Lecture 12: Knapsack Problems

Prof. Krishna R. Pattipati
Dept. of Electrical and Computer Engineering
University of Connecticut
Contact: krishna@engr.uconn.edu; (860) 486-2890

Outline

- Why solve this problem?
- Various versions of Knapsack problem
- Approximation algorithms
- Optimal algorithms
  - Dynamic programming
  - Branch-and-bound

0–1 Knapsack problem

- A hitch-hiker has to fill up his knapsack of size V by selecting from among various possible objects those which will give him maximum comfort
- Suppose want to invest (all or in part) a capital of V dollars among n possible investments with different expected profits and with different investment requirements to maximize total expected profit
- Other applications: cargo loading, cutting stock
- Mathematical formulation of 0–1 Knapsack problem

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

$$\text{s.t.} \quad \sum_{i=1}^{n} w_i x_i \leq V$$

$$x_i \in \{0,1\}$$

- A knapsack is to be filled with different objects of profit p_i and of weights w_i without exceeding the total weight V
- p_i & w_i are integers (if not, scale them)
- $w_i \leq V, \forall i$
- $\sum_{i=1}^{n} w_i > V$

Other related formulations

- Bounded Knapsack problem
  - Suppose there are b_i items of type i

    $\Rightarrow$ Change $x_i \in \{0, 1\}$ to $0 \le x_i \le b_i$ and x_i integer
  - Unbounded Knapsack problem

    $\Rightarrow b_i = \infty$, $0 \le x_i \le \infty$, x_i integer
  - Subset-sum problem
    - Find a subset of weights whose sum is closest to, without exceeding capacity

$$\begin{aligned} &\max \sum_{i=1}^{n} w_i x_i \\ &\text{s.t.} \quad \sum_{i=1}^{n} w_i x_i \le V \\ &\qquad x_i \in \{0,1\} \end{aligned}$$

    - Knapsack problem with $w_i = p_i$
    - Subset-sum is NP-hard $\Rightarrow$ knapsack is NP-hard
  - Change-making problem

$$\begin{aligned} &\max \sum_{i=1}^{n} x_i \\ &\text{s.t.} \quad \sum_{i=1}^{n} w_i x_i = V \\ &\qquad 0 \le x_i \le b_i, x_i \text{ an integer}, i = 1, \ldots, n \end{aligned}$$

    - b_i finite $\Rightarrow$ bounded change-making problem
    - $b_i = \infty \Rightarrow$ unbounded change-making problem

Other related formulation

- 1-dimensional knapsack with a cost constraint

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

$$\text{s.t.} \quad \sum_{i=1}^{n} w_i x_i \le V$$

$$\sum_{i=1}^{n} c_i x_i \le C$$

$$x_i \in \{0,1\}$$

- Multi-dimensional knapsack (m knapsacks)

$$\max \sum_{i=1}^{n} \sum_{j=1}^{m} p_i x_{ij}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} w_i x_{ij} \le V; \; j = 1, \ldots, m$$

$$\sum_{j=1}^{m} x_{ij} \le 1; i = 1, \ldots, n$$

$$x_{ij} \in \{0,1\}; i = 1, \ldots, n; \; j = 1, \ldots, m$$

- Multi-dimensional knapsack in which profit and weight of each item depends on the knapsack selected for the item

$$\max \sum_{i=1}^{n} \sum_{j=1}^{m} p_{ij} x_{ij}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} w_{ij} x_{ij} \le V_j; \; j = 1, \ldots, m$$

$$\sum_{j=1}^{m} x_{ij} \le 1; i = 1, \ldots, n$$

$$x_{ij} \in \{0,1\}; i = 1, \ldots, n; \; j = 1, \ldots, m$$

Other related formulation

- Loading problem or variable-sized bin-packing problem
  - Given n objects with known volumes w_i & m boxes with limited capacity c_j, $j=1,\ldots,m$, minimize the number of boxes used
  - $y_i = 1$ if box j is used
  - $x_{ij} = 1$ if object i is put in box j

$$\min \sum_{j=1}^{n} y_j$$

$$\text{s.t.} \quad \sum_{i=1}^{m} x_{ij} = 1; i = 1,\ldots,n$$

$$\sum_{i=1}^{n} w_i x_{ij} \le c_j y_j; \; j = 1,\ldots,m$$

$$y_j, \; x_{ij} \in \{0,1\}$$

  - $c_j = c \Rightarrow$ bin-packing problem
- See S. Martello and P. Toth, Knapsack Problems: Algorithms and Computer Implementation, John Wiley, 1990 for an in-depth survey of these problems
- Here we consider only 0–1 Knapsack problem

