Dynamic Programming

- The Knapsack Problem
- Dynamic Programming Formulation
- Implementation
- Fractional Knapsack and Subset Sum Problem

Imdad ullah Khan

Knapsack Problem

Input:

- Items: $U = \{a_1, \ldots, a_n\}$ $\triangleright$ Fixed order
- Weights: $w : U \to \mathbb{Z}^+$ $\triangleright$ $(w_1, \ldots, w_n)$
- Values: $v : U \to \mathbb{R}^+$ $\triangleright$ $(v_1, \ldots, v_n)$
- Capacity: $C \in \mathbb{R}^+$

Output:

- A subset $S \subset U$
- Capacity constraint:

$$\sum_{a_i \in S} w_i \leq C$$

- Objective: Maximize

$$\sum_{a_i \in S} v_i$$

Knapsack Problem: Fractional Version

- Unlike $0-1$ Knapsack, where you either take or leave an item
- Here you are allowed to take part of an item
- Easy solution
- Greedily choose the best value per unit weight item
- Proof of optimality follows from the the cut and paste type argument

Knapsack Problem: Fractional vs $0-1$ Knapsack

- Fractional solution is often not feasible for $0-1$ knapsack problem
- Select 1.27 of a software or 2.9 dish washers

ID	w	v	v/w
a_1	20	30	1.5
a_2	50	60	1.2
a_3	50	50	1

$C = 110$

- GREEDY-1 (most valued first) yields $\{a_2, a_1\}$, value 90, weight 70
- GREEDY-2 (least weighted first) yields $\{a_1, a_2\}$, value 90, weight 70
- GREEDY-3 (highest val/wt ratio first) yields $\{a_1, a_2\}$, value 90, weight 70
- INTEGRAL-OPT yields $\{a_2, a_3\}$, value 110, weight 100
- FRACTIONAL-OPT yields $\{1(a_1), 1(a_2), 4/5(a_3)\}$, value 130, weight 100
- INTEGRAL-OPT may not use total capacity
- $value(\text{FRACTIONAL-OPT}) \geq value(\text{INTEGRAL-OPT})$

Subset Sum Problem

Input:

- Items: $U = \{a_1, \ldots, a_n\}$ $\triangleright$ fixed order
- Weights: $w : U \rightarrow \mathbb{Z}^+$ $\triangleright$ $(w_1, \ldots, w_n)$
- Capacity: $C \in \mathbb{R}^+$

Output:

- A subset $S \subset U$
- Capacity constraint: $\sum\limits_{a_i \in S} w_i \leq C$
- Objective: Maximize $\sum\limits_{a_i \in S} w_i$

- A CPU with C MFLOPS
- n jobs: job i requires w_i MFLOPS
- Select jobs to get fewest idle CPU cycles

Subset Sum Problem

Brute Force Solution

- Find all subsets of U
- Calculate their sums
- Choose the one with the max sum

But there are 2^n subsets

Try Greeedy Algorithms

- It is a special case of the $0 - 1$ Knapsack problem
- When all weights are equal to values

Subset Sum Problem: Dynamic Programming

Let $\text{OPT-VAL}(n, C)$ be the value of the optimal solution

Let $\text{OPT-SET}(n, C)$ be the optimal solution (the subset)

w_1	w_2	w_3	$\dots$	$\dots$	$\dots$	w_{n-1}	w_n

- Either n^{th} item is part of the solution
  - $\text{OPT-VAL}(n, C) = \text{OPT-SET}(n-1, C - w_n) + w_n$
- Or it is not
  - $\text{OPT-VAL}(n, C) = \text{OPT-SET}(n-1, C)$
- And we take maximum of the two

Subset Sum Problem: Dynamic Programming

The Recurrence

$$\textsc{opt-val}(n, C) = \begin{cases} \textsc{opt-val}(n-1, C-w_n) + w_n, & \text{if } v_n \in \textsc{opt-set}(n, C) \\ \textsc{opt-val}(n-1, C) & \text{if } v_n \notin \textsc{opt-set}(n, C) \end{cases}$$

- Some (base) special cases
  - If no items: $\textsc{opt-val}(0, \cdot) = 0$
  - If no space: $\textsc{opt-val}(\cdot, 0) = 0$
  - If $w_n > C$: $\textsc{opt-val}(n, C) = \textsc{opt-val}(n-1, C)$