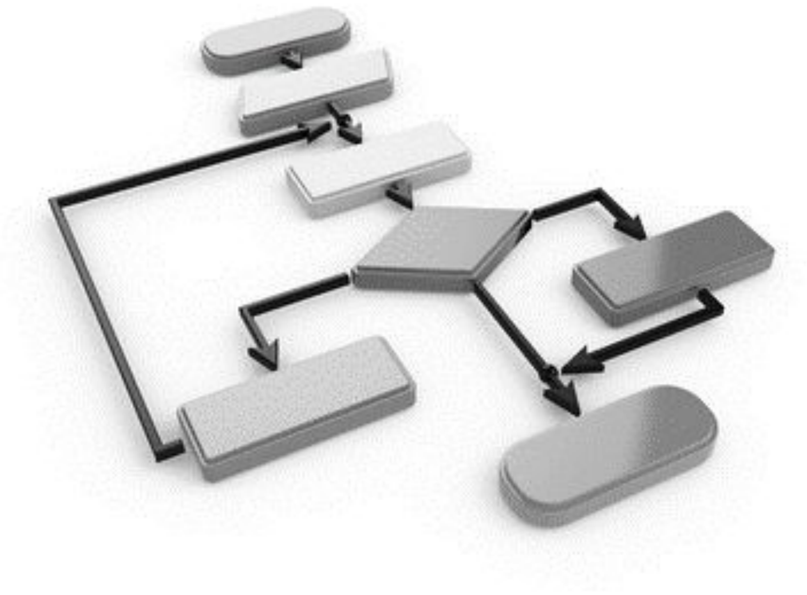


Dynamic Programming: Knapsack

CSE 301: Combinatorial Optimization

The Knapsack Problem

The 0-1 knapsack problem

A thief robbing a store finds n items: the i-th item is worth v_i dollars and weights w_i pounds (v_i, w_i integers)

The thief can only carry W pounds in his knapsack

Items must be taken entirely or left behind

Which items should the thief take to maximize the value of his load?

The fractional knapsack problem

Similar to above

The thief can take fractions of items

The 0-1 Knapsack Problem

Thief has a knapsack of capacity W

There are n items: for i-th item value v_i and weight w_i

Goal:

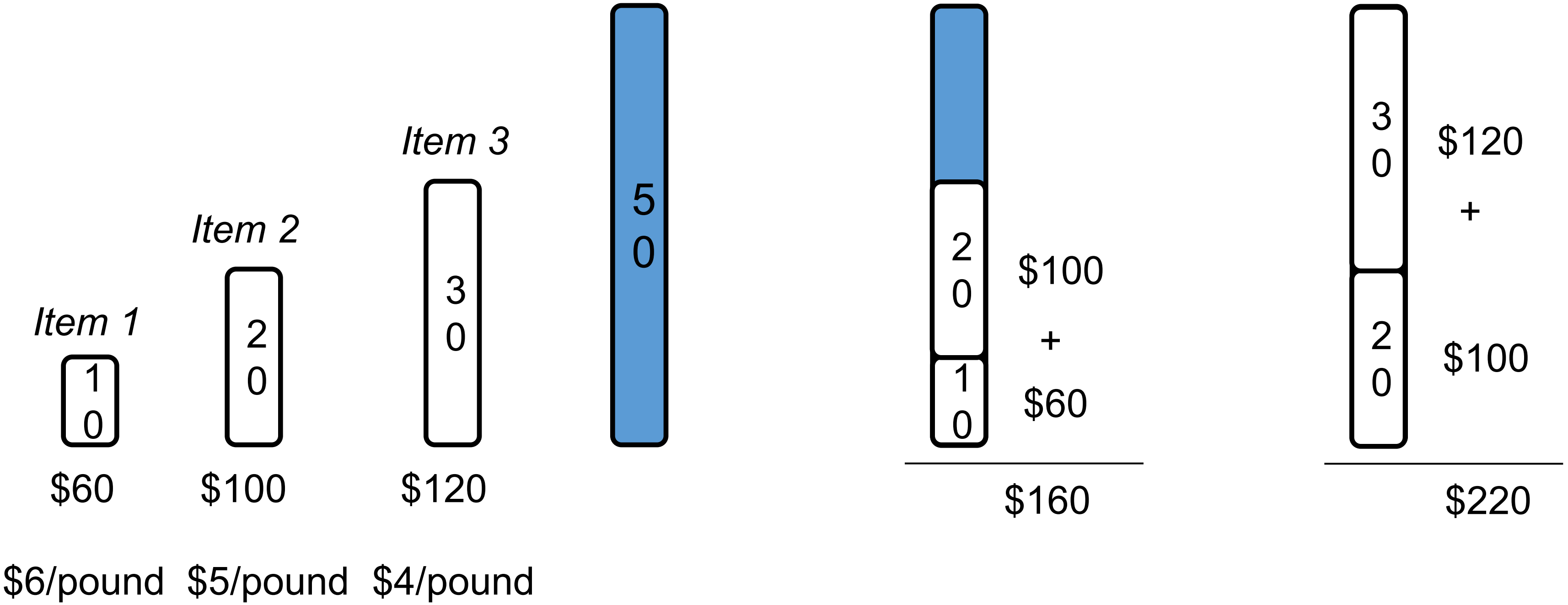
find x_i such that for all $x_i = \{0, 1\}$, $i = 1, 2, .., n$

