CompSci 260P
Spring 2023 Lecture 8: Coping with Complexity
About that TSP Running Time

- Using my “speed run” $O(n^2 2^n)$ algorithm
- $n = 23$ 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
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

\[
\begin{align*}
\text{for } i = 1 \ldots n \text{ do} \\
\quad \text{for } j = 1 \ldots W \text{ do} \\
\qquad \text{Fill in } Knap(i, j) \text{ by recurrence } // \mathcal{O}(1)
\end{align*}
\]
Problems with Knapsack

How big is \( W \) (bits)?

- Suppose with \texttt{std::uint8\_t}, 1 second timing.
- How long with \texttt{std::uint16\_t}? 4.2 minutes
- How long with \texttt{std::uint32\_t}? 194 days
- How long with \texttt{std::uint64\_t}? \( \approx 4.8 \times 194 \) 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
- 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:
    \[1110\ 1110\ 0110\ 1011\ \text{61035}\]
▶ New \( W = 1110\ 1110\ 0110\ 1011 \) (binary) = 61035
  ▶ Similar for each \( \{w_i\} \)
▶ There are two problems with this

Feasible?
Alternate algorithm for Knapsack

- 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 at least \(V\).
- Let's find a recursive solution to this.
- Decision being made for the \(i\)th item?

\[
\begin{align*}
\text{take}_i &= \text{wi} + \text{Knap}(i-1, \max(V-v_i, 0)) \\
\text{notake}_i &= \text{Knap}(i-1, V) \\
\text{if } V > \sum_{j=1}^{i} v_j \text{ then return } \text{take}_i \\
\text{return } \min(\text{take}_i, \text{notake}_i)
\end{align*}
\]
Iterative Algorithm

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 = $w_i + M[i - 1, \max(0, V - v_i)]$
      $M[i, V] = \min(\text{nousei}, \text{usei})$
  
return Max value $V$ such that $M[n, V] \leq W$
But we’re not done

- But $O(n^2 v^*)$ 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 $\tilde{v}_i = \lceil v_i/b \rceil b$
- $v_i \leq \tilde{v}_i \leq v_i + b$

Run knapsack w/ $(w_i, \tilde{v}_i)$
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 $\varepsilon$
► Set $b = (\varepsilon / 2n) \max_i v_i$
► Solve with associated $\hat{v}_i$ values.

$$\hat{v}_i = \left\lfloor \frac{v_i}{b} \right\rfloor \cdot b$$

Claim: This is polynomial time if $\varepsilon$ is a constant

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

$$\left\lfloor \frac{v_j}{b} \right\rfloor = \frac{n}{\varepsilon}$$

$$O(n^2 \cdot v^*) = O(n^3 \cdot \frac{1}{\varepsilon})$$
How accurate?

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

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

Why is this?

- We found optimal on $\{\tilde{v}_i\}$ so:

$$\sum_{i \in S} \tilde{v}_i \geq \sum_{i \in S^*} \tilde{v}_i$$

- And the rounded values are close to real ones:
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

\[ \sum_{i \in S} \tilde{v}_i \geq \tilde{v}_j = \frac{2nb}{\varepsilon} \]

- Combine:

\[ \sum_{i \in S} \tilde{v}_i \leq \sum_{i \in S} v_i + nb \leq \sum_{i \in S} v_i \geq ((2/\varepsilon) - 1)nb \]
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)