Relaxed LP version of Knapsack problem

- Let us consider relaxed LP version of Knapsack problem
  - Gives us an upper and lower bound on the Knapsack problem
  - Assume that objects are ordered as

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

  - LP relaxation to find an upper bound

$$\begin{array}{lll} \max \sum_i p_i x_i & \text{relax } 0-1 & \max \sum_i p_i x_i \\ \text{s.t.} \quad \sum_i w_i x_i \leq V & \Rightarrow & \text{s.t.} \quad \sum_i w_i x_i \leq V \\ x_i \in \{0,1\} & \text{constraints} & 0 \leq x_i \leq 1 \\ f^* & \leq & f_{LP} \end{array}$$

  - Dual of relaxed LP

$$\min \left\{ \lambda V + \sum_{i=1}^{n} \mu_i \right\} \quad \text{s.t.} \quad w_i \lambda + \mu_i \geq p_i, \quad \lambda, \mu_i \geq 0$$

$$\Rightarrow \min \left\{ \lambda V + \sum_{i=1}^{n} \mu_i \right\} \quad \text{s.t.} \quad \mu_i \geq p_i - w_i \lambda, \quad \mu_i \geq 0$$

$$\Rightarrow \min_{\lambda \geq 0} \left\{ \lambda V + \sum_{i=1}^{n} \max(0, p_i - w_i \lambda) \right\}$$

$$\lambda^* = \frac{p_{r+1}}{w_{r+1}}; r : \sum_{i=1}^{r} w_i < V < \sum_{i=1}^{r+1} w_i$$

$$\mu_i^* = \max(0, p_i - w_i \frac{p_{r+1}}{w_{r+1}})$$

Relaxed LP version of Knapsack problem

- Complementary slackness condition
  - $x_i = 1 \Rightarrow \mu_i > 0$ and $w_i\lambda + \mu_i = p_i$
  - $x_i < 1 \Rightarrow \mu_i = 0$
- Optimal solution is as follows

if $\sum_{i=1}^{r} w_i < V < \sum_{i=1}^{r+1} w_i$ then

$x_i = 1, i = 1, \ldots, r \Rightarrow \mu_i = (p_i - \lambda w_i)$

$x_i = 0, i = r+2, \ldots, n \Rightarrow \mu_i = 0$

$$x_{r+1} = \frac{V - \sum_{i=1}^{r} w_i}{w_{r+1}}; \mu_{r+1} = 0$$

$\Rightarrow$ one upper bound is:

$$f^* \le f_{LP}^* = \sum_{i=1}^{r} p_i + \left(V - \sum_{i=1}^{r} w_i\right)\left(\frac{p_{r+1}}{w_{r+1}}\right)$$

optimal $\lambda = \frac{p_{r+1}}{w_{r+1}}$

$$\Rightarrow \mu_i = \left(p_i - p_{r+1}\frac{w_i}{w_{r+1}}\right); i = 1, \ldots, r$$

- Clearly, f_{LP}^* = optimal dual objective function value

LP relaxation upper bound

- LP relaxation solution provides the upper bound $U_1 = \text{int}(f_{LP}^*) \leq 2f^*$
  - Both $[p_1, p_2, \ldots, p_r]$ and p_{r+1} are feasible Knapsack solutions
  - Feasible solution $\leq f^*$
  - $U_1 \leq$ sum of two feasible solution $\leq 2f^*$
- We can obtain an even tighter bound than LP relaxation (Martello and Toth)
  - Consider the possibility that $x_{r+1} = 1$ or $x_{r+1} = 0$

$x_{r+1} = 0 \Rightarrow x_{r+2}$ could be a fraction

$$\Rightarrow \hat{U} = \sum_{i=1}^{r} p_i + \left\lfloor \frac{\left(V - \sum_{i=1}^{r} w_i\right)}{w_{r+2}} (p_{r+2}) \right\rfloor$$

