EE381V: Genomic Signal Processing

Lecture #6

Today

• A brief review of alignment in practice
  – heuristic algorithms: FASTA, BLAST
  – Burrows-Wheeler transformation

• Multiple sequence alignment

• Hidden Markov Model in sequence analysis
Suppose that we have two strings of approximately the same length (m and n) \( t \) and \( s \).

- If the optimal alignment of \( s \) and \( t \) has few gaps, then the path of an alignment will be close to the diagonal.
- So, search inside a diagonal band of the matrix.

If the width of the band is \( k \), the dynamic programming takes \( O(kn) \) time.

A few details to address:

- Where is the banded diagonal?
  - Need not be the main diagonal.
- Which subsequences to select?
  - Heuristically find potential diagonals and evaluate them.
- Heuristic sequence searching packages: FASTA, BLAST.
  - FASTA: allows gaps in the alignment; BLAST: primarily ungapped.
• Uses multistep approach to find local high scoring
  • start from exact short matches and then attempt to extend them

Formalizing the algorithm:
• Input: strings s and t, and a parameter “ktup”
  • The parameter “ktup” is the length of short exact matches that we will look for. It should be long enough to efficiently initiate the alignment, but not too long so that the matches are rare
• Output: A highly scored local alignment obtained via the following steps
  1. Find pairs of matching substrings (seeds)
     \[ s[i \ldots i+ktup] = t[j \ldots j+ktup] \]
  2. Extend to ungapped diagonals
  3. Extend to gapped matches using banded diagonal

• The seeds are typically short (4-6 for DNA, 2 for proteins)
  • otherwise, we find fewer potential diagonals
BLAST

- BLAST: Basic Local Alignment Search Tool
- Similar idea to FASTA
  - search for short seeds (typically, length 11 for DNA, 3 for proteins)
  - attempt to extend them as far as possible in a greedy manner
  - extensions are scored until the alignment score drops below a threshold (typically determined in a statistical manner)
  - the maximal-scoring segments are combined where possible

BLAST

- Statistical significance of the solution
Burrows-Wheeler Transform

- Idea: Find exact matches of query substrings in the database
  - find an alignment only near the queries
- Burrows & Wheeler, 1994
  1. Original text = “googol”
  2. Append ‘$’ to mark the end = X = “googol$”
  3. Sort all rotations of the text in lexicographic order
  4. Take the last column.

```
0: googol$
1: googol$
g
2: oogol$g
3: gol$go
4: l$goog
5: l$goog
6: l$goog
```

String Sorting

Suffix array
Burrows-Wheeler Transform

- All occurrences of substrings with a common suffix, W appear next to each other, defining an interval \([ R(W), \overline{R}(W) ]\)

- Suffix array interval of “go” = [1,2]

\[
\begin{align*}
W &= \text{“go”} \\
0 &\rightarrow 6: \text{googo } l \\
1 &\rightarrow 3: \text{go } l \text{go } o \\
2 &\rightarrow 0: \text{googo } l \$ \\
3 &\rightarrow 5: \$ \text{goog } o \\
4 &\rightarrow 2: \text{go } ol \$ \text{go } o \\
5 &\rightarrow 4: \text{go } ol \$ \text{go } o \\
6 &\rightarrow 1: \text{googol } \$
\end{align*}
\]

\[X = \text{googol } \$
\]

Burrows-Wheeler Transform

- Calculate the SA interval of the query \(W\)
- Can be done iteratively from the end of \(W\):

\[
\begin{align*}
R(aW) &= C(a) + O(a, R(W) - 1) + 1 \\
\overline{R}(aW) &= C(a) + O(a, \overline{R}(W)) \\
\end{align*}
\]

if \(W\) is an empty string, \(R(W) = 1\) and \(\overline{R}(W) = n - 1\)

\[
\begin{align*}
C(a) &= \text{Number of symbols in } X[0, n-2] \text{ that are lexicographically smaller than } a \\
O(a, i) &= \text{Number of occurrences of } a \text{ in } B[0, i]
\end{align*}
\]
BWT: Inexact Matching

