The Accelrys Wisconsin Package and pairwise alignment applications

Week 4, Thursday, February 1, 2007

Author and Instructor: Steven M. Thompson

The Accelrys Wisconsin Package®, also known by the company that began it, the Genetics Computer Group (GCG®), is one of the most powerful and comprehensive suites of sequence analysis programs available. It is used in universities, institutions, and for-profit firms worldwide. The Graphical User Interface (GUI) to the Wisconsin Package is SeqLab®.

Today’s computational lab tutorial has several objectives:

1) it teaches how to connect to mendel.scs.fsu.edu, the GCG server at FSU;
2) it introduces the GCG package and its X11 driven front-end SeqLab;
3) it lays out the fundamentals of global and local pairwise alignment algorithms — dynamic programming, substitution matrices, hashing/heuristic searches and their limitations, dot plots, homology versus similarity, and significance ascertainment.

All using our Group Project example, the human SRY protein, encoded by the Sry gene.
Standard Disclaimer

Tutorials are written from a ‘lowest-common-denominator’ biologist’s perspective. That is, I only assume that you have fundamental molecular biology knowledge, but are relatively inexperienced regarding computers. As a consequence of this they are written quite explicitly. Therefore, if you do exactly what is written, they will work. However, this requires two things: 1) you must read very carefully and not skim over vital steps, and 2) you mustn’t take offense if you already know what I’m discussing. I’m not insulting your intelligence.

I use three writing conventions in the tutorials. I use **bold** type for those commands and keystrokes that you are to type in at your keyboard or for buttons or menus that you are to click in a graphical user interface (GUI). I also use **bold** type for section headings. Screen traces are shown in a ‘typewriter’ style Courier font and “///////////” indicates abridged data. The dollar symbol ($) indicates the system prompt and should not be typed as a part of commands. Really **important statements may be underlined**. The actual tutorial doesn’t begin until page 20, entitled “Week Four Tutorial;” it’s all background material before that. It is not essential to read the background material before performing the tutorial, but please do at some point.

Introduction

Given the nucleotide or amino acid sequence of a biological molecule, what can we know about that molecule? We can find relevant information in sequences by searching for particular patterns that reflect some function of the molecule such as certain motifs and domains. What about comparisons with other sequences? Is this worthwhile? Yes, naturally it is; inference through homology is fundamental to all the biological sciences. We can learn a tremendous amount by comparing our sequence against others.

However, a ‘brute force’ approach just won’t work. Even without considering the introduction of gaps, the computation required to compare all possible alignments between two sequences requires time proportional to the product of the lengths of the two sequences. Therefore, if the two sequences are approximately the same length (N), this is a $N^2$ problem. To include gaps, we would have to repeat the calculation $2N$ times to examine the possibility of gaps at each possible position within the sequences, now a $N^{4N}$ problem. Waterman illustrated the problem in 1989 stating that to align two sequences 300 symbols long, $10^{88}$ comparisons would be required, about the same number as the number of elementary particles estimated to exist in the universe! Part of the solution to this problem is the dynamic programming algorithm.

Dynamic programming

Let’s begin with an overview of dynamic programming. I’ll use an oversimplified example first — we’ll consider matching symbols to be worth one point, and will not consider gapping at all. The solution occurs in two stages. The first begins very much like dot matrix methods; the second is totally different. Instead of calculating the ‘score matrix’ on the fly, as is often taught as one proceeds through the graph, I like to completely fill in an original ‘match matrix’ first, and then add points to those positions that produce favorable
alignments next. Points are added based on a “looking-back-over-your-left-shoulder” algorithm rule where the only allowable trace-back is diagonally behind and above. The illustration follows below:

a) A completed match matrix using one point for matching and zero points for mismatching:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>A</th>
<th>T</th>
<th>G</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

d) The score matrix is now complete:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>A</th>
<th>T</th>
<th>G</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0+1</td>
<td>0+1</td>
<td>0+1</td>
<td>1+2</td>
</tr>
</tbody>
</table>

