Integer Programming

The Knapsack Problem.

The simplest IP - a playground for developing methods

F. Vanderbeck
fv@math.u-bordeaux1.fr
http://www.math.u-bordeaux.fr/∼fv/cours/
login: coursfv ; password: promoEtudiantBx

Outline

1. Formulation and Variants
2. Complexity
3. LP Solution
4. IP Solution Heuristic
5. Correlation between profits and weights
6. Specialized Branch-and-Bound Approach
7. Dynamic Programming Approaches
8. Extended formulation
9. Variants

Outline

1. Formulation and Variants
2. Complexity
3. LP Solution
4. IP Solution Heuristic
5. Correlation between profits and weights
6. Specialized Branch-and-Bound Approach
7. Dynamic Programming Approaches
8. Extended formulation
9. Variants

Example: Portfolio optimization

Given a portfolio of possible investments, select a subset that maximizes expected return subject to a budget constraint:

$$\begin{aligned}
\textbf{maximize} \quad & \text{profit}_1\, x_1 + \text{profit}_2\, x_2 + \text{profit}_3\, x_3 \\
\textbf{st} \quad & \text{invest}_1\, x_1 + \text{invest}_2\, x_2 + \text{invest}_3\, x_3 \leq \text{budget} \\
& x_i = 0 \text{ or } 1 \quad \text{for } i = 1, \ldots, 3
\end{aligned}$$

Applications

- Hiker
- Finance
- Packing: f.i. airplane freit
- Cutting: f.i. window frame raw material
- IP subproblem: any single constraint extracted from an IP

Formulation : 0-1 or Pure integer program

$$\max \sum_{i=1}^{n} p_i \, x_i$$

s.t.

$$\sum_{i=1}^{n} w_i \, x_i \leq W$$

$$x_i \in \{0, 1\} \text{ or } \in \mathbb{N} \quad \forall i$$

With bounds:

$$x_i \leq u_i \leq \left\lfloor \frac{W}{w_i} \right\rfloor$$

Assumptions

i $w_i \leq W \quad \forall i$

ii $\sum_i w_i > W$

iii $p_i > 0$ and $w_i > 0 \quad \forall i$

Indeed,

- If $p_i > 0$ and $w_i \leq 0$, setting $x_i = 1$ (or $x_i = u_i$ in the IP case) is optimum;
- If $p_i \leq 0$ and $w_i \geq 0$, setting $x_i = 0$ is optimum;
- If $p_i < 0$ and $w_i < 0$, setting $y_i = 1 - x_i$ in remplacement if x_i (0-1 case) or $y_i = u_i - x_i$ (IP case).

Outline

1. Formulation and Variants
2. Complexity
3. LP Solution
4. IP Solution Heuristic
5. Correlation between profits and weights
6. Specialized Branch-and-Bound Approach
7. Dynamic Programming Approaches
8. Extended formulation
9. Variants

NP-Hardness

Proposition

The binary knapsack problem is NP-Hard

Proof: polynomial reduction from PARTITION

PARTITION:

$$\exists\ ?\ x \in \{0,1\}^n : \sum_{i=1}^{n} w_i x_i = W = \frac{1}{2}\sum_{i=1}^{n} w_i$$

KNAPSACK:

$$\max\{\sum_{i=1}^{n} p_i x_i : \sum_{i=1}^{n} w_i x_i \leq W,\ x \in \{0,1\}^n\}$$

PART $\propto$ *SAD* Algo for PARTITION:

Set $p_i = w_i$.

Call Knapsack algo

If x^* yields a profit $= W$, then return x^*.

Else, retrun NO.

