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 $\sum_{i \in x} w_i \leq W$
  wlog assume $\sum_i w_i > W$, $\forall i : w_i < W$
- Maximize profit $\sum_{i \in x} p_i$
Reminder?: Linear Programming

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

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

- **\( m \) constraints** of the form \( a_i \cdot x \triangleright_i b_i \) where \( \triangleright_i \in \{\leq, \geq, =\} \), \( a_i \in \mathbb{R}^n \) We have

\[
L = \left\{ x \in \mathbb{R}^n : \forall 1 \leq i \leq m : x_i \geq 0 \land a_i \cdot x \triangleright_i b_i \right\} .
\]

Let \( a_{ij} \) denote the \( j \)-th component of vector \( a_i \).
Complexity

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

- Worst case bounds are rather high
- The algorithm used in practice (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

\[
\text{maximize } p \cdot x
\]

subject to

\[
w \cdot x \leq W, \ x_i \in \{0, 1\} \text{ for } 1 \leq i \leq n.
\]

\(x_i = 1\) iff item \(i\) is put into the knapsack.

0/1 variables are typical for ILPs
Linear relaxation for the knapsack problem

maximize \boldsymbol{p} \cdot \mathbf{x}

subject to

\mathbf{w} \cdot \mathbf{x} \leq W, \quad 0 \leq x_i \leq 1 \quad \text{for} \quad 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 $\sum_{i \in x} w_i \leq W$
  - wlog assume $\sum_i w_i > W$, $\forall i : w_i < W$
- Maximize profit $\sum_{i \in x} p_i$
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

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

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

Set $x_i = \begin{cases} 
1 & \text{if } i \in B \\
\frac{W - \sum_{i \in B} w_i}{w_k} & \text{if } i = k \\
0 & \text{if } i \in S 
\end{cases}$
Lemma 2. \( x \) is the optimal solution of the linear relaxation.
Lemma 2. \( x \) is the optimal solution of the linear relaxation.

Proof. Let \( x^* \) denote the optimal solution

- \( w \cdot x^* = W \) otherwise increase some \( x_i \)

\[
x_i = \begin{cases} 
1 & \text{if } i \in B \\
\frac{W - \sum_{i \in B} w_i}{w_k} & \text{if } i = k \\
0 & \text{if } i \in S 
\end{cases}
\]
Lemma 2. \( x \) is the optimal solution of the linear relaxation.

Proof. Let \( x^* \) denote the optimal solution

- \( w \cdot x^* = W \) 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 \( x_k^* \)
Lemma 2. x is the optimal solution of the linear relaxation.

Proof. Let $x^*$ denote the optimal solution

- $w \cdot x^* = W$ 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 $x_k^*$
Lemma 2. $x$ is the optimal solution of the linear relaxation.

Proof. Let $x^*$ denote the optimal solution

- $\mathbf{w} \cdot x^* = W$ 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 $x_k = \frac{W - \sum_{i \in B} w_i}{w_k}$
Lemma 3. For the optimal solution $x$ of the linear relaxation:

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

Proof. We have

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

We get

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

\[ P(i, C) = \text{optimal profit from items } 1, \ldots, i \text{ using capacity } \leq C. \]

Lemma 4.

\[ \forall 1 \leq i \leq n : P(i, C) = \max (P(i - 1, C), P(i - 1, C - w_i) + p_i) \]
Lemma 4.

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

Proof

\[ P(i, C) \geq P(i - 1, C) : \text{Set } x_i = 0, \text{ use optimal subsolution.} \]
Lemma 4.
\( \forall 1 \leq i \leq n : P(i, C) = \max(P(i - 1, C), P(i - 1, C - w_i) + p_i) \)

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

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

Therefore \( P(i, C) \geq \max(P(i - 1, C), P(i - 1, C - w_i) + p_i) \).
Lemma 4. 
\[ \forall 1 \leq i \leq n : P(i, C) = \max(P(i - 1, C), P(i - 1, C - w_i) + p_i) \]

**Proof**

\[ P(i, C) \leq \max(P(i - 1, C), P(i - 1, C - w_i) + p_i) \].

