7.1 Variations

We would like to obtain an optimal alignment between the entire sequences A and B. As described in the previous lecture, this problem can be interpreted as an automaton over \( D = (\Sigma + \{-\})^2 \). The set of all alignments between A and B can be represented as \( \Lambda(A) \otimes \Lambda(B) \) (where \( \otimes \) denotes the cross product). Additionally, any alignment between A and B is a path from \((i,j)\) to \((g,h)\):

![Path from (i,j) to (g,h)](image)

The scoring set, \( \delta : \Gamma \to \mathbb{R} \), assigns a real number to every possible edge of the path. We would like to find the optimal maximum or minimum \( \theta - \phi \) path. This problem is simplified by the fact that we have an acyclic graph. We can optimize computation of this optimal alignment with respect to time or space (memory).

Suppose \( \delta \) satisfies the following axioms, where \( a \) and \( b \) are characters.

1. \( \delta(a,a) = 0, \forall a \)
2. \( \delta(a,b) = \delta(b,a) \geq 0 \)
3. \( \delta(a,b) \leq \delta(a,c) + \delta(c,b) \)

These three axioms imply that \( \delta \) is a metric over \( \Sigma + \{-\} \). We can extend the concept of \( \delta \) to strings and let \( \delta(A,B) = \) the minimum cost alignment between strings A and B, where A and B are elements of \( \Sigma^* \). Sometimes \( \delta(A,B) \) is referred to as the “edit distance,” which comes from the idea that it represents the minimum number of character edits to get from A to B.

In the context of strings, the following lemma follows.

1. \( \delta(A,A) = 0, \forall A \)
2. \( \delta(A, B) = \delta(B, A) \geq 0 \)
3. \( \delta(A, B) \leq \delta(A, C) + \delta(C, B) \)

Thus \( \delta \) being a metric over the alphabet \( \Sigma + \) \( ' - ' \) implies that \( \delta \) is a metric over \( \Sigma^* \).

### 7.2 Mapping Scores

There are several variations to the general problem of minimum cost alignments: global alignment, searching problem, local alignment variation, and overlap variation.

1. In a global alignment, we compute \( \delta(A, B) \) regardless of whether \( \delta \) is a metric.

2. Searching Problem
   
   If the length of sequence \( A \) were much longer than that of \( B \), \( |A| \gg |B| \), we have a searching problem. As an example, \( A \) could be the human genome and \( B \) could be a small segment of DNA. Given a threshold \( \tau \), the goal would be to find substrings of \( A \) such that \( \delta(A[b, e], B) \leq \tau \), where \( A[b, e] \) denotes the substring of \( A \) starting at index \( b \) (beginning) and ending at \( e \) (end).

3. Local Alignment Variation
   
   This variation is motivated by the fact that usually a small part of an enzyme is responsible for its catalytic activity. Thus if variations of this enzyme maintain this activity, the segment of DNA coding for this active site is conserved. The goal of local alignment variation is to find substrings in \( A \) and \( B \), \( A[b_1, e_1] \) and \( B[b_2, e_2] \), such that \( \delta(A[b_1, e_1], B[b_2, e_2]) \) is optimized. The edit graph would resemble:
4. Overlap Variation

In this variation, we find the best alignment where one prefix and one suffix can be ignored. For example,

```
A

B
```

The core alignment is contiguous, but the “dangling ends” are ignored.

Each one of these variations is explored in the literature, but they are all rather minor variations of the general goal. Many times in practice these definitions are inadequate.

In the searching problem, different matches may result from the same two sequences. For example,

```
i−1

A

i j

B
```

There is an exact match between the two sequences from i to j, a 1-match from i-1 to j, or a 2-match from i-2 to j. One core match may return all possible pairs of matches, resulting in a quadratic number of pairs and a waste of time. The nontrivial question that is suggested is, “what is the concept of a core match”?

Let $\delta$ be a positive scoring scheme. If adding any edge increases the score, a long and perhaps irrelevant match may result. Therefore, $\delta$ must be negatively biased. The Smith-Waterman algorithm is the simplest of this scoring scheme class. It basically computes the log probability that A evolved into B. Because a sum of the logs of numbers is a log of the product of the same numbers, $\delta(p) = \log$ of the product of the change probabilities. Thus computing the score is equivalent to evaluating the probability of alignment by evolution.
Problems with overlap variation occur in cases such as

In both cases, maximizing the score would be good, but may lead to alignments beyond the relevant regions. If we minimize the score, a result of a trivial small match may be found.

An open question occurs with highly repetitive regions, such as:

A: \( \text{ATGATGATGATG} \)

B: \( \text{ATGATGATGATG} \)

If we view this alignment in terms of the edit graph, we would see:
All alignments would be perfect and separated by 3 diagonals. The maximum scoring alignment may not be the correct one:

Possible methods of resolving this issue include small microvariations that account for 2-4% variation in the periodic sequence. The drawback is that the sequencing noise may overwhelm an algorithm looking for these microvariations.