$x_{r+1} = 1 \Rightarrow x_r$ could be a fraction

$$\Rightarrow \tilde{U} = \sum_{i=1}^{r-1} p_i + p_{r+1} + \left\lfloor \frac{\left(V - \sum_{i=1}^{r-1} w_i - w_{r+1}\right)}{w_r} p_r \right\rfloor$$

$$= \sum_{i=1}^{r} p_i + \left\lfloor p_{r+1} - \left\lceil \frac{w_{r+1} - \left(V - \sum_{i=1}^{r} w_r\right)}{w_r} \right\rceil p_r \right\rfloor$$

LP relaxation upper bound

- Clearly,

$$U_2 = \max\left\{\tilde{U}, \hat{U}\right\} \geq f^*$$

- Why is this a better upper bound than LP relaxation?
  - Clearly, $\hat{U} \leq U_1$
  - Can show that

$$U_1 - \tilde{U} = \left[\frac{p_r}{w_r} - \frac{p_{r+1}}{w_{r+1}}\right]\left[\sum_{i=1}^{r+1} w_i - V\right] \geq 0$$

Knapsack Example

$$\widehat{U} = \left\lfloor \frac{(V - \sum_{i=1}^{r} w_i)}{w_{r+2}} (p_{r+2}) \right\rfloor$$

$$\widetilde{U} = \left\lfloor \frac{\left(V - \sum_{i=1}^{r-1} w_i - w_{r+1}\right)}{w_r} p_r \right\rfloor$$

- $n = 8$
- $\underline{p} = [15, 100, 90, 60, 40, 15, 10, 1]$
- $\underline{w} = [2, 20, 20, 30, 40, 30, 60, 10]$
- $V = 102$
- Optimal solution: $\underline{x} = [1, 1, 1, 1, 0, 1, 0, 0]$
- Optimal value: $f^* = 280$
- LP relaxation: $(r + 1) = 5 \Rightarrow U_1 = 265 + \left(\frac{(30)(40)}{40}\right) = 295$
- Bound $\widehat{U} = 265 + \left(\frac{(30)(15)}{30}\right) = 280$
- Bound $\widetilde{U} = 245 + \left(\frac{(20)(60)}{30}\right) = 285$
- Maximum of latter two bounds = 285

- These bounds can be further improved by solving two continuous relaxed LPs with the constraints that $x_{r+1} = 1$ or $x_{r+1} = 0$, respectively
- Algorithm for solving Knapsack problem
  - Approximation algorithms
  - Dynamic programming
  - Branch-and-bound
- Although the problem is NP-hard, can solve problems of size $n = 100{,}000$ in a reasonable time
- Approximate algorithms…lower bounds on the optimal solution
  - Can obtain from a greedy heuristic

Greedy heuristic

- 0^{th} order greedy heuristic
  - Load objects on the basis of $\frac{p_i}{w_i}$

$$L(0) = \sum_{i:\frac{p_i}{w_i} \geq \frac{p_i+1}{w_i+1} \;\&\; \sum w_i < V} p_i \leq f^*$$

- K^{th} order greedy heuristic
  - We can obtain a series of lower bounds by assuming that a certain set of objects J are in the knapsack where

$$\sum_{i \in J} w_i < V$$

  - And assigning the rest on the basis of greedy method
  - This is basically **rollout** concept of dynamic programming

k^{th} order Greedy heuristic

- The bound can be computed as follows:
  - $z = V - \sum_{i \in J} w_i$
  - y=$\sum_{i \in J} p_i$
  - For $i = 1, \ldots, n$ do
    - If ($i \notin J$ & $w_i \leq z$) then
      - $J = J \cup i$
      - $z = z - w_i$
      - $y = y + p_i$
    - End if
  - End do
  - $L(J) = y \leq f^*$

k-approximation algorithm

- Idea: what if I consider all possible combination of objects of size $|J| = k$ & find the best lower bound & use it as a solution of the Knapsack problem
  - If $|J| = n$, optimal
    - We can control the complexity of the algorithm by varying k
      k-approximation algorithm…Knapsack(k)
    - $f(k) \leftarrow 0$
      - Enumerate all subset $J \subset \{1, \ldots, n\}$ such that

$$\sum_{i \in J} | J | \leq V \text{ and do}$$

$$f(k) \leftarrow \max\{ f(k), L(J) \}$$

  - Note: in practice, $k = 2$ would suffice to produce a solution within 2-5% of optimal
- Example

