Lecture 10

*Ab initio* gene finding
Uses of probabilistic sequence models/HMMs

- Segmentation
- Multiple alignment using profile HMMs
- Prediction of sequence function (gene family models)
- ** Gene finding **
Review

• **Gene**
  – A sequence of nucleotides that are translated into proteins

• **Gene prediction**
  – Given the model of a gene above, determine the beginning and end positions of all genes in a genome.
Central dogma of molecular biology

• Information is stored in DNA

• Genome is processed into messenger RNA molecules (transcription)

• RNA molecules are processed to form proteins (translation)
General facts about the genetic code

• There is inherent redundancy; 64 possibilities and 20 amino acids

• Most of the “flexibility” is in the third position
  – “wobble” position

• Deletions that are not multiples of 3 change the resulting amino acid
DNA → transcription → mRNA → translation → Protein

CCTGAGCCAACTATTGATGAA → CCUGAGCCAACUAUUGAUGAA → PEPTIDE
Source and interesting link:
http://www.bioscience.org/atlases/genecode/genecode.htm
Facts about gene finding

• Prediction is usually easier in prokaryotes

• Small fraction of most genomes is genes
  – 3% of the human genome

• Many false signals

• Long introns and small exons
  – Difficult for mathematical models
In more detail (color ~state)

**Promoter**

**5' UTR**

**Transcription start**

**Start codon**

**Donor site**

**Exon (Left)**

**Intron (Removed)**

**Stop codon**

**Acceptor site**

**Poly-A site**

**3' UTR**
Gene prediction is usually broken into three types:

- **Ab initio**: making predictions based on some statistical model (GENSCAN, FGENE)

- Knowledge-based: making predictions based on known genes (tBLASTx)

- Comparative: making predictions based on a related genome (TwinScan)
Figure 4: Multi-Species Comparative Analysis. (This picture is from Sanja Rogenic’s lecture slides, “Computational Gene Finding”)
Observed sequence, hidden path and Viterbi path

<table>
<thead>
<tr>
<th>Rolls</th>
<th>Die</th>
<th>Viterbi</th>
</tr>
</thead>
<tbody>
<tr>
<td>315116246464642453113216311641521336251445436316566265666666</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FFFFFF</td>
<td>FFFFFF</td>
<td>FFFFFF</td>
</tr>
<tr>
<td>65116645313265124563666463163663162326455236266666625151631</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LLLLLL</td>
<td>LLLLLL</td>
<td>LLLLLL</td>
</tr>
<tr>
<td>22255541666566563564324364131513465146353411126414626253356</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FFFFFF</td>
<td>FFFFFF</td>
<td>FFFFFF</td>
</tr>
<tr>
<td>36616366646623253441366166116325624622552652652664353535336</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LLLLLL</td>
<td>LLLLLL</td>
<td>LLLLLL</td>
</tr>
<tr>
<td>23312162536441443233516324363365562666662632666612355245242</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FFFFFF</td>
<td>FFFFFF</td>
<td>FFFFFF</td>
</tr>
</tbody>
</table>

Figure 3.5 The numbers show 300 rolls of a die as described in the example. Below is shown which die was actually used for that roll (F for fair and L for loaded). Under that the prediction by the Viterbi algorithm is shown.

From Durbin
The occasionally dishonest casino: Posterior decoding

**Figure 3.6** The posterior probability of being in the state corresponding to the fair die in the casino example. The $x$ axis shows the number of the roll. The shaded areas show when the roll was generated by the loaded die.
Example from Durbin

Note transition probabilities are different from real ones. Partly a result of local minima, but it’s never possible to estimate parameters exactly.
Figure 1. HMM architecture for a parser for E.coli DNA with a simple intergenic model. The central state (shaded circle), generates no nucleotides and is used to connect all the models. The 61 triplet or codon models above the central state all have identical structures, shown in detail for the codon AAC. Squares represent main states; diamonds denote a state where a nucleotide can be inserted between consecutive codon nucleotides whereas circles generate no nucleotide and can be used to delete one of the three nucleotides. The thickness of the arrows indicate the fraction of sequences making the given transition. The insert state in the middle of the intergenic model (diamond) produces random sequences from a base distribution estimated from the actual distribution of bases in the intergenic regions of the training set. The four bases have almost the same frequency.

