Master Course
Sequence Alignment

Lecture 9

Motif searches
Pattern matching

• Functional genomics: finding out the function of all genes (and other parts) in a genome
• Ability to recognise protein function paramount
• Database searching is crucial strategy
  – trypsin has *catalytic triad* (His, Asp, Ser). How to recognize this?
  – (local) alignments not always suitable
    • short patterns, too many ‘don’t cares’, etc.
pattern matching

This lecture:

• Regular expressions
• Hidden Markov models (brief)
Rationale for regular expressions

- “I want to see all sequences that ...
  - ... contain a C”
  - ... contain a C or an F”
  - ... contain a C and an F”
  - ... contain a C immediately followed by an F”
  - ... contain a C later followed by an F”
  - ... begin with a C”
  - ... do not contain a C”
  - ... contain at least three Cs”
  - ... contain exactly three Cs”
  - ... has a C at the seventh position”
  - ... either contain a C, an E, and an F in any order except CFE, unless there are also at most three Ps, or there is a ....
regular expressions

- **alphabet**: set of symbols
  - \{A, C, T, G\}

- **string**: sequence of symbols from alphabet
  - AACTG, CATG, GGA, ACFT, ε

- **regex**: formal method to define (sub)set of strings
  - \[^C\].AG?T*
  - used for pattern matching
    - check if database sequence ∈ regex
construction of a regex

- regex contains:
  - symbols from alphabet
    - \( C \rightarrow \{ C \} \)
  - operators
    - operations on regex(es) yield new regex
    - concatenation, union, repetition, ...
basic operators

\[ r_1 r_2 \]  \hspace{1cm} \text{concatenation}

\[
\begin{align*}
\text{AC} & \rightarrow \{ \text{AC} \} \\
\text{AAC} & \rightarrow \{ \text{AAC} \}
\end{align*}
\]

\[ [s_1 s_2 \ldots s_n] \]  \hspace{1cm} \text{union (of symbols)}

\[
\begin{align*}
[\text{ACG}] & \rightarrow \{ \text{A, C, G} \} \\
[\text{AC}]\text{G} & \rightarrow \{ \text{AG, CG} \}
\end{align*}
\]

\[ r_1 | r_2 \]  \hspace{1cm} \text{union (of regexes)}

\[
\begin{align*}
\text{A}|\text{CC} & \rightarrow \{ \text{A, CC} \} \\
[\text{AC}]|\text{AC} & \rightarrow \{ \text{A, C, AC} \}
\end{align*}
\]

\[ r^{+} \]  \hspace{1cm} \text{repeat once or more}

\[
\begin{align*}
\text{C}^{+} & \rightarrow \{ \text{C, CC, CCC, CCCC, ...} \} \\
\text{A}[\text{AC}]^{+} & \rightarrow \{ \text{AA, AC, AAA, AAC, ACA, ACC, AAAA, AAAC, ...} \}
\end{align*}
\]
derived operators

\[ r? \] optional
\[ C? \rightarrow \{ \varepsilon, C \} \]
\[ AC?G \rightarrow \{ AG, ACG \} \]

\[ r^* \] repeat zero or more times
\[ C^* \rightarrow \{ \varepsilon, C, CC, CCC, CCCC, ... \} \]
\[ A^*C \rightarrow \{ C, AC, AAC, AAAC, ... \} \]
\[ [AC]^* \rightarrow \{ \varepsilon, A, C, AA, AC, CA, CC, AAA, AAC, ACA, ACC, ... \} \]

\[ r^{n-m} \] repeat \( n - m \) times
\[ C^4 \rightarrow \{ CCCC \} \]
\[ C^{2-4} \rightarrow \{ CC, CCC, CCCC \} \]
\[ C^{-3} \rightarrow \{ \varepsilon, C, CC, CCC \} \]
\[ C^{3-} \rightarrow \{ CCC, CCCCC, CCCCCC, ... \} \]
miscellaneous

.  
any symbol
  .  → { A, C, G, T }
A.C  → { AAC, ACC, AGC, ATC }
.?  → { ε, A, C, G, T }
.*  → { ε, A, C, G, T, AA, AC, AG, AT, CA, CC, CG, CT, GA, ... }

[^s₁s₂ ... sₙ]  
exclude symbols
[^A]  → { C, G, T }
[^AC]  → { G, T }

(r)  
grouping
(AC)?  → { ε, AC }
AC?  → { A, AC }
(AC)*  → { ε, AC, ACAC, ACACAC, ACACACAC, ... }
AC*  → { A, AC, ACC, ACCC, ... }
limitations

- regex cannot remember indeterminate counts !!!

