Lecture 4, Sequence Alignment

Young-Rae Cho
Associate Professor
Department of Computer Science
Baylor University

Overview

- Backgrounds
- Manhattan Tourist Problem
- Longest Common Subsequence Problem
- Edit Distance
- Global Sequence Alignment
- Local Sequence Alignment
- Alignment with Gap Penalty
Sequence Similarity

- Homologs
  - similar sequence + common ancestor (divergent evolution)
  - Orthlogs: homologs in different species by species divergence
  - Paralogs: homologs in the same species by gene duplication
- Analogs
  - similar sequence + no common ancestor (convergent evolution)

How to measure sequence similarity

1. Counting identical letters on each position

   A C G T T A T
   T C G T A C T

2. Inserting gaps to maximize the number of identical letters

   A C G T T A T
   T C G T A C T

Comparison of Sequence Similarity Measures

- Measure (1)
  - Compares the letters on the same position between two sequences
  - Not applicable to measurement of evolutionary distance

- Measure (2)
  - Compares the letters in the same order (even on different positions) between two sequences
  - More applicable to measurement of evolutionary distance
  - Why?
  - Example?
Sequence Alignment

Definition
- Arranging two or more DNA or protein sequences by inserting gaps to maximize their sequence similarity score
e.g., the number of identical positions
e.g., the sum of scores by any scoring scheme

Applications
- Given gene sequences, infer their evolutionary history (phylogenetics)
- Given gene sequences of known functions, infer the functions of newly sequenced genes
- Given genes of known functions in one organism, infer the functions of genes in another organism

Overview

Backgrounds
- Manhattan Tourist Problem
- Longest Common Subsequence Problem
- Edit Distance
- Global Sequence Alignment
- Local Sequence Alignment
- Alignment with Gap Penalty
Manhattan Tourist Problem (MTP)

- Problem Definition
  - A tourist seeks a path to travel with the most attractions in Manhattan road map (grid structure)
  - Restrictions
    - A path from a source to a sink
    - A path only eastward and southward

Formulation of MTP

- Goal
  - Finding the strongest path from a source to a sink in a weighted grid
    - The weight of an edge is defined as the number of attractions
    - The path strength is measured by summing the weights on the path

- Input
  - A weighted grid $G$ with two distinct vertices, source and sink

- Output
  - A strongest path in $G$ from the source to the sink
Example of MTP

- Example

![Graph showing example of MTP](image)

Solving by Exhaustive Search

- Algorithm
  1. Enumerate all possible paths from the source to the sink
  2. Compute the path strength for all possible paths
  3. Find the strongest path

- Problems?
Solving by Greedy Algorithm

- **Algorithm**
  1. Start from the source
  2. Select the edge having the highest weight
  3. Repeat (2) until it reaches the sink

- **Problems ?**
- **Runtime ?**

Solving by Recursive Algorithm

- **Algorithm**
  
  ```
  MTP(m, n)
  if m = 0 and n = 0
      return 0
  else if m = 0 and n ≠ 0
      return MTP(m, n - 1) + w((m, n - 1), (m, n))
  else if m ≠ 0 and n = 0
      return MTP(m - 1, n) + w((m - 1, n), (m, n))
  else
      x ← MTP(m - 1, n) + w((m - 1, n), (m, n))
      y ← MTP(m, n - 1) + w((m, n - 1), (m, n))
      return max(x, y)
  ```

- **Problems ?**
- **Runtime ?**
Solving by Dynamic Programming

- Algorithm

```plaintext
MTP(m, n)
S_{0,0} = 0
for i = 1 to m
    S_{i,0} = S_{i-1,0} + w((i-1,0),(i,0))
for j = 1 to n
    S_{0,j} = S_{0,j-1} + w((0,j-1),(0,j))
for i = 1 to m
    for j = 1 to n
        S_{i,j} = \max\{ S_{i-1,j} + w((i-1,j),(i,j)), S_{i,j-1} + w((i,j-1),(i,j)) \}
return S_{m,n}
```

