Sequence Alignment
Previous Lecture

- Alignment Graph, Manhattan Tours, Edit Distance and LCS
- Sequence Alignment in Linear Space
Manhattan Tourist Problem (MTP)

Imagine seeking a path (from source to sink) to travel (only eastward and southward) with the most number of attractions (*) in the Manhattan grid.

![Manhattan Grid Diagram with Source and Sink]
Problem: Given 2 sequences, \( X = \langle x_1, \ldots, x_m \rangle \) and \( Y = \langle y_1, \ldots, y_n \rangle \), find a common subsequence whose length is maximum.

\[
\begin{align*}
X: & \quad \text{springtime} \\
Y: & \quad \text{printing} \\
\text{LCS}(X,Y): & \quad \text{printing}
\end{align*}
\]
Every path is a common subsequence.

Every diagonal edge adds an extra element to common subsequence

**LCS Problem:** Find a path with maximum number of diagonal edges
Computing LCS

Let $v_i = \text{prefix of } v \text{ of length } i$: $v_1 \ldots v_i$
and $w_j = \text{prefix of } w \text{ of length } j$: $w_1 \ldots w_j$
The length of $\text{LCS}(v_i, w_j)$ is computed by:

$$s_{i, j} = \max \begin{cases} 
  s_{i-1, j} \\
  s_{i, j-1} \\
  s_{i-1, j-1} + 1 \text{ if } v_i = w_j
\end{cases}$$
Every Path in the Grid Corresponds to an Alignment
Aligning DNA Sequences

Requires Traceback in addition to Score Computation in the Alignment Graph

\[ V = ATCTGATG \quad n = 8 \]
\[ W = TGCATAC \quad m = 7 \]

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

Indels

- Deletions
- Insertions
Today:

- Hamming Distance and Edit Distance
- Scoring Matrices
- Global versus Local Sequence Alignment
- Overlap ("Ends Free") Alignment
Sequence Alignment Scoring Schemes:

Maximizing Similarity (e.g. LCS) Versus Minimizing Distance (Hamming, Edit distance)
Aligning Sequences without Insertions and Deletions: Hamming Distance

Given two DNA sequences \( \mathbf{v} \) and \( \mathbf{w} \):

\[
\mathbf{v} : \quad \text{ATATATATAT}
\]
\[
\mathbf{w} : \quad \text{TATATATA}
\]

- The Hamming distance: \( d_H(\mathbf{v}, \mathbf{w}) = 8 \) is large but the sequences are very similar.
Aligning Sequences with Insertions and Deletions

By shifting one sequence over one position:

\[ v : \text{ATATATATAT} \]
\[ w : \text{--TATATATATA} \]

- The edit distance: \( d_H(v, w) = 2 \).
- Hamming distance neglects insertions and deletions in DNA.
Edit Distance

Levenshtein (1966) introduced edit distance between two strings as the minimum number of elementary operations (insertions, deletions, and substitutions) to transform one string into the other

\[ d(v, w) = \text{MIN number of elementary operations to transform } v \rightarrow w \]

Edge weights in the Alignment Grid?
Edit Distance

Levenshtein (1966) introduced edit distance between two strings as the minimum number of elementary operations (insertions, deletions, and substitutions) to transform one string into the other

\[ d(v,w) = \text{MIN number of elementary operations to transform } v \rightarrow w \]

Edge weights (penalties) in the Alignment Grid:
- Diagonal Match: 0
- Horizontal, Vertical and Diagonal Mismatch: 1
Edit Distance vs Hamming Distance

Hamming distance always compares
\(i^{th}\) letter of \(v\) with
\(i^{th}\) letter of \(w\)

\(v = \text{ATA}\text{TATATAT}\)
\(w = \text{TATATATA}\)

Hamming distance:
\(d(v, w) = 8\)
Computing Hamming distance is a trivial task.
Edit Distance vs Hamming Distance

Hamming distance always compares $i^{th}$ letter of $v$ with $i^{th}$ letter of $w$

- $v = \text{ATATATATA}$
- $w = \text{TATATATA}$

Hamming distance:

$$d(v, w) = 8$$

Computing Hamming distance is a trivial task

Edit distance may compare $i^{th}$ letter of $v$ with $j^{th}$ letter of $w$

- $v = -\text{ATATATATA}$
- $w = \text{TATATATA}$

Edit distance:

$$d(v, w) = 2$$

Computing edit distance is a non-trivial task

Just one shift
Make it all line up
Edit Distance vs Hamming Distance

Hamming distance always compares $i^{th}$ letter of $v$ with $i^{th}$ letter of $w$

