Biological Sequence Comparison

- **Types of biological sequences:**
  1. Nucleic acid sequences over 4-letter alphabet (DNA/RNA sequences)
  2. Amino acid sequences over 20-letter alphabet (protein sequences)

- **Why sequence comparison?**
  - High sequence similarity usually implies functional or structural similarity.

- **Measurements of sequence comparison:**
  - edit distance (difference) and alignment (similarity)
Edit Distance of Two Strings

- **Edit distance**: the minimum number of edit operations needed to transform \( S_1 \) to \( S_2 \)
- **Three kinds of edit operations**:
  1. Substitution
  2. Insertion
  3. Deletion

**Example**: \( S_1 = \text{VINTNER}, \ S_2 = \text{WRITERS} \)

\[
\begin{array}{ccccccc}
S_1: & V & - & I & N & T & N & E & R & - \\
\text{substitution} & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\
S_2: & W & R & I & - & T & - & E & R & S
\end{array}
\]

Alignment of Two Strings

- **Alignment of strings** \( S_1 \) and \( S_2 \):
  a pair of strings \((S'_1, S'_2)\) obtained by insertion of spaces in \( S_1 \) and \( S_2 \) such that
  1. \(|S'_1| = |S'_2|\), and
  2. for each \( i \), \( S'_1[i] \) is aligned with \( S'_2[i] \) and either \( S'_1[i] \) or \( S'_2[i] \) is not a space

**Example**: \( S_1 = \text{VINTNER}, \ S_2 = \text{WRITERS} \)

\[
\begin{array}{ccccccc}
S_1: & V & - & & & & & & & - \\
S_2: & W & R & I & - & T & - & E & R & S \\
\boxed{1}: & \text{An alignment} \\
S_1: & V & & & & & & & - \\
S_2: & W & R & I & - & T & - & E & R & S \\
\boxed{2}: & \text{Not an alignment}
\end{array}
\]

By R.C.T. Lee and C.L. Lu
**Similarity of Two Strings**

- \( \sigma(x, y) \): score of aligning char. \( x \) with char. \( y \)
- If \( x = y \), then \( \sigma(x, y) \geq 0 \); otherwise, \( \sigma(x, y) < 0 \)
- Emphasize matches, penalize mismatches or inserted spaces

- \( \sigma(S'_1, S'_2) \): score of an alignment \( (S'_1, S'_2) \)
  \[
  \sigma(S'_1, S'_2) = \sum_{1 \leq i \leq |S'_1|} \sigma(S'_1[i], S'_2[i])
  \]

- \( \text{Sim}(S_1, S_2) \): similarity of two strings \( S_1 \) and \( S_2 \)
  \[
  \text{Sim}(S_1, S_2) = \max \{ \sigma(S'_1, S'_2) \} \text{ all align. } (S'_1, S'_2)
  \]

---

**Two-String Alignment**

- **Global alignment:**
  - To find the optimal alignment between two entire strings \( S_1 \) and \( S_2 \)

- **Local alignment:**
  - To find the optimal alignment between a substring of \( S_1 \) with a substring of \( S_2 \)

- **Semiglobal alignment:**
  - To find the optimal alignment between suffix (prefix) of \( S_1 \) with prefix (suffix) of \( S_2 \)
Global Alignment Problem

- Global alignment problem:
  - To find the best alignment between two strings $S_1$ and $S_2$

1. Exhaustive method:
   - Generate all possible alignments and then pick the best one (# of alignments is exponential)
   - **Time-complexity**: exponential time

2. Dynamic programming:
   - Proposed by Needleman & Wunsch [NW70]
   - **Time-complexity**: $\mathcal{O}(|S_1| \times |S_2|)$

---

Needleman & Wunsch Method

- $A(i, j)$: score of an optimal alignment of $S_1[1, i]$ and $S_2[1, j]$ (i.e., $Sim(S_1[1, i], S_2[1, j])$)

- $Sim(S_1, S_2) = A(|S_1|, |S_2|)$

- How to compute $A(i, j)$?

