2.1 Alignment Recap

Recall from the previous lecture that an alignment for strings \( x \) and \( y \) over alphabet \( \Sigma \) is a sequence over pairs of elements of \( \Sigma \), no more than one of which may be replaced by the special gap character “-”. Taking the first member of every pair yields the first sequence, with gaps inserted, and similarly taking the second member of every pair yields the second sequence, gaps inserted.

Recall also that a similarity (or scoring) matrix \( \delta \) is used to score an alignment, and the alignment score is simply the sum of similarities of value pairs in the alignment.

We also discussed the edit graph, and now, we are ready to use the edit graph to give a dynamic programming algorithm to efficiently compute this score.

2.2 Pairwise Global Alignment

2.2.1 Problem Description

The following describes the given inputs and desired outputs for the pairwise global alignment algorithm:

**Given:**

- An alphabet, \( \Sigma \).
- A similarity (or scoring) matrix, \( \delta \).
- Two sequences, \( X \) and \( Y \), such that

\[
X = x_1x_2x_3 \ldots x_n = \vec{x} \\
Y = y_1y_2y_3 \ldots y_m = \vec{y}
\]

where \( x_i, y_j \in \Sigma \) for all \( i \in [1,n] \) and \( j \in [1,m] \). In other words, \( \vec{x} \in \Sigma^n \) and \( \vec{y} \in \Sigma^m \).

**Produce:**

The score for an alignment \( \vec{a} \) of length \( l \) of \( \vec{x} \) and \( \vec{y} \) that maximizes the alignment similarity score \( \sum_{i=1}^{l} \delta(\vec{a}_{i,0}, \vec{a}_{i,1}) \).
2.2.2 Pseudocode

Below is pseudocode for the pairwise global alignment algorithm:

```plaintext
1: function Global Alignment(x ∈ Σ^n, y ∈ Σ^m)
2: S_{0,0} <- 0
3: for i ∈ {1, 2, ..., n} do
4:   S_{i,0} <- S_{i-1,0} + δ(x_i, -)
5: for j ∈ {1, 2, ..., m} do
6:   S_{0,j} <- S_{0,j-1} + δ(-, y_j)
7: for i ∈ {1, 2, ..., n} do
8:   S_{i,j} <- max \{ S_{i-1,j-1} + δ(x_i, y_j), S_{i-1,j} + δ(x_i, -), S_{i,j-1} + δ(-, y_j) \}
9: return S_{n,m}
```

You may see alternate versions of the pseudocode/algorithm that complete the dynamic programming table column by column. We will be doing it row by row in this class, though, so when constructing your dynamic programming tables, be sure to have your X span the top of the table in your visual representation, while having Y span the left hand side of the table. This will ensure that our loops are carrying out the appropriate row by row filling in of the DP table. This can be confusing, so spend some time thinking about it and ask questions!

2.2.3 Recurrence Relation

Recall the recursive definition of alignment scores:

\[
S_{0,0} = 0 \\
S_{i,j} = \max \{ S_{i-1,j-1} + \delta(x_i, y_j), S_{i-1,j} + \delta(x_i, -), S_{i,j-1} + \delta(-, y_j) \}
\]

The pseudocode above correctly implements this recurrence by handling the edge cases (lines 2, 3, 4, and 6), and then every time \(S_{i,j}\) is evaluated for \(i,j > 0\), the algorithm ensures that \(S_{i-1,j}, S_{i,j-1}, \text{ and } S_{i-1,j-1}\) have already been calculated.

There are other orderings in which we could fill the matrix, for instance we could fill columnwise or diagonally, and the result would be identical. Any ordering that ensures that all recursive dependencies of a given cell are calculated before the cell in question produces an identical solution.

2.3 Pairwise Local Alignment

2.3.1 Problem Description

So far, we have examined the problem of calculating the optimal alignment between strings \(x\) and \(y\). We now study a related problem; that of local alignment.
Lecture 2: January 31

What if we have 2 strings, and we know that the majority of these strings differ greatly, but we want to identify the sections of each string that most strongly align to one another. A motivating biological example would be if we had 2 distantly related genomes, and we wanted to identify genes that were highly conserved between the two. In this case, we can’t expect the strings to align well globally, but if we can find two regions that are highly conserved between the strings, then we can expect these regions to align well.

The following describes the given inputs and desired outputs for the pairwise global alignment algorithm:

**Given:**

- An alphabet, $\Sigma$.
- A similarity (or scoring) matrix, $\delta$.
- Two sequences, $X$ and $Y$, such that
  \[
  X = x_1x_2x_3\ldots x_n = \vec{x}
  \]
  \[
  Y = y_1y_2y_3\ldots y_m = \vec{y}
  \]
  where $x_i, y_j \in \Sigma$ for all $i \in [1, n]$ and $j \in [1, m]$. In other words, $\vec{x} \in \Sigma^n$ and $\vec{y} \in \Sigma^m$.

**Produce:**

The optimal global alignment similarity score over any strings $X', Y'$ where $X'$ is a substring of $X$ and $Y'$ a substring of $Y$.

