COMPSCI 311: Introduction to Algorithms

Lecture 15: Dynamic Programming – Sequence Alignment

Marius Minea

University of Massachusetts Amherst

slides credit: Dan Sheldon

20 March 2019

Dynamic Programming Recipe

Step 1: Devise simple recursive algorithm

Flavor: make "first choice",
then recursively solve remaining part of the problem

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)

Today: similarity between sequences

A Simple Case: Minimum Edit Distance

How many edits to go from PLEASANT to PRESENT ?

Levenshtein distance: an edit is

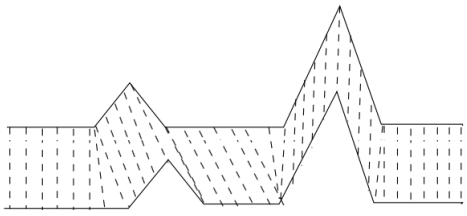
- substituting a letter
- deleting a letter
- inserting a letter

Application: spelling correction

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

Dynamic Time Warping

Measure similarity between two temporal sequences



Speech recognition, speaker recognition, gait recognition

Testing embedded systems (sensor response profile,
behavior in given scenario, e.g., braking)

Source: https://en.wikipedia.org/wiki/Dynamic_time_warping

Sequence Alignment: Motivation

- Biologists use genetic similarity to determine evolutionary relationships.
- How do we evaluate if two gene sequences are similar or not, and how similar they are ?
- We *align* them: Needleman-Wunsch algorithm (global alignment)
  Also: Smith-Waterman for local alignment (similar regions), not discussed here
- Need efficiency for long sequences
- Also used in spell-checkers, `diff` program, search engines.

Sequence Alignment: Definition

- 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 $\{1, \dots, n\}$.

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

- 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.
    How many gaps? $m + n - 2|M|$
  - Alignment cost. For a match (i, j), you pay $C(x_i, y_j)$.
    (in general, depends on the pair of mismatched symbols)

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

Sequence Alignment: Cost Example

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

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

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

Example 2. TAIL vs TALE, $\delta = 5$, $C(x, y) = \mathbf{1}[x \neq y]$.

```
TAIL
TALE
```

Clicker Question 1

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 $\delta = 0$, and alignment cost 0 for matching characters and ∞ for non-matching characters

C. Yes, with gap penalty $\delta = 1$, and alignment cost 0 for matching characters and ∞ for non-matching characters

D. No

Toward an Algorithm

- Let O be optimal alignment.
  Try binary choice: pair (m, n) aligned or not.
  - 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 (if both were matched, we'd have a crossing).
- Value $\mathrm{OPT}(m, n)$ of optimal alignment is either:
  - $C(x_m, y_n) + \mathrm{OPT}(m-1, n-1)$, If (m, n) matched
  - $\delta + \mathrm{OPT}(m-1, n)$, If m unmatched
  - $\delta + \mathrm{OPT}(m, n-1)$. If n unmatched

Could m and n *both* be unmatched? Yes. Usual for DP: $\mathrm{OPT}(m-1, n)$ and $\mathrm{OPT}(m, n-1)$ share subproblems

Recurrence

Let $\mathrm{OPT}(i, j)$ be optimal alignment cost of $x_1 x_2 ... x_i$ and $y_1 y_2 ... y_j$.

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

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 $Y = y_1 \dots y_j$
- $\mathrm{OPT}(i, 0) = i\delta$ similar

Clicker Question 2

Suppose we try to align $X =$ "banana" with $Y =$ "ana". 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 the gap penalty and alignment cost.

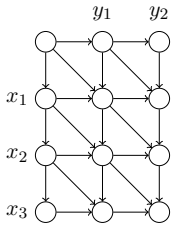
Sequence Alignment Pseudocode

```
align(X,Y)
  Initialize M[0..m,0..n] = null
  M[i,0] = iδ, M[0,j] = jδ for all i, j
  for j = 1, ..., n do
    for i = 1, ..., m do
      v1 = C(xi, yj) + M[i − 1, j − 1]
      v2 = δ + M[i − 1, j]
      v3 = δ + M[i, j − 1]
      M[i, j] ← min{v1, v2, v3}
```

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

Sequence Alignment

- Running time is $O(mn)$.
- Recovering optimal matching is easy (store choice at each step).
- Related to shortest path in weighted directed graph.



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

Clicker Question 3

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 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] $\leftarrow \min\{v_1, v_2, v_3\}$

- Computing column $M(\cdot, j)$ only needs $M(\cdot, j-1)$
  $\Rightarrow$ keep just two columns (curr, prev)
  $\Rightarrow$ linear space $O(m+n)$

But: can only compute cost, not recover alignment!

Sequence Alignment in Linear Space

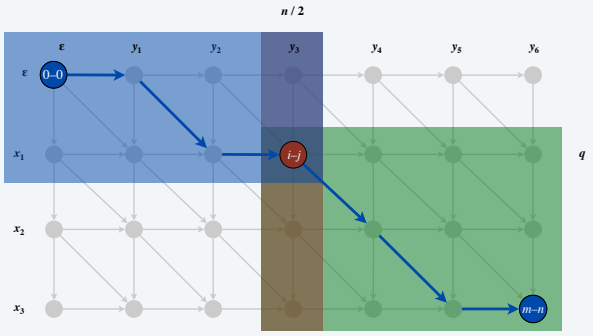
Hirschberg's algorithm: clever combination of DP and divide-and-conquer

- Split Y in half $(n/2 + n/2)$. Split will match X at some point q
  first q chars of X matched with first half of Y
  last $m - q$ chars of X matched with last half of Y
- What is the optimal q?
- Match *all* of X with first half of Y (going forward)
  Let $f(i, j)$ = cost of shortest path from $(0, 0)$ to (i, j)
  Computes column $f(\cdot, n/2)$ by increasing i
- Match *all* of X with last half of Y (going backward)
  Let $g(i, j)$ = cost of shortest path from (i, j) to (m, n)
  Computes column $g(\cdot, n/2)$ by decreasing i
- Meet in the middle: find $\min_q f(q, n/2) + g(q, n/2)$

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

Hirschberg's Linear-Space Algorithm

`align(X,Y)`
  if m < 2 or n < 2 **then** solve directly
  Compute $f(:, n/2)$ and $g(:, n/2)$ in linear space
  Find q minimizing $f(q, n/2) + g(q, n/2)$.
  Store pair $(q, n/2)$ ▷ part of alignment
  `align(X[0:q], Y[0:n/2])`
  `align(X[q+1:m], Y[n/2+1:n])` ▷ reuse memory

What is the recurrence for memory usage? Assume $m \leq n$.

$f(:, n/2)$ and $g(:, n/2)$ are $O(m)$ each, discarded after finding q.

Splitting in half on n needs space $O(\min(m, n))$

Complexity Analysis

Recurrence

$O(mn)$ work to build array of alignment costs

$T(m, n) \leq c \cdot mn + T(q, n/2) + T(m - q, n/2)$

Two-dimensional recurrence, don't know q.

Intuition: simplified case $m = n$ and assuming $q = n/2$,
we get $T'(n) \leq cn^2 + 2T'(n/2)$, for $T'(n) = T(n, n)$
This solves to $T'(n) = O(n^2)$

Can guess $T(m, n) \leq k \cdot mn$, prove by induction

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*