For written notes on this lecture, please read chapter 10 of *The Practical Bioinformatician*

CS2220: Introduction to Computational Biology
Lecture 6: Essence of Sequence Comparison

Lisa Tucker-Kellogg
4 March 2010
Most slides the same as 6-Mar-2009 (Prof. Wong)
Plan

• Dynamic Programming
• String Comparison

• Sequence Alignment
  – Pairwise Alignment
    • Needleman-Wunsch global alignment algorithm
    • Smith-Waterman local alignment algorithm
  – Multiple Alignment

• Popular tools
  – FASTA, BLAST, Pattern Hunter
What is Dynamic Programming?
What is Dynamic Programming?

• A poster child for why programmers should have some formal education in computer science

• A good way to find the best solution to certain types of problems
  – when there are discrete, finite decisions;
  – when the arrangement can be broken into phases;
  – when there is independence between the cost/benefit of each sub-decision
The 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 so as to maximize the total benefit?
Formulation of a Solution

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

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

Why is $g(w)$ optimal?

• Where
  – $w_j$ and $b_j$ be weight and benefit for item $j$
  – $g(w)$ be max benefit that can be gained from a $w$-pound knapsack
An Example: Direct Recursive Evaluation

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

<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>

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

Find the error
“Memoize” to avoid recomputation

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];
    }

<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)}
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];
}
Characteristics of Dynamic Programming

- Problem can be divided into stages with a decision required at each stage
- Each stage has a # of states associated
- Decision at one stage transforms one state into a state in the next stage
- Given current state, the optimal decision for each remaining states does not depend on next states or decisions
- There is a recursive relationship that identifies the optimal decision for stage \( j+1 \), given stage \( j \) has already been solved
- The initial stages must be solvable by themselves

Exercise: What is a stage in the Knapsack problem?

Exercise: What is a state in the Knapsack problem?

E.g., \( g(2) \) doesn't depend on \( g(3) \)

E.g., \( g(0) = 0 \)
Sequence Alignment
Motivations for Sequence Comparison

• DNA is blue print for living organisms

⇒ Evolution causes mutations=changes in DNA

⇒ By comparing DNA seqs (or protein seqs) we can infer evolutionary relationships betw seqs w/o knowledge of the evolutionary events themselves

(Be careful not to use wordings that imply you know what happened during evolution.)

• Sequence similarity is a foundation concept for inferring what the sequences do. Why?
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

\[
\begin{align*}
\text{PDGF-2} & \quad 1 \quad \text{SLGSLTIAEPAMIAECKTREEVFICICRRL?DR??} \quad 34 \\
p28sis & \quad 61 \quad \text{LARGKRSLGSLSVAEPAMIAECKTRTEVFIEISRRLIDRTN} \\
& \quad 100
\end{align*}
\]
How To Define Sequence Similarity?

- Hamming distance – among the most common ways in computer science for measuring similarity between two character strings. How many bits are flipped, or how many characters are altered?

  ABCDEFGABCDEFG
  | |*| | |**| | |*
  ABXDEFQWBCDEAG

Can you suggest any improvements?

DNASEQUENCECOMPARISON
|**********|*********
DEOXYRIBONUCLEICACIDSEQUENCECOMPARISON

What’s the best method generally depends on what you expect biology will throw at you.
Alignment

• Key aspect of seq comparison is seq alignment

• A seq alignment maximizes the number of positions that are in agreement in the sequences
Sequence Alignment: Poor Example

- 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**

<table>
<thead>
<tr>
<th>Amicyanin</th>
<th>MPHNVHFVAVGLGEAAKGMKKEQAYSLTFTEAGTYDYHCTPHPFMRGKVVE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ascorbate</td>
<td>ILQRGTPWADGTASISQCAINPGETFFYNFTVDNPGETFFYHGHLMQRSAGLYGSL</td>
</tr>
<tr>
<td>Oxidase</td>
<td></td>
</tr>
</tbody>
</table>

No obvious match between Amicyanin and Ascorbate Oxidase
Sequence Alignment: Good Example

- 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]
gil140274931|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%)

