Chapter 7: Rapid alignment methods: FASTA and BLAST

- The biological problem
- Search strategies
- FASTA
- BLAST
The biological problem

- Global and local alignment algorithms are slow in practice
- Consider the scenario of aligning a *query sequence* against a large database of sequences
  - New sequence with unknown function

- For instance, the size of NCBI GenBank in January 2007 was 65,369,091,950 bases (61,132,599 sequences)
Problem with large amount of sequences

- Exponential growth in both number and total length of sequences
- Possible solution: Compare against model organisms only
- With large amount of sequences, changes are that matches occur by random
  - Need for statistical analysis
Application of sequence alignment: shotgun sequencing

- Shotgun sequencing is a method for sequencing whole-organism genomes
  - First, a large number of short sequences (~500-1000 bp), or *reads* are generated from the genome
  - Reads are contiguous subsequences (substrings) of the genome
  - Due to sequencing errors and repetitions in the reads, the genome has be covered multiple times by reads
Shotgun sequencing

- Ordering of the reads is initially unknown
- Overlaps resolved by aligning the reads
- In a $3 \times 10^9$ bp genome with 500 bp reads and 5x coverage, there are $\sim 10^7$ reads and $\sim 10^7(10^7-1)/2 = \sim 5 \times 10^{13}$ pairwise sequence comparisons
Shotgun sequencing

- ~$5 \times 10^{13}$ pairwise sequence comparisons
- Recall that local alignment takes $O(nm)$ time, where $n$ and $m$ are sequence lengths
- Already with $n=m=500$, the computation cost is prohibitive
Search strategies

- How to speed up the computation?
  - Find ways to limit the number of pairwise comparisons
- Compare the sequences at word level to find out common words
  - Word means here a k-tuple (or a k-word), a substring of length $k$
Analyzing the word content

- Example query string $I$: TGATGATGAAGACATCAG
- For $k = 8$, the set of $k$-tuples of $I$ is
  
  TGATGATG
  GATGATGA
  ATGATGAA
  TGATGAAG
  GACATCAG
  ...

GACATCAG
Analyzing the word content

- There are $n-k+1$ $k$-tuples in a string of length $n$
- If at least one word of $I$ is not found from another string $J$, we know that $I$ differs from $J$
- Need to consider statistical significance: $I$ and $J$ might share words by chance only
- Let $n=|I|$ and $m=|J|$
**Word lists and comparison by content**

- The k-words of I can be arranged into a table of word occurrences $L_w(I)$
- Consider the k-words when $k=2$ and $I=\text{GCATCGGC}$:
  
  $\text{GC, CA, AT, TC, CG, GG, GC}$

<table>
<thead>
<tr>
<th>K-Word</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>AT</td>
<td>3</td>
</tr>
<tr>
<td>CA</td>
<td>2</td>
</tr>
<tr>
<td>CG</td>
<td>5</td>
</tr>
<tr>
<td><strong>GC</strong></td>
<td><strong>1, 7</strong></td>
</tr>
<tr>
<td>GG</td>
<td>6</td>
</tr>
<tr>
<td>TC</td>
<td>4</td>
</tr>
</tbody>
</table>

Start indices of k-word GC in I:

Building $L_w(I)$ takes $O(n)$ time.
Common k-words

- Number of common k-words in I and J can be computed using \( L_w(I) \) and \( L_w(J) \)
- For each word \( w \) in I, there are \( |L_w(J)| \) occurences in J
- Therefore I and J have \( \sum_w |L_w(I)||L_w(J)| \) common words

This can be computed in \( O(n + m + 4^k) \) time
  - \( O(n + m) \) time to build the lists
  - \( O(4^k) \) time to calculate the sum
Common k-words