- Recursive Formula

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

Example of Dynamic Programming

- Example

```
source  0  0  0  1  1
  0  1  2  3  4
  1  3  5  6  7
  1  4  5  9  10
   1  4  5  9  sink
```

- Runtime ?
Traversing Strategies

- Three Different Strategies
  - Column by column
  - Row by row
  - Along diagonals

Overview

- Backgrounds
- Manhattan Tourist Problem
- Longest Common Subsequence Problem
- Edit Distance
- Global Sequence Alignment
- Local Sequence Alignment
- Alignment with Gap Penalty
Longest Common Subsequences (1)

- Subsequence of $x$
  - An ordered sequence of letters from $x$
  - Not necessarily consecutive
  - e.g., $x$=“ATTGCTA”, “AGCA”, “TCG”, “ATCT”, “TGAT”?

- Common Subsequence of $x$ and $y$
  - e.g., $x$=“ATCTGAT” and $y$=“TGCATA”, “TCTA”, “TGAT”, “TATA”?

- Longest Common Subsequence (LCS) of $x$ and $y$?

Longest Common Subsequences (2)

- Definition of LCS
  - Given two sequences, $v = \langle v_1, v_2, ..., v_m \rangle$ and $w = \langle w_1, w_2, ..., w_n \rangle$,
  - LCS of $v$ and $w$ is a sequence of positions in
    - $v$: $1 \leq i_1 < i_2 < ... < i_t \leq m$
    - and a sequence of positions in
      - $w$: $1 \leq j_1 < j_2 < ... < j_t \leq n$
    - such that $i_t$-th letter of $v$ equals to $j_t$-letter of $w$, and $t$ is maximal
LCS in 2-Row Representation (1)

- Example
  - x = "ATCTGATG" (m = 8), y = "TGCATAC" (n = 7)
  
<table>
<thead>
<tr>
<th>x</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>AT-------</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>C.........</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>T........</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>G.........</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>A.........</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>T.........</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>G.........</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>-.........</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>y</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>T..........</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>G.........</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>C.........</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>A.........</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>T.........</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>A.........</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>C.........</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
</tbody>
</table>

- Position in x: 2 < 3 < 4 < 6
- Position in y: 1 < 3 < 5 < 6
- Common subsequence: "TCTA"

LCS in 2-Row Representation (2)

- Example - Continued
  - x = "ATCTGATG" (m = 8), y = "TGCATAC" (n = 7)

<table>
<thead>
<tr>
<th>x</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.........</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>T.........</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>C.........</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>T..........</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>G.........</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>A.........</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>T.........</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>G.........</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>-.........</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>y</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>T..........</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>G.........</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>C.........</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>A.........</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>T.........</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>A.........</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>C.........</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>x</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.........</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>T.........</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>C.........</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>T..........</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>G.........</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>A.........</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>T.........</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>G.........</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>-.........</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>y</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>T..........</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>-.........</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>G.........</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>C.........</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>A.........</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>T.........</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>A.........</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>C.........</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
</tbody>
</table>
LCS in 2-D Grid Representation

- Edit Graph
  - 2-D grid structure having diagonals on the position of the same letter

- Example
  - x="ATGTTAT" (m=7)
  - y="ATCGTAC" (n=7)
  - Strongest path in edit graph
    (0,0) → (1,1) → (2,2) → (2,3) → (3,4) → (4,5) → (5,5) → (6,6) → (7,6) → (7,7)

Formulation of LCS Problem

- Goal
  - Finding the longest common subsequence (LCS) of two sequences (length-\(m\), length-\(n\))
  - Finding the strongest path from a source to a sink in a weighted edit graph
  - The path strength is measured by summing the weights on the path

- Input
  - A weighted edit graph \(G\) with source \((0,0)\) and sink \((m,n)\)

- Output
  - A strongest path in \(G\) from the source to the sink
Solving by Exhaustive Search

- Algorithm
  1. Enumerate all possible paths from the source to the sink
  2. Compute the path strength for all possible paths
  3. Find the strongest path

- Problems ?

