CompSci 260P
Spring 2023 Lecture 8:
Coping with Complexity

About that TSP Running Time

- Using my "speed run" $\mathcal{O}(n^2 2^n)$ algortihm
- $n = 23$ case, about 40 seconds.
- Estimates:
  - $n = 30$ about 1.5 hours
    - But it would need $\approx$ 30 GB of RAM
  - $n = 35$ about 45 hours
    - But it would need over 1 TB of RAM

Recall the 0-1 Knapsack Problem

- Maximum weight we can carry: W
- Set S of n possibly useful items; each
  - w_i (positive integer) weight
  - b_i (positive, numeric) benefit
- Goal: pick items such that:
  - Total weight selected $\leq W$
  - Maximize benefit

Dynamic Programming Algorithm

Define Knap(i, j) to be the maximum value obtainable

- Using only items $1 \ldots i$
- Using at most j total weight

Fill in base cases
for $i = 1 \ldots n$ **do**
  for $j = 1 \ldots W$ **do**
    Fill in Knap(i, j) by recurrence // $\mathcal{O}(1)$

$O(nW)$

Problems with Knapsack

How big is W (bits)?

- Suppose with `std::uint8_t`, 1 second timing.
- How long with `std::uint16_t`? 4.2 minutes
- How long with `std::uint32_t`? 194 days
- How long with `std::uint64_t`? $\approx$ 4B x 194 days
- Even if it is hard to imagine for Knapsack, this applies to other problems too.
- With Knapsack, we can do something about it.

What to do with $\mathcal{NP}$-complete

Knapsack is $\mathcal{NP}$-complete
We do not expect a solution that is both:

- Polynomial time : efficiency
- Correct

The algorithm from week 1 is the second.

What about looking for polynomial time?

- How close is close enough?

Trimming the Knapsack

- 16-bit weights were much faster.
- Why not scale weights to solve 32 bits?
  - Take the 16 most significant bits of W and $\{w_i\}$
- Example:
  - If $W = 4B$
    binary:
    1110 1110 0110 1011 ~~0010 1000 0000 0000~~
  - New $W = 1110\ 1110\ 0110\ 1011$ (binary) $= 61035$
  - Similar for each $\{w_i\}$
- There are two problems with this

Feasible?

Alternate algorithm for Knapsack

$Knap(i, 0) = 0$

- Suppose values are unsigned integer.
- Define Knap(i, V) to be the smallest knapsack weight W such that a subset of items $\{1, \ldots i\}$ ~~can be taken with value~~ that weigh at least V.
- Let's find a recursive solution to this.
- Decision being made for the ith item?

take_ith = W_i + Knap$(i-1, \max(V - v_i, 0))$

notake_ith = Knap$(i-1, V)$

if $V > \sum_{j=1}^{i-1} v_j$ then return take_ith

return min(take_ith, notake_ith)

Iterative Algorithm

$O(n^2 \cdot v^*)$

used to estimate running time

where $v^* = \max_j v_j$

$\leq n \cdot v^*$

iterations inner loop

for $i = 0, \ldots, n$ **do**
  $M[i, 0] = 0$
for $i = 1, \ldots, n$ **do**
  for $V = 1, \ldots, \sum_{j=1}^{i} v_j$ **do**
    if $V > \sum_{j=1}^{i-1} v_j$ **then**
      $M[i, V] = w_i + M[i-1, \max(0, V - v_i)]$
    else
      nousei $= M[i-1, V]$
      usei $= w_i + M[i-1, \max(0, V - v_i)])$
      $M[i, V] = \min($ nousei, usei $)$
return Max value V such that $M[n, V] \leq W$

But we're not done

- But $\mathcal{O}(n^2 v^*)$ is still pseudo-polynomial!
- Run time now depends on values not weights.
- Rounding values keeps a feasible solution
- I will also show you it's close to optimal.

Rounding values

Key:

- v_i : the real value from input
- $\tilde{v}_i$: the rounded value
- $\hat{v}_i$: rounded and truncated value

Pick a value b to use in rounding.

- For each value, set $\tilde{v}_i = \lceil v_i/b \rceil b$
- $v_i \leq \tilde{v}_i \leq v_i + b$

$\hat{v}_i = \lceil v_i/b \rceil$

Run knapsack w/ $(w_i, \hat{v}_i)$

How much to round?

- This is a trade-off
- You have seen trade-offs in previous classes.
  - Hash table trades space for time
- We are going to trade accuracy for time

Choosing a rounding

- Pick a value ε
- Set $b = (\varepsilon/2n)\max_i v_i$
- Solve with associated $\hat{v}_i$ values.

$$\tilde{v}_i = \left\lceil \frac{v_i}{b} \right\rceil \cdot b$$

Claim: This is polynomial time **if** ε **is a constant**

$$v^* = \max_i v_i = \hat{v}_j \quad (\text{some } j)$$

$$\left\lceil \frac{v_j}{b} \right\rceil = n/\varepsilon \qquad O(n^2 \cdot v^*) = O\left(n^3 \cdot \frac{1}{\varepsilon}\right)$$

How accurate?

$$\sum_{i\in S^*} v_i \le \sum_{i\in S^*} \tilde{v}_i \le \sum_{i\in S} \tilde{v}_i \le \sum_{i\in S} (v_i + b) \le nb + \sum_{i\in S} v_i$$

- My algorithm returns a set S
- Let S^* denote any feasible: $\sum_{i\in S^*} w_i \le W$
- I **claim** the following:

$$(1+\varepsilon)\sum_{i\in S} v_i \ge \sum_{i\in S^*} v_i$$

Why is this?

- We found optimal on $\{\tilde{v}_i\}$ so: $\sum_{i\in S} \tilde{v}_i \ge \sum_{i\in S^*} \tilde{v}_i$
- And the rounded values are close to real ones:

$\cancel{\sum_{i\in S^*} v_i}$

How inaccurate?

- We saw that:

$$\sum_{i \in S} v_i \geq \sum_{i \in S} \tilde{v}_i - nb$$

- Because any given item can fit in the Knapsack

$$\sum_{i \in S} \tilde{v}_i \geq \tilde{v}_j = 2nb/\varepsilon$$

$nb \leq \varepsilon \sum_{i \in S} v_i$

- Combine:

$$\sum_{i \in S} \tilde{v}_i \leq \sum_{i \in S} v_i + nb \leq$$

$$\sum_{i \in S} v_i \geq ((2/\varepsilon) - 1)nb$$

$(1+\varepsilon) \sum_{i \in S} v_i$

Where to go from here?

- Practice at the edge of your ability
- Do you play musical instruments or sports?
- For the exam:
  - Recognize complex (for a computer) problems
  - Solve exact via dynamic programming
    (Not approximations like today's lecture)