Biomolecular sequence comparison

Genome Projects

Molecular biology has two important goals:

1) Identify all key molecules of a given organism especially proteins
2) Identify all main interactions between molecules.

The genome projects mainly focus on the first goal via sequencing the genome (determining the entire DNA of the organism) and performing a computational analysis, to determine all genes. This will enable identification of proteins.

See:
geta.lifesci.uic.edu/~nikos/genomes.html for the status of ongoing genome projects.

In at most a year the human genome is expected to be sequenced. Note that 99.9% of the sequences between two humans are identical. Once this is done, an interesting challenge is to identify common polymorphisms, genomic variations that occur in a non-negligible fraction of population.
Sequence analysis

Once the genome is completely sequenced we will have new challenges:

1) Identify all genes
2) Determine their functions (via hypothesizing the function through another gene with a known function whose sequence is similar)
   Sequence similarity usually implies functional similarity.
3) Identify proteins involved in gene expression regulation
4) Identify sequence repeats
5) Identify other functional regions. E.g. origins of replication, pseudo genes (genes that are not expressed), sequences responsible for compact folding of DNA etc.

Sequence Similarity:

The identification of genes and the proteins they synthesize is usually done as follows:

1) Translate the coding regions into sequences of amino acids.
2) Search for similar sequences in protein databases which contain known proteins and their functions.
   - Close matches give ideas about functions as well as 3-D structures.
String Alignment Problem

How can we determine similarity between sequences or strings?

\[ \text{e.g. } \text{a g t - c + g - } 3 \text{ - indicates } \]
\[ \text{g + a c - g a } 3 \text{ - deletion} \]

Scoring function: if a match scores +2, a mismatch or deletion scores -1 then:

\[ 4 \text{ matches } \times 2 - 4 \text{ deletes } \times 1 = 4 \]

There are many possible alignments between two strings.

If \( x, y \) are single characters or spaces then \( 6(x, y) \) denotes the score for aligning \( x \) with \( y \).

In the example above \( 6(x, y) = 2 \) for any \( x = y \in \{a, g, c, +3\ \} \) and \( 6(x, y) = -1 \) for any \( x \neq y \)

The alignment between two sequences \( S \) and \( T \) is a mapping from \( S \) to \( S' \) and \( T \) to \( T' \) such that \( |S'| = |T'| \) and the removal of \\
"_"'s from \( S' \) gives \( S \) and from \( T' \) gives \( T \).
Hence the alignment score is:
\[ \sum_{i=1}^{ls'} c(S'[i:], T'[i:]) \]

An optimal alignment is one that maximizes the above score.

An obvious algorithm to find optimal alignment:

A subsequence of a string is a sequence of characters which are not necessarily consecutive, e.g. agct is a subsequence of a gcata.

Given S, T assume \(|S| = |T| = n\). Also assume a score function which has \(c(-, -) < 0\). Thus spaces do not get aligned without a penalty.

Optimal alignment:

for \( 0 \leq i \leq n \)

for all subsequences \( A \) of \( S \) with \(|A| = i\)

for all subsequences \( B \) of \( S \) with \(|B| = i\)

Form an alignment between \( AC[i:j], BE[i:j] \)

Match rest of the characters with "-".

Determine the value of alignment.

Pick the one with best score.
Running time of this algorithm:

A string of length $n$ has $\binom{n}{i}$ subsequences of length $i$. Thus there are $\binom{n}{i}^2$ pairs of subsequences with length $i$. For such an alignment there will be $n - i$ "-" characters giving a total # of characters $n + (n - i)$ for $15'$ or $1T'$). Thus

$$
\sum_{i=0}^{n} \binom{n}{i}^2 (2n-i) \geq \sum_{i=0}^{n} \binom{n}{i}^2 \\
= \binom{2n}{n} > 2^{2n}
$$

(last inequality is from Stirling approximation)

For $n = 20$, this algorithm runs in $2^{2^{20}} = 2^{40}$ steps.
Sequence Alignment by Dynamic Programming

Given strings $S, T$ let $V(i, j)$ be the optimal alignment of prefixes $S[1:i]$ and $T[1:j]$. Our goal is to compute $V(n, m)$ where $|S| = n$, $|T| = m$

We will solve this problem iteratively, incrementing $i$ and $j$ starting from $i = 0$ and $j = 0$.