Solving by Greedy Algorithm

- Algorithm
  1. Start from the source
  2. Select the edge having the highest weight (i.e., if there is a diagonal edge, select it. Otherwise, select one of the other edges.)
  3. Repeat (2) until it reaches the sink

- Problems ?
- Runtime ?
Solving by Dynamic Programming

- **Recursive Formula**
  \[ S_{i,j} = \begin{cases} 
  S_{i-1,j} + 0 & \text{if } x_i = y_j \\
  S_{i,j-1} + 0 & \text{if } x_i \neq y_j \\
  S_{i-1,j-1} + 1 & \text{if } x_i = y_j 
  \end{cases} \]

- **Algorithm**

  ```plaintext
  LCS(x, y)
  for i = 0 to m
    S_{i,0} = 0
  for j = 1 to n
    S_{0,j} = 0
  for i = 1 to m
    for j = 1 to n
      if x_i = y_j
        S_{i,j} = max(S_{i-1,i}, S_{i,j-1}, S_{i-1,j-1} + 1)
      else
        S_{i,j} = max(S_{i-1,i}, S_{i,j-1})
  return S_{m,n}
  ```

Example of LCS

- **Example**
  - x="ATGTTAT" (m=7), y="ATCGTAC" (n=7)
Finding LCS

- Storing Directions
  \[ D_{i,j} = \begin{cases} 
  "\downarrow" & \text{if } S_{i,j} = S_{i-1,j} \\
  "\rightarrow" & \text{if } S_{i,j} = S_{i,j-1} \\
  "\triangledown" & \text{if } S_{i,j} = S_{i-1,j-1} + 1 
  \end{cases} \]

- Backtracking
  ```
  BACKTRACKING(D, x, i, j)
  if \( i > 0 \) and \( j > 0 \)
  if \( D_{i,j} = "\downarrow" \)
  BACKTRACKING(D, x, i - 1, j)
  else if \( D_{i,j} = "\rightarrow" \)
  BACKTRACKING(D, x, i, j - 1)
  else
  BACKTRACKING(D, x, i - 1, j - 1)
  print \( x_i \)
  ```

Overview

- Backgrounds
- Manhattan Tourist Problem
- Longest Common Subsequence Problem
  - Edit Distance
- Global Sequence Alignment
- Local Sequence Alignment
- Alignment with Gap Penalty
Edit Distance (1)

➢ Definition
  ▪ Edit distance between two sequences \(x\) and \(y\) : the minimum number of editing operations (insertion, deletion, substitution) to transform \(x\) into \(y\)

➢ Example
  ▪ \(x=\text{TGCATAT} \ (m=7), \ y=\text{ATCCGAT} \ (n=7)\)

<table>
<thead>
<tr>
<th>TGCATAT</th>
<th>ATGCATAT</th>
<th>ATCCATAT</th>
<th>ATCCGATAT</th>
<th>ATCCGATT</th>
<th>ATCCGAT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>insertion of &quot;A&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>insertion of &quot;G&quot; with &quot;C&quot;</td>
<td></td>
<td></td>
<td>deletion of &quot;A&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>insertion of &quot;G&quot;</td>
<td></td>
<td>deletion of &quot;T&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>edit distance = 5?</td>
<td></td>
</tr>
</tbody>
</table>

Edit Distance (2)

➢ Example
  ▪ \(x=\text{TGCATAT} \ (m=7), \ y=\text{ATCCGAT} \ (n=7)\)

<table>
<thead>
<tr>
<th>TGCATAT</th>
<th>ATGCAATAT</th>
<th>ATGCAAT</th>
<th>ATCCCAAT</th>
<th>ATCCGAT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>insertion of &quot;A&quot;</td>
<td>deletion of &quot;T&quot;</td>
<td>substitute of &quot;G&quot; with &quot;C&quot;</td>
<td>substitute of &quot;A&quot; with &quot;G&quot;</td>
</tr>
<tr>
<td></td>
<td>edit distance = 4?</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

➢ Can it be done in 3 steps?

