Sequence Alignment

https://cs.pomona.edu/classes/cs140/

Outline

Topics and Learning Objectives

- Discuss the dynamic programming paradigm
- Investigate the sequence alignment problem

Assessments

- None

Sequence Alignment

- Compute the similarity between two strings.
- For example, using the Needleman-Wunsch Similarity Score

A	G	G	G	C	T
A	G	G	--	C	A

- Total penalty = $p_{gap} + p_{AT}$
- Assume these penalties are based on biological principles

Sequence Alignment

Input:

- Two strings X = $x_1, \ldots, x_m$; and Y = $y_1, \ldots, y_n$; over the alphabet Σ
  - For example, Σ = {A, C, G, T} for genomes
- Also given a penalty value for each possible error
  - For example, p_{gap} , p_{AC} , p_{AG} , p_{AT} , p_{CG} , p_{CT} , p_{GT}

Output:

- Out of all possible alignments, output the one that minimizes total error

Sequence Alignment

Input:

- Two strings X = $x_1, \ldots, x_m$; and Y = $y_1, \ldots, y_n$; over the alphabet Σ
  - For example, $\Sigma = \{A, C, G, T\}$ for genomes
- Also given a penalty value for each possible error
  - For example, p_{gap} , p_{AC} , p_{AG} , p_{AT} , p_{CG} , p_{CT} , p_{GT}

Output:

- Out of all possible alignments, output the one that minimizes total error

How many possible alignments exist?

Example

Assume a penalty of

- 1 for each gap and
- 2 for a mismatch between symbols

A	G	T	A	C	G
A	C	A	T	A	G

What is the minimum penalty for these two strings?

Example

Assume a penalty of

- 1 for each gap and
- 2 for a mismatch between symbols

A	--	--	G	T	A	C	G
A	C	A	--	T	A	--	G

We'll say that these sequences have a common length of L

What is the minimum penalty for these two strings?

- 4

Optimal Substructure

- Let’s zoom in on the last column of the alignment

X has m values

A	G	G	G	C	**X_m?**
A	G	G	--	C	**Y_n?**

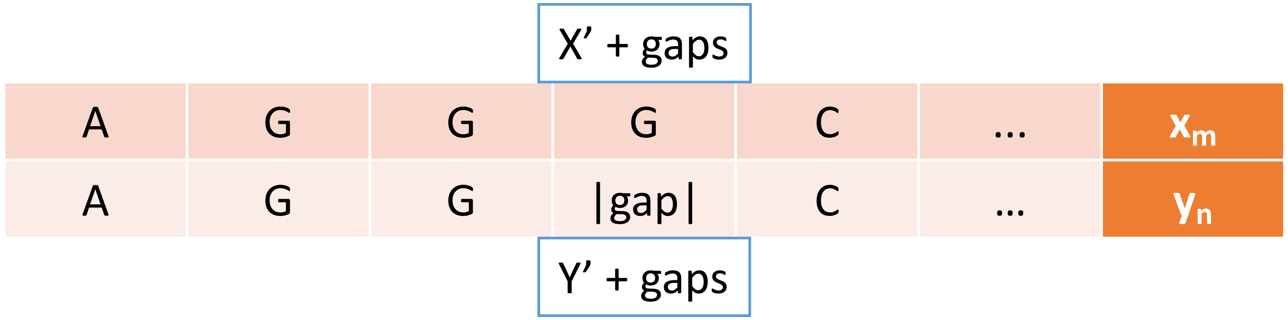
Y has n values

- How many possibilities are there for the contents of the final column of an *optimal* alignment?
  - Case 1: x_m and y_n
  - Case 2: x_m and gap (handles case where y_n is matched with something else)
  - Case 3: gap and y_n (handles case where x_m is matched with something else)

Case 1: x_m and y_n (no gap)

- Let P denote the final alignment penalty after matching x_m and y_n
- Then the penalty of the part before the final match is

