CSEP 527
Computational Biology
Spring 2016
Lecture 2
Sequence Alignment
“HW 0” Background Poll

In your own words, what is DNA? Its main role?
What is RNA? What is its main role in the cell?
How many amino acids are there? Are used in proteins?
Did human beings, as we know them, develop from earlier species of animals?
What are stem cells?
What did Viterbi invent?
What is dynamic programming?
What is a likelihood ratio test?
What is the EM algorithm?
How would you find the max of f(x) = ax^3 + bx^2 + cx + d in the interval -10<x<25?

Don’t worry, we’ll talk about all this stuff before the course ends
Sequence Alignment

What
Why
A Dynamic Programming Algorithm
Sequence Alignment

Goal: position characters in two strings to “best” line up identical/similar ones with one another

We can do this via Dynamic Programming
What is an alignment?

Compare two strings to see how “similar” they are
E.g., maximize the # of identical chars that line up

ATGTTAT vs ATCGTAC

<table>
<thead>
<tr>
<th>A</th>
<th>T</th>
<th>–</th>
<th>G</th>
<th>T</th>
<th>T</th>
<th>A</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>T</td>
<td>C</td>
<td>G</td>
<td>T</td>
<td>–</td>
<td>A</td>
<td>C</td>
</tr>
</tbody>
</table>
What is an alignment?

Compare two strings to see how “similar” they are
E.g., maximize the # of identical chars that line up

ATGTTAT vs ATCGTAC

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

matches

mismatches
Sequence Alignment: Why

Biology
Among most widely used comp. tools in biology
DNA sequencing & assembly
New sequence always compared to data bases

**Similar sequences often have similar origin and/or function**
Recognizable similarity after $10^8 - 10^9$ yr

Other
spell check/correct, diff, svn/git/…, plagiarism, …
### Taxonomy Report

<table>
<thead>
<tr>
<th>Taxonomy</th>
<th>Hits</th>
<th>Orgs</th>
</tr>
</thead>
<tbody>
<tr>
<td>root</td>
<td>64</td>
<td>16</td>
</tr>
<tr>
<td>Eukaryota</td>
<td>62</td>
<td>14</td>
</tr>
<tr>
<td>Fungi/Metazoa</td>
<td>57</td>
<td>11</td>
</tr>
<tr>
<td>Bilateria</td>
<td>38</td>
<td>7</td>
</tr>
<tr>
<td>Coelomata</td>
<td>36</td>
<td>6</td>
</tr>
<tr>
<td>Tetrapoda</td>
<td>26</td>
<td>5</td>
</tr>
<tr>
<td>Eutheria</td>
<td>24</td>
<td>4</td>
</tr>
<tr>
<td>Homo sapiens</td>
<td>20</td>
<td>1</td>
</tr>
<tr>
<td>Murinae</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Mus musculus</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Sus scrofa</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Xenopus laevis</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Drosophila melanogaster</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>Caenorhabditis elegans</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Ascomycota</td>
<td>19</td>
<td>4</td>
</tr>
<tr>
<td>Schizosaccharomyces pombe</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>Saccharomycetales</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>Saccharomyces</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>Saccharomyces cerevisiae</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>Saccharomyces kluyveri</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Candida albicans</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Arabidopsis thaliana</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Apicomplexa</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Plasmodium falciparum</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Toxoplasma gondii</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>synthetic construct</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Lymphocystis disease virus</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

### BLAST Demo

**Try it!**

pick any protein, e.g. hemoglobin, insulin, exportin,… BLAST to find distant relatives.

