For written notes on this lecture, please read chapter 10 of *The Practical Bioinformatician*, and chapter 2 and 5 of *Algorithms in Bioinformatics*.

CS2220: Introduction to Computational Biology
Unit 4: Essence of Sequence Comparison

Wong Limsoon
Plan

• Dynamic programming

• Protein evolution

• String comparison

• Sequence alignment
  – Pairwise alignment
  – Multiple alignment

• Popular tools
  – FASTA, BLAST, Pattern Hunter
Dynamic programming
Knapsack problem

- Each item that can go into the knapsack has a size and a benefit

- The knapsack has a certain capacity

- What should go into the knapsack to maximize the total benefit?
Formulation of a solution

Source: http://mat.gsia.cmu.edu/classes/dynamic/node6.html

• Intuitively, to fill a \( w \)-pound knapsack, we must start by adding some item. If we add item \( j \), we end up with a knapsack \( k' \) of size \( w - w_j \) to fill …

\[
g(w) = \max_j \{ b_j + g(w - w_j) \}
\]

where

– \( w_j \) and \( b_j \) be weight and benefit for item \( j \)
– \( g(w) \) is max benefit that can be gained from a \( w \)-pound knapsack


Exercise #1

• Does $g(w)$ produce the optimal benefit? Prove it

$$
g(w) = \max_j \{ b_j + g(w - w_j) \}
$$

where

– $w_j$ and $b_j$ be weight and benefit for item $j$
– $g(w)$ is max benefit that can be gained from a $w$-pound knapsack
Direct recursive evaluation is inefficient

- $g(1), g(2), \ldots$ are computed many times

\[
g(w) = \max_j \{ b_j + g(w - w_j) \}
\]
“Memoize” to avoid recomputation

int s[]; s[0] := 0;
\[ g'(w) = \begin{cases} 
\text{if } s[w] \text{ is defined} \\
\text{then return } s[w]; \\
\text{else} \\
\quad s[w] := \max_j \{b_j + g'(w - w_j)\}; \\
\quad \text{return } s[w]; 
\end{cases} \]

<table>
<thead>
<tr>
<th>Item ((j))</th>
<th>Weight ((w_j))</th>
<th>Benefit ((b_j))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>65</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>80</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>30</td>
</tr>
</tbody>
</table>

\[ g(w) = \max_j \{b_j + g(w - w_j)\} \]
Exercise #2

- In what order do $s[0]$, $s[1]$, ... get defined?

```c
int s[]; s[0] := 0;
g'(w) = if s[w] is defined
    then return s[w];
else {
    s[w] := max_j{b_j + g'(w - w_j)};
    return s[w]; }
```
Remove recursion: Dynamic programming

int s[]; s[0] := 0;
g'(w) = if s[w] is defined
    then return s[w];
else {
    s[w] := max_{j}{b_j + g'(w – w_j)};
    return s[w];
}

int s[]; s[0] := 0; s[1] := 30;
for i := 4 .. w do
    s[i] := max_{j}{b_j + s[i – w_j]};
return s[w];

\begin{align*}
g(0) &= 0 \\
g(1) &= 30, \text{ item 3} \\
g(2) &= \max\{65 + g(0) = 65, 30 + g(1) = 60\} = 65, \text{ item 1} \\
g(3) &= \max\{65 + g(1) = 95, 80 + g(0) = 80, 30 + g(2) = 95\} \\
    &= 95, \text{ item 1/3} \\
g(4) &= \max\{65 + g(2) = 130, 80 + g(1) = 110, 30 + g(3) = 125\} = 130, \text{ item 1} \\
g(5) &= \max\{65 + g(3) = 160, 80 + g(2) = 145, 30 + g(4) = 160\} = 160, \text{ item 1/3}
\end{align*}
Protein evolution
A protein is a ...

- A protein is a large complex molecule made up of one or more chains of amino acids
- Proteins perform a wide variety of activities in the cell
In the course of evolution...
Exercise #3

Let \( a = \text{AFPHQHRVP} \)
Let \( b = \text{PQVYNIMKE} \)

Suppose each generation differs from the previous by 1 residue

What is the max difference between the 2\textsuperscript{nd} generation of \( a \)?

What is the min difference between the 2\textsuperscript{nd} generation of \( a \) and \( b \)?
Therefore...

Two proteins inheriting their function from a common ancestor have very similar amino acid sequences
Sequence alignment
Why we compare sequences

• The structure of a protein defines its function
  – In order for a protein to have a specific function, it must satisfy specific structural constraints

• Protein evolves $\rightarrow$ amino acid seq changes $\rightarrow$ protein structure changes $\rightarrow$ breaks those structural constraints $\rightarrow$ protein loses function

• The more similar two proteins’ amino acid sequences are, the more likely they come from the same ancestor $\rightarrow$ the more likely they have the same structure and function
Earliest research in seq comparison

Source: Ken Sung

- Doolittle et al. (*Science*, July 1983) searched for platelet-derived growth factor (PDGF) in his own DB. He found that PDGF is similar to v-sis oncogene

```plaintext
PDGF-2  1     SLGSLTIAEPAIAECKTREVFICICRRL?DR??  34
p28sis  61    LARGKRSGLSLSVAEPAMIAECKTRTEVFEISRRLIDRTN  100
```
Sequence alignment

- Key aspect of seq comparison is seq alignment
- A seq alignment maximizes the number of positions that are in agreement in two sequences
Applications of sequence comparison

- **Infer protein function**
  - When two proteins look similar, we conjecture they come from the same ancestor and inherit the ancestor’s function (i.e. they are homologous)

- **Find evolution distance between two species**
  - Evolution modifies the DNA of species → Similarity of their genome correlates with their evolutionary distance

- **Help genome assembly**
  - Human genome project reconstructs the whole genome based on overlapping info of a huge amount of short DNA pieces
Poor sequence alignment

- Poor seq alignment shows few matched positions
  ⇒ The two proteins are not likely to be homologous

Alignment by FASTA of the sequences of amicyanin and domain 1 of ascorbate oxidase

