Master Course

Algorithms in Sequence Analysis

Lecture 2: Pairwise alignment II

heringa@few.vu.nl, http://ibi.vu.nl, Tel. 87649, Rm P1.28
Global dynamic programming

\[ H(i,j) = \max \begin{cases} H(i-1,j-1) + S(i,j) & \text{diagonal} \\ H(i-1,j) - g & \text{vertical} \\ H(i,j-1) - g & \text{horizontal} \end{cases} \]

Two-step approach (+ initialisation of first row and column with gap penalties):

• Forward step: calculating matrix cells
  ○ Alignment score in lower rightmost cell
• Traceback step: reconstructing the alignment path and thus the alignment itself
Global dynamic programming

Forward step

Boundary conditions for end gap penalties

Initialising row

- \(L---\) ACVL or \(-L-\) ACVL or \(--L--\) ACVL

Initialising column

---\(L\) ACVL
Global dynamic programming
Trace back

\[ j-1 \rightarrow j \]

match/mismatch
gap
gap
Global dynamic programming

\[ H(i,j) = \max \left\{ \begin{array}{ll}
H(i-1,j-1) + S(i,j) & \text{diagonal} \\
H(i-1,j) - g & \text{vertical} \\
H(i,j-1) - g & \text{horizontal}
\end{array} \right. \]

Problem with simple DP approach:

- Can only do linear gap penalties
- Not suitable for affine or concave penalties, but algorithm can be extended to deal with affine penalties
Global dynamic programming using affine penalties

Looking back from cell \((i, j)\) we can adapt the algorithm for affine gap penalties by looking back to four more cells (magenta).

If you came from here, gap was opened so apply gap-open penalty.

If you came from here, gap was already open, so apply gap-extension penalty.
Affine gaps

- Penalties:
  - $g_o$ - gap opening (e.g. -8)
  - $g_e$ - gap extension (e.g. -1)

\[ M[i, j] = \text{score}(X[i], Y[j]) + \max \left\{ M[i-1, j-1], I_x[i-1, j-1], I_y[i-1, j-1] \right\} \]

- @ home: think of boundary values $M[*0], I[*0]$ etc.

\[ X_1 \ldots X_i \]
\[ Y_1 \ldots Y_j \]

\[ M[i, j] = \text{score}(X[i], Y[j]) + \max \]
\[ \left\{ \begin{array}{l}
M[i-1, j-1] \\
I_x[i-1, j-1] \\
I_y[i-1, j-1]
\end{array} \right. \]

\[ I_x[i, j] = \max \left\{ M[i, j-1] + g_o, X_{1 \ldots i}, Y_{1 \ldots j-1}, Y_j \right\} \]

\[ I_y[i, j] = \max \left\{ M[i-1, j] + g_o, I_x[i-1, j-1] + g_e \right\} \]
Time and memory complexity of DP

- The \textit{time complexity} is $O(n^2)$, \textit{i.e. order of} $n^2$: for aligning two sequences of $n$ residues, you need to perform $n^2$ algorithmic steps (square search matrix has $n^2$ cells that need to be filled)

- The \textit{memory (space) complexity} is also $O(n^2)$: for aligning two sequences of $n$ residues, you need a square search matrix of $n$ by $n$ containing $n^2$ cells
Alternative DP recipe for using linear, affine and concave gap penalties

- $M[i,j]$ is optimal alignment (highest scoring alignment until $[i,j]$)
- Check
  - Cell[$i-1$, $j-1$]: apply score for cell[$i-1$, $j-1$]
  - preceding row until $j-2$: apply appropriate gap penalties
  - preceding column until $i-2$: apply appropriate gap penalties
Global dynamic programming
all types of gap penalty

\[ S_{i,j} = s_{i,j} + \max \begin{cases} \max_{0<x<i-1} \{ S_{0<x<i-1,j-1} - \text{Gap}(i-x-1) \} \\ S_{i-1,j-1} \\ \max_{0<y<j-1} \{ S_{i-1,0<y<j-1} - \text{Gap}(j-y-1) \} \end{cases} \]

The complexity of this DP algorithm is increased from \( O(n^2) \) to \( O(n^3) \)

The gap length is known exactly and so any gap penalty regime can be used
DP is a two-step process

- **Initialise**
- **Forward step**: calculate scores
- **Trace back**: start at lower-right corner cell and reconstruct the path leading to the highest score (global alignment)
  - Use pointers from end to beginning of path
  - Or use cell values

These two steps lead to the highest scoring global alignment (the optimal alignment)

*This is guaranteed when you use DP!*
There are three kinds of alignments

- Global alignment (preceeding slides)
- **Semi-global alignment**
- Local alignment
Variation on global alignment

• *Global* alignment: previous algorithm is called *global* alignment because it uses all letters from both sequences.
  