$$\max \; 100x_1 + 50x_2 + 20x_3 + 10x_4 + 7x_5 + 3x_6$$

$$\text{s.t.} \quad 100x_1 + 50x_2 + 20x_3 + 10x_4 + 7x_5 + 3x_6 \leq 165$$

$$\frac{p_i}{w_i} = 1, \forall i$$

$L(0) = 163 \ldots 0^{th}$ order heuristic

$k = 1: L(\{1\}) = 163, L(\{2\}) = 163, L(\{3\}) = 140, L(\{4\}) = 163, L(\{5\}) = 160, L(\{6\}) = 163$

$\Rightarrow$ best 1^{st} order heuristic solution = 163… in fact, this is optimal!

k-approximation algorithm

- Can we say anything about the performance of the approximation algorithm? Yes!
  - The recursive approximation algorithm Knapsack(k) provides a solution $f(k)$ ∋

$$\left|\frac{f^* - f(k)}{f^*}\right| \leq \frac{1}{k+1}$$

  - Before we provide a proof, let us consider its implications
    - If want a solution within ϵ of optimal

$$\frac{1}{k+1} \leq \epsilon \Rightarrow k \geq \frac{1}{\epsilon} - 1$$

    - Time complexity
      - ❖ Number of time greedy (0) is executed

$$\sum_{i=0}^{k} \binom{n}{i} \leq \sum_{i=0}^{k} n^i = \frac{n^{k+1}-1}{n-1} = O(n^k) \approx O(n^{\frac{1}{\epsilon}})$$

      - ❖ Each greedy takes $O(n)$ operations

Proof of the bound: setting the stage

- Let R^* be the set of objects included in the knapsack in an optimal solution

$$\Rightarrow \sum_{i \in R^*} p_i = f^* \;\&\; \sum_{i \in R^*} w_i \le V$$

- If $|R^*| \le k$, our approximation algorithm would have found it…so, assume otherwise

$$\Rightarrow |R^*| > k$$

- Suppose $(\hat{p}_i, \hat{w}_i)$, $1 \le i \le |R^*|$ be the set of objects in R^*
- Assume that we order these objects as follows:
  - First k: $\hat{p}_1 > \hat{p}_2 > \cdots > \hat{p}_k$ are the largest profits
  - The rest: $\frac{\hat{p}_i}{\hat{w}_i} > \frac{\hat{p}_{i+1}}{\hat{w}_{i+1}}, k < i < |R^*|$
- $\hat{p}_1 + \hat{p}_2 + \cdots \hat{p}_{|R^*|} = f^*$
- $f^* \ge \sum_{i=1}^{k} \hat{p}_i + \hat{p}_{k+1} \ge (k+1)\hat{p}_{k+t}; t = k+1, \ldots, |R^*|$

  $\Rightarrow \hat{p}_{k+t} \le \frac{f^*}{k+1}$

Proof of the k-approximation bound - 1

- Let us look at what *k*-approximation algorithm does … it must have looked at the set *J* of largest prices in *R** at least once … let

$$f_J = \sum_{i \in J} p_i = \sum_{i=1}^{k} \hat{p}_i$$

- Let us consider what happens when *k*-approximation algorithm executed this iteration
  - Suppose that the approximation algorithm did not include an object that is in the optimal solution
  - Let *l* be the first such object, that is, $l \notin J$
  - Suppose *l* corresponds to $(\hat{p}_m, \hat{w}_m)$ in *R**
- Why was *l* not included in the Knapsack(*k*)?
  - Residual capacity of knapsack $z < w_l = \hat{w}_m$
  - Must have included (*m* - 1) tasks

$$f(k) \geq \underbrace{\sum_{i=1}^{k} \hat{p}_i}_{f_J} + \overbrace{\sum_{i=k+1}^{m-1} \hat{p}_i + \frac{\hat{p}_m}{\hat{w}_m} \underbrace{\left(V - \sum_{i=1}^{m-1} \hat{w}_i - z \right)}_{\delta}}^{S}$$

$$f(k) \geq \underbrace{\sum_{i=1}^{k} \hat{p}_i}_{f_J} + S$$

Proof of the k-approximation bound - 2

- Also, from LP relaxation bound

$$\sum_{i=m}^{|R^*|} \hat{p}_i \le \sum_{i=1}^{m-1} \hat{p}_i + \frac{\hat{p}_m}{\hat{w}_m}\left(V - \sum_{i=1}^{m-1} \hat{w}_i\right)$$