e) Now pick the bottom, right-most, highest scores in the matrix and work your way back through it, in the opposite direction as before. This is called the trace-back stage and the matrix is now referred to as the path graph. In this case that highest score is in the right-hand corner, but it need not be:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>A</th>
<th>T</th>
<th>G</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

f) The completed traceback is shown with outline characters; these are all optimal alignments:
The following alignments are all generated from the above path graph (f). All five have three matches. Gap penalties would have eliminated the last two of them; however, that still leaves three:

```
AG . GC   A . GGC   . AGGC   A . GC   A . GGC
  |   |     |   |     |   |     |   |     |   |
AATGC   AATGC   AATGC   AATG . C   AATG . C
```

The software will arbitrarily (based on some rule) choose only one of these to report as optimal. This decision can be partly controlled in some programs with a highroad/lowroad option.

The next example will be slightly more difficult. Unlike the previous example without gap penalties, I will now impose a very simple gap penalty function. Matching symbols will still be worth one point, non-matching symbols will still be worth zero points, but we will penalize the scoring scheme by subtracting one point for every gap inserted, unless they are at the beginning or end of the sequence. In other words, end gaps will not be penalized, i.e. both sequences do not have to begin or end at the same point in the alignment.

This zero penalty end-weighting scheme is the default for most alignment programs, but can often be changed with a program option, if desired. However, the gap function described here and used in the example below is a much simpler gap penalty function than normally used in alignment programs. Normally an ‘affine,’ function is used, the standard $y = mx + b$ equation for a line:

$$
\text{total penalty} = \left( \text{length of gap} \right) \times \left( \text{gap extension penalty} \right) + \text{gap opening penalty}
$$

(To run most alignment programs with the type of simple DNA gap penalty used here, you have to designate a gap ‘creation’ or ‘opening’ penalty of zero and a gap ‘extension’ penalty of whatever counts in that particular program as an identical base match for DNA sequences.)

This example uses two randomly generated sequences that happen to fit the tata promoter region consensus of eukaryotes and bacteria. The most conserved bases within the consensus are capitalized. The sample eukaryote promoter sequence is along the X-axis and the bacterial sequence is along the Y-axis in my example on the next page.

The solution illustration begins in the left panel below on the following page:
a) First complete a match matrix using one point for matching and zero points for mismatching between bases, just like in the previous example without any gap penalties:

\[
\begin{array}{cccccccc}
  & c & T & A & T & t & A & a & g & g \\
  c & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
  g & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
  T & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
  A & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\
  t & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
  A & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\
  a & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\
  T & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

b) Now add and subtract points based on the best path through the matrix, working diagonally, left to right and top to bottom, just as before. However, when you have to jump a box to make the path, subtract one point per box jumped, except at the beginning or end of the alignment, so that end gaps are not penalized. Fill in all additions and subtractions, and calculate the sums and differences as you go. The score matrix is shown with calculations below:

\[
\begin{array}{cccccccc}
  & c & T & A & T & t & A & a & g & g \\
  c & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
  g & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
  T & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
  A & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\
  t & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
  A & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\
  a & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\
  T & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

c) Clean up the score matrix next. I'll only show the totals in each cell in the matrix shown below:

\[
\begin{array}{cccccccc}
  & c & T & A & T & t & A & a & g & g \\
  c & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
  g & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
  T & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\
  A & 0 & 0 & 2 & 1 & 2 & 0 & 2 & 1 & 0 \\
  t & 0 & 1 & 0 & 3 & 1 & 3 & 1 & 2 & 1 \\
  A & 0 & 0 & 2 & 1 & 4 & 2 & 4 & 3 & 2 \\
  a & 0 & 0 & 1 & 2 & 3 & 4 & 4 & 5 & 3 \\
  T & 0 & 1 & 0 & 2 & 2 & 4 & 4 & 4 & 5 \\
\end{array}
\]

d) Finally, convert the score matrix into a traceback path graph by picking the bottom-most, furthest right and highest scoring coordinates. Then choose traceback route that got you there, to connect them all the way back to the beginning using the same 'over-your-left-shoulder' rule. These are shown in outline font in the matrix below:

