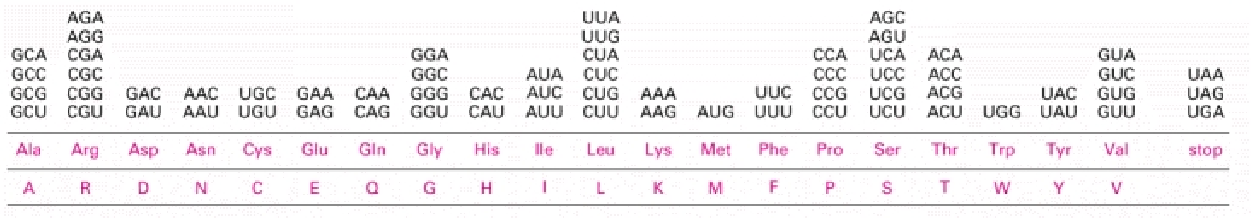
Genomics and Bioinformatics: Week 3

4 October 2011

In this chapter, we will cover several approaches to locate protein-coding genes in a genome sequence.

1 Finding ORFs



	AGA									UUA					AGC					
	AGG									UUG					AGU					
GCA	CGA						GGA			CUA				CCA	UCA	ACA			GUA	
GCC	CGC						GGC		AUA	CUC				CCC	UCC	ACC			GUC	UAA
GCG	CGG	GAC	AAC	UGC	GAA	CAA	GGG	CAC	AUC	CUG	AAA		UUC	CCG	UCG	ACG		UAC	GUG	UAG
GCU	CGU	GAU	AAU	UGU	GAG	CAG	GGU	CAU	AUU	CUU	AAG	AUG	UUU	CCU	UCU	ACU	UGG	UAU	GUU	UGA
Ala	Arg	Asp	Asn	Cys	Glu	Gln	Gly	His	Ile	Leu	Lys	Met	Phe	Pro	Ser	Thr	Trp	Tyr	Val	stop
A	R	D	N	C	E	Q	G	H	I	L	K	M	F	P	S	T	W	Y	V	

Figure 1: The genetic code.

code

The first and simplest analysis that can be done on a genome sequence is the *6-frame translation* of the sequence using the genetic code: read the nucleic acids 3 by 3 and translate the triplets into the corresponding amino acid. Repeat starting from the second base, then the third base, and then with the reverse complement of the sequence, as in this example:

```
atgatcgacgcctcctcagcaagctga
M  I  D  A  S  S  A  S  *
 *  S  T  P  P  Q  Q  A
  D  R  R  L  L  S  K  L
tcagcttgctgaggaggcgtcgatcat
S  A  C  *  G  G  V  D  H
 Q  L  A  E  E  A  S  I
  S  L  L  R  R  R  R  S
```

Every sequence between two stops (∗) is an open reading frame (ORF). Since `AUG` is the start codon, a functional ORF must begin with `M`. Using a mimimum size of 100 amino acids allows to reduce the number of potential ORFs (mean size of human proteins is 300 a.a.).

While this approach allows to find nearly all genes in Yeast, most genes in higher eukaryotes have alternative splicing, meaning that the ORF will be split over non-consecutive *exon* sequences. We will later see how to search for potential spliced genes in a genome using HMMs.

