Lecture 18:
Approximate Pattern Matching

Study Chapter 9.6 – 9.8
Approximate vs. Exact Pattern Matching

- Previously we have discussed exact pattern matching algorithms.
- Usually, because of mutations, it makes much more biological sense to find approximate pattern matches.
- Biologists often use fast heuristic approaches to find approximate matches.
Heuristic Similarity Searches

• **Why heuristics?**
  – Genomes are huge: Smith-Waterman quadratic alignment algorithms are too slow

• **Observation:** Good alignments of two sequences usually have short identical or highly similar subsequences

• **Many heuristic methods** (i.e., BLAST, FASTA) are based on the idea of *filtration*
  – Find short exact matches, and use them as “seeds” for potential match extension
  – “Filter” out positions with no extendable matches
Dot Plot

• A dot matrix or dot plot shows similarities between two sequences

• FASTA makes an implicit dot matrix of length \( l \) matches,
  - tries to find long diagonals (allowing for some mismatches)

• Nucleotide matches

\[ l = 1 \]
Dot Plot

- A dot matrix or dot plot shows similarities between two sequences.
- FASTA makes an implicit dot matrix of length $l$ matches,
  - tries to find long diagonals (allowing for some mismatches)
- Dinucleotide matches

$$l = 2$$
Dot Plot

- Identify diagonals above a threshold length

- Diagonals in the dot matrix indicate exact substring matching

\[ l = 2 \]
Diagonals in Dot Plots

- Extend diagonals and try to link them together, allowing for minimal mismatches/indels
- Linking diagonals reveals approximate matches over longer substrings

<table>
<thead>
<tr>
<th></th>
<th>G</th>
<th>A</th>
<th>T</th>
<th>T</th>
<th>C</th>
<th>G</th>
<th>C</th>
<th>T</th>
<th>T</th>
<th>A</th>
<th>G</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*</td>
</tr>
<tr>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*</td>
</tr>
<tr>
<td>G</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*</td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td></td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td></td>
</tr>
</tbody>
</table>

\(l = 2\)
A Realistic Dot-Plot

• On the right is a dot-plot of approximately ~200 KB of genomic sequence compared to itself.
• L = 20 with >= 90% concordance
• What do the off diagonal traces represent?
Approximate Pattern Matching (APM)

- **Goal**: Find all approximate occurrences of a pattern in a text

- **Input**:
  - pattern $p = p_1 \ldots p_n$
  - text $t = t_1 \ldots t_m$
  - the maximum number of mismatches $k$

- **Output**: All positions $1 \leq i \leq (m - n + 1)$ such that $t_i \ldots t_{i+n-1}$ and $p_1 \ldots p_n$ have at most $k$ mismatches
  - i.e., Hamming distance between $t_i \ldots t_{i+n-1}$ and $p \leq k$
APM: A Brute-Force Algorithm

**ApproximatePatternMatching**\((p, t, k)\)

1. \(n \leftarrow \text{length of pattern } p\)
2. \(m \leftarrow \text{length of text } t\)
3. for \(i \leftarrow 1 \text{ to } m - n + 1\)
4. \(\quad \text{dist} \leftarrow 0\)
5. \(\quad \text{for } j \leftarrow 1 \text{ to } n\)
6. \(\quad \quad \text{if } t_{i+j-1} \neq p_j\)
7. \(\quad \quad \quad \text{dist} \leftarrow \text{dist} + 1\)
8. \(\quad \text{if } \text{dist} \leq k\)
9. \(\quad \quad \text{output } i\)
APM: Running Time

• That algorithm runs in $O(nm)$.  

• Extend “Approximate Pattern Matching” to a more general “Query Matching Problem”:
  – Match *n*-length substring of the query (not the full pattern) to a substring in a text with at most $k$ mismatches
  – **Motivation**: we may seek similarities to some gene, but not know which parts of the gene to consider
Query Matching Problem

• **Goal**: Find all substrings of the query that approximately match the text

• **Input**: Query \( q = q_1 \ldots q_w \),
  text \( t = t_1 \ldots t_m \),
  \( n \) (length of matching substrings \( n \leq w \leq m \)),
  \( k \) (maximum number of mismatches)

• **Output**: All pairs of positions \((i, j)\) such that the
  \( n \)-letter substring of \( q \) starting at \( i \)
  approximately matches the
  \( n \)-letter substring of \( t \) starting at \( j \),
  with at most \( k \) mismatches
Approximate Pattern Matching vs Query Matching

(a) Approximate Pattern Matching

(b) Query Matching
Query Matching: Main Idea

- Approximately matching strings share some perfectly matching substrings.