Outline

1. Formulation and Variants
2. Complexity
3. **LP Solution**
4. IP Solution Heuristic
5. Correlation between profits and weights
6. Specialized Branch-and-Bound Approach
7. Dynamic Programming Approaches
8. Extended formulation
9. Variants

Preview on solution approaches: a numerical example

Sort items by non-increasing profit ratio

$$\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq \dots \geq \frac{p_n}{w_n}$$

$n = 5,\ W = 6$

i	1	2	3	4	5
p_i	30	28	27	34	7
w_i	2	2	3	4	1
$\frac{p_i}{w_i}$	15	14	9	8.5	7

$$\text{Lp sol} : x^*_{LP} = (1, 1, \frac{2}{3}, 0, 0) \rightarrow UB = 76$$

$$\text{Ip Heuristic} : x^a_{IP} = (1, 1, 0, 0, 1) \rightarrow LB = 65$$

$$65 \leq c\, x^*_{IP} \leq 76$$

$$\text{Gap} : \alpha = \frac{76 - 65}{65} = 0.169$$

Proposition

A greedy algorithm solves the LP in $o(n \log n)$

GREEDY ALGO Sort items by non-increasing profit ratio; let c such that

$$\sum_{i=1}^{c-1} w_i \leq W \text{ but } \sum_{i=1}^{c} w_i > W$$

Set

$$\begin{aligned} x_i &= 1 \text{ pour } i = 1, \ldots, c-1 \\ x_c &= \frac{(W - \sum_{i=1}^{c-1} w_i)}{w_c} \\ x_i &= 0 \text{ pour } i > c+1 \end{aligned}$$

LP Dual Bound

Proposition

A greedy algorithm solves the LP in $o(n \log n)$

Proof

$$\lambda = \frac{p_c}{w_c},\ \nu_i = w_i \left(\frac{p_i}{w_i} - \frac{p_c}{w_c}\right) \text{ pour } i = 1, \ldots, c-1,$$

while $\nu_i = 0$ for $i = c, \ldots, n$ provides a feasible dual solution that achieve the same value:

$$\begin{array}{rclc|rcl}
\max \sum_i p_i x_i & & & & \min W\lambda + \sum_i \nu_i & & \\
\sum_i w_i x_i & \leq & W & & w_i\lambda + \nu_i & \geq & p_i \ \forall i \\
x_i & \leq & 1 \ \forall i & & \nu_i & \geq & 0 \ \forall i \\
x_i & \geq & 0 & & \lambda & \geq & 0
\end{array}$$

Outline

1. Formulation and Variants
2. Complexity
3. LP Solution
4. IP Solution Heuristic
5. Correlation between profits and weights
6. Specialized Branch-and-Bound Approach
7. Dynamic Programming Approaches
8. Extended formulation
9. Variants

Greedy IP heuristic

for $k = 1, \ldots, n$ do
    if $(w_k \leq W)$, then $x_k = 1$ and $W = W - w_k$;
        else $x_k = 0$.

Example $n = 5$, $W = 6$,

i	1	2	3	4	5
p_i	30	28	27	34	7
w_i	2	2	3	4	1
$\frac{p_i}{w_i}$	15	14	9	8.5	7

Solution: $x = (1, 1, 0, 0, 1) \rightarrow LB = 65$

Proposition

$$Z^G \leq Z^* \leq Z^{LP} \leq \sum_{i=1}^{c} p_i \leq Z^G + p_c \leq Z^G + p_{\max}$$

Greedy Worse Case Performance is ∞

Example: $n = 2$, $W = k$

i	1	2
p_i	1	k
w_i	1	k
$\frac{p_i}{w_i}$	1	1

$$\text{greedy} : x = (1, 0) \rightarrow Z^G = 1$$

$$\text{optimum} : x = (0, 1) \rightarrow Z^* = k$$

$$Performance\ ratio = \frac{Z^G}{Z^*} \rightarrow 0$$

Alternative Greedy with Worse Case Analysis

noindent Sort items per non decrasing profitp_i:

$$p_1 \geq p_2 \geq \ldots \geq p_n$$

and apply greedy...
Example: n, $W = n - 1$

i	1	2	$\ldots$	n
p_i	2	1	$\ldots$	1
w_i	n-1	1	$\ldots$	1

$$\textbf{Performance ratio} = \frac{Z^{G'}}{Z^*} = \frac{2}{n-1} \to 0$$

Combined Greedy: G''