$V(0, 0) = 0$

$V(i, 0) = V(i-1, 0) + 6 (S[i], T[j])$ ($i > 0$)

$V(0, j) = V(0, j-1) + 6 (S[i], T[j])$ ($j > 0$)

Algorithm:

For $i \leftarrow 1$ to $n$

For $j \leftarrow 1$ to $m$

$V(i, j) \leftarrow \max \left[ V(i-1, j-1) + 6 (S[i], T[j]) 
V(i, j-1) + 6 (S[i], -) 
V(i-1, j) + 6 (-, T[j]) \right]$

This recurrence works because the table $V(i, j)$ reduces the computation of $V(i, j)$ to three cases:

1) If $S[i]$ is aligned with $T[j]$ then $S[1:i-1]$ should be aligned with $T[1:j-1]$ whose cost we already know to be $V(i-1, j-1)$.

2) Either a) $S[i]$ or b) $T[j]$ is aligned with a "-" to the right of $T[j]$ or $S[i]$ respectively. If this is the case the cost of the rest of the alignment is already computed: $V(i-1, j)$ (or $V(i, j-1)$ respectively).
E.g. Suppose \( d(x,y) = \begin{cases} 
2 & \text{if } x = y \\
-1 & \text{if } x \neq y \text{ or } x = y = "-" 
\end{cases} \)

Let's try to align \( S: \text{acgc+g} \)
\( T: \text{catg+} \)

<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>0</td>
<td></td>
<td></td>
<td>-1</td>
<td>-2</td>
<td>-3</td>
<td>-4</td>
</tr>
<tr>
<td>1</td>
<td>a</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>-1</td>
<td>-2</td>
</tr>
<tr>
<td>3</td>
<td>g</td>
<td>-3</td>
<td>-1</td>
<td>0</td>
<td>-1</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>
</tr>
<tr>
<td>5</td>
<td>t</td>
<td>-5</td>
<td>-2</td>
<td>-2</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>
</tr>
</tbody>
</table>

How to recover opt. alignment:

Trace the dynamic programming steps back from \((n,m)\) entry, determining which preceding entries are responsible for current one - see the highlighted entries of array \( D(i,j) \) above.

Accordingly the possible "optimal" alignments are:

\( S':\text{acgc+g} \)
\( S':\text{acac+g} \)
\( S':\text{acgc+g} \)
\( T':\text{catg+} \)
\( T':\text{cg+g} \)
\( T':\text{c+a+g} \)
Time: $O(n \cdot m)$: only 3 comparisons per table entry - same for tracing back

Notice: Computing all optimal alignments can be costly: $O(2^{\frac{nm}{2}})$ - consider $S = \ldots a a a a a \ldots a \frac{n \cdot m}{2}$

Local Similarity Search:

Problem: Given S, T find two substrings A and B (of S, T respectively) so that the alignment score between A, B is maximum among all substrings of S and T.

Naive algorithm for Local Alignment:

for all substrings A of S do
for all substrings B of S do
    Find OPT alignment between A, B via dynamic programming
    If the score is best ever, retain this value
Output best aligned A, B.

Running time: $O(n^2)$ substrings of S, $O(m^2)$ substrings of T, each need $O(n \cdot m)$ time
Total: $O(n^3 \cdot m^3)$
Faster method for Local Alignment:

Define $\phi$ to be the empty string. Let $v(i,j)$ be the maximum alignment score between a suffix $S[i':i]$ and a suffix $T[j':j]$. The maximum $v(i,j)$ entry gives the best alignment: a substring is a prefix of a suffix.

