of the inequalities is proper.

Solution $\mathbf{x}$ is Pareto optimal if

$$\nexists\mathbf{x}' : \mathbf{x}' \text{ dominates } \mathbf{x}$$

Natural Question: Find all Pareto optimal solutions.

In General

- d objectives
- various problems
- various objective functions
- arbitrary mix of minimization and maximization

Enumerating only Pareto Optimal Solutions

[Nemhauser Ullmann 69]

$\mathcal{L} := \langle(0,0)\rangle$ // invariant: $\mathcal{L}$ is sorted by weight and profit

for $i := 1$ **to** n **do**

$\mathcal{L}' := \langle(w+w_i, p+p_i) : (w,p) \in \mathcal{L}\rangle$ // time $O(|\mathcal{L}|)$

$\mathcal{L} := \mathsf{merge}(\mathcal{L}, \mathcal{L}')$ // time $O(|\mathcal{L}|)$

scan $\mathcal{L}$ and eliminate dominated solutions // time $O(|\mathcal{L}|)$

- Now we easily lookup optimal solutions for various constraints on C or P
- We can prune $\mathcal{L}$ if a constraint is known beforehand

Example

Items: $(1,10)$, $(3,20)$, $(2,15)$, $(4,20)$, prune at $W = 5$

```
        L =<(0,0)>
(1,10)->L'=      <(1,10)>
merge ->L =<(0,0),(1,10)>
(3,20)->L'=             <(3,20),(4,30)>
merge ->L =<(0, 0),(1,10),(3,20),(4,30)>
(2,15)->L'=          <(2,15),(3,25),(5,35)>
merge ->L=<(0, 0),(1,10),(2,15),(3,25),(4,30),(5,35)>
(4,20)->L'=                            <(4,20),(5,30)>
merge ->L=<(0, 0),(1,10),(2,15),(3,25),(4,30),(5,35)>
```

Fully Polynomial Time Approximation Scheme

Algorithm $\mathcal{A}$ 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: $f(\mathbf{x}) \begin{matrix}\leq\\ \geq\end{matrix} \left(\begin{matrix}1+\varepsilon\\ 1-\varepsilon\end{matrix}\right)\text{opt}$

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

Example 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 profit

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

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

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

output $\mathbf{x}'$

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

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

$$\mathbf{p} \cdot \mathbf{x}^* - K\mathbf{p}' \cdot \mathbf{x}^* = \sum_{i \in \mathbf{x}^*} \left(p_i - K \left\lfloor \frac{p_i}{K} \right\rfloor \right)$$

$$\leq \sum_{i \in \mathbf{x}^*} \left(p_i - K \left(\frac{p_i}{K} - 1 \right) \right) = |x^*| K \leq nK,$$

i.e., $K\mathbf{p}' \cdot \mathbf{x}^* \geq \mathbf{p} \cdot \mathbf{x}^* - nK$. Furthermore,

$$K\mathbf{p}' \cdot \mathbf{x}^* \leq K\mathbf{p}' \cdot \mathbf{x}' = \sum_i K \left\lfloor \frac{p_i}{K} \right\rfloor \leq \sum_i K \frac{p_i}{K} = \mathbf{p} \cdot \mathbf{x}'.$$ Hence,

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

□

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

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

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

□

A Faster FPTAS for Knapsack

Simplifying assumptions:

$1/\varepsilon \in \mathbb{N}$: Otherwise $\varepsilon := 1/\lceil 1/\varepsilon \rceil$.

Upper bound $\hat{P}$ is known: Use linear relaxation.

$\min_i p_i \geq \varepsilon \hat{P}$: Treat small profits separately. For these items greedy works well. (Costs a factor $O(\log(1/\varepsilon))$ time.)

A Faster FPTAS for Knapsack

$M := \frac{1}{\varepsilon^2}$; $\quad K := \hat{P}\varepsilon^2$

$p'_i := \left\lfloor \frac{p_i}{K} \right\rfloor$ // $p'_i \in \{1, \ldots, M\}$

$C_j := \{i \in 1..n : p'_i = j\}$

remove all but the $\left\lfloor \frac{M}{j} \right\rfloor$ lightest items from C_j

do dynamic programming on the remaining items

Lemma 8. $\mathbf{px}' \geq (1-\varepsilon)\mathrm{opt}$.

Proof. Similar as before, note that $|\mathbf{x}| \leq 1/\varepsilon$ for any solution. □

Lemma 9. *Running time* $O(n + \mathrm{Poly}(1/\varepsilon))$.

Proof.

preprocessing time: $O(n)$

values: $M = 1/\varepsilon^2$

pieces: $\sum_{i=1/\varepsilon}^{M} \left\lfloor \frac{M}{j} \right\rfloor \le M \sum_{i=1/\varepsilon}^{M} \frac{1}{j} \le M \ln M = O\left(\frac{\log(1/\varepsilon)}{\varepsilon^2}\right)$

time dynamic programming: $O\left(\frac{\log(1/\varepsilon)}{\varepsilon^4}\right)$

□

The Best Known FPTAS

[Kellerer, Pferschy 04]

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

- Less buckets C_j (nonuniform)
- Sophisticated dynamic programming

Optimal Algorithm for the Knapsack Problem

The best work in near linear time for almost all inputs! Both in a probabilistic and in a practical sense.

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

Main additional tricks:

- reduce to core items with good profit density,
- Horowitz-Sahni decomposition for dynamic programming

Horowitz-Sahni Decomposition

- Partition items into two sets A, B
- Find all Pareto optimal solutions for A, $\mathcal{L}_A$
- Find all Pareto optimal solution for B, $\mathcal{L}_B$
- The overall optimum is a combination of solutions from $\mathcal{L}_A$ and $\mathcal{L}_B$. Can be found in time $O(|\mathcal{L}_A| + |\mathcal{L}_B|)$
- $|\mathcal{L}_A| \leq 2^{n/2}$

Question: What is the problem in generalizing to three (or more) subsets?