$\sum w_i x_i \leq W$ and

$\sum x_i v_i$ is maximum

0-1 Knapsack - Greedy Strategy

E.g.:



- None of the solutions involving the greedy choice (item 1) leads to an optimal solution
  - The greedy choice property does not hold

0-1 Knapsack - Dynamic Programming

$P(i, w)$ – the maximum profit that can be obtained from items 1 to i, if the knapsack has capacity w

Case 1: thief takes item i

$$P(i, w) = v_i + P(i - 1, w - w_i)$$

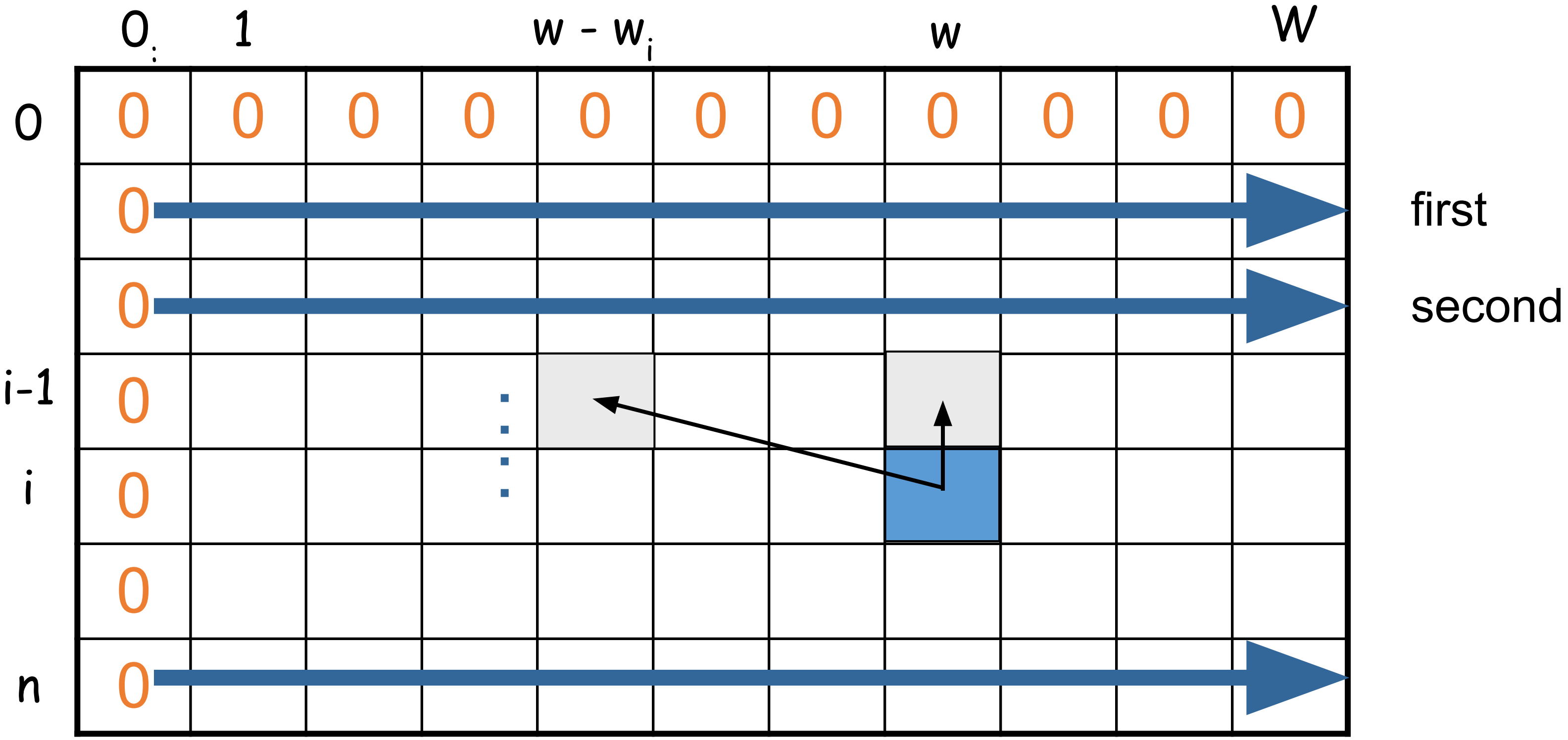
Case 2: thief does not take item i

$$P(i, w) = P(i - 1, w)$$

0-1 Knapsack - Dynamic Programming (DP)

$$P(i, w) = \max\{\overbrace{v_i + P(i-1, w-w_i)}^{\text{Item i was taken}}, \overbrace{P(i-1, w)}^{\text{Item i was not taken}}\}$$

