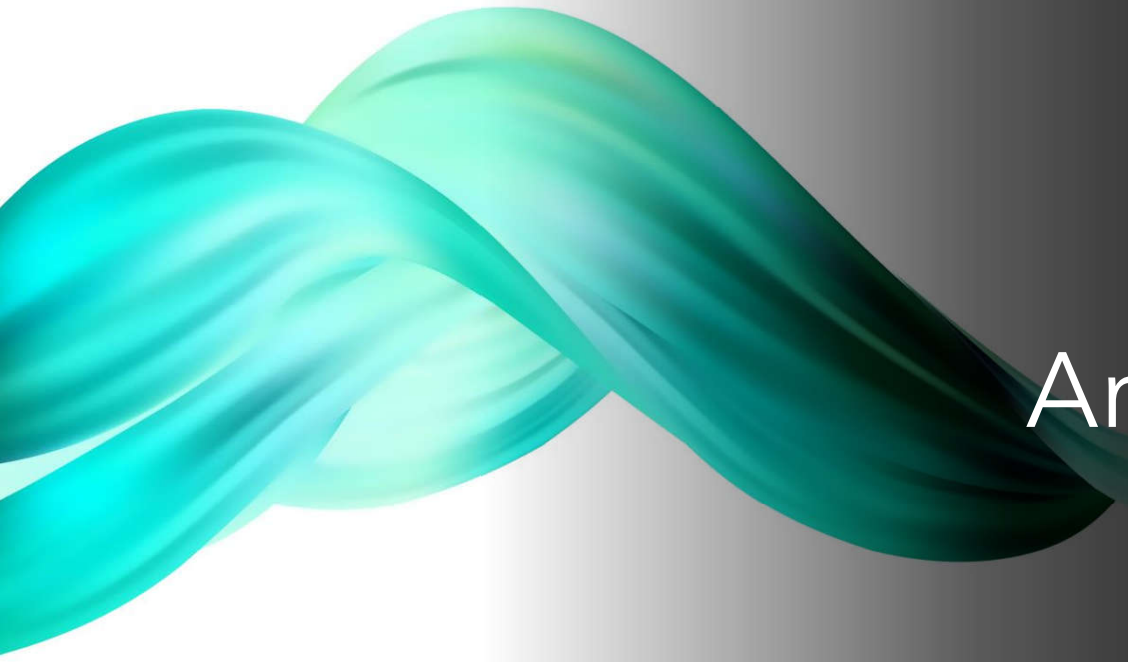


Artificial Intelligence

Dr. Mubashir Ahmad (Ph.D.)

Greedy algorithms

- A greedy algorithm, as the name suggests, **always makes the choice that seems to be the best at that moment**.
- This means that it makes a locally-optimal choice in the hope that this choice will lead to a globally-optimal solution.
- For most optimization problems you want to find, not just a solution, but the best solution. A greedy algorithm sometimes works well for optimization problems. It works in phases.
- At each phase:
- You take the best you can get right now, without regard for future consequences.
- You hope that by choosing a local optimum at each step, you will end up at a global optimum.

Greedy Algorithm

- A greedy algorithm always makes the choice that looks best at the moment.
- A greedy algorithm works on the following properties.

1. Greedy choice property: It makes locally optimal choice in the hope that this choice lead to globally optimal solution.
2. Optimal structures: Optimal solutions contains suboptimal solutions.

History of Greedy Algorithms

- Greedy algorithms were conceptualized for many graph walk algorithms in the 1950s.
- Esdger Djikstra conceptualized the algorithm to generate minimal spanning trees. He aimed to shorten the span of routes within the Dutch capital, Amsterdam.
- In the same decade, Prim and Kruskal achieved optimization strategies that were based on minimizing path costs along weighed routes.

Optimization Problems

- Shortest path is an example of an optimization problem: we wish to find the path with lowest weight.
- What is the general character of an optimization problem?

Optimization Problems

- Ingredients:
  - Instances: The possible inputs to the problem.
  - Solutions for Instance: Each instance has an exponentially large set of valid solutions.
  - Cost of Solution: Each solution has an easy-to-compute cost or value.
- Specification
  - Preconditions: The input is one instance.
  - Post conditions: A valid solution with optimal cost. (minimum or maximum)

Greedy Solutions to Optimization Problems

- Every two-year-old knows the greedy algorithm.
- In order to get what you want, just start grabbing what looks best.
- Surprisingly, many important and practical optimization problems can be solved this way.

Example 1: Making Change

- Problem: Find the minimum # of quarters, dimes, nickels, and pennies that total to a given amount.