- By definition

$$\begin{aligned}
f^* &= f_J + \sum_{i=k+1}^{|R^*|} p_i \\
&= f_J + \sum_{i=k+1}^{m-1} \hat{p}_i + \sum_{i=m}^{|R^*|} \hat{p}_i \\
&\le f_J + S - \frac{\hat{p}_m}{\hat{w}_m}\delta + \frac{\hat{p}_m}{\hat{w}_m}\left(V - \sum_{i=1}^{m-1} \hat{w}_i\right) \\
&= f_J + S + \frac{\hat{p}_m}{\hat{w}_m} z \\
&< f_J + S + \hat{p}_m \le f(k) + \hat{p}_m \\
\Rightarrow \frac{f^* - f(k)}{f^*} &< \frac{\hat{p}_m}{f^*} \le \frac{1}{k+1}
\end{aligned}$$

Proof of the k-approximation bound - 3

- Since $\hat{p}_m$ is one of $\hat{p}_{k+1} \cdots \hat{p}_{|R^*|} \Rightarrow \hat{p}_m \leq \bar{p} = (k+1)^{\text{st}}$ largest element of $\hat{p}_1 \cdots \hat{p}_m$

$$\Rightarrow \frac{f^* - f(k)}{f^*} \leq \frac{\hat{p}_m}{f^*} \leq \frac{\hat{p}_m}{f(k)} \leq \frac{\bar{p}}{f(k)}$$

$$\Rightarrow \left| \frac{f^* - f(k)}{f^*} \right| \leq \min\left\{ \frac{1}{k+1}, \frac{\bar{p}}{f(k)} \right\}$$

- Example
  - $\underline{p}$ = [11, 21, 31, 33, 43, 53, 55, 65] optimal: 1 2 3 5 6
  - $\underline{w}$ = [1, 11, 21, 23, 33, 43, 45, 55] f^* = 159, $\sum w_i = 109 < 110$
  - V = 110
  - $k = 0 \Rightarrow \underline{x}$ = [1, 1, 1, 1, 1, 0, 0, 0], f = 139 $\Rightarrow \frac{f^* - f}{f^*} = \frac{20}{159} = 0.126, \sum w_i = 89$
  - $k = 1 \Rightarrow \underline{x}$ = [1, 1, 1, 1, 0, 0, 1, 0], f = 151 $\Rightarrow \frac{f^* - f}{f^*} = \frac{8}{159} = 0.05, \sum w_i = 101$
  - $k = 2 \Rightarrow \underline{x}$ = [1, 1, 1, 0, 1, 1, 0, 0], f = 159 $\Rightarrow \frac{f^* - f}{f^*} = \frac{0}{159} = 0, \sum w_i = 109$

Example

$$\max \; 9x_1 + 5x_2 + 3x_3 + x_4$$

$$\text{s.t.} \quad 7x_1 + 4x_2 + 3x_3 + 2x_4 \le 10$$

note: $\frac{9}{7} \ge \frac{5}{4} \ge \frac{3}{3} \ge \frac{1}{2}$ (OK)

$k = 0 : L(\{0\}) = 12, \underline{x} = [1,0,1,0]$

$k = 1 : L(\{1\}) = 12, L(\{2\}) = 9, L(\{3\}) = 12, L(\{4\}) = 10$

$k = 2 : L(\{1,3\}) = 12, L(\{1,4\}) = 10, L(\{2,3\}) = 9, L(\{2,4\}) = 10, L(\{3,4\}) = 10, \text{etc.}$

$f(2) = 12 = f^*$

- Usually $k = 1$ or $k = 2$ works OK (within 2-5% of optimal)

Dynamic programming approach

- Views the problem as a sequence of decisions related to variables $x_1, x_2, \ldots, x_n$
- That is, we decide whether $x_1 = 0$ or 1, then consider x_2, and so on
- Consider the "generalized" Knapsack problem:

$$\left.\begin{aligned} &\max \sum_{i=k}^{l} p_i x_i \\ &\text{s.t.} \quad \sum_{i=k}^{l} w_i x_i \le U \\ &\qquad x_i \in \{0,1\}, i = k, \ldots, l \end{aligned}\right\} \text{Knap}(k, l, U)$$