\[
\begin{array}{cccccccc}
  & c & T & A & T & t & A & a & g & g \\
  c & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
  g & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
  T & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\
  A & 0 & 0 & 2 & 1 & 2 & 0 & 2 & 1 & 0 \\
  t & 0 & 1 & 0 & 3 & 1 & 3 & 1 & 2 & 1 \\
  A & 0 & 0 & 2 & 1 & 4 & 2 & 4 & 3 & 2 \\
  a & 0 & 0 & 1 & 2 & 3 & 4 & 4 & 5 & 3 \\
  T & 0 & 1 & 0 & 2 & 2 & 4 & 4 & 4 & 5 \\
\end{array}
\]

There may be more than one best path through the matrix. This time, starting at the top and working down as we did, then tracing back, I found one optimum alignment:

\[
\text{cTAATaagg}
\]
With our example’s scoring scheme this alignment has a final score of 5, the highest bottom-right score in the trace-back path graph, and the sum of six matches minus one interior gap. This is the number optimized by the algorithm, not any type of a similarity or identity percentage! The software will arbitrarily (based on some rule) choose one optimal solution. Some programs offer a highroad/lowroad option to help explore this solution space. The above solution is the GCG HighRoad solution found when running their program Gap with the above example’s parameter settings (note that GCG uses a score of 10 for base matches, not 1):

GAP of: Euk_Tata.Seq to: Bact_Tata.Seq  
Euk_Tata: A random example Eukaryotic promoter TATA Box  
Bact_Tata: A random sequence that fits the consensus from the standard E. coli RNA polymerase promoter ‘Pribnow’ box -10 region.

<table>
<thead>
<tr>
<th></th>
<th>HighRoad option</th>
<th>LowRoad option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gap Weight:</td>
<td>0</td>
<td>Average Match:</td>
</tr>
<tr>
<td>Length Weight:</td>
<td>10</td>
<td>Average Mismatch:</td>
</tr>
<tr>
<td>Quality:</td>
<td>50</td>
<td>Quality:</td>
</tr>
<tr>
<td>Ratio:</td>
<td>6.250</td>
<td>Ratio:</td>
</tr>
<tr>
<td>Percent Similarity:</td>
<td>75.000</td>
<td>Percent Similarity:</td>
</tr>
<tr>
<td>Length:</td>
<td>10</td>
<td>Length:</td>
</tr>
<tr>
<td>Gaps:</td>
<td>2</td>
<td>Gaps:</td>
</tr>
<tr>
<td>Percent Identity:</td>
<td>75.000</td>
<td>Percent Identity:</td>
</tr>
</tbody>
</table>

1 cTATATaAagg 10          1 cTATATaAagg 10
| |||||                      |||||  
1 cg.TATAtaA. 8           1 .cgTATAtaA. 8

Do you have any ideas about how others could be discovered? Answer: Often if you reverse the solution of the entire dynamic programming process, other solutions are found! In other words, reverse the sequences in software programs to see alternative alignments.

To recap, and for those people that like equations, an optimal pairwise alignment is defined as an arrangement of two sequences, 1 of length \( i \) and 2 of length \( j \), such that:

1) you maximize the number of matching symbols between 1 and 2;
2) you minimize the number of gaps within 1 and 2; and
3) you minimize the number of mismatched symbols between 1 and 2.

Therefore, the actual solution can be represented by:

\[
S_{ij} = s_{ij} + \max \begin{cases} \max_{2 < x < i} S_{i-x} j-1 + w_{x-1} \quad \text{or} \quad \max_{2 < y < i} S_{i-1} j-y + w_{y-1} \end{cases}
\]

where \( S_{ij} \) is the score for the alignment ending at \( i \) in sequence 1 and \( j \) in sequence 2, \( s_{ij} \) is the score for aligning \( i \) with \( j \), \( w_x \) is the score for making a \( x \) long gap in sequence 1,
wy is the score for making a $y$ long gap in sequence 2, allowing gaps to be any length in either sequence.