Take the best solution from both procedures

Proposition

The Combined Greedy has

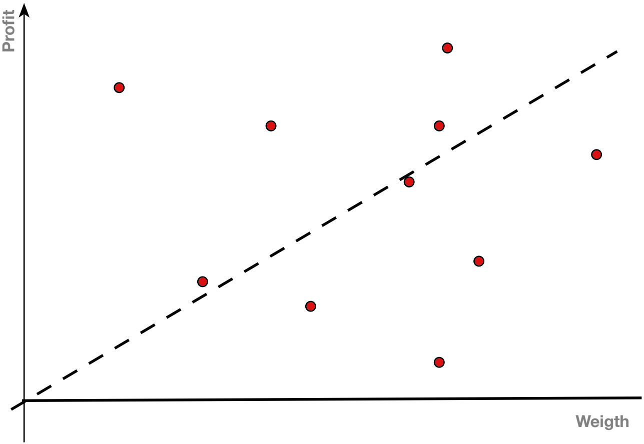
Performance Ratio $= \max \frac{\max\{Z^G, Z^{G'}\}}{Z^*} = \frac{1}{2}$.

<u>PROOF:</u> $\max\{Z^G, Z^{G'}\} \geq \frac{Z^G + Z^{G'}}{2} \geq \frac{Z^* - p_{\max} + p_{\max}}{2} \geq \frac{Z^*}{2}$

Outline

1. Formulation and Variants
2. Complexity
3. LP Solution
4. IP Solution Heuristic
5. Correlation between profits and weights
6. Specialized Branch-and-Bound Approach
7. Dynamic Programming Approaches
8. Extended formulation
9. Variants

Correlation is bad for Greedy Heuristics



Variable Fixing

Let $U_i^0 = UB(x_i = 0)$ and $U_i^1 = UB(x_i = 1)$.
Let INC be the best known LB. Then,

$$\begin{aligned} U_i^0 \leq INC & \quad \Rightarrow x_i^* = 1 \\ U_i^1 \leq INC & \quad \Rightarrow x_i^* = 0 \end{aligned}$$

The Core of a Knapsack problem

Index items by non-decreasing profit ratio

$$\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq \ldots \geq \frac{p_n}{w_n}$$

The core is the smallest interval $[a, b]$ such that $1 \leq a \leq b \leq n$ t.q.

$$Z^* = \sum_{i<a} p_i + \max\{\sum_{i=a}^{b} p_j\, x_j : \sum_{i=a}^{b} w_i\, x_i \leq W - \sum_{i<a} w_i,$$

$$x_i \in \{0, 1\}\ i = a, \ldots, b\}$$

Outline

1. Formulation and Variants
2. Complexity
3. LP Solution
4. IP Solution Heuristic
5. Correlation between profits and weights
6. Specialized Branch-and-Bound Approach
7. Dynamic Programming Approaches
8. Extended formulation
9. Variants

Depth-First-Search B-a-B (Horowitz et Sahni - 1974)

Step 0: **Sort** items s.t. $\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq \ldots \geq \frac{p_n}{w_n}$

Step 1: **Initialize:** $i = 1$, $c = W$, $z = 0$,
$INC = 0$, $UB = \infty$, $x_k^* = x_k = 0$ pour $k = 1, \ldots, n$

Step 2: **Compute UB:** $UB = z + PL(i, n, c)$
Si $UB \leq INC$, goto Step 6.

Step 3: **Plunge with fixation to 1:** while $i \leq n$ & $c \geq w_i$,
$x_i = 1$, $z = z + p_i$, $c = c - w_i$, $i = i + 1$

Step 4: **A fixation to 0:** If $(i < n)$,
set $x_i = 0$, $i = i + 1$ & goto Step 2.

Step 5: **Update LB:** If $(i = n)$,
si $z > LB$, $x^* = x$, $LB = z$.

Step 6: **Backtrack:** If $i = n$,
set $x_n = 0$, $z = z - p_n$, $c = c + w_n$, $i = i - 1$.
While $x_i = 0$, do $i = i - 1$. If $i = 0$, STOP.
Set $x_i = 0$, $z = z - p_i$, $c = c + w_i$, $i = i + 1$.
Goto Step 2.