Query: 1 MKPGRALASIAlAIIFLPMAVpHAATIETMEnLVIISPTEVSAKVGDTIRWVNKDVFAT 60
       MK G L ++ MA PA AATIE+T++ LV SP V AKVGDTI WVN DV AHT
Sbjct: 1 MKAGALIRLSWLAALALMAAAPAAATIEVTIDKLVFSpATVEAKVGDTIEWVNNDVVAHT 60
```

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

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

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**
  - $S(A) = \log(\text{prob}(A)) - [m \log(s) + h \log(s)]$, 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

• Given sequences $U$ and $V$ of lengths $n$ and $m$, then number of possible alignments is given by
  \[- f(n, m) = f(n-1, m) + f(n-1, m-1) + f(n, m-1) \]
  \[- f(n,n) \sim (1 + \sqrt{2})^{2n+1} n^{-1/2} \]

Exercise: Explain the recurrence above

• 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
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

$$S_{i,j} = \max \left\{ S_{i-1,j-1} + s(u'_i, v'_j), S_{i-1,j} - \delta, S_{i,j-1} - \delta \right\}$$

Exercise: What is the effect of a large $\delta$? This is the basic idea of the Needleman-Wunsch algorithm
Needleman-Wunsch Algorithm (I)
Source: Ken Sung

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

• **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

\[
\begin{array}{c}
nXXX...xx \\
\mid \\
nXXX...yy
\end{array} \quad \begin{array}{c}
nXXX...xx \\
\mid \\
yYYY...y_
\end{array} \quad \begin{array}{c}
nXXX...x_ \\
\mid \\
yYYY...yy
\end{array}
\]

\[
\begin{array}{c}
\text{Match/mismatch} \\
\text{Delete} \\
\text{Insert}
\end{array}
\]

Source: Ken Sung
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>-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>

\[
S_{i+1} = \max \left\{ \begin{array}{c} S_{i+1} = 0 + 2 \\ S_{i+1} = 1 + \max \left\{ -1, -1 \right\} \\ 1 + \max \left\{ -1, -1 \right\} \end{array} \right\}
\]
### 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>
<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 (IV)
Source: Ken Sung

Exercise: Can you tell from these entries what are the values of $s(A,G)$, $s(A,C)$, $s(A,A)$, etc.?
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**

<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></td>
<td>-1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>-1</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>

- **d=3**

A\_CAATCC

AGCA\_TGC
Recursive Equation for O(dn)-Time Algo

\[
v(i, j, d) = \max \begin{cases} 
  v(i-1, j-1, d) + s(S[i], S[j]) \\
  v(i-1, j, d-1) - \delta & \text{if } d > 0 \\
  v(i, j-1, d-1) - \delta & \text{if } d > 0 
\end{cases}
\]

Exercise: Write down the base cases, the memoized version, and the non-recursive version.
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 Algorithm w/ General Gap Penalty (I)

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 Algorithm w/ General Gap Penalty (II)

Source: Ken Sung

- Recurrence for $i>0$ and $j>0$,

$$ V(i, j) = \max \begin{cases} V(i - 1, j - 1) + \delta(S[i], T[j]) & \text{Match/mismatch} \\ \max_{0 \leq k \leq j-1} \{ V(i, k) + g(j - k) \} & \text{Insert } T[k+1..j] \\ \max_{0 \leq k \leq i-1} \{ V(k, j) + g(i - k) \} & \text{Delete } S[k+1..i] \end{cases} $$
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
  ⇒ Time complexity = $O(nm \max\{n, m\})$
  ⇒ Space complexity = $O(nm)$
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

Source: Ken Sung

- X is a **suffix** of S[1..n] if X=S[k..n] for some k\geq 1
- X is a **prefix** of S[1..n] if X=S[1..k] for some k\leq 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:**

  \[
  V(i, j) = \max \left\{ \begin{array}{ll}
  0 & \text{Ignore initial segment} \\
  V(i - 1, j - 1) + s(S[i], T[j]) & \text{Match/mismatch} \\
  V(i - 1, j) - \delta & \text{Delete} \\
  V(i, j - 1) - \delta & \text{Insert} \\
  \end{array} \right. 
  \]
- Score for match = 2
- Score for insert, delete, mismatch = -1

**Example (I)**

Source: Ken Sung

<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></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>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>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>
Example (II)
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>0</td>
<td>2</td>
<td>1</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>0</td>
<td>2</td>
<td>1</td>
<td>2</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>
Example (III)
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>0</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>6</td>
<td>5</td>
</tr>
</tbody>
</table>

An optimal local alignment is
C_AT_G
CAATCG

What is the other optimal local alignment?
Analysis
Source: Ken Sung

• Need to fill in all entries in the n×m matrix
• Each entries can be computed in O(1) time
• Finally, finding the entry with the max value
  ⇒ Time complexity = ??
  ⇒ Space complexity = O(nm)

Exercise: What is the time complexity?
Multiple Sequence Alignment
Multiple Sequence Alignment

- Multiple seq alignment maximizes the number of positions in agreement across several sequences

... but it is so much more!

(and much harder)
Multiple Alignment: Naïve Approach

• Suppose we have 3 sequences to align, S1, S2, S3, and they’re all moderately similar to each other with no one sequence serving as a bridge between the other two.

How could we compute the best alignment if we had all the time and space in the world?
Multiple Alignment:
Naïve Approach

• Suppose we already have a dynamic programming table for aligning S1 and S2, and suppose we want to compare that with a third sequence S3.

How could we compute the best alignment with more reasonable efficiency?

What would the score function look like?

• This requires $O(2^r)$ steps
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
MDLWFFVLLLGSGLISVGATNVTTEPPTTVPTSTRIPTKAPTAAPDGTTTRPRVSSLNVSSPMTPASAPASE
PPTTATSTISPATTASLNASTPGSVPSTSAPVAISLPPSATASTPSALLTAPSTEAMTERNVSATVTQEG
TSSASHNGNSDRLDETPIIAVMVHALSSLILVVFIIVLHYLMRFKKYKQAQGSHNSFRLPNGRTDDAEPOS
MPLLARSPSTNRKYPPLPVDKLEEINRRIGDDNKLRFREEFNALPACPIQATCIEASKEENKEKNRYVIN
LPYDHSRVHLPVEGVPDDSHYINTSFINSYEQKNKFIAAQPKEETVNDFWRMIWEQNTATIVMVNLKE
RKECKCAQYPDQAQGCWTVGVRVSVVSDVTWLVDFLYTVRFCIQQQVGDVTNKQQRZTQFHTSWPDGFVPS
FTPIGMLKFLKKVNTCPNYAGAIVVHCASAGVGRGTFTIVIDAMLDMMHAERKVDYGFVSRIRAQRCQM
VQDMQYVFYIQAHELHYLYGDETELEVTSLEIHLQYINKVPGTSSNGLEEFEKKLTSIKQNDKMTG
LPANMNKNRVLQIIYPENRVIIPVKGCEENTDYNASFDYGYRRTPTCPQPRPVQHTIEDFWRMIWEWK
SCSKMLTELEERQGEKCAQYPDSPDSGYDINVELKKEEEECSYTVRDLLVTNRENKSRQIRQFHF
GWPEVGVPSDGKGMINDIAAVQKQQQGNGMHNCASAGRTGTFCALSTVLEVKDAEILDVFQTVK
SLRLQPRHMPHVQTFQYEFCYKVQEYIDAFSDYANFK

• 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: An 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
MSA: 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 $S_{m_1,\ldots,m_r}$:

$$S_{m_1,\ldots,m_r} = \max_{\epsilon_1 \in \{0,1\}, \ldots, \epsilon_r \in \{0,1\}} \left\{ S_{m_1-\epsilon_1,\ldots,m_r-\epsilon_r} + s(\epsilon_1 \cdot u'_1,m_1,\ldots,\epsilon_r \cdot u'_r,m_r) \right\}$$

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
MSA: Heuristic Approach

• Progressive technique (a.k.a. hierarchical or tree)
  – Find pairwise alignments beginning with the most similar pair and ending with most distant.
  – First stage is to build a guide tree
    • Using a clustering method such as neighbor-joining
  – Second stage is to add additional sequences onto the MSA according to the guide tree.

• Progressive alignments aren’t globally optimal.
  – When errors are made at any early step, they propagate and grow
  – Progressive alignments are efficient enough to handle hundreds of sequences.

Popular Tools for Sequence Comparison:
FASTA, BLAST, Pattern Hunter
Scalability of Software

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

Growth of GenBank (1982 - 2005)
Need Heuristics for Pairwise 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 MUMmer
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