However, just because dynamic programming guarantees an optimal alignment, it is not necessarily the only optimal alignment. Furthermore, the optimal alignment is not necessarily the ‘right’ or biologically relevant alignment! As always, question the results of any computerized solution based on what you know about the biology of the system. The above example illustrates the Needleman and Wunsch (1970) global solution. Later refinements (Smith and Waterman, 1981) demonstrated how dynamic programming could also be used to find optimal local alignments. To solve dynamic programming using local alignment (without going into all the gory details) programs use the following two tricks:

1) An identity match matrix that uses negative numbers for mismatches is incorporated. Therefore, bad paths quickly become very bad. This leads to a trace-back path matrix with many alternative paths, most of which do not extend the full length of the graph.

2) The best trace-back within the graph is chosen. This does not have to begin or end at the edges of the graph — it is looking for the best segment of alignment!

**Scoring matrices**

What about protein sequences — conservative replacements and similarities, as opposed to identities? This is definitely an additional complication. Certain amino acids are very much alike, structurally, chemically, and genetically. How can we take advantage of the similarity of amino acids in our alignments? People have been struggling with this problem since the late 1960’s. Margaret Dayhoff (Schwartz and Dayhoff, 1979) unambiguously aligned closely related protein datasets (no more than 15% difference, and in particular cytochrome c) available at that point in time and noticed that certain residues, if they mutate at all, are prone to change into certain other residues. As it works out, these propensities for change fell into the same categories that chemists had known for years — those same chemical and structural classes mentioned above, conserved through the evolutionary constraints of natural selection. However, Dayhoff’s empirical observation quantified these changes. Based on the multiple sequence alignments that she created, the assumption that estimated mutation rates in closely related proteins can be extrapolated to more distant relationships, and matrix and logarithmic mathematics, she was able to empirically specify the relative probabilities at which different residues mutated into other residues through evolutionary history as appropriate within some level of divergence between the sequences considered. This is the basis of the famous PAM (corrupted acronym of accepted point mutation) 250 (meaning that the matrix has been multiplied by itself 250 times) log odds matrix.

Since Dayhoff’s time other biomathematicians (esp. see Henikoff and Henikoff’s [1992] BLOSUM series of tables, and for a controversial matrix see Gonnet et al. [1992]) have created newer matrices with more or less success than Dayhoff’s original, but the concept remains the same. Furthermore, Dayhoff’s original PAM 250 table remains a classic as historically the most widely used amino acid substitution matrix. Collectively these
types of tables are known as symbol comparison tables, log odds matrices, or scoring matrices, and they are fundamental to all sequence comparison techniques. The default amino acid scoring matrix for most protein similarity comparison programs is now the BLOSUM62 table (Henikoff and Henikoff, 1992). It follows below; values whose magnitude is \( \geq 4 \) are drawn in shadowed characters to make them easier to recognize:

|   | A | B | C | D | E | F | G | H | I | K | L | M | N | P | Q | R | S | T | V | W | X | Y | Z |
| A | 4 | -2 | -2 | -1 | -2 | 0 | 0 | -2 | -1 | -1 | -1 | -1 | -1 | 1 | 1 | 0 | 0 | -3 | -1 | -2 | -1 |
| B | -2 | 6 | -3 | 0 | 6 | 2 | -3 | -1 | -1 | -1 | 3 | -1 | -2 | -1 | -1 | -1 | 0 | 0 | -3 | -1 | -3 | -2 |
| C | 0 | -3 | 9 | -3 | -4 | -2 | -3 | -3 | -1 | -1 | -1 | -1 | -1 | -1 | -3 | -3 | -3 | -3 | -3 | -1 | -1 | -2 | -1 | -2 | -4 |
| D | -2 | 6 | -3 | 6 | 2 | -3 | -1 | -1 | -1 | -3 | -1 | -4 | -3 | 1 | 1 | -1 | 0 | 2 | 0 | -1 | -3 | -1 | -4 | -1 | -3 | 2 |
| E | -1 | 2 | -4 | 2 | 5 | -3 | -2 | 0 | -3 | 1 | -1 | -3 | -2 | 0 | -1 | 2 | 0 | 0 | -1 | -2 | -3 | -1 | -2 | 2 | 5 |
| F | -2 | -3 | -2 | -3 | 6 | -3 | -1 | 0 | -3 | 0 | 0 | -3 | -4 | -3 | -3 | -2 | -2 | -1 | 1 | -1 | 1 | 3 | -1 | -3 | 3 |
| G | 0 | -1 | -1 | -1 | -2 | -3 | 6 | -2 | -4 | -2 | -4 | -3 | 0 | -2 | -2 | -2 | 0 | -2 | -3 | -1 | -3 | -2 | -1 | -3 | 2 |
| H | -2 | -1 | -3 | -1 | -1 | 0 | -1 | -2 | -3 | -1 | -3 | -2 | 1 | -2 | 0 | 0 | -1 | -2 | -3 | -1 | -2 | 1 | -2 | 2 | 0 |
| I | -1 | -3 | -1 | -3 | -3 | 0 | -4 | -3 | -4 | 4 | -3 | 2 | 1 | -3 | -3 | -3 | -3 | -2 | -1 | 3 | -3 | -1 | -1 | -3 |
| K | -1 | -1 | -3 | -1 | 1 | -3 | -2 | -1 | -3 | 5 | -2 | -1 | 0 | -1 | 1 | 2 | 0 | -1 | 2 | -3 | -1 | -2 | 1 |
| L | -1 | -4 | -1 | -4 | -3 | 0 | -4 | -3 | 2 | -2 | 4 | 2 | -3 | -3 | -2 | -2 | -2 | -1 | 1 | -2 | -1 | -1 | -3 |
| M | -1 | -3 | -1 | -3 | -2 | 0 | -3 | -2 | -1 | 1 | -2 | 5 | -2 | -2 | 0 | -1 | -1 | 1 | 1 | -1 | -1 | -1 | -2 |
| N | -2 | 1 | -3 | 1 | 0 | -3 | 0 | 1 | 1 | 3 | 0 | -3 | -2 | 6 | -2 | 0 | 0 | 1 | 0 | -3 | -4 | -1 | -2 | 0 |
| P | -1 | -1 | -3 | -1 | -1 | -4 | -2 | -2 | -3 | -1 | -3 | -2 | -2 | 7 | -1 | -2 | -1 | -1 | -2 | -4 | -1 | -3 | -1 |
| Q | -1 | 0 | -3 | 0 | 2 | -3 | -2 | 0 | -3 | 1 | -2 | 0 | 0 | -1 | 5 | 1 | 0 | -1 | 2 | -2 | -1 | -1 | 1 |
| R | -1 | -2 | -3 | -2 | 0 | -3 | -2 | 0 | -3 | 2 | -2 | -1 | 0 | 0 | -1 | 5 | 1 | -1 | -3 | -3 | -3 | -2 | 0 |
| S | 1 | 0 | 1 | 0 | 0 | -2 | 0 | -1 | -2 | 0 | -2 | 1 | -1 | 1 | 1 | 0 | -1 | 4 | 1 | -2 | -3 | -1 | -2 |
| T | 0 | -1 | -1 | -1 | -2 | -2 | -1 | -1 | -1 | -1 | 0 | -1 | -1 | -1 | 1 | 5 | 0 | -2 | 1 | -2 | -1 |
| V | 0 | -3 | -1 | -3 | -2 | -1 | -3 | -3 | 3 | -2 | 1 | 1 | -3 | -2 | -3 | -2 | 0 | 4 | -3 | -1 | -1 |
| W | -3 | -4 | -2 | -4 | -3 | 1 | -2 | -2 | -3 | -1 | -4 | -4 | -2 | -3 | -3 | -3 | -2 | 3 | 11 | 1 | -2 |
| X | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | 1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| Y | -2 | -3 | -2 | -3 | -2 | 3 | -3 | 2 | -1 | -2 | 1 | -1 | -2 | -3 | -3 | -1 | -2 | -2 | 1 | -2 | 1 | 7 | -2 |
| Z | -1 | 2 | -4 | 2 | 5 | -3 | -2 | 0 | -3 | 1 | -3 | -2 | 0 | 1 | 2 | 0 | 0 | -1 | -2 | -3 | -1 | -2 | 5 |