- Instead of searching for approximately matching strings (difficult) search for perfectly matching substrings first (easy).
Filtration in Query Matching

• We want all \( n \)-matches between a query and a text with up to \( k \) mismatches

• “Filter” out positions that do not match between text and query

• \textbf{Potential match detection}: find all matches of \( \ell \)-tuples in query and text for some small \( \ell \)

• \textbf{Potential match verification}: Verify each potential match by extending it to the left and right, until \((k + 1)\) mismatches are found
Filtration: Match Detection

- If \(x_1 \ldots x_n\) and \(y_1 \ldots y_n\) match with at most \(k \ll n\) mismatches they must share \(l\)-mers that are perfect matches, with \(l = \lfloor n/(k + 1) \rfloor\)

- Break string of length \(n\) into \(k+1\) parts, each of length \(\lfloor n/(k + 1) \rfloor\)
  - \(k\) mismatches can affect at most \(k\) of these \(k+1\) parts
  - At least one of these \(k+1\) parts is perfectly matched
Filtration: Match Detection (cont’d)

• Suppose \( k = 3 \). We would then have \( l = n/(k+1) = n/4 \):

<table>
<thead>
<tr>
<th>1...l</th>
<th>l +1...2l</th>
<th>2l +1...3l</th>
<th>3l +1...n</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>k</td>
<td>k +1</td>
</tr>
</tbody>
</table>

• There are at most \( k \) mismatches in \( n \), so at the very least there must be one out of the \( k+1 \) \( l \)-tuples without a mismatch.
Filtration: Match Verification

- For each $l$-match we find, try to extend the match further to see if it is substantial.
### Filtration: Example

<table>
<thead>
<tr>
<th>$k$</th>
<th>$l$-tuple length</th>
<th>$k = 0$</th>
<th>$k = 1$</th>
<th>$k = 2$</th>
<th>$k = 3$</th>
<th>$k = 4$</th>
<th>$k = 5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>$n$</td>
<td>$n/2$</td>
<td>$n/3$</td>
<td>$n/4$</td>
<td>$n/5$</td>
<td>$n/6$</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Shorter **perfect matches** required

Performance **decreases**
Local alignment is too slow…

- Quadratic local alignment is too slow when looking for similarities between long strings (e.g. the entire GenBank database)
- Guaranteed to find the optimal local alignment
- Sets the standard for sensitivity
- **Basic Local Alignment Search Tool**
  - Altschul, S., Gish, W., Miller, W., Myers, E. & Lipman, D.J. Journal of Mol. Biol., 1990
- Search sequence databases for local alignments to a query
**BLAST**

- Great improvement in speed, with only a modest decrease in sensitivity
-Opts to minimizes search space instead of exploring entire search space between two sequences
- Finds short exact matches (“seeds”), explore locally around these “hits”
Similarity

- BLAST only continues its search as long as regions are sufficiently similar.
- Measuring the extent of similarity between two sequences:
  - Based on percent sequence identity
  - Based on conservation
Percent Sequence Identity

- The extent to which two nucleotide or amino acid sequences are invariant

70% identical

mismatch

indel
Conservation

- Amino acid changes that preserve the physico-chemical properties of the original residue
  - Polar to polar
    - aspartate $\rightarrow$ glutamate
  - Nonpolar to nonpolar
    - alanine $\rightarrow$ valine
  - Similarly behaving residues
    - leucine to isoleucine
- Nucleotide changes that preserve molecular shape
  - Transitions (A-G, C-T) are more similar than Transversions (A-C, A-T, C-G, G-T)
Assessing Sequence Similarity

- How good of a local alignment score can be expected from chance alone
- “Chance” relates to comparison of sequences that are generated randomly based upon a certain sequence model
- Sequence models may take into account:
  - nucleotide frequency
  - dinucleotide frequency (e.g. C+G content in mammals)
  - common repeats
  - etc.
BLAST: Segment Score

