Pairwise Sequence Alignment

BMI/CS 776
www.biostat.wisc.edu/~craven/776.html
Mark Craven
craven@biostat.wisc.edu
January 2002

Pairwise Alignment: Task Definition

• Given
  – a pair of sequences (DNA or protein)
  – a method for scoring the similarity of a pair of characters
• Do
  – determine the correspondences between substrings in the sequences such that the similarity score is maximized
Motivation

- comparing sequences to gain information about the structure/function of a query sequence
- putting together a set of sequenced fragments (fragment assembly)
- comparing a segment sequenced by two different labs

The Role of Homology

- 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
Homology

• homologous sequences can be divided into two groups
  – orthologous sequences: sequences that differ because they are found in different species (e.g. human α-globin and mouse α-globin)
  – paralogous sequences: sequences that differ because of a gene duplication event (e.g. human α-globin and human β-globin, various versions of both)

Issues in Sequence Alignment

• the sequences we’re comparing probably 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
Gaps

• sequences may have diverged from a common ancestor through various types of mutations:
  – substitutions (ACGA → AGGA)
  – insertions (ACGA → ACCGA)
  – deletions (ACGA → AGA)
• the latter two will result in gaps in alignments

Insertions/Deletions and Protein Structure

*loop structures: insertions/deletions here not so significant*
Example Alignment

GSAQVKGHGKKVADALTNAHV----D----DMFNALSSALSDLHAKL
++  ++++H+ KV   +  A  ++      +L+  L+++H+  K
NNPELQAHAGKVFKLVEAAIQLQVTGVVVTDATLKNLGSVHSKG

- gaps depicted with –
- middle line shows matches
  - identical matches shown with letters
  - similar amino acids shown with +
  - dissimilar amino acids/gaps indicated by space

Alignments in the Olden Days:
Dot Plots
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

Pairwise Alignment Via Dynamic Programming

- Needleman & Wunsch, *Journal of Molecular Biology*, 1970
- dynamic programming: solve an instance of a problem by taking advantage of computed solutions for smaller subparts of the problem
- determine alignment of two sequences by determining alignment of all prefixes of the sequences
Scoring Scheme Components

- substitution matrix
  - $s(a,b)$ indicates score of aligning character $a$ with character $b$
- gap penalty function
  - $w(k)$ indicates cost of a gap of length $k$

Linear Gap Penalty Function

- different gap penalty functions require somewhat different DP algorithms
- the simplest case is when a linear gap function is used
  \[ w(k) = gk \]
  where $g$ is a constant
- we’ll start by considering this case
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

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

\[
\begin{array}{c|c}
\text{AAA} & C \\
\hline
\text{AGC} & - \\
\end{array} \quad \text{consider best alignment of these prefixes}
\]

\[
\begin{array}{c|c}
\text{AAA} & C \\
\hline
\text{AG} & C \\
\end{array} \quad \text{score of aligning this pair}
\]

Dynamic Programming Idea

- given an \( n \)-character sequence \( x \), and an \( m \)-character sequence \( y \)
- construct an \((n+1) \times (m+1)\) matrix \( F \)
- \( F[i, j] = \text{score of the best alignment of } x[1 \ldots i] \text{ with } y[1 \ldots j] \)
Announcements

• next lecture: BLAST & PSI-BLAST
  read Altshul et al., *Nucleic Acids Research*, 1997

• interested in an AI reading group for grad students? see www.cs.wisc.edu/~richm/airg/

Dynamic Programming Idea

\[
\begin{align*}
F[i-1, j-1] &+ g \\
+ s(x[i], y[j]) &+ g \\
F[i-1, j] &+ g \\
F[i, j] &
\end{align*}
\]

Dynamic Programming Idea

• in extending an alignment, we have 3 choices:
  – align $x[1...i-1]$ with $y[1...j-1]$ and match $x[i]$ with $y[i]$
  – align $x[1...i]$ with $y[1...j-1]$ and match a gap with $y[j]$
  – align $x[1...i-1]$ with $y[1...j]$ and match a gap with $x[i]$
• choose highest scoring choice to fill in $F[i,j]$

DP Algorithm for Global Alignment with Linear Gap Penalty

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

$$F(i, j) = \max \begin{cases} F(i - 1, j - 1) + s(x_i, y_j) \\ F(i - 1, j) + g \\ F(i, j - 1) + g \end{cases}$$
Initializing Matrix: Global Alignment with Linear Gap Penalty

