Divide and Conquer
Greedy Method-knapsack problem

Dr. K.RAGHAVA RAO, Professor in CSE, KL University, krraocse@gmail.com,
http://mcadaa.blog.com

Greedy Technique Definition

Constructs a solution to an *optimization problem* piece by piece through a sequence of choices that are: *feasible, i.e. satisfying the constraints locally optimal (with respect to some neighborhood definition) greedy (in terms of some measure), and irrevocable.*

For some problems, it yields a globally optimal solution for every instance. For most, does not but can be useful for fast approximations. We are mostly interested in the former case in this class.

Generic Algorithm

```
Algorithm Greedy(a,n) {
//a[1..n] contains the n inputs.
solution:= ∅;
For i:= 1to n do {
X=select(a);
If Feasible(solution , x) then
solution:= union(solution, x);
}
return solution;
}
```

Applications of the Greedy Strategy

Optimal solutions:

- change making for “normal” coin denominations
- minimum spanning tree (MST)
- single-source shortest paths
- simple scheduling problems
- Huffman codes

Approximations/heuristics:

- traveling salesman problem (TSP)
- knapsack problem
- other combinatorial optimization problems

Change-Making Problem

Given unlimited amounts of coins of denominations $d_1 > \ldots > d_m$,
give change for amount *n* with the least number of coins

Q: What are the objective function and constraints?

Example: $d_1 = 25c$, $d_2 = 10c$, $d_3 = 5c$, $d_4 = 1c$ and $n = 48c$

Greedy solution: <1, 2, 0, 3>

Greedy solution is

- optimal for any amount and “normal” set of denominations

Ex: Prove the greedy algorithm is optimal for the above denominations.

- may not be optimal for arbitrary coin denominations

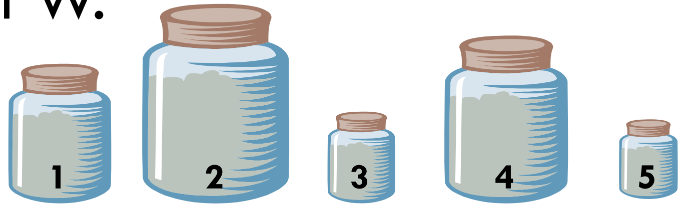
The Fractional Knapsack Problem

- Given: A set S of *n* items, with each item i having
  - b_i - a positive benefit
  - w_i - a positive weight
- Goal: Choose items with maximum total benefit but with weight at most W.
- If we are allowed to take fractional amounts, then this is the **fractional knapsack problem.**
  - In this case, we let x_i denote the amount we take of item i
  - Objective: maximize $$\sum_{i \in S} b_i (x_i / w_i)$$
  - Constraint: $$\sum_{i \in S} x_i \le W$$

Example model-1

in this model items are arranged by their values, maximum selected first, process continous till minimum value

- Given: A set S of n items, with each item i having
  - b_i - a positive benefit
  - w_i - a positive weight
- Goal: Choose items with maximum total benefit but with weight at most W.



Items:	1	2	3	4	5
Weight:	4 ml	8 ml	2 ml	6 ml	1 ml
Benefit:	Rs.12	Rs.32	Rs.40	Rs.30	Rs.50
Value: (Rs. per ml)	3	4	20	5	50

"knapsack"

10 ml

Solution:
- 1 ml of 5
- 2 ml of 3
- 6 ml of 4
- 1 ml of 2

Knapsack Problem model-2

in this model items are arranged by their weights, lightest weight selected first, process continuous till the maximum weight.

- You have a knapsack that has capacity (weight) C.
- You have several items $I_1,\ldots,I_n$.
- Each item I_i has a weight w_i and a benefit b_i.
- You want to place a certain number of copies of each item I_i in the knapsack so that:
  - The knapsack weight capacity is not exceeded and
  - The total benefit is maximal.

Key question

- Suppose f(w) represents the *maximal possible benefit* of a knapsack with weight w.
- We want to find (in the example) f(5).
- Is there anything we can say about f(w) for arbitrary w?

Key observation

- To fill a knapsack with items of weight w, we must have added items into the knapsack in some order.
- Suppose the last such item was I_i with weight w_i and benefit b_i.
- Consider the knapsack with weight ($w - w_i$). Clearly, we chose to add I_i to this knapsack because of all items with weight w_i or less, I_i had the max benefit b_i.

Key observation

- Thus, $f(w) = MAX \{ b_i + f(w-w_i) \mid I_i \text{ is an item}\}$.
- This gives rise to an immediate recursive algorithm to determine how to fill a knapsack.

Example

Item	Weight	Benefit
A	2	60
B	3	75
C	4	90

f(0), f(1)

- f(0) = 0. Why? The knapsack with capacity 0 can have nothing in it.
- f(1) = 0. There is no item with weight 1.

f(2)

- f(2) = 60. There is only one item with weight 60.
- **Choose A.**

f(3)

- $f(3) = MAX\{ b_i + f(w-w_i) \mid I_i \text{ is an item}\}$.

$= MAX\{ 60+f(3-2), 75 + f(3-3)\}$

$= MAX\{ 60 + 0, 75 + 0 \}$

$= 75$.

Choose B.

f(4)

- $f(4) = MAX \{ b_i + f(w-w_i) \mid I_i \text{ is an item}\}$.