The Greedy Choice

- Commit to the object that looks the "best";
- Must prove that this locally greedy choice does not have negative global consequences.

Making Change Example

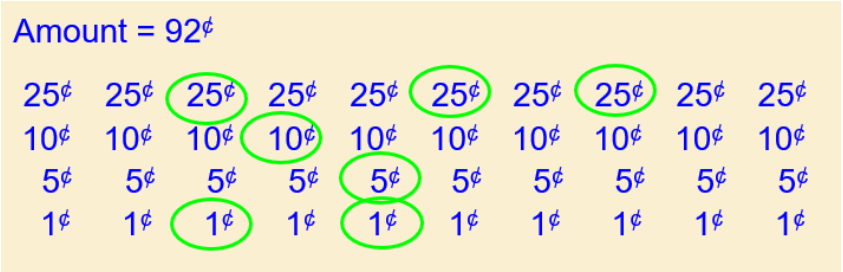
- Instance: A drawer full of coins and an amount of change to return
  - Solutions
    at total
    the amou

Making Change Example

- Instance: A drawer full of coins and an amount of change to return
  - Solutions for Instance: A subset of the coins that total the amount.
- Cost of Solution: The number of coins in the solution = 14
  - Goal: Find an optimal valid solution.

Making Change Example

- Instance: A drawer full of coins and an amount of change to return
- Greedy Choice: Start by grabbing quarters until exceeds amount,
  then dimes, then nickels, then pennies.
- Does this lead to an optimal # of coins?
- Cost of Solution: 7



Hard Making Change Example

- Problem: Find the minimum # of 4, 3, and 1 cent coins to make up 6 cents.
- Greedy Choice: Start by grabbing a 4-cent coin.
- Consequences:

  4+1+1 = 6 mistake

  3+3=6 better
- Greedy Algorithm does not work!

When Does It Work?

- Greedy Algorithms: Easy to understand and to code, but do they work?
- For most optimization problems, all greedy algorithms tried do **not** work (i.e. yield sub-optimal solutions)
- But **some** problems **can** be solved optimally by a greedy algorithm.

Knapsack problem

- The **knapsack problem** or **rucksack problem** is a problem in combinatorial optimization: Given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible.
- It derives its name from the problem faced by someone who is constrained by a fixed-size knapsack and must fill it with the most valuable items.
- The problem often arises in resource allocation where there are financial constraints and is studied in fields such as combinatorics, computer science, complexity theory, cryptography and applied mathematics.

Types of Knapsack problem

- **Fractional Knapsack problem.**
- In Fractional Knapsack, we can break items for maximizing the total value of knapsack. This problem in which we can break an item is also called the fractional knapsack problem.
- **0-1 knapsack problem.**
- In the 0-1 Knapsack problem, we are not allowed to break items. We either take the whole item or don’t take it.

Knapsack problem

- A **brute-force solution** would be to try all possible subset with all different fraction but that will be too much time taking.
- An **efficient solution** is to use Greedy approach. The basic idea of the greedy approach is to calculate the ratio value/weight for each item and sort the item on basis of this ratio. Then take the item with the highest ratio and add them until we can't add the next item as a whole and at the end add the next item as much as we can. Which will always be the optimal solution to this problem.

Fractional Knapsack problem

Items N = 7
Bag capacity M = 15

Objects		1	2	3	4	5	6	7
Profits	p	10	5	15	7	6	18	3
Weights	w	2	3	5	7	1	4	1

Fractional Knapsack problem

- Now we have a bag with capacity of 15. lets suppose 15 kg.
- Now we want to transfer this bag from one location to another.
- We are facing this problem in daily life.
- Now problem is to filling of the objects in a container.
- If you put all the objects in the bag with the capacity, there is no problem.
- But the objects weight is more than the bag capacity, here the problem start.

Fractional Knapsack problem

- We should maximize the profit.
- So this problem is optimization problem.
- Can we apply the greedy method to solve this problem?
- Yes, if it is suitable.
- So what are the constraints here?
- The bag capacity is 15, so all the objects which we want to transfer should be <= 15.

