Analysis of Algorithms, I

CSOR W4231

Eleni Drinea
Computer Science Department

Columbia University

More dynamic programming: sequence alignment

Today

1 Sequence alignment

String similarity

This problem arises when comparing strings.

Example: consider an online dictionary.

- **Input:** a word, e.g., "ocurrance"
- **Output:** *did you mean* "occurrence" *?*

Similarity: intuitively, two words are similar if we can "almost" line them up by using gaps and mismatches.

Aligning strings using gaps and mismatches

A dash (–) denotes a gap.

We can align "ocurrance" and "occurrence" using

- one gap and one mismatch

o	c	–	u	r	r	a	n	c	e
o	c	c	u	r	r	e	n	c	e

gap mismatch

- or, three gaps

Alternative alignment.

o	–	c	u	r	r	–	a	n	c	e
o	c	c	u	r	r	e	–	n	c	e

gap gap gap

Strings in biology

- Similarity of english words is rather intuitive.
- Determining similarity of biological strings is a central computational problem for molecular biologists.
  - Chromosomes again: an organism's genome (set of genetic material) consists of chromosomes (giant linear DNA molecules)
  - We may think of a chromosome as an enormous linear tape containing a string over the alphabet $\{A, C, G, T\}$.
  - The string encodes instructions for building protein molecules.

Why similarity?

Why are we interested in similarity of biological strings?

- Roughly speaking, the sequence of symbols in an organism's genome determines the properties of the organism.
- So similarity can guide decisions about biological experiments.

How do we define similarity between two strings?

Similarity based on the notion of "lining up" two strings

Informally, an alignment between two strings tells us which pairs of positions will be lined up with one another.

Example: $X = \text{GCAT}$, $Y = \text{CATG}$

	x_1	x_2	x_3	x_4	
X	G	C	A	T	-
	-	C	A	T	G
Y		y_1	y_2	y_3	y_4

The set of pairs $\{(2,1),(3,2),(4,3)\}$ is an **alignment** of X, Y: these are the pairs of positions in X, Y that are **matched**.

Definition of alignment of two strings

An alignment L of $X = x_1 \dots x_m$, $Y = y_1 \dots y_n$ is a set of **ordered** pairs of indices (i, j) with $i \in [1, m]$, $j \in [1, n]$ such that the following two properties hold:

P1. every $i \in [1, m]$, $j \in [1, n]$ appears at most once in L;

P2. pairs do not *cross*: if $(i, j), (i', j') \in L$ and $i < i'$, then $j < j'$.

P1: You can only match any symbol of X w/ at most one symbol of Y, and vice versa.

Example: $X = \text{GCAT}$, $Y = \text{CATG}$

x_1	x_2	x_3	x_4	
G	C	A	T	-
-	C	A	T	G
	y_1	y_2	y_3	y_4

Goal: Align the two strings so that we use few gaps and mismatches. If you can, then the two are pretty similar.

1. $\{(2, 1), (3, 2), (4, 3)\}$ is an alignment; but
2. $\{(2, 1), (3, 2), (4, 3), (1, 4)\}$ is **not** an alignment (violates P2).

Cost of an alignment

Let L be an alignment of $X = x_1 \dots x_m$, $Y = y_1 \dots y_n$.

1. Gap penalty δ: there is a cost δ for every position of X and every position of Y that is not matched.
2. Mismatch cost: there is a cost α_{pq} for every pair of alphabet symbols p, q that are matched in L.
   - So every pair $(i, j) \in L$ incurs a cost of $\alpha_{x_i y_j}$.
   - **Assumption:** $\alpha_{pp} = 0$ for every symbol p (matching a symbol with itself incurs no cost).

The cost of alignment L is the sum of all the gap and the mismatch costs.

Cost of alignment in symbols

In symbols, given alignment L, let

- $X_i^L = 1$ **iff** position i of X is not matched (gap),
- $Y_j^L = 1$ **iff** position j of Y is not matched (gap).

Then the cost of alignment L is given by

$$cost(L) = \sum_{1 \le i \le m} X_i^L \delta \quad + \quad \sum_{1 \le j \le n} Y_j^L \delta \quad + \quad \sum_{(i,j) \in L} \alpha_{x_i y_j}$$

Examples

Example 1.