**Alternate demo:**

- find “BLAST” button about ½ way down page, under “Sequences”, just above big grey box with the amino sequence of this protein
- click “go” button
- after a minute or 2 you should see the 1<sup>st</sup> of 10 pages of “hits” – matches to similar proteins in other species
- you might find it interesting to look at the species descriptions and the “identity” column (generally above 50%, even in species as distant from us as fungus -- extremely unlikely by chance on a 1071 letter sequence over a 20 letter alphabet)
- Also click any of the colored “alignment” bars to see the actual alignment of the human XPO1 protein to its relative in the other species – in 3-row groups (query 1<sup>st</sup>, the match 3<sup>rd</sup>, with identical letters highlighted in between)
**Terminology**

- **String**: ordered list of letters
- **Prefix**: consecutive letters from front
- **Suffix**: consecutive letters from back
- **Substring**: consecutive letters from anywhere
- **Subsequence**: any ordered, nonconsecutive letters, i.e. AAA, TAG

The diagram illustrates a sequence of letters (TATAG) with connections to the definitions of string, prefix, suffix, substring, and subsequence.
Formal definition of an alignment

An **alignment** of strings $S$, $T$ is a pair of strings $S'$, $T'$ with dash characters “–” inserted, so that

1. $|S'| = |T'|$, and
2. Removing dashes leaves $S$, $T$

Consecutive dashes are called “a gap.”

(Note that this is a definition for a general alignment, not optimal.)
Scoring an arbitrary alignment

Define a score for pairs of aligned chars, e.g.

\[ \sigma(x, y) = \begin{cases} 
\text{match} & 2 \\
\text{mismatch} & -1 
\end{cases} \]

Apply that per column, then add.

```
  a  c  -  -  g  c  t  g \\
-  c  a  t  g  t  -  -  
```

\[-1 \quad +2 \quad -1 \quad -1 \quad +2 \quad -1 \quad -1 \quad -1\]

Total Score = -2
More Realistic Scores: BLOSUM 62
(the “σ” scores)