Fractional Knapsack problem

- For this problem we have many solutions.
- The optimal result is to gain a maximum profit.
- X = {x1, x2, x3, x4, x5, x6, x7}
- We can say this problem is a fractional problem
- $0 \leq X \leq 1$

Fractional Knapsack problem

- Items n = 7
- Bag capacity m = 15

Objects	0	1	2	3	4	5	6	7
Profits	p	10	5	15	7	6	18	3
Weights	w	2	3	5	7	1	4	1
Max profit	p/w	5	1.3	3	1	6	4.5	3

Fractional Knapsack problem

Items n = 7
Bag capacity m = 15

- Profit of each item per kg is
- X1 = 5, x2=1.3, x3=3, x4=1, x5=6, x6=4.5, x7=3

Objects	0	1	2	3	4	5	6	7
Profits	p	10	5	15	7	6	18	3
Weights	w	2	3	5	7	1	4	1
Max profit	p/w	5	1.3	3	1	6	4.5	3

Fractional Knapsack problem

Items n = 7
Bag capacity m = 15

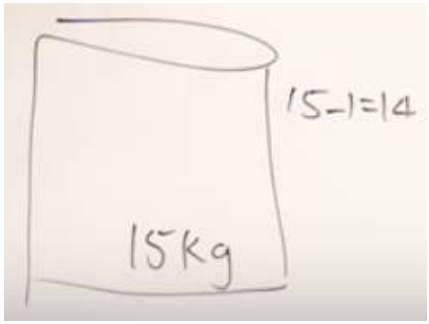
- Now we select the items on the basis of their profit

Objects	0	1	2	3	4	5	6	7
Profits	p	10	5	15	7	6	18	3
Weights	w	2	3	5	7	1	4	1
Max profit	p/w	5	1.3	3	1	6	4.5	3

Fractional Knapsack problem

- Now we select the items on the basis of their profit

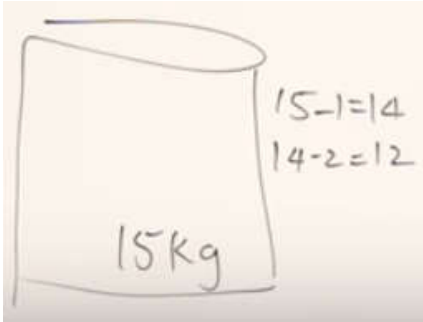
Objects	0	1	2	3	4	5	6	7
Profits	p	10	5	15	7	6	18	3
Weights	w	2	3	5	7	1	4	1
Max profit	p/w	5	1.3	3	1	6	4.5	3



Fractional Knapsack problem

- Now we select the items on the basis of their profit

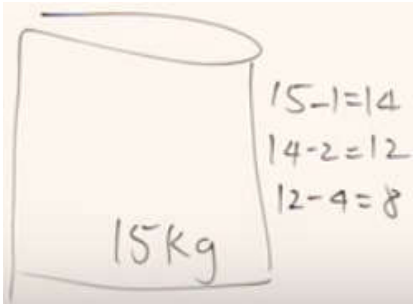
Objects	0	1	2	3	4	5	6	7
Profits	p	10	5	15	7	6	18	3
Weights	w	2	3	5	7	1	4	1
Max profit	p/w	5	1.3	3	1	6	4.5	3



Fractional Knapsack problem

- Now we select the items on the basis of their profit

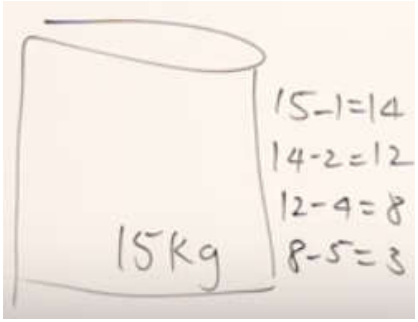
Objects	0	1	2	3	4	5	6	7
Profits	p	10	5	15	7	6	18	3
Weights	w	2	3	5	7	1	4	1
Max profit	p/w	5	1.3	3	1	6	4.5	3



Fractional Knapsack problem

- Now we select the items on the basis of their profit

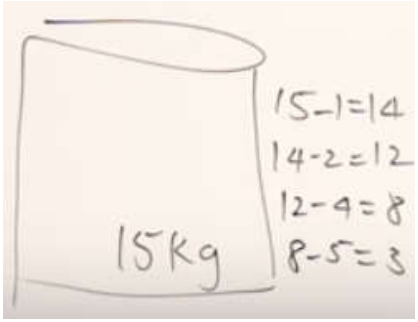
Objects	0	1	2	3	4	5	6	7
Profits	p	10	5	15	7	6	18	3
Weights	w	2	3	5	7	1	4	1
Max profit	p/w	5	1.3	3	1	6	4.5	3