- Search for SA intervals of substrings of X that match query W with no more than z mismatches
- Start from the end of W and keep moving toward the start
- Prefix with each possible base and look for match
- If match requires a base different from that in the real query, increment mismatch count
- Abort if mismatch count exceeds z

Today

- A brief review of alignment in practice
  - heuristic algorithms: FASTA, BLAST
  - Burrows-Wheeler transformation
- Multiple sequence alignment
- Hidden Markov Model in sequence analysis
Multiple Sequence Alignment

Find a common alignment of several sequences simultaneously

• multiple similarity implies a common structure/function/evolutionary source
• more information than in the pair-wise case (e.g., match a protein against a family of proteins)
• clearly, even more possible alignments than in the pair-wise case:

Possible alignment

\[ S_1 = \text{AGGTC} \]
\[ S_2 = \text{GTTCG} \]
\[ S_3 = \text{TGAAC} \]

A formal description of the task:

• A multiple (global) alignment of sequences \( S_1, S_2, \ldots, S_k \) is a series of sequences \( S'_1, S'_2, \ldots, S'_k \) which may contain gaps, where
  1. \( |S'_1| = |S'_2| = \ldots = |S'_k| \) (the lengths of \( S'_1, S'_2, \ldots, S'_k \) are the same)
  2. \( S'_i \) is an extension of \( S_i \) obtained by insertion of spaces, i.e., the removal of gaps from \( S'_i \) leaves \( S_i \)

• The alignment is typically represented by a matrix

\[ A = \begin{bmatrix}
M & Q & - & I & L & L & L \\
M & L & R & - & L & L & - \\
M & K & - & I & L & L & L \\
M & P & P & V & L & L & -
\end{bmatrix} \]

• No column consists of spaces only
• Possible scoring function: score each column, score(A) = \( \Sigma \text{score}(\text{column}(i)) \)
Dynamic Programming Solution

The best alignment of \( r \) sequences: use an \( r \)-dimensional hypercube of scores

- Let \( D(j_1,j_2,\ldots,j_r) \) be the best scores for aligning prefixes of lengths \( j_1,j_2,\ldots,j_r \) of the sequences \( x_1,x_2,\ldots,x_r \).
- Set the initial condition \( D(0,0,\ldots,0) = 0 \)

- We compute
  \[
  D(j_1,j_2,\ldots,j_r) = \max_{\epsilon \in \{0,1\}^r, \epsilon \neq (1,1,\ldots,1)} (D(j_1 - \epsilon_1, \ldots, j_r - \epsilon_r) + c(\epsilon_1, \ldots, \epsilon_r))
  \]

- Note: \( c \) is the cost function, \( \epsilon \) is the direction of extending alignment
  - compare with 2D direction:

    \[
    \begin{array}{c|c}
    F(i-1,j-1) & F(i-1,j) \\
    \hline
    F(i,j-1) & F(i,j)
    \end{array}
    \]

Sum-of-pairs scores

- A problem with the former: how to come up with meaningful scores (i.e., \( c \))
  - e.g., taking a probabilistic approach, scoring 3-sequence alignment would require finding joint probabilities \( p(a,b,c) \) for all \( (a,b,c) \) from the alphabet
  - not enough reliable training data to do this!

- An alternative: sum of scores of projected pair-wise alignments

\[
A = \begin{array}{cccccc}
M & Q & _ & I & L & L \\
M & L & R & - & L & L \\
M & K & _ & I & L & L \\
M & P & P & V & L & I & L \\
\end{array}
\]

\[
SP_{score}(A) = \sum_{i<j} \text{score}(x^i, x^j)
\]

- \( x^i \) denotes the \( i^{th} \) sequence

- A column may have several indels and thus we need to define \( c(-,-) \)
  - typically, it is set to 0 since it does not affect the projected pairwise alignments
Sum-of-pairs scores

- **Sum-of-pairs (SP)** score for a multiple global alignment is the sum of scores of all pairwise alignments induced by it.
  - There are \( \binom{k}{2} \) such alignments if we have the total of \( k \) sequences.
  - As an example, consider the following:
    
    \[
    \begin{align*}
    \text{AC-CDB-} \\
    \text{--C-ADBD} \\
    \text{A-BCDAD}
    \end{align*}
    \]
  - Adopt the following scoring scheme: match = 0, mismatch/indel = -1.
  - Then the sum-of-pairs score is \( S_{\text{score}} = -3-4-5 = -12. \)
  - To find the best alignment of \( r \) sequences: still use an \( r \)-dimensional hypercube of scores.
  - E.g., we simply replace \( c(a,b,c) \) by \( c(a,b)+c(b,c)+c(a,c) \).
  - Nothing else changes in terms of the dynamic program execution.

