COMPSCI 311: Introduction to Algorithms

Lecture 16: Dynamic Programming – Sequence Alignment

Dan Sheldon

University of Massachusetts Amherst

Dynamic Programming Recipe

Step 1: Devise simple recursive algorithm

- Flavor: make “first choice”, then recursively solve subproblem

Step 2: Write recurrence for optimal value

Step 3: Design bottom-up iterative algorithm

- Weighted interval scheduling: first-choice is binary
- Rod-cutting: first choice has n options
- Subset Sum: need to “add a variable” (one more dimension)

- Now: similarity between sequences

Sequence Alignment

Example. TAIL vs TALE

For two strings $X = x_1x_2 \dots x_m, Y = y_1y_2 \dots y_n$, an alignment M is a matching between $\{1, \dots, m\}$ and $\{1, \dots, n\}$.

M is valid if

- Matching. Each element appears in at most one pair in M.
- No crossings. If $(i, j), (k, \ell) \in M$ and $i < k$, then $j < \ell$.

Cost of M:

- Gap penalty. For each unmatched character, you pay δ.
- Alignment cost. For a match (i, j), you pay $C(x_i, y_j)$.

$$\text{cost}(M) = \delta(m + n - 2|M|) + \sum_{(i,j) \in M} C(x_i, y_j).$$

Sequence Alignment

Problem. Given strings X, Y gap-penalty δ and cost matrix C, find valid alignment of minimal cost.

Example 1. TAIL vs TALE, $\delta = 0.5$, $C(x, y) = \mathbf{1}[x \neq y]$.

Example 2. TAIL vs TALE, $\delta = 10$, $C(x, y) = \mathbf{1}[x \neq y]$.

Example Recap

Example 1. TAIL vs TALE, $\delta = 0.5$, $C(x, y) = \mathbf{1}[x \neq y]$.

`TAIL-`	I not matched (gap)
`TA-LE`	E not matched (gap)

Example 2. TAIL vs TALE, $\delta = 10$, $C(x, y) = \mathbf{1}[x \neq y]$.

```
TAIL
TALE
```

Applications

Genomics

- Biologists use genetic similarity to determine evolutionary relationships.
- Genetic similarity $=$ cost of aligning DNA sequences

Spell-checkers, `diff` program, search engines.

- "preffered": (0) proffered (1) preferred (2) referred . . .

Clicker

Consider the longest common subsequence (LCS) problem: given two strings X and Y, find the longest substring (not necessarily contiguous) common to both. Is LCS a special case of sequence alignment?

A. Yes, with gap penalty $\delta = 0$ and alignment cost $\mathbf{1}[x \neq y]$

B. Yes, with gap penalty $\delta = 1$, and alignment cost ∞ if $x \neq y$, else 0

C. Yes, with gap penalty $\delta = 0$, and alignment cost ∞ if $x \neq y$, else 0

D. No

Clicker

Suppose we try to align $X =$ “banana” with $Y =$ “ana”. Assume $\delta > 0$ and the cost of a match is zero. In an optimal alignment:

A. Y will match the first occurrence of “ana” in X.

B. Y will match the second occurrence of “ana” in X.

C. Y may match any occurrence of “ana” in X.

D. The optimal alignment depends on values of δ and the mismatch cost.

Toward an Algorithm

Let O be optimal alignment. Is pair (m, n) matched in O?

- If $(m, n) \in O$ we can align $x_1 x_2 ... x_{m-1}$ with $y_1 y_2 ... y_{n-1}$.
- If $(m, n) \notin O$ then either x_m or y_n must be unmatched (by no crossing).

Value $\mathrm{OPT}(m, n)$ of optimal alignment is one of:

- $C(x_m, y_n) + \mathrm{OPT}(m-1, n-1)$, If (m, n) matched
- $\delta + \mathrm{OPT}(m-1, n)$, If m unmatched
- $\delta + \mathrm{OPT}(m, n-1)$. If n unmatched

Recurrence

Let $\mathrm{OPT}(i,j)$ be optimal alignment cost of $x_1x_2...x_i$ and $y_1y_2...y_j$.

$$\mathrm{OPT}(i,j) = \min \left\{ \begin{array}{r} C(x_i,y_j) + \ \mathrm{OPT}(i-1,j-1) \\ \delta + \ \mathrm{OPT}(i-1,j) \\ \delta + \ \mathrm{OPT}(i,j-1) \end{array} \right\}$$

And (i,j) is in optimal alignment $\iff$ first term is the minimum.

Base case?

- $\mathrm{OPT}(0,j) = j\delta$ align $X = \emptyset$ to $Y = y_1 \ldots y_j$
- $\mathrm{OPT}(i,0) = i\delta$ similar

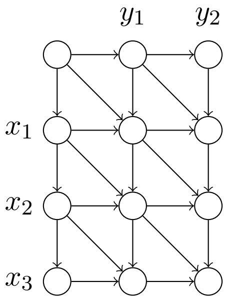
Sequence Alignment Pseudocode

```
align($X$,$Y$)
    Initialize $M[0..m, 0..n]$ = null
    $M[i,0] = i\delta$, $M[0,j] = j\delta$ for all $i, j$
    for $j = 1, \ldots, n$ do
        for $i = 1, \ldots, m$ do
            $v_1 = C(x_i, y_j) + M[i-1, j-1]$
            $v_2 = \delta + M[i-1, j]$
            $v_3 = \delta + M[i, j-1]$
            $M[i,j] \leftarrow \min\{v_1, v_2, v_3\}$
```

- Blue = recurrence, rest = DP “boilerplate”
- Running time? $\Theta(mn)$
- Example. TALE and TAIL, $\delta = 1, C(x,y) = 2 \cdot \mathbf{1}[x \neq y]$.