$$P = P_{first} + P_{end}$$
$$P_{first} = P - P_{end}$$



X' + gaps

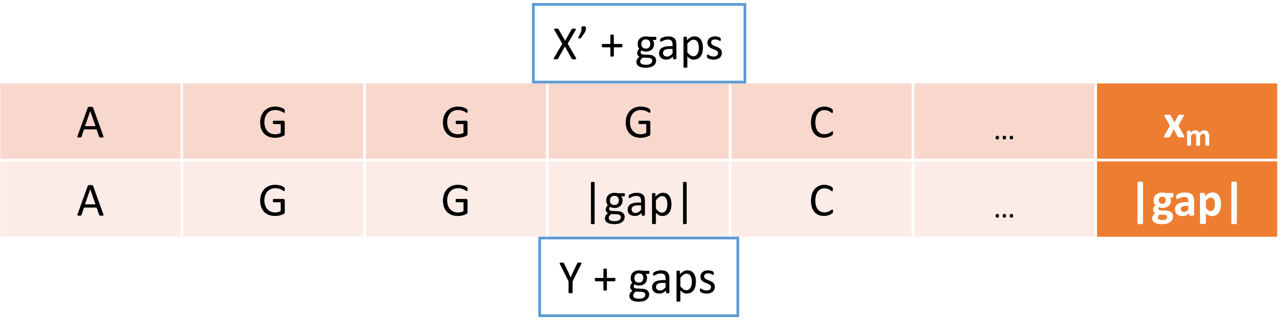
A	G	G	G	C	…	**x_m**
A	G	G	|gap|	C	…	**y_n**

Y' + gaps

- To get an optimal alignment, we want P_{first} to be optimal.

Case 2: x_m and gap

- In this case we match x_m with a gap
- We’ve removed one symbol from X (we’ll call it X’)
- But we still have the entire Y string



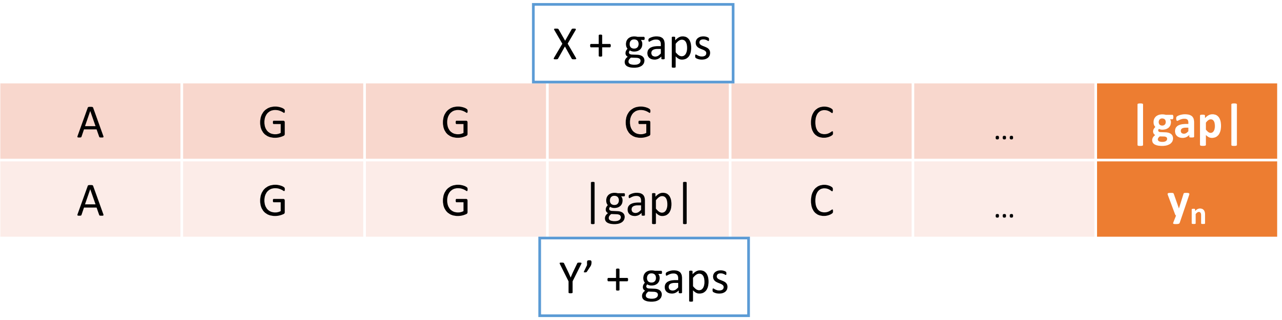
X’ + gaps

A	G	G	G	C	…	**x_m**
A	G	G	|gap|	C	…	**|gap|**

Y + gaps

Case 3: gap and y_n

- In this case we match y_n with a gap
- We’ve removed one symbol from Y (we’ll call it Y’)
- But we still have the entire X string



X + gaps

A	G	G	G	C	…	**|gap|**
A	G	G	|gap|	C	…	**y_n**

Y’ + gaps

Optimal Substructure

An optimal alignment of two strings X and Y is one of

1. An optimal alignment of X' and Y' with x_m and y_n at the end
2. An optimal alignment of X' and Y with x_m and a gap at the end
3. An optimal alignment of X and Y' with a gap and y_n at the end

What if one of X or Y is empty at this stage?

Recurrence

$$P_{i,j} = \min \begin{cases} P_{i-1,j-1} + p_{x_i,y_j} \\ P_{i-1,j} + p_{gap} \\ P_{i,j-1} + p_{gap} \end{cases}$$

Code and Running Time

A good practice problem

Things to consider

- What size is the dynamic programming table?
- What are the base cases?
- What can we fill the table in with at the beginning?
- How many loops do we need?
- What is the running time?

Proof

A good practice problem

Things to consider

- What kind of proof seems natural?
- What are the base cases?
- What is our inductive hypothesis?
- What reasoning do we need for the inductive step?

```
FUNCTION ReconstructSequence(penalties, X, Y)
    i = penalties.x_length
    j = penalties.y_length
    alignedX = ""
    alignedY = ""
    WHILE i > 0 && j > 0
        MATCH penalties[i][j]
            IF case 1
                alignedX += X[i]; i -= 1
                alignedY += Y[j]; j -= 1
            IF case 2
                alignedX += X[i]; i -= 1
                alignedY += "gap"
            IF case 3
                alignedX += "gap"
                alignedY += Y[j]; j -= 1
    fillAsNeeded(X, alignedX, Y, alignedY)
```