➢ Features
  ▪ Allows comparison of two sequences of different lengths
2-Row Representation

- Example in 2-row representation
  - $x=\text{"ATCTGATG"}$ (m=8), $y=\text{"TGCATA"}$ (n=7)

```
AT CTGATG
TG CAT A C
```

- 4 matches
- 4 deletions
- 3 insertions

```
AT CTGATG
TG CAT A C
```

- 4 matches
- 3 deletions
- 2 insertions
- 1 substitution

Edit distance = #insertions + #deletions + #mismatches

Edit Distance in 2D Grid Representation

- Edit Graph
Solving by Exhaustive Search or Greedy Algorithm

- Exhaustive Search Algorithm
- Greedy Algorithm

Solving by Dynamic Programming

- Recursive Formula
- Dynamic Programming Algorithm
Overview

- Backgrounds
- Manhattan Tourist Problem
- Longest Common Subsequence Problem
- Edit Distance
- Global Sequence Alignment
- Local Sequence Alignment
- Alignment with Gap Penalty

from LCS to Sequence Alignment

- LCS problem
  - Allows only insertions and deletions - no substitutions
  - Scores 1 for a match and 0 for an insertion or deletion

- Sequence Alignment Problem
  - Allows gaps (insertions and deletions) and mismatches (substitutions)
  - Uses any scoring schemes
Formulation of Global Alignment Problem

- **Goal**
  - Finding the best alignment of two sequences under a given scoring schema

- **Input**
  - Two sequences $x$ (length-$m$) and $y$ (length-$n$), and a scoring schema

- **Output**
  - An alignment of $x$ and $y$ with the maximal score

Basic Scoring Scheme

- **Simplest Scoring Scheme**
  - Match premium: $+\alpha$
  - Mismatch penalty: $-\mu$
  - Insertion and deletion (gap) penalty: $-\sigma$

  \[ \text{Score} = \alpha \#\text{matches} - \mu \#\text{mismatches} - \sigma (\#\text{insertions} + \#\text{deletions}) \]

- **Recursive Formula**
  \[
  S_{i,j} = \max \begin{cases} S_{i-1,j} - \sigma \\ S_{i,j-1} - \sigma \\ S_{i-1,j-1} - \mu & \text{if } x_i \neq y_j \\ S_{i-1,j-1} + \alpha & \text{if } x_i = y_j \end{cases}
  \]

- **Runtime?**
Example of Sequence Alignment

- Example
  - x="ATGTTAT" (m=7), y="ATCGTAC" (n=7)

Solving by Dynamic Programming

- Algorithm

```plaintext
GLOBALALIGNMENT(x, y)
S_{0,0} ← 0
for i ← 1 to m
    S_{i,0} ← S_{i-1,0} - σ
for j ← 1 to n
    S_{0,j} ← S_{0,j-1} - σ
for i ← 1 to m
    for j ← 1 to n
        if x_i = y_j
            S_{i,j} ← max(S_{i-1,j} - σ, S_{i,j-1} - σ, S_{i-1,j-1} + α)
        else
            S_{i,j} ← max(S_{i-1,j} - σ, S_{i,j-1} - σ, S_{i-1,j-1} - µ)
return S_{m,n}
```
Advanced Scoring Scheme (1)

- Percent Identity
  - Percentage of identical matches

- Percent Similarity
  - Percentage of nucleotide pairs with the same type
  - Percentage of similar amino acid pairs in biochemical structure