Assume the contrary:

\[ \exists x \text{ that is optimal for the subproblem such that} \]

\[ P(i - 1, C) < p \cdot x \quad \land \quad P(i - 1, C - w_i) + p_i < p \cdot x \]
Lemma 4.
\[ \forall 1 \leq i \leq n : P(i, C) = \max(P(i - 1, C), P(i - 1, C - w_i) + p_i) \]

Proof
\[ P(i, C) \leq \max(P(i - 1, C), P(i - 1, C - w_i) + p_i) \]

Assume the contrary:
\[ \exists x \text{ that is optimal for the subproblem such that} \]
\[ P(i - 1, C) < p \cdot x \land P(i - 1, C - w_i) + p_i < p \cdot x \]

Case \( x_i = 0 \): \( x \) is also feasible for \( P(i - 1, C) \). Hence,
\[ P(i - 1, C) \geq p \cdot x. \text{ Contradiction} \]
Lemma 4.
\( \forall 1 \leq i \leq n : P(i, C) = \max(P(i - 1, C), P(i - 1, C - w_i) + p_i) \)

Proof

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

Assume the contrary:

\( \exists x \) that is optimal for the subproblem such that

\[ P(i - 1, C) < p \cdot x \quad \land \quad P(i - 1, C - w_i) + p_i < p \cdot x \]

Case \( x_i = 0 \): \( x \) is also feasible for \( P(i - 1, C) \). Hence,

\[ P(i - 1, C) \geq p \cdot x \]. Contradiction

Case \( x_i = 1 \): Setting \( x_i = 0 \) we get a feasible solution \( x' \) for

\[ P(i - 1, C - w_i) \] with profit \( p \cdot x' = p \cdot x - p_i \). Add \( p_i \)...
Computing $P(i, C)$ bottom up:

**Procedure** knapsack($p, c, n, W$)

array $P[0 \ldots W] = [0, \ldots, 0]$

bitarray $\text{decision}[1 \ldots n, 0 \ldots W] = [(0, \ldots, 0), \ldots, (0, \ldots, 0)]$

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

// invariant: $\forall C \in \{1, \ldots, W\} : P[C] = P(i - 1, C)$

for $C := W$ downto $w_i$ do

if $P[C - w_i] + p_i > P[C]$ then

$P[C] := P[C - w_i] + p_i$

decision[$i, C$] := 1
Recovering a Solution

\[ C := W \]
array \( x[1 \ldots n] \)

for \( i := n \) downto 1 do

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

if \( x[i] = 1 \) 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 \((10, 20, 15, 20) \cdot x\)
subject to \((1, 3, 2, 4) \cdot x \leq 5\)

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

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>10, (1)</td>
<td>10, (1)</td>
<td>10, (1)</td>
<td>10, (1)</td>
<td>10, (1)</td>
</tr>
<tr>
<td>2</td>
<td>0, (0)</td>
<td>10, (1)</td>
<td>10, (1)</td>
<td>10, (1)</td>
<td>10, (1)</td>
<td>10, (1)</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example: A Knapsack Instance

maximize $(10, 20, 15, 20) \cdot x$

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

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

<table>
<thead>
<tr>
<th>$i \setminus C$</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0, (0)</td>
<td>10, (1)</td>
<td>10, (1)</td>
<td>10, (1)</td>
<td>10, (1)</td>
<td>10, (1)</td>
</tr>
<tr>
<td>2</td>
<td>0, (0)</td>
<td>10, (0)</td>
<td>10, (0)</td>
<td>20, (1)</td>
<td>30, (1)</td>
<td>30, (1)</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example: A Knapsack Instance