  CAGC\text{ACTTGGATTCTCGG}  
  CAGC\text{-----G–T-----GG}  

• *Semi-global* alignment: uses all letters but does not penalize for end gaps
  
  CAGC\text{A–CTTGGATTCTCGG}  
  \text{CCCAGC\text{GTGG}-----------}
Semi-global alignment

- Global alignment: all gaps are penalised
- Semi-global alignment: N- and C-terminal (5’ and 3’) gaps (end-gaps) are not penalised

MSTGAVLIY--TS-----
---GGILLFHTSNGTNS

End-gaps
Semi-global alignment

Applications of *semi-global*:
- Finding a gene in genome
- Placing marker onto a chromosome
- One sequence much longer than the other

*Risk:* if gap penalties high -- really bad alignments for divergent sequences

Protein sequences have N- and C-terminal amino acids that are often small and hydrophilic
Semi-global alignment

- Ignore 5' or N-terminal end gaps
  - First row/column set to 0
- Ignore C-terminal or 3' end gaps
  - Read the result from last row/column (select the highest scoring cell)
Global alignments - review

- Take two sequences: $X[j]$ and $Y[j]$
- The best alignment for $X[1...i]$ and $Y[1...j]$ is called $M[i, j]$
- Initiation: $M[0,0]=0$
  - Global alignment: initial gap penalties (see earlier slides)
- Apply the equation
- Find the alignment using trace-back
There are three kinds of alignments

- Global alignment (preceding slides)
- Semi-global alignment
- **Local alignment**
There are three kinds of pairwise alignments

- Global alignment – align all residues in both sequences; all gaps are penalised
- Semi-global alignment – align all residues in both sequences; end gaps are not penalised (zero end gap penalties)
- Local alignment – align one part of each sequence; end gaps are not applicable
Local alignment

• What’s local?
  – Allow only **parts of the sequence** to match
  – Results in **High Scoring Segments**
  – **Locally maximal**: cannot make it better by trimming/extending the alignment
Local alignment

- Why local?

  - Parts of sequence diverge faster
    evolutionary pressure does not put constraints on the whole sequence

  - Proteins have modular construction
    sharing domains between sequences
Domains - example
Immunoglobulin domain

Representative ig domain proteins

**1A01_GORGO** [Gorilla gorilla gorilla (lowland gorilla)] class I histocompatibility antigen, gogo-a0101 alpha chain precursor

![MHC_I ig](image1)

- **ALC1_GORGO** [Gorilla gorilla gorilla (lowland gorilla)] ig alpha-1 chain c region

- **AMAL_DROME** [Drosophila melanogaster (fruit fly)] amalgam protein precursor

- **B2MG_BOVIN** [Bos taurus (bovine)] beta-2-microglobulin precursor (lactollin)
Global → local alignment

a) global alignment

- s e q u e

b) retrieve the result

CAGCACTTGGATTCTCG-
CA-C-----GATTCGT-G

c) sum score along the result

- Take the old, good equation
- Look at the result of the *global* alignment
Local alignment – breaking the alignment

• A recipe
  – Just don’t let the score go below 0
  – Start new alignment when it happens
  – Where is the result in the matrix?

Before:

```
-  s  e  q  u  e
-  0
s
e
q
```

After:

```
-  s  e  q  u  e
-  0
s
e
q
```

The image shows graphs indicating the alignment positions for both the before and after states.
Local dynamic programming
(Smith & Waterman, 1981)

Amino Acid Exchange Matrix

Gap penalties (e.g., open, extension)
Local dynamic programming
(Smith and Waterman, 1981)

**basic algorithm**

\[
H(i,j) = \begin{cases} 
H(i-1,j-1) + S(i,j) & \text{diagonal} \\
H(i-1, j) - g & \text{vertical} \\
H(i, j-1) - g & \text{horizontal} \\
0 & 
\end{cases}
\]

