CSE 102 11-10-21

Supplemental lecture 3

Greedy Algorithms

Ex. Continuous Knapsack

As before

(1) maximize $\sum_{i=1}^{n} x_i v_i$

(2) subject to $\sum_{i=1}^{n} x_i w_i \leq W$

where $W > 0$, $v_i > 0$, $w_i > 0$ $(1 \leq i \leq n)$

However, instead of $x_i \in \{0, 1\}$ we allow $0 \leq x_i \leq 1$ for $1 \leq i \leq n$.

We call a vector $x = (x_1, \ldots, x_n)$ a feasible solution iff (2) is satisfied.

A Greedy strategy consists of making some 'locally optimal' choice for an object, then solving resulting subproblem.

This means 'steal the best object', then the 'next best', etc.

see pseudo-code

Knapsack (v, w, W)

```
1.) n ← length[v]
2.) for i ← 1 to n
3.)     x[i] ← 0
4.) weight ← 0
5.) while weight < W                                   ┐
*6.)    i ← THE "BEST" REMAINING OBJECT                │
7.)    if weight + w[i] ≤ W                            │
8.)        x[i] ← 1                                    │ GREEDY
9.)        weight ← weight + w[i]                      │ LOOP
10.)       mark i AS INCLUDED.                         │
11.)   else                                            │
12.)       x[i] ← (W − weight) / w[i]                   │
13.)       weight ← W                                  ┘
14.) return x
```

THE "BEST" OBJECT IN (6) CAN BE INTERPRETED IN SEVERAL WAYS.

IN GENERAL WE DEFINE A <u>SELECTION FUNCTION</u> $f(i)$ WHICH ENCODES THE DESIRABILITY OF OBJECT i. LINE (6) THEN MAXIMIZES f OVER ALL REMAINING (UNMARKED) OBJECTS.

To define the 'best' object, we choose a selection function $f(i)$, and maximize it.

Possible choices for $f(i)$:

- $f(i) = v_i$: Process objects in order of decreasing v_i.
- $f(i) = \frac{1}{w_i}$: Process obj in order of increasing weight.
- $f(i) = \frac{v_i}{w_i}$: Process objects in order of decreasing value to weight ratio.

Ex. $n = 5$, $W = 10$

i	1	2	3	4	5
v_i	2	3	6.6	4	5
w_i	1	2	3	4	5
$\frac{v_i}{w_i}$	2	1.5	2.2	1	1.2

$f(i)$	x					Val
v_i	0	0	1	.5	1	14.6
$\frac{1}{w_i}$	1	1	1	1	0	15.6
$\frac{v_i}{w_i}$	1	1	1	0	.8	(16.4)

Theorem

If we maximize $\frac{v_i}{w_i}$ on line (6), then Knapsack( ) returns an optimal solution.

Proof.

Without loss of generality, Assume the objects are already sorted

$$\frac{v_1}{w_1} \geq \frac{v_2}{w_2} \geq \frac{v_3}{w_3} \geq \cdots \geq \frac{v_n}{w_n}$$

Let $x = (x_1, x_2, \ldots, x_n)$ be the solution returned by Knapsack( ).

If all $x_i = 1$, then x is clearly optimal.

Assume not all $x_i = 1$.

Let j be the first index for which $x_j < 1$. Inspecting the algorithm, we see that

$$x_i = 1 \quad \text{for } 1 \le i < j$$

$$x_j < 1$$

$$x_i = 0 \quad \text{for } j < i \le n$$

Also

$$\sum_{i=1}^{n} x_i w_i = W .$$

Let $y = (y_1, y_2, \ldots, y_n)$ be any feasible solution, i.e.

$$\sum_{i=1}^{n} y_i w \le W .$$

Define

$$V(y) = \sum_{i=1}^{n} y_i v_i$$

and $V(x) = \sum_{i=0}^{n} x_i v_i$

we must show that $V(x) \geq V(y)$.