- Advanced Scoring Scheme
  - Varying scores for matches
  - Varying, strong penalties for mismatches
  - Relative likelihood of evolutionary relationship → Probability of mutations
  - Define scoring matrix for DNA or protein sequences

Advanced Scoring Scheme (2)

- Scoring Matrix $\delta$
  - Also called substitution matrix
  - $4 \times 4$ array representation for DNA sequences
  - $20 \times 20$ array representation for protein sequences
  - Entry of $\delta (i,j)$ has the score between $i$ and $j$
    i.e., the rate at which $i$ is substituted with $j$ over time

- Recursive Formula

$$ S_{i,j} = \max \left\{ S_{i-1,j} + \delta(x_i,-) \quad S_{i,j-1} + \delta(-,y_j) \quad S_{i-1,j-1} + \delta(x_i,y_j) \right\} $$

- Runtime?
Scoring Matrix Example (1)

- **PAM (Point Accepted Mutations)**
  - For protein sequence alignment
  - Amino acid substitution frequency in mutations
  - Logarithmic matrix of mutation probabilities
  - PAM120: Results from 120 mutations per 100 residues
  - PAM120 vs. PAM240

- **BLOSUM (Block Substitution Matrix)**
  - For protein sequence alignment
  - Applied for local sequence alignments
  - Substitution frequencies between clustered groups
  - BLOSUM-62: Results with a threshold (cut-off) of 62% identity
  - BLOSUM-62 vs. BLOSUM-50

Scoring Matrix Example (2)

- **Substitution Matrix Examples**
  - BLOSUM-62
  - PAM120
Overview

- Backgrounds
- Manhattan Tourist Problem
- Longest Common Subsequence Problem
- Edit Distance
- Global Sequence Alignment
- Local Sequence Alignment
- Alignment with Gap Penalty

Local Sequence Alignment (1)

- Example
  - $x = \text{"TCAGTGTCGAAGTTA"}$
  - $y = \text{"TAGGCTAGCAGTGTG"}$

- Global Alignment

```
TCAG  -  T  -  G  T  C  G  A  A  G  T  -  TA
|     | |     |     |     | |     |  
T   -  A  G  G  C  T  A  G  -  C  -  A  -  G  T  G  T  G
```

- Local Alignment

```
TCAGTGT  C  G  A  A  G  T  TA
|     |     |     |     |     |     |
TAGGCTAG  C  A  G  T  G  T  G
```
Local Sequence Alignment (2)

- Example - Continued

T A G G C T A G C A G T G T G

Global vs. Local Alignment

- Global Alignment Problem
  - Finds the path having the largest weight between vertices (0,0) and (m,n) in the edit graph

- Local Alignment Problem
  - Finds the path having the largest weight between two arbitrary vertices, (i,j) and (i',j'), in the edit graph

- Score Comparison
  - The score of local alignment must be greater than (or equal to) the score of global alignment
Formulation of Local Alignment Problem

- **Goal**
  - Finding the best local alignment between two sequences

- **Input**
  - Two sequences $x$ and $y$, and a scoring matrix $\delta$

- **Output**
  - An alignment of substrings of $x$ and $y$ with the maximal score among all possible substrings of them

Implementation of Local Alignment

- **Strategy**

![Diagram of local alignment strategy](image)

Compute “mini” global alignment for local alignment
Solving by Exhaustive Search (1)

- **Process**
  1. Enumeration of all possible pairs of substrings
  2. Global alignment for each pair of substrings

- **Process re-written**
  1. Enumeration of all possible pairs of start position \((i,j)\) and end position \((i',j')\)
  2. Global alignment from each position \((i,j)\) to each position \((i',j')\)

- **Runtime**
  - Suppose two sequences have the same length \(n\)
  - Global alignment :
  - Total runtime :

Solving by Exhaustive Search (2)

- **Process improved**
  1. Enumeration of all possible starting positions \((i,j)\)
  2. Global alignment from each \((i,j)\)