2 Sequence comparison

There exists large collections of Expressed Sequence Tags (*EST*), fully sequenced *clones* and protein data (see `http://www.ncbi.nlm.nih.gov/genbank/`). We can attempt to locate genes in the genome by searching these sequences (including those from other species) in the genome. For example below is an alignment of a shark EST on the mouse genome (the region corresponding to the Hoxd12 gene):

```
chr2_ GAGCACAGCCGAGGCCCTTTGTTGGAGATGTGTGAGCGCAGTCTCTACAGAGCTGGCTAT
                                                             :::::
Shark -------------------------------------------------------GCTAT

chr2_ GTGGGCTCGCTTCTGAATTTACAGTCACCGGACTCTTTCTACTTTTCCAACCTGAGAGCC
      :: ::::: ::  : :::::       :::::  : ::::::::  :::::::: :  :
Shark GTCGGCTCCCTGTTAAATTTTACCAGCCCGGAGCCCTTCTACTTCCCCAACCTGCGTCCG

chr2_ AATGGCAGCCAGTTGGCCGCGCTTCCCCCCATCTCATACCCTCGCAGCGCGCTGCCCTGG
      :::::
Shark AATGG-------------------------------------------------------

chr2_ GCTACTACGCCCGCCTCATGCACCCCTGCGCAGCCTGCCACCGCCTCTGCCTTTGGAGGC
                  : ::::                ::: ::    ::::: :
Shark ------------GGCTCAA---------------CTGGCA---ACTCTGTC---------

chr2_ TTCTCTCAGCCTTACTTGACCGGCTCTGGGCCAATTGGCCTGCAGTCTCCAGGCGCCAAG
                                           ::: ::: :::::
Shark -------------------------------------GCCAGCACTCTCC----------

chr2_ GACGGACCCGAAGACCAGGTCAAGTTCTATACGCCTGATGCGCCCACCGCATCTGAGGAA
                                 :::::              :::::   ::::
Shark ---------------------------TATAC--------------CCGCAGG-GAGGTG

chr2_ CGCAGCCGGACTAGGCCGCCCTTCGCCCCCGAGTCTAGTCTGGTTCATTCGGCTCTCAAA
       ::                   :::: :::: :
Shark TGC-------------------TCGCTCCCGTG---------------------------
```

Formally, the definition of a sequence alignment is

Definition 1. *An **alignment** of the two sequences X (of length n) and Y (of length m) is a sequence of operations match (M), delete (D) and insert (I) such that $\#M + \#D = n$ and $\#M + \#I = m$.*

Example: X = `CACCGCATCTG`, Y = `CCGCAGGGA`, the operations `DDMMMMMMMDMI` generate the alignment:

```
CACCGCATCTG-
DDMMMMMMMDMI
--CCGCAGG-GA
```

Given two sequences, the total number of possible alignments is $\binom{n+m}{m}$, which is exponential in $\min(n, m)$. Therefore we need a notion of which alignment is more accurate than the others (makes fewer mistakes) and an algorithm to find it efficiently (without trying all possibilities). We first define a score for every alignment as follows: we introduce a *scoring matrix*:

$$M = \begin{array}{c} \\ \mathrm{A} \\ \mathrm{C} \\ \mathrm{G} \\ \mathrm{T} \\ - \end{array} \begin{array}{c} \begin{array}{ccccc} \mathrm{A} & \mathrm{C} & \mathrm{G} & \mathrm{T} & - \end{array} \\ \begin{pmatrix} 2 & -1 & -1 & -1 & -\gamma \\ -1 & 2 & -1 & -1 & -\gamma \\ -1 & -1 & 2 & -1 & -\gamma \\ -1 & -1 & -1 & 2 & -\gamma \\ -\gamma & -\gamma & -\gamma & -\gamma & -\infty \end{pmatrix} \end{array} .$$

We call the aligned sequences (including the gaps) X' and Y' and the alignment length L, then the alignment score is

$$S(X', Y'|M) = \sum_{k=1}^{L} M(X'_k, Y'_k) .$$

Choosing $\gamma = 2$, the alignment in definition 1 has a score of

$$-\gamma - \gamma + 2 + 2 + 2 + 2 + 2 - 1 - 1 - \gamma + 2 - \gamma = 10 - 4\gamma = 2 .$$

This is called the *linear gap penalty* (a stretch of n gaps contributes $-n\gamma$ to the score). An often better scoring is obtained with an *affine gap penalty*: each stretch of n gaps contributes $-\delta - n\gamma$ to the score, with the parameter δ making it more costly to open a gap than to extend it. An alignment containing G stretches of gaps therefore has a score of:

$$S(X', Y'|M) = \sum_{k=1}^{L} M(X'_k, Y'_k) - G\delta .$$

3 Global alignment: The Needlman-Wunsch algorithm

This is one example of the technique known as *dynamic programming*: this is a strategy for solving complex problems by recursively breaking them into smaller ones. It assumes that a solution to the largest problem can be obtained by extending the solution to a smaller subproblem. In our case, an optimal alignment of sequences of sizes (n, m) can be found by extending an optimal alignment of the subsequences of lengths $(n-1, m-1)$, $(n, m-1)$, and $(n-1, m)$.