Fractional Knapsack problem

- Now we select the items on the basis of their profit

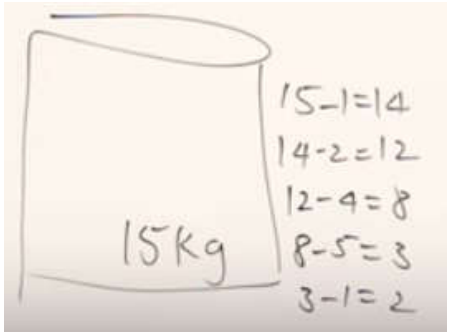
Objects	0	1	2	3	4	5	6	7
Profits	p	10	5	15	7	6	18	3
Weights	w	2	3	5	7	1	4	1
Max profit	p/w	5	1.3	3	1	6	4.5	3



Fractional Knapsack problem

- Now we select the items on the basis of their profit

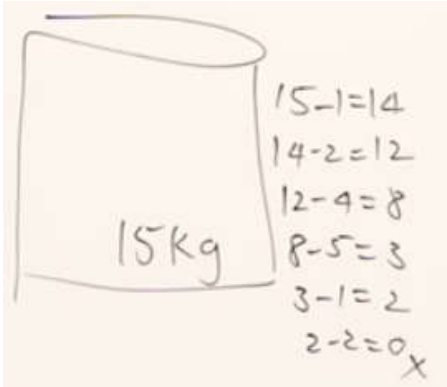
Objects	0	1	2	3	4	5	6	7
Profits	p	10	5	15	7	6	18	3
Weights	w	2	3	5	7	1	4	1
Max profit	p/w	5	1.3	3	1	6	4.5	3



Fractional Knapsack problem

- Now we select the items on the basis of their profit

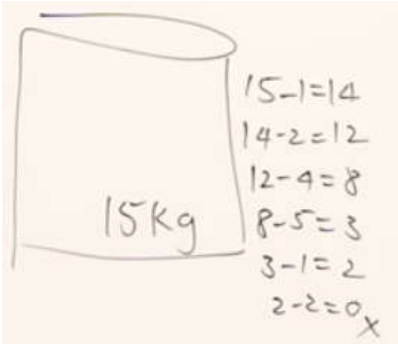
Objects	0	1	2	3	4	5	6	7
Profits	p	10	5	15	7	6	18	3
Weights	w	2	3	5	7	1	4	1
Max profit	p/w	5	1.3	3	1	6	4.5	3



Fractional Knapsack problem

- Now we select the items on the basis of their profit

Objects	0	1	2	3	4	5	6	7
Profits	p	10	5	15	7	6	18	3
Weights	w	2	3	5	7	1	4	1
Max profit	p/w	5	1.3	3	1	6	4.5	3

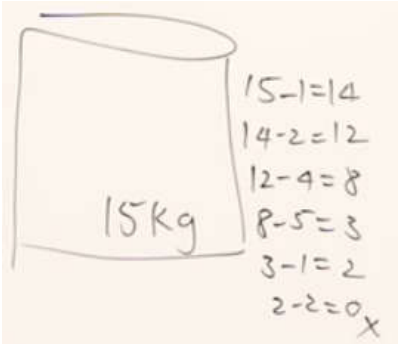


$1\times2+\frac{2}{3}\times3+1\times5+0\times7+1\times1+1\times4+1\times1$

Fractional Knapsack problem

- Now we select the items on the basis of their profit

Objects	0	1	2	3	4	5	6	7
Profits	p	10	5	15	7	6	18	3
Weights	w	2	3	5	7	1	4	1
Max profit	p/w	5	1.3	3	1	6	4.5	3



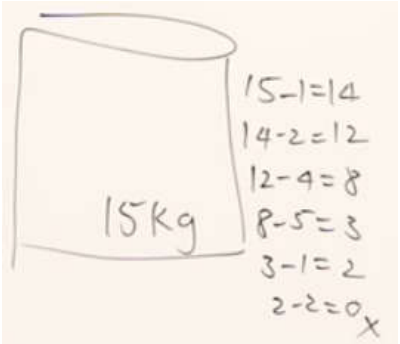
$1\times2+\frac{2}{3}\times3+1\times5+0\times7+1\times1+1\times4+1\times1$

$2+2+5+0+1+4+1=15$

Fractional Knapsack problem

- Now we select the items on the basis of their profit

Objects	0	1	2	3	4	5	6	7
Profits	p	10	5	15	7	6	18	3
Weights	w	2	3	5	7	1	4	1
Max profit	p/w	5	1.3	3	1	6	4.5	3



