Pairwise Sequence Alignment

BMI/CS 576
www.biostat.wisc.edu/bmi576/
Mark Craven
craven@biostat.wisc.edu

Pairwise alignment: task definition

Given
– a pair of sequences (DNA or protein)
– a method for scoring a candidate alignment

Do
– determine the correspondences between substrings in the sequences such that the similarity score is maximized
The role of homology in alignment

- **homology**: similarity due to descent from a common ancestor

- often we can infer homology from similarity

- thus we can sometimes infer structure/function from sequence similarity

Alignment of the PhaK protein from Pseudomonas putida and OprD protein from Pseudomonas aeruginos.
Homology example: evolution of the globins

Homology
- homologous sequences can be divided into two groups
  - orthologous sequences: sequences that differ because they are found in different species (e.g. human $\alpha$-globin and mouse $\alpha$-globin)
  - paralogous sequences: sequences that differ because of a gene duplication event (e.g. human $\alpha$-globin and human $\beta$-globin, various versions of both)
DNA sequence edits

- substitutions: ACGA → AGGA
- insertions: ACGA → ACCGGAGA
- deletions: ACGGAGA → AGA
- transpositions: ACGGAGA → AAGCGGA
- inversions: ACGGAGA → ACTCCGA

Mismatches and gaps

- substitutions in *homologous* sequences result in mismatches in an alignment
- insertions/deletions in *homologous* sequences result in mismatches in an alignment

CA--GATTCGAAT
CGCCGATT---AT

*gap* *mismatch*
Alignment scales

- for short DNA sequences (gene scale) we will generally only consider
  - substitutions
  - insertions/deletions

- for longer DNA sequences (genome scale) we will consider additional events
  - transpositions
  - inversions

- in this course we will focus on the case of short sequences

Insertions/deletions and protein structure

- Why is it that two “similar” sequences may have large insertions/deletions?
  - some insertions and deletions may not significantly affect the structure of a protein

loop structures: insertions/deletions here not so significant
Example alignment: globins

- figure at right shows prototypical structure of globins
- figure below shows part of alignment for 8 globins

Issues in sequence alignment

- the sequences we’re comparing typically differ in length
- there may be only a relatively small region in the sequences that matches
- we want to allow partial matches (i.e. some amino acid pairs are more substitutable than others)
- variable length regions may have been inserted/deleted from the common ancestral sequence
Types of alignment

- *global*: find best match of both sequences in their entirety
- *local*: find best subsequence match
- *semi-global*: find best match without penalizing gaps on the ends of the alignment

Scoring an alignment: what is needed?

- substitution matrix
  - $s(a,b)$ indicates score of aligning character $a$ with character $b$

- gap penalty function
  - $w(g)$ indicates cost of a gap of length $g$
Linear gap penalty function

- different gap penalty functions require somewhat different dynamic programming algorithms

- the simplest case is when a linear gap function is used

\[ w(g) = -g \times d \]

where \( d \) is a constant

- we'll start by considering this case
Scoring an alignment

• the score of an alignment is the sum of the scores for pairs of aligned characters plus the scores for gaps

• example: given the following alignment

VAHV--D--DMPNALSALSDDLHAHKL
AIQLQVTGVVVTDATLKNLGSVHVS

• we would score it by
  \[ s(V,A) + s(A,I) + s(H,Q) + s(V,L) - 3d + s(D,G) - 2d \]

The space of global alignments

• some possible global alignments for ELV and VIS

ELV  -ELV  --ELV  ELV-
VIS  VIS-  VIS--  -VIS

E-LV  ELV--  EL-V
VIS-  --VIS  -VIS

• Can we find the highest scoring alignment by enumerating all possible alignments and picking the best?
Number of possible alignments

• given sequences of length \( m \) and \( n \)

• assume we don’t count as distinct \( \text{CG} \) and \( \text{GC} \)

