CH-231-A

Algorithms and Data Structures

ADS

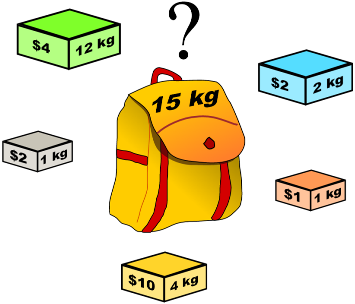
Lecture 30

Dr. Kinga Lipskoch

Spring 2022

The Knapsack Problem (1)

A thief breaks into a museum. Fabulous paintings, sculptures, and jewels are everywhere. The thief has a good eye for the value of these objects, and knows that each will fetch hundreds or thousands of dollars on the clandestine art collector's market. But, the thief has only brought a single knapsack to the scene of the robbery, and can take away only what he can carry. What items should the thief take to maximize the haul?



The Knapsack Problem (2)

Problem:

- Given some items, pack the knapsack to get the maximum total value.
- Each item has some weight and some value.
- Total weight that we can carry is no more than some fixed number W.
- We must consider weights of items as well as their values.

Item #	Weight	Value
1	1	8
2	3	6
3	5	5

The Knapsack Problem (3)

- Given a knapsack with maximum capacity W, and a set S consisting of n items
- Each item i has some weight w_i and value v_i (assuming that all w_i and W are integer values)
- How to pack the knapsack to achieve maximum total value of packed items?
- Mathematically:

$$\max \sum_{i \in T} v_i \text{ subject to } \sum_{i \in T} w_i \leq W$$

where T is a set of indices of the items from another set $S' \subseteq S$

Brute-Force Approach for the Knapsack Problem

Algorithm:

- Generate all 2^n subsets
- Discard all subsets whose sum of the weights exceed W (not feasible)
- Select the maximum total benefit of the remaining (feasible) subsets

Time complexity: $O(2^n)$

Example: Brute-Force Approach for the Knapsack Problem

$S = \{(item_1, 5, \$70), (item_2, 10, \$90), (item_3, 25, \$140)\}, W = 25$

Subsets:

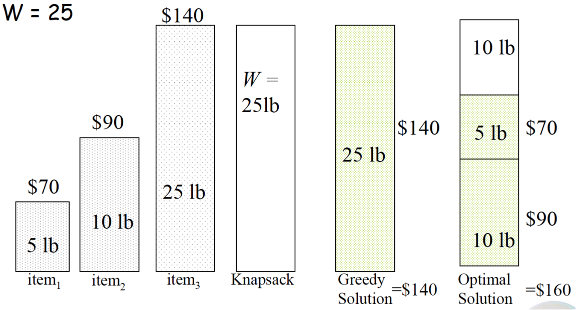
1. $\{\}$ $Profit = \$0$
2. $\{(item_1, 5, \$70)\}$ $Profit = \$70$
3. $\{(item_2, 10, \$90)\}$ $Profit = \$90$
4. $\{(item_3, 25, \$140)\}$ $Profit = \$140$
5. $\{(item_1, 5, \$70), (item_2, 10, \$90)\}$ $Profit = \$160$
6. $\{(item_2, 10, \$90), (item_3, 25, \$140)\}$ exceeds W
7. $\{(item_1, 5, \$70), (item_3, 25, \$140)\}$ exceeds W
8. $\{(item_1, 5, \$70), (item_2, 10, \$90), (item_3, 25, \$140)\}$ exceeds W

Greedy Approach for the Knapsack Problem

- Use a greedy approach to be more efficient
- What would be the greedy choice?
  - Maximum beneficial item
  - Minimum weight item
  - Maximum weight item
  - Maximum value per unit item
- Asymptotic time complexity: $O(n \lg n)$