|   | A   | R   | N   | D   | C   | Q   | E   | G   | H   | I   | L   | K   | M   | F   | P   | S   | T   | W   | Y   | V   |
|---|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| A |  4  | -1  | -2  | -2  | 0   | -1  | -1  | 0   | -2  | -1  | -1  | -1  | -1  | -2  | -1  | -1  | -3  | -2  | 0   |     |     |     |
| R | -1  |  5  |  0  | -2  | -3  |  1  |  0  | -2  |  0  | -3  | -2  | -1  | -3  | -2  | -2  | -1  | -3  | -2  | -3  |     |     |     |
| N | -2  |  0  |  6  |  1  | -3  |  0  |  0  |  0  |  1  | -3  | -3  | -2  | -3  | -2  |  1  |  0  | -4  | -2  | -3  |     |     |     |
| D | -2  | -2  |  1  |  6  | -3  |  0  |  2  | -1  | -1  | -3  | -4  | -1  | -3  | -3  | -1  |  0  | -1  | -4  | -3  | -3  |     |     |
| C |  0  | -3  | -3  | -3  |  9  | -3  | -4  | -3  | -3  | -1  | -1  | -3  | -1  | -2  | -3  | -1  | -1  | -2  | -2  | -1  |     |     |     |
| Q | -1  |  1  |  0  |  0  | -3  |  5  |  2  | -2  |  0  | -3  | -2  |  1  |  0  | -3  | -1  |  0  | -1  | -2  | -1  | -2  |     |     |     |
| E | -1  |  0  |  0  |  2  | -4  |  2  |  5  | -2  |  0  | -3  | -3  |  1  | -2  | -3  | -1  |  0  | -1  | -3  | -2  | -2  |     |     |     |
| G |  0  | -2  |  0  | -1  | -3  | -2  | -2  |  6  | -2  | -4  | -4  | -2  | -3  | -3  | -2  |  0  | -2  | -2  | -3  | -3  |     |     |     |
| H | -2  |  0  |  1  | -1  | -3  |  0  |  0  | -2  |  8  | -3  | -3  | -1  | -2  | -1  | -2  | -1  | -2  | -2  |  2  | -3  | -3  | -1  |  3  |
| I | -1  | -3  | -3  | -3  | -1  | -3  | -3  | -4  | -3  |  4  |  2  | -3  |  1  |  0  | -3  | -2  | -1  | -3  | -1  |  3  |     |     |     |
| L | -1  | -2  | -3  | -4  | -1  | -2  | -3  | -4  | -3  |  2  | -2  |  2  |  0  | -3  | -2  | -1  | -2  | -1  | -1  |  1  |     |     |     |
| K | -1  |  2  |  0  | -1  | -3  |  1  |  1  | -2  | -1  | -3  | -2  |  5  | -1  | -3  | -1  |  0  | -1  | -3  | -2  | -2  |     |     |     |
| M | -1  | -1  | -2  | -3  | -1  |  0  | -2  | -3  | -2  |  1  |  2  | -1  |  5  |  0  | -2  | -1  | -1  | -1  | -1  | -1  |  1  |     |     |     |
| F | -2  | -3  | -3  | -3  | -2  | -3  | -3  | -3  | -1  |  0  |  0  | -3  |  6  | -4  |  2  | -2  |  1  |  3  | -3  | -1  |     |     |     |
| P | -1  | -2  | -2  | -1  | -3  | -1  | -1  | -2  | -2  | -3  | -3  | -1  | -2  | -4  |  7  | -1  | -1  | -4  | -3  | -2  |     |     |     |
| S |  1  | -1  |  1  |  0  | -1  |  0  |  0  |  0  | -1  | -2  | -2  |  0  | -1  | -2  | -1  |  4  | -1  | -3  | -2  | -2  |     |     |     |
| T |  0  | -1  |  0  | -1  | -1  | -1  | -1  | -2  | -2  | -1  | -1  | -1  | -1  | -2  | -1  |  1  |  5  | -2  | -2  |  0  |     |     |     |
| W | -3  | -3  | -4  | -4  | -2  | -2  | -3  | -2  | -2  | -3  | -2  |  1  | -1  | -1  | -4  |  3  | -2  |  1  |  1  |  1  |  2  | -3  |     |
| Y | -2  | -2  | -2  | -3  | -2  | -1  | -2  | -3  | -2  | -1  | -1  | -2  | -1  |  3  | -3  | -2  |  2  |  7  | -1  |     |     |     |
| V |  0  | -3  | -3  | -3  | -1  | -2  | -2  | -3  | -3  |  3  |  1  | -2  |  1  | -1  | -2  | -2  |  0  | -3  | -1  |  4  |     |     |     |
Optimal Alignment: A Simple Algorithm

for all subseqs A of S, B of T s.t. |A| = |B| do
    align A[i] with B[i], 1 ≤ i ≤ |A|
    align all other chars to spaces
    compute its value
    retain the max
end
output the retained alignment

S = agct   A = ct
T = wxyz   B = xz
-agc-t    a-gc-t
w--xyz    -w-xyz
Analysis

Assume $|S| = |T| = n$

Cost of evaluating one alignment: $\geq n$

How many alignments are there: $\geq \binom{2n}{n}$

- pick $n$ chars of $S,T$ together
- say $k$ of them are in $S$
- match these $k$ to the $k$ unpicked chars of $T$

Total time: $\geq n \binom{2n}{n} > 2^{2n}$, for $n > 3$

E.g., for $n = 20$, time is $> 2^{40}$ operations
Polynomial vs Exponential Growth
Can we use Dynamic Programming?

1. Can we decompose into subproblems?
   
   E.g., can we align smaller substrings (say, prefix/suffix in this case), then combine them somehow?