Notice that positive values for identity range from 4 to 11 and negative values for rare substitutions go as low as negative 4. The most conserved residue is tryptophan with an identity score of 11; cysteine is next with a score of 9; histidine gets 8; both proline and tyrosine get scores of 7. Also check out the hydrophobic substitution triumvirate — isoleucine, leucine, valine, and to a lesser extent methionine — all easily swap places. So rather than using the one/zero match function that we used in the simple tata dynamic programming example above, protein sequence alignments use the match function provided by a scoring matrix like this. The concept of similarity becomes very important with some amino acids being way 'more similar' than others!

### Database searching

After all of these concepts are considered we can screen databases to look for sequences to compare ours to. But what do database searches tell us and what can we gain from them? Why even bother? As I stated earlier, inference through homology is a fundamental principle in biology. When a sequence is found to fall into a preexisting group we can infer function, mechanism, evolution, and possibly even structure based on homology with its neighbors. Database searches can even provide valuable insights into enzymatic mechanism. What ‘family’ does your sequence fall into? Even if no similarity can be found, the very fact that
your sequence is new and different could be very important. It’s going to be a lot more difficult to discover functional and structural data about it, but in the long run its characterization might prove very rewarding.

**Significance**

A big question and a particularly common misnomer made in this area is the concept of homology versus similarity: There is a huge difference! Similarity is merely a statistical parameter that describes how much two sequences, or portions of them, are alike according to some set scoring criteria. It can be normalized to ascertain statistical significance as seen in the database searching methods described below, but it’s still just a number. Homology, in contrast and by definition, implies an evolutionary relationship — more than just the fact that we have all evolved from the same old primordial ‘ooze.’ You need to be able to demonstrate some type of lineage between the organisms or genes of interest in order to claim homology. Even better, be able show some experimental evidence, structural, morphological, genetic, or fossil, that corroborates your assertion. There really is no such thing as percent homology; something is either homologous or it is not. The famous molecular evolutionist Walter Fitch likes to relate the joke “homology is like pregnancy — you can’t be 45% pregnant, just like something can’t be 45% homologous. You either are or you are not.” Do not make the all too commonly made mistake of calling any old sequence similarity homology. Highly significant similarity can argue for homology, but never the other way around.

So, how do you tell if a similarity, in other words, an alignment discovered by some program, means anything? Is it statistically significant, is it truly homologous, and even more importantly, does it have anything to do with real biology? Many of the programs generate percent similarity scores, however these really don’t mean a whole lot. Do not use percent similarities or identities to compare sequences except in the roughest way. They are not optimized or normalized in any manner by the programs. The ‘quality’ scores mean a lot more but are difficult to interpret. At least they take the length of similarity, all of the necessary gaps introduced, and the matching of symbols all into account, but quality scores are only relevant within the context of a particular comparison or search. The quality ratio is the metric optimized by dynamic programming divided by the length of the shorter sequence. As such it represents a fairer comparison metric but it also is relative to the particular scoring matrix and gap penalties used in the procedure. Some of the programs can generate histograms of score distributions, but again, they can be confusing. Read the documentation on the various methods to get a better handle on what these various scores mean.