- BLAST uses scoring matrices ($\delta$) to improve on efficiency of match detection (we did this earlier for pairwise alignments)
  - Some proteins may have very different amino acid sequences, but are still similar (PAM, Blosum)
- For any two $l$-mers $x_1...x_l$ and $y_1...y_l$:
  - Segment pair: pair of $l$-mers, one from each sequence
  - Segment score: $\sum_{i=1}^{l} \delta(x_i, y_i)$
BLAST: Locally Maximal Segment Pairs

- A segment pair is **maximal** if it has the best score over all segment pairs.
- A segment pair is **locally maximal** if its score can’t be improved by extending or shortening.
- Statistically significant *locally maximal* segment pairs are of biological interest.
- BLAST finds all locally maximal segment pairs (MSPs) with scores above some threshold.
  - A significantly high threshold will filter out some statistically insignificant matches.
BLAST: Statistics

• Threshold: Altschul-Dembo-Karlin statistics
  – Identifies smallest segment score that is unlikely to happen by chance

• # matches with score > θ is approximately Poisson-distributed with mean:

\[ E(\theta) = Kmne^{-\lambda\theta} \]

\( K \) is a constant, \( m \) and \( n \) are the lengths of the two compared sequences, \( \lambda \) is a positive root of:

\[ \sum_{x,y \text{ in } A} (pxpye^{\lambda\delta(x,y)}) = 1 \]

where \( px \) and \( py \) are frequencies of amino acids \( x \) and \( y \), \( \delta \) is the scoring matrix, and \( A \) is the twenty letter amino acid alphabet
P-values

• The probability of finding exactly \( k \) MSPs with a score \( \geq \theta \) is given by:

\[
\frac{(E(\theta)^k \cdot e^{-E(\theta)})}{k!}
\]

• For \( k = 0 \), that chance is:

\[
e^{-E(\theta)}
\]

• Thus the probability of finding at least one MSP with a score \( \geq \theta \) is:

\[
p(MSP > 0) = 1 - e^{-E(\theta)}
\]
BLAST algorithm

- **Keyword search** of all substrings of length $w$ from the query of length $n$, in database of length $m$ with score above threshold
  - $w = 11$ for DNA queries, $w = 3$ for proteins

- **Local alignment extension** for each found keyword
  - Extend result until longest match above threshold is achieved

- Running time $O(nm)$
Original BLAST

- **Dictionary**
  - All words of length $w$

- **Alignment**
  - *Ungapped* extensions until score falls below some statistical threshold

- **Output**
  - All local alignments with score > threshold
Original BLAST: Example

- \( w = 4 \)
- Exact keyword match of GGTC
- Extend diagonals with mismatches until score is under some threshold (65%)
- Trim to until all mismatches are interior
- Output result:

\[
\begin{align*}
\text{GTAAGGTCC} \\
\text{|| ||} \\
\text{GTTAGGTCC}
\end{align*}
\]
Gapped BLAST : Example

- Original BLAST exact keyword search, then:
- Extend with gaps around ends of exact match until score < threshold
- Output result:
  GTAAGGTCCAGT
  || |||| | ||
  GTTAGGTC - AGT

From lectures by Serafim Batzoglou (Stanford)
Incarnations of BLAST

- blastn: Nucleotide-nucleotide
- blastp: Protein-protein
- blastx: Translated query vs. protein database
- tblastn: Protein query vs. translated database
- tblastx: Translated query vs. translated database (6 frames each)
Incarnations of BLAST (cont’d)

- PSI-BLAST
  - Find members of a protein family or build a custom position-specific score matrix
- Megablast:
  - Search longer sequences with fewer differences
- WU-BLAST: (Wash U BLAST)
  - Optimized, added features
Timeline

• 1970: Needleman-Wunsch global alignment algorithm
• 1981: Smith-Waterman local alignment algorithm
• 1985: FASTA
• 1990: BLAST (basic local alignment search tool)
• 2000s: BLAST has become too slow in “genome vs. genome” comparisons - new faster algorithms evolve!
  – Pattern Hunter
  – BLAT