$V = \text{ATATATAT}$
$W = \text{TATATATA}$

Hamming distance:
$d(v, w) = 8$

Edit distance may compare $i^{th}$ letter of $v$ with $j^{th}$ letter of $w$

$V = \text{-ATATATAT}$
$W = \text{TATATATA}$

Edit distance:
$d(v, w) = 2$

(one insertion and one deletion)

How to find what $j$ goes with what $i$ ???
LCS Example

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>T</th>
<th>C</th>
<th>G</th>
<th>T</th>
<th>A</th>
<th>C</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Initialize $1^{st}$ row and $1^{st}$ column to be all zeroes.

Or, to be more precise, initialize $0^{th}$ row and $0^{th}$ column to be all zeroes.
LCS Example

\[ S_{i,j} = \begin{cases} S_{i-1,j-1} \quad \text{value from NW} +1, & \text{if } v_i = w_j \\ \max \{ S_{i-1,j}, S_{i,j-1} \} & \text{value from North (top)} \\ S_{i,j-1} & \text{value from West (left)} \end{cases} \]
Alignment: Backtracking

Arrows show where the score originated from.

- if from the top
- if from the left
- if $v_i = w_j$
Backtracking Example

Find a match in row and column 2.

i=2, j=2,5 is a match (T).

j=2, i=4,5,7 is a match (T).

Since $v_i = w_j$, $s_{i,j} = s_{i-1,j-1} + 1$

$s_{2,2} = \lfloor s_{1,1} = 1 \rfloor + 1$
$s_{2,5} = \lfloor s_{1,4} = 1 \rfloor + 1$
$s_{4,2} = \lfloor s_{3,1} = 1 \rfloor + 1$
$s_{5,2} = \lfloor s_{4,1} = 1 \rfloor + 1$
$s_{7,2} = \lfloor s_{6,1} = 1 \rfloor + 1$
Backtracking Example

Continuing with the dynamic programming algorithm gives this result.
Alignment: Dynamic Programming

\[ s_{i,j} = \begin{cases} 
\max & \begin{cases} 
\begin{align*}
& s_{i-1, j-1} + 1 \text{ if } v_i = w_j \\
& s_{i-1, j} \\
& s_{i, j-1}
\end{align*}
\end{cases} 
\end{cases} \]
Alignment: Dynamic Programming

\[ s_{i,j} = \max \begin{cases} s_{i-1, j-1} + 1 & \text{if } v_i = w_j \\ s_{i-1, j} + 0 \\ s_{i, j-1} + 0 \end{cases} \]

This recurrence corresponds to the Manhattan Tourist problem (three incoming edges into a vertex) with all horizontal and vertical edges weighted by zero.
Sequence Modifications

- Three types of changes
  - Substitution (point mutation)
  - Insertion
  - Deletion \{ Indel (replication slippage) \}

TCAGT → TCCGT → TCGA GT → TCAGT
TCAGT → TCGT → TCGA GT → TCAGT
Choosing Alignments

There are many possible alignments

For example, compare:

\[
\begin{align*}
&\text{GCGC} - \text{ATGGATTGAGCGA} \\
&\text{TGCGCCATTGAT} - \text{GACC} - \text{A}
\end{align*}
\]

to

\[
\begin{align*}
&\text{--------} - \text{GCGC} - \text{ATGGATTGAGCGA} \\
&\text{TGCGCC} - \text{ATTGATGACCA} - \text{--}
\end{align*}
\]

Which one is better?
Another example

Given two sequences:

X: TGCATAT
Y: ATCCGAT

**Question:**

How can X be transformed into Y?

Or,

How did Y evolve from X?
One possible transformation

TGCATA\textsuperscript{X}

\hspace{1cm} delete T

\hspace{1cm} delete A

\hspace{1cm} insert A

\hspace{1cm} G \rightarrow C

\hspace{1cm} insert G

\hspace{1cm} 5 operations

Alignment:

\begin{tabular}{c}
\hline
TGC & ATAT \\
\hline
ATCC & GAT
\end{tabular}
Another possible transformation

Which one is better?
In order to align two sequences we need a quantitative model to evaluate similarity between sequences.