Krogh, Mian and Haussler (1994)
Burge and Karlin (1997)
Open reading frames

• Generally defined as regions in genes between a start (ATG) and stop (eg. TGA) codon.

• Size is a multiple of 3

• Six possibilities given any DNA sequence
  – 0 offset, + strand; 1 offset, + strand, 2 offset, + strand
  – 0 offset, - strand; 1 offset, - strand, 2 offset, - strand
Examples from the text

• *M. genitalium* (example 2.6)
  – 402 bp is a significant value

• *H. influenzae* (2.7)
  – 573 bp is significant
  – Larger genome than *M. genitalium*, so this makes sense.
Not all codons are equal

### Codon Usage in *E. coli* Genes

<table>
<thead>
<tr>
<th>Codon</th>
<th>Amino Acid</th>
<th>%</th>
<th>Ratio</th>
<th>Codon</th>
<th>Amino Acid</th>
<th>%</th>
<th>Ratio</th>
<th>Codon</th>
<th>Amino Acid</th>
<th>%</th>
<th>Ratio</th>
<th>Codon</th>
<th>Amino Acid</th>
<th>%</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>UUU</td>
<td>Phe (F)</td>
<td>1.9</td>
<td>0.51</td>
<td>UCU</td>
<td>Ser (S)</td>
<td>1.1</td>
<td>0.19</td>
<td>UAU</td>
<td>Tyr (Y)</td>
<td>1.6</td>
<td>0.53</td>
<td>UGU</td>
<td>Cys (C)</td>
<td>0.4</td>
<td>0.43</td>
</tr>
<tr>
<td>UUC</td>
<td>Phe (F)</td>
<td>1.8</td>
<td>0.49</td>
<td>UCC</td>
<td>Ser (S)</td>
<td>1.0</td>
<td>0.17</td>
<td>UAC</td>
<td>Tyr (Y)</td>
<td>1.4</td>
<td>0.47</td>
<td>UGC</td>
<td>Cys (C)</td>
<td>0.6</td>
<td>0.57</td>
</tr>
<tr>
<td>UUA</td>
<td>Leu (L)</td>
<td>1.0</td>
<td>0.11</td>
<td>UCA</td>
<td>Ser (S)</td>
<td>0.7</td>
<td>0.12</td>
<td>UAA</td>
<td>STOP</td>
<td>0.2</td>
<td>0.62</td>
<td>UGA</td>
<td>STOP</td>
<td>0.1</td>
<td>0.30</td>
</tr>
<tr>
<td>UUG</td>
<td>Leu (L)</td>
<td>1.1</td>
<td>0.11</td>
<td>UCG</td>
<td>Ser (S)</td>
<td>0.8</td>
<td>0.13</td>
<td>UAG</td>
<td>STOP</td>
<td>0.03</td>
<td>0.09</td>
<td>UGG</td>
<td>Trp (W)</td>
<td>1.4</td>
<td>1.00</td>
</tr>
<tr>
<td>C</td>
<td>Leu (L)</td>
<td>1.0</td>
<td>0.10</td>
<td>CCU</td>
<td>Pro (P)</td>
<td>0.7</td>
<td>0.16</td>
<td>CAU</td>
<td>His (H)</td>
<td>1.2</td>
<td>0.52</td>
<td>CGU</td>
<td>Arg (R)</td>
<td>2.4</td>
<td>0.42</td>
</tr>
<tr>
<td>CUC</td>
<td>Leu (L)</td>
<td>0.9</td>
<td>0.10</td>
<td>CCC</td>
<td>Pro (P)</td>
<td>0.4</td>
<td>0.10</td>
<td>CAC</td>
<td>His (H)</td>
<td>1.1</td>
<td>0.48</td>
<td>CGC</td>
<td>Arg (R)</td>
<td>2.2</td>
<td>0.37</td>
</tr>
<tr>
<td>CUA</td>
<td>Leu (L)</td>
<td>0.3</td>
<td>0.03</td>
<td>CCA</td>
<td>Pro (P)</td>
<td>0.8</td>
