Pattern Matching

Overview of pattern matching algorithms
- exact matching
- regular expressions
- alignment

**Exact Match**
- The first set of methods to consider are exact match (substring) searches
- We have a text $T$ of length $m$ and a pattern $P$ of length $n$
- Goal: find all occurrences of $P$ in $T$
- Example:
  $$P: \text{ATG}$$
  $$T: \text{ATATGATACGAATGAAC}$$
- Notation used in these slides (and in literature on pattern matching):
  - $S_i$: the $i$th symbol (character) in string $S$
  - $S_{i..j}$: the substring from $S_i$ to $S_j$

**Naïve Algorithm**
- The “obvious” way to search for $P$ in $T$:
  - align left ends of $P$, $T$
  - compare all letters in $P$ with corresponding letters in $T$
  - on mismatch move $P$ right one symbol

<table>
<thead>
<tr>
<th>$T$: ATATGATACGAATGAAC</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATG</td>
</tr>
<tr>
<td>(3 comparisons before shift)</td>
</tr>
</tbody>
</table>

**Better Exact Match**
- Improvement: slide $P$ to the right by an amount that depends on the symbols in $T$

<table>
<thead>
<tr>
<th>$T$: ATATGATACGAATGAAC</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATG</td>
</tr>
<tr>
<td>(mismatch at 3rd position)</td>
</tr>
</tbody>
</table>
- The first mismatch occurs where there is an $\Lambda$ in $T$
- Move $P$ to the right so the first $\Lambda$ in $P$ lines up with this $\Lambda$
Preprocessing

- The improved algorithm "discovers" information about \( T \) as it runs
- It is possible to gather this information before doing any comparisons
- Break the matching algorithm into two phases:
  - preprocessing
  - searching
- Preprocessing stores info about either \( P \) or \( T \)
- Searching uses the info to direct the comparisons

Preprocessing (cont’d)

- Algorithms that preprocess \( P \):
  - linear time exact match (example next slide)
- Algorithms that preprocess \( T \):
  - web crawlers -- \( T \) is the internet, \( P \) is user’s search string
  - suffix tree algorithms (following topic)
  - BLAST and other homology search programs -- \( T \) is the sequence database

Linear Time Exact Match

- The naïve algorithm requires \( O(n \times m) \) comparisons
- It is possible to achieve
  - \( O(n + m) \) expected behavior
  - Sublinear (< \( O(n) \) ) average behavior
    - (when \( P \) is long and \( P, T \) have few symbols in common)
- The linear algorithms preprocess the pattern \( P \):
  - \( O(n) \) steps for preprocessing
  - \( O(m) \) (or better) comparisons in searching

Suffix Trees

- A data structure known as a suffix tree can implement very efficient string searches
- Build a tree for text \( T \) in \( O(m) \) steps
- Search the tree for a pattern \( P \) in \( O(n) \) comparisons
- Cool facts about suffix trees:
  - The tree can be built without knowledge of \( P \)
  - The search for \( P \) is independent of the size of \( T \)
  - One search returns all occurrences of \( P \)
Suffix Trees (cont’d)

- In a suffix tree for a string S of length m:
  - the interior edges are labeled with substrings of S
  - each edge leaving a node has a different first character
  - there are m leaves, numbered 1..m
  - the concatenation of labels on the edges in the path from the root to node i is S_{i..m} (i.e. the suffix of S that starts at position i)

Suffix Trees (cont’d)

- Suffix tree for GATGAC

Exact Match with Suffix Tree

- To find a pattern P in a string T start traversing from the root
  - i = 0
  - while i < n:
    - traverse edge labeled S where S_{1..k} = P_{1..k}
    - i += k
  - Example: find “ATG”

Exact Match in Bioinformatics

- Situation where efficient exact string match is important: Sequence tagged site (STS)
  - DNA sequence of 200-300 bases
  - ends (20-30 bases) are unique
  - EST (expressed sequence tag) is an STS from a coding region

- Goal: given newly sequenced DNA, can we identify which part of the (human) genome it’s from?
Exact Match in Bioinformatics (cont’d)

- “Shotgun sequencing”
  - Clone large segment of DNA
  - Break clones at random places
  - Sequence each piece
  - Reassemble
- Use exact match to find pairs of fragments with overlapping ends

![Exact Match in Bioinformatics Diagram]

[graphics from article by Myers, TIGR]

Regular Expressions

- A regular expression is a pattern that may include metasymbols allowing the pattern to match more than one string
- Example:
  
  \( \text{TCA, TCT, TCC, TCG} \)
- The metasymbols \([\) and \(]\) mean “match any one character between the brackets”