How do we quantitate sequence similarity?
Scoring Similarity

• Assume independent mutation model
  • Each site considered separately

• Score at each site
  • Positive if the same
  • Negative if different

• Sum to make final score
  • Can be positive or negative
  • Significance depends on sequence length
Pairwise Alignment - Identity

Human Hemoglobin (HH) vs Sperm Whale Myoglobin (SWM):

(HH) VLSPADKTNVKAAWGKVGAGHAGYEG
     || || || || || || || || || |

(SWM) VLSEGEWQLVLHVWAKVEADVAGHG

• Percent Identity: 36.000 (| only)
D and E are similar:
1. structure is similar.
2. both are acidic and hydrophilic
3. one mutation can separate them from one to the other.

\[
\begin{align*}
(\text{HH}) & \quad \text{VLSPADKTNVKAAWGKVGAHAGYEG} \\
(\text{SWM}) & \quad \text{VLSEGEWQLVLHVWAKVEADVAGHG}
\end{align*}
\]

- Percent Similarity: 40.000 (| and .)
- Percent Identity: 36.000 (| only)
Pairwise Alignment – Gap insertion

(HH)  \[ \text{VLSPADKTNVKAAWGKVG AH-AGYEG} \]

(SWM)  \[ \text{VLSEGEWQLVLHVWAKVEADVAGH-G} \]

- Gaps: 2
- Percent Similarity: 54.167
- Percent Identity: 45.833 (12/26)
Pairwise Alignment - Scoring

- The final score of the alignment is the sum of the positive scores and penalty scores:
  
  \[ \text{Alignment score} = + \text{Number of Identities} + \text{Number of Similarities} - \text{Number of gap insertions} \]
Pairwise Alignment - Scoring

(HH) VLSPADKTNVKAAWGKVGAG-AGYEG
    | | | . | | | | | | | | |
(SWM) VLSEGEWQLVLHVWAKVEADVAGH-G

Final score:

(V,V) + (L,L) + (S,S) + (D,E) + ...
- (penalty for gap insertion) \times (number of gaps)

We are interested in both the score and the alignment trace.
Optimum Alignment

The score of an alignment is a measure of its quality

Optimum alignment problem: Given a pair of sequences $X$ and $Y$, find an alignment (global or local) with maximum score

The similarity between $X$ and $Y$, denoted $sim(X, Y)$, is the maximum score of an alignment of $X$ and $Y$
From LCS to Alignment: Change the Scoring

- The Longest Common Subsequence (LCS) problem—the simplest form of sequence alignment—allows only insertions and deletions (no mismatches).
- In the LCS Problem, we scored 1 for matches and 0 for indels.
- Consider penalizing indels and mismatches with negative scores.
- Simplest scoring schema:
  - $+1$: match premium
  - $-\mu$: mismatch penalty
  - $-\sigma$: indel penalty
Simple Scoring

• When mismatches are penalized by $-\mu$, indels are penalized by $-\sigma$, and matches are rewarded with $+1$, the resulting score is:

$$\#\text{matches} - \mu(\#\text{mismatches}) - \sigma(\#\text{indels})$$
The Global Alignment Problem

Find the best alignment between two strings under a given scoring schema

**Input**: Strings \( v \) and \( w \) and a scoring schema

**Output**: Alignment of maximum score

\[
\begin{align*}
\uparrow \rightarrow &= -\sigma \\
&= 1 \text{ if match} \\
&= -\mu \text{ if mismatch}
\end{align*}
\]

\[
s_{i,j} = \max \begin{cases} 
    s_{i-1,j-1} + 1 & \text{if } v_i = w_j \\
    s_{i-1,j-1} - \mu & \text{if } v_i \neq w_j \\
    s_{i-1,j} - \sigma \\
    s_{i,j-1} - \sigma 
\end{cases}
\]

\( \mu \): mismatch penalty
\( \sigma \): indel penalty
Examples 1 and 2
Scoring Matrices

To generalize scoring, consider a \((4+1) \times (4+1)\) scoring matrix \(\delta\).

In the case of an amino acid sequence alignment, the scoring matrix would be a \((20+1) \times (20+1)\) size. The addition of 1 is to include the score for comparison of a gap character “-”.