Amicyanin

60  70  80  90  100
MPHNVHFVAGVLGEAALKPMKKEQAASLTFTTEAGTYHDCTRPHPFMRGKV

Ascorbate Oxidase

70  80  90  100  110  120
ILQRGTPWADGTASISQCAINPGETFYNNFTVDNPFTFFYHGHLGMQRSAAGLYGSL
Good sequence alignment

- Good alignment usually has clusters of extensive matched positions

⇒ The two proteins are likely to be homologous

```
>gil13476732|ref|NP_108301.11 unknown protein [Mesorhizobium loti]
gil14027493|dbj|BAB53762.11 unknown protein [Mesorhizobium loti]
Length = 105

Score = 105 bits (262), Expect = 1e-22
Identities = 61/106 (57%), Positives = 73/106 (68%), Gaps = 1/106 (0%)
```

**good match between Amicyanin and unknown M. loti protein**
Alignment:

**Simple-minded probability & score**

Let $p, q, r$ be respectively the probability of a match, a mismatch, and an indel. Then the probability of an alignment $A = (X, Y)$ is

$$prob(A) = p^m \cdot q^n \cdot r^h$$

where

$$m = |\{i \mid x'_i = y'_i \neq -\}|$$
$$n = |\{i \mid x'_i \neq y'_i, x'_i \neq -, y'_i \neq -\}|$$
$$h = |\{i \mid x'_i = -, y'_i \neq -\} \cup \{i \mid x'_i \neq -, y'_i = -\}|$$

- **Define score** $S(A)$ by simple log likelihood as
  - $S(A) = \log(prob(A)) - [m \log(s) + h \log(s)]$, with $\log(p/s) = 1$

- **Then** $S(A) = \text{#matches} - \mu \text{#mismatches} - \delta \text{#indels}$

Exercise: Derive $\mu$ and $\delta$
Global pairwise alignment:
Problem definition

• The problem of finding a global pairwise alignment is to find an alignment $A$ so that $S(A)$ is max among exponential number of possible alternatives.

• Given sequences $U$ and $V$ of lengths $n$ and $m$, then number of possible alignments is given by
  - $f(n, m) = f(n-1,m) + f(n-1,m-1) + f(n,m-1)$
  - $f(n,n) \sim (1 + \sqrt{2})^{2n+1} n^{-1/2}$
Global pairwise alignment: Dynamic programming solution

- Define an indel-similarity matrix \( s(.,.) \); e.g.,
  - \( s(x,x) = 2 \)
  - \( s(x,y) = -\mu, \) if \( x \neq y \)
- Then

Let \( U \) and \( V \) be two sequences of length \( n \) and \( m \). Then their global pairwise alignment can be extracted from the dynamic programming computation of \( S_{n,m} \), where

\[
S_{i,j} = \max \left\{ \begin{array}{l}
S_{i-1,j-1} + s(u'_i, v'_j) \\
S_{i-1,j} - \delta \\
S_{i,j-1} - \delta
\end{array} \right\}
\]

This is the basic idea of the Needleman-Wunsch algorithm
Exercise #4

• What happens when $\delta$ is large?

Let $U$ and $V$ be two sequences of length $n$ and $m$. Then their global pairwise alignment can be extracted from the dynamic programming computation of $S_{n,m}$, where

$$S_{i,j} = \max \begin{cases} 
S_{i-1,j-1} + s(u'_i, v'_j) \\
S_{i-1,j} - \delta \\
S_{i,j-1} - \delta
\end{cases}$$
Needleman-Wunsch algorithm (I)

Source: Ken Sung

- Consider two strings \( S[1..n] \) and \( T[1..m] \)
- Let \( V(i, j) \) be score of optimal alignment between \( S[1..i] \) and \( T[1..j] \)

- Basis:
  - \( V(0, 0) = 0 \)
  - \( V(0, j) = V(0, j-1) - \delta \)
    - Insert \( j \) times
  - \( V(i, 0) = V(i-1, 0) - \delta \)
    - Delete \( i \) times
Needleman-Wunsch algorithm (II)

Source: Ken Sung

• **Recurrence: For** $i>0$, $j>0$

$$V(i, j) = \max \begin{cases} V(i-1, j-1) + s(S[i], T[j]) & \text{Match/mismatch} \\ V(i-1, j) - \delta & \text{Delete} \\ V(i, j-1) - \delta & \text{Insert} \end{cases}$$

• **In the alignment, the last pair must be either**

match/mismatch, delete, insert
Example (I)

Source: Ken Sung

<table>
<thead>
<tr>
<th></th>
<th>_</th>
<th>A</th>
<th>G</th>
<th>C</th>
<th>A</th>
<th>T</th>
<th>G</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>_</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
<td>-3</td>
<td>-4</td>
<td>-5</td>
<td>-6</td>
<td>-7</td>
</tr>
<tr>
<td>A</td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>-2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>-3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>-4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>-5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>-6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>-7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Example (II)

Source: Ken Sung

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

\[ S_{1,1} = \max \begin{cases} S_{0,0} + s(A, A) & 0 + 2 \\ S_{0,1} - 1 & = \max \begin{cases} -1 - 1 & = 2 \\ S_{1,0} - 1 & \end{cases} \end{cases} \]
### Example (III)

Source: Ken Sung

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

\[
S_{1,2} = \max \begin{cases} 
S_{0,1} + s(A, G) \\
S_{0,2} - 1 \\
S_{1,1} - 1 
\end{cases}
\]

\[
\begin{align*}
S_{0,1} &= -1 + 1 = 0 \\
S_{0,2} &= -2 - 1 = -3 \\
S_{1,1} &= 2 - 1 = 1
\end{align*}
\]

\[
S_{1,2} = \max \begin{cases} 
0 & \quad -1 + 1 = 0 \\
-3 & \quad -2 - 1 = -3 \\
1 & \quad 2 - 1 = 1
\end{cases}
\]

\[
S_{1,2} = 1
\]
Example (IV) / Exercise #5
Source: Ken Sung

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

Can you tell from these entries what are the values of \( s(A,G) \), \( s(A,C) \), \( s(A,A) \), etc.?
**Example (V) / Exercise #6**

Source: Ken Sung

What is the alignment corresponding to this?