- Actually want to solve: Knap (1, n, V)
- To solve Knap (1, n, V), the DP algorithm employs the principle of optimality
  - **"The optimal sequence of decisions has the property that, whatever the initial state and past decisions are, the remaining decisions must constitute an optimal decision sequence with regard to the state resulting from the current decision"**

Backward DP approach

- Suppose $f_0^*(V)$ = optimal value for Knap (1, n, V)
- $f_j^*(U)$ = optimal value for Knap (j + 1, n, V), $1 \le j \le n-1$
- Clearly, $f_n^*(U) = 0, \forall U$
- Let us look at $f_0^*(V)$ … suppose we make the decision for x_i
- If $x_1 = 0, x_2 \cdots x_n$ must be the optimal solution for Knap (2, n, V) = $f_1^*(V)$
- If $x_1 = 1, x_2 \cdots x_n$ must be the optimal solution for Knap (2, n, $V - w_1$) = $f_1^*(V - w_1)$

$$f_0^*(V) = \max\left\{ f_1^*(V),\ f_1^*(V - w_1) + p_1 \right\}$$

$x_1 = 0$ $\quad$ $x_1 = 1$

- Similarly, $f_j^*(U)$ = optimal solution of Knap (j + 1, n, U)

$$f_j^*(U) = \max\left\{ f_{j+1}^*(U),\ f_{j+1}^*(U - w_{j+1}) + p_{j+1} \right\},\ j = n-1, \ldots, 0$$

$x_{j+1} = 0$ $\quad$ $x_{j+1} = 1$

$f_n(U) = 0, \forall U$

Forward DP

- Start with $f_n^*(U) = 0, \forall U = 0, 1, 2, \ldots, V$…successively evaluate $f_{n-1}^*(U), \forall U$, then $f_{n-2}^*(U), \forall U$, etc.
- Note: decision x_{j+1} depends on x_{j+2},…, x_n ⇒ <u>backward recursion</u>
- Alternately, suppose we know the optimal solution to Knap (1, j, U) … we could have solved it in one of two ways:
  - $x_j = 0$ and knew the solution to Knap (1, j–1, U) = $S_{j-1}^*(U)$
  - $x_j = 1$ and knew the solution to Knap (1, j–1, U–w_j) = $S_{j-1}^*(U - w_j) + p_j$
  - So we can get a <u>forward recursion</u>

$$S_j^*(U) = \max\left\{S_{j-1}^*(U), S_{j-1}^*(U - w_j) + p_j\right\}$$

where

$$S_0(U) = 0, \forall U \geq 0 \text{ and } S_0(U) = -\infty, \forall U < 0$$

  - Note: decision x_j depends on x_1,…, x_{j-1} ⇒ <u>forward recursion</u>

Example

$$S_j^*(U) = \max\left\{S_{j-1}^*(U), S_{j-1}^*(U - w_j) + p_j\right\}$$

$$S_0(U) = 0, \forall U \geq 0 \text{ and } S_0(U) = -\infty, \forall U < 0$$

- $\underline{p}$ = [1, 2, 5]; $\underline{w}$ = [2, 3, 4]; V = 6

U	0 (S_0, x_0)	1 (S_1, x_1)	2 (S_2, x_2)	3 (S_3, x_3)
0	(0, –)	(0 ,0)	(0 ,0)	(0, 0)
1	(0, –)	(0 ,0)	(0, 0)	(0, 0)
2	(0 ,–)	(1 ,1)	(1, 0)	(1, 0)
3	(0, –)	(1, 1)	(2 ,1)	(2, 0)
4	(0 ,–)	(1, 1)	(2, 1)	(5, 1)
5	(0, –)	(1, 1)	(3, 1)	(5, 1)
6	(0, –)	(1, 1)	(3, 1)	(6 ,1)