This will simplify the algorithm as follows:

\[
\begin{align*}
    s_{i,j} &= \max \left\{ s_{i-1,j-1} + \delta(v_i, w_j), \\
    &\quad s_{i-1,j} + \delta(v_i, -), \\
    &\quad s_{i,j-1} + \delta(-, w_j) \right\}
\end{align*}
\]

<table>
<thead>
<tr>
<th>Scoring Matrix (\delta)</th>
</tr>
</thead>
</table>
| \(\begin{array}{cccccc}
    - & -1 & -1 & -1 & -1 \\
    a & -1 & 1 & -1 & -1 & -1 \\
    c & -1 & -1 & 1 & -1 & -1 \\
    g & -1 & -1 & -1 & 1 & -1 \\
    t & -1 & -1 & -1 & -1 & 1
    \end{array}\) |
Making a Scoring Matrix

- Scoring matrices are created based on biological evidence.
- Alignments can be thought of as two sequences that differ due to mutations.
- Some of these mutations have little effect on the protein’s function, therefore some penalties, $\delta(v_i, w_j)$, will be less harsh than others.
Scoring Matrix: Example

- Notice that although R and K are different amino acids, they have a positive score.
- Why? They are both positively charged amino acids → will not greatly change function of protein.

\[
\begin{array}{ccccc}
& A & R & N & K \\
A & 5 & -2 & -1 & -1 \\
R & - & 7 & -1 & 3 \\
N & - & - & 7 & 0 \\
K & - & - & - & 6 \\
\end{array}
\]

AKRANR
KAAANK

-1 + (-1) + (-2) + 5 + 7 + 3 = 11
Conservation

• Amino acid changes that tend to preserve the physico-chemical properties of the original residue are more likely to be accepted
  • Polar to polar
    • aspartate → glutamate
  • Nonpolar to nonpolar
    • alanine → valine
  • Similarly behaving residues
    • leucine to isoleucine
Scoring matrices

• Amino acid substitution matrices
  • PAM
  • BLOSUM

• DNA substitution matrices
  • DNA is less conserved than protein sequences
  • Less effective to compare coding regions at nucleotide level
# The Blosum50 Scoring Matrix