2. Do we have optimal substructure?
   
   I.e., is optimal solution to a subproblem independent of context? E.g., is appending two optimal alignments also be optimal? Perhaps, but some changes at the interface might be needed?
Optimal Substructure
(In More Detail)

Optimal alignment *ends* in 1 of 3 ways:
- last chars of S & T aligned with each other
- last char of S aligned with dash in T
- last char of T aligned with dash in S
  ( never align dash with dash; $\sigma(-, -) < 0$ )

*In each case, the rest of S & T should be optimally aligned to each other*
Optimal Alignment in $O(n^2)$ via “Dynamic Programming”

Input: $S$, $T$, $|S| = n$, $|T| = m$
Output: value of optimal alignment

Easier to solve a “harder” problem:

$$V(i,j) = \text{value of optimal alignment of } S[1], \ldots, S[i] \text{ with } T[1], \ldots, T[j]$$

for all $0 \leq i \leq n$, $0 \leq j \leq m$. 
Base Cases

$V(i,0)$: first $i$ chars of $S$ all match dashes

$$V(i,0) = \sum_{k=1}^{i} \sigma(S[k],-)$$

$V(0,j)$: first $j$ chars of $T$ all match dashes

$$V(0,j) = \sum_{k=1}^{j} \sigma(-,T[k])$$
General Case

Opt align of $S[1], \ldots, S[i]$ vs $T[1], \ldots, T[j]$:

$$V(i, j) = \max \begin{cases} V(i-1, j-1) + \sigma(S[i], T[j]) \\ V(i-1, j) + \sigma(S[i], -) \\ V(i, j-1) + \sigma(-, T[j]) \end{cases}$$

for all $1 \leq i \leq n, 1 \leq j \leq m$. 

Opt align of $S_1 \ldots S_{i-1}$ & $T_1 \ldots T_{j-1}$
Calculating One Entry

\[ V(i,j) = \max \begin{cases} 
V(i-1,j-1) + \sigma(S[i], T[j]) \\
V(i-1,j) + \sigma(S[i], -) \\
V(i,j-1) + \sigma(-, T[j]) 
\end{cases} \]
### Example

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
<td>-3</td>
<td>-4</td>
<td>-5</td>
</tr>
<tr>
<td>1</td>
<td>a</td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>c</td>
<td>-2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>g</td>
<td>-3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>c</td>
<td>-4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>t</td>
<td>-5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>g</td>
<td>-6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Mismatch = -1  
Match = 2  
Score(c, -) = -1
Example

<table>
<thead>
<tr>
<th></th>
<th>j</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td></td>
<td>c</td>
<td>a</td>
<td>t</td>
<td>g</td>
<td>t</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
<td>-3</td>
<td>-4</td>
<td>-5</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>a</td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>c</td>
<td>-2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>g</td>
<td>-3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>c</td>
<td>-4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>t</td>
<td>-5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>g</td>
<td>-6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Mismatch = -1
Match = 2

Score(-,a) = -1
Example

<table>
<thead>
<tr>
<th>i</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
<td>-3</td>
<td>-4</td>
<td>-5</td>
</tr>
<tr>
<td>1</td>
<td>a</td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>c</td>
<td>-2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>g</td>
<td>-3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>c</td>
<td>-4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>t</td>
<td>-5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>g</td>
<td>-6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Mismatch = -1
Match = 2

Score(-,c) = -1
### Example

<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>←T</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>0</td>
<td>-1</td>
<td>-2</td>
<td>-3</td>
<td>-4</td>
<td>-5</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>a</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>c</td>
<td>-2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>g</td>
<td>-3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>c</td>
<td>-4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>t</td>
<td>-5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>g</td>
<td>-6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Mismatch = -1
Match = 2

\( \sigma(a,a) = +2 \)
\( \sigma(-,a) = -1 \)
\( \sigma(a,-) = -1 \)
### Example

<table>
<thead>
<tr>
<th>i</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>j</td>
<td></td>
<td>c</td>
<td>a</td>
<td>t</td>
<td>g</td>
<td>t</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
<td>-3</td>
<td>-4</td>
<td>-5</td>
</tr>
<tr>
<td>1 a</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 c</td>
<td>-2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 g</td>
<td>-3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 c</td>
<td>-4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5 t</td>
<td>-5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6 g</td>
<td>-6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Mismatch = -1  
Match = 2  
Time = $O(mn)$
Example