A traditional way of deciding alignment significance relies on an old statistics trick — Monte Carlo simulations. This type of significance estimation has implicit statistical problems; however, few practical alternatives exist for just comparing two sequences. Monte Carlo techniques continue to be used because of their ease and speed, and will remain important in the field for a long time. Monte Carlo methods compare an actual score, in this case the quality score of an alignment, against the distribution of scores of alignments of a randomized sequence. Therefore, one way of deciding alignment significance is to take advantage of the Monte Carlo style randomizations option available in the two GCG dynamic programming comparison programs BestFit and Gap. To utilize this strategy, compare two sequences using the appropriate algorithm, either Gap or
BestFit depending on whether you’re trying to compare the entire length of each sequence or only the best regions of similarity of each, respectively, and specify the command line option “--randomizations=100”. This option jumbles the second sequence of the comparison 100 times after the initial alignment is produced and then generates scores and a standard deviation based on the jumbled matches. Comparing the quality scores of the randomized alignments to the initial alignment can help give a feeling for the relative meaning of the scores. You can compare the mean of the random scores to the unjumbled score using a ‘Z score’ calculation to help decide significance. An old ‘rule-of-thumb’ that people often use is, if the actual score is much more than three standard deviations above the mean of the randomized scores, the analysis may be significant; if it is much more than five, than it probably is significant; and if it is above nine, than it definitely is significant. Many Z scores measure the distance from a mean using this simplistic Monte Carlo model assuming a normal distribution, in spite of the fact that ‘sequence-space’ actually follows what is know as an ‘extreme value distribution;’ however, the method does approximate significance estimates quite well and is calculated with the following formula:

$$Z \text{ score } = \frac{ ( \text{ actual score } ) - ( \text{ mean of randomized scores } ) }{ ( \text{ standard deviation of randomized score distribution } ) }$$

When the two TATA sequences from the previous dynamic programming example are compared to one another using the same scoring parameters as before, but incorporating a Monte Carlo Z score calculation, their similarity is found, surprisingly, not to be at all significant. It is merely a reflection of the compositional bias of the two sequences to contain lots of T’s and A’s. Those results follow: Average quality based on 100 randomizations: 41.8 +/- 7.4. Plugged into the formula: ( 50 – 41.8 ) / 7.4 = 1.11, i.e. there is no significance to the match in spite of 75% identity! Composition can make a huge difference — the similarity is merely a reflection of the relative abundance of A’s and T’s in the sequences!

The FastA (Pearson and Lipman, 1988; and Pearson, 1998), BLAST (Altschul, et al., 1990), ProfileSearch (Gribskov, et al., 1987), and HmmrSearch (Eddy, 1998) algorithms, all use a similar approach but base their statistics on the distance of the query matches from the actual, or a simulated, extreme value distribution from the rest of the, ‘insignificantly similar,’ members of the database being searched. BLAST, FastA, and HmmrSearch generate Expectation values in this manner; ProfileSearch returns extreme value distribution Z scores, which follow the same guidelines as described above. Expectation values are printed in scientific notation and the smaller the number, i.e. the closer it is to 0, the more significant the match is. Expectation values show us how often we could expect that particular alignment match to occur merely by chance alone in a search of that size database. In all cases, these are the numbers to pay attention to.

Rough, conservative guidelines to Z scores and Expectation values from a typical protein search follow:

<table>
<thead>
<tr>
<th>~Z score</th>
<th>~E value</th>
<th>Inference</th>
</tr>
</thead>
<tbody>
<tr>
<td>≤3</td>
<td>≥0.1</td>
<td>little, if any evidence for homology, but impossible to disprove!</td>
</tr>
<tr>
<td>=5</td>
<td>=10^{-2}</td>
<td>probably homologous, but may be due to convergent evolution</td>
</tr>
<tr>
<td>≥10</td>
<td>≤10^{-3}</td>
<td>definitely homologous</td>
</tr>
</tbody>
</table>
Be very careful with any guidelines such as these, though, because they are entirely dependent on both the size and content of the database being searched as well as how often you perform the search! Think about it — the odds are way different for rolling a “Yahtzee” depending on how many dice you roll, whether they are ‘loaded’ or not, and how often you try.

