COMPSCI 311: Introduction to Algorithms
Lecture 14: Dynamic Programming – Sequence Alignment

Marius Minea
University of Massachusetts Amherst

slides credit: Dan Sheldon (adapted)

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
“prefered”: (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)


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

Example. TAIL vs TALE

▶ For two strings \( X = x_1 x_2 \ldots x_m, Y = y_1 y_2 \ldots y_n \), an alignment \( M \) is a matching between \( \{1, \ldots, m\} \) and \( \{1, \ldots, n\} \).

▶ \( M \) 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 \( \delta \).
▶ 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: Running Example**

**Problem.** Given strings $X,Y$ gap-penalty $\delta$ and cost matrix $C$, find valid alignment of minimal cost.

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

TAIL- I not matched
TA-LE E not matched

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

TAIL
TALE

**Clicker Question 1**

Suppose we try to align “banana” with “ana” (occurs twice).

The optimal alignment should be with

A: the first match
B: the second match
C: any of the matches
D: depends on the gap and letter mismatch penalties

**Recurrence**

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

$$OPT(i,j) = \min \begin{cases} C(x_i, y_j) + OPT(i-1, j-1) & \text{if } (i,j) \text{ matched} \\ \delta + OPT(i-1, j) & \text{if } i \text{ unmatched} \\ \delta + OPT(i, j-1) & \text{if } j \text{ unmatched} \end{cases}$$

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

**Sequence Alignment Pseudocode**

```python
def align(X, Y):
    M = [null for i in range(len(X)+1)]
    for j in range(len(Y)+1):
        M[j] = [null for i in range(len(X)+1)]
    M[0][0] = 0
    for i in range(1, len(X)+1):
        v1 = C(X[i-1], Y[0]) + M[i-1][0]
        v2 = \delta + M[i-1][1]
        v3 = \delta + M[i][0]
        M[i][0] = min(v1, v2, v3)
    for j in range(1, len(Y)+1):
        v1 = C(X[0], Y[j-1]) + M[0][j-1]
        v2 = \delta + M[1][j]
        v3 = \delta + M[0][j-1]
        M[0][j] = min(v1, v2, v3)
    for i in range(1, len(X)+1):
        for j in range(1, len(Y)+1):
            v1 = C(X[i-1], Y[j-1]) + M[i-1][j-1]
            v2 = \delta + M[i-1][j]
            v3 = \delta + M[i][j-1]
            M[i][j] = min(v1, v2, v3)
    return M[len(X)][len(Y)]
```

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

**Toward an Algorithm**

- Try what we did before: Let $O$ be optimal alignment.
  - If $(m,n) \in O$ we can align $x_1 x_2 ... x_m \cdot y_1 y_2 ... y_n$.
  - 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 $OPT(m,n)$ of optimal alignment is either:
  - $C(x_m, y_n) + OPT(m-1, n-1)$, if $(m,n)$ matched
  - $\delta + OPT(m-1, n)$, if $m$ unmatched
  - $\delta + OPT(m, n-1)$, if $n$ unmatched

**Clicker Question 2**

Suppose we try to align “banana” with “ana” (occurs twice).

The optimal alignment should be with

A: the first match
B: the second match
C: any of the matches
D: depends on the gap and letter mismatch penalties
Can We Use Less Space?

So far we’ve focused on time complexity.
But space matters too!
Two sequences of length $10^5$ each: $10^{10}$ (10 GB)

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

Computing column $C(\cdot, j)$ only requires column $C(\cdot, j - 1)$
⇒ can keep only two columns (curr, prev), linear space
But: can only compute cost, not recover alignment!

Hirschberg’s algorithm

**Edit distance graph.**
- Let $(i, j)$ denote length of shortest path from $(0,0)$ to $(i,j)$.
- Lemma: $f(i,j) = \text{OPT}(i,j)$ for all $i$ and $j$.
- Can compute $f(\cdot,j)$ for any $j$ in $O(mn)$ time and $O(m+n)$ space.

Sequence Alignment in Linear Time

Hirschberg’s algorithm: Divide and Conquer
Approach problem from both ends: forward and backward

Denote: $f(i, j) =$ cost of shortest path from $(0,0)$ to $(i, j)$ in alignment graph (solution so far)
Define $g(i, j) =$ cost of shortest path from $(i, j)$ to $(m, n)$

$$g(i, j) = \min \begin{cases} C(x_{i+1}, y_{j+1}) + \text{OPT}(i + 1, j + 1) \\ \delta + g(i + 1, j) \\ \delta + g(i, j + 1) \end{cases}$$

Same recurrence, but going backward ⇒ meet in the middle

Dijkstra’s shortest-path algorithm runs in $O(|E| \log |V|)$.
Sequence alignment runs in $O(mn)$ on a graph with $O(mn)$ nodes and edges.
What can we derive from here?
A: We could do shortest paths faster with dynamic programming
B: The $\log |V|$ does not matter compared to $O(|E|)$
C: The graph in sequence alignment is a special case
D: Dijkstra’s algorithm works on undirected graphs

Clicker Question 3

- Sequence Alignment in Linear Time
- Hirschberg’s algorithm: Divide and Conquer
- Approach problem from both ends: forward and backward
- Denote: $f(i, j) =$ cost of shortest path from $(0,0)$ to $(i, j)$ in alignment graph (solution so far)
- Define $g(i, j) =$ cost of shortest path from $(i, j)$ to $(m, n)$
- Hirschberg’s algorithm works on undirected graphs
- Dijkstra’s shortest-path algorithm runs in $O(|E| \log |V|)$.
- Sequence alignment runs in $O(mn)$ on a graph with $O(mn)$ nodes and edges.

What can we derive from here?
A: We could do shortest paths faster with dynamic programming
B: The $\log |V|$ does not matter compared to $O(|E|)$
C: The graph in sequence alignment is a special case
D: Dijkstra’s algorithm works on undirected graphs

Can We Use Less Space?

So far we’ve focused on time complexity.
But space matters too!
Two sequences of length $10^5$ each: $10^{10}$ (10 GB)

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

Computing column $C(\cdot, j)$ only requires column $C(\cdot, j - 1)$
⇒ can keep only two columns (curr, prev), linear space
But: can only compute cost, not recover alignment!

Hirschberg’s algorithm

**Edit distance graph.**
- Let $(i, j)$ denote length of shortest path from $(0,0)$ to $(i,j)$.
- Lemma: $f(i,j) = \text{OPT}(i,j)$ for all $i$ and $j$.
- Can compute $f(\cdot,j)$ for any $j$ in $O(mn)$ time and $O(m+n)$ space.
**How to Divide and Conquer?**

**Fact 1** The length of the shortest path through any point $(i, j)$ from $(0, 0)$ to $(m, n)$ is $f(i, j) + g(i, j)$ (shortest path has optimal substructure)

$\Rightarrow$ can split in two parts at some point $(i, j)$ -- which?

**Fact 2** Fix a column, $0 < k < n$ and minimize $f(q, k) + g(q, k)$ over all $0 \leq q \leq m$.

Then the shortest path from $(0, 0)$ to $(m, n)$ passes through $(q, k)$.

---

**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.

---

**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

---

**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 $(q, n/2)$ $\triangleright$ part of alignment

align(X[0:q], Y[0:n/2])

align(X[q+1:m], Y[n/2+1:n]) $\triangleright$ reuse memory

---

**Sequence Alignment: Summary**

**Problem structure:** simple

**Memory requirement:** more subtle

**DP + Divide and Conquer**

**More sequences:** RNA secondary structure

**match maximum number of bases**

**problem substructure:** over intervals