- “I want to see all sequences with ... 
  - six Cs followed by six Ts”
    - $C^6T^6$
  - any number of Cs followed by any number of Ts”
    - $C^*T^*$
  - Cs followed by an equal number of Ts”
    - $C^nT^n$
    - $(CT|CCTT|CCCTTT|C^4T^4| ... )$?

- use (context-free) grammar
regexes in pattern matching

- pattern described by regex
- check if sequence $\subseteq$ regex
- matching done very efficiently
  - $O(n)$
  - using state machine
state machines

- compile regex to state machine
- match sequence with regex

AC*T|GGC
Example from BLAST: Determining Query Words

• Given:
  – query sequence: QLNFSAGW
  – word length \( w = 3 \)
  – word score threshold \( T = 8 \)

• Step 1: determine all words of length \( w \) in query sequence
  QLN LNF NFS FSA SAG AGW
Example from BLAST: Determining Query Words

• Step 2: determine all words that score at least $T$ when compared to a word in the query sequence:

<table>
<thead>
<tr>
<th>words from sequence</th>
<th>query words w/ $T=8$</th>
</tr>
</thead>
<tbody>
<tr>
<td>QLN</td>
<td>QLN=11, QMD=9, HLN=8, ZLN=9,…</td>
</tr>
<tr>
<td>LNF</td>
<td>LNF=9, LBF=9, LBY=8, FNW=8,…</td>
</tr>
<tr>
<td>NFS</td>
<td>NFS=12, AFS=8, NYS=8, DFT=10,…</td>
</tr>
<tr>
<td>…</td>
<td>none</td>
</tr>
<tr>
<td>SAG</td>
<td>none</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>
Example from BLAST: Scanning the Database

• search database for all occurrences of query words

• approach:
  – build a DFA (deterministic finite-state automaton) that recognizes all query words
  – run DB sequences through DFA
  – remember hits
**Example from BLAST:** Scanning the Database

- Consider a DFA to recognize the query words: QL, QM, ZL

- All that a DFA does is read strings, and output "accept" or "reject."

- Use Mealy paradigm (accept on transitions) to save space and time

**Moore paradigm:** the alphabet is (a, b), the states are q0, q1, and q2, the start state is q0 (denoted by the arrow coming from nowhere), the only accepting state is q2 (denoted by the double ring around the state), and the transitions are the arrows. The machine works as follows. Given an input string, we start at the start state, and read in each character one at a time, jumping from state to state as directed by the transitions. When we run out of input, we check to see if we are in an accept state. If we are, then we accept. If not, we reject.

**Moore paradigm:** accept/reject states

**Mealy paradigm:** accept/reject transitions
Example from BLAST: a DFA to recognize query words: QL, QM, ZL

Mealy paradigm

Accept on red transitions (Mealy paradigm)
other uses

• many programs use regular expressions
  – command-line interpreter
    • del *.*
  – editor
    • search
    • replace
  – compilers
  – perl, grep, sed, awk

😊 many different syntaxes
**local vs. global matching**

- **global:** regex describes entire string to be matched
  - ACCCCTG $\approx$ C$^3$-

- **local:** regex describes substring to be matched
  - ACCCCTG $\gamma$ C$^3$-
  - ^ matches start-of-string
    - ^CG: match everything starting with CG
    - ^[^CG]: match everything not starting with C or G
  - $ matches end-of-string
    - AC$: match everything ending with AC
Regular expressions

Alignment

ADLGAVFALCDRYFQ
SDVGPRSCFCERFYQ
ADLGRTQNRCDRYYQ
ADIGQPHSLCERYFQ

For short sequence stretches, regular expressions are often more suitable to describe the information than alignments (or profiles)

Regular expression

\[ [\text{AS}] - \text{D} - [\text{IVL}] - \text{G} - x4 - \{\text{PG}\} - \text{C} - [\text{DE}] - \text{R} - [\text{FY}] 2 - \text{Q} \]