$= MAX \{ 60 + f(4-2), 75 + f(4-3), 90+f(4-4)\}$

$= MAX \{ 60 + 60, 75 + f(1), 90 + f(0)\}$

$= MAX \{ 120, 75, 90\}$

$=120$.

Choose A.

f(5)

- $f(5) = MAX \{ b_i + f(w-w_i) \mid I_i \text{ is an item}\}$.

$= MAX \{ 60 + f(5-2), 75 + f(5-3), 90+f(5-4)\}$

$= MAX \{ 60 + f(3), 75 + f(2), 90 + f(1)\}$

$= MAX \{ 60 + 75, 75 + 60, 90+0\}$

$= 135$.

Choose A or B.

Result

- Optimal knapsack weight is 135.
- Two possible optimal solutions:
  - Choose A during computation of f(5). Choose B in computation of f(3).
  - Choose B during computation of f(5). Choose A in computation of f(2).
- Both solutions coincide. Take A and B.

Another example model-2

- Knapsack of capacity 50.
- 3 items
  - Item 1 has weight 10, benefit 60
  - Item 2 has weight 20,benefit 100
  - Item 3 has weight 30, benefit 120.

f(0),..,f(9)

- All have value 0.

f(10),..,f(19)

- All have value 10.
- **Choose Item 1.**

f(20),..,f(29)

- F(20) = MAX { 60 + f(10), 100 + f(0) }

= MAX { 60+60, 100+0}

=120.

Choose Item 1.

f(30),…,f(39)

f(30) = MAX { 60 + f(20), 100 + f(10), 120 + f(0) }

= MAX { 60 + 120, 100+60, 120+0}

= 180

Choose item 1.

f(40),…,f(49)

- F(40) = MAX { 60 + f(30), 100 + f(20), 120 + f(10)}

= MAX { 60 + 180, 100+120, 120 + 60}

= 240.

Choose item 1.

f(50)

- f(50) = MAX { 60 + f(40), 100 + f(30), 120 + f(20) }

= MAX { 60 + 240, 100+180, 120 + 120}

= 300.

Choose item 1.

Fractional knapsack

- Much easier
- For item I_i, let $r_i = b_i/w_i$. This gives you the benefit per measure of weight.
- Sort the items in descending order of r_i
- Pack the knapsack by putting as many of each item as you can walking down the sorted list.

Fractional knapsack model-3 example

A thief enters a store and sees the following items:

Cost Rs. 100-A , Rs. 10-B, Rs. 120-C for Weight: 2 kg, 2kg , 3 kg

His Knapsack holds 4 Kgs. What should he steal to maximize profit?

Thief can take a fraction of an item.

Solution: 2 kg of item A + 2 kg of item C = Rs.100 + Rs 80=180

Fractional knapsack example model-3

n=3, m=20,(p1,p2,p3)=(25,24,15), and (w1,w2,w3)=(18,15,10)

Four feasible solutions are:

(x1,x2,x3)	$\sum Wixi$	$\sum pixi$
1. (1/2,1/3,1/4)	16.5	24.25
2. (1,2/15, 0)	20	28.2
3. (0,2/3, 1)	20	31
4 (0,1,1/2)	20	31.5

Solution 4 yields the maximum profit, so this is optimal solution for the given problem.

Fractional knapsack example model-3

I=<I1,I2,I3,I4,I5> W=<5,10,20,30,40> V=<30,20,100,90,160> knapsack capacity W=60, the solution to the fractional knapsack problem is given as:

Initially

Item	Wi	Vi
I1	5	30
I2	10	20
I3	20	100
I4	30	90
I5	40	160

Fractional knapsack example model-3

I=<I1,I2,I3,I4,I5> W=<5,10,20,30,40> V=<30,20,100,90,160> knapsack capacity W=60, the solution to the fractional knapsack problem is given as:

Taking value per weight ratio

Item	wi	vi	Pi=vi/wi
I1	5	30	6.0
I2	10	20	2.0
I3	20	100	5.0
I4	30	90	3.0
I5	40	160	4.0

Fractional knapsack example model-3

I=<I1,I2,I3,I4,I5> W=<5,10,20,30,40> V=<30,20,100,90,160> knapsack capacity W=60, the solution to the fractional knapsack problem is given as: Arranging item with decreasing order of Pi

Item	wi	vi	Pi=vi/wi
I1	5	30	6.0
I2	20	100	5.0
I3	40	160	4.0
I4	30	90	3.0
I5	10	20	2.0

Filling knapsack according to decreasing value of Pi, max. value = v1+v2+new(v3)=30+100+140=270

exercise

- Find an optimal solution to knapsack problem instance $n=7, m=15$, $\{p1,p2,p3,\ldots p7\}=(10,5,15,7,6,18,3)$, and $(w1,w2,\ldots w7)=(2,3,5,7,1,4,1)$.

- N=3 m=20 , w1,w2,w3=18,15,10
- P1,p2,p3=25,24,15 find optimal solution for knapsack problem.

The Fractional Knapsack Algorithm

Algorithm for greedy strategy for knapsack problem

```
Algorithm GreedyKnapsack(m,n)
//p[1:n] and w[1:n] contain profits and weights respectively of n objects ordered such that
//p[i]/w[i] >=p[i+1]/w[i+1].m is the knapsack size and x[1:n] is the solution vector
{
For i:= 1 to n do x[i]:=0.0; //initialize x
U:=m;
{
If (w[i]>U) then break;
x[i]:=1.0; U:=U-w[i];
}
If (i<=n) then x[i]:=U/w[i];
}
```