Recitation 2 Notes: Sequence Alignment
Outline

- Review of generative motif model from recitation 1
  See recitation 1 notes for details (we derived the numerator $P(s)$ by marginalizing the joint distribution and introducing priors $P(B)$ and $P(M)$).

- Variations on the alignment theme
  Global / Local / Semi-global alignment review

- Affine gap penalties
  Augmenting the state-space

- Generalized gap penalties

- Multiple alignment introduction
  Optimally aligning $D$ sequences of length $n$ takes time $O(n^D)$... we extend the matrix for 2 sequences up to a $D$-dimensional matrix and do the same algorithm (with appropriate generalized scoring functions for blocks of bases and gaps).
Semi-global alignment and overlap detection
Different types of overlaps

• Application: genome assembly
  – Search for consecutive sequence segments
  – Boundaries of compared sequences not fully known
A variant of the basic algorithm:

- Allow unlimited # of gaps in the beginning and end
- No penalty for end gaps
The Overlap Detection variant

Changes:

1. **Initialization**
   - For all \( i, j \),
     - \( F(i, 0) = 0 \)
     - \( F(0, j) = 0 \)

1. **Termination**
   - \( F_{OPT} = \max \{ \max_i F(i, N), \max_j F(M, j) \} \)
Local Alignment
Intro to Local Alignments

• Statement of the problem
  – A local alignment of strings $s$ and $t$ is an alignment of a substring of $s$ with a substring of $t$

• Why local alignments?
  – Small domains of a gene may be only conserved portions
  – Looking for a small gene in a large chromosome (search)
  – Large segments often undergo rearrangements
The local alignment problem

Given two strings \( x = x_1 \ldots x_M, \)
\[ y = y_1 \ldots y_N \]

Find substrings \( x', y' \) whose similarity (optimal global alignment value) is maximum

e.g. \( x = \textcolor{red}{\text{aaaacc}} \textcolor{blue}{\text{ccccgggg}} \textcolor{red}{\text{ggg}} \)
\[ y = \textcolor{red}{\text{ttgc}} \textcolor{blue}{\text{cctgggg}} \textcolor{red}{\text{aaccaacc}} \]

How can we use dynamic programming for local alignment?
Global Alignment

Initialization:
- Top left: 0

Update Rule:
\[ A(i,j) = \max\{ \]
- \[ A(i-1, j) - 2 \]
- \[ A(i, j-1) - 2 \]
- \[ A(i-1, j-1) \pm 1 \] \[
\}\]

Termination:
- Bottom right
Local Alignment

**Initialization:**
- Top left: 0

**Update Rule:**
\[
A(i,j) = \max\{ \\
A(i-1, j) - 2, \\
A(i, j-1) - 2, \\
A(i-1, j-1) \pm 1, \\
0 \}
\]

**Termination:**
- Anywhere

- **Initialization**:
  - Top left: 0

- **Update Rule**:
  \[
  A(i,j)=\max\{ \\
  A(i-1, j) - 2, \\
  A(i , j-1) - 2, \\
  A(i-1 , j-1) \pm 1, \\
  0 \}
  \]

- **Termination**:
  - Anywhere
Local Alignment issues

• Resolving ambiguities
  – When following arrows back, one can stop at any of the zero entries. Only stop when no arrow leaves. Longest.
Best local alignment vs. all local alignments

**Termination:**

1. If we want the **best** local alignment...

   \[ F_{\text{OPT}} = \max_{i,j} F(i, j) \]

1. If we want all local alignments scoring > t

   For all i, j find \( F(i, j) > t \), and trace back
**Global Alignment** vs. **Local alignment**

**Needleman-Wunsch algorithm**

- **Initialization**: $F(0, 0) = 0$
- **Iteration**: 
  
  $F(i, j) = \max \begin{cases} 
  F(i-1, j) - d \\
  F(i, j-1) - d \\
  F(i-1, j-1) + s(x_i, y_j) 
  \end{cases}$

- **Termination**: Bottom right

**Smith-Waterman algorithm**

- **Initialization**: $F(0, j) = F(i, 0) = 0$
- **Iteration**: 
  