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

Source: Ken Sung

Create the table $V[0..n,0..m]$ and $P[1..n,1..m]$;
$V[0,0] = 0$;
For $j=1$ to $m$, set $V[0,j] := v[0,j-1] - \delta$;
For $i=1$ to $n$, set $V[i,0] := V[i-1,0] - \delta$;
For $j=1$ to $m$
  For $i = 1$ to $n$
    set $V[i,j] := V[i,j-1] - \delta$;
    set $P[i,j] := (0, -1)$;
    if $V[i,j] < V[i-1,j] - \delta$ then
      set $V[i,j] := V[i-1,j] - \delta$;
      set $P[i,j] := (-1, 0)$;
    if $(V[i,j] < V[i-1, j-1] + s(S[i], T[j]))$ then
      set $V[i,j] := V[i-1, j-1] + s(S[i], T[j])$;
      set $P[i,j] := (-1, -1)$;
  }
Backtracking $P[n,m]$ to $P[0,0]$ to find optimal alignment;
Analysis

Source: Ken Sung

• We need to fill in all entries in the $n \times m$ matrix
• Each entry can be computed in $O(1)$ time

$\Rightarrow$ **Time complexity** $= O(nm)$

$\Rightarrow$ **Space complexity** $= O(nm)$

Exercise: Write down the memoized version of Needleman-Wunsch. What is its time/space complexity?
Problem on speed

Source: Ken Sung

- **Aho, Hirschberg, Ullman 1976**
  - If we can only compare whether two symbols are equal or not, the string alignment problem can be solved in $\Omega(nm)$ time

- **Hirschberg 1978**
  - If symbols are ordered and can be compared, the string alignment problem can be solved in $\Omega(n \log n)$ time

- **Masek and Paterson 1980**
  - Based on Four-Russian’s paradigm, the string alignment problem can be solved in $O(nm/\log 2 n)$ time

- Let $d$ be the total number of inserts and deletes. Thus $0 \leq d \leq n+m$. If $d$ is smaller than $n+m$, can we get a better algorithm? Yes!
O(dn)-time algorithm

Source: Ken Sung

- The alignment should be inside the 2d+1 band

⇒ No need to fill-in the lower and upper triangle

⇒ Time complexity: O(dn)
Example

\[ d=3 \]

A\_CAATCC
AGCA\_TGC

<table>
<thead>
<tr>
<th></th>
<th>_</th>
<th>A</th>
<th>G</th>
<th>C</th>
<th>A</th>
<th>T</th>
<th>G</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>_</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
<td>-3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>-1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>-2</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>-3</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>-2</td>
<td>0</td>
<td>3</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>2</td>
<td>5</td>
<td>5</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Exercise #7 / Recursive equation for $O(dn)$-time algo

$$v(i, j) = \max \begin{cases} 
  v(i - 1, j - 1) + s(S[i], S[j]) \\
  v(i - 1, j) - \delta, & \text{if } |i - j| < d \\
  v(i, j - 1) - \delta, & \text{if } |i - j| < d
\end{cases}$$

Write down the base cases, the memoized version, and the non-recursive version.
Problem on space

• Dynamic programming requires $O(mn)$ space

• When we compare two very long sequences, space may be the limiting factor

• Can we solve the string alignment problem in linear space?
Easy, if no need to recover alignment

- When filling row 4, it depends only on row 3
  - No need to keep rows 1 and 2
- I.e., we only need to keep two rows

⇒ “Cost only” algo

<table>
<thead>
<tr>
<th></th>
<th>_</th>
<th>A</th>
<th>G</th>
<th>C</th>
<th>A</th>
<th>T</th>
<th>G</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>_</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
<td>-3</td>
<td>-4</td>
<td>-5</td>
<td>-6</td>
<td>-7</td>
</tr>
<tr>
<td>A</td>
<td>-1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
<td>-3</td>
<td>-4</td>
</tr>
<tr>
<td>C</td>
<td>-2</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>A</td>
<td>-3</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>A</td>
<td>-4</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>T</td>
<td>-5</td>
<td>-2</td>
<td>-2</td>
<td>0</td>
<td>3</td>
<td>6</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>C</td>
<td>-6</td>
<td>-3</td>
<td>-3</td>
<td>0</td>
<td>2</td>
<td>5</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>C</td>
<td>-7</td>
<td>-4</td>
<td>-4</td>
<td>-1</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>7</td>
</tr>
</tbody>
</table>
Recovering alignment in $O(n+m)$ space

- Use cost-only algo to find mid-point of alignment
- Divide the problem into two halves
- Recursively deduce alignments for the two halves
How to find mid-point

\[ V(S[1..n], T[1..m]) = \max_{0 \leq j \leq m} \left\{ V(S[1..\frac{n}{2}], T[1..j]) + V(S[\frac{n}{2} + 1..n], T[j + 1..m]) \right\} \]

- **Do cost-only dynamic programming for 1st half**
  - I.e., find \( V(S[1..n/2], T[1..j]) \) for all \( j \)

- **Do cost-only dynamic programming for 2nd half**
  - I.e., find \( V(S[n/2+1..n], T[j+1..m]) \) for all \( j \)

- **Determine \( j \) which maximizes the sum above**
Example

**Step 1**

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

**Step 2**

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

**Step 4: Recursive on subproblems**

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

**Step 3**

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

• **Space**
  – \( O(m) \) working memory for finding mid-point
  – Once mid-point is found, can free working memory \( \rightarrow \) In each recursive call, we only need to store the alignment path
  – Alignment subpaths are disjoint \( \rightarrow \) total space required is \( O(n+m) \)

• **Time?** This one is for you to think about 😊
Global pairwise alignment:
More Realistic Handling of Indels

• In Nature, indels of several adjacent letters are not the sum of single indels, but the result of one event

• So reformulate as follows:

Let $g(k)$ be the indel weight for an indel of $k$ letters. Typically, $g(k) \leq k \cdot g(1)$. Let $U$ and $V$ be two sequences of length $n$ and $m$. Then their global pairwise alignment can be extracted from the dynamic programming computation of $S_{n,m}$, where

$$S_{0,0} = 0, \quad S_{0,j} = -g(j), \quad S_{i,0} = -g(i)$$

$$S_{i,j} = \max \left\{ S_{i-1,j-1} + s(u'_i, v'_j), \max_{1 \leq k \leq j} \{ S_{i,j-k} - g(k) \}, \max_{1 \leq k \leq i} \{ S_{i-k,j} - g(k) \} \right\}$$
Gap penalty

Source: Ken Sung

- $g(q) : \mathbb{N} \rightarrow \mathbb{R}$ is the penalty of a gap of length $q$
- Note $g()$ is subadditive, i.e, $g(p+q) \leq g(p) + g(q)$

- If $g(k) = \alpha + \beta k$, the gap penalty is called affine
  - A penalty ($\alpha$) for initiating the gap
  - A penalty ($\beta$) for the length of the gap
N-W algo w/ general gap penalty

Source: Ken Sung

- **Global alignment of S[1..n] and T[1..m]:**
  - Denote $V(i, j)$ be the score for global alignment between $S[1..i]$ and $T[1..j]$
  - **Base cases:**
    - $V(0, 0) = 0$
    - $V(0, j) = g(j)$
    - $V(i, 0) = g(i)$
N-W algo w/ general gap penalty (II)

Source: Ken Sung

• **Recurrence for** $i>0$ **and** $j>0$,

$$V(i, j) = \max \begin{cases} 
V(i-1, j-1) + \delta(S[i], T[j]) \\
\max_{0 \leq k \leq j-1} \{V(i, k) + g(j-k)\} \\
\max_{0 \leq k \leq i-1} \{V(k, j) + g(i-k)\}
\end{cases}$$

- **Match/mismatch**
- **Insert** $T[k+1..j]$
- **Delete** $S[k+1..i]$
Analysis

Source: Ken Sung

• We need to fill in all entries in the $n \times m$ table

• Each entry can be computed in $O(\max\{n, m\})$ time

$\Rightarrow$ Time complexity = $O(nm \max\{n, m\})$

$\Rightarrow$ Space complexity = $O(nm)$
Variations of pairwise alignment

- Fitting a “short” seq to a “long” seq

- Indels at beginning and end are not penalized

- Find “local” alignment

- Find $i, j, k, l$, so that
  - $S(A)$ is maximized,
  - $A$ is alignment of $u_i \ldots u_j$ and $v_k \ldots v_l$
Local alignment

Source: Ken Sung

- Given two long DNAs, both of them contain the same gene or closely related gene
  - Can we identify the gene?

- Local alignment problem: Given two strings $S[1..n]$ and $T[1..m]$, among all substrings of $S$ and $T$, find substrings $A$ of $S$ and $B$ of $T$ whose global alignment has the highest score
Brute-force solution

Source: Ken Sung

• **Algorithm:**
  – For every substring A of S, for every substring B of T, compute the global alignment of A and B
  – Return the pair (A, B) with the highest score

• **Time:**
  – There are $n^2$ choices of A and $m^2$ choices of B
  – Global alignment computable in $O(nm)$ time
  – In total, time complexity = $O(n^3m^3)$

• **Can we do better?**
Some background / Exercise #8

Source: Ken Sung

• **X** is a suffix of **S[1..n]** if **X** = **S[k..n]** for some **k ≥ 1**
• **X** is a prefix of **S[1..n]** if **X** = **S[1..k]** for some **k ≤ n**

• **E.g.**
  – Consider **S[1..7]** = ACCGATT
  – ACC is a prefix of S, GATT is a suffix of S
  – Empty string is both prefix and suffix of S

Which other string is both a prefix and suffix of S?
Dynamic programming for local alignment problem

Source: Ken Sung

• Define $V(i, j)$ be max score of global alignment of $A$ and $B$ over
  – all suffixes $A$ of $S[1..i]$ and
  – all suffixes $B$ of $T[1..j]$

• Then, score of local alignment is
  – $\max_{i,j} V(i,j)$
Smith-Waterman algorithm

Source: Ken Sung

• Basis:

\[ V(i, 0) = V(0, j) = 0 \]

• Recursion for \( i > 0 \) and \( j > 0 \):

\[
V(i, j) = \max \begin{cases} 
0 & \text{Ignore initial segment} \\
V(i-1, j-1) + s(S[i], T[j]) & \text{Match/mismatch} \\
V(i-1, j) - \delta & \text{Delete} \\
V(i, j-1) - \delta & \text{Insert} 
\end{cases}
\]
Example (I)

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

Score for match = 2
Score for insert, delete, mismatch = −1

Source: Ken Sung
### Example (II) / Exercise #9

Source: Ken Sung

<table>
<thead>
<tr>
<th></th>
<th>_</th>
<th>C</th>
<th>T</th>
<th>C</th>
<th>A</th>
<th>T</th>
<th>G</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>_</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>A</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>T</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>2</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>
<tr>
<td>C</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>
</tbody>
</table>
Analysis / Exercise #10

Source: Ken Sung

- Need to fill in all entries in the $n \times m$ matrix
- Each entries can be computed in $O(1)$ time
- Finally, finding the entry with the max value

$\Rightarrow$ Time complexity = ??

$\Rightarrow$ Space complexity = $O(nm)$
Local alignment with at most d indels

1. The modified algorithm is as follows:

\[
H(i, j, k) = \begin{cases} 
0, & \text{if } i = 0 \text{ or } j = 0 \text{ or } k < 0 \\
\max \left\{ H(i-1, j-1, k) + w(a_i, b_j) \right. \\
\left. H(i-1, j, k-1) + w(a_i, -) \right. \\
\left. H(i, j-1, k-1) + w(-, b_j) \right. \right\} \\
\end{cases} \\
\text{1 \leq i \leq m, 1 \leq j \leq n, 0 \leq k \leq d}
\]

Where:
- \(a, b\) are the string compared
- \(m = \text{length of } a\)
- \(n = \text{length of } b\)
- \(H(i, j, k)\) is the maximum similarity score between \(a[1..i]\) and \(b[1..j]\) with \(k\) indel.
- \(w(c, d)\) as the match scoring scheme

Then find \(\max(H(i, j, k))\) with \(1 \leq i \leq m, 1 \leq j \leq n, 1 \leq k \leq d\)

2. This is just a modification of Smith-Waterman where indel usage is tracked in the form of \(k\). Since \(k \leq d\) then it is clear that none of the values use more than \(d\) indels.

