Announcements

- HW 7. Partially Posted
- I’ve moved
  - My new office: CSE2 344

Optimal linear interpolation

\[ \text{Error} = \sum (y_i - ax_i - b)^2 \]

Optimal solution with \( k \) segments extends an optimal solution of \( k-1 \) segments on a smaller problem

Optimal linear interpolation with \( K \) segments

Variable number of segments

- Segments not specified in advance
- Penalty function associated with segments
- Cost = Interpolation error + \( C \times \#\text{Segments} \)

Penalty cost measure

\[ \text{Opt}[j] = \min_i \{ \text{Opt}_{k-1}[i] + E_{ij} \} \text{ for } 0 < i < j \]

\[ \text{Opt}[j] = \min_i (E_{1,j}, \min_i (\text{Opt}[i] + E_{ij} + P)) \]
Subset Sum Problem

- Let \( w_1, \ldots, w_n = \{6, 8, 9, 11, 13, 16, 18, 24\} \)
- Find a subset that has as large a sum as possible, without exceeding 50

Adding a variable for Weight

- \( \text{Opt}[j, K] \) the largest subset of \( \{w_1, \ldots, w_j\} \) that sums to at most \( K \)
- \( \{2, 4, 7, 10\} \)
  - \( \text{Opt}[2, 7] = \)
  - \( \text{Opt}[3, 7] = \)
  - \( \text{Opt}[3, 12] = \)
  - \( \text{Opt}[4, 12] = \)

Subset Sum Recurrence

- \( \text{Opt}[j, K] \) the largest subset of \( \{w_1, \ldots, w_j\} \) that sums to at most \( K \)

Subset Sum Grid

\[
\begin{array}{cccccccccccc}
4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

\( \{2, 4, 7, 10\} \)

Subset Sum Code

\[
\text{for } j = 1 \text{ to } n \\
\text{for } k = 1 \text{ to } W \\
\text{Opt}[j, k] = \max(\text{Opt}[j-1, k], \text{Opt}[j-1, k-w_j] + w_j)
\]

Knapsack Problem

- Items have weights and values
- The problem is to maximize total value subject to a bound on weight
- Items \( \{I_1, I_2, \ldots, I_n\} \)
  - Weights \( \{w_1, w_2, \ldots, w_n\} \)
  - Values \( \{v_1, v_2, \ldots, v_n\} \)
  - Bound \( K \)
- Find set \( S \) of indices to:
  - Maximize \( \sum_{i \in S} v_i \) such that \( \sum_{i \in S} w_i \leq K \)
Knapsack Recurrence

Subset Sum Recurrence:
\[ \text{Opt}[j, K] = \max(\text{Opt}[j - 1, K], \text{Opt}[j - 1, K - w_j] + w_j) \]

Knapsack Recurrence:

Knapsack Grid

\[ \text{Opt}[j, K] = \max(\text{Opt}[j - 1, K], \text{Opt}[j - 1, K - w_j] + v_j) \]

Weights \{2, 4, 7, 10\}  Values: \{3, 5, 9, 16\}

Dynamic Programming Examples

- Examples
  - Optimal Billboard Placement
    - Text, Solved Exercise, Pg 307
  - Linebreaking with hyphenation
    - Compare with HW problem 6, Pg 317
  - String approximation
    - Text, Solved Exercise, Page 309

Billboard Placement

- Maximize income in placing billboards
  - \(b_i = (p_i, v_i)\), \(v_i\): value of placing billboard at position \(p_i\)
- Constraint:
  - At most one billboard every five miles
- Example
  - \{(6,5), (8,6), (12, 5), (14, 1)\}

Design a Dynamic Programming Algorithm for Billboard Placement

- Compute \(\text{Opt}[1], \text{Opt}[2], \ldots, \text{Opt}[n]\)
- What is \(\text{Opt}[k]\)?

\(\text{Opt}[k] = \text{fun}(\text{Opt}[0], \ldots, \text{Opt}[k-1])\)

- How is the solution determined from sub problems?
Solution

\[ j = 0; \quad \text{if \ } j \text{ is five miles behind the current position} \]

// the last valid location for a billboard, if one placed at \( P[k] \)
for \( k = 1 \) to \( n \)
   while \( (P[j] < P[k] - 5) \)
      \( j := j + 1; \)
   \( j := j - 1; \)
   \( \text{Opt}[k] = \max(\text{Opt}[k-1], V[k] + \text{Opt}[j]); \)