$$P(i, w) = 0, \text{ if } i = 0 \text{ or } w = 0$$



0-1 Knapsack – DP Algorithm

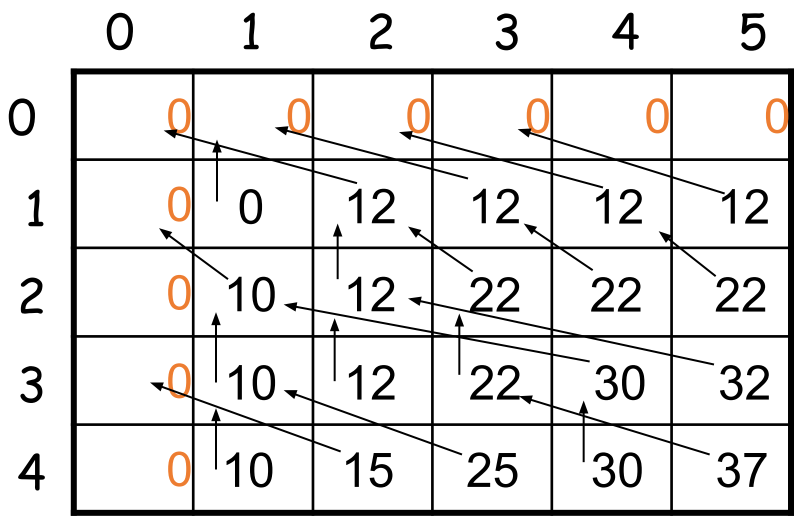
```
for i ← 0 to n
    do P(i,0) = 0
for w ← 0 to W
    do P(0,w) = 0
for i from 1 to n
    do for w from 0 to W
            do if $w_i$ > w //if the capacity is not enough
          then P(i, w) = P(i-1, w)
           else
          P(i,w) = max{ $v_i$ + P(i-1, w-$w_i$), P(i-1, w) }
```

Example:

W = 5

Item	Weight	Value
1	2	12
2	1	10
3	3	20
4	2	15

$P(i, w) = \max\{v_i + P(i - 1, w-w_i), P(i - 1, w)\}$



	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0	10	12	22	22	22
3	0	10	12	22	30	32
4	0	10	15	25	30	37

P(1, 1) = P(0, 1) = 0

P(1, 2) = max{12+P(0,0), 0} = max{12+0, 0} = 12

P(1, 3) = max{12+P(0,1), 0} = max{12+0, 0} = 12

P(1, 4) = max{12+P(0,2), 0} = max{12+0, 0} = 12

P(1, 5) = max{12+P(0,3), 0} = max{12+0, 0} = 12

P(2, 1)= max{10+P(1,0), 0} = 10

P(2, 2)= max{10+P(1,1), P(1,2)}
= 12

P(2, 3)= max{10+P(1,2), P(1,3)}
= max(10+12,12) = 22

P(2, 4)= max{10+12, 12} = 22

P(2, 5)= max{10+12, 12} = 22

P(3, 1)= P(2,1) = 10

P(3, 2)= P(2,2) = 12

P(3, 3)= max{20+0, 22}=22

P(3, 4)= max{20+10,22}=30

P(3, 5)= max{20+12,22}=32

P(4, 1)= P(3,1) = 10

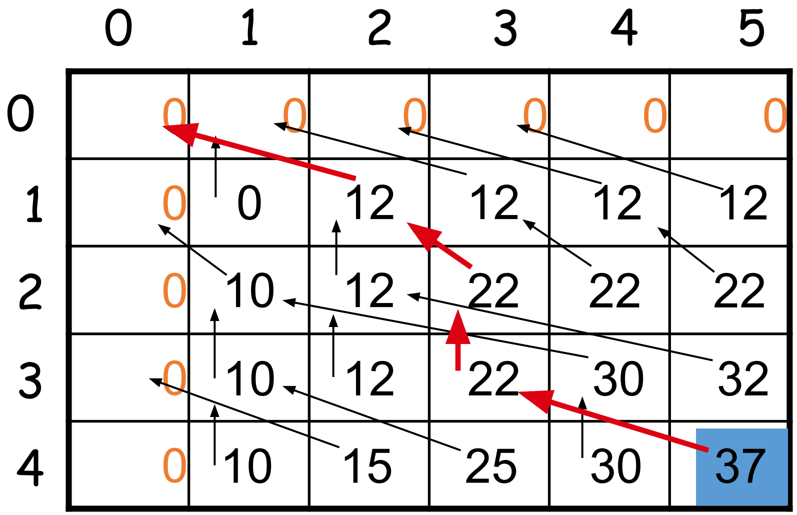
P(4, 2)= max{15+0, 12} = 15

P(4, 3)= max{15+10, 22}=25

P(4, 4)= max{15+12, 30}=30

P(4, 5)= max{15+22, 32}=37

Reconstructing the Optimal Solution



	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0	10	12	22	22	22
3	0	10	12	22	30	32
4	0	10	15	25	30	37

- Item 4
- Item 2
- Item 1

- Start at *P(n, W)*
- When you go left-up ⇒ item i has been taken
- When you go straight up ⇒ item i has not been taken