DP Algorithm Sketch

- 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
**DP Algorithm Example**

- suppose we choose the following scoring scheme:
  
  \[ s(x[i], y[j]) = \]
  
  \[ +1 \text{ when } x[i] = y[j] \]
  \[ -1 \text{ when } x[i] \neq y[j] \]
  \[ g (\text{penalty for aligning with a gap}) = -2 \]

---

**DP Algorithm Example**

<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>-2</td>
<td>-4</td>
</tr>
<tr>
<td>A</td>
<td>-2</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>A</td>
<td>-4</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>A</td>
<td>-6</td>
<td>-3</td>
<td>-2</td>
</tr>
<tr>
<td>C</td>
<td>-8</td>
<td>-5</td>
<td>-4</td>
</tr>
</tbody>
</table>

one optimal alignment

\[ x: A A A C \]
\[ y: A G - C \]
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 loadroad alignments show the two most different optimal alignments
Highroad & Lowroad Alignments

A -2
A -4
A -6
C -8
1 -1 -3
-1 0 -2
A -5 -4 -1
C -8 -5 -4 -1

Dynamic Programming Analysis

• there are

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

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

• e.g. two sequences of length 1000 have \(\approx 10^{600}\) possible alignments

• but the DP approach finds an optimal alignment efficiently
Computational Complexity

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

$$O(n^2)$$

Local Alignment

• so far we have discussed global alignment, where we are looking for best match between sequences from one end to the other.
• more commonly, we will want a local alignment, the best match between subsequences of $x$ and $y$. 
Local Alignment Motivation

• useful for comparing protein sequences that share a common domain but differ elsewhere
• useful for comparing against genomic sequences (long stretches of uncharacterized sequence)
• more sensitive when comparing highly diverged sequences

Local Alignment DP Algorithm

• original formulation: Smith & Waterman, *Journal of Molecular Biology*, 1981
• interpretation of array values is somewhat different
  \[ F[i, j] = \text{score of the best alignment of a suffix of } x[1...i] \text{ and a suffix of } y[1...j] \]
Local Alignment DP Algorithm

• the recurrence relation is slightly different than for global algorithm

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

• 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

A A G A
0 0 0 0 0
T 0 0 0 0 0
T 0 0 0 0 0
A 0 1 1 0 1
A 0 1 2 0 1
G 0 0 0 3 1

x: A A G
y: A A G

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 \( h \) associated with opening a gap
  – a smaller penalty \( g \) for extending the gap
Gap Penalty Functions

- linear
  \[ w(k) = gk \]

- affine
  \[ w(k) = \begin{cases} 
  h + gk, & k \geq 1 \\
  0, & k = 0 
  \end{cases} \]

Dynamic Programming for the Affine Gap Penalty Case

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

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

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

  \[ I_y(i, j) \] best score given that \( y[j] \) is aligned to a gap
Global Alignment DP for the Affine Gap Penalty Case

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

\[
I_x(i, j) = \max \begin{cases} 
M(i-1, j) + h + g \\
I_x(i-1, j) + g
\end{cases}
\]

\[
I_y(i, j) = \max \begin{cases} 
M(i, j-1) + h + g \\
I_y(i, j-1) + g
\end{cases}
\]

Global Alignment DP for the Affine Gap Penalty Case

- **initialization**
  \[
  M(0, 0) = 0 \\
  I_x(i, 0) = h + g \times i \\
  I_y(0, j) = h + g \times j
  \]
  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 any of $M(0, 0), I_x(0, 0), I_y(0, 0)$
Local Alignment DP for the Affine Gap Penalty Case

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

\[ I_x(i, j) = \max \begin{cases} 
M(i-1, j) + h + g \\
I_x(i-1, j) + g 
\end{cases} \]

\[ I_y(i, j) = \max \begin{cases} 
M(i, j-1) + h + g \\
I_y(i, j-1) + g 
\end{cases} \]

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 \( M(i, j) = 0 \)
Computational Complexity and Gap Penalty Functions

- linear: \( O(n^2) \)
- affine: \( O(n^2) \)
- general: \( O(n^3) \)

Alignment (Global) with General Gap Penalty Function

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

- consider every previous element in the row
- consider every previous element in the column