3. Since there is \(dmn\) values we have to calculate, The time complexity is \(O(dmn)\).

- Cf. global alignment with at most d index has time complexity \(O(dn)\)
Photos

Limsoon & Temple Smith  Ken & Michael Waterman
Scoring function
Scoring function for DNA

- For DNA, since we only have 4 nucleotides, the score function is simple
  - BLAST matrix
  - Transition-transversion matrix: Give mild penalty for replacing purine by purine. Similar for replacing pyrimidine by pyrimidine

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
<th>G</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>5</td>
<td>-4</td>
<td>-4</td>
<td>-4</td>
</tr>
<tr>
<td>C</td>
<td>-4</td>
<td>5</td>
<td>-4</td>
<td>-4</td>
</tr>
<tr>
<td>G</td>
<td>-4</td>
<td>-4</td>
<td>5</td>
<td>-4</td>
</tr>
<tr>
<td>T</td>
<td>-4</td>
<td>-4</td>
<td>-4</td>
<td>5</td>
</tr>
</tbody>
</table>

BLAST Matrix

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
<th>G</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>-5</td>
<td>-1</td>
<td>-5</td>
</tr>
<tr>
<td>C</td>
<td>-5</td>
<td>1</td>
<td>-5</td>
<td>-1</td>
</tr>
<tr>
<td>G</td>
<td>-1</td>
<td>-5</td>
<td>1</td>
<td>-5</td>
</tr>
<tr>
<td>T</td>
<td>-5</td>
<td>-1</td>
<td>-5</td>
<td>1</td>
</tr>
</tbody>
</table>

Transition-Transversion Matrix
Scoring function for protein

• Commonly, it is devised based on two criteria:
  – Chemical/physical similarity
  – Observed substitution frequencies
Scoring function for protein using physical/chemical properties

• An amino acid is more likely to be substituted by another if they have similar property [Karlin & Ghandour, PNAS, 82:8597, 1985]

• The score matrices can be derived based on hydrophobicity, charge, electronegativity, & size

• E.g., give higher score for substituting nonpolar amino acid to another nonpolar amino acid
Scoring function for protein based on statistical model

• Most often used approaches

• Two popular matrices:
  – Point Accepted Mutation (PAM) matrix
  – BLOSUM

• Both methods define the score as the log-odds ratio between the observed substitution rate and the expected substitution rate

Point Accepted Mutation (PAM)

• PAM was developed by Dayhoff (1978)

• A point mutation means substituting one residue by another
  – It is called an accepted point mutation if the mutation does not change the protein’s function or is not fatal

• Two sequence S1 and S2 are said to be 1 PAM diverged if a series of accepted point mutations can convert S1 to S2 with an average of 1 accepted point mutation per 100 residues
PAM matrix by example (I)

• Ungapped alignment is constructed for high similarity amino acid sequences (usually >85%)

• Below is a simplified gap-free global multiple alignment of some highly similar amino acid seqs
  – IACGCTAFK
  – IGCGCTAFK
  – LACGCTAFK
  – IGCGCTGFK
  – IGCGCTLFK
  – LASGCTAFK
  – LACACTAFK
  – LACACTAFK
PAM matrix by example (II)

• Build the phylogenetic tree for the sequences

```
- IACGCTAFK
  - IGCGCTAFK
    - IGCGCTGFK
    - IGCGCCTLFK
  - IACGCTAFK
    - IGCGCTAFK
      - ICGCTGFK
      - ICGCCTLFK
  - LACGCTAFK
    - LASGCTAFK
      - GACCTAFK
    - LACACTAFK

A→G
I→L
A→G
A→L
C→S
G→A
```
PAM-1 matrix

\[ \delta(a, b) = \log_{10} \frac{O_{a,b}}{E_{a,b}} \]

- \( O_{a,b} \) and \( E_{a,b} \) are observed and expected freq
  - \( O_{a,a} = 99/100 \), as PAM-1 assumes 1 mutation per 100 residues
  - For \( a \neq b \), \( O_{a,b} = \frac{F_{a,b}}{(100 \sum_x \sum_y F_{x,y})} \) where \( F_{a,b} \) is freq of substituting \( a \) by \( b \) or \( b \) by \( a \)
  - \( E_{a,b} = f_a \times f_b \) where \( f_x \) is # of \( x \) divided by total residues

- E.g., \( F_{A,G} = 3, F_{A,L} = 1, f_A = f_G = 10/63 \), then \( O_{A,G} = \frac{3}{(100 \times 2 \times 6)} = 0.0025 \), \( E_{A,G} = (10/63)(10/63) = 0.0252 \), \( \delta(A,G) = \log (0.0025 / 0.0252) = \log (0.09925) = -1.0034 \)
Exercise #11

• \( O_{A,G} = \frac{3}{(100 \times 2 \times 6)} \)

• Where do the 2 and 6 come from?