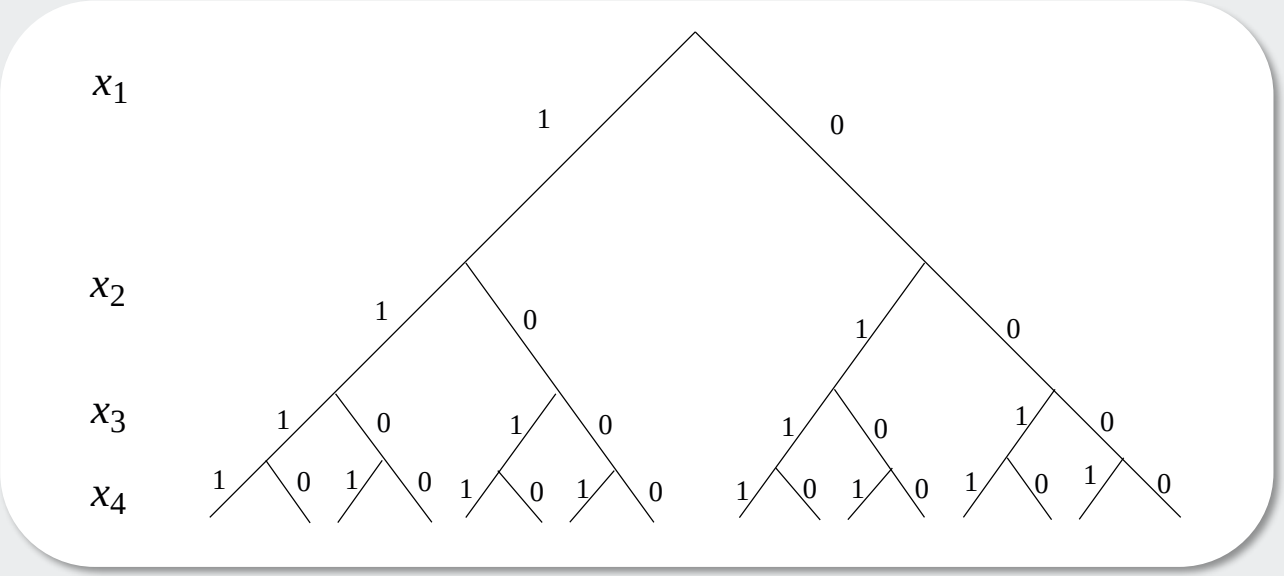
$x_1 = 1; x_2 = 0; x_3 = 1$

$f^* = 6$

- Computational load and storage $O(nV)$
- It is considered a pseudo-polynomial algorithm, since V is not bounded by a $\log_2 V$ function
- By encoding $\underline{x}$ as a bit string, can reduce storage to $O\left(\left(1 + \left\lceil \frac{n}{d} \right\rceil\right) V\right)$, where d is the word length on the computer (see Martello and Toth's book)

Branch-and-bound method: Basic Idea

- Finds the solution via a systematic search of the solution space
- For Knapsack problem, the solution space consists of 2^n vectors of 0s and 1s
- The solution space can be represented as a tree



  - Leaves correspond to "potential solutions," not necessarily feasible
  - Key:
    - Don't want to search the entire tree
    - Don't want to generate infeasible solutions
    - Don't want to generate tree nodes that do not lead to a better solution than the one on hand (⇒ a bounding function)

B&B for knapsack problem

- Bounding function can be derived from the LP relaxation
  - Given the current contents of the knapsack J with profit P and weight W, we can construct the bounding function as follows:
    - Suppose k is the last object considered, i.e., p & W correspond to $y_1 \cdots y_k$ (y_i = temp x_i), then

$$\max\ p + \sum_{i=k+1}^{n} p_i y_i$$

$$\text{s.t.} \quad \sum_{i=k+1}^{n} w_i y_i \le V - W$$

$$0 \le y_i \le 1$$

    - If the UB $\le$ current best solution, then further search from a tree node is worthless
      - So, backtrack ⇒ move to the right, if it was in the left one or go back to the first variable with value