3.1 Scoring

Given the sequences X and Y, we compute $(n+1) \times (m+1)$ scores F_{ij} as follows: the initial score is $F_{0,0} = 0$ and the other scores are computed recursively according to the rule:

$$F_{i,j} = \max \begin{cases} F_{i-1,j-1} + M(X_i, Y_j) \\ F_{i,j-1} + M(-, Y_j) \\ F_{i-1,j} + M(X_i, -) \end{cases} .$$

We keep track of which of the three cases realizes the maximum. This can be implemented in a table, which we fill starting from the top left down to the bottom right

$$\begin{tikzcd}
 & - & C & A & \cdots & G \\
- & 0 \arrow[dr, "M_{CC}"] & -\gamma \arrow[d, "-\gamma"] \arrow[dr, "M_{AC}"] & -2\gamma \arrow[d, "-\gamma"] & \cdots & -n\gamma \\
C & -\gamma \arrow[r, "-\gamma"] \arrow[dr, "M_{CC}"] & F_{11} \arrow[r, "-\gamma"] \arrow[d, "-\gamma"] \arrow[dr, "M_{AC}"] & F_{12} \arrow[d, "-\gamma"] & \cdots & F_{1n} \\
C & -2\gamma \arrow[r, "-\gamma"] & F_{21} \arrow[r, "-\gamma"] & F_{22} & \cdots & F_{2n} \\
\vdots & \vdots & \vdots & \vdots & \ddots \arrow[dr, "M_{GA}"] & \vdots \arrow[d, "-\gamma"] \\
A & -m\gamma & F_{m1} & F_{m2} & \cdots \arrow[r, "-\gamma"] & F_{mn}
\end{tikzcd}$$

The element F_{mn} represents the maximum alignment score achievable. To construct one alignment reaching that score we use the procedure called *backtracking*.

3.2 Backtracking

Starting from the bottom right corner F_{mn} we move back and up in the table following the arrow from which F_{jk} was generated (the one that realized the maximum score). The corresponding path through the table generates an optimal alignment according to the rule: $\leftarrow = D$, $\uparrow = I$, and $\nwarrow = M$. For example with $\gamma = 2$:

$$\begin{tikzcd}
 & - & C & A & C & C & G \\
- & 0 & -2 & -4 & -6 & -8 & -10 \\
C & -2 & \textcircled{2} \arrow[ul, "M"'] & \textcircled{0} \arrow[l, "D"] & -2 \arrow[ul, "M"'] & -4 \arrow[ul, "M"'] & -6 \arrow[l, "D"] \\
C & -4 & 0 \arrow[ul, "M"'] & 1 \arrow[ul, "M"'] & \textcircled{2} \arrow[ul, "M"'] & 0 \arrow[ul, "M"'] & -2 \arrow[l, "D"] \\
C & -6 & -2 \arrow[ul, "M"'] & -1 \arrow[ul, "M"'] & 3 \arrow[ul, "M"'] & \textcircled{4} \arrow[ul, "M"'] & 2 \arrow[l, "D"] \\
G & -8 & -4 \arrow[u, "I"] & -3 \arrow[ul, "M"'] & 1 \arrow[u, "I"] & 2 \arrow[ul, "M"'] & \textcircled{6} \arrow[ul, "M"']
\end{tikzcd}$$

The optimal score is 6, the optimal path (circles) generates the following alignment:

```
CACCG
MDMMM
C-CCG
```

The complexity of this algorithm is $\mathcal{O}(nm)$ in time and in memory.

4 Local alignment: The Smith-Waterman algorithm

The Needlman-Wunsch algorithm generates a *global alignment*, spanning the entire length of both input sequences. We are next going to make two simple modifications to this procedure and find *local alignments* of substrings of X to substrings of Y.