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

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

<table>
<thead>
<tr>
<th>(i \setminus C)</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0, (0)</td>
<td>10, (1)</td>
<td>10, (1)</td>
<td>10, (1)</td>
<td>10, (1)</td>
<td>10, (1)</td>
</tr>
<tr>
<td>2</td>
<td>0, (0)</td>
<td>10, (0)</td>
<td>10, (0)</td>
<td>20, (1)</td>
<td>30, (1)</td>
<td>30, (1)</td>
</tr>
<tr>
<td>3</td>
<td>0, (0)</td>
<td>10, (0)</td>
<td>15, (1)</td>
<td>25, (1)</td>
<td>30, (0)</td>
<td>35, (1)</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ P_0 = \text{value}(0) \]
\[ P_i = \text{value}(i) \]
Example: A Knapsack Instance

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

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

<table>
<thead>
<tr>
<th>(i) (\backslash) (C)</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>(0)</td>
<td>10, (1)</td>
<td>10, (1)</td>
<td>10, (1)</td>
<td>10, (1)</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>(0)</td>
<td>10, (0)</td>
<td>10, (0)</td>
<td>20, (1)</td>
<td>30, (1)</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>(0)</td>
<td>10, (0)</td>
<td>15, (1)</td>
<td>25, (1)</td>
<td>30, (0)</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>(0)</td>
<td>10, (0)</td>
<td>15, (0)</td>
<td>25, (0)</td>
<td>30, (0)</td>
</tr>
</tbody>
</table>
Dynamic Programming by Profit
for the Knapsack Problem

Define
\[ C(i, P) = \text{smallest capacity from items } 1, \ldots, i \text{ giving profit } \geq P. \]

Lemma 5.
\[
\forall 1 \leq i \leq n : C(i, P) = \min(C(i - 1, P), C(i - 1, P - p_i) + w_i)
\]
Dynamic Programming by Profit

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

Thus \( \hat{\mathcal{P}} \) is the value (profit) of this solution.

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

Space: \( \hat{\mathcal{P}} + O(n) \) words plus \( \hat{\mathcal{P}}n \) 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 problem $\Pi$ if:

**Input:** Instance $I$, error parameter $\epsilon$

**Output Quality:** $f(x) \leq \left(1 + \epsilon\right)\text{opt}$

**Time:** Polynomial in $|I|$ (and $1/\epsilon$)
### Example Bounds

<table>
<thead>
<tr>
<th>PTAS</th>
<th>FPTAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n + 2^{1/\varepsilon}$</td>
<td>$n^2 + \frac{1}{\varepsilon}$</td>
</tr>
<tr>
<td>$n^{\log \frac{1}{\varepsilon}}$</td>
<td>$n + \frac{1}{\varepsilon^4}$</td>
</tr>
<tr>
<td>$\frac{1}{\varepsilon}$</td>
<td>$n/\varepsilon$</td>
</tr>
<tr>
<td>$n^{42/\varepsilon^3}$</td>
<td>:</td>
</tr>
<tr>
<td>$n + 2^{1000/\varepsilon}$</td>
<td>:</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
</tr>
</tbody>
</table>
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

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

output \( x' \)
FPTAS for Knapsack

\[ P := \max_i p_i \]  
\[ K := \frac{\varepsilon P}{n} \]  
\[ p'_i := \left\lfloor \frac{p_i}{K} \right\rfloor \]  
\[ x' := \text{dynamicProgrammingByProfit}(p', c, C) \]
output \( x' \)

Example:
\[ \varepsilon = 1/3, n = 4, P = 20 \rightarrow K = 5/3 \]
\[ p = (11, 20, 16, 21) \rightarrow p' = (6, 12, 9, 12) \]
(equivalent to \( p' = (2, 4, 3, 4) \))
Lemma 6. $p \cdot x' \geq (1 - \varepsilon)\text{opt}$.

Proof. Consider the optimal solution $x^*$.

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

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

i.e., $Kp' \cdot x^* \geq p \cdot x^* - nK$. 

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

Proof. Consider the optimal solution $x^*$.

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

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

i.e., $Kp' \cdot x^* \geq p \cdot x^* - nK$. Furthermore,

$$Kp' \cdot x^* \leq Kp' \cdot x' = \sum_{i \in x'} K \left\lfloor \frac{p_i}{K} \right\rfloor \leq \sum_{i \in x'} K \frac{p_i}{K} = p \cdot x'.$$
Lemma 6. \( p \cdot x' \geq (1 - \varepsilon)\text{opt} \).