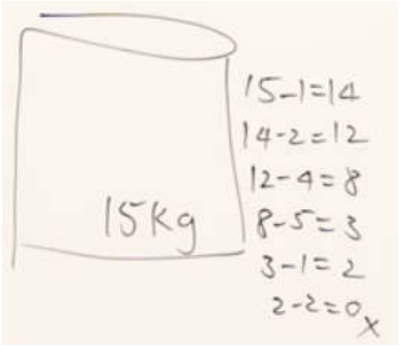
$$\sum x_i w_i = 1\times2 + \tfrac{2}{3}\times3 + 1\times5 + 0\times7 + 1\times1 + 1\times4 + 1\times1$$

$$2 + 2 + 5 + 0 + 1 + 4 + 1 = 15$$

Fractional Knapsack problem

- Now we select the items on the basis of their profit

Objects	0	1	2	3	4	5	6	7
Profits	p	10	5	15	7	6	18	3
Weights	w	2	3	5	7	1	4	1
Max profit	p/w	5	1.3	3	1	6	4.5	3



$$\sum x_i w_i = 1\times2 + \tfrac{2}{3}\times3 + 1\times5 + 0\times7 + 1\times1 + 1\times4 + 1\times1$$

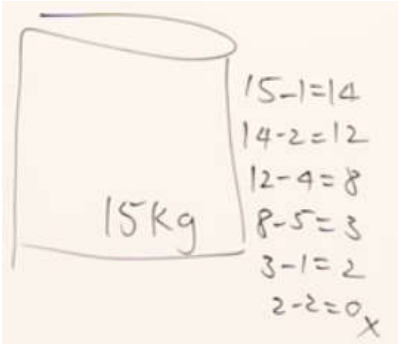
$$2 + 2 + 5 + 0 + 1 + 4 + 1 = 15$$

$$\sum x_i p_i = 1\times10 + \tfrac{2}{3}\times5 + 1\times15 + 1\times6 + 1\times18 + 1\times3$$

Fractional Knapsack problem

- Now we select the items on the basis of their profit

Objects	0	1	2	3	4	5	6	7
Profits	p	10	5	15	7	6	18	3
Weights	w	2	3	5	7	1	4	1
Max profit	p/w	5	1.3	3	1	6	4.5	3



$\sum x_i w_i = 1\times2 + \frac{2}{3}\times3 + 1\times5 + 0\times7 + 1\times1 + 1\times4 + 1\times1$

$2+2+5+0+1+4+1 = 15$

$\sum x_i p_i = 1\times10 + \frac{2}{3}\times5 + 1\times15 + 1\times6 + 1\times18 + 1\times3$

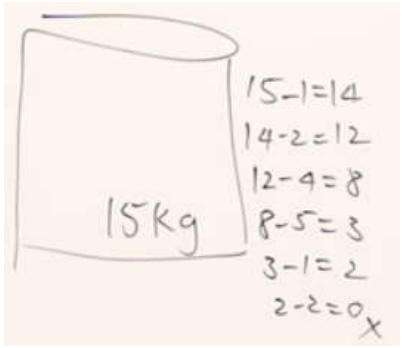
$= 10 + 2\times1.3 + 15 + 6 + 18 + 3 = 54.3$

Fractional Knapsack problem

Items n = 7
Bag capacity m = 15

- Now we select the items on the basis of their profit

Objects	0	1	2	3	4	5	6	7
Profits	p	10	5	15	7	6	18	3
Weights	w	2	3	5	7	1	4	1
Max profit	p/w	5	1.3	3	1	6	4.5	3



$\sum x_i w_i = 1\times2 + \frac{2}{3}\times3 + 1\times5 + 0\times7 + 1\times1 + 1\times4 + 1\times1$

$2+2+5+0+1+4+1 = 15$

$\sum x_i p_i = 1\times10 + \frac{2}{3}\times5 + 1\times15 + 1\times6 + 1\times18 + 1\times3$

$= 10 + 2\times1.3 + 15 + 6 + 18 + 3 = 54.3$

Constraint

$\sum x_i w_i \leq m$

Objective

$\max \sum x_i p_i$

Fractional Knapsack problem

Algorithm

- Sort the given array of items according to **weight / value(W /V)** ratio in descending order.
- Start adding the item with the maximum **W / V** ratio.
- Add the whole item, if the current weight is less than the capacity, else, add a portion of the item to the knapsack.
- Stop, when all the items have been considered and the total weight becomes equal to the weight of the given knapsack.

Difference between Brute force and Greedy algorithms

- Bruce-force Algorithm
- O(2^n)
- Greedy Approach
- **Time Complexity:** Time complexity of the sorting + Time complexity of the loop to maximize profit **=** O(logN) + O(N) = O(NlogN)