1 Brief Review

A few weeks ago we discussed pairwise alignment. Our goal was to find the best alignment of two sequences. In order to achieve this goal we assigned scores to the labels - match, mismatch and indels. Upon this notation we developed a dynamic programming algorithm whose objective was to output the alignment with the highest score. Nevertheless, this doesn’t provide us with enough biological knowledge. We need the ability to perform this analysis on a larger number of sequences. This is why today we’ll discuss Multiple Sequence Alignment and the methods that may solve this problem.

2 Problem Definition

We are given $k$ sequences $S_1, S_2, \ldots, S_k$. We wish to find an alignment of $S_1, S_2, \ldots, S_k$ having the highest score.

$S_1 : C T A A - C$
$S_2 : - - - A G C$
$S_3 : - - - A G -$
$\ldots$
$S_k : C T A - G -$

Figure 1: A possible alignment of $S_1, \ldots, S_k$

How shall we do it?

3 Implementation Methods

3.1 Guided Alignment

Choose a specific reference sequence, for instance $S_1$ and align all other sequences against it.

$$\sum_{j=2}^{k} \text{Score}(S_1, S_j)$$ (1)
• Advantages: By comparing the reference sequence to all other sequences we may obtain valuable information on it.

• Disadvantages: We don’t obtain any information about the comparison between all other sequences $S_2, \ldots, S_k$. This causes a great loss of data as illustrated in the following example:

**Example 3.1** All sequences $S_2, \ldots, S_k$ have on the position $i$ a C and only the reference sequence $S_1$ has on this position a G.

If we compared all sequences to one another, we would infer that $i$ is a conserved position. But, according to this procedure $i$ isn’t conserved.

Conclusion: This method isn’t good enough for us.

### 3.2 All against all alignment

Align every pair of sequences one against the other.

\[
\sum_{i \neq j} \text{Score}(S_i, S_j)
\]

This procedure is problematic since the sequences aren’t necessarily consistent.

For instance, assume you have the best alignment of $S_1$ and $S_2$ and the one of $S_1$ and $S_3$. You would like to align them all one against the other in this manner. Though this alignment is optimal for $S_1$ it is not necessarily the best one for $S_2$ and $S_3$.

Position $i$ correlate to position $j$ in $S_2$ and to position $k$ in $S_3$. However, position $j$ in $S_2$ doesn’t match position $k$ in $S_3$. This is illustrated in the following figure:

![Figure 2: $S_2$ and $S_3$ aren’t consistent](image)

There are some solutions to this problem. However, we’ll not discuss them.

### 3.3 All pairs algorithm

Let’s think of the problem thoroughly. What we truly want is an alignment of $S_1, \ldots, S_k$ which will output the maximum score for each position.

Which operations should be done to produce this score?

- For each position $i$ calculate all the scores on this position.
- Sum the scores of all the positions.

The definition of the score for MSA:
3.3.1 Score for MSA - All Pairs

Denote \( S \) a matrix that contains an alignment of \( S_1, \ldots, S_k \).

\( S[i, j] \) - The letter at the \( i \)'th row at the \( j \)'th column.

Definition 3.2 \( \text{Score}(MSA) = \sum_{i} \left( \sum_{j \neq k} \sigma(S[k, i], S[j, i]) \right) \)

This score is called All Pairs.

Notice that this is a combinatorial problem with exponential number of possibilities.

How shall we make this problem solvable? As always with this kind of problems, we’ll develop a dynamic programming algorithm. It should resemble the algorithm of the pairwise alignment we previously learned. Only this time the table of the scores should be \( k \)-dimensional and not two-dimensional. The algorithm should compute the scores of all \( \binom{k}{2} \) alignments and store them in the \( k \)-dimensional table.

3.3.2 Table definition

MSA is defined as a matrix of scores:

Definition 3.3 \( V[i_1, i_2, \ldots, i_k] \) - Score of best alignment of prefixes of sequences \( S_1(1, \ldots, i_1), \ldots, S_k(1, \ldots, i_k) \).

Denote \( X_{i_j} \) as the letter on the \( j \)'th sequence in the \( i \)'th position.

The dynamic programming algorithm is:

\[
V[i_1, i_2, \ldots, i_k] = \max \begin{cases}
V[i_1 - 1, i_2 - 1, \ldots, i_k - 1] + \sigma(X_{i_1}, X_{i_2}, \ldots, X_{i_k}) \\
V[i_1, i_2 - 1, \ldots, i_k - 1] + \sigma(-, X_{i_2}, \ldots, X_{i_k}) \\
V[i_1 - 1, i_2, \ldots, i_k - 1] + \sigma(X_{i_1}, -, \ldots, X_{i_k}) \\
\vdots \\
V[i_1 - 1, i_2 - 1, \ldots, i_k - 1] + \sigma(X_{i_1}, X_{i_2}, \ldots, -) \\
V[i_1, i_2, i_3 - 1, \ldots, i_k - 1] + \sigma(-, -, \ldots, X_{i_k}) \\
\vdots \\
V[i_1, i_2, \ldots, i_{k-1} - 1, i_k] + \sigma(-, X_{i_2}, \ldots, -) \\
\vdots 
\end{cases}
\]  

(2)

This is a precise algorithm that completely solves the problem. Yet, it has a fatal problem - it has a very high time complexity.