Local alignment – the equation

- Init the matrix with 0’s
- Fill in the search matrix (forward step)
- Read the maximal value from anywhere in the matrix
- Find the result by performing trace-back

\[
M[i, j] = \max \begin{cases} 
  M[i-1, j-1] + \text{score}(X[i], Y[j]) \\
  M[i, j-1] - g \\
  M[i-1, j] - g \\
  0
\end{cases}
\]

Great contribution to science!
Example: *local* alignment of two sequences

- Align two DNA sequences:
  - GAGTGA
  - GAGGCGA (note the length difference)

- Parameters of the algorithm:
  - **Match**: $\text{score}(A,A) = 1$
  - **Mismatch**: $\text{score}(A,T) = -1$
  - **Gap**: $g = -2$
  
  $$
  M[i, j] = \begin{cases} 
  M[i-1, j-1] \pm 1 & \\
  M[i, j-1] - 2 & \\
  M[i-1, j] - 2 & \\
  0 & 
  \end{cases}
  $$
  
  $M[i, j]$ = max
The algorithm. Step 1: init

- Create the matrix
- Initiation
  - No beginning row/column
  - Just apply the equation...

\[
M[i, j] = \max \begin{cases} 
M[i-1, j-1] \pm 1 \\
M[i, j-1] - 2 \\
M[i-1, j] - 2 \\
0 
\end{cases}
\]

\[
\begin{array}{ccccccc}
\text{j} \rightarrow & 1 & 2 & 3 & 4 & 5 & 6 \\
\text{i} \downarrow & \text{G} & \text{A} & \text{G} & \text{T} & \text{G} & \text{A} \\
1 & \text{G} \\
2 & \text{A} \\
3 & \text{G} \\
4 & \text{G} \\
5 & \text{C} \\
6 & \text{G} \\
7 & \text{A} \\
\end{array}
\]
The algorithm. Step 2: fill in

- Perform the forward step…

\[
M[i, j] = \max \left\{ \begin{array}{l}
M[i-1, j-1] + 1 \\
M[i, j-1] - 2 \\
M[i-1, j] - 2 \\
0
\end{array} \right. 
\]

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

Bold ‘0’s represent cells would have scored negative otherwise (SW’s 0)
The algorithm. Step 2: fill in

- Perform the forward step...

\[
M[i, j] = \max \left\{ M[i-1, j-1] \pm 1, M[i, j-1] - 2, M[i-1, j] - 2, 0 \right\}
\]

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

Bold '0's represent cells would have scored negative otherwise (SW's 0)
The algorithm. Step 2: fill in

- We’re done

- Find the highest cell anywhere in the matrix

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

Bold ‘0’s represent cells would have scored negative otherwise (SW’s 0)
The algorithm. Step 3: trace back

- Reconstruct path leading to highest scoring cell
- Trace back until zero or start of sequence: alignment path can begin and terminate anywhere in matrix
- Alignment: GAG GAG

\[
M[i, j] = \max \begin{cases} 
M[i-1, j-1] \pm 1 \\
M[i, j-1] - 2 \\
M[i-1, j] - 2 \\
0
\end{cases}
\]

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

Bold ‘0’s represent cells would have scored negative otherwise (SW’s 0)
Local dynamic programming
Match/mismatch = 1/-1, Gap = 2

Fill the matrix (forward pass), then do trace back from highest cell anywhere in the matrix till you reach 0 or the beginning of a sequence
Local dynamic programming

Match/mismatch = 1/-1, Gap = 2

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

Fill the matrix (forward pass), then do trace back from highest cell anywhere in the matrix till you reach 0 or the beginning of a sequence.
General DP algorithm for using all types of gap penalties

- M[i,j] is optimal alignment (highest scoring alignment until [i, j])
- At each cell [i, j] in search matrix, check Max coming from:
  - any cell in preceding row until j-2: add score for cell[i, j] minus appropriate gap penalties;
  - any cell in preceding column until i-2: add score for cell[i, j] minus appropriate gap penalties;
  - cell[i-1, j-1]: add score for cell[i, j]
    or 0
- Select highest scoring cell anywhere in matrix and do trace-back until zero-valued cell or start of sequence(s)
Local dynamic programming
(Smith & Waterman, 1981; Gotoh, 1984)

This is the general DP algorithm, which is suitable for linear, affine and concave penalties, although for the example here affine penalties are used.