Outline

1. Formulation and Variants
2. Complexity
3. LP Solution
4. IP Solution Heuristic
5. Correlation between profits and weights
6. Specialized Branch-and-Bound Approach
7. Dynamic Programming Approaches
8. Extended formulation
9. Variants

Dynamic Programming for the Knapsack Problem

$$KNP \equiv \max \sum_{i=1}^{n} p_i \; x_i$$

$$\sum_{i=1}^{n} w_i \; x_i \leq W$$

$$x_i \in \{0, 1\} \quad \forall i$$

- Stages $\equiv$ subsets of items $\{1, \ldots, k\}$.
- States $\equiv$ capacity used up to the current stage.

$$V^k(b) = \max\{\sum_{i=1}^{k} p_i x_i : \sum_{i=1}^{k} w_i x_i \leq b, x_i \in \{0, 1\} \; i = 1, \ldots k\}$$

defines the maximum profit possible using item $1, \ldots, k$ and capacity b.

Dynamic Programming for the Knapsack Problem

- $V^k(b) = \max\{\sum_{i=1}^k p_i\, x_i : \sum_{i=1}^k w_i\, x_i \leq b, x_i \in \{0,1\}\ i = 1, \ldots k\}$.
- **Algorithm:**

Step 1: Initialize

$$V^0(b) = 0 \qquad \text{for } b = 0, \ldots, W$$

Step 2: Recursively compute

$$V^k(b) = \max\{\underbrace{V^{k-1}(b)}_{x_k=0}, \underbrace{V^{k-1}(b - w_k) + p_k}_{x_k=1}\}$$

for $k = 1, \ldots, n$ and $b = 1, \ldots, W$.

Step 3: The optimum value is given by $\max_{b \leq W} V^n(b)$.

Backward Recursions for the 0-1 Knapsack : in practice

$$V^k(b) = \max\{\sum_{i=1}^{k} p_i x_i : \sum_{i=1}^{k} w_i x_i \leq b, x_i \in \{0,1\}\ i = 1,\ldots k\} \rightarrow O(nW)$$

b / k	0	1	2	3	4	5
0						
1						
⋮						
W						

Initialisation: For $b = 0, \ldots, W$, let $V[b] = 0$;
Recursion: For $k = 1, \ldots, n$, do
For $b = W, \ldots, w_k$, if $V[b - w_k] + p_k > V[b]$
then set $V[b] = V[b - w_k] + p_k$.
Output solution value : $V[W]$

Dynamic Programming for the Knapsack Problem

- **Outputing the Solution :**
  - either one **records** the associated **solutions** one can keep track of:

$$X^k(b) = \begin{cases} 1 & \text{si } V^{k-1}(b - w_k) + p_k > V^{k-1}(b) \\ 0 & \text{sinon} \end{cases}$$

  - or, one uses backtracking.
- **Complexity** : $O(n\,W) \ll O(2^n)$
  (pseudo-polynomial complexity)
- **Memory space** : $O(n\,W)$

$$B^k(p) = \min\{\sum_{i=1}^{k} w_i\, x_i \,:\, \sum_{i=1}^{k} p_i\, x_i \geq p, x_i \in \{0,1\}\ i = 1, \ldots k\} \rightarrow O(nP)$$

- **Algorithm:**

p / k	0	1	2	3	4	5
0						
1						
⋮						
P						

- **Complexity** : $O(n\,P) \ll O(2^n)$
  (pseudo-polynomial complexity)
- **Memory space** : $O(n\,P)$

Forward Recursion by list for the 0-1 Knapsack

- **State:** $= (w, p) =$ (current weight, current profit)
- **Dominance:** $\boxed{(w^1, p^1) \succ (w^2, p^2)}$,
  if $(w^1 < w^2$ and $p^1 \geq p^2)$ or $(w^1 = w^2$ and $p^1 > p^2)$
- **Forward DP:**

  Step 1: Let $list = \{(0, 0)\}$
  Step 2: For $k = 1, \ldots, n$, do
  {
  Let $list'$ be empty. For all $s = (w, p) \in list$, do
  { if $(w + w_k \leq W)$, **(feasibility test)**
  add $(w + w_k, p + p_k)$ to $list'$.
  }
  Merge $list'$ to $list$ while eliminating dominated states.
  }
  Step 3: The optimum value is given by

$$\max\{p : (w, p) \in list\}$$

- **Prune by bound:** if $p + LP(n - k, W - w) \leq INC$, cut (w, p).