To get a glimpse of the time complexity, notice that a calculation of a single position in the matrix sums up to \( 2^k \). It is explained in the following passage:

For each position in the matrix the algorithm computes the scores of \( k \) assignments, each of which is either a letter or a gap. On the whole it takes \( 2^k \) computations.

Thus it is practical only for small values of \( k \).

So we need to think of another solution. Let’s review additional methods.

3.4 Finding the best common ancestor

Assume the \( k \) sequences diverged from a common evolutionary ancestor (not necessarily at the same period) as shown in Figure 3. Discovering their best common ancestor may simplify our problem.
First, we must think of a technique to achieve the best common ancestor. Suggested solution:

$$\sum_j \sigma(X_i, S[j, i])$$

Yet, we don’t know $X$ and want the highest score. So, let’s add $\max$ to the equation:

$$\max_{X_i} \sum_j \sigma(X_i, S[j, i])$$

This method is problematic due to two reasons:

1. This algorithm is computationally complex - it is exponential in $k$.
2. It doesn’t take into account gaps in the sequences.

Till now we disregarded the fact that certain positions in the sequence have different probabilities. Regarding this feature would provide another aspect to this problem.

Recall that a few weeks ago we were first introduced to PSI-BLAST. PSI-BLAST is an algorithm that computes the best score function for each position.

In the next section we’ll exploit some of PSI-BLAST’s features.

## 4 Generating sequences by using a finite Automaton

Describe the properties of each position in the MSA by probabilities. Let $P_1, P_2, \ldots, P_n$, the distributions in positions $1, \ldots, n$ respectively.

First, here is the skeleton of a finite automaton:

![Diagram of a finite automaton with match states only](image)

The automaton consists of a series of nodes, each of which corresponds to a position in the alignment that it describes. It has an initial and ending states. Each node except start and end can be in a single state - a match state.

For each state $M_i$ sample a character from distribution $P_i$ and write it in the output sequence. When the automaton finish running it will output a new sequence of characters.

This automaton produces sequences merely according to the distributions $P_1, \ldots, P_n$.

We would expect the product sequences to resemble the sequences we wish to align.

Notice that this automaton doesn’t include gaps. How shall we represent them?
• Option 1: For each gap in position i draw an arrow that would skip the i’th state (see Figure 5). Since there is a variable length of gaps this method adds an exceedingly large number of edges and thereby complicates the structure of the automaton.
Let’s try something simpler:

• Option 2: Add states $D_1, D_2, \ldots, D_n$ that denote gaps. These states are silent states as the automaton doesn’t write a thing when it traverses them.

How shall we represent insertions?
Add states $I_1, \ldots, I_n$ as insertion states to the automaton.
The final automaton: see Figure 7

A run of the automaton produces a sequence of states, for instance:

**Example 4.1** start $M_1$ $I_1$ $I_1$ $M_2$ $D_3$ $D_4$ $M_5$ \ldots
Some attributes of the automaton:

- **S** - States
- **τs** - Transition probabilities
- **Ps** - Emission probabilities

**translation:**

τs - The probability to traverse from one state to another.

Ps - In a given state, this is the probability to write a specific character.

Each state has both transition probability and emission probability.

Now that we are acquainted with this special kind of automaton it’s time to call it with its name:

### 5 Hidden Markov Model (HMM)

This is a very useful tool for analyzing sequences.

It describes an automaton that generates sequences based on predefined distributions.

- **H** - The states of the automaton, they are hidden from the observer.
- **X** - The observations:
  - The letter which is written on each stage by the automaton. This includes silent states denoted by ε.

Formal definition of the automaton:

**Definition 5.1** A HMM automaton is represented by a 4-tuple \((S, \sum, \tau, P)\) where:

- **S** - A set of states.
- **\(\sum\)** - The output alphabet
- **\(\tau\)** - Transition probabilities
- **\(P\)** - Emission probabilities

Here is a possible run of the automaton:

<table>
<thead>
<tr>
<th>t(time):0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>H: start</td>
<td>M1</td>
<td>I1</td>
<td>I1</td>
<td>M2</td>
<td>D3</td>
<td>end</td>
</tr>
<tr>
<td>X:</td>
<td>C</td>
<td>T</td>
<td>G</td>
<td>A</td>
<td>ε</td>
<td></td>
</tr>
</tbody>
</table>

What is the probability to get this sequence?

\[
P\left(\vec{X}, \vec{H}\right) = \tau_{\text{start}}(M_1) \cdot \tau_{M_1}(I_1) \cdot \tau_{I_1}(I_1) \cdot \tau_{I_1}(M_2) \cdot \tau_{M_2}(D_3) \cdot \tau_{D_3}(\text{end}) \cdot P_{M_1}(C) \cdot P_{I_1}(T) \cdot P_{I_1}(G) \cdot P_{M_2}(A) \cdot P_{D_3}(\varepsilon)
\]

The hidden \((H)\) act like Markov’s chain.

The probability that this model would produce a sequence \(S\) is:

\[
P(S) = P(X|\text{seq}(X) = S) = \sum_H \sum_X P(X,H)
\]

\(\text{seq}(X)\) - An operator which deletes \(\varepsilon\) from the sequence \(X\).

The number of possibilities to write a specific sequence of characters is exceedingly large. In the next lesson we’ll learn how to solve this problem efficiently with dynamic programming, and how to estimate these probabilities.