Let L_1 be the alignment shown below.

x =	x_1	x_2		x_3	x_4	x_5	x_6	x_7	x_8	x_9
X =	o	c	-	u	r	r	a	n	c	e
Y =	o	c	c	u	r	r	e	n	c	e
Y =	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	y_9	y_{10}

$L_1 = \{(1,1), (2,2), (3,4), (4,5), (5,6), (6,7), (7,8), (8,9), (9,10)\}$

$cost(L_1) = \delta + \alpha_{ae}$

mis-matched but the positions are matched together.

Examples

Example 1.

Let L_1 be the alignment shown below.

x_1	x_2		x_3	x_4	x_5	x_6	x_7	x_8	x_9
o	c	-	u	r	r	a	n	c	e
o	c	c	u	r	r	e	n	c	e
y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	y_9	y_{10}

$L_1 = \{(1,1),(2,2),(3,4),(4,5),(5,6),(6,7),(7,8),(8,9),(9,10)\}$

$cost(L_1) = \delta + \alpha_{ae}$ (This is $Y_3^{L_1} + \alpha_{x_6 y_7}$.)

$S' =$ A - - - B - C -

$S =$ A C A C B B C B

Example 2.

Let L_2 be the alignment shown below.

x_1		x_2	x_3	x_4	x_5		x_6	x_7	x_8	x_9
o	-	c	u	r	r	-	a	n	c	e
o	c	c	u	r	r	e	-	n	c	e
y_1	y_2	y_3	y_4	y_5	y_6	y_7		y_8	y_9	y_{10}

$L_2 = \{(1,1),(2,3),(3,4),(4,5),(5,6),(7,8),(8,9),(9,10)\}$

$\text{cost}(L_2) = 3\delta = Y_2^{L_2} + Y_7^{L_2} + X_6^{L_2}$

$S' = B\ C\ C\ A\ B\ B\ C\ A$

$S = A\ C\ A\ C\ B\ B\ C\ B$

Example 2.

Let L_2 be the alignment shown below.

x_1		x_2	x_3	x_4	x_5		x_6	x_7	x_8	x_9
o	-	c	u	r	r	-	a	n	c	e
o	c	c	u	r	r	e	-	n	c	e
y_1	y_2	y_3	y_4	y_5	y_6	y_7		y_8	y_9	y_{10}

$L_1 = \{(1,1),(2,3),(3,4),(4,5),(5,6),(7,8),(8,9),(9,10)\}$

$cost(L_2) = 3\delta$ (This is $X_6^{L_2} + Y_2^{L_2} + Y_7^{L_2}$.)

Example 3.

Let L_3, L_4 be the alignments shown below.

L_3

	x_1	x_2	x_3	x_4
X	G	C	A	T
Y	C	A	T	G
	y_1	y_2	y_3	y_4

L_4

	x_1	x_2	x_3	x_4	
X	G	C	A	T	-
Y	-	C	A	T	G
		y_1	y_2	y_3	y_4

$L_3 = \{(1,1), (2,2), (3,3), (4,4)\}$

$\text{cost}(L_3) = \alpha_{GC} + \alpha_{CA} + \alpha_{AT} + \alpha_{TG}$

$L_4 = \{(2,1), (3,2), (4,3)\}$

$\text{cost}(L_4) = 2\delta$

Note: Assume mismatch costs are symmetric: $\alpha_{GC} = \alpha_{CG}$.

Example 3.

Let L_3, L_4 be the alignments shown below.

x_1	x_2	x_3	x_4
G	C	A	T
C	A	T	G
y_1	y_2	y_3	y_4

x_1	x_2	x_3	x_4	
G	C	A	T	-
-	C	A	T	G
	y_1	y_2	y_3	y_4

$L_3 = \{(1,1),(2,2),(3,3),(4,4)\}$

$cost(L_3) = \alpha_{\mathrm{GC}} + \alpha_{\mathrm{CA}} + \alpha_{\mathrm{AT}} + \alpha_{\mathrm{TG}}$

$L_4 = \{(2,1),(3,2),(4,3)\}$

$cost(L_4) = 2\delta$

The sequence alignment problem

think of strings of the same order in size. ($m = \Theta(n)$)

$m = \Theta(n)$

Input:

- **two** strings X, Y consisting of m, n symbols respectively; each symbol is from some alphabet Σ
- the gap penalty δ
- the mismatch costs $\{\alpha_{pq}\}$ for every pair $(p, q) \in \Sigma^2$

Output: the **minimum** cost to align X and Y, and an optimal alignment.

