Computational Biology

Lecture 7

Database Search

- Quadratic complexity not suitable for searching large databases
  - e.g., need to compare a query sequence to all sequences in a large database.
  - Alternative: Heuristics
    - BLAST
    - FAST

- Simple scoring scheme such as (+1, -1, -2) is not suitable for comparing protein sequences.
  - e.g., amino acids of similar size are more likely to get substituted for one another.
  - Alternative: Substitution matrix, $S(a,b)$ = score for aligning $a$ with $b$
    - General approach for substitution matrices
      - PAM
      - BLOSUM

BLAST
(Basic Local Alignment Search Tool)

- BLAST returns a list of high scoring segment pairs between the query sequence and sequences in the database.

- A segment is a substring of a sequence.

- A segment pair is a pair of segments of the same length that can form a gapless alignment.

- Basic BLAST is ungapped.

- Given a query sequence, BLAST returns all segment pairs between the query and a database sequence with score above a threshold $S$.

- $S$ can be set by the user.
**HOW does BLAST work?**

- It finds certain “seeds” which are very short segment pairs between the query and the database sequence.
- These seeds are then extended in both directions without gaps, until the maximum possible score for extensions is reached.
- Time reduction: the extension stops when the score falls below a carefully computed limit $X$.

**BLAST Algorithm**

- For a given query sequence, compile a list of short high scoring strings (words in BLAST jargon)
- Search for hits – each hit gives a “seed”
- Extend “seeds”
- Return segments pairs with score > $S$.

**$k$-mers**

- How is the list of short high scoring strings obtained?
- $k$-mers: substrings of length $k$.
  - DNA sequence: all $k$-mers.
  - Protein sequence: all $k$-mers in addition to neighboring $k$-mers. A neighboring $k$-mer is a $k$ length string that scores high with some $k$-mer of the sequence.
- Typical $k$: 3 or 4
Database

- The database is hashed and indexed by all words of size $k$.
- Each word will point to the locations where it exists in the database.
- We have only $4^k$ words in case of DNA sequences and $20^k$ words in case of proteins.
- This is much less than the number of sequences stored in the database.

Overview

- If it scores $> S$, return segment pair.
- Split query into overlapping words of length $k$ (k-mers).
- For each word, find neighboring words that score at least $T$.
- Look into database where these words occur: seeds
- Extend each seed until score drops below $X$.
Generating neighbors

- For every amino acid in the word, try all possibilities
- Score the words
- Keep those with within threshold

Looking in database

- Each neighboring word gives a list of locations where it's found
- Follow pointers to obtain seeds

Extending seeds

- Extend seed until score drops below X.
- Return highest scoring segment pair.
Example (thanks to serafim badriglou)

Why \( k \)-mers make sense?

- If two sequences have some level of similarity (say \( L \%)\), they must contain a preserved \( k \)-mer for some \( k \).

- Why?
  - pigeonhole principle!

Example pigeonhole

- If we have 91 smurfs and 10 holes, there must be at least one hole with at least 10 smurfs.

- Proof: if none of the holes contain 10 smurfs, we have at most \( 9 \times 10 = 90 \) smurfs!
Application to k-mers

- Two sequences of length 100 with > 90% similarity.
- There must be a preserved 10-mer.

Random model

- In the previous model, we cannot guarantee a k-mer for k > 10.
- What happens if we distribute the 91 similarities randomly?
- We get even better chance of having k-mers for other k.

Running time

- n: length of query sequence
- s: number of seeds
- L: length of alignment
- Running time = O(n + Ls)
- For one sequence in the database, s = O(n), L = O(n) ⇒ O(n²)
  But in practice faster than Smith-Waterman.
Variations

- 2-hit BLAST
  - Require two seeds that are within 40 amino acids of each other to start considering a database sequence.
  - Reduce the space of potential hits, speeding up the algorithm.

- Gapped BLAST
  - BLAST with gaps, find a seed, then find more seeds and extend them, then join segments with gaps in a band around the main seed.

FAST

- Record all occurrences of windows of certain size \( k \) in the two sequences \( x \) and \( y \) (1-2 for DNA, 3-4 for proteins).

- If a window occurs at \( x_i \) and at \( y_j \), we say it occurs at an offset \( i - j \).

- Offset range is \( 1 - n \) to \( m - 1 \).

Example

- Window of size 2
- \( x = \text{AGAGAG} \)
- \( y = \text{AAGAGAG} \)

  - The window AG occurs at \( x_i \) and \( y_j \), so it occurs at offset \( 1 - 4 = -3 \). It also occurs at other offsets.

  - What does it mean? Aligning \( x \) and \( y \) at offset -3 aligns the window AG.

\[
\begin{array}{c}
\text{AGAGAG} \\
\text{AAGAGAG}
\end{array}
\]

- What is the offset that maximizes the number of aligned windows?
FAST algorithm

- Need
  - lookup table: contains all possible windows of size $k$, e.g., 4$^k$ and their occurrence in $x$ and $y$.
  - Offset vector: for each offsets, holds how many times that offset occurred.
- Fill the lookup table
- Compute the offset vector
- Choose the most frequent offset
- Align $x$ and $y$ at that offset

Example

$x = $ AGAG
$y = $ AAGAG

<table>
<thead>
<tr>
<th></th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
<th>$x_5$</th>
<th>$y_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>AA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AG</td>
<td>$x_1$</td>
<td>$x_2$</td>
<td>$x_3$</td>
<td>$x_4$</td>
<td>$x_5$</td>
<td>$y_2$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GA</td>
<td>$x_2$</td>
<td>$x_3$</td>
<td>$x_4$</td>
<td>$x_5$</td>
<td></td>
<td>$y_3$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Variation

Run a bounded dynamic programming in a band centered at the offset diagonal.