Another very powerful empirical method of determining significance is to repeat a database search with the entry in question. If that entry finds more significant ‘hits’ with the same sorts of sequences as the original search, then the entry in question is undoubtedly homologous to the original entry. That is, homology is transient. If it finds entirely different types of sequences, then it probably is not. Modular proteins with distinctly separate domains confuse issues considerably, but the principles remain the same, and can be explained through domain swapping and other examples of non-vertical transmission. And, finally, the ‘Gold-standard’ of homology is shared structural folds — if you can demonstrate that two proteins have the same structural fold, then, regardless of similarity, at least that particular domain is homologous between the two.

The searching programs

Database searching programs use elements of all the concepts discussed above; however, classic dynamic programming techniques take far too long when used against most databases with a ‘normal’ computer. Therefore, the programs use tricks to make things happen faster. These tricks fall into two main categories, that of hashing and that of approximation. Hashing is the process of breaking your sequence into small ‘words’ or ‘k-tuples’ of a set size and creating a ‘look-up’ table with those words keyed to numbers. Then when any of the words match part of an entry in the database, that match is saved. In general, hashing reduces the complexity of the search problem from $N^2$ for dynamic programming to $N$, the length of all the sequences in the database. Approximation techniques are collectively known as ‘heuristics.’ Webster’s defines heuristic as “serving to guide, discover, or reveal; . . . but unproved or incapable of proof.” In database searching techniques the heuristic usually restricts the necessary search space by calculating some sort of a statistic that allows the program to decide whether further scrutiny of a particular match should be pursued. This statistic may miss things depending on the parameters set — that’s what makes it heuristic. The exact implementation varies between the different programs, but the basic ideas follow in all of them.

Two predominant versions exist: the Fast and BLAST programs. Both return local alignments. Both are not a single program, but rather a family of programs with implementations designed to compare a sequence to a database in about every which way imaginable. These include: a DNA sequence against a DNA database (not recommended unless forced to do so because you are dealing with a nontranslated region of the genome), a translated (where the translation is done ‘on-the-fly’ in all six frames) version of a DNA sequence against a translated (‘on-the-fly’) version of the DNA database (only available in BLAST), a translated (‘on-the-fly’) version of a DNA sequence against a protein database, a protein sequence against a translated (‘on-the-fly’) version of the DNA database, or a protein sequence against a protein database. Many implementations allow the recognition of frame shifts in translated comparisons. In more detail:
FastA and family, developed at the University of Virginia (Pearson and Lipman, 1988; Pearson, 1998)

1) Works well for DNA against DNA searches (within limits of possible sensitivity);
2) Can find only one gapped region of similarity;
3) Relatively slow, should usually be run in the background;
4) Does not require specially prepared, preformatted databases.

FastA is an older algorithm than BLAST. It was the first widely used, powerful sequence database searching program. Pearson continually refines the algorithm such that it remains a viable alternative to BLAST, especially if one is restricted to searching DNA against DNA without translation. It is also helpful in situations where BLAST finds no significant alignments; FastA may be more sensitive than BLAST in these situations.

The algorithm:

FastA also builds words of a set k-tuple size, by default two for peptides. It then identifies all exact word matches between the sequence and the database members. Scores are assigned to each continuous, ungapped, diagonal by adding all of the exact match BLOSUM values. The ten highest scoring diagonals for each query-database pair are then re-scored using BLOSUM similarities as well as identities and ends are trimmed to maximize the score. The best of each of these is called the Init1 score. Next the program ‘looks’ around to see if nearby off-diagonal Init1 alignments can be combined by incorporating gaps. If so, a new score, Initn, is calculated by summing up all the contributing Init1 scores, penalizing gaps with a penalty for each. The program then constructs an optimal local alignment for all Initn pairs with scores better than some set threshold using a variation of dynamic programming “in a band.” A sixteen residue band centered at the highest Init1 region is used by default with peptides. A score is generated from this step known as the opt score. Next, FastA uses a simple linear regression against the natural log of the search set sequence length to calculate a normalized z-score for the sequence pair. Finally, it compares the distribution of these z-scores to the actual extreme-value distribution of the search. Using this distribution, the program estimates the number of sequences that would be expected to have, purely by chance