```
GAGTACTCAACACCCAACTTAGTGGCAATGGAAAAT...
```

```
GAATACTCAACACGCAACACTAATGGCAGCAGAAAAT...
```

```
111010010100110111
```

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

Who cares which bits you check first? Doesn’t it all add up to the same amount of effort and the same results?
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 hit to detect one homology
  ⇒ Wasteful

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

```
TTGACCTCACC?
| | | | | | | | |
TTGACCTCACC?
11111111111
11111111111
1/4 chances to have 2nd hit next to the 1st hit

CAA?A??A?C??TA?TGG?
| | | ? | ?? | ? | ?? | | ? | | ?
CAA?A??A?C??TA?TGG?
111010010100110111
111010010100110111
```

1/4<sup>6</sup> 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 $(L - M + 1) \times p^W$

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 ~1/4 of these overlap each other. So likely to have only ~750 \( p^{11}\) 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 ~1000 \( p^{11}\) 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!

*Nature, 420:520-522, 2002*
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 $1000/2^{11}$ hits
  – 2 weight-12 length-18 models expect $2 \times \frac{1000}{2^{12}} = \frac{1000}{2^{11}}$ hits

⇒ When comparing regions w/ >50% similarity, using 2 weight-12 spaced seeds together is more sensitive than using 1 weight-11 spaced seed!

Exercise: Prove 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.
PatternHunter II
Li et al, GIW, 164-175, 2003

• 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 the 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
- PatternHunter

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

- Megablast
- PatternHunter

Image credit: Bioinformatics Solutions Inc

Copyright 2010 © Limsoon Wong & Lisa Tucker-Kellogg
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)
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 and Smith-Waterman are based on those given to me by Ken Sung
References

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