  1. $S_1[1, i - 1]$ $S_1[i]$, $S_2[1, j - 1]$ $S_2[j]$

     $A(i - 1, j - 1) + \sigma(i, j)$

  2. $S_1[1, i - 1]$ $S_1[i]$, $S_2[1, j]$ $-$

     $A(i - 1, j) + \sigma(i, -)$

  3. $S_1[1, i]$ $-$, $S_2[1, j - 1]$ $S_2[j]$

     $A(i, j - 1) + \sigma(-, j)$
Recursive Function

\[ A(i, j) = \max \left\{ A(i - 1, j - 1) + \sigma(i, j), A(i - 1, j) + \sigma(i, -), A(i, j - 1) + \sigma(-, j) \right\} \]

Computation of Optimal Score
**Backtracking: Global Alignment**

```
By R.C.T. Lee and C.L. Lu
```

```
Optimal Alignment:
- C T L
- R C T E
-2 1 1 1
```

**Longest Common Subsequence**

- **Subsequence of string** $S$: $S[i_1]S[i_2] \cdots S[i_k]$ with $i_1 < i_2 < \cdots < i_k$
- **Example**: LCS is a subsequence of LongestCommonSubsequence.

- **Longest common subsequence problem**:
  Given two strings $S_1$ and $S_2$, find the longest common subsequence of $S_1$ and $S_2$

- **Equal to the global alignment problem with**
  match = 1, mismatch = 0, space = 0

- **Ex**: $S'_1: V - I N T N E R -$
  $S'_2: W R I - T - E R S$

```
Local Alignment Problem

- Local alignment problem:
  - To find the best alignment between a substring of $S_1$ and a substring of $S_2$

- Exhaustive method:
  - Compute the global alignment for each pair of substrings and then pick the best one
  - **Time-complexity:** $O(|S_1|^3 \times |S_2|^3)$

- Dynamic programming:
  - Proposed by Smith & Waterman [SW81]
  - **Time-complexity:** $O(|S_1| \times |S_2|)$

Smith-Waterman Method

- $A(i, j)$: the highest alignment score of a suffix of $S_1[1, i]$ and a suffix of $S_2[1, j]$

1. $S_1[1, i - 1]$ $S_1[i]$ $A(i - 1, j - 1) + \sigma(i, j)$
2. $S_2[1, j - 1]$ $S_2[j]$ $A(i - 1, j) + \sigma(i, -)$
3. $S_1[1, i - 1]$ $S_1[i]$ $S_2[1, j]$ $A(i, j - 1) + \sigma(-, j)$
4. $S_1[1, i - 1]$ $S_1[i]$ $S_2[1, j - 1]$ $S_2[j]$ $0$
### Recursive Function

\[ A(i, j) = \max \begin{cases} 
A(i - 1, j - 1) + \sigma(i, j) \\
A(i - 1, j) + \sigma(i, -) \\
A(i, j - 1) + \sigma(-, j) \\
0 
\end{cases} \]

### Computation of Optimal Score

\[ \begin{array}{cccccc}
R & C & T & E \\
0 & 0 & 0 & 0 & 0 \\
-2 & -2 & -2 & -2 & -2 \\
-2 & -2 & -2 & -2 & -2 \\
-2 & -2 & -2 & -2 & -2 \\
\end{array} \]

- **Match:** 1
- **Mismatch:** -1
- **Gap:** -2

By R.C.T. Lee and C.L. Lu
**Backtracking: Local Alignment**

- Match: 1
- Mismatch: -1
- Gap: -2

Optimal Local Alignment:

```
C T
C T
```

**Semiglobal Alignment**

- To find the optimal alignment between suffix (prefix) of $S_1$ with prefix (suffix) of $S_2$
- End space:

```
C A G C A C T T G G A T T C T C G G
- - - C A G C G T G G - - - - - - - -
```

End Space: End Space

**Sequence Comparison: Two Sequence Alignment**  p.17
Semiglobal Alignment

Application to assembly of shotgun sequences:

Where Spaces Are Not Charged?

1. Beginning of first sequence $S_1$
2. End of the first sequence $S_1$
3. Beginning of the second sequence $S'_1$
4. End of the second sequence $S'_1$
Ignoring Final Spaces in $S'_1$

Equal to find the optimal alignment between $S_1$ and a prefix of $S_2$

$Sim(S_1, S_2) = \max\{A(|S_1|, j) : 1 \leq j \leq |S_2|\}$
2 Backtracking: Semiglobal

![Diagram of a semiglobal alignment matrix with arrows indicating the optimal alignment path.]

By R.C.T. Lee and C.L. Lu

1 Ignoring Initial Spaces in $S'_1$

- Equal to find the optimal alignment between $S_1$ and a suffix of $S_2$
2 Computation of Optimal Score

By R.C.T. Lee and C.L. Lu

1 Backtracking: Semiglobal

By R.C.T. Lee and C.L. Lu
Summary: Semiglobal Alignment

<table>
<thead>
<tr>
<th>Where spaces are not charged</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>❶ Beginning of 1st sequence</td>
<td>Initialize first row</td>
</tr>
<tr>
<td></td>
<td>with zeros</td>
</tr>
<tr>
<td>❷ End of 1st sequence</td>
<td>Look for maximum</td>
</tr>
<tr>
<td></td>
<td>in last row</td>
</tr>
<tr>
<td>❸ Beginning of 2nd sequence</td>
<td>Initialize first column</td>
</tr>
<tr>
<td></td>
<td>with zeros</td>
</tr>
<tr>
<td>❹ End of 2nd sequence</td>
<td>Look for maximum</td>
</tr>
<tr>
<td></td>
<td>in last column</td>
</tr>
</tbody>
</table>

Application: Semiglobal Alignment

Inexact matching problem: Given a text $T$, a pattern $P$ and a constant $\delta > 0$, determine if there is a substring $T'$ in $T$ such that the optimal alignment of $P$ to $T'$ has score at least $\delta$?
Gaps

- **Gap**: a consecutive number of spaces
- **Why gaps?**
  - Mutation might insert/delete several residues at once.
  - Match cDNA (without intron) to genomic DNA (with exons and introns)

![Diagram showing DNA and cDNA with exons and introns](image)

Penalties of a Gap with $k$ Spaces

1. **Linear gap penalty**: $P_l(k) = k \times w_s$
   - $w_s$ (the cost of a space) = constant
   - **Time-complexity**: $O(mn)$
2. **Affine gap penalty**: $P_a(k) = w_g + k \times w_s$
   - $w_g$ (the cost of a gap) = constant
   - **Time-complexity**: $O(mn)$
3. **Convex gap penalty**: $P_c(k) = w_g + \log k$
   - **Time-complexity**: $O(mn \log n)$
4. **General gap penalty**: $P_g(k)$: any function
   - **Time-complexity**: $O(m^2n + mn^2)$
**Blocks**

(1) Two aligned chars

(2) A maximal series of consecutive chars in $S_2$ aligned with spaces in $S_1$

(3) A maximal series of consecutive chars in $S_1$ aligned with spaces in $S_2$

$$S_1 \begin{array}{|c|c|c|c|c|c|}
\hline
A & A & C & - & - & C \\
\hline
S_2 \begin{array}{|c|c|c|c|c|c|}
\hline
A & C & T & A & C & C & - & - & G & C \\
\hline
\end{array}
\end{array}$$

Blocks of type 2 or 3 cannot follow another block of the same type.

---

**General Gap Penalty**

- $A(i, j)$: similarity between $S_1[1, i]$ and $S_2[i, j]$ with "$S_1[i]$ is aligned with $S_2[j]$"

- $B(i, j)$: similarity between $S_1[1, i]$ and $S_2[i, j]$ with "space is aligned with $S_2[j]$"

- $C(i, j)$: similarity between $S_1[1, i]$ and $S_2[i, j]$ with "$S_1[i]$ is aligned with space"

$$Sim(S_1, S_2) = \max \left\{ A(|S_1|, |S_2|), B(|S_1|, |S_2|), C(|S_1|, |S_2|) \right\}$$