|     | A | R | N | D | C | Q | E | G | H | I | L | K | M | F | P | S | T | W | Y | V | B | Z | X *
| A   | 5 | -2| -1| -2| -1| -1| 0 | -2| -1| -2| -1| -1| 0 | -3| -2| 0 | -2| -1| -1| 5 |
| R   | -2| 7 | -1| -2| -4| 1 | 0 | -3| 0 | -4| -3| 2 | -3| -3| 3 | -1| -3| -1| -3| 0 |
| N   | -1| 1 | 7 | 2 | 2 | 2 | 0 | 0 | 1 | -3| -4| 0 | -2| -4| -2| 1 | 0 | -4| -2| 3 |
| D   | -2| 2 | 8 | -4| 0 | 2 | -1| -1| -4| -1| -4| -5| 1 | 0 | -1| -1| -5| -3| -3| 4 |
| C   | -1| -4| -2| 13| 3 | 3 | 3 | -3| 2 | -3| 2 | -3| -3| 2 | -3| 4 | 1 | -1| -1| 3 |
| Q   | 1 | 0 | 0 | -3| 7 | 2 | 2 | 1 | -3| -2| 2 | 0 | -4| -1| 0 | -1| -1| -1| -3| 0 |
| E   | -1| 0 | 1 | 2 | 1 | 0 | 0 | -4| 1 | -2| 3 | -1| -3| -1| 1 | -1| -3| -1| -2| 5 |
| G   | 0 | -3| 0 | -1| -3| -2| 3 | -8| -2| -4| -2| -3| -4| 1 | -2| 0 | -2| -3| -4| 1 |
| H   | -2| 0 | 1 | 1 | 3 | 1 | 0 | -2| 10| 4 | 0 | -3| 0| -1| 1 | 2 | -1| 2 | -3| 2 |
| I   | -1| -4| -3| -4| -2| 5 | 2 | -3| 2 | 0 | 3 | -3| -1| 3 | -1| 4 | -4| -4| -3| 1 |
| L   | 2 | -3| -4| -4| -2| 2 | 3 | -4| -3| 2 | 5 | -3| -3| 1 | -4| 3 | -1| 2 | -1| 1 |
| K   | -3| 1 | 0 | 1 | 2 | 1 | 0 | -3| 3 | 6 | 2 | -4| 1 | 0 | 1 | -3| 2 | 3 | 0 | 1 |
| M   | 1 | 2 | -2| -4| -2| 0 | 2 | -3| -1| 2 | 3 | -2| 7 | 0 | 0 | 3 | -2| 1 | 0 | 1 |
| F   | -3| 3 | -4| -5| -2| -4| -3| -4| 1 | 0 | 1 | -4| 0 | 8 | -4| 3| -2| 1 | 4 | 1 |
| P   | 1 | -3| -2| -1| -4| -1| -2| -3| 4 | 1 | 3 | -4| 10| -1| 1| -4| 3 | -3| 2 | 1 |
| S   | 1 | -1| 1 | 0 | -1| 0 | 1 | 0 | -1| 3 | 3 | 0 | -2| -3| 1 | 5 | 2 | 4 | -2| 2 |
| T   | 0 | -1| 0 | -1| -1| -1| -2| 1 | -1| 2 | 5 | -3| 2 | 0 | 0 | -1| 0 | 0 | 5 |
| W   | 3 | 3 | 4 | 5 | 5 | 1 | 3 | 3 | 3 | 2 | 3 | -1 | 4 | 4 | 3 | 1 | 5 | 2 | 3 | 5 |
| Y   | -2| 1 | -2| -3| -1| -2| -3| 2 | -1| -1| -2| 0 | 4 | 3 | -2| 2 | 2 | 8 | -1| 3 |
| V   | 0 | -3| -3| -4| -1| -3| -4| -4| 1 | 3 | 1 | 1 | -3| -3| 2 | 0 | 3 | -1| 5 | -4| 3 |
| B   | -2| 1 | -4| 5 | 3 | 0 | 1 | 1 | 0 | 4 | 0 | 3 | 4 | 2 | 0 | 0 | 5 | 3 | 4 | 5 |
| Z   | -1| 0 | 0 | 1 | -3| 4 | 5 | -2| 0 | 3 | -3| 1 | -4| 1 | 0 | -1| 2 | -2| 3 | 2 |
| X   | -1| -1| -1| -2| -1| -1| -1| -1| -1| -1| -1| -1| -1| -1| -1| -1| -1| -1| -1| 1 |
| *   | -5| -5| -5| -5| -5| -5| -5| -5| -5| -5| -5| -5| -5| -5| -5| -5| -5| -5| -5| 1 |
Local vs. Global Alignment

- The **Global Alignment Problem** tries to find the longest path between vertices \((0,0)\) and \((n,m)\) in the edit graph.

- The **Local Alignment Problem** tries to find the longest path among paths between arbitrary vertices \((i,j)\) and \((i’, j’)\) in the edit graph.
Two Sequence Alignment Problems

Global Alignment.

A = ctacga gac

| | | | | | | |

B = aacgacgat

Local Alignment.

A = ct acga gac

| | | | | | |

B = aacgacgat

The Scoring Matrix

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>c</th>
<th>g</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>a</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>c</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>g</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>t</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>
Two Sequence Alignment Problems

Global Alignment.

A = c t a c g a  g a c
   | | | | | | | |
B = a a c g a c g a  t

Value: 2

Local Alignment.

A = c t a c g a  g a c
   | | | | | | | |
B = a a c g a c g a  t

Value: 5

The Scoring Matrix

\[
\begin{array}{cccccc}
  & - & a & c & g & t \\
- & -1 & -1 & -1 & -1 & -1 \\
a & -1 & 1 & -1 & -1 & -1 \\
c & -1 & -1 & 1 & -1 & -1 \\
g & -1 & -1 & -1 & 1 & -1 \\
t & -1 & -1 & -1 & -1 & 1 \\
\end{array}
\]
The \( O(n^2) \) time, Classical Dynamic Programming Algorithm

The Alignment Graph

\[ |B| = n \]

\[ |A| = n \]

The Scoring Matrix \( \delta \)

\[
\begin{array}{cccccc}
- & a & c & g & t \\
- & -1 & -1 & -1 & -1 \\
a & -1 & 1 & -1 & -1 & -1 \\
c & -1 & -1 & 1 & -1 & -1 \\
g & -1 & -1 & -1 & 1 & -1 \\
t & -1 & -1 & -1 & -1 & 1 \\
\end{array}
\]
Computing the Optimal Global Alignment Value