<td>0.20</td>
<td>CAA</td>
<td>Gin (Q)</td>
<td>1.3</td>
<td>0.31</td>
<td>CGA</td>
<td>Arg (R)</td>
<td>0.3</td>
<td>0.05</td>
</tr>
<tr>
<td>CUG</td>
<td>Leu (L)</td>
<td>5.2</td>
<td>0.55</td>
<td>CGC</td>
<td>Pro (P)</td>
<td>2.4</td>
<td>0.55</td>
<td>CAG</td>
<td>Gin (Q)</td>
<td>2.9</td>
<td>0.69</td>
<td>CGG</td>
<td>Arg (R)</td>
<td>0.5</td>
<td>0.08</td>
</tr>
<tr>
<td>A</td>
<td>Ile (I)</td>
<td>2.7</td>
<td>0.47</td>
<td>ACU</td>
<td>Thr (T)</td>
<td>1.2</td>
<td>0.21</td>
<td>AUA</td>
<td>Asn (N)</td>
<td>1.6</td>
<td>0.39</td>
<td>AGU</td>
<td>Ser (S)</td>
<td>0.7</td>
<td>0.13</td>
</tr>
<tr>
<td>AUC</td>
<td>Ile (I)</td>
<td>2.7</td>
<td>0.46</td>
<td>ACC</td>
<td>Thr (T)</td>
<td>2.4</td>
<td>0.43</td>
<td>AAC</td>
<td>Asn (N)</td>
<td>2.6</td>
<td>0.61</td>
<td>AGC</td>
<td>Ser (S)</td>
<td>1.5</td>
<td>0.27</td>
</tr>
<tr>
<td>AUA</td>
<td>Ile (I)</td>
<td>0.4</td>
<td>0.07</td>
<td>ACA</td>
<td>Thr (T)</td>
<td>0.1</td>
<td>0.30</td>
<td>AAA</td>
<td>Lys (K)</td>
<td>3.8</td>
<td>0.76</td>
<td>AGA</td>
<td>Arg (R)</td>
<td>0.2</td>
<td>0.04</td>
</tr>
<tr>
<td>AUG</td>
<td>Met (M)</td>
<td>2.6</td>
<td>1.00</td>
<td>AAC</td>
<td>Thr (T)</td>
<td>1.3</td>
<td>0.23</td>
<td>AAG</td>
<td>Lys (K)</td>
<td>1.2</td>
<td>0.24</td>
<td>AGG</td>
<td>Arg (R)</td>
<td>0.2</td>
<td>0.03</td>
</tr>
<tr>
<td>G</td>
<td>Val (V)</td>
<td>2.0</td>
<td>0.29</td>
<td>GCU</td>
<td>Ala (A)</td>
<td>1.8</td>
<td>0.19</td>
<td>GAU</td>
<td>Asp (D)</td>
<td>3.3</td>
<td>0.59</td>
<td>GGU</td>
<td>Gly (G)</td>
<td>2.8</td>
<td>0.38</td>
</tr>
<tr>
<td>GUC</td>
<td>Val (V)</td>
<td>1.4</td>
<td>0.20</td>
<td>GCC</td>
<td>Ala (A)</td>
<td>2.3</td>
<td>0.25</td>
<td>GAC</td>
<td>Asp (D)</td>
<td>2.3</td>
<td>0.41</td>
<td>GGC</td>
<td>Gly (G)</td>
<td>3.0</td>
<td>0.40</td>
</tr>
<tr>
<td>GUA</td>
<td>Val (V)</td>
<td>1.2</td>
<td>0.17</td>
<td>GCA</td>
<td>Ala (A)</td>
<td>2.1</td>
<td>0.22</td>
<td>GAA</td>
<td>Glu (E)</td>
<td>4.4</td>
<td>0.70</td>
<td>GGA</td>
<td>Gly (G)</td>
<td>0.7</td>
<td>0.09</td>
</tr>
<tr>
<td>GUG</td>
<td>Val (V)</td>
<td>2.4</td>
<td>0.34</td>
<td>GCG</td>
<td>Ala (A)</td>
<td>3.2</td>
<td>0.34</td>
<td>GAG</td>
<td>Glu (E)</td>
<td>1.9</td>
<td>0.30</td>
<td>GGG</td>
<td>Gly (G)</td>
<td>0.9</td>
<td>0.13</td>
</tr>
</tbody>
</table>


2. The letter in parenthesis represents the one-letter code for the amino acid.

3. % represents the average frequency this codon is used per 100 codons.

