CSEP 521
Applied Algorithms
Richard Anderson
Lecture 6
Dynamic Programming
Announcements

• Midterm today!
  – 60 minutes, start of class, closed book

• Reading for this week
  – 6.1, 6.2, 6.3., 6.4

• Makeup lecture
  – February 19, 6:30 pm.
    • Still waiting on confirmation on MS room.
Dynamic Programming

• Weighted Interval Scheduling
• Given a collection of intervals $I_1, \ldots, I_n$ with weights $w_1, \ldots, w_n$, choose a maximum weight set of non-overlapping intervals.
Optimality Condition

- $\text{Opt}[j]$ is the maximum weight independent set of intervals $I_1, I_2, \ldots, I_j$
- $\text{Opt}[j] = \max(\text{Opt}[j - 1], w_j + \text{Opt}[p[j]])$
  - Where $p[j]$ is the index of the last interval which finishes before $I_j$ starts
Algorithm

MaxValue(j) =
    if j = 0 return 0
else
    return max( MaxValue(j-1),
                w_j + MaxValue(p[ j ]))

Worst case run time: $2^n$
A better algorithm

M[ j ] initialized to -1 before the first recursive call for all j

MaxValue(j) =
  if j = 0 return 0;
  else if M[ j ] != -1 return M[ j ];
  else
    M[ j ] = max(MaxValue(j-1), w_j + MaxValue(p[ j ]));
  return M[ j ];
Iterative version

MaxValue (j) {
    M[ 0 ] = 0;
    for (k = 1; k <= j; k++) {
        M[ k ] = max(M[ k-1 ], w_k + M[ P[ k ] ]);  
    }
    return M[ j ];
}
Fill in the array with the Opt values

$$\text{Opt}[j] = \max (\text{Opt}[j-1], w_j + \text{Opt}[p[j]])$$
Computing the solution

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

Record which case is used in Opt computation
Dynamic Programming

• The most important algorithmic technique covered in CSEP 521

• Key ideas
  – Express solution in terms of a polynomial number of sub problems
  – Order sub problems to avoid recomputation
Optimal linear interpolation

$$\text{Error} = \sum (y_i - ax_i - b)^2$$
What is the optimal linear interpolation with three line segments?
What is the optimal linear interpolation with two line segments?
What is the optimal linear interpolation with n line segments
Notation

- Points $p_1, p_2, \ldots, p_n$ ordered by x-coordinate ($p_i = (x_i, y_i)$)
- $E_{i,j}$ is the least squares error for the optimal line interpolating $p_i, \ldots, p_j$
Optimal interpolation with two segments

• Give an equation for the optimal interpolation of $p_1, \ldots, p_n$ with two line segments

• $E_{i,j}$ is the least squares error for the optimal line interpolating $p_i, \ldots, p_j$
Optimal interpolation with $k$ segments

- Optimal segmentation with three segments
  - $\min_{i,j}\{E_{1,i} + E_{i,j} + E_{j,n}\}$
  - $O(n^2)$ combinations considered

- Generalization to $k$ segments leads to considering $O(n^{k-1})$ combinations
Opt\(_k[j]\) : Minimum error approximating \(p_1 \ldots p_j\) with \(k\) segments

How do you express \(\text{Opt}_k[j]\) in terms of \(\text{Opt}_{k-1}[1], \ldots, \text{Opt}_{k-1}[j]\)?
Optimal sub-solution property

Optimal solution with $k$ segments extends an optimal solution of $k-1$ segments on a smaller problem.
Optimal multi-segment interpolation

Compute $\text{Opt}[k, j]$ for $0 < k < j < n$

for $j := 1$ to $n$
    $\text{Opt}[1, j] = E_{1,j}$;
for $k := 2$ to $n-1$
    for $j := 2$ to $n$
        $t := E_{1,j}$
        for $i := 1$ to $j - 1$
            $t = \min(t, \text{Opt}[k-1, i] + E_{i,j})$
        $\text{Opt}[k, j] = t$
Determining the solution

- When Opt\([k, j]\) is computed, record the value of \(i\) that minimized the sum
- Store this value in an auxiliary array
- Use to reconstruct the solution
Variable number of segments

- Segments not specified in advance
- Penalty function associated with segments
- Cost = Interpolation error + C x #Segments
Penalty cost measure

• $\text{Opt}[j] = \min(E_{1,j}, \min_i(\text{Opt}[i] + E_{i,j} + P))$
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

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

Opt[ j, K] = max(Opt[ j – 1, K], Opt[ j – 1, K – w_j] + w_j)

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

Opt\[ j, K\] = max(Opt\[ j - 1, K\], 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 $b_1, \ldots, b_n$, 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 \( b_1, \ldots, b_n \), where \( b_i = (p_i, v_i) \), position and value of billboard i
j = 0; // j 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;
    Opt[k] = Max(Opt[k-1], V[k] + Opt[j]);
Optimal line breaking and hyphenation

• Problem: break lines and insert hyphens to make lines as balanced as possible

• Typographical considerations:
  – Avoid excessive white space
  – Limit number of hyphens
  – Avoid widows and orphans
  – Etc.
Penalty Function

- Pen(i, j) – penalty of starting a line at position i, and ending at position j

Optimal line breaking and hyphenation is computed with dynamic programming

- Key technical idea
  - Number the breaks between words/syllables
String approximation

• Given a string S, and a library of strings B = \{b_1, \ldots b_m\}, construct an approximation of the string S by using copies of strings in B.

\[ B = \{abab, bbbaaa, cccbb, ccaacc\} \]
\[ S = abacccbbaaabbcccbbcccaabab \]
Formal Model

- Strings from B assigned to non-overlapping positions of S
- Strings from B may be used multiple times
- Cost of $\delta$ for unmatched character in S
- Cost of $\gamma$ for mismatched character in S
  - MisMatch($i, j$) – number of mismatched characters of $b_j$, when aligned starting with position $i$ in $s$. 
Design a Dynamic Programming Algorithm for String Approximation

• Compute Opt[1], Opt[2], . . . , Opt[n]
• What is Opt[k]?

Target string $S = s_1s_2...s_n$
Library of strings $B = \{b_1,...,b_m\}$
$\text{Mismatch}(i,j) =$ number of mismatched characters with $b_j$ when aligned starting at position $i$ of $S$. 
Opt[k] = fun(Opt[0],...,Opt[k-1])

- How is the solution determined from subproblems?

Target string $S = s_1s_2...s_n$
Library of strings $B = \{b_1,...,b_m\}$
$\text{MisMatch}(i,j) = \text{number of mismatched characters with } b_j \text{ when aligned starting at position } i \text{ of } S.$
Solution

for $i := 1$ to $n$

\[ \text{Opt}[k] = \text{Opt}[k-1] + \delta; \]

for $j := 1$ to $|B|$

\[ p = i - \text{len}(b_j); \]

\[ \text{Opt}[k] = \min(\text{Opt}[k], \text{Opt}[p-1] + \gamma \text{MisMatch}(p, j)); \]