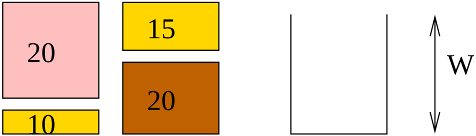
The Knapsack Problem



- 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 $\sum_{i \in \mathbf{x}} p_i$

Optimization problem

- Set of instances I
- Function F that gives for all $w \in I$ the set of feasible solutions $F(w)$
- Goal function g that gives for each $s \in F(w)$ the value $g(s)$

Optimization goal: Given input w, maximize or minimize the value $g(s)$ among all $s \in F(w)$

Decision problem: Given $w \in I$ and $k \in N$, decide whether

- $OPT(w) \leq k$ (minimization)
- $OPT(w) \geq k$ (maximization)

where $OPT(w)$ is the optimal function value among all $s \in F(w)$.

Optimization problem

Karlsruhe Institute of Technology

- Set of instances I
- Function F that gives for all $w \in I$ the set of feasible solutions $F(w)$
- Goal function g that gives for each $s \in F(w)$ the value $g(s)$

Optimization goal: Given input w, maximize or minimize the value $g(s)$ among all $s \in F(w)$

Decision problem: Given $w \in I$ and $k \in N$, decide whether

- $OPT(w) \leq k$ (minimization)
- $OPT(w) \geq k$ (maximization)

where $OPT(w)$ is the optimal function value among all $s \in F(w)$.

Quality of approximation algorithms

Recall: An approximation algorithm A producing a solution of value $A(w)$ on a given input $w \in I$ has approximation ratio r iff

$$\frac{A(w)}{OPT(w)} \leq r \quad \forall w \in I$$

(for maximization problems) or

$$\frac{OPT(w)}{A(w)} \leq r \quad \forall w \in I$$

(for minimization problems)
How good your approximation algorithm is depends on the value of r **and its running time.**

Negative result

We cannot find a result with bounded difference to the optimal solution in polynomial time.
Interestingly, the problem remains NP-hard if all items have the same weight to size ratio!

Reminder?: Linear Programming

Definition

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

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

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

Reminder?: Linear Programming

Karlsruhe Institute of Technology

Definition

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

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

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

Complexity

Theorem

A linear program can be solved in polynomial time.

- Worst case bounds are rather high
- The algorithm used in practice (simplex algorithm) might take exponential worst case time
- Reuse is not only possible but almost necessary

Integer Linear Programming

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

Linear Relaxation: Remove the integrality constraints from an ILP

Example: The Knapsack Problem

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

subject to

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

$x_i = 1$ iff item i is put into the knapsack.
0/1 variables are typical for ILPs

Linear relaxation for the knapsack problem

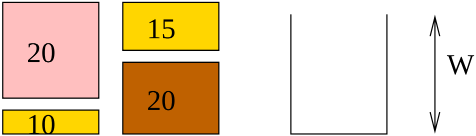
$$\text{maximize } \mathbf{p} \cdot \mathbf{x}$$

subject to

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

We allow items to be picked “fractionally”
$x_1 = 1/3$ means that $1/3$ of item 1 is put into the knapsack
This makes the problem much easier. How would you solve it?

The Knapsack Problem



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

How to Cope with ILPs

Karlsruhe Institute of Technology

- – 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.

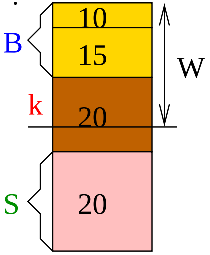
Linear Time Algorithm for Linear Relaxation of Knapsack

Karlsruhe Institute of Technology

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

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

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



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

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

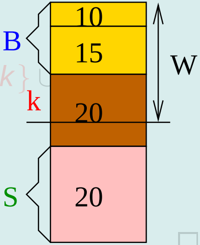
Lemma

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



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

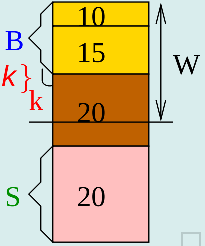
Lemma

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 $x_k = \frac{W - \sum_{i \in B} w_i}{w_k}$



□

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

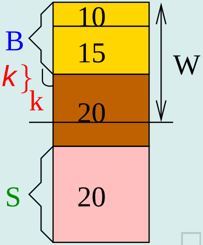
Lemma

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 $x_k = \frac{W - \sum_{i \in B} w_i}{w_k}$



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

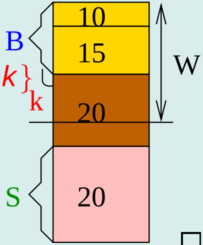
Lemma

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

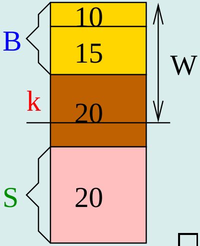
$$\mathrm{opt} \leq \sum_i x_i p_i \leq 2\mathrm{opt}$$

Proof.

We have $\sum_{i \in B} p_i \leq \mathrm{opt}$. Furthermore, since $w_k < W$, $p_k \leq \mathrm{opt}$. We get

$$\mathrm{opt} \leq \sum_i x_i p_i \leq \sum_{i \in B} p_i + p_k$$

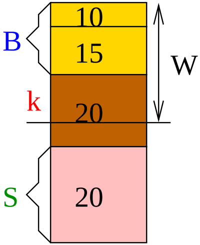
$$\leq \mathrm{opt} + \mathrm{opt} = 2\mathrm{opt}$$



□

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.

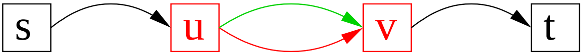
Dynamic Programming
— Building it Piece By Piece

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,...,i using capacity $\leq C$.

Lemma

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

Of course this only holds for C large enough: we must have $C \geq w_i$.

Lemma

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

Proof.

To prove: $P(i, C) \le \max(P(i-1, C), P(i-1, C-w_i) + p_i)$
Assume the contrary $\Rightarrow$
$\exists \mathbf{x}$ that is optimal for the subproblem such that

$$P(i-1, C) < \mathbf{p} \cdot \mathbf{x} \quad \wedge \quad P(i-1, C-w_i) + p_i < \mathbf{p} \cdot \mathbf{x}$$

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

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

□

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 $\mathrm{P}[C - w_i] + p_i > \mathrm{P}[C]$ **then**
        $\mathrm{P}[C] := \mathrm{P}[C - w_i] + p_i$
        $\mathrm{decision}[i, C] := 1$

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: $\mathcal{O}(nW)$ pseudo-polynomial

Space: $W + \mathcal{O}(n)$ words plus Wn bits.

Example: A Knapsack Instance

maximize $(10, 20, 15, 20) \cdot \mathbf{x}$
subject to $(1, 3, 2, 4) \cdot \mathbf{x} \leq 5$

$i \setminus C$	0	1	2	3	4	5
0	0	0	0	0	0	0
1						
2						
3						
4						

Example: A Knapsack Instance

maximize $(10, 20, 15, 20) \cdot \mathbf{x}$
subject to $(1, 3, 2, 4) \cdot \mathbf{x} \leq 5$
Entries in table are $P(i, C), (\mathrm{decision}[i, C])$

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

We check each time whether $\mathrm{P}[C - w_i] + p_i > \mathrm{P}[C]$

Example: A Knapsack Instance

maximize $(10, 20, 15, 20) \cdot \mathbf{x}$
subject to $(1, 3, 2, 4) \cdot \mathbf{x} \leq 5$
Entries in table are $P(i, C), (\text{decision}[i, C])$

$i \setminus C$	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						

We check each time whether $\mathrm{P}[C - w_i] + p_i > \mathrm{P}[C]$

Example: A Knapsack Instance

maximize $(10, 20, 15, 20) \cdot \mathbf{x}$
subject to $(1, 3, 2, 4) \cdot \mathbf{x} \leq 5$
Entries in table are $P(i, C), (\text{decision}[i, C])$

$i \setminus C$	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						

We check each time whether $\mathrm{P}[C - w_i] + p_i > \mathrm{P}[C]$

Example: A Knapsack Instance

maximize $(10, 20, 15, 20) \cdot \mathbf{x}$
subject to $(1, 3, 2, 4) \cdot \mathbf{x} \leq 5$
Entries in table are $P(i, C), (\text{decision}[i, C])$

$i \setminus C$	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						

We check each time whether $\mathrm{P}[C - w_i] + p_i > \mathrm{P}[C]$

Example: A Knapsack Instance

maximize $(10, 20, 15, 20) \cdot \mathbf{x}$
subject to $(1, 3, 2, 4) \cdot \mathbf{x} \leq 5$
Entries in table are $P(i, C), (\text{decision}[i, C])$

$i \setminus C$	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)

We check each time whether $\mathrm{P}[C - w_i] + p_i > \mathrm{P}[C]$

Dynamic Programming by Profit for the Knapsack Problem

Karlsruhe Institute of Technology

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

Lemma

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

Dynamic Programming by Profit

Karlsruhe Institute of Technology

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

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

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

A Faster Algorithm

Dynamic programs are only pseudo-polynomial-time

A polynomial-time solution is not possible (unless P=NP...), because this problem is NP-hard

However, it *would* be possible if the numbers in the input were small (i.e. polynomial in n)

To get a good *approximation* in polynomial time, we are going to ignore the least significant bits in the input

Fully Polynomial Time Approximation Scheme

Algorithm $\mathcal{A}$ is a
(Fully) Polynomial Time Approximation Scheme

for minimization / maximization problem Π if:

Input: Instance I, error parameter ε

Output Quality: $f(\mathbf{x}) \begin{smallmatrix}\leq\\ \geq\end{smallmatrix} (\begin{smallmatrix}1+\varepsilon\\ 1-\varepsilon\end{smallmatrix})\mathrm{opt}$

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

Problem classes

We can classify problems according to the approximation ratios which they allow.

- APX: constant approximation ratio achievable in time polynomial in *n* (Metric TSP, Vertex Cover)
- PTAS: $1 + \varepsilon$ achievable in time polynomial in *n* for any $\varepsilon > 0$ (Euclidean TSP)
- FPTAS: $1+$ achievable in time polynomial in *n* **and** $1/\varepsilon$ for any > 0 (Knapsack)

FPTAS $\rightarrow$ optimal solution

By choosing ε small enough, you can guarantee that the solution you find is in fact optimal. The running time will depend on the size of the optimal solution, and will thus again not be strictly polynomial-time (for all inputs).

FPTAS for Knapsack

Recall that $p_i \in \mathbb{N}$ for all i!

$P := \max_i p_i$ // maximum profit

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

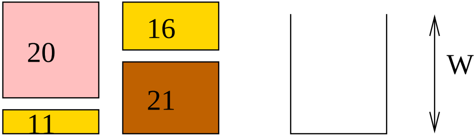
$p'_i := \left\lfloor \frac{p_i}{K} \right\rfloor$ // scale profits

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

output $\mathbf{x}'$

FPTAS for Knapsack

Recall that $p_i \in \mathbb{N}$ for all i!



$P :=\ \max_i p_i$ // 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 $\mathbf{x}'$

Example:

$\varepsilon = 1/3, n = 4, P = 20 \rightarrow K = 5/3$

Lemma

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

Proof.

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

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

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

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

We use that $\mathbf{x}'$ is an optimal solution for the modified problem.

Lemma

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

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,

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

We use that $\mathbf{x}'$ is an optimal solution for the modified problem.

Lemma

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

Proof.

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

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

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

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

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

Running time $\mathcal{O}\left(n^3/\varepsilon\right)$.

Proof.

The running time of dynamic programming dominates.

Recall that this is $\mathcal{O}\left(n\widehat{P'}\right)$ where $\widehat{P'} = \lfloor \mathbf{p}' \cdot \mathbf{x}^* \rfloor$.

We have

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

□

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 to get a quick 2-approximation.

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

A Faster FPTAS for Knapsack

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

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

value of optimal solution was at most $\hat{P}$, is now M

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

keep only the $\left\lfloor \frac{M}{j} \right\rfloor$ lightest (smallest) items from each C_j

do dynamic programming on the remaining items

Lemma

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

Proof.

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

Lemma

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

Proof.

preprocessing time: $\mathcal{O}(n)$

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

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

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

□

The Best Known FPTAS

Karlsruhe Institute of Technology

[Kellerer, Pferschy 04]

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

- 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