- **Runtime**
  - Suppose two sequences have the same length \(n\)
  - Global alignment :
  - Total runtime :

- **Solution**
  - Free ride!
Solving by Dynamic Programming

- **Free Ride**
  - Assigns 0 weights from (0,0) to any other nodes (i,j)

- **Recursive Formula**

\[
S_{i,j} = \max \begin{cases} 
0 \\
S_{i-1,j} + \delta(x_i, -) \\
S_{i,j-1} + \delta(-, y_j) \\
S_{i-1,j-1} + \delta(x_i, y_j)
\end{cases}
\]

- **Runtime ?**

---

Overview

- **Backgrounds**
- **Manhattan Tourist Problem**
- **Longest Common Subsequence Problem**
- **Edit Distance**
- **Global Sequence Alignment**
- **Local Sequence Alignment**
- **Alignment with Gap Penalty**
Scoring Insertions/Deletions

- **Naïve Approach**
  - $-\sigma$ for 1 insertion/deletion,
  - $-2\sigma$ for 2 consecutive insertions/deletions
  - $-3\sigma$ for 3 consecutive insertions/deletions, etc.
  - → too severe penalty for a series of 100 consecutive insertions/deletions

- **Example**
  - $x$="ATAGC", $y$="ATATTGC"
    - $\text{ATA}_\text{GC}$
    - $\text{ATATTGC}$
  - $x$="ATAGGC", $y$="ATGTGC"
    - $\text{ATAG}_\text{GC}$
    - $\text{ATGTGC}$

Scoring Gaps of Insertions/Deletions

- **Gap**
  - Contiguous sequence of spaces in one of the rows
  - Contiguous sequence of insertions or deletions in 2-row representation

- **Linear Gap Penalty**
  - Score for a gap of length $x$ : $-\sigma x$ (Naïve approach)

- **Constant Gap Penalty**
  - Score for a gap of length $x$ : $-\rho$

- **Affine Gap Penalty**
  - Score for a gap of length $x$ : $-(\rho + \sigma x)$
  - $-\rho$ : gap opening (existence) penalty / $-\sigma$ : gap extension penalty  ($\rho \gg \sigma$)
Solving Constant/Affine Gap Penalty

- Edit Graph Update
  - Add “long” horizontal or vertical edges to the edit graph
  - Runtime?

Improved Solution for Constant/Affine Gap Penalty

- 3-Layer Grid Structure
  - Middle layer (Main layer) for diagonal edges
    - Extends matches and mismatches
  - Upper layer for horizontal edges
    - Creates/extends gaps in a sequence $y$
  - Lower layer for vertical edges
    - Creates/extends gaps in a sequence $x$

- Gap opening penalty ($-\rho$) for jumping from middle layer to upper/lower layer
- Gap extension penalty ($-\sigma$) for extending on upper/lower layer
Example of 3-Layer Grid

upper layer (gaps in $x$)

main layer (matches/mismatches)

lower layer (gaps in $y$)

Solving by Dynamic Programming

- **Recursive Formula**

  $S_{i,j} = \max \begin{cases} S_{i-1,j-1} + \delta(x_i, y_j) & \text{→ match / mismatch} \\ S_{i,j}^{\text{lower}} \\ S_{i,j}^{\text{upper}} \end{cases}$

  $S_{i,j}^{\text{lower}} = \max \begin{cases} S_{i-1,j}^{\text{lower}} - \sigma & \text{→ continuing gap in } x \\ S_{i,j} - (\rho + \sigma) & \text{→ starting gap in } x \end{cases}$

  $S_{i,j}^{\text{upper}} = \max \begin{cases} S_{i,j-1}^{\text{upper}} - \sigma & \text{→ continuing gap in } y \\ S_{i,j} - (\rho + \sigma) & \text{→ starting gap in } y \end{cases}$

- **Runtime ?**
Questions?

- Lecture Slides are found on the Course Website,
  web.ecs.baylor.edu/faculty/cho/3360