Knapsack Problem

Choice Point

Knapsack problem.

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

Ex: { 3, 4 } has value 40.

{3.4} 18+22 =40

W = 11

2+5: 6+28 =34

Item	Value	Weight	v/w
1	1	1	1
2	6	2	3
3	18	5	3.6
4	22	6	3.63...
5	28	7	4

28+6+1 = 35

Greedy: repeatedly add item with maximum ratio v_i / w_i.

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

Dynamic Programming: False Start

Def. OPT(i) = max profit subset of items 1, …, i.

$\{1, 2, \ldots, \cancel{i}\}$ ~~i+1, i+2~~

Leave i
- Case 1: OPT does not select item i.
  - OPT selects best of { 1, 2, …, i-1 }

    OPT(i) = OPT(i-1)

i W W-w_i

2 Parameters

Weight

items.

Take i
- Case 2: OPT selects item i. v_i + OPT(i-1)
  - accepting item i does not immediately imply that we will have to reject other items
  - without knowing what other items were selected before i, we don't even know if we have enough room for i

Conclusion. Need more sub-problems!

Dynamic Programming: Adding a New Variable

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

p_1 p_2

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

max

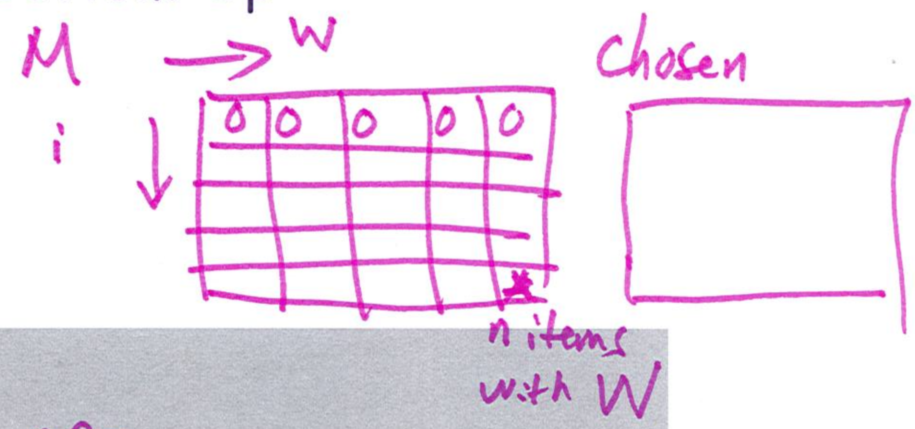
- Case 2: OPT selects item i.
  - new weight limit = w – w_i $OPT(i, w) = v_i + OPT(i-1, w-w_i)$
  - OPT selects best of { 1, 2, …, i-1 } using this new weight limit

$$OPT(i,w) = \begin{cases} 0 & \text{Base Case} \quad \text{if } i = 0 \\ OPT(i-1, w) & w_i > w \text{ leave } i \quad \text{if } w_i > w \\ \max\{\underbrace{OPT(i-1, w)}_{\text{NOT select } i},\ \underbrace{v_i + OPT(i-1, w-w_i)}_{\text{Select } i}\} & \text{otherwise} \end{cases}$$

i overweight the limit

choose the better choice.

Knapsack Problem: Bottom-Up



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

```
Input: n, w1,…,wN, v1,…,vN
                                        M: OPT
for w = 0 to W
   M[0, w] = 0 ← Base case when i=0. M[0,…] = 0

for i = 1 to n
   for w = 1 to W ← Limit
      if (wi > w) overweight
         M[i, w] = M[i-1, w]  OPT(i,w) = OPT(i-1,w)
         Chosen[i, w] = Chosen[i-1, w]  Record selected items.
      else                          NOT take i        take i
         M[i, w] = max {M[i-1, w], vi + M[i-1, w-wi]}
          If (M[i-1, w] is greater) NOT select i
          Then Chosen[i, w] = Chosen[i-1, w]
          Else Chosen[i, w] = i∪ Chosen[i-1, w-wi]
return M[n, W]
```

Knapsack Algorithm

1M. 10k → 20k → 30k →

$(n+1)(W+1) \approx nW$

M. OPT

0 → 5 → 10 → 11

W + 1

Value is NOT size.

W=0 M[1, 3]

n + 1

	0	1	2	3	4	5	6	7	8	9	10	11
φ i=0	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

M[3, 5] ~~max~~ M[2, 5] = 7

$V_3 + M[2, 0] = 18 + 0 = 18$

$W - W_i = 5 - W_3 = 0$

5.1M

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

W = 11

Input Size NOT changed. 1M 1,000,000

input Value. 5.1M

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

Knapsack Problem: Running Time

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

- Not polynomial in input size!
- "Pseudo-polynomial." ← depend on value of input.
- Decision version of Knapsack (subset sum) is NP-complete. [Chapter 8]

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