Complexity of finding a multiple alignment

- Assume that we have \( k \) strings of length \( n_i \) – the previously mentioned dynamic program is much more complex than in the pair-wise case:
  - Instead of a 2-dimensional table we have a \( k \)-dimensional table.
  - Each dimension is of length \( n_i + 1 \).
  - Each entry depends on \( 2^{k-1} \) adjacent entries.
- As a result, the complexity is exponential in the number of sequences.
  - Complexity: \( O(2^k n_i^k) \).
- If all the lengths are equal, the complexity is \( O(2^k n^k) \).
  - The problem is known to be NP-hard (the proof for sum-of-pairs scoring).
- As in pair-wise case, there are computationally efficient heuristics:
  - MSA algorithms [Carrilo & Lipman]
  - The (familiar) idea: if the subsequences are relatively similar, the alignment path is close to the diagonal of the hypercube.
Another look at the sum-of-pairs scores

- Introduced to avoid having to come up with \( p(a,b,c) \) for all \((a,b,c)\) from the alphabet
  - not enough reliable training data to do this!
- Using pair-wise scores and summing them up is easy to do
  - how meaningful it is?

![Sum-of-pairs alignment](image)

- The correct extension of the probabilistic framework for multiple scoring would be to evaluate
  \[
  c(a, b, c) = \log \frac{p(a, b, c)}{q(a)q(b)q(c)}
  \]
- Instead, in sum-of-pairs we evaluate
  \[
  c(a, b, c) = \log \frac{p(a, b)}{q(a)q(b)} + \log \frac{p(a, c)}{q(a)q(c)} + \log \frac{p(b, c)}{q(b)q(c)}
  \]

Alternative alignment schemes

- So, the probabilistic argument supporting sum-of-pairs scoring is not very convincing
  - each sequence is scored as if it descended from \( n-1 \) other sequences instead of a common ancestor
- We may consider schemes other than sum-of-pairs but with scores still fundamentally defined by pair-wise distances

![Sum-of-pairs alignment](image)  ![Star alignment](image)  ![Tree-based alignment](image)
**Star alignment**

The ‘star’ algorithm: Select a fixed sequence $S'$ as a center, and align all to it

Input: $\Gamma$ - set of $k$ strings $S_1, \ldots, S_k$.

0. For each $i<j$ calculate $D(S_i, S_j)$.
1. Find the string $S'$ (center) that minimizes $\sum_{S \in \Gamma} D(S', S)$.
2. Denote $S'_1=S'$ and the rest of the strings as $S_2, \ldots, S_k$.
3. Iteratively add $S_2, \ldots, S_k$ to the alignment as follows:
   a. Suppose $S_1, \ldots, S_{i-1}$ are already aligned as $S'_1, \ldots, S'_{i-1}$.
   b. Align $S_i$ to $S'_1$ to produce $S'_i$ and $S''_1$ aligned.
   c. Adjust $S'_2, \ldots, S'_{i-1}$ by adding spaces where spaces were added to $S''_1$.
   d. Replace $S'_1$ by $S''_1$.

Essentially, every time we visit step 3., another sequence is aligned with the center, keeping old gaps and possibly adding new ones.

- complexity: $O(k^2n^2)$!

---

**Tree alignment**

Input: $X$ - set of sequences

$T$ – phylogenetic tree on $X$ (leaves labeled by $X$)

Output: labels on internal vertices of $T$, s.t. sum of costs of all edges of $T$ is min

Tree-based alignment

- Consider a tree representation of the alignment problem
  - associate a sequences with each node
  - tree score is computed as a sum of scores of all leaves
- Useful in phylogenetics, we will not spend time on it…