• we can have as few as 0 and as many as \( \min\{m, n\} \) aligned pairs

• therefore the number of possible alignments is given by

\[
\sum_{k=0}^{\min\{m, n\}} \binom{n}{k} \binom{m}{k} = \binom{n+m}{n}
\]

Number of possible alignments

• there are

\[
\binom{2n}{n} = \frac{(2n)!}{(n!)^2} \approx \frac{2^{2n}}{\sqrt{\pi n}}
\]

possible global alignments for 2 sequences of length \( n \)

• e.g. two sequences of length 100 have \( \approx 10^{77} \) possible alignments

• but we can use dynamic programming to find an optimal alignment efficiently
Pairwise alignment via dynamic programming


- *dynamic programming*: solve an instance of a problem by taking advantage of computed solutions for smaller subparts of the problem

- determine best alignment of two sequences by determining best alignment of all prefixes of the sequences

Dynamic programming idea

- consider last step in computing alignment of *AAAC* with *AGC*

- three possible options; in each we’ll choose a different pairing for end of alignment, and add this to best alignment of previous characters

\[
\begin{array}{c|c}
\text{AAA} & C \\
\text{AG} & C \\
\text{AAA} & C \\
\text{AGC} & - \\
\end{array}
\quad \begin{array}{c|c}
\text{AAAC} & - \\
\text{AG} & C \\
\end{array}
\]

consider best alignment of these prefixes

score of aligning this pair
Dynamic programming idea

• given an $n$-character sequence $x$, and an $m$-character sequence $y$

• construct an $(n+1) \times (m+1)$ matrix $F$

• $F(i, j) = \text{score of the best alignment of } x[1...i] \text{ with } y[1...j]$

DP algorithm for global alignment with linear gap penalty

• one way to specify the DP is in terms of its recurrence relation:

\[
F(i, j) = \max \begin{cases} 
F(i - 1, j - 1) + s(x_i, y_j) \\
F(i - 1, j) - d \\
F(i, j - 1) - d 
\end{cases}
\]
Initializing matrix: global alignment with linear gap penalty

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>G</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>-d</td>
<td>-2d</td>
</tr>
<tr>
<td>A</td>
<td>-d</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>-2d</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>-3d</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>-4d</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

DP algorithm sketch: global alignment

- Initialize first row and column of matrix
- Fill in rest of matrix from top to bottom, left to right
- For each $F(i, j)$, save pointer(s) to cell(s) that resulted in best score
- $F(m, n)$ holds the optimal alignment score; trace pointers back from $F(m, n)$ to $F(0, 0)$ to recover alignment
Global alignment example

• suppose we choose the following scoring scheme:

\[ s(x_i, y_i) = \]

\[ +1 \text{ when } x_i = y_i \]

\[ -1 \text{ when } x_i \neq y_i \]

\[ d \text{ (penalty for aligning with a gap) } = 2 \]

Global alignment example

\begin{array}{ccc}
A & G & C \\
\hline
A & 0 & -2 & -4 & -6 \\
A & -2 & 1 & -1 & -3 \\
A & -4 & -1 & 0 & -2 \\
A & -6 & -3 & -2 & -1 \\
C & -8 & -5 & -4 & -1 \\
\end{array}

one optimal alignment

x: A A A A C
y: A G - C
DP comments

- works for either DNA or protein sequences, although the substitution matrices used differ
- finds an optimal alignment
- the exact algorithm (and computational complexity) depends on gap penalty function (we’ll come back to this issue)

Equally optimal alignments

- many optimal alignments may exist for a given pair of sequences
- can use preference ordering over paths when doing traceback
- *highroad* and *lowroad* alignments show the two most different optimal alignments
Highroad & lowroad alignments

Computational complexity

- initialization: $O(m), O(n)$ where sequence lengths are $m, n$
- filling in rest of matrix: $O(mn)$
- traceback: $O(m + n)$
- hence, if sequences have nearly same length, the computational complexity is

$$O\left(n^2\right)$$
Local alignment

- so far we have discussed *global alignment*, where we are looking for best match between sequences from one end to the other