Also here the highest cell for starting the trace-back can be anywhere in the matrix.

\[ S_{i,j} = \max \begin{cases} S_{i,j} + \max \{ S_{0<x<i-1,j-1} - P_i - (i-x-1)Px \} \\ S_{i,j} + S_{i-1,j-1} \\ S_{i,j} + \max \{ S_{i-1,0<y<j-1} - P_i - (j-y-1)Px \} \\ 0 \end{cases} \]
Let's do yet another example: *local* alignment

Gotoh’s DP algorithm with affine gap penalties (PAM250, Pi=10, Pe=2)

<table>
<thead>
<tr>
<th></th>
<th>D</th>
<th>W</th>
<th>V</th>
<th>T</th>
<th>A</th>
<th>L</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>0</td>
<td>-5</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>4</td>
<td>-7</td>
<td>-2</td>
<td>0</td>
<td>0</td>
<td>-4</td>
<td>0</td>
</tr>
<tr>
<td>W</td>
<td>-7</td>
<td>17</td>
<td>-6</td>
<td>-5</td>
<td>-6</td>
<td>-2</td>
<td>-3</td>
</tr>
<tr>
<td>V</td>
<td>-2</td>
<td>-6</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>-2</td>
</tr>
<tr>
<td>L</td>
<td>-4</td>
<td>-2</td>
<td>2</td>
<td>1</td>
<td>-2</td>
<td>6</td>
<td>-3</td>
</tr>
<tr>
<td>K</td>
<td>0</td>
<td>-3</td>
<td>-2</td>
<td>0</td>
<td>-1</td>
<td>-3</td>
<td>5</td>
</tr>
</tbody>
</table>

![Search Matrix](https://via.placeholder.com/150)

Extra start/end columns/rows not necessary (no end-gaps). Each negative scoring cell is set to zero. Highest scoring cell may be found *anywhere in search matrix* after calculating it. Trace highest scoring cell back to first cell with zero value (or the beginning of one or both sequences)
Finding second best alignment

- We can find the best local alignment in the sequence
- But where is the second best one?

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>A</th>
<th>G</th>
<th>A</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>18</td>
<td>16</td>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>16</td>
<td>19</td>
<td>17</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>14</td>
<td>17</td>
<td>20</td>
<td>18</td>
<td>16</td>
</tr>
<tr>
<td>A</td>
<td>15</td>
<td>18</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>13</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Scoring:
- 1 for match
- -2 for a gap

Best alignment

A clump
Clump of an alignment

- Alignments sharing at least one aligned pair
Example: repeat proteins

Local alignment traces showing similarity between repeats

Note: repeats are similar motifs but need not be identical
Clumps

The figure shows two proteins with so-called tandem repeats (similar motifs adjacent in the sequence).

‘Shadows’ of various alignments are visible – these all derive from parts of a same locally optimal alignment.
Finding second best alignment

• Don’t let any matched pair contribute to the next alignment

```
   A   G   A
--- --- ---
A    0   0   1
G    0   0   0
A    1   0   0
G    2   0
A    0   3   1
--- --- ---
```

“Clear” the best alignment

Recalculate the clump
The figure shows two proteins with so-called tandem repeats (similar motifs adjacent in the sequence)
The figure shows two proteins with so-called tandem repeats (similar motifs adjacent in the sequence).

The highest scoring local alignment is indicated, with its clump recalculated.
Multiple extraction of local alignments

Waterman-Eggert algorithm

1. Repeat
   a. Retrieve the highest scoring alignment
   b. Set its trace to 0
      • Recalculate clump (shadow)

Alternative to Waterman-Eggert: remember alignment trace cells, and let any next trace not intersect with earlier alignments (e.g. Heringa, 1993)
General local DP algorithm for local alignment using affine gap penalties

- $M[i,j]$ is optimal alignment (highest scoring alignment until $[i, j]$)
- At each cell $[i, j]$ in search matrix, check Max coming from:
  - any cell in preceding row until $j-2$: add score for cell$[i, j]$ minus appropriate gap penalties;
  - any cell in preceding column until $i-2$: add score for cell$[i, j]$ minus appropriate gap penalties;
  - or cell$[i-1, j-1]$: add score for cell$[i, j]$
- Select highest scoring cell anywhere in matrix and do trace-back until zero-valued cell or start of sequence(s)