Exploiting the Core

- Index items s.t.

$$\text{avec } \frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq \ldots \geq \frac{p_n}{w_n}$$

- Let $V^{a,b}(d)$ be the maximum profit made using items $i \in [a, b]$ using capacity d, given that items $i < a$ are fixed to 1 and items $i > b$ are fixed to 0:

$$V^{a,b}(d) = \sum_{i<a} p_i + P^{ab}(d - \sum_{i<a} w_i)$$

where $P^{ab}(Cap) \equiv$

$$\max\{\sum_{i=a}^{b} p_i\, x_i : \sum_{i=a}^{b} w_i\, x_i \leq Cap, x_i \in \{0, 1\}\ i \in [a, b]\}$$

Exploiting the Core

- **Initialization**

$$V^{c,c-1}(d) = \sum_{i<c} p_i \quad \forall d > \sum_{i<c} w_i$$

- **Recursion** $\forall a, b$ **and** $d = 0, \ldots 2\,W$**:**

$$V^{a,b}(d) = \max\{\underbrace{V^{a+1,b}(d)}_{x_a=1},$$

$$\underbrace{V^{a+1,b}(d+w_a) - p_a}_{x_a=0},$$

$$\underbrace{V^{a,b-1}(d)}_{x_b=0},$$

$$\underbrace{V^{a,b-1}(d-w_b) + p_b}_{x_b=1}\}\ .$$

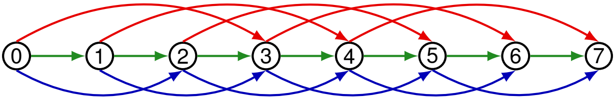
- **Sequence** $\forall d, \quad V^{c,c-1},\ V^{c,c},\ V^{c-1,c},\ V^{c-1,c+1}, \ldots,\ V^{1,n}$

Outline

1. Formulation and Variants
2. Complexity
3. LP Solution
4. IP Solution Heuristic
5. Correlation between profits and weights
6. Specialized Branch-and-Bound Approach
7. Dynamic Programming Approaches
8. **Extended formulation**
9. Variants

Unbounded Knapsack Problem: Network Flow Reformulation

a cutting pattern defines a path



$$\max \sum_{i=1}^{n} \sum_{b} p_i x_{ib} \tag{1}$$

$$\sum_{i} x_{i,b-w_i} - \sum_{i} x_{ib} = 0 \qquad b = 1, \cdots, W-1 \tag{2}$$

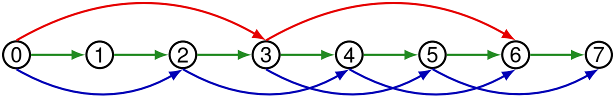
$$x_{ib} \geq 0 \qquad i, b \tag{3}$$

Arc flow:

$x_{ib} =$ nb of copies of item i cut from position b in a pattern.

Unbounded Knapsack Problem: Network Flow Reformulation

a cutting pattern defines a path



$$\max \sum_{i=1}^{n} \sum_{b} p_i x_{ib} \quad (1)$$

$$\sum_{i} x_{i,b-w_i} - \sum_{i} x_{ib} = 0 \qquad b = 1, \cdots, W-1 \quad (2)$$

$$x_{ib} \geq 0 \qquad i, b \quad (3)$$

Arc flow:

$x_{ib} =$ nb of copies of item i cut from position b in a pattern.

Dominance rule to eliminate some symmetric solutions

Outline

1. Formulation and Variants
2. Complexity
3. LP Solution
4. IP Solution Heuristic
5. Correlation between profits and weights
6. Specialized Branch-and-Bound Approach
7. Dynamic Programming Approaches
8. Extended formulation
9. Variants

The Integer Knapsack problem: bounded case

$$\max \sum_{i=1}^{n} p_i \, x_i$$

$$\sum_{i=1}^{n} w_i \, x_i \leq W$$

$$u_i \geq x_i \in \mathbb{N} \quad \forall i$$

DP : $O(n\,W^2)$ (in case of a bounded problem)
Le $V^k(b)$ be the maximum profit that can be made with items 1 to k and capacity b.

$$V^0(b) = 0 \qquad\qquad \text{pour } b = 0, \ldots, W$$

$$V^k(b) = \max_{t=0,\ldots,u_i} \{\underbrace{V^{k-1}(b - t\,w_k) + t\,p_k}_{x_k = t}\}$$

for $k = 1, \ldots, n$ and $b = 1, \ldots, W$.
The optimum solution value is $\max_{b \leq W} V^n(b)$.