PAM-1 matrix

\[
\delta(a, b) = \log_{10} \frac{O_{a,b}}{E_{a,b}}
\]

• \( O_{a,b} \) and \( E_{a,b} \) are observed and expected freq
  – \( O_{a,a} = \frac{99}{100} \), as PAM-1 assumes 1 mutation per 100 residues
  – For \( a \neq b \), \( O_{a,b} = \frac{F_{a,b}}{(100 \sum_x \sum_y F_{x,y})} \) where \( F_{a,b} \) is freq of substituting \( a \) by \( b \) or \( b \) by \( a \)
  – \( E_{a,b} = f_a \times f_b \) where \( f_x \) is \# of \( x \) divided by total residues

• E.g., \( F_{A,G} = 3, F_{A,L} = 1, f_A = f_G = \frac{10}{63}, \) then \( O_{A,G} = \frac{3}{(100 \times 2 \times 6)} = 0.0025, E_{A,G} = \frac{10}{63} \times \frac{10}{63} = 0.0252, \delta(A,G) = \log \frac{0.0025}{0.0252} = \log \frac{0.09925}{0.0252} = -1.0034 \)
PAM-n matrix

Let $M_{a,b} = O_{a,b} / f_a$ be prob that $a$ is mutated to $b$

$M^n(a,b)$ is prob that $a$ is mutated to $b$ after $n$ mutations

PAM-n matrix is created by extrapolating PAM-1

PAM-n matrix is computed as follows.

- At time $t$, suppose the residue is $a$
- At time $t+1$, prob that it becomes $j$ is $M(a,b)$
- At time $t+2$, prob that it becomes $j$ is $M^2(a,b)$
- …
- At time $t+n$, prob that it becomes $j$ is $M^n(a,b)$

$(a,b)$ entry of PAM-n matrix is $\log(f_a M^n(a,b)/f_a f_b) = \log(M^n(a,b)/f_b)$
BLOSUM (BLOck SUbstitution Matrix)

- PAM did not work well for aligning evolutionarily divergent sequences since the matrix is generated by extrapolation

- Henikoff and Henikoff (1992) proposed BLOSUM

- Unlike PAM, BLOSUM matrix is constructed directly from the observed alignment (instead of extrapolation)
Generating conserved blocks

• In BLOSUM, the input is a set of multiple alignments for nonredundant groups of protein families

• Based on PROTOMAT, blocks of nongapped local alignments are derived

• Each block represents a conserved region of a protein family
Extract frequencies from blocks

- From all blocks, we count the frequency $f_a$ for each amino acid residue $a$.
- For any two amino acid residues $a$ and $b$, we count the frequency $p_{ab}$ of aligned pair of $a$ and $b$.

For example,
- ACGCTAFKI
  - GCGCTAFKI
  - ACGCTAFKL
  - GCGCTGFKI
  - GCGCTLFKI
  - ASGCTAFKL
  - ACACTAFKL

- There are $7 \times 9 = 63$ residues, including 9’s A and 10’s G. Hence, $F_A = 9/63$, $F_G = 10/63$.
- There are $9 \times \binom{7}{2} = 189$ aligned residue pairs, including 23 (A,G) pairs. Hence, $p_{AG} = 23 / 189$. 
BLOSUM scoring function

• For each pair of aligned residues a and b, the alignment score \( \delta(a,b) = (1/\lambda)(\ln p_{ab}/(p_ap_b)) \)
  – \( p_{ab} \) is prob that a and b are observed to align together
  – \( p_a \) and \( p_b \) are freq of residues a and b
  – \( \lambda \) is a normalization constant

• Example: \( p_L=0.099, p_A=0.074, p_{AL} = 0.0044 \). With \( \lambda=0.347 \), \( \delta(A,L) = -1.47 \)
What is BLOSUM 62?

- To reduce multiple contributions to amino acid pair freq from the most closely related members of a family, similar seqs are merged within block.
- BLOSUM p matrix is created by merging seqs with ≥p% similarity.

**Example**

- AVAAA, AVAAA, AVAAA, AVLAA, VVAAL
- First 4 seqs have ≥80% similarity. Similarity of last seq with the other 4 sequences is <62%.
- For BLOSUM 62, we group first 4 seqs and get AV[A_{0.75}L_{0.25}]AA, VVAAL. Then p_{AV} = 1/5, p_{AL} = (0.25 + 1)/5.
BLOSUM vs PAM

- BLOSUM 80 ≈ PAM 1
- BLOSUM 62 ≈ PAM 120
- BLOSUM 45 ≈ PAM 250

- BLOSUM 62 is the default matrix for BLAST 2.0
Multiple sequence alignment
What is a domain

• **A domain** is a component of a protein that is self-stabilizing and folds independently of the rest of the protein chain
  – Not unique to protein products of one gene; can appear in a variety of proteins
  – Play key role in the biological function of proteins
  – Can be "swapped" by genetic engineering between one protein and another to make chimeras

• **May be composed of one, more than one, or not any structural motifs** (often corresponding to active sites)
Discovering domain and active sites

>gi|475902|emb|CAA83657.1| protein-tyrosine-phosphatase alpha
MDLWFFVLLLSGSLISVGATNVTTEPPTTVPTSTRIPTKAPTAAPGGTTTPRVSSLNVSSPMTTAPASE
PPTTTATSISPNATTICLNSPSTPGTSVPTSPAPVAISLPPPSTPSALLTAPSTEAMERMENMTSATVTTQE
TSSASHNGNNSDRDDDRPIIAVMVALSLVLVVIFIIIVLYMLRFKSKYQQAGSHSNSFRPRLPNGRTDDAPQS
MPLLARSPSTNRKYPPLPVDKLEEEEINRRIGDDNGLKFREEFNLAPACPIQATCEAASKEENKEKNRYVIN
LPYDHSLRHLTPVEGVPDHYINTSFINSYQKEKNFIAAQQPKEETVNDFWRMIWEQNTATIVMVTNLKE
RKECKCAQYWPQGCWTVNGIRVSVEDVTVLVDYTVRFKCIQQGVDVTNKPPQRLLTQFHTSWPFGVFP
FTPIGMKLKLKKVKTNCNPQYAGAIVVHCASGVGRTGFIIVIDAMLDMMHAKVVDVYGFVSRIARQRCQM
VQTDQYFYFQIALEHYLYGDESTLELVSTLEIHLQKIQNKVPGTSSNGLEEFKKLTSIKIQNDKMRTGN
LPANMKKNRLQIPYEYFNRVIPVKRGEENTDYVNASIDGYRRTPTCQPVRPVQHTIEDFWRMIWEWK
SCSIVMLTELEERGQECKCAQYWPSDGVSYSDDINVELKKEEEECCYTVDVLVLVTNRENSRQIRQFHFH
GWPEVVGIPSDGKGMNIIIAAVQKQQQSGNHPMHCHCSAGAGRTGFCALSTVLERVAEGILDFQTVK
SLRLQRPHMVQETLEQYEFICYKVVQVEYIDAFSDYANFK

* How do we find the domain and associated active sites in the protein above?
Domain/active sites as emerging patterns

• How to discover active site and/or domain?