We modify the scoring rule as follows:

$$F_{i,j} = \max \begin{cases} 0 \\ F_{i-1,j-1} + M(X_i, Y_j) \\ F_{i,j-1} + M(-, Y_j) \\ F_{i-1,j} + M(X_i, -) \end{cases} .$$

This means that we never allow scores to become negative. Then, in the backtracking step, we first look for the maximum score inside the whole table (not just the bottom right), and follow the path from there until we reach a score of 0. This gives the best local alignment of the two sequences. We can then restart with the next best score in the table and so on. With the same sequences as above, we get:

$$\begin{tikzcd}
 & - & C & A & C & C & G \\
- & 0 & 0 & 0 & 0 & 0 & 0 \\
C & 0 & \textcircled{2} \arrow[ul, "M"'] & 0 & 0 & 0 & 0 \\
C & 0 & 2 \arrow[ul, "M"'] & \textcircled{1} \arrow[ul, "M"'] & \textcircled{2} \arrow[ul, "M"'] & 2 \arrow[ul, "M"'] & 0 \\
C & 0 & 2 \arrow[ul, "M"'] & 1 \arrow[ul, "M"'] & \textcircled{3} \arrow[ul, "M"'] & \textcircled{4} \arrow[ul, "M"'] & 2 \arrow[l, "D"] \\
G & 0 & 0 & 1 \arrow[ul, "M"'] & 1 \arrow[u, "I"] & 2 \arrow[ul, "M"'] & \textcircled{6} \arrow[ul, "M"']
\end{tikzcd}$$

giving rise to two local alignments:

```
CCG   CAC
MMM   MMM
CCG   CCC
```

5 Remarks on scoring

In this section I would like to discuss more precisely the meaning and the construcution of the scoring matrix M. The numbers 2 for a match and -1 for a mismatch were quite arbitrary. They could be replaced by others depending on what kind of alignments we are looking for. For aligning protein (amino acid) sequences, several different kind of matrices have been proposed that reflect an empirical notion of *chemical similarity* between residues (see `http://en.wikipedia.org/wiki/Substitution_matrix`).

In general these matrices are constructed as follows: count the frequency q_α of each base (residue). The frequency of having α and β at the same position in two unrelated sequences is $q_\alpha \cdot q_\beta$. Let us now design a model for actually related sequences: the probability of α in one sequence being replaced by β in the other is some value $p_{\alpha\beta}$. The total likelihood of the alignment is $L_{\text{model}}(X,Y) = \prod_i p_{x_i y_i}$ in the second case and $L_{\text{random}}(X,Y) = \prod_i q_{x_i} q_{y_i}$ in the random case. The ratio of these two numbers is the *likelihood ratio* or the *odds ratio*. It is most convenient to take the logarithm of the ratio (it transforms products into sums):

$$\mathcal{L}(X,Y) = \log\left(\frac{L_{\text{model}}(X,Y)}{L_{\text{random}}(X,Y)}\right) = \sum_i (\log p_{x_i y_i} - \log q_{x_i} - \log q_{y_i}) = \sum_i M(x_i, y_i) \; .$$

Thus the matrix entries are interpreted as the *log-odds ratios* between a biologically-motivated model of substitutions and a random alignment.

Remark that the model values p_{ij} must be such that the expected value of $\mathcal{L}(X, Y)$ for a random alignment is negative to justify the Smith-Waterman choice of discarding negative scores.

6 BLAST

The most popular local alignment program is called *BLAST* (Basic Local Alignment Search Tool: `http://www.ncbi.nlm.nih.gov/BLAST/`). It uses the following strategy to find alignments between large query sequences (the size of a protein or gene) and large databases (genome or proteome).

1. Remove low-complexity regions from query sequence

2. Cut query in small words (DNA: 11 bases, AA: 3 residues) look for exact matches in the database (pre-computed table)

3. Perform a Smith-Waterman alignment in the neighborhood of each hit to produce a high-scoring segment pair (HSP)

BLAST alignments significance is evaluated using the *E-value* (the expected number of similar alignements found in a random database of the same size). This is computed as follows: we assume (for lack of a better model) that large alignment scores follow an extreme value distribution:

$$E = Kmne^{-\lambda S} ,$$

m and n are the length of the sequences, S the alignment score, and K, λ are unknown parameters of the distribution, which have been estimated from large families of random sequences. The units of S are somewhat unspecified and it is usual to re-express the score as a *bit-score*:

$$S' = \frac{\lambda S - \log K}{\log 2} ,$$

from which we have the expression $E = mn2^{-S'}$.

Reference

1. Durbin, R., Eddy, S. Krogh, A., and Mitchison, G. *Biological sequence analysis*, Cambridge University Press (1998).

2. Altschul, S. F., et al. *Gapped BLAST and PSI-BLAST: a new generation of protein database search programs*, NAR:25 (1997).