I = GCATCGGC
J = CCATCGGCCATCG

<table>
<thead>
<tr>
<th>L_w(I)</th>
<th>L_w(J)</th>
<th>Common words</th>
</tr>
</thead>
<tbody>
<tr>
<td>AT: 3</td>
<td>AT: 3, 9</td>
<td>2</td>
</tr>
<tr>
<td>CA: 2</td>
<td>CA: 2, 8</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>CC: 1, 7</td>
<td>0</td>
</tr>
<tr>
<td>CG: 5</td>
<td>CG: 5, 11</td>
<td>2</td>
</tr>
<tr>
<td>GC: 1, 7</td>
<td>GC: 6</td>
<td>2</td>
</tr>
<tr>
<td>GG: 6</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>TC: 4</td>
<td>TC: 4, 10</td>
<td>2</td>
</tr>
</tbody>
</table>

10 in total
Properties of the common word list

- Exact matches can be found using binary search (e.g., where TCGT occurs in I?)
  - $O(\log 4^k)$ time
- For large $k$, the table size is too large to compute the common word count in the previous fashion
- Instead, an approach based on merge sort can be utilised (details skipped, see course book)
- The common $k$-word technique can be combined with the local alignment algorithm to yield a rapid alignment approach
Chapter 7: Rapid alignment methods: FASTA and BLAST

- The biological problem
- Search strategies
  - FASTA
  - BLAST
FASTA

- FASTA is a multistep algorithm for sequence alignment (Wilbur and Lipman, 1983)
- The sequence file format used by the FASTA software is widely used by other sequence analysis software
- Main idea:
  - Choose regions of the two sequences that look promising (have some degree of similarity)
  - Compute local alignment using dynamic programming in these regions
FASTA outline

FASTA algorithm has five steps:

1. Identify common k-words between I and J
2. Score diagonals with k-word matches, identify 10 best diagonals
3. Rescore initial regions with a substitution score matrix
4. Join initial regions using gaps, penalise for gaps
5. Perform dynamic programming to find final alignments
Dot matrix comparisons

- Word matches in two sequences I and J can be represented as a *dot matrix*

- Dot matrix element (i, j) has "a dot", if the word starting at position i in I is identical to the word starting at position j in J

- The dot matrix can be plotted for various k
Dot matrix (k=1,4,8,16) for two DNA sequences
X85973.1 (1875 bp)
Y11931.1 (2013 bp)
Dot matrix 
(k=1,4,8,16) for two 
protein sequences 
CAB51201.1 (531 aa) 
CAA72681.1 (588 aa)

Shading indicates now the match score according to a score matrix (Blosum62 here)
Computing diagonal sums

- We would like to find high scoring diagonals of the dot matrix
- Let's index diagonals by the offset, \( l = i - j \)
Computing diagonal sums

As an example, let's compute diagonal sums for $I = \text{GCATCGGC}$, $J = \text{CCATCGCCATCG}$, $k = 2$

1. Construct $k$-word list $L_w(J)$

2. Diagonal sums $S_1$ are computed into a table, indexed with the offset and initialised to zero

<table>
<thead>
<tr>
<th>1</th>
<th>-10</th>
<th>-9</th>
<th>-8</th>
<th>-7</th>
<th>-6</th>
<th>-5</th>
<th>-4</th>
<th>-3</th>
<th>-2</th>
<th>-1</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>$S_1$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Computing diagonal sums

3. Go through k-words of I, look for matches in $L_w(J)$ and update diagonal sums

For the first 2-word in I, GC, $L_{GC}(J) = \{6\}$.

We can then update the sum of diagonal $l = i - j = 1 - 6 = -5$ to $S_{-5} := S_{-5} + 1 = 0 + 1 = 1$
Computing diagonal sums

3. Go through k-words of I, look for matches in $L_w(J)$ and update diagonal sums

Next 2-word in I is CA, for which $L_{CA}(J) = \{2, 8\}$.

Two diagonal sums are updated:

$I = i - j = 2 - 2 = 0$
$S_0 := S_0 + 1 = 0 + 1 = 1$

$I = i - j = 2 - 8 = -6$
$S_{-6} := S_{-6} + 1 = 0 + 1 = 1$
Computing diagonal sums

3. Go through k-words of I, look for matches in $L_w(J)$ and update diagonal sums

Next 2-word in I is AT, for which $L_{AT}(J) = \{3, 9\}$.