One approach to solving the four variations of the minimum cost alignment is to find the optimal $\theta - \phi$ path. We have a searching matrix and we look for paths whose score is less than or equal to $\tau$. Additionally, we start the alignment at the top of the matrix and end at the bottom. In the context of the automaton, we let every vertex on the top row potentially be a $\theta$ and every vertex on the bottom row potentially be a $\phi$.

In the dynamic programming approach, we optimize $c(u)$, the best path to $u$ from a start state in $\theta$.

$$c(u) = \begin{cases} 
0 & \text{if } u \in \theta \quad \text{(start for free)} \\
\text{opt}(c(w) + \delta(w \to u)) & \text{else}
\end{cases}$$

To obtain the optimal alignment, we simply read off the bottom row and see if some score $\leq \tau$. Thus for each node, we calculate a three way minimum:

If we seek all possible alignments, $\theta$ should be everything. Then we have the opportunity to start at 0 if no good paths connect to a present node. If the scores are negatively biased, so that non-alignments are penalized, occasionally a positive score will accumulate. Viewed in 3-d, this relevant alignment will
appear as a ridge. When the relevant alignment is over the score will rapidly decay. If we set the end of the alignment \( \phi \) to \( V \), we look for the highest peak in the matrix an trace back to get the entire alignment.

Overlap variations will result in a searching matrix where we set \( \theta \) to the top an left sides and \( \phi \) to the bottom and right sides.

Question: Can transitions advance by more than 1 step, for example, can a "GGG" match between the two sequences be more important than three G’s sequentially matching? (i.e., can small string matches be significant?)

Answer: Yes, and there are chaining algorithms to align based on this principle. The question of how to build the best possible chain if certain segments are interesting is answered in a paper by Galeil, Epstein, and Myers. We do not allow chaining of sequences that overlap because that would be too complicated.

### 7.3 Transforming Scores

The minimization and maximization of scores are dual problems. For example, to transform from a maximization to a minimization problem, we simply multiply \( \delta \) by -1 to make all scores negative.

\[
\delta'(\begin{bmatrix} a \\ b \end{bmatrix}) = \delta(\begin{bmatrix} a \\ b \end{bmatrix}) \cdot \sigma + \pi l(\begin{bmatrix} a \\ b \end{bmatrix})
\]

\[
\delta'(e) = \delta(e) \cdot \sigma + \pi l(e)
\]

where
- \( l(e) = \text{length}(e) = 1 \)
- \( A \) projection \( (e) = 1 \) iff \( a \neq - \)
- \( B \) projection \( (e) = 1 \) iff \( b \neq - \)
- col \( (e) = A \text{proj}(e) + B \text{proj}(e) \)

**Lemma.** The score of the derived path,

\[
\delta'(p) = \Sigma \delta'(e) = \Sigma_{e \in p} \delta(e) \cdot \sigma + \pi l(e)
\]

\[
= \sigma \Sigma_{e \in p} \delta(e) + \pi \Sigma_{e \in p} l(e)
\]

\[
= \sigma \delta(p) + \pi \text{len}(p)
\]

Suppose the path goes from \( (i,j) \rightarrow (g,h) \). If we set the length of \( e \) to \( A \text{proj}(e) \), we would simply substitute \( g-i \) for \( l(p) \). Similarly, if we set length of \( e \) to \( B \text{proj}(e) \), we would substitute \( h-j \) for \( l(p) \). Setting the length of \( e \) to \( \text{col}(e) \) contributes nothing additional.

Give a metric \( \delta \) and a threshold \( \tau > 0 \), we want to find all local alignments such that \( \frac{\delta(p)}{l(p)} < \tau \). This action simply normalizes the score over the length of the alignment, e.g., looking for less than 5% difference in a local alignment. The equation corresponding to this situation is given by

\[
\delta'(p) = \delta(p) \cdot (-1) + \tau l(p) > 0
\]

Another example of how transforming scores may be interesting is the following.
\[
\delta_{lcs}(a, b) = \begin{cases} 
1 & \text{if } a = b \\
0 & \text{else}
\end{cases}
\]

\[
\delta'(a, b) = \frac{1}{2} \col(a) - \delta(a, b)
\]

\[
\delta'(a, b) = \begin{cases} 
0 & \text{if } a = b \\
\frac{1}{2} & \text{if } a = - \text{ or } b = - \\
1 & \text{else}
\end{cases}
\]

\[
\delta'_{\text{min}}(A, B) = \min_p \delta'(p) = \min_p \frac{1}{2} \col(p) - \delta_{lcs}(p)
\]

A proj is M, B proj is N. Therefore, \(\col(p) = (M+N)\).

\[
\delta'_{\text{min}}(A, B) = \min_p \delta'(p) = \frac{1}{2} (M + N) + \min_p (-\delta_{lcs}(p))
\]

\[
= \frac{1}{2} (M + N) - \max_p (\delta_{lcs}(p))
\]

\[
= \frac{1}{2} (M + N) - \delta_{lcs}(A, B)
\]

Thus we have shown:

\[
\delta'(A, B) + \delta_{lcs}(A, B) = \frac{1}{2} (M + N)
\]

which indicates that the longest common subsequence and the score of the derived path are duals.