DYNAMIC PROGRAMMING SUBSET SUM

CS340

Subset Sum

- You have n items {1, 2,…, n}
- Each item has a weight $\{w_1, w_2, \ldots, w_n\}$
- You want add as much weight as possible, without exceeding a maximum weight, W

Real life examples

- Production Scheduling
  - An assembly line is available for W minutes
  - You want to schedule jobs to maximize utilization of the line
- Filling a knapsack with the maximum possible weight
  - Carry as many books as possible to school
  - Don't overload it the knapsack or it will break

How to solve this problem

- Items = {2, 1, 13, 4, 2, 8, 1}
- Total weight = 11

- Optimal solution(s)?

How about a greedy solution?

- Bag will hold W weight, there are 3 items
- Sort by weight, largest to smallest…
  - W/2 + 1, W/2, W/2

- Sort by weight, smallest to largest
  - 1, W/2, W/2

- What is the greedy solution?
- What is the optimal solution?

Optimal Solution

- What properties does an optimal solution have?
  - How many items are in the optimal solution?
  - What total weight does the optimal solution have?

Optimal Solution

- Imagine the optimal solution consists of a subset of the available items, S
- Before we add the last item, the optimal set (so far) is S'
- What do we know about the optimal set before the last item was added?
  - W(S') < W(S) <= W

Optimal Solution

- Why wasn't item x added?
  - It would have gone over the weight limit
    - $W(S') + W(x) > W$
  - It wasn't optimal
    - $W(S') + W(x) < W(S')$ + the last item added ($=W(S)$)

Optimal Solution

- At the first step, the first item is added to the knapsack
- At each successive step, either
  - The next item is added to the knapsack
  - The next item is not added to the knapsack
- One of these choices will lead to the optimal solution
  - It doesn't matter which if we calculate them all
- There are N items that can be added. Max weight is W
  - How many distinct problems are there?

Optimal Solution

- There are actually only n x W distinct subproblems to consider!
- Hence: Create an n x W matrix
- Each entry corresponds to the OPT total value for the first i items subject to a total weight of W

An example

- Items $w_i = \{2,3,4\}$
- $W = 6$

- What is the optimal solution?

An example

	3 (4)	0						
	2 (3)	0						
i	1 (2)	0						
i-1	0	0	0	0	0	0	0	0
		0	1	2	3	4	5	6
			$w-w_i$		w			

W

Items $w_i = \{2,3,4\}$
N = 3
W = 6

A row represents:
The better of

- The optimum so far (from the row below) = OPT(i-1,w)
- The optimum so far + the next item = w_i + OPT(i-1, $w-w_i$)

An example

	3 (4)	0	0	2	3	4	5	6
	2 (3)	0	0	2	3	3	5	5
i	1 (2)	0	0	2	2	2	2	2
i-1	0	0	0	0	0	0	0	0
		0	1	2	3	4	5	6

W

$w - w_i$ w

Items $w_i = \{2,3,4\}$
N = 3
W = 6

Each cell = OPT(i,w)
if $w < w_i$ then OPT(i,w) = OPT(i-1,w)
else OPT(i,w) = MAX(OPT(i-1,w), w_i + OPT(i-1, $w - w_i$))

Another example

Items = {2,1,2,3}
N = 4
W = 6

i

4 (3)	0						
3 (2)	0						
2 (1)	0						
1 (2)	0						
0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6

w

What is the complexity

- What is the complexity of each computation?
- How many computations? (nxW)
- Is this polynomial time, based on the size of the input?
  - What is the size of the input?
  - What happens if W is large?
- W is not the representation of W
  - W depends on how many bits are used to encode it
  - Think about a binary representation of W requiring 2^k bits
  - If we double W, the input size increases by 1 bit, but running time doubles
  - Running time is exponential in k $= O(n2^k)$

The Knapsack Problem

- A lot like subset sum, but each item has a value in addition to a weight.
- Value vector $\{v_1, v_2, \ldots, v_n\}$
- We want to maximize *value*, while not exceeding maximum weight. Some items might be proportionately more valuable than others.

The Knapsack Problem

- Each cell = OPT(i,w)
- if $w < w_i$ then OPT(i,w) = OPT(i-1,w)
- else OPT(i,w) = MAX(OPT(i-1,w), v_i + OPT(i-1, w- w_i)

What's different?

The Knapsack Problem

Weights = {2,1, 2, 3}
Values = {4, 5, 1, 2}
N = 4
W = 6

	4	0						
	3	0						
	2	0						
i	1	0						
	0	0	0	0	0	0	0	0
		0	1	2	3	4	5	6

w