\(|B| = n\)

\(|A| = n\)

Classical Dynamic Programming: \(O(n^2)\)
The O(n²) time, Classical Dynamic Programming Algorithm

The Alignment Graph

| B | = n

| A | = n

A: a c g
B: a c g

A: a c g a
B: a c g a

A: a c g a
B: a c g a

A: a c g a
B: a c g a

O = max(I_x + edge[I_x, O])

x = 1

Can the quadratic complexity of the optimal alignment value computation be reduced without relaxing the problem?
Computing an Optimal Local Alignment Value

| B | = n

| A | = n

The recursion For Local Alignment?

Classical Dynamic Programming: O(n^2)
Find crucial (conserved) features

VERY DIFFERENT

VERY SIMILAR
Conserved between many organisms

Conservation

CATATTGCAGTGGTCCCGCGTCAGGCT
TAAATTGCACGT-GGTCGCACTGCACGCT
Local vs. Global Alignment

- The **Global Alignment Problem** tries to find the longest path between vertices \((0,0)\) and \((n,m)\) in the edit graph.

- The **Local Alignment Problem** tries to find the longest path among paths between *arbitrary vertices* \((i,j)\) and \((i',j')\) in the edit graph.

- In the edit graph with negatively-scored edges, Local Alignment may score higher than Global Alignment.
Local vs. Global Alignment (cont’d)

- **Global Alignment**
  
  --T--CC--C--AGT--TATGT--CAGGGGACACG--A--GCATGCAGA--GAC
  
  AATTGCCGCC--GTCGT--T--TTCAG----CA--GT'TATG--T--CAGAT--C

- **Local Alignment**—better alignment to find conserved segment

  tccCAGTTATGTCAGgggacacgagcatgcagagac
  
  aattgccgccgtcgtttttcagCAGTTATGTCAGatc
Global vs. Local alignment

DorothyHodkin
DorothyCrowfootHodkin

Global alignment:
DOROTHY--------HODGKIN
DOROTHYCROWFOOTHODGKIN

Local alignment:
DOROTHY
DOROTHY

HODGKIN
HODGKIN
Global vs. Local Alignment

Source: Jones and Pevzner
The Local Alignment Problem

- **Goal**: Find the best local alignment between two strings
- **Input**: Strings \( v, w \) and scoring matrix \( \delta \)
- **Output**: Alignment of substrings of \( v \) and \( w \) whose alignment score is maximum among all possible alignment of all possible substrings
The Problem with this Problem

- Naive run time $O(n^4)$:
  - In the grid of size $n \times n$ there are $\sim n^2$ vertices $(i,j)$ that may serve as a source.
  - For each such vertex computing alignments from $(i,j)$ to $(i',j')$ takes $O(n^2)$ time.

- This can be remedied by giving “free rides”
Local Alignment: Example

Compute a “mini” Global Alignment to get Local
Local Alignment: Example
Local Alignment: Example
Local Alignment: Example
Local Alignment: Example
Local Alignment: Example
Local Alignment: Running Time

- Naive run time $O(n^4)$:
  - In the grid of size $n \times n$ there are $\sim n^2$ vertices $(i,j)$ that may serve as a source.
  - For each such vertex computing alignments from $(i,j)$ to $(i',j')$ takes $O(n^2)$ time.

This can be remedied by giving “free rides”
Local Alignment: Free Rides

Vertex (0,0)

The dashed edges represent the free rides from (0,0) to every other node.
The Local Alignment Recurrence

• The largest value of $s_{i,j}$ over the whole edit graph is the score of the best local alignment.

• The recurrence:

$$s_{i,j} = \max \begin{cases} 
0 \\
 s_{i-1,j-1} + \delta(v_i, w_j) \\
 s_{i-1,j} + \delta(v_i, -) \\
 s_{i,j-1} + \delta(-, w_j) 
\end{cases}$$

Notice there is only this change from the original recurrence of a Global Alignment.
The Local Alignment Recurrence

• The largest value of $s_{i,j}$ over the whole edit graph is the score of the best local alignment.