Recursive Function of $A(i, j)$

$$A(i, j) = \sigma(i, j) + \max \left\{ A(i - 1, j - 1), B(i - 1, j - 1), C(i - 1, j - 1) \right\}$$

- $A(i - 1, j - 1)$
- $B(i - 1, j - 1)$
- $C(i - 1, j - 1)$

Recursive Function of $B(i, j)$

$$B(i, j) = \max \left\{ A(i, j - k) - P_g(k), C(i, j - k) - P_g(k) \right\}$$

where $1 \leq k \leq j$
Recursive Function of $C(i, j)$

$C(i, j) = \max \left\{ \begin{array}{c} A(i - k, j) - P_g(k) \\ B(i - k, j) - P_g(k) \end{array} \right\}$

where $1 \leq k \leq j$

Complexity: General Gap Penalty

Initialization: $k \geq 1$
- $A(0, 0) = 0, B(0, k) = C(k, 0) = -P_g(k)$, others $= -\infty$
- Computing $A(i, j), B(i, j)$ and $C(i, j)$ needs to perform $3 + 2j + 2i$ accesses to previous entries.

Let $|S_1| = m$ and $|S_2| = n$.
Then time-complexity for general gap penalty is:

$$\sum_{i=1}^{m} \sum_{j=1}^{n} (3 + 2j + 2i) = O(m^2n + mn^2)$$
Affine Gap Penalty Function

- Affine function $P_a(k)$ for a gap with $k$ spaces:
  
  $$P_a(k) = \begin{cases} 
  w_g + kw_s & \text{if } k \geq 1 \\
  0 & \text{if } k = 0 
  \end{cases}$$

- The first space costs $w_g + w_s$, but the others cost $w_s$.

- From the evolutionary viewpoint, the occurrence of a gap with $k$ spaces (i.e., a mutation) is more probable than the occurrence of $k$ isolated spaces (i.e., $k$ mutations).

Affine Gap Penalty

- $A(i, j)$: similarity between $S_1[1, i]$ and $S_2[i, j]$ with "$S_1[i]$ is aligned with $S_2[j]$"

- $B(i, j)$: similarity between $S_1[1, i]$ and $S_2[i, j]$ with "space is aligned with $S_2[j]$"

- $C(i, j)$: similarity between $S_1[1, i]$ and $S_2[i, j]$ with "$S_1[i]$ is aligned with space"

$$Sim(S_1, S_2) = \max \left\{ A(|S_1|, |S_2|), B(|S_1|, |S_2|), C(|S_1|, |S_2|) \right\}$$
Recursive Function of $A(i, j)$

$A(i, j) = \sigma(i, j) + \max \left\{ \begin{array}{l}
A(i-1, j-1)
\end{array} \right. \begin{array}{l}
B(i-1, j-1)
C(i-1, j-1)
\end{array}$

Recursive Function of $B(i, j)$

$B(i, j) = \max \left\{ \begin{array}{l}
A(i, j-1) - (w_g + w_s)
B(i, j-1) - w_s
C(i, j-1) - (w_g + w_s)
\end{array} \right.$
Recursive Function of $C(i, j)$

$C(i, j) = \max \begin{cases} \mathcal{A}(i - 1, j) - (w_g + w_s) \\ \mathcal{B}(i - 1, j) - (w_g + w_s) \\ \mathcal{C}(i - 1, j) - w_s \end{cases}$

Sequence Comparison: Two Sequence Alignment

Complexity: Affine Gap Penalty

- Initializations: $k \geq 1$, $0 \leq i \leq m$, $0 \leq j \leq n$
  - $\mathcal{A}(0, 0) = 0$, $\mathcal{A}(k, 0) = \mathcal{A}(0, k) = -\infty$
  - $\mathcal{B}(i, 0) = -\infty$, $\mathcal{B}(0, k) = -(w_g + kw_s)$
  - $\mathcal{C}(k, 0) = -(w_g + kw_s)$, $\mathcal{C}(0, j) = -\infty$
- Computing $\mathcal{A}(i, j)$, $\mathcal{B}(i, j)$ and $\mathcal{C}(i, j)$ needs to perform $3 + 3 + 3$ accesses to previous entries.
- The time-complexity for affine gap penalty is:
  $$\sum_{i=1}^{m} \sum_{j=1}^{n} 9 = \mathcal{O}(mn)$$
## Summary: Gap Penalty Functions

<table>
<thead>
<tr>
<th>Type</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear gap penalty</td>
<td>$O(mn)$</td>
<td>1 array</td>
</tr>
<tr>
<td>$P_l(k) = kw_s$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Affine gap penalty</td>
<td>$O(mn)$</td>
<td>3 arrays</td>
</tr>
<tr>
<td>$P_a(k) = w_g + kw_s$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>General gap penalty</td>
<td>$O(m^2n + mn^2)$</td>
<td>3 arrays</td>
</tr>
</tbody>
</table>

By R.C.T. Lee and C.L. Lu