Look for greedy or DP alg. For DP: 1st come up w/ a recursive solution to the problem (if you can). Then see if the problem exhibits optimal substructure, and if there are overlapping subproblems that you can exploit to organize the computations in a dynamic programming fashion.

$X = x_1\ x_2\ x_3 \cdots x_m$

$Y = y_1\ y_2\ y_3 \cdots y_n$

Let L be an optimal alignment of X and Y.

FACTS about L:

$L = \{(\ ,\ ), (\ ,\ ), \ldots, (\ ,\ )\}$

- Either $(m,n) \in L$
- OR $(m,n) \notin L \Rightarrow$ then $m \notin L$ or $n \notin L$ (otherwise, they would "cross")

Let $OPT(m,n)$ = min cost of aligning X and Y.

$$OPT(m,n) = \min \begin{cases} OPT(m-1,n-1) + \alpha_{x_m y_n}, & \text{if } (m,n) \in L \\ OPT(m-1,n) + \delta, & \text{if } m \notin L \\ OPT(m,n-1) + \delta, & \text{if } n \notin L \end{cases}$$

How to Solve this Problem:

Let L be the optimal alignment of X and Y

$X = x_1 x_2 \dots x_m$

$Y = y_1 y_2 \dots y_n$

L is a set of ordered pairs of indices.

$L = \{(\ ), (\ ), \dots, (\ )\}$

- Start from the last entry.

– What can we say about m and n?

Either $(m,n) \in L$ (the optimal alignment)

or $(m,n) \notin L \Rightarrow$ either $m \notin L$ or $n \notin L$

Suppose $(m,n) \notin L$. Then either $m \notin L$ or $n \notin L$.

$\Rightarrow$ Then either y_n is mapped w/ a gap or x_m is mapped with a gap.

Regarding m and n, there are 3 possible cases:

1. $(m,n) \in L$
2. $m \notin L$
3. $n \notin L$

only one can be true for a given L.

$OPT(m,n)$ = min cost to align $x_1 x_2 \dots x_m$ with $y_1 y_2 \dots y_n$

$$OPT(m,n) = \min \begin{cases} \alpha_{x_m y_n} + OPT(m-1,n-1), & \text{if } (m,n) \in L \\ \delta + OPT(m-1,n), & \text{if } n \notin L \\ \delta + OPT(m,n-1), & \text{if } n \notin L \end{cases}$$

For case where $n \notin L$, we still have to match $x_1 x_2 \dots x_{m-1}$ to whole string $y_1 y_2 \dots y_n$:

$x_1 x_2 \dots x_{m-1}$ ~~x_m~~

$y_1 y_2 \dots y_{n-1} y_n$

Hence, the $OPT(m-1, n)$

For case where $n \notin L$, we still have to match $y_1 y_2 \dots y_{n-1}$ to whole string $x_1 x_2 \dots x_m$

$x_1 x_2 \dots x_{m-1} x_m$

$y_1 y_2 \dots y_{n-1}$ ~~y_n~~

Hence, the $OPT(m, n-1)$

Boundary Conditions:

Match string of length i to empty string.

- $OPT(i, 0) = i \cdot \delta$ ⟶

  X $x_1 x_2 x_3 \cdots x_m$

  Y ------

- $OPT(0, j) = j \cdot \delta$ ⟶

  X ------

  Y $y_1 y_2 y_3 \cdots y_m$

We don't know which of the three cases (i.) $(m,n) \in L$, (ii) $m \notin L$, or (iii) $n \notin L$, is happening for a given position. The minimization recursion decides that for us! Whichever gives the minimum cost is what occurs in L. Then, the recursion is simply:

$$OPT(m,n) = \min \begin{cases} \alpha_{x_m y_n} + OPT(m-1, n-1) \\ \delta + OPT(m-1, n) \\ \delta + OPT(m, n-1) \end{cases}$$

We know that one of the three will occur at each position.