Sequence Alignment

- Recovering optimal matching: store each choice, trace back.
- Related to shortest path in weighted directed graph.



Graph has $\sim mn$ nodes and $\sim 3mn$ edges.

Clicker

Dijkstra's algorithm runs in $O(|E|\log|V|) \implies O(mn\log(mn))$ time for a graph with $\Theta(mn)$ nodes and edges. Sequence alignment takes only $O(mn)$ time. What can we conclude?

A. We could use dynamic programming to compute shortest paths in any graph asymptotically faster than Dijkstra's algorithm.

B. By the multiplicativity property of big-O, the $\log|V|$ factor is dominated by $|E|$, so Dijkstra's running time is $O(|E|) = O(mn)$.

C. The graph in sequence alignment is a special case where we can compute shortest paths faster.

D. Dijkstra's algorithm only works on undirected graphs.

Can We Use Less Space?

We've focused on **time** complexity, but **space** matters too!

Two sequences of length 10^5: $mn = 10^{10}$ (10 GB)

```
**for** $j = 1, \ldots, n$ **do**
    **for** $i = 1, \ldots, m$ **do**
        $v_1 = C(x_i, y_j) + M[i-1, j-1]$
        $v_2 = \delta + M[i-1, j]$
        $v_3 = \delta + M[i, j-1]$
        $M[i, j] \leftarrow \min\{v_1, v_2, v_3\}$
```

Can we save space?

- Computing column $M[\cdot, j]$ only needs $M[\cdot, j-1]$
  $\Longrightarrow$ keep just two columns (currrent, previous)
  $\Longrightarrow$ linear space $O(m+n)$
- But: can only compute cost, not recover alignment!

Sequence Alignment in Linear Space

Hirschberg's algorithm: clever combination of DP and divide-and-conquer

Goal: find shortest path from $(0,0) \to (m,n)$

Board work

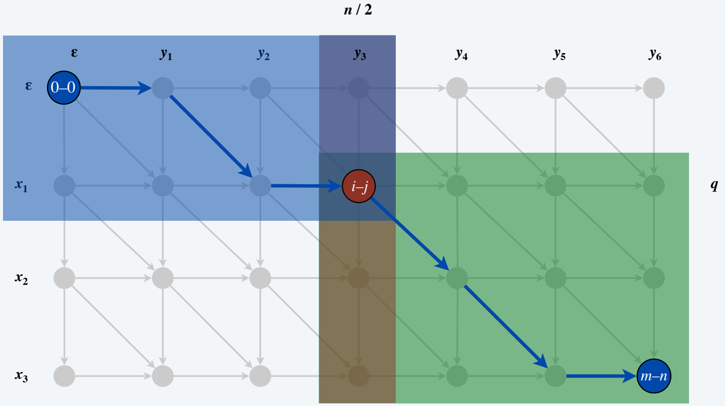
1. $\mathrm{OPT}(i,j) = f(i,j) =$ length of shortest path from $(0,0) \to (i,j)$
2. For any j, can compute $f(\cdot,j)$ in $O(mn)$ time and $O(m+n)$ space
3. Let $g(i,j) =$ length of shortest path from $(i,j) \to (m,n)$
4. For any j, can compute $g(\cdot,j)$ in $O(mn)$ time and $O(m+n)$ space
5. **Key idea:** find *one* node on shortest path. Fix $j = n/2$ and find q to maximize
$$f(q,n/2) + g(q,n/2)$$
$\implies$ node $(q,n/2)$ is on shortest path.
6. Recursively find shortest-path from $(0,0) \to (q,n/2)$
7. Recursively find shortest-path from $(q,n/2) \to (m,n)$.
8. Time $T(m,n) = T(q,n/2) + T(m-q,n/2) + O(mn)$. Solves to $O(mn)$ (recursion tree)

Space still $O(m+n)$.

Hirschberg's algorithm

Divide. Find index q that minimizes $f(q, n/2) + g(q, n/2)$; save node i–j as part of solution.

Conquer. Recursively compute optimal alignment in each piece.



slide credit: Kevin Wayne / Pearson

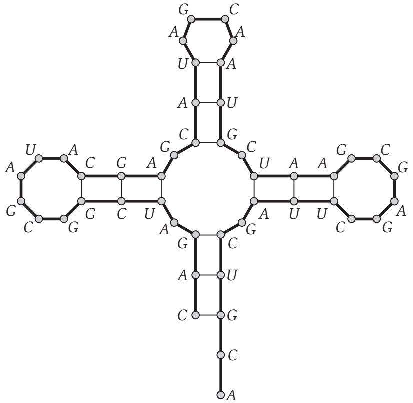
Sequence Alignment: Summary

Align sequences X, Y

- Binary choice
- Recurse on prefixes
- $O(mn)$ time
- $O(m+n)$ space: more subtle
    - DP + Divide and Conquer

More sequences:

- RNA secondary structure
- match max. $\#$ of bases
- problem substructure: over *intervals*


G
C
A
A
U
A
A
U
C
G
G
C
U
A
C
G
A
U
A
A
G
C
G
A
A
G
C
G
G
C
U
A
U
U
C
G
A
A
G
C
G
C
A
U
C
G
C
A