\[
\begin{array}{cccccc}
  & j & 0 & 1 & 2 & 3 & 4 & 5 \\
  i \\
 0 & & 0 & -1 & -2 & -3 & -4 & -5 \\
1 & a & -1 & -1 & 1 & 0 & -1 & -2 \\
2 & c & -2 & 1 & 0 & 0 & -1 & -2 \\
3 & g & -3 & 0 & 0 & -1 & 2 & 1 \\
4 & c & -4 & -1 & -1 & -1 & 1 & 1 \\
5 & t & -5 & -2 & -2 & 1 & 0 & 3 \\
6 & g & -6 & -3 & -3 & 0 & 3 & 2 \\
\end{array}
\]

Mismatch = -1
Match = 2
Finding Alignments: Trace Back

Arrows = (ties for) max in V(i,j); 3 LR-to-UL paths = 3 optimal alignments

<table>
<thead>
<tr>
<th></th>
<th>j</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td></td>
<td>c</td>
<td>a</td>
<td>t</td>
<td>g</td>
<td>t</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>0</td>
<td>-1</td>
<td>-2</td>
<td>-3</td>
<td>-4</td>
<td>-5</td>
</tr>
<tr>
<td>1</td>
<td>a</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
</tr>
<tr>
<td>2</td>
<td>c</td>
<td>-2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
</tr>
<tr>
<td>3</td>
<td>g</td>
<td>-3</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>c</td>
<td>-4</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>t</td>
<td>-5</td>
<td>-2</td>
<td>-2</td>
<td>1</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>g</td>
<td>-6</td>
<td>-3</td>
<td>-3</td>
<td>0</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

↑ S

Ex: what are the 3 alignments? C.f. slide 11.
Complexity Notes

Time = $O(mn)$, (value and alignment)

Space = $O(mn)$

Easy to get value in Time = $O(mn)$ and Space = $O(\min(m,n))$

Possible to get value and alignment in Time = $O(mn)$ and Space = $O(\min(m,n))$, but tricky (DEKM 2.6)
Weekly Bio Interlude

DNA Replication
DNA Replication: Basics
Issues & Complications, I

1st ~10 nt’s added are called the *primer*

In simple model, DNA pol has 2 jobs: prime & extend

Priming is error-prone

So, specialized *primase* does the priming; pol specialized for fast, accurate extension

Still doesn’t solve the accuracy problem (hint: primase makes an RNA primer)
“Replication Fork”: DNA double helix is progressively unwound by a DNA helicase, and both resulting single strands are duplicated. DNA polymerase synthesizes new strand 5’ -> 3'(reading its template strand 3’ -> 5’). That means on one (the “leading”) strand, DNA pol is chasing/pushing the replication fork. But on the other “lagging” strand, DNA pol is running away from it.
Issue 3: Fragments

Lagging strand gets a series of “Okazaki fragments” of DNA (~200nt in eukaryotes) following each primer.

The RNA primers are later removed by a nuclease and DNA pol fills gaps (more accurate than primase; primed by DNA from adjacent Okazaki frag).

Fragments joined by ligase.
Issue 4: Coord of Leading/Lagging

Alberts et al., Mol. Biol. of the Cell, 3rd ed, p258
Very Nice DNA Repl. Animation

https://www.youtube.com/watch?v=yqESR7E4b_8
Issue 5: Twirls & Tangles

Unwinding helix (~10 nucleotides per turn) would cause stress. *Topoisomerase I* cuts DNA backbone on *one* strand, allowing it to spin about the remaining bond, relieving stress.