If $c(x,-) \leq 0$ and $c(-,y) \leq 0$ then $v(i,0) = 0$ and (optimal alignment is between two empty suffixes).

Algorithm:

For $i \leftarrow 1$ to $n$

For $j \leftarrow 1$ to $m$

$v(i,j) \leftarrow \max\left[ 0,
\begin{align*}
&v(i-1,j-1) + c(\text{SE}_i,\text{TE}_j) \\
&v(i-1,j) + c(\text{SE}_i,-)
\end{align*}
\right]

This is pretty much identical to global alignment.
The reason is one needs to find an alignment between one suffix of $T$ with another suffix of $S$. The suffix can be:

1) empty for both (one empty one non-empty can not be optimal)
2) both non-empty and $\text{TE}_j$ is and $\text{SE}_i$ are aligned
3) $\text{SE}_i$ is aligned with "-" and hence $v(i-1,j)$ gives the best suffix alignment for the rest.
4) $\text{TE}_j$ is aligned with "-".
e.g. $S: abcddef$, $T: fffcde$

$L(x,y) = \begin{cases} 
\delta + 2 & \text{if } x=y \\
-1 & \text{if } x \neq y \text{ or } x=y = ''
\end{cases}$

<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>0</td>
<td>f</td>
<td>f</td>
<td>f</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td></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></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></td>
</tr>
<tr>
<td>3</td>
<td>c</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>f</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>d</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>e</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>f</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

OPT local alignment score is $5$

OPT Alignments are $cfdde$, $f-de$, $c-de$, $f-cde$

Time: $O(nm)$ for table construction

Space: $O(nm)$: the original formulation of Smith-Waterman

(can be improved to $O(n+m)$): see Hirschberg
Gaps

In the GC(.,.) model each mismatch has a fixed penalty independent from the surroundings. Sometimes you may want to give penalties decreasing with "gap" size.

Affine gap model:

Let the penalty of a gap (i.e. a mismatch of a character with blank symbol) be \( W_g + q \) \( W_s \) where \( W_g \) and \( W_s \) are constants and \( q \) is the size of the gap. So far we have fixed \( W_g = 0 \).

Global Alignment under Affine Gap Model

\[
\text{Similarity score} = \sum_{i=1}^{l} \left( S'(i, j), T'(i, j) \right) - W_g \cdot \# \text{gaps} - W_s \cdot \# \text{spaces}
\]

Here \( l = |S'| = |T'| \)

Thus dynamic programming can be updated as:

\[
\begin{align*}
V(0, 0) &= 0 \\
V(i, 0) &= -W_g - iW_s \quad \text{for } i > 0 \\
V(0, j) &= -W_g - jW_s \quad \text{for } j > 0 \\
E(i, 0) &= -\infty \quad \text{for } i > 0 \\
F(0, j) &= -\infty \quad \text{for } j > 0
\end{align*}
\]
Algorithm

For i ← 1 to n
    For j ← 1 to m
        \[ V(i, j) = \max \left( 6(i, j), F(i, j), E(i, j) \right) \]
        \[ 6(i, j) = V(i-1, j-1) + 6(SEi, TEj) \]
        \[ F(i, j) = \max \left( F(i-1, j) - W_s, V(i-1, j) - W_g - W_s \right) \]
        \[ E(i, j) = \max \left( E(i, j-1) - W_s, V(i, j-1) - W_g - W_s \right) \]

\[ V(i, j) \]: value of opt alignment of \( SEi \) with \( TEj \)
\[ 6(i, j) \]: value of opt alignment whose last pair matches \( SEi \) with \( TEj \)
\[ F(i, j) \]: here \( SEi \) is matched with a space
\[ E(i, j) \]: here \( TEj \) is matched with a space

Time Analysis: \( O(nm) \) time for preparing tables \( V, 6, F \)