• If you are lucky, domain has already been modelled
  – BLAST, HMMPFAM, …

• If you are unlucky, domain not yet modelled
  – Find homologous seqs
  – Do multiple alignment of homologous seqs
  – Determine conserved positions
⇒ Emerging patterns relative to background
⇒ Candidate active sites and/or domains
In the course of evolution…
Multiple alignment: Example

- Multiple seq alignment maximizes number of positions in agreement across several seqs
- Seqs belonging to same “family” usually have more conserved positions in a multiple seq alignment

Conserved sites
Multiple alignment: Naïve approach

- Let $S(A)$ be the score of a multiple alignment $A$. The optimal multiple alignment $A$ of sequences $U_1, \ldots, U_r$ can be extracted from the following dynamic programming computation of $S_{m_1, \ldots, m_r}$:

$$S_{m_1, \ldots, m_r} = \max_{\epsilon_1 \in \{0,1\}, \ldots, \epsilon_r \in \{0,1\}} \left\{ S_{m_1 - \epsilon_1, \ldots, m_r - \epsilon_r} + s(\epsilon_1 \cdot u_1^l, m_1, \ldots, \epsilon_r \cdot u_r^l, m_r) \right\}$$

where

$$\epsilon_i \cdot a = \begin{cases} a & \text{if } \epsilon_i = 1 \\ - & \text{if } \epsilon_i = 0 \end{cases}$$

- This requires $O(2^r)$ steps

Exercise for the Brave:
Propose a practical approximation
Popular tools for sequence comparison:
FASTA, BLAST, Pattern Hunter
Scalability

- Increasing # of sequenced genomes: yeast, human, rice, mouse, fly, ...
- S/w must be “linearly” scalable to large datasets
Database search

- Consider a database D of genomic sequences (or protein sequences)

- Given a query string Q,
  - Look for string S in D which is the closest match to the query string Q
  - Two meanings for closest match:
    - S and Q has a semi-global alignment (forgive the spaces at the two ends of Q)
    - S and Q have a local alignment
Goodness of a search algorithm

• **Sensitivity**
  – Ability to detect “true positive”
  – Measured as the probability of finding the match given the query and the database sequence has only x% similarity

• **Specificity**
  – Ability to reject “false positive”

• **A good search algorithm should be both sensitive and specific**
Need heuristics for sequence comparison

- Time complexity for optimal alignment is $O(n^2)$, where $n$ is seq length

- Given current size of seq databases, use of optimal algorithms is not practical for database search

- Heuristic techniques:
  - BLAST
  - FASTA
  - Pattern Hunter
  - MUMmer, ...

- Speed up:
  - 20 min (optimal alignment)
  - 2 min (FASTA)
  - 20 sec (BLAST)

Exercise: Describe MUMer
Basic idea: Indexing & filtering

• Good alignment includes short identical, or similar fragments

⇒ Break entire string into substrings, index the substrings

⇒ Search for matching short substrings and use as seed for further analysis

⇒ Extend to entire string find the most significant local alignment segment
BLAST in 3 steps

- **Similarity matching of words (3 aa’s, 11 bases)**
  - No need identical words

- **If no words are similar, then no alignment**
  - Won’t find matches for very short sequences

- **MSP: Highest scoring pair of segments of identical length.** A segment pair is locally maximal if it cannot be improved by extending or shortening the segments

- **Find alignments w/ optimal max segment pair (MSP) score**
  - Gaps not allowed

- **Homologous seqs will contain a MSP w/ a high score; others will be filtered out**
BLAST in 3 steps

Step 1
- **For the query, find the list of high scoring words of length w**

```
Query Sequence of length L
```

```
Maximum of L-w+1 words (typically w = 3 for proteins)
```

```
For each word from the query sequence find the list of words that will score at least T when scored using a pair-score matrix (e.g. PAM 250).
```

Image credit: Barton
BLAST in 3 steps

Step 2

• Compare word list to db & find exact matches

Image credit: Barton
BLAST in 3 steps

Step 3

- For each word match, extend alignment in both directions to find alignment that score greater than a threshold $s$

Image credit: Barton
Spaced seeds

• **111010010100110111** is an example of a spaced seed model with
  – 11 required matches (weight=11)
  – 7 “don’t care” positions

```
GAGTACTCAACACCAACATTAAGTGCGCAATGGAAAAT...
  || || || || || || || || || || || || || || ||
GAATACTCAACAGCAACACACTAATGGCAGCAGAAAAT...
```

• **11111111111** is the BLAST seed model for comparing DNA seqs

```
111010010100110111
```
Observations on spaced seeds

- **Seed models w/ different shapes can detect different homologies**
  - the 3rd base in a codon “wobbles” so a seed like 110110110… should be more sensitive when matching coding regions

⇒ **Some models detect more homologies**
  - More sensitive homology search
  - PatternHunter I

⇒ **Use >1 seed models to hit more homologies**
  - Approaching 100% sensitive homology search
  - PatternHunter II

Exercise: Why does the 3rd base wobble?
PatternHunter I
Ma et al., *Bioinformatics* 18:440-445, 2002

- BLAST’s seed usually uses more than one hits to detect one homology
  ⇒ Wasteful

- Spaced seeds uses fewer hits to detect one homology
  ⇒ Efficient

\[
\begin{align*}
\text{TTGACCTCAC} \text{C\_?} \\
| | | | | | | | | ? \\
\text{TTGACCTCAC}\_? \\
11111111111 \\
11111111111 \\
\end{align*}
\]