### 2.3.2 Pseudocode

Below is pseudocode for the pairwise global alignment algorithm:

1: function LOCAL ALIGNMENT ($x \in \Sigma^n, y \in \Sigma^m$)
2:     $S_{0,0} \leftarrow 0$
3:     for $i \in \{1, 2, \ldots, n\}$ do
4:         $S_{i,0} \leftarrow 0$
5:     for $j \in \{1, 2, \ldots, m\}$ do
6:         $S_{0,j} \leftarrow 0$
7:     for $i \in \{1, 2, \ldots, n\}$ do
8:         $S_{i,j} \leftarrow \max \left\{ 0, S_{i-1,j-1} + \delta(x_i, y_j), S_{i-1,j} + \delta(x_i, -), S_{i,j-1} + \delta(-, y_j) \right\}$
9:     return $\max_{i \in \{0, 1, \ldots, n\}, j \in \{0, 1, \ldots, m\}} S_{i,j}$

Note the beauty in how we went from global to local alignment algorithms with just a couple changes, namely, adding in a 0 in the recurrence relation, and taking the maximum of our DP table rather than the bottom right corner.

### 2.3.3 Recurrence Relation

Thus, for local alignment, we have the following recurrence relation:
\[ S_{0,0} = 0 \]
\[ S_{i,j} = \max \begin{cases} 
0 \\
S_{i-1,j-1} + \delta(x_i, y_j) \\
S_{i-1,j} + \delta(x_i, -) \\
S_{i,j-1} + \delta(-, y_j) 
\end{cases} \]

### 2.4 On Edit Graphs and Dynamic Programming Matrices

Recall that the edit graph has three types of edges:

1. Horizontal edges, represented as a gap in \( Y \): \((i,j) \rightarrow (i-1,j)\), for \( i \in \{1,2,\ldots,n\}\)
2. Vertical edges, represent a gap in \( X \): \((i,j) \rightarrow (i,j-1)\), for \( j \in \{1,2,\ldots,m\}\)
3. Diagonal edges, represent a match or a mismatch: \((i-1,j-1) \rightarrow (i,j)\)

What you’ll notice is that the edit graph for two sequences and dynamic programming table for those sequences when run through the alignment algorithm can be used together to determine the optimal alignment! Constructing the actual alignment from the DP matrix (using ideas related to the edit graph) is done through a process called **traceback**.

Traceback essentially starts at the cell in the DP matrix that holds the score returned by your alignment algorithm, and moves up, left, or diagonally to the top left, based on which score \((S_{i-1,j-1}, S_{i-1,j}, S_{i,j-1})\) gave you the score in \(S_{i,j}\). By moving along and replicating the path in the edit graph, you will reconstruct your maximal score alignment.

#### 2.4.1 A Note on Optimality

The key idea behind the DP algorithm in alignment (and in general) has to do with the **Principle of Optimality**, which states that the optimal answer to a problem is expressed in terms of its subproblems. In our alignment example, the subproblems can be thought of as the expressions in the right hand side of the recurrence relation, while the overarching problem can be thought of as finding the optimal score for \(S_{i,j}\).

### 2.5 Informal Introduction to Alignment Statistics

Maximum similarity alignments make intuitive sense, as we can use them to select for alignments that try to match similar reasons. However, we can also interpret them from a much deeper statistical framework. We get an informal look at this here.

Consider the following alignment:

\[
\begin{align*}
\text{GT-ACGTT} \\
\text{ACGAC-TT}
\end{align*}
\]
Here we have 4 matches, 2 gaps, and 2 mismatches.

Let $p$ be the probability of a match under alignment of uniformly random sequences, $q$ be the probability of a mismatch, and $r$ the probability of an indel (or gap).

The probability of the above alignment occurring would then be $p^4q^2r^2$, and we have the following:

$$P(\text{alignment}) = p^4q^2r^2$$

$$\log(P(\text{alignment})) = 4 \log(p) + 2 \log(q) + 2 \log(r)$$

Let $S'$ represent the log probability from above, so that we have:

$$S' = 4 \log(p) + 2 \log(q) + 2 \log(r)$$

Now, we will do a little bit of statistical magic and “normalize” or “calibrate” the score to $p$ using a new variable, $k$. We select $k$ such that:

$$\log\left(\frac{p}{k}\right) = 1$$

Next, we will “calibrate” our $S'$ using $k$ to get the score we are interested in (call it $S$):

$$S = S' - 8 \log(k)$$

$$S = 4 \log(p) + 2 \log(q) + 2 \log(r) - 8 \log(k)$$

$$S = 4 \log(p) - 4 \log(k) + 2 \log(q) - 2 \log(k) + 2 \log(r) - 2 \log(k)$$

$$S = 4 \log\left(\frac{p}{k}\right) + 2 \log\left(\frac{q}{k}\right) + 2 \log\left(\frac{r}{k}\right)$$

Note that the 8 that appears in $8 \log(k)$ comes from the fact that there are 4 matches, 2 mismatches, and 2 indels ($4 + 2 + 2 = 8$).

We will now introduce the following variables:

$$\mu = -\log\left(\frac{q}{k}\right)$$

$$\eta = -\log\left(\frac{r}{k}\right)$$

Finally, we have a statistically significant representation of an alignment score based on probabilites for matches, mismatches, and indels:

$$S = 4 \log\left(\frac{p}{k}\right) + 2 \log\left(\frac{q}{k}\right) + 2 \log\left(\frac{r}{k}\right)$$

$$S = 4 - 2\mu - 2\eta$$

In this framework, we have match score 1, mismatch score $-\mu$, and gap score $-\eta$. We see here that maximizing the alignment score corresponds to determining the most likely alignment under this simple probabilistic model.