Optimal linear interpolation

Optimal linear interpolation with $K$ segments

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.
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

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

- Opt[ j, K ] the largest subset of \{w_1, \ldots, w_j\} that sums to at most K
Subset Sum Grid

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

\{2, 4, 7, 10\}
Subset Sum Code

for j = 1 to n
  for k = 1 to W
    Opt[j, k] = max(Opt[j-1, k], 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 Opt[1], Opt[2], ..., Opt[n]
- What is Opt[k]?

Input $b_1, ..., b_n$, where $b_i = (p_i, v_i)$, position and value of billboard $i$
Opt[k] = fun(Opt[0], ..., Opt[k-1])

• How is the solution determined from subproblems?

Input $b_1, ..., b_n$, where $b_i = (p_i, v_i)$, position and value of billboard $i$
Solution

\[ j = 0; \quad \text{// } j \text{ is five miles behind the current position} \]
\[ \text{// the last valid location for a billboard, if one placed at } P[k] \]

for \( k := 1 \) to \( n \)

\[ \text{while } (P[j] < P[k] - 5) \]
\[ j := j + 1; \]
\[ j := j - 1; \]
\[ \text{Opt}[k] = \text{Max}(\text{Opt}[k-1], V[k] + \text{Opt}[j]); \]