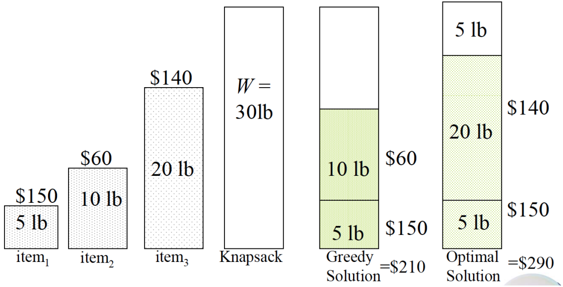
Greedy 1: Maximum Beneficial Item

$$S = \{(item_1, 5, \$70), (item_2, 10, \$90), (item_3, 25, \$140)\}, W = 25$$



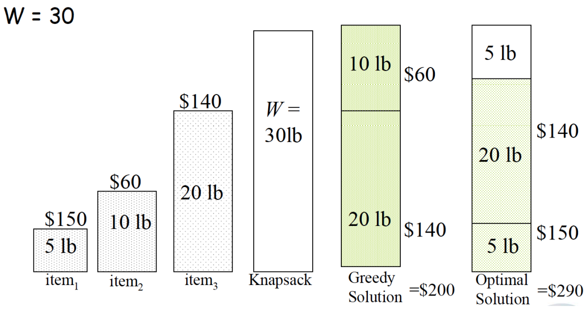
Greedy 2: Minimum Weight Item

$$S = \{(item_1, 5, \$150), (item_2, 10, \$60), (item_3, 20, \$140)\}, W = 30$$



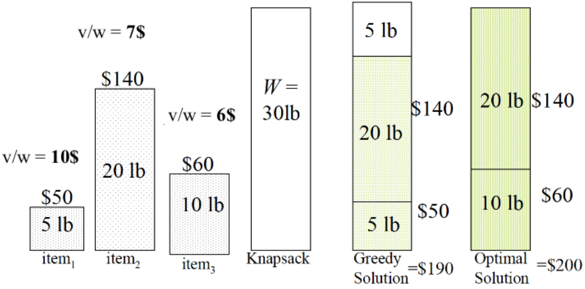
Greedy 3: Maximum Weight Item

$$S = \{(item_1, 5, \$150), (item_2, 10, \$60), (item_3, 20, \$140)\}, W = 30$$



Greedy 4: Maximum Value per Weight

$$S = \{(item_1, 5, \$50), (item_2, 20, \$140), (item_3, 10, \$60)\}, W = 30$$

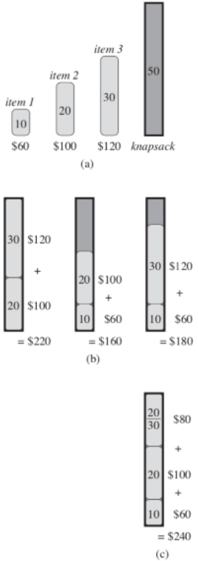


Conclusions: Greedy Approach for the Knapsack Problem

- As already mentioned, the locally optimal choice of a greedy approach does not necessary lead to a globally optimal one.
- For the knapsack problem, the greedy approach actually fails to produce a globally optimal solution.
- However, it produces an approximation, which sometimes is good enough.

0-1 vs. Fractional Knapsack Problem

- 0-1 knapsack problem
  - Either take (1) or leave an object (0)
  - Greedy fails to produce global optimum
- fractional knapsack problem
  - You can take fractions of an object
  - Greedy strategy: value per weight $v/w \rightarrow$ begin taking as much as possible of item with greatest v/w , then with next greater v/w, ...
  - Leads to global optimum (proof by contradiction)
- What is the difference?



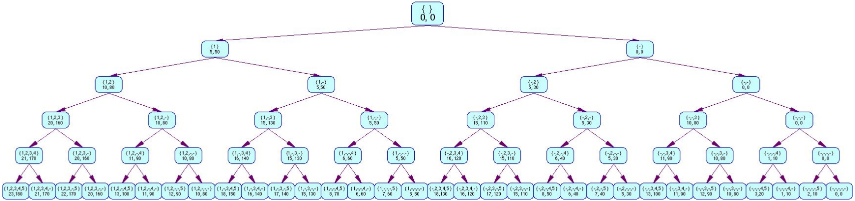
Alternatives for 0-1 Knapsack (1)

Brute-Force:

- Benefit: it finds the optimum
- Drawback: it takes very long - O(2^n)
- Because recomputing the results of the same subproblems over and over again

State Tree for the Knapsack Problem

```
Assume nodes 1-5 with given "costs" & "benefits"
1: 5,50    2: 5,30    3: 10,80    4: 1,10    5: 2,10
```



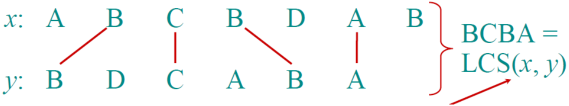
Alternatives for 0-1 Knapsack (2)

Dynamic programming:

- Optimal substructure:
  - optimal solution to problem consists of optimal solutions to subproblems
- Overlapping subproblems:
  - few subproblems in total, many recurring instances of each
- Main idea:
  - use a table to store solved subproblems

Dynamic Programming: Problem

- Given two sequences $x[1..m]$ and $y[1..n]$, find a longest subsequence common to both of them.
- Example:



Brute-Force Solution

Check every subsequence of $x[1..m]$ to see if it is also a subsequence of $y[1..n]$.

Analysis:

- Checking per subsequence is done in $O(n)$.
- As each bit-vector of m determines a distinct subsequence of x, x has 2^m subsequences.
- Hence, the worst-case running time is $O(n \cdot 2^m)$, i.e., it is exponential.