• The recurrence:

$$ s_{i,j} = \max \begin{cases} 0 \\ s_{i-1,j-1} + \delta(v_i, w_j) \\ s_{i-1,j} + \delta(v_i, -) \\ s_{i,j-1} + \delta(-, w_j) \end{cases} $$

**Power of ZERO**: there is only this change from the original recurrence of a Global Alignment - since there is only one “free ride” edge entering into every vertex.
Local Alignment: Algorithm

\[ C[i, j] = \text{Score of optimally aligning a suffix of } S_{1 \ldots i} \text{ with a suffix of } T_{1 \ldots j}. \]

\[ C[i, j] = \max \begin{cases} 
  C[i - 1, j - 1] + \text{score}(s[i], t[j]) \\
  C[i - 1, j] - \gamma \\
  C[i, j - 1] - \gamma \\
  0 
\end{cases} \]

Initialize top row and leftmost column to zero.
<table>
<thead>
<tr>
<th></th>
<th>λ</th>
<th>C</th>
<th>T</th>
<th>C</th>
<th>G</th>
<th>C</th>
<th>A</th>
<th>G</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>λ</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

+1 for a match, -1 for a mismatch, -5 for a space
Example 3
Variants of Sequence Alignment

We have seen two basic variants of sequence alignment:

- Global alignment (Needelman-Wunsch)
- Local alignment (Smith-Waterman)

We will pose and solve another problem:

- Finding the best overlap alignment
Overlap Alignment

Consider the following question:

Can we find the most significant overlap between two sequences $s,t$?

Possible overlap relations:

- a. 
- b. 

The difference between this problem and local alignment is that here we require alignment between the endpoints of the two sequences.
End-gap free alignment

- Gaps at the start or end of alignment are not penalized

\[ S = c a c t g t a c \]
\[ T = g a c a c t t g \]

Match: +2
Mismatch and space: -1

Best global
\[
\begin{align*}
  &c a c - - t - g t a c \\
g a c a c t t g &- - -
\end{align*}
\]
Score = 1

Best end-gap free
\[
\begin{align*}
  &- - c a c - t g t a c \\
g a c a c t t g &- - -
\end{align*}
\]
Score = 9
Motivation: Shotgun assembly

Figure 2.6: Sequence assembly.
Motivation: Shotgun assembly

Figure 2.6: Sequence assembly.
Motivation: Shotgun assembly

- Shotgun assembly produces large set of partially overlapping subsequences from many copies of one unknown DNA sequence.
- Problem: Use the overlapping sections to "paste" the subsequences together.
- Overlapping pairs will have low global alignment score, but high end-space free score because of overlap.
- **HOW CAN THIS BE SOLVED?**
Formally, given $s[1..n]$ and $t[1..m]$, find $i, j$ such that $d = \max\{d(s[1..i], t[j..m]),
\quad d(s[i..n], t[1..j]),
\quad d(s[1..n], t[i..j]),
\quad d(s[i..j], t[1..m])\}$ is maximal.

**Solution:** Same as Global alignment except that the dynamic programming should not penalize overhanging ends.
Overlap Alignment

**Initialization:** $V[i,0]=0$, $V[0,j]=0$.

- **Recurrence:** as in global alignment
  \[
  V[i+1,j+1] = \max \left\{ V[i,j] + \sigma(s[i+1],t[j+1]),
  V[i,j+1] + \sigma(s[i+1],-),
  V[i+1,j] + \sigma(-,t[j+1]) \right\}
  \]

  
  **Path** starts at the top or left border of the matrix and finishes on the right or bottom border.

- **Score:** maximum value at the bottom line and rightmost line

---

![Diagram](image_url)
Algorithm

- Same as global alignment, except:
  1. Initialize with zeros (free gaps at start)
- Locate max in the last row/column (free gaps at end)
Overlap Alignment

- **Initialization:** $V[i,0] = 0, V[0,j] = 0$

- **Recurrence:** as in global alignment

\[
V[i+1,j+1] = \max \left( \begin{array}{c}
V[i,j] + \sigma(s[i+1], t[j+1]) \\
V[i,j+1] + \sigma(s[i+1], -) \\
V[i+1,j] + \sigma(-, t[j+1])
\end{array} \right)
\]

- a **match** starts at the top or left border of the matrix and finishes on the right or bottom border.

- **Score:** maximum value at the bottom line and rightmost line in the matrix
Example 4