\{\text{PG}\} = \text{not (P or G)}
## Regular expressions

<table>
<thead>
<tr>
<th>Regular expression</th>
<th>No. of exact matches in DB</th>
</tr>
</thead>
<tbody>
<tr>
<td>D-A-V-I-D</td>
<td>71</td>
</tr>
<tr>
<td>D-A-V-I-[DENQ]</td>
<td>252</td>
</tr>
<tr>
<td>[DENQ]-A-V-I-[DENQ]</td>
<td>925</td>
</tr>
<tr>
<td>[DENQ]-A-[VLI]-I-[DENQ]</td>
<td>2739</td>
</tr>
<tr>
<td>[DENQ]-[AG]-[VLI]2-[DENQ]</td>
<td>51506</td>
</tr>
<tr>
<td>D-A-V-E</td>
<td>1088</td>
</tr>
</tbody>
</table>
Motif-based function prediction

- Prediction of protein functions based on identified sequence motifs
- **PROSITE** contains patterns specific for more than a thousand protein families.

**ScanPROSITE** -- it allows to scan a protein sequence for occurrence of patterns and profiles stored in PROSITE.
Example of HMM repository:
The PFAM Database

Pfam is a large collection of multiple sequence alignments and hidden Markov models covering many common protein domains and families. For each family in Pfam you can:

• Look at multiple alignments
• View protein domain architectures
• Examine species distribution
• Follow links to other databases
• View known protein structures
• Search with Hidden Markov Model (HMM) for each alignment
The PFAM Database

Pfam is a database of two parts, the first is the curated part of Pfam containing about 9000 protein families (Pfam-A). Pfam-A comprises manually crafted multiple alignments and profile-HMMs.

To give Pfam a more comprehensive coverage of known proteins we automatically generate a supplement called Pfam-B. This contains a large number of small families taken from the PRODOM database that do not overlap with Pfam-A. Although of lower quality Pfam-B families can be useful when no Pfam-A families are found.
The PFAM Database

Sequence coverage Pfam-A: 74% (Yellow)
Sequence coverage Pfam-B: 13% (Blue)
Other (Grey)

74% of proteins have at least one match with Pfam.

Version 21.0 - November 2006: Pfam-A contains 8957 families
Clan pages in Pfam. (A) A screen shot of a clan summary page, containing the description, annotation and membership of the clan. From this page, the user can view the family relationship diagram (B). Each family in the clan is represented by a blue box and its relationship to other families is represented by solid lines (significant profile–profile comparison score) or dashed lines (non-significant profile-profile comparison score). Beside each line, the profile–profile comparison E-value score is presented. This score is also linked to a visualization of the profile–profile comparison alignment (C). The clan summary page also provides a link to the clan alignment (D) (for more details see text). The clan alignment is a multiple sequence alignment of all of the clan members seed alignments (each set of seed sequences are separated by the alternate background shading). The alignments are coloured using Jalview.
HMM-based homology searching

Transition probabilities and Emission probabilities

Gapped HMMs also have insertion and deletion states
Profile HMM: m=match state, I=insert state, d=delete state; go from left to right. I and m states output amino acids; d states are ‘silent’.

Transition probabilities and Emission probabilities
A hidden Markov model accompanying a PFAM alignment

HMMER2.0 [2.2g]
NAME cytochrome_b_N
ACC PF00033
DESC Cytochrome b(N-terminal)/b6/petB
LENG 222 ALPH Amino RF no CS no MAP yes COM hmmbuild -F HMM_ls.ann SEED.ann COM hmmcalibrate --seed 0
HMM_ls.ann NSEQ 8 DATE Thu Dec 12 02:48:53 2002
CKSUM 8731
GA -41.9 -41.9 TC -41.9 -41.9 NC -42.4 -42.4 XT -8455 -4 -1000 -1000 -8455 -4 -8455 -4 NULT -4 -8455 NULE 595 -1558 85 338 -294 453 -1158 197 249 902 -1085 -142 -21 -313 45 531 201 384 -1998 -644 EVD -170.913223 0.138730
HMM A C D E F G H I K L M N P Q R S T V W Y
m->m m->i m->d i->m i->i d->m d->d b->m m->e -300 * -2414
--564 -3141 -2265 -289 -2463 -701 -1378 -1300 –8788

HMMs are good for profile searches… but optimising the many parameters when using HMMs to do alignments from scratch is a problem.