*Topoisomerase II* can cut & rejoin *both* strands, after allowing another double strand to pass through the gap, de-tangling it.
Issue 6: Proofreading

Error rate of pol itself is $\sim 10^{-4}$, but overall rate is $\approx 10^{-8}$, due to proofreading & repair, e.g.

pol itself can back up & cut off a mismatched base if one happens to be inserted

priming the new strand is hard to do accurately, hence RNA primers, later removed & replaced

other enzymes scan helix for “bulges” caused by base mismatch, figure out which strand is original, cut away new (faulty) copy; DNA pol fills gap

which strand is original? Bacteria: “methylate” some A’s, eventually. Euks: strand nicking
Replication Summary

Speed: 50 (eukaryotes) to 500 (prokaryotes) bp/sec
Accuracy: 1 error per $10^8$–$10^9$ bp
Complex & highly optimized
Highly similar across all living cells

More info:
   Alberts et al., *Mol. Biol. of the Cell*
Sequence Alignment

Part II
Local alignments & gaps
Variations

Local Alignment
  Preceding gives \textit{global} alignment, i.e. full length of both strings;
  Might well miss strong similarity of part of strings amidst dissimilar flanks

Gap Penalties
  10 adjacent spaces cost 10 x one space?

Many others
Similarly fast DP algs often possible
Local Alignment: Motivations

“Interesting” (evolutionarily conserved, functionally related) segments may be a small part of the whole

“Active site” of a protein
Scattered genes or exons amidst “junk”, e.g. retroviral insertions, large deletions
Don’t have whole sequence

Global alignment might miss them if flanking junk outweighs similar regions
Local Alignment

Optimal *local alignment* of strings S & T: Find substrings A of S and B of T having max value global alignment

\[
S = abcxdex \quad A = c \times d e
\]
\[
T = xxxcde \quad B = c - d e \quad \text{value} = 5
\]
Local Alignment: “Obvious” Algorithm

for all substrings A of S and B of T:
  Align A & B via dynamic programming
  Retain pair with max value
end;

Output the retained pair

Time: \(O(n^2)\) choices for A, \(O(m^2)\) for B,
  \(O(nm)\) for DP, so \(O(n^3m^3)\) total.

[Best possible? Lots of redundant work…]
Local Alignment in $O(nm)$ via Dynamic Programming

Input: $S$, $T$, $|S| = n$, $|T| = m$
Output: value of optimal local alignment

Better to solve a “harder” problem for all $0 \leq i \leq n$, $0 \leq j \leq m$:

$$V(i,j) = \max \text{ value of opt (global) alignment of a suffix of } S[1], \ldots, S[i] \text{ with a suffix of } T[1], \ldots, T[j]$$

Report best $i,j$
Base Cases

Assume $\sigma(x,-) \leq 0$, $\sigma(-,x) \leq 0$

$V(i,0)$: some suffix of first $i$ chars of $S$; all match spaces in $T$; best suffix is empty

$V(i,0) = 0$

$V(0,j)$: similar

$V(0,j) = 0$
Opt **suffix** align $S[1], \ldots, S[i]$ vs $T[1], \ldots, T[j]$:
\[
\begin{bmatrix}
\text{-----} S[i] \\
\text{-----} T[j] \\
\end{bmatrix}, \quad \begin{bmatrix}
\text{-----} S[i] \\
\text{-----} \quad - \\
\end{bmatrix}, \quad \begin{bmatrix}
\text{-----} - \\
\text{-----} T[j] \\
\end{bmatrix}, \text{ or } \begin{bmatrix}
\end{bmatrix}
\]

Opt align of suffix of
$S_1 \ldots S_{i-1}$ &
$T_1 \ldots T_{j-1}$

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

opt suffix alignment has:
2, 1, 1, 0
chars of $S/T$

for all $1 \leq i \leq n$, $1 \leq j \leq m$. 

48
### Scoring Local Alignments

<table>
<thead>
<tr>
<th></th>
<th>j</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<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>1</td>
<td>a</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>b</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>c</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>x</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>d</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>e</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>x</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The alignment is represented by the matrix, where each cell corresponds to a score for an alignment at that position. The arrow indicating the alignment path from i to j.
Finding Local Alignments

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>c</td>
<td>d</td>
<td>e</td>
</tr>
<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>
</tr>
<tr>
<td>1</td>
<td>a</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>b</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>c</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>x</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>d</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>e</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>x</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>

Again, arrows follow max term (not max neighbor)
Notes

Time and Space = $O(mn)$
Space $O(\text{min}(m,n))$ possible with time $O(mn)$, but finding alignment is trickier

Local alignment: “Smith-Waterman”
Global alignment: “Needleman-Wunsch”
Significance of Alignments

Is “42” a good score?