Penalty = $P_{\text{open}} + \text{gap}_\text{length} \times P_{\text{extension}}$

$$S_{i,j} = \begin{cases} S_{i,j} + \text{Max} \{S_{0<x<i-1,j-1} - P_i - (i-x-1)P_x\} \\ S_{i,j} + S_{i-1,j-1} \\ S_{i,j} + \text{Max} \{S_{i-1,0<y<j-1} - P_i - (j-y-1)P_x\} \\ 0 \end{cases}$$
Example: general algorithm for *local* alignment

DP algorithm with affine gap penalties (PAM250, Po=10, Pe=2)

<table>
<thead>
<tr>
<th></th>
<th>D</th>
<th>W</th>
<th>V</th>
<th>T</th>
<th>A</th>
<th>L</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>0</td>
<td>-5</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>4</td>
<td>-7</td>
<td>-2</td>
<td>0</td>
<td>0</td>
<td>-4</td>
<td>0</td>
</tr>
<tr>
<td>W</td>
<td>-7</td>
<td>17</td>
<td>-6</td>
<td>-5</td>
<td>-6</td>
<td>-2</td>
<td>-3</td>
</tr>
<tr>
<td>V</td>
<td>-2</td>
<td>-6</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>-2</td>
</tr>
<tr>
<td>L</td>
<td>-4</td>
<td>-2</td>
<td>2</td>
<td>1</td>
<td>-2</td>
<td>6</td>
<td>-3</td>
</tr>
<tr>
<td>K</td>
<td>0</td>
<td>-3</td>
<td>-2</td>
<td>0</td>
<td>-1</td>
<td>-3</td>
<td>5</td>
</tr>
</tbody>
</table>

Extra start/end columns/rows not necessary (no end-gaps). Each negative scoring cell is set to zero. Highest scoring cell may be found *anywhere in search matrix* after calculating it. Trace highest scoring cell back to first cell with zero value (or the beginning of one or both sequences)
Pitfalls of alignments

- Alignment is often not a reconstruction of evolution
  (common ancestor is usually extinct by the time of alignment)

- Repeats: matches to the same fragment
Summary

1. **Global**
   e.g. Needleman-Wunsch algorithm

2. **Semi-global**

3. **Local**
   e.g. Smith-Waterman algorithm

4. **Multiple local alignments**
   aka Waterman-Eggert algorithm

- What’s the number of steps in these algorithms?
- How much memory is used?

These questions deal with the algorithmic **complexity** (time, memory) of alignment
Global alignment ("simple" recursive DP)

- Fill pre-row and pre-column with gap penalties to account for initial end gaps
- Perform trace-back from lower-right matrix cell, thereby accounting for terminal end gaps

<table>
<thead>
<tr>
<th></th>
<th>-</th>
<th>G</th>
<th>A</th>
<th>G</th>
<th>T</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>0</td>
<td>-2</td>
<td>-4</td>
<td>-6</td>
<td>-8</td>
<td>-10</td>
</tr>
<tr>
<td>A</td>
<td>-2</td>
<td>-1</td>
<td>-1</td>
<td>-3</td>
<td>-4</td>
<td>-5</td>
</tr>
<tr>
<td>G</td>
<td>-4</td>
<td>-1</td>
<td>-2</td>
<td>0</td>
<td>-2</td>
<td>-3</td>
</tr>
<tr>
<td>T</td>
<td>-6</td>
<td>-3</td>
<td>-2</td>
<td>-2</td>
<td>1</td>
<td>-1</td>
</tr>
</tbody>
</table>
Semi-global alignment

• Ignore initial end gaps
  – First row/column set to 0

• Ignore terminal end gaps
  – Read the result from last row/column

<table>
<thead>
<tr>
<th></th>
<th>-</th>
<th>G</th>
<th>A</th>
<th>G</th>
<th>T</th>
<th>G</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>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>2</td>
<td>0</td>
<td>-2</td>
</tr>
<tr>
<td>T</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

Match/mismatch = 1/-1
Constant gap penalty = -2
Local alignment

• Ignore initial end gaps
  – First row/column set to 0

• Ignore terminal end gaps
  – Read the result from highest scoring cell anywhere in matrix
  – Perform trace-back from this cell until zero cell or beginning of sequence

Match/mismatch = 1/-1
Constant gap penalty = -2

<table>
<thead>
<tr>
<th></th>
<th>-</th>
<th>G</th>
<th>A</th>
<th>G</th>
<th>T</th>
<th>G</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>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
What can sequence alignment tell us about structure

HSSP  Sander & Schneider, 1991

Sequence identity implies structural similarity!