  $F(i, j) = \max \begin{cases} 
  F(i-1, j) - d \\
  F(i, j-1) - d \\
  F(i-1, j-1) + s(x_i, y_j) 
  \end{cases}$

- **Termination**: Anywhere
DP Algorithm Variations

Global Alignment

S

-   -2   -4   -6
A   0     1     -1
A   -2   1     -1
G   -4   -1   -2
C   -6   -1   0

Semi-Global Alignment

S

-   -2   -4   -6
A   0     1     -1
A   0     1     -2
G   0     -1   2
C   0     -1   0

Local Alignment

S

-   0     0     0     0
A   0     1     0     0
A   0     1     0     0
G   0     0     2     0
A   0     1     0     1
Affine gap penalties

• Linear gap models don't adequately reflect the sources of biological variation
• Oftentimes the generation of a gap is a single event, and the length follows a broad distribution
• Therefore, we should penalize multiple gaps more than penalizing longer gaps
Affine gap penalty

penalty(n) = gap + (n – 1) * extend

- **Gap open**
  - To introduce the first gap, cost of introducing a DNA break

- **Gap extend**
  - Larger segments more costly than smaller ones
  - Linear cost increment for increasing number of gaps

- **How can we compute this using dynamic programming?**
  - Achieve O(m*n) running time and space
Additional Matrices

• A “greedy” extension of the standard global algorithm will not work (e.g. looking at previous traceback pointers when making each choice)
  – Sometimes we should make a locally suboptimal choice (open a gap) because it will pay off in the future (resulting in a better-scoring longer single gap instead of multiple smaller gaps)

• The amount of state needed increases
  – In scoring a single entry in our matrix, we need to remember an extra piece of information
    • Are we continuing a gap in s? (if not, start is more expensive)
    • Are we continuing a gap in t? (if not, start is more expensive)
    • Are we continuing from a match between s(i) and t(j)?

• Dynamic programming framework
  – We encode this information in three different matrices
  – For each element (i,j) we use three variables
    • A(i,j): best alignment of s[1..i] & t[1..j] that aligns s[i] with t[j]
    • B(i,j): best alignment of s[1..i] & t[1..j] that aligns gap with t[j]
    • C(i,j): best alignment of s[1..i] & t[1..j] that aligns s[i] with gap
Update rules

When $s[j]$ and $t[j]$ are aligned (diagonal movement from all matrices to $A$):

$$A(i, j) = \max \begin{cases} 
A(i - 1, j - 1) + s(x_i, y_j) \\
B(i - 1, j - 1) + s(x_i, y_j) \\
C(i - 1, j - 1) + s(x_i, y_j)
\end{cases}$$

When $t[j]$ aligns with a gap in $s$ (vertical movement from all matrices to $B$):

$$B(i, j) = \max \begin{cases} 
A(i, j - 1) + \text{gap} + \text{extend} \\
B(i, j - 1) + \text{extend} \\
C(i, j - 1) + \text{gap} + \text{extend}
\end{cases}$$

Start a gap in $s$, extending a gap in $s$, stopping a gap in $t$, and starting one in $s$

When $s[i]$ aligns with a gap in $t$ (horizontal movement from all matrices to $C$):

$$C(i, j) = \max \begin{cases} 
A(i - 1, j) + \text{gap} + \text{extend} \\
B(i - 1, j) + \text{gap} + \text{extend} \\
C(i - 1, j) + \text{extend}
\end{cases}$$

Find maximum over all three arrays $\max(a[m,n], b[m,n], c[m,n])$.
Follow arrows back, skipping from matrix to matrix
General gap dynamic programming

- What if we want an arbitrary gap penalty function? Need to consider all possible gap lengths when computing the value for each cell.

**Initialization:** same

**Iteration:**

\[
F(i, j) = \max_{k=0\ldots i-1} F(k, j) - \text{penalty}(i-k) \]
\[
\max_{k=0\ldots j-1} F(i, k) - \text{penalty}(j-k) \]

**Termination:** same

**Running Time:** \(O(NM^*(N+M))\) (cubic)

**Space:** \(O(NM)\)

Can we do better?
To generalize a little…

… think of how you would compute optimal alignment with this gap function

\[ \gamma(n) \]

….in time \( O(MN) \)