1/4 chances to have 2nd hit next to the 1st hit

\[
\begin{align*}
\text{CAA\_?A\_?A\_?C\_?TA\_TGG\_?} \\
\text{CAA\_?A\_?A\_?C\_?TA\_TGG\_?} \\
111010010100110111 \\
11110100101001100001 \\
\end{align*}
\]

1/4^6 chances to have 2nd hit next to the 1st hit
Proposition. The expected number of hits of a weight-$W$ length-$M$ model within a length-$L$ region of similarity $p$ is $(L - M + 1) * p^W$

Proof.
For any fixed position, the prob of a hit is $p^W$.
There are $L - M + 1$ candidate positions.
The proposition follows.
Implication

- **For \( L = 1017 \)**
  - BLAST seed expects \((1017 - 11 + 1) \times p^{11} = 1007 \times p^{11}\) hits
  - But \(\sim 1/4\) of these overlap each other. So likely to have only \(\sim 750 \times p^{11}\) distinct hits
  - Our example spaced seed expects \((1017 - 18 + 1) \times p^{11} = 1000 \times p^{11}\) hits
  - But only \(1/4^6\) of these overlap each other. So likely to have \(\sim 1000 \times p^{11}\) distinct hits

Spaced seeds likely to be more sensitive & more efficient
Sensitivity of PatternHunter I

Image credit: Li
Speed of PatternHunter I

- Mouse Genome Consortium used PatternHunter to compare mouse genome & human genome

- PatternHunter did the job in a 20 CPU-days --- it would have taken BLAST 20 CPU-years!

How to increase sensitivity?

• **Ways to increase sensitivity:**
  – “Optimal” seed
  – Reduce weight by 1
  – Increase number of spaced seeds by 1

• **Intuitively, for DNA seq,**
  – Reducing weight by 1 will increase number of matches 4 folds
  – Doubling number of seeds will increase number of matches 2 folds

• **Is this really so?**
How to increase sensitivity?

- **Ways to increase sensitivity:**
  - “Optimal” seed
  - Reduce weight by 1
  - Increase number of spaced seeds by 1

- **For** \( L = 1017 \) & \( p = 50\% \)
  - 1 weight-11 length-18 model expects \( \frac{1000}{2^{11}} \) hits
  - 2 weight-12 length-18 models expect \( 2 \times \frac{1000}{2^{12}} = \frac{1000}{2^{11}} \) hits

\[ \Rightarrow \text{When comparing regions w/ >50\% similarity, using 2 weight-12 spaced seeds together is more sensitive than using 1 weight-11 spaced seed!} \]

**Exercise #12:** Proof this claim

---

**Proposition.** The expected number of hits of a weight-\( W \) length-\( M \) model within a length-\( L \) region of similarity \( p \) is \((L - M + 1) \times p^W\).

**Proof.** For any fixed position, the prob of a hit is \( p^W \). There are \( L - M + 1 \) positions. The proposition follows.
• Idea
  – Select a group of spaced seed models
  – For each hit of each model, conduct extension to find a homology

• Selecting optimal multiple seeds is NP-hard

• Algorithm to select multiple spaced seeds
  – Let A be an empty set
  – Let s be the seed such that $A \cup \{s\}$ has the highest hit probability
  – $A = A \cup \{s\}$
  – Repeat until $|A| = K$

• Computing hit probability of multiple seeds is NP-hard

But see also Ilie & Ilie, “Multiple spaced seeds for homology search”, *Bioinformatics*, 23(22):2969-2977, 2007
Sensitivity of PatternHunter II

- Solid curves: Multiple (1, 2, 4, 8, 16) weight-12 spaced seeds
- Dashed curves: Optimal spaced seeds with weight = 11, 10, 9, 8

⇒ “Double the seed number” gains better sensitivity than “decrease the weight by 1”
Expts on real data

- 30k mouse ESTs (25Mb) vs 4k human ESTs (3Mb)
  - downloaded from NCBI genbank
  - “low complexity” regions filtered out

- SSearch (Smith-Waterman method) finds “all” pairs of ESTs with significant local alignments

- Check how many percent of these pairs can be “found” by BLAST and different configurations of PatternHunter II
In fact, at 80% similarity, 100% sensitivity can be achieved using 40 weight-9 seeds.
Farewell to Supercomputer Age of sequence comparison!

**Computer:** PIII 700Mhz Redhat 7.1, 1G main memory

<table>
<thead>
<tr>
<th>Sequence Length</th>
<th>Blastn</th>
<th>PatternHunter</th>
</tr>
</thead>
<tbody>
<tr>
<td>816k vs 580k</td>
<td>47 sec</td>
<td>9 sec</td>
</tr>
<tr>
<td>4639k vs 1830k</td>
<td>716 sec</td>
<td>44 sec</td>
</tr>
<tr>
<td>20M vs 18M</td>
<td>out of memory</td>
<td>13 min</td>
</tr>
</tbody>
</table>

**Time required to compare Arabidopsis chromosomes 2 and 4**

- Megablast: 20000 Seconds
- PatternHunter: 0 Seconds

**Memory required to compare Arabidopsis chromosomes 2 and 4**

- Megablast: 1000 MB
- PatternHunter: 200 MB

Image credit: Bioinformatics Solutions Inc
About the inventor: Ming Li

- Ming Li
  - Canada Research Chair Professor of Bioinformatics, University Professor, Univ of Waterloo
  - Fellow, Royal Society of Canada. Fellow, ACM. Fellow, IEEE
Concluding remarks
What have we learned?

• **General methodology**
  – Dynamic programming

• **Dynamic programming applications**
  – Pairwise Alignment
    • Needleman-Wunsch global alignment algorithm
    • Smith-Waterman local alignment algorithm
  – Multiple Alignment

• **Important tactics**
  – Indexing & filtering (BLAST)
  – Spaced seeds (Pattern Hunter)
Any question?
Acknowledgements

- Some slides on popular sequence alignment tools are based on those given to me by Bin Ma and Dong Xu

- Some slides on Needleman-Wunsch, Smith-Waterman, and scoring functions are based on those given to me by Ken Sung
References


References


- M. Li et al. “PatternHunter II: Highly sensitive and fast homology search”, *GIW*, 164-175, 2003