≥30% sequence identity

Don't know region

Number of residues aligned

Percentage sequence identity
Note about gap penalties

• Some affine schemes use
  \[ \text{gap\_penalty} = -g_{\text{open}} - g_{\text{extension}} \times (L-1), \]
  while others use
  \[ \text{gap\_penalty} = -g_{\text{open}} - g_{\text{extension}} \times L, \]
  where \( L \) is the length of the gap.

One can easily be converted into the other by adapting the \( g_{\text{open}} \) penalty.
Global and local alignment

**Problem:**
Can semi-global alignment lead to another solution?
Globin fold
α protein
myoglobin
PDB: 1MBN

Alpha-helices are labelled ‘A’ (blue) to ‘H’ (red). The D helix can be missing in some globins:

What happens with the alignment if D-helix containing globin sequences are aligned with ‘D-less’ ones?
Immunoglobulin structures have variable regions where numbers of amino acids can vary substantially. A challenge for exchange matrices and gap penalty regimes.
The evolutionary history of this protein family has been the subject of rigorous debate. Arguments have been made in favor of both convergent and divergent evolution. Because of the general lack of sequence similarity, the ancestry of this molecule is still a mystery.
What does all this mean for alignments?

• Alignments need to be able to skip secondary structural elements to complete domains (i.e. putting gaps opposite these motifs in the shorter sequence).

• Gap situations can be even more complex in cases of alternatively spliced protein variants.

• Depending on gap penalties chosen, the algorithm might have difficulty with making such long gaps (for example when using high affine gap penalties), resulting in incorrect alignment.

• Alignments are only meaningful for homologous sequences (having a common ancestor)
Dot plots

• Way of representing (visualising) sequence similarity without doing dynamic programming (DP)
• Make search matrix as for DP, but locally represent sequence similarity by averaging using a sliding window
Dot plots are calculated using a diagonal window of preset length that is slid through the search matrix -- typically the central cell holds the window score (e.g. sum, average).
Dot-plots

*a simple way to visualise sequence similarity*

Can be a bit messy, though...

Filter:

6/10 residues have to match...
Dot-plots, what about...

- Insertions/deletions -- DNA and proteins
- Duplications (e.g. tandem repeats) – DNA and proteins
- Inversions -- DNA

Dot plots are calculated using a diagonal window of preset length that is slid through the search matrix -- typically the central cell holds the window score (e.g. sum, average)
Dot-plots, self-comparison

- Direct repeat
- Inverted repeat
- Tandem repeat
Measuring Alignment Similarity

- **Sequence identity** (number of identical exchanges per unit length)
- **Raw alignment score**
- **Sequence similarity** (alignment score normalised to a maximum possible)
- **Alignment score normalised** to a randomly expected situation (database/homology searching)
Statistical alignment score
Chance or common ancestry?

- **Alignment score normalised** to a randomly expected situation (database/homology searching)
  - Shuffle (scramble) one or both sequences
  - Calculate alignment scores
  - Repeat N times and calculate mean and stdev of the random (scrambled) alignment scores
  - Calculate real alignment score $x$

$$Z\text{-score} = \frac{(x - \text{mean})}{\text{stdev}}$$
Take-home messages

• Know global, semi-global and local alignment (three types)
• Know three types of gap penalties
• ‘easy’ (fast) and general DP algorithm
• Pitfalls of global, semi-global and local alignment
Wrapping up

• 3 types of alignment: global, semi-global, local
• 3 types of gap penalty: linear (simple scheme), affine (most widely used), concave (most biologically appropriate)
• Dynamic programming
  – Forward step: calculating cumulative probabilities
  – Trace-back: following the arrows back to reconstruct optimal (highest scoring) alignment path, and hence the optimal alignment
    • Global DP: initialising row/column with gap penalty values; trace back from lower-rightmost cell
    • Semi-global DP: no initialising row/column (zero end gaps); trace back from highest scoring cell in bottom row or rightmost column
    • Local DP (Smith-Waterman): no initialising row/column; set negative cells to zero; traceback from highest scoring cell anywhere in matrix, terminating at zero (not including) or beginning of sequence
• General DP strategy for all three types of gap penalty
  – Implementation using ‘long jumps’ searching preceding row + column
• Waterman-Eggert: multiple local alignment calculation
• Dot plots
• Statistical scoring using sequence scrambling (Z-scores)