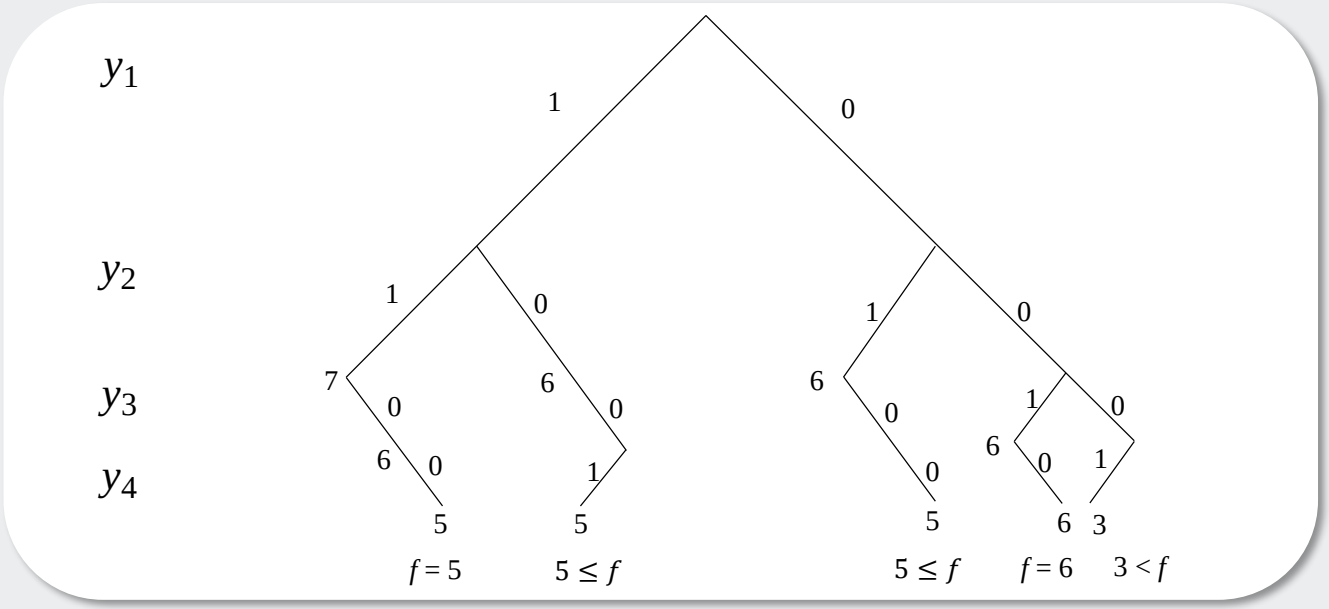
Example

- We will explain the algorithm by means of an example (Syslo, Deo, and Kowalik)

$$\begin{aligned} \max \quad & 2x_1 + 3x_2 + 6x_3 + 3x_4 \\ \text{s.t.} \quad & x_1 + 2x_2 + 5x_3 + 4x_4 \le 5 \\ & x_i \in \{0,1\} \end{aligned}$$

- Initially $p = W = 0$, $k = 0$ & $f = -1$
  - Partial solution: $y_1 = 1$, $y_2 = 1$ with profit = 5 and
  - Weight = 3 $\Rightarrow$ $p = 5$, $W = 3$, $k + 1 = 3$
  - Bound: $b = 5 + \text{int}\left(\frac{(2)(6)}{5}\right) = 7$
  - Bound > f … perform a forward move

Example



- $y_3 = 0$ (since can not fit in it) and $y_4 = 1$ is infeasible
- Set $y_4 = 0$ and $k + 1 = 5 > n = 4$ and $f = -1$, the current solution is the best found so far ... so, set $f = 5$ and $k = 4$
- Backtrack to the last object assigned to knapsack ... remove the object ... $y_2 = 0 \Rightarrow p = 2$, $W = 1$, $y_3 = 1$ fails $\Rightarrow y_3 = 0$ … but, $y_4 = 1$ O.K. $\Rightarrow \underline{y} = [1, 0, 0, 1]$ and $p = 5$ … this does not improve the current best solution of $f = 5$
- Backtrack to y_1 … set $y_1 = 0$ after assigning $y_2 = 1$, it returns bound $b = 6$ … $b > f$, we do another forward move ... the next call to bound results in a bound $b \le f \Rightarrow$ backward move
- Backtrack to y_2 … set $y_2 = 0$ after assigning $y_3 = 1$, it returns bound $b = 6$ and a solution $p = 6$, $W = 5$, which is better than the best solution f.... Actually you could you have stopped here because upper bound = feasible solution
- Finally, $y_3 = 0$, and bound assigns $y_4 = 1$ and returns 3 ... since there are no other objects in the knapsack to be removed, the algorithm terminates
- There exist variations that provide better computational performance ... see the book by Martello and Toth

Summary

- Various versions of Knapsack problem
- Approximation algorithms
- Optimal algorithms
  - Dynamic programming
  - Branch-and-bound