note:

$$\sum_{i=1}^{n} (x_i - y_i) w_i = \sum_{i=1}^{n} x_i w_i - \sum_{i=1}^{n} y_i w_i$$

$$= W - \sum_{i=1}^{n} y_i w_i \geq 0$$

so

$$V(x) - V(y) = \sum_{i=1}^{n} (x_i - y_i) v_i$$

$$= \sum_{i=1}^{n} (x_i - y_i) w_i \cdot \left(\frac{v_i}{w_i}\right)$$

Observe that

- $i < j \Rightarrow x_i = 1 \Rightarrow x_i - y_i \geq 0 \overset{!}{\Leftrightarrow} \frac{v_i}{w_i} \geq \frac{v_j}{w_j}$

$$\therefore (x_i - y_i)\frac{v_i}{w_i} \geq (x_i - y_i)\frac{v_j}{w_j}$$

- $i > j \Rightarrow x_i = 0 \Rightarrow x_i - y_i \leq 0 \overset{!}{\Leftrightarrow} \frac{v_i}{w_i} \leq \frac{v_j}{w_j}$

$$\therefore (x_i - y_i)\frac{v_i}{w_i} \geq (x_i - y_i)\frac{v_j}{w_j}$$

- $i = j \Rightarrow (x_i - y_i)\frac{v_i}{w_i} = (x_i - y_i)\frac{v_j}{w_j}$

In all cases therefore:

$$(x_i - y_i)\frac{v_i}{w_i} \geq (x_i - y_i)\frac{v_j}{w_j}$$

Hence

$$V(x) - V(y) = \sum_{i=1}^{n} (x_i - y_i) w_i \cdot \frac{v_i}{w_i}$$

$$\geq \sum_{i=1}^{n} (x_i - y_i) w_i \cdot \left(\frac{v_j}{w_j}\right)$$

$$= \left(\frac{v_j}{w_j}\right) \cdot \sum_{i=1}^{n} (x_i - y_i) w_i$$

$$\geq 0$$

$\therefore$ $V(x) \geq V(y)$, as required. ∎

Examining algorithm, we see its runtime is $\Theta(n \log n)$.

The Dynamic Programming Soln to 0-1 Knapsack cost $\Theta(n \cdot W)$. if $W = \Theta(n)$, then this is $\Theta(n^2)$

Idea: modify Knapsack() to work on the discrete version. If you can't steal all of next best object, skip it and go to the next one.

Ex. Same weights & values, But 0-1 Problem.

$n = 5$, $W = 10$

i	1	2	3	4	5
v_i	2	3	6.6	4	6
w_i	1	2	3	4	5
$\frac{v_i}{w_i}$	2	1.5	2.2	1	1.2

modified greedy solution

$x = (1, 1, 1, 1, 0)$ { value = 15.6, weight = 10

Exercise Check that dynamic programming gives same solution.

Ex. $n=5$, $W=11$

i	1	2	3	4	5
v_i	1	6	18	22	28
w_i	1	2	5	6	7
$\frac{v_i}{w_i}$	1	3	3.6	3.67	4

modified greedy solution:

$x = (1, 1, 0, 0, 1)$ { Value = 35, Weight = 10

Exercise

check that dynamic Programming yields

$x = (0, 0, 1, 1, 0)$ { Value = 40, Weight = 11

Coin Changing Problem

As before have denominations

$$d = (d_1, d_2, \ldots, d_n)$$

and an amount N to be paid.
Goal: use fewest # of coins.

Greedy strategy:

- from amongst all denominations whose addition to the sum to exceed N, choose the largest.
- stop when sum is N.

Assume $d_1 = 1$ so we can pay any amount N.

Exercise (hard):

show that for $d = (1, 5, 10, 25, 100)$, the greedy strategy yields an optimal solution for any $N \geq 0$.

Exercise (easy):

show that for $d = (1, 10, 25, 100)$ the greedy strategy does not always yield an opt. soln.

(Hint: $N = 30$).

Exercise.

let $b \in \mathbb{Z}^+$, $b \geq 2$, $d_i := b^{i-1}$ $(1 \leq i \leq n)$, so that $d = (1, b, b^2, \ldots, b^{n-1})$.

show that the greedy strategy does give an opt soln for all $N \geq 0$.

Exercise (very hard)

Characterize all denomination sets $d = \{d_1, d_2, \ldots, d_n\}$ for which the greedy strategy always yields an optimal solution, for all $N \geq 0$.

In general: the ingredients to look for in an opt. Problem to determine if a greedy strategy works are:

- optimal substructure:
  Optimal solutions contain optimal sub-instance solutions
- greedy choice Property:
  a globally optimal soln. can be constructed by making a seq. of locally optimal choices.