- often we want a *local alignment*, the best match between subsequences of $x$ and $y$

Example local alignment

- aligning my name against the sequence for dTDP-4-dehydrorhamnose reductase from the bacterium *opitutus terrae*

  …LSGAYHLAASGHTSWHGFASAIIDLMPLDARKCRAVEAIT…

  MARKCRAVEN
Local alignment motivation

- useful for comparing protein sequences that share a common **motif** (conserved pattern) or **domain** (independently folded unit) but differ elsewhere

- useful for comparing DNA sequences that share a similar **motif** but differ elsewhere

- useful for comparing protein sequences against **genomic DNA sequences** (long stretches of uncharacterized sequence)

- more sensitive when comparing highly diverged sequences

Local alignment DP algorithm


- interpretation of array values is somewhat different: $F(i, j) =$ score of the best alignment of a suffix of $x[1...i]$ and a suffix of $y[1...j]$
Local alignment DP algorithm

- the recurrence relation is slightly different than for global algorithm

\[ F(i, j) = \max \begin{cases} 
  F(i-1, j-1) + s(x_i, y_j) \\
  F(i-1, j) - d \\
  F(i, j-1) - d \\
  0 
\end{cases} \]

Local alignment DP algorithm

- initialization: first row and first column initialized with 0's
- traceback:
  - find maximum value of \(F(i, j)\); can be anywhere in matrix
  - stop when we get to a cell with value 0
Local alignment example

More on gap penalty functions

- A gap of length \( k \) is more probable than \( k \) gaps of length 1
  - A gap may be due to a single mutational event that inserted/deleted a stretch of characters
  - Separated gaps are probably due to distinct mutational events

- A linear gap penalty function treats these cases the same

- It is more common to use gap penalty functions involving two terms
  - A penalty \( d \) associated with opening a gap
  - A smaller penalty \( e \) for extending the gap
Gap penalty functions

linear
\[ w(g) = -g \times d \]

affine
\[ w(g) = \begin{cases} 
-d - (g - 1)e, & g \geq 1 \\
0, & g = 0 
\end{cases} \]

Dynamic programming for the affine gap penalty case

• to do in \( O(n^2) \) time, need 3 matrices instead of 1

\[ M(i, j) \quad \text{best score given that } x[i] \text{ is aligned to } y[j] \]

\[ I_x(i, j) \quad \text{best score given that } x[i] \text{ is aligned to a gap} \]

\[ I_y(i, j) \quad \text{best score given that } y[j] \text{ is aligned to a gap} \]
Global alignment DP for the affine gap penalty case

\[ M(i, j) = \max \begin{cases} M(i-1, j-1) + s(x_i, y_j) \\ I_x(i-1, j-1) + s(x_i, y_j) \\ I_y(i-1, j-1) + s(x_i, y_j) \end{cases} \]

\[ I_x(i, j) = \max \begin{cases} M(i-1, j) - d \\ I_x(i-1, j) - e \end{cases} \]

\[ I_y(i, j) = \max \begin{cases} M(i, j-1) - d \\ I_y(i, j-1) - e \end{cases} \]

Global alignment DP for the affine gap penalty case

- **initialization**
  \[ M(0, 0) = 0 \]
  \[ I_x(i, 0) = -d - (i-1)e \quad \text{for} \quad i > 0 \]
  \[ I_y(0, j) = -d - (j-1)e \quad \text{for} \quad j > 0 \]
  
  other cells in top row and leftmost column \( = -\infty \)

- **traceback**
  - start at largest of \( M(m, n), I_x(m, n), I_y(m, n) \)
  - stop at \( M(0, 0) \)
  - note that pointers may traverse all three matrices
Global alignment example
(affine gap penalty)

\[ d = 4, \; e = 1 \]