*Compared to what?*

Usual approach: compared to a specific “null model”, such as “random sequences”

More on this later; a taste now, for use in next HW
Overall Alignment Significance, II
Empirical (via randomization)

You just searched with x, found “good” score for x:y
Generate N random “y-like” sequences (say N = 10³ - 10⁶)
Align x to each & score
If k of them have score than better or equal to that of x to y, then the (empirical) probability of a chance alignment as good as observed x:y alignment is (k+1)/(N+1)
e.g., if 0 of 99 are better, you can say “estimated p ≤ .01”

How to generate “random y-like” seqs? Scores depend on:
Length, so use same length as y
Sequence composition, so uniform 1/20 or 1/4 is a bad idea; even background pᵢ can be dangerous (if y unusual)
Better idea: permute y N times
Generating Random Permutations

for (i = n-1; i > 0; i--){
    j = random(0..i);
    swap X[i] <-> X[j];
}

All $n!$ permutations of the original data equally likely: A specific element will be last with prob $1/n$; given that, another specific element will be next-to-last with prob $1/(n-1)$, …; overall: $1/(n!)$

Alignment With Gap Penalties

**Gap:** maximal run of dashes in S’ or T’

- ag--ttc-t  2 gaps in S’
- a---ttcgt  1 gap in T’

Motivations, e.g.:
- mutation might insert/delete several or even many residues at once
- matching mRNA (no introns) to genomic DNA (exons and introns)
- some parts of proteins less critical
A Protein Structure:
(Dihydrofolate Reductase)

http://www.rcsb.org/pdb/explore/jmol.do?structureId=5CC9&bionumber=1
Alignment of 5 Dihydrofolate reductase proteins

- Mouse: P00375
- Human: P00374
- Chicken: P00378
- Fly: P17719
- Yeast: P07807

<table>
<thead>
<tr>
<th></th>
<th>P00375</th>
<th>P00374</th>
<th>P00378</th>
<th>P17719</th>
<th>P07807</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dihydrofolate reductase proteins</td>
<td>MVRPLNCIVAVSQRNGKGLPWPPLRNEFKYFQRMTTSVESGQNLVIMGRK</td>
<td>MVGSLNCIVAVSQRNGKGLPWPPLRNEFRYFQRMTTSVESGQNLVIMGRK</td>
<td>VRSLNIVAVCQRNGKGLGNLWPPLRNEYKFQRTSTSVHVEGQNAVIMGRK</td>
<td>MLR-FNLIVAVCENFGIRGDPWR-IKSELKYSRTTSTSDPTKQNAVMGRK</td>
<td>MAGKIPIVGIVACLQPEMGIFRGLPWR-LPSEMKYFRVTSLTLDPNKKNALIMGRA</td>
</tr>
<tr>
<td></td>
<td>TWFSIPEKNRPLKDRINIVLSELKPEP-PRGAHFLAKSDDLALRQTEQPELASKVD</td>
<td>TWFSIPEKNRPLKGRINLVLSSELKPEP-PQGAHFLSKLDALKLETQPELANKVD</td>
<td>TWFSIPEKNRPLKDRINIVLSELKEA-PKGAHYLSKDLDALDSPELKSVD</td>
<td>TYFGVPEKRLPDRLNIIVLSTTTQESDLPKG-VLLCPNLETAMKILEEE-QNEVEN</td>
<td>TWESIPPFKRPLPNRMNVIISRSDKDFVHDKRESIVQSNLSANAINLESN-FKEHLER</td>
</tr>
<tr>
<td></td>
<td>VWIVGGSVVQTEAMNPQGHLRFVTRIMQEFESDTFFPEIDLGKYKLLPEYPG-</td>
<td>VWIVGSSVYKEAMNHGPGLKLFVTRIMQDFESDTFFPEIDLEKYKLLPEYPG-</td>
<td>VWIVGTAAYKAAMETPINHRLVFTRILHESDFTFFPEIDYKDFKLLTEYPG-</td>
<td>IWVGGSgyeeamaspchrlyitkimqfCDDTFFPAIP-DSSREVAPVSD-</td>
<td>IVIGGGEVSQIFSITDHWLITKINPLDNAPMDTFDAKLEEVFSEQDPAQLKEF</td>
</tr>
<tr>
<td></td>
<td>VLSEVQ-EEKGIKYKFENVYKKEEE</td>
<td>VLSDVQ-EEKGIKYKFENVYKEED</td>
<td>VPADIVQ-EEgiqykFENVYQKSVLAQ</td>
<td>MPLGVO-EEgiKFEYKILEKH</td>
<td>LPPKVELPARDQCQYSEKGYCFFTKLYNKR</td>
</tr>
</tbody>
</table>