(Recall: $\alpha_{x_m y_n} = 0$ if $x_m = y_n$)

What output are we looking for?

$OPT(m,n)$

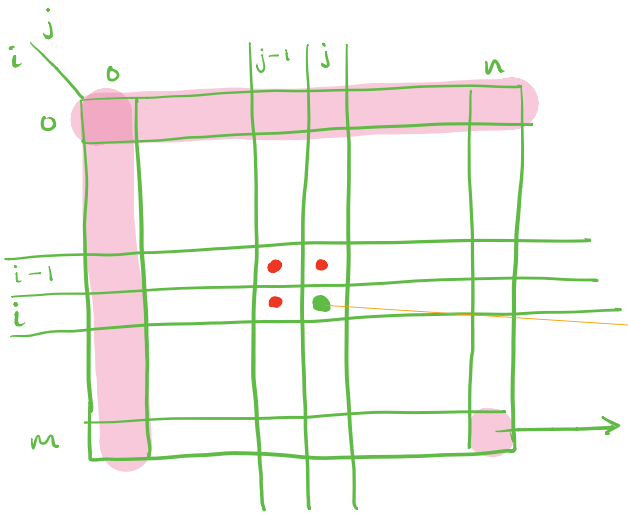
//

Space Requirement: $O(mn)$

subproblems: $\Theta(mn)$

time per subproblem: $O(1)$

$\Rightarrow$ Time for DP algorithm: $O(mn)$



0^{th} row & 0^{th} column initialized to the boundary conditions.

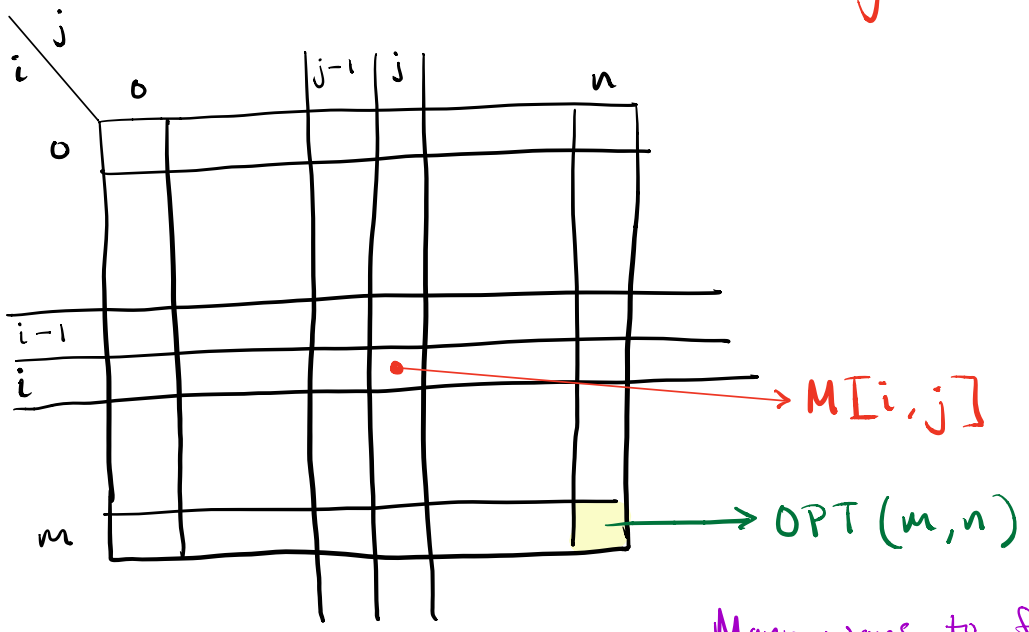
$\rightarrow M[i,j]$

$\rightarrow M[m,n] = OPT(m,n)$

- Entry i,j depends on $OPT(i-1, j-1)$, $OPT(i-1, j)$ and $OPT(i, j-1)$ $\rightarrow$ marked in red in matrix (•).
- Fill out matrix row-by-row or column-by-column, or diagonal-by-diagonal. However, what we prefer in practice is to fill it out the easiest way: row-by-row (or column-by-column).
  - You need to give information in the algorithm if you are to give an iterative bottom-up computation
- Need to give order in which you fill out the matrix in your solution and pseudocode!