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
<th>A</th>
<th>C</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>M : 0</td>
<td>-\infty</td>
<td>-\infty</td>
<td>-\infty</td>
<td>-\infty</td>
<td>-\infty</td>
</tr>
<tr>
<td>( \ell_x ) : -\infty</td>
<td>-\infty</td>
<td>-\infty</td>
<td>-\infty</td>
<td>-\infty</td>
<td>-\infty</td>
</tr>
<tr>
<td>( \ell_y ) : -\infty</td>
<td>-4</td>
<td>-5</td>
<td>-6</td>
<td>-7</td>
<td>-8</td>
</tr>
</tbody>
</table>

three optimal alignments:

- AC ACT
- A A--T
- AC ACT
- A--AT
- AC ACT
- --AAT
Why three matrices are needed

• consider aligning the sequences \( \text{WFP} \) and \( \text{FW} \) using \( d = 5 \), \( e = 1 \) and the following values from the BLOSUM-62 substitution matrix:

\[
\begin{align*}
s(\text{F, W}) &= 1 \\
     s(\text{W, W}) &= 11 \\
     s(\text{F, F}) &= 6 \\
     s(\text{W, P}) &= -4 \\
     s(\text{F, P}) &= -4 \\
\end{align*}
\]

• the matrix shows the highest-scoring partial alignment for each pair of prefixes:

\[
\begin{array}{cccc}
  & W & F & P \\
W & 0 & -5 & -6 & -7 \\
F & -5 & 1 & 1 & -4 \\
W & -6 & 6 & 2 & 0 \\
\end{array}
\]

Local alignment DP for the affine gap penalty case

\[
M(i, j) = \max \begin{cases} 
M(i - 1, j - 1) + s(x_i, y_j) \\
I_x(i - 1, j - 1) + s(x_i, y_j) \\
I_y(i - 1, j - 1) + s(x_i, y_j) \\
0 
\end{cases}
\]

\[
I_x(i, j) = \max \begin{cases} 
M(i - 1, j) - d \\
I_x(i - 1, j) - e 
\end{cases}
\]

\[
I_y(i, j) = \max \begin{cases} 
M(i, j - 1) - d \\
I_y(i, j - 1) - e 
\end{cases}
\]
Local alignment DP for the affine gap penalty case

- **initialization**
  \[ M(0, 0) = 0 \]
  \[ M(i, 0) = 0 \]
  \[ M(0, j) = 0 \]
  cells in top row and leftmost column of \( I_x, I_y = -\infty \)

- **traceback**
  - start at largest \( M(i, j) \)
  - stop at \( M(i, j) = 0 \)

Gap penalty functions

- **linear:** \( w(g) = -g \times d \)

- **affine:**
  \[ w(g) = \begin{cases} 
  -d - (g - 1)e, & g \geq 1 \\
  0, & g = 0 
  \end{cases} \]

- **convex:** as gap length increases, magnitude of penalty for each additional character decreases

  e.g. \( w(g) = -d - \log(g) \times e \)
### Computational complexity and gap penalty functions

<table>
<thead>
<tr>
<th>Type</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>Affine</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>Convex</td>
<td>$O(n^2 \log n)$</td>
</tr>
<tr>
<td>General</td>
<td>$O(n^3)$</td>
</tr>
</tbody>
</table>

* assuming two sequences of length $n$

### Alignment (global) with general gap penalty function

why the general case has time complexity $O(n^3)$

$$F(i, j) = \max \begin{cases} F(i-1, j-1) + s(x_i, y_j) \\ F(k, j) + \gamma(i-k) \\ F(i, k) + \gamma(j-k) \end{cases}$$

$k$ ranges over previous coordinates

consider every previous element in the row

consider every previous element in the column
Pairwise alignment summary

- The number of possible alignments is exponential in the length of sequences being aligned.
- Dynamic programming can find optimal-scoring alignments in polynomial time.
- The specifics of the DP depend on:
  - Local vs. global alignment.
  - Gap penalty function.
- Affine penalty functions are most commonly used.