CS 381

Dynamic Programming

6.4 Knapsack Problem

Knapsack Problem

Knapsack problem.

- Given n objects and a "knapsack."
- Item i weighs $w_i > 0$ pounds and has value $v_i > 0$.
- Knapsack has capacity of W pounds.
- Goal: fill knapsack so as to maximize total value.

Example:

#	value	weight
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

W = 11

Knapsack Problem

Knapsack problem.

- Given n objects and a "knapsack."
- Item i weighs $w_i > 0$ pounds and has value $v_i > 0$.
- Knapsack has capacity of W pounds.
- Goal: fill knapsack so as to maximize total value.

Example:

#	value	weight
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

W = 11

Greedy: repeatedly add item with maximum ratio v_i / w_i provided the capacity constraint is met.

Q: Is this solution optimal?

Knapsack Problem

Knapsack problem.

- Given n objects and a "knapsack."
- Item i weighs $w_i > 0$ pounds and has value $v_i > 0$.
- Knapsack has capacity of W pounds.
- Goal: fill knapsack so as to maximize total value.

Example:

#	value	weight
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

W = 11

Greedy: repeatedly add item with maximum ratio v_i / w_i provided the capacity constraint is met.

Ex: { 5, 2, 1 } achieves only value = 35 $\Rightarrow$ greedy not optimal.

Knapsack Problem

Knapsack problem.

- Given n objects and a "knapsack."
- Item i weighs $w_i > 0$ pounds and has value $v_i > 0$.
- Knapsack has capacity of W pounds.
- Goal: fill knapsack so as to maximize total value.

Example:

#	value	weight
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

W = 11

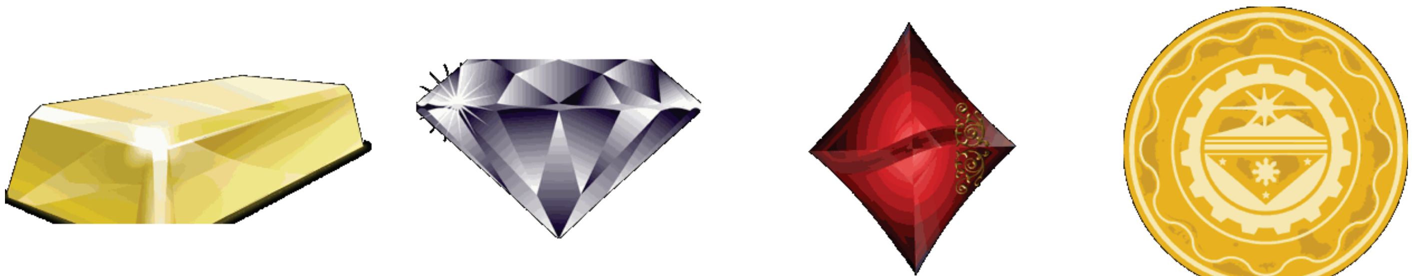
Greedy: repeatedly add item with maximum ratio v_i / w_i provided the capacity constraint is met.

Ex: { 5, 2, 1 } achieves only value = 35 $\Rightarrow$ greedy not optimal.

Another combination: { 3, 4 } has value 40.

Dynamic Programming

How do we define the subproblems and use them to solve the whole instance?



Wt. = 5
Value = 10

Wt. = 3
Value = 20

Wt. = 8
Value = 25

Wt. = 4
Value = 8

Maximum wt. = 13

Dynamic Programming: Adding a New Variable

Def. OPT(i, w) = max profit subset of items 1, …, i with weight limit w.

- Case 1: OPT does not select item i.
  - OPT selects best of { 1, 2, …, i-1 } using weight limit w

- Case 2: OPT selects item i.
  - new weight limit = w – w_i
  - OPT selects best of { 1, 2, …, i–1 } using this new weight limit

$$OPT(i,w) = \begin{cases} 0 & \text{if } i = 0 \\ OPT(i-1, w) & \text{if } w_i > w \\ \max\{ OPT(i-1, w),\quad v_i + OPT(i-1, w-w_i) \} & \text{otherwise} \end{cases}$$

Knapsack Problem: Bottom-Up

Knapsack. Fill up an n-by-W array.

```
Input: n, W, w1,…,wN, v1,…,vN

for w = 0 to W
   M[0, w] = 0

for i = 1 to n
   for w = 1 to W
      if (wi > w)
         M[i, w] = M[i-1, w]
      else
         M[i, w] = max {M[i-1, w], vi + M[i-1, w-wi ]}

return M[n, W]
```

Knapsack Algorithm

W + 1 →

n + 1 ↓

	0	1	2	3	4	5	6	7	8	9	10	11
ϕ	0	0	0	0	0	0	0	0	0	0	0	0
{ 1 }	0	1	1	1	1	1	1	1	1	1	1	1
{ 1, 2 }	0	1	6	7	7	7	7	7	7	7	7	7
{ 1, 2, 3 }	0	1	6	7	7	18	19	24	25	25	25	25
{ 1, 2, 3, 4 }	0	1	6	7	7	18	22	24	28	29	29	40
{ 1, 2, 3, 4, 5 }	0	1	6	7	7	18	22	28	29	34	34	40

Item	Value	Weight
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

W = 11

OPT: { 4, 3 }
value = 22 + 18 = 40

Knapsack Problem: Running Time

Running time. $\Theta(n\ W)$.

- Not polynomial in input size!
  - Only need $\log_2 W$ bits to encode each weight
  - Problem can be encoded with $O(n \log_2 W)$ bits
- "Pseudo-polynomial."
- Decision version of Knapsack is NP-complete. [Chapter 8]

Knapsack approximation algorithm. There exists a poly-time algorithm that produces a feasible solution that has value within 0.01% of optimum. [Section 11.8]