Two diagonal sums are updated:

$I = i - j = 3 - 3 = 0$

$S_0 := S_0 + 1 = 1 + 1 = 2$

$I = i - j = 3 - 9 = -6$

$S_{-6} := S_{-6} + 1 = 1 + 1 = 2$
Computing diagonal sums

After going through the k-words of I, the result is:

\[
\begin{array}{cccccccccccccccc}
1 & 10 & -9 & -8 & -7 & -6 & -5 & -4 & -3 & -2 & -1 & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\
S_1 & 0 & 0 & 0 & 0 & 4 & 1 & 0 & 0 & 0 & 0 & 4 & 1 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]
Algorithm for computing diagonal sum of scores

\[ S_l := 0 \text{ for all } 1 - m \leq l \leq n - 1 \]

Compute \( L_w(J) \) for all words \( w \)

for \( i := 1 \) to \( n - k - 1 \) do

\[ w := l_i l_{i+1} \ldots l_{i+k-1} \]

for \( j \in L_w(J) \) do

\[ l := i - j \]

\[ S_l := S_l + 1 \]

end

end

Match score is here 1
FASTA outline

FASTA algorithm has five steps:

1. Identify common k-words between I and J
2. Score diagonals with k-word matches, identify 10 best diagonals
3. Rescore initial regions with a substitution score matrix
4. Join initial regions using gaps, penalise for gaps
5. Perform dynamic programming to find final alignments
Rescoring initial regions

- Each high-scoring diagonal chosen in the previous step is rescored according to a score matrix.
- This is done to find subregions with identities shorter than $k$.
- Non-matching ends of the diagonal are trimmed.

$$
\begin{align*}
I: & \quad C \quad C \quad A \quad T \quad C \quad G \quad C \quad C \quad A \quad T \quad C \quad G \\
J: & \quad C \quad C \quad A \quad C \quad G \quad C \quad A \quad A \quad T \quad C \quad A \\
\end{align*}
$$

75% identity, no 4-word identities

$$
\begin{align*}
I': & \quad C \quad C \quad A \quad T \quad C \quad G \quad C \quad C \quad A \quad T \quad C \quad G \\
J': & \quad A \quad C \quad A \quad T \quad C \quad A \quad A \quad A \quad A \quad A \quad A \\
\end{align*}
$$

33% identity, one 4-word identity
Joining diagonals

- Two offset diagonals can be joined with a gap, if the resulting alignment has a higher score
- Separate gap open and extension are used
- Find the best-scoring combination of diagonals
FASTA outline

FASTA algorithm has five steps:

1. Identify common k-words between I and J
2. Score diagonals with k-word matches, identify 10 best diagonals
3. Rescore initial regions with a substitution score matrix
4. Join initial regions using gaps, penalise for gaps
5. *Perform dynamic programming to find final alignments*
Local alignment in the highest-scoring region

- Last step of FASTA: perform local alignment using dynamic programming around the highest-scoring.
- Region to be aligned covers \(-w\) and \(+w\) offset diagonal to the highest-scoring diagonals.
- With long sequences, this region is typically very small compared to the whole \(n \times m\) matrix.

Dynamic programming matrix \(M\) filled only for the green region.
Properties of FASTA

- Fast compared to local alignment using dynamic programming only
  - Only a narrow region of the full matrix is aligned
- Increasing parameter k decreases the number of hits: increases specificity, decreases sensitivity
- FASTA can be very specific when identifying long regions of low similarity
  - Specific method does not produce many incorrect results
  - Sensitive method produces many of the correct results
Properties of FASTA

- FASTA looks for initial exact matches to query sequence
  - Two proteins can have very different amino acid sequences and still be biologically similar
  - This may lead into a lack of sensitivity with diverged sequences
Demonstration of FASTA at EBI

- http://www.ebi.ac.uk/fasta/

- Note that parameter ktup in the software corresponds to parameter k in lectures