

Lecture 12: Greedy Algorithms

Fractional Knapsack Problem

Dr. Khalil Yousef

Adopted from the Slides of the ECE 608 Computational Models and Methods Course at Purdue University

Read Chapter 16 of *Introduction to Algorithms*

Greedy Algorithms

A greedy algorithm makes the choice that looks best at the moment, i.e., it makes a locally optimal choice in hopes of achieving a globally optimal solution.

It is a powerful and widely applicable method, but it may not always lead to the optimal solution.

Examples:

- Activity-selection problem
- Fractional knapsack problem
- Huffman coding
- Minimum-spanning-tree algorithms
- Single-source shortest path

When is a Greedy Algorithm Applicable?

A greedy algorithm obtains an optimal solution by making a sequence of choices that seem best at the moment they are chosen. This heuristic strategy; however, does not always produce an optimal solution for all problems.

There are two properties that are exhibited by most of the problems that lend themselves to a greedy strategy:

- **Greedy-choice property:** It is possible to make locally-optimal choices in order to arrive at a globally optimal solution. Note that greedy choices are made without relying on having solutions to subproblems first. This is an important difference from the dynamic programming method
- **Optimal substructure:** The optimal solution contains subproblems which are optimal solutions to subproblems. This property is important for the dynamic programming method as well.

Greedy Strategy versus Dynamic Programming

To illustrate how to choose between dynamic programming and a greedy approach, we will consider two variants of a classical optimization problem, **the knapsack problem**. The basic idea is to maximize the value of goods in a knapsack, which can only hold W pounds. Note that each item to consider for packing has both a weight and a value:

item 1 has weight w_1 and value v_1 .

item 2 has weight w_2 and value v_2 .

item 3 has weight w_3 and value v_3 .

⋮

item n has weight w_n and value v_n .

0-1 knapsack problem: must take or leave the entire item.

Fractional knapsack problem: can take fractional items.

Greedy Strategy versus Dynamic Programming *continued*

Both knapsack problems exhibit the optimal-substructure property:

- **0-1 knapsack problem:** consider the most valuable load weighing W pounds. If we remove j , the remaining load must be the most valuable load weighing $W - w_j$ using the $n - 1$ items excluding j .
- **fractional knapsack problem:** consider removing a weight w of one item j from an optimal load weighing W pounds. The remaining load must be the most valuable load weighing $W - w$ given the $n - 1$ items and $w_j - w$ pounds of j .

Even though the fractional knapsack problem can be solved with a greedy approach, the 0-1 knapsack problem cannot be.

Greedy Strategy for the Fractional Knapsack Problem

- First, for each item i , compute the value per pound $\frac{v_i}{w_i}$ and sort the items by this value in $O(n \lg n)$ time.
- Obeying a greedy strategy, take as much of the item with the **greatest value per pound**.
- If there is still capacity, continue with the next most valuable item, taking as much as possible.
- Continue with the next most valuable item until the knapsack is full.
- This greedy algorithm runs in $O(n \lg n)$ time.

Greedy Strategy for the 0-1 Knapsack Problem

Consider the packings for a 0-1 knapsack problem with the following items and a knapsack with capacity, $W = 50$ pounds.

item 1: $w_1 = 10, v_1 = \$70, \frac{v_1}{w_1} = \7

item 2: $w_2 = 20, v_2 = \$120, \frac{v_2}{w_2} = \6

item 3: $w_3 = 30, v_3 = \$135, \frac{v_3}{w_3} = \4.5

1. **The greedy packing:** pick item 1 and then item 2, leaving a capacity too small for item 3. The weight is 30 pounds, the value is \$190.
2. **A better non-greedy packing:** pick items 1 and 3, leaving a capacity too small for item 2. The weight is 40 pounds, the value is \$205.
3. **An optimal non-greedy packing:** pick items 2 and 3, leaving a capacity too small for item 1. The weight is 50 pounds, the value is \$255.

This problem can be solved using dynamic programming because of the overlapping subproblems and optimal-substructure property.