The Integer Knapsack problem: unbounded case

$$\max \sum_{i=1}^{n} p_i\, x_i$$

$$\sum_{i=1}^{n} w_i\, x_i \leq W$$

$$x_i \in \mathbb{N} \quad \forall i$$

DP: $O(n\,W)$ (in case of an unbounded problem)

$$\text{for } b = 0, \ldots, W, \quad V^k(b) = \max\{\underbrace{V^{k-1}(b)}_{x_k=0}, \underbrace{V^k(b - w_k) + p_k}_{x_k \geq 1}\}$$

where

$$V(b) = \max_{i:\, w_i \leq b} \{\underbrace{V(b - w_i) + p_i}_{x_i = x_i + 1}\}$$

Knapsack problem with SOS Constraints

Given the set of items $\{1, \ldots, I\}$ with weights $w_i \geq 0$, profits $p_i \geq 0$ and a set of knapsacks $\{1, \ldots, K\}$ with disjoint SOS Constraints over subsets S_k, solve the following problem:

$$
\begin{aligned}
\max \sum_{i=1}^{I} p_i x_i & & & \\
\text{s.t.} \sum_{i=1}^{I} w_i x_i & \leq & W & \\
\sum_{i \in S^k} x_i & = & 1 & \quad \forall k \in \{1, \ldots, K\} \\
x_i & \in & \{0, 1\} & \quad \forall i \in \{1, \ldots, I\}
\end{aligned}
$$

Knapsack problem with SOS Constraints: solution

Proposition

A greedy algorithm solves the LP in $O(n \log n)$

Proof

For each S_k sort the items $i \in S_k$ by $w_{k,j0} \leq w_{k,j1} \leq \ldots$.
Define $y_{kj} = x_{k,j} - x_{k,j-1}$ for $j = 1, \ldots, |S_k| - 1$

$$\max \sum_{k=1}^{K} \sum_{j=1}^{|S_k|-1} (p_{k,j} - p_{k,j-1}) y_{kj}$$

$$\begin{aligned}
\text{s.t.} \sum_{k=1}^{K} \sum_{j=1}^{|S_k|-1} (w_{k,j} - w_{k,j-1}) y_{kj} &\leq W - \sum_{k} w_{k,j0} \\
\sum_{j=1}^{|S_k|-1} y_{kj} &\leq 1 \quad \forall k \in \{1, \ldots, K\} \\
y_{kj} &\in \{0, 1\} \quad \forall kj
\end{aligned}$$

All-capacity knapsack problem

Given a set of knapsacks $\{1, \ldots, K\}$ with capacities W_k and usage costs c_k, solve the following problem:

$$
\begin{aligned}
\max \sum_{i=1}^{I} p_i x_i - \sum_{k=1}^{K} c_k z_k & & & \\
\text{s.t. } \sum_{i=1}^{I} w_i x_i & \leq & \sum_{k=1}^{K} W_k z_k & \\
\sum_{k=1}^{K} z_k & = & 1 & \forall k \in \{1, \ldots, K\} \\
x_i & \in & \{0, 1\} & \forall i \in \{1, \ldots, I\} \\
z_k & \in & \{0, 1\} & \forall k \in \{1, \ldots, K\}
\end{aligned}
$$

All-capacity knapsack problem: solution

Assume $0 = W_0 \leq W_1 \leq W_2 \leq \ldots \leq W_K$ and define $y_k = \sum_{\kappa=k}^{K} z_\kappa$

$$
\begin{aligned}
\max \sum_{i=1}^{I} p_i x_i - \sum_{k=1}^{K} (c_k - c_{k-1}) y_k & & & \\
\text{s.t.} \sum_{i=1}^{I} w_i x_i &\leq \sum_{k=1}^{K} (W_k - W_{k-1}) y_k & & \\
y_k &\geq y_{k+1} & \forall k \in \{1, \ldots, K-1\} & \\
x_i &\in \{0, 1\} & \forall i \in \{1, \ldots, I\} & \\
y_k &\in \{0, 1\} & \forall k \in \{1, \ldots, K\} &
\end{aligned}
$$