CLUSTAL W (1.82) multiple sequence alignment
[http://pir.georgetown.edu/cgi-bin/multialn.pl](http://pir.georgetown.edu/cgi-bin/multialn.pl)
2/11/2013
Topoisomerase I

http://www.rcsb.org/pdb/explore.do?structureId=1a36
Affine Gap Penalties

Gap penalty = g + e*(gaplen-1), g ≥ e ≥ 0

Note: no longer suffices to know just the score of best subproblem(s) – state matters: do they end with ‘-’ or not.
Global Alignment with Affine Gap Penalties

V(i,j) = value of opt alignment of S[1], ..., S[i] with T[1], ..., T[j]

G(i,j) = ..., s.t. last pair matches S[i] & T[j]

F(i,j) = ..., s.t. last pair matches S[i] & –

E(i,j) = ..., s.t. last pair matches – & T[j]

Time: O(mn) [calculate all, O(1) each]
Affine Gap Algorithm

Gap penalty $= g + e \times (\text{gaplen}-1)$, $g \geq e \geq 0$

$V(i,0) = E(i,0) = V(0,i) = F(0,i) = -g-(i-1)\times e$

$V(i,j) = \max(G(i,j), F(i,j), E(i,j))$

$G(i,j) = V(i-1,j-1) + \sigma(S[i],T[j])$

$F(i,j) = \max( F(i-1,j)-e, V(i-1,j)-g )$

$E(i,j) = \max( E(i,j-1)-e, V(i,j-1)-g )$

Q. Why is the “V” case a “new gap” when V includes E & F?
Other Gap Penalties

Score = f(gap length)
Kinds, & best known alignment time

- affine: \( O(n^2) \) [really, \( O(mn) \)]
- convex: \( O(n^2 \log n) \)
- general: \( O(n^3) \)
Summary: Alignment

Functionally similar proteins/DNA often have recognizably similar sequences even after eons of divergent evolution.

Ability to find/compare/experiment with “same” sequence in other organisms is a huge win.

Surprisingly simple scoring works well in practice: score positions separately & add, usually w/ fancier affine gap model.

Simple dynamic programming algorithms can find optimal alignments under these assumptions in poly time (product of sequence lengths).

This, and heuristic approximations to it like BLAST, are workhorse tools in molecular biology, and elsewhere.
Summary: Dynamic Programming

Keys to D.P. are to

a) identify the subproblems (usually repeated/overlapping)
b) solve them in a careful order so all small ones solved before they are needed by the bigger ones, and
c) build table with solutions to the smaller ones so bigger ones just need to do table lookups (*no* recursion, despite recursive formulation implicit in (a))
d) Implicitly, optimal solution to whole problem devolves to optimal solutions to subproblems

A *really* important algorithm design paradigm