4. Ratio represents the abundance of that codon relative to all of the codons for that particular amino acid.

Simple example

• Lets only consider two states: coding and non-coding (+ or -)

• Further, we will consider only four output states
  – ACGT

• At most we will have 8 states:
Long ORFs

• At random, we’d expect a stop codon every 64 nucleotides.

• Many bacteria genes are much longer than this.

• These can be used to train a statistical model.
Significance

• An important question often is “Are these interesting?”

• Many bioinformatics tools compute a \( \rho \)-value, or the probability of observing something in a random collection.
  – Example: e-value in BLAST
Randomization test

- There are two ways to determine significant ORF lengths:
  - Permutation
    - Shuffle the original sequence in a simple or more complicated way
  - Bootstrapping
    - Generate random sequences with the same statistical properties
    - Uses Markov chain models as in Hw #1
But wait!

- Finding ORFs alone isn’t sufficient. \textit{E. coli} has up to 6500 ORFs but only 1100 “real” genes

- Complicated by the fact ORFs can overlap on different strands and be correct (figure in class)
Homework to the rescue (or at least part of the way there)

• We know codons are triplets.

• We also know from last class codons are not equal.

• A simple gene finder is no different from your Markov assignment except now we’ll need to compute two probabilities
Simple gene finding

• Score every ORF using all seven models

• Normalize the scores such that they represent the probability of coding

• Choose the highest
Durbin: First order Markov
(p74)

Figure 3.11 Histograms of the log-odds per nucleotide for all NORFs (grey) and genes (black line) according to a first order Markov chain. Because of the large number of NORFs, the histogram bin size is five times smaller for the NORFs.
Take home

• Interestingly, there is some signal in the previous figure:
  – Average log-odds for genes: 0.018
  – Average log-odds for non-genes: 0.009

• The variance is huge, though, so it is hard to tell which is which in general.
Selecting the proper Markov order

• Higher order models remember more “history,” which helps predictions

• Examples (from Colin Dewey):
  – “… you ___”
  – “… can you ___”
  – “… say can you ___”
  – “… oh say can you ___”
One fix is to use a third order model
Less simple gene finder

• Build a Markov model of order 3 on very long open reading frames (say 1000bp)

• Build a Markov model on everything else.

• A log-likelihood (or log-odds) ratio can tell us how what is more like a gene
  – Uses a Chi^2 distribution
Interpolated Markov Models (IMMs) overcome the training problem by generating models of variable order.

Bias is put towards higher models if and only if there is enough training data.

Achieved via a linear combination of probabilities based on varied lengths.
Simple linear interpolation

\[ P_{\text{IMM}}(x_i \mid x_{i-n}, \ldots, x_{i-1}) = \lambda_0 P(x_i) + \lambda_1 P(x_i \mid x_{i-1}) + \cdots + \lambda_n P(x_i \mid x_{i-n}, \ldots, x_{i-1}) \]

- where \( \sum_i \lambda_i = 1 \)
GLIMMER

• Addressed the fundamental training problem of markov models

• As mentioned before, we want the highest order model possible

• However, a $k^{th}$ order model requires $4^{k+1}$ probabilities to be estimated
  – Impractical for small genomes
Only 1 of the 37 genes missed by GLIMMER was found by the 5th order model.

On the other hand, it found 107 more true genes.

Table 1. Comparison of the IMM model used in GLIMMER to a 5th-order Markov model

<table>
<thead>
<tr>
<th>Model</th>
<th>Genes found</th>
<th>Genes missed</th>
<th>Additional genes</th>
</tr>
</thead>
<tbody>
<tr>
<td>GLIMMER IMM</td>
<td>1680 (97.8%)</td>
<td>37</td>
<td>209</td>
</tr>
<tr>
<td>5th-Order Markov</td>
<td>1574 (91.7%)</td>
<td>143</td>
<td>104</td>
</tr>
</tbody>
</table>

The first column indicates how many of the 1717 annotated genes in *H. influenzae* were found by each algorithm. The ‘additional genes’ column shows how many extra genes, not included in the 1717 annotated entries, were called genes by each method.