![Regular Expressions Diagram]

Algorithms for Regular Expressions

- A common technique for implementing RE pattern matching is to “compile” the RE
- Any RE can be transformed into a graph
- Example: the graph for \( \text{GT. *AG} \) (a pattern that describes most introns in eukaryotes)
Regular Expressions in Bioinformatics

- Regular expressions can be used to describe patterns in short amino acid strings
  - Domain: building block of protein structure (sheets, helices)
  - Motif: short amino acid structure that defines protein features
- PROSITE database [1992]: collection of protein motifs described by patterns

Motif Example: Zinc Finger

- Transcription factors are proteins used in DNA transcription
- A common motif is the C2H2 “zinc finger”
- Two Cs and two Hs bind to a Zinc atom
- Intervening amino acids form “finger” that projects into the groove in the DNA double helix

Zinc Finger (cont’d)

- A regular expression that describes this motif is
  \[ C.\{4\}C.\{8,12\}H.\{3\}[MLIV]H \]
- The above expression is from the gref program of the SEALS system [Walker, NCBI]
  ```
  % gref expr human.fa
  ```
  will print sequences in a Fasta file that match expr

Alignment

- A string alignment algorithm is another way of comparing two strings that are not 100% identical
- Examples:
  - similarity search: given a database of strings \( T \) find members that are “similar” to a string \( S \)
  - evolution: what is the degree of similarity between strings \( S_1 \) and \( S_2 \)? Is the pair \((S_1, S_2)\) more similar than \((S_1, S_3)\)?
Alignment (cont’d)
- Rest of this lecture:
  - definitions
  - quantifying similarity
  - basic (hand) alignment method
- Next lecture:
  - dynamic programming algorithm
- Future: applications based on alignment
  - similarity search (e.g. BLAST)
  - phylogeny inference

Definitions
- An alignment of two strings is a mapping that associates each character of one string with
  a) a character in the other string, or
  b) no character (a “gap”)
- Example: alignment of two 7-character strings
  \[
  \text{AA-TACCG} \\
  \text{AAACA-CG}
  \]
- In general the two strings do not have to be the same length before alignment

Definitions (cont’d)
- A **global alignment** maps each character of each string
  - the result is two equal-length strings (but gaps may occur on either end)
- A **local alignment** maps substrings of each string
  - useful for comparing motifs or domains
  - best for less similar strings
- A **multiple alignment** is a simultaneous (local or global) mapping between sets of three or more strings
  \[
  \text{AA–TACCG} \\
  \text{AAACA–CG} \\
  \text{AAACACCG}
  \]

Edit Distance
- A simple method for measuring similarity uses an *edit distance*
- Based on four simple string operations:
  - match (M)
  - replace (R)
  - insert (I)
  - delete (D)
- An *edit transcript* is a sequence of edit operations
Edit Distance (cont’d)

- Example (Gusfield): transform vintner into writers:
  - Transcript: RIMMMDMMI
- An alignment can be constructed from an edit transcript -- for this example:
  - S1  v-intner-
  - S2  wri-t-ers

Alignment as Optimization

- We can assign a cost to each operation
  - e.g. M = 0, R = 1, D = I = 2
- The edit distance between two strings is the sum of the costs of the edit operations in the lowest cost transcript that transforms one string into the other
- The concept of edit distance allows us to specify the goal of an alignment algorithm:
  - align strings S1 and S2 by finding a transcript that transforms S1 into S2 with a minimum edit distance
- Looking ahead: dynamic programming is an optimization method well suited to this application....

Cost Functions

- Edit distances can be adapted to model more realistic situations
- Examples:
  - if insertions and deletions are less likely than mismatches (e.g. point mutations vs cross-overs) assign a different weight to I/D vs R
  - starting a gap may be more expensive than extending a gap
  - individual substitutions may have different weights (e.g. in proteins S→T more likely than S→Y)

Dot Matrix Method

- To transform S1 (n chars) into S2 (m chars) using a simple edit distance with I = D = R = 1 and M = 0:
  - draw a n x m matrix
  - label columns with characters of S1, rows with characters from S2
  - enter a dot in each cell where the row and column labels are the same
- Example: matrix for aligning S1 = ATTCA with S2 = ATGTA

"A" "T" "C" "A"
"T"    "T"    "T"
"G"    "G"    "G"
"T"    "T"    "T"
"A"    "A"    "A"
Dot Matrix (cont’d)

- Find a **minimum weight path** from the upper left to the lower right
- Each move on the path corresponds to an edit operation
  - Diagonal: M or R
  - Vertical: I
  - Horizontal: D
- For the path shown at right: MMIMDM
- Note there may be several equal-weight paths