Semiglobal Alignment

- We score alignments ignoring start and end gaps.
- Start gaps occur before the first character in a sequence.
- End gaps occur after the last character in a sequence.

\[
\begin{align*}
\text{CAG} & \rightarrow \text{CTTGGATTCTCGG} \\
\text{start gaps} & \rightarrow \text{-----}
\end{align*}
\]

- Score = +1(6) -1(1) -2(1) -11(0) = 3
- This is not the best global alignment between the two sequences (-12 for global optimal against -19 if scored globally)

Why this variation?

- Maybe it is OK to have unlimited number of gaps at the ends, e.g. detecting significant overlap
- Different possible overlaps
Modifying Needleman-Wunsch

- Ignoring start gaps
  - $A(i,0) = 0$
  - $A(0,j) = 0$

- Ignoring end gaps

$$A^{\text{opt}} = \max \begin{cases} 
\max A(m,j) \\
\max A(i,n) 
\end{cases}$$

Explanation

- Initializing the first row and first column to zeros eliminates any starting gap penalty.

- An alignment with end gaps in $x$ aligns $x$ with a prefix of $y$.

  ...xxxxx------
  --yyyyyyyyyy

Therefore, the $\max A(m,j)$ (last row) gives the maximum score for such an alignment.

- Same reasoning for $y$.

Generalization

<table>
<thead>
<tr>
<th>Place where gaps are not penalized</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start of $x$</td>
<td>Initialize first row to zeros</td>
</tr>
<tr>
<td>End of $x$</td>
<td>Look for max in last row</td>
</tr>
<tr>
<td>Start of $y$</td>
<td>Initialize first column to zeros</td>
</tr>
<tr>
<td>End of $y$</td>
<td>Look for max in last column</td>
</tr>
</tbody>
</table>
Local Alignment

- Given two sequences x and y, find a highest score alignment between a substring of x and a substring of y
- Example:

<table>
<thead>
<tr>
<th>Global scores -1</th>
<th>GGAGTA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GC-GTC</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Local scores 2</th>
<th>GAGTA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GGCT C</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Local alignment</th>
<th>GGA GT A</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GC GT C</td>
</tr>
</tbody>
</table>

Why Local Alignments?

- Comparative genomics:
  - Genes are shuffled in different genomes

- Finding preserved sequences
  - Different proteins have preserved repetitive patterns

Greedy Way

- \(O(m^2)\) substrings of x
- \(O(n^2)\) substrings of y
- \(O(m^2n^2)\) algorithm using Needleman-Wunsch on each pair
Facts

- The two substrings with the best score must be matched at both ends (why?)
- The score of the best local alignment has to be positive (why?)
- More generally: The best local alignment cannot start or end with an alignment with negative score (why?)

Modifying Needleman-Wunsch

- Same as before, but
  - \( A(i,j) \) is now best score for aligning a suffix of \( x_1...x_i \) and a suffix of \( y_1...y_j \)
  - Is the update rule still valid? Almost...
  - Negative entries in the \( A \) matrix are meaningless
    if \( A(i,j) < 0 \), replace it with 0
    i.e. starting over from \( A(i+1,j+1) \), \( A(i,j+1) \), and \( A(i+1,j) \).

Illustration

The best local alignment cannot start with a suffix of \( x_1...x_i \) aligned with a suffix of \( y_1...y_j \).
Smith-Waterman

- Initialization
  - $A(0,0) = 0$
  - $A(i,0) = 0$

- Main iteration
  $A(i,j) = \max\left\{\begin{align*}
  &A(i-1,j-1) + s(x_i,y_j), \\
  &A(i-1,j) - d, \\
  &A(i,j-1) - d, \\
  &0
  \end{align*}\right.$

- Termination
  $A^{opt} = \max_{i,j} A(i,j)$

Example

<table>
<thead>
<tr>
<th>G</th>
<th>C</th>
<th>G</th>
<th>T</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Trace back from $\max_{i,j} A(i,j)$ until you hit a 0

Saving Space
(for Needleman-Wunsch)

- These basic algorithms require $O(mn)$ time and $O(mn)$ space.

- No algorithm is known that uses asymptotically less time and has the same generality.

- It is possible to improve space complexity from $O(mn)$ to $O(m+n)$. 

First Attempt

- We only need to keep one row of the matrix $A$ at any given time.

Therefore, we only need $O(m+n)$ space to compute the optimal score.

- What about producing the alignment itself?

Saving Space (cont.)

- We know how to compute the optimal score in linear space.
  - In fact, we can compute the best scores between a given prefix of $x$ and all prefixes of $y$ (How?)

- The hard part is to obtain the optimal alignment itself in linear space
  - the previous algorithms depend on the whole matrix to do this job.
  - Use a Divide and Conquer to remove this difficulty.

Key Idea

- Consider $x_i$ in an optimal alignment of $x$ and $y$
  - $x_i$ is aligned with $y_j$ for $1 \leq j \leq n$ OR
  - $x_i$ is aligned to a gap between $y_j$ and $y_{j+1}$ for $0 \leq j \leq n$

  - $y_1$ - $y_2$ - $y_3$ - $y_4$ - $y_5$ - $\cdots$ - $y_n$

  2$n+1$ possibilities
Key Idea (cont.)

Given i, guess what is matched to x_i in an optimal alignment by finding the best alignments that match x_i to each of the 2n+1 positions.

\[
\begin{align*}
\text{OPT} \begin{bmatrix} x_{i-1} \\ y_{j-1} \end{bmatrix} & \quad x_i \\
\text{OPT} \begin{bmatrix} x_{i+1} \\ y_{j+1} \end{bmatrix}
\end{align*}
\]

Key Idea (cont.)

- Compute the best scores between
  - \( x_i \) \( y_{k+1} \), and all prefixes of \( y \)
  - \( x_{k+1} \) \( y_{1} \) and all suffixes of \( y \) (the reverse)

\[
\begin{align*}
x_{i} & \quad \hat{j} \\
x_{i+1} & \quad y_{j+1}
\end{align*}
\]

- Find \( k \) that maximizes
  \[
  \max \left\{ A(m/2 - 1, k - 1) + B(m/2 + 1, n - k) \right\}
  \]

Divide and Conquer

- Align \( x_{m/2} \)

- Recursively do the same for the two remaining chunks of the alignment.
Analysis

- $T(m,n) =$ time to align $x_1...x_m$ and $y_1...y_n$
- $T(m,n) = c.mn + T(m/2, k) + T(m/2, n - k)$
- Assume $T(m,n) \leq 2c.mn$, verify by substitution:
  - $T(m,n) \leq c.mn + 2c.m/2.k + 2c.m/2.(n - k)$
  - $= c.mn + 2c.m/2(k + n - k)$
  - $= c.mn + 2c.m/2.n = c.mn + c.mn$
  - $= 2c.mn$