M

$M[i,j] = OPT(i,j)$



Many ways to fill in M, but row-by-row is simplest (left to right, top to bottom).

- We want: $OPT(m,n)$
- Time per subproblem: $O(1)$
- # subproblems: $\Theta(mn)$
- Total time: $O(m,n)$
- Order to fill in M: Row-by-row is simplest (Left to right, top to bottom)
- Total space: $\Theta(mn)$

Towards a recursive solution

Claim 1.

Let L be the optimal alignment. Then either

1. *the last two symbols x_m, y_n of X, Y are matched in L, hence the pair $(m, n) \in L$; or*
2. *x_m, y_n are not matched in L, hence $(m, n) \notin L$. In this case, at least one of x_m, y_n is not matched in L, hence at least one of m, n does not appear in L.*

Proof of Claim 1

By contradiction.

Suppose $(m, n) \notin L$ but x_m and y_n are **both** matched in L. That is,

1. x_m is matched with y_j for some $j < n$, hence $(m, j) \in L$;
2. y_n is matched with x_i for some $i < m$, hence $(i, n) \in L$.

Since pairs (i, n) and (m, j) cross, L is not an alignment.

Rewriting Claim 1

The following equivalent way of stating Claim 1 will allow us to easily derive a recurrence.

Fact 4.

In an optimal alignment L, at least one of the following is true

1. $(m, n) \in L$; *or*
2. x_m *is not matched; or*
3. y_n *is not matched.*

The subproblems for sequence alignment

Let

$OPT(i, j) =$ minimum cost of an alignment between $x_1 \dots x_i$, $y_1 \dots y_j$

We want $OPT(m, n)$. From Fact 4,

1. If $(m, n) \in L$, we pay $\alpha_{x_m y_n} + OPT(m-1, n-1)$.
2. If x_m is not matched, we pay $\delta + OPT(m-1, n)$.
3. If y_n is not matched, we pay $\delta + OPT(m, n-1)$.

How do we decide which of the three to use for $OPT(m, n)$*?*

The recurrence for the sequence alignment problem

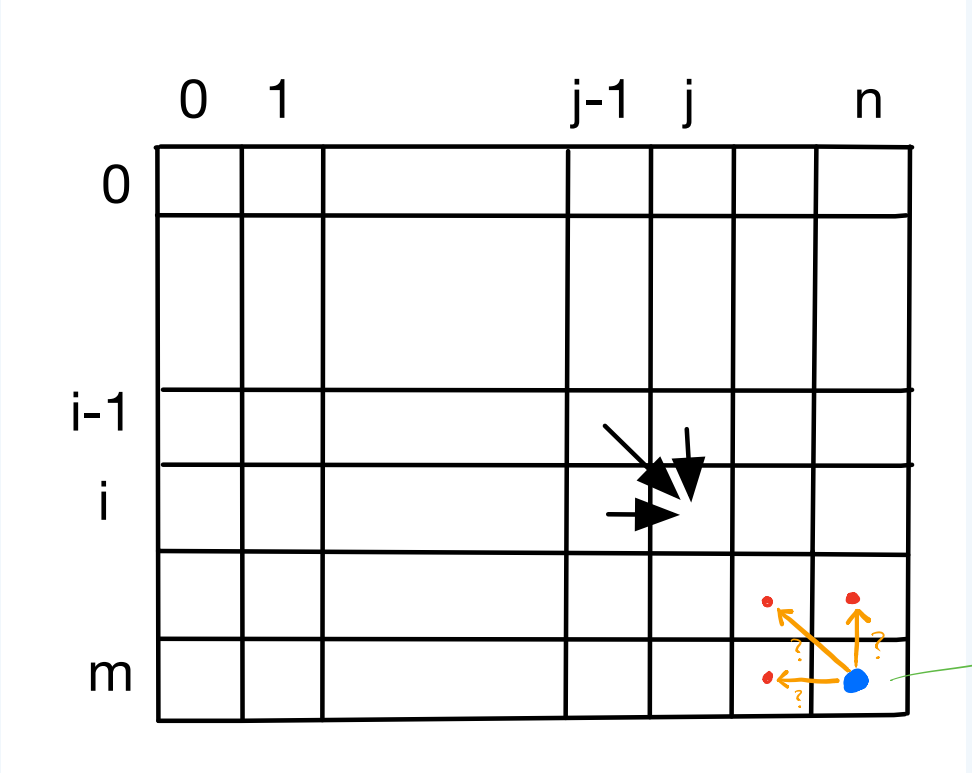
$$OPT(i,j) = \begin{cases} j\delta & \text{, if } i = 0 \\ \min \begin{cases} \alpha_{x_i y_j} + OPT(i-1, j-1) \\ \delta + OPT(i-1, j) \\ \delta + OPT(i, j-1) \end{cases} & \text{, if } i, j \geq 1 \\ i\delta & \text{, if } j = 0 \end{cases}$$