Proof. Consider the optimal solution \( x^* \).

\[
p \cdot x^* - Kp' \cdot x^* = \sum_{i \in x^*} \left( p_i - K \left\lfloor \frac{p_i}{K} \right\rfloor \right)
\leq \sum_{i \in x^*} \left( p_i - K \left( \frac{p_i}{K} - 1 \right) \right) = |x^*|K \leq nK,
\]
i.e., \( Kp' \cdot x^* \geq p \cdot x^* - nK \). Furthermore,

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

Hence,

\[
p \cdot x' \geq Kp' \cdot x^* \geq p \cdot x^* - nK = \text{opt} - \varepsilon P \geq (1 - \varepsilon)\text{opt}
\]

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

Proof. The running time of dynamic programming dominates. Recall that this is $O\left(n^{\hat{P}'}\right)$ where $\hat{P}' = \lfloor p' \cdot x^* \rfloor$.

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/ε ∈ ℕ: Otherwise ε := 1/⌈1/ε⌉.

Upper bound ˆP is known: Use linear relaxation.

min_i p_i ≥ ε ˆP: Treat small profits separately. For these items greedy works well. (Costs a factor O(log(1/ε)) time.)
A **Faster** FPTAS for Knapsack

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

\[ p_i' := \left\lfloor \frac{p_i}{K} \right\rfloor \quad \text{II } p_i' \in \left\{ \frac{1}{\varepsilon}, \ldots, M \right\} \]

value of optimal solution was \( \hat{P} \), is now \( M \)

\( C_j := \{ i \in 1..n : p_i' = j \} \)

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

do dynamic programming on the remaining items

**Lemma 8.** \( px' \geq (1 - \varepsilon)\text{opt.} \)

**Proof.** Similar as before, note that \( |x| \leq 1/\varepsilon \) for any solution.
Lemma 9. Running time $O(n + \text{Poly}(1/\varepsilon))$.

Proof.

preprocessing time: $O(n)$

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

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

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.


Main additional tricks:

- reduce to core items with good profit density,
- Horowitz-Sahni decomposition for dynamic programming
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 $\sum_{i \in x} w_i$
- Maximize total profit $\sum_{i \in x} p_i$

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 Céлиny]

Solution \( x \) **dominates** solution \( x' \) iff

\[
p \cdot x \geq p \cdot x' \land c \cdot x \leq c \cdot x'
\]

and one of the inequalities is proper.

Solution \( x \) is **Pareto optimal** if

\[
\not\exists x' : x' \text{ dominates } x
\]

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]

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

for \( i := 1 \) to \( n \) do

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

\[ \mathcal{L} := \text{merge}(\mathcal{L}, \mathcal{L}') \quad \text{// time } O(|\mathcal{L}|) \]

scan \( \mathcal{L} \) and eliminate dominated solutions \quad \text{// 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 \(W = 5\)

\[
\begin{align*}
L &= \langle (0, 0) \rangle \\
(1, 10) \rightarrow L' &= \langle (1, 10) \rangle \\
\text{merge} \rightarrow L &= \langle (0, 0), (1, 10) \rangle \\
(3, 20) \rightarrow L' &= \langle (3, 20), (4, 30) \rangle \\
\text{merge} \rightarrow L &= \langle (0, 0), (1, 10), (3, 20), (4, 30) \rangle \\
(2, 15) \rightarrow L' &= \langle (2, 15), (3, 25), (5, 35) \rangle \\
\text{merge} \rightarrow L &= \langle (0, 0), (1, 10), (2, 15), (3, 25), (4, 30), (5, 35) \rangle \\
(4, 20) \rightarrow L' &= \langle (4, 20), (5, 30) \rangle \\
\text{merge} \rightarrow L &= \langle (0, 0), (1, 10), (2, 15), (3, 25), (4, 30), (5, 35) \rangle
\end{align*}
\]
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?