Remarks

- Boundary cases: $OPT(0, j) = j\delta$ and $OPT(i, 0) = i\delta$.
- Pair (i, j) appears in the optimal alignment for subproblem $x_1 \dots x_i, y_1 \dots y_j$ if and only if the minimum is achieved by the first of the three values inside the min computation.

Computing the cost of the optimal alignment

- M is an $(m+1) \times (n+1)$ dynamic programming table.
- Fill in M so that all subproblems needed for entry $M[i,j]$ have already been computed when we compute $M[i,j]$ (e.g., column-by-column).



Outputting the optimal solution:
- Start @ M[m,n] and move backwards
- Determine which cell gave M[m,n] and repeat.

Pseudocode

`SequenceAlignment`(X, Y)

Initialize $M[i, 0]$ to $i\delta$
Initialize $M[0, j]$ to $j\delta$
for $j = 1$ to n **do**
  for $i = 1$ to m **do**

$$M[i, j] = min\Big\{\alpha_{x_i y_j} + M[i-1, j-1],$$
$$\delta + M[i-1, j], \delta + M[i, j-1]\Big\}$$

  end for
end for
return $M[m, n]$

Running time?

Reconstructing the optimal alignment

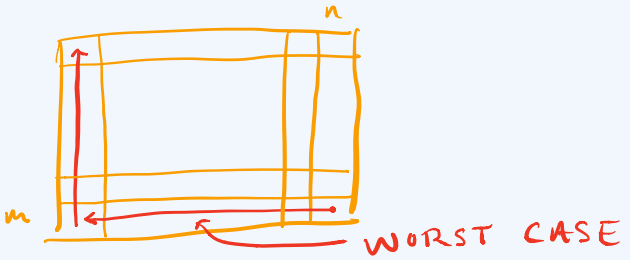
Given M, we can reconstruct the optimal alignment as follows.

```
TraceAlignment(i, j)
  if i == 0 or j == 0 then return
  else
    if M[i, j] == α_{x_i y_j} + M[i − 1, j − 1] then
      TraceAlignment(i − 1, j − 1)
      Output (i, j),
    else
      if M[i, j] == δ + M[i − 1, j] then TraceAlignment(i − 1, j)
      else TraceAlignment(i, j − 1)
      end if
    end if
  end if
```

Initial call: `TraceAlignment`(m, n)

Running time?

$O(m + n)$



Resources used by dynamic programming algorithm

- Time: $O(mn)$
- Space: $O(mn)$
  - English words: $m, n \leq 10$
  - Computational biology: $m = n = 100000$
    - Time: 10 billion ops
    - Space: 10GB table!
- *Can we avoid using quadratic space while maintaining quadratic running time?*

Using only $O(m+n)$ space

1. First, suppose we are only interested in the **cost** of the optimal alignment.
   Easy: keep a table M with 2 columns, hence $2(m+1)$ entries.

2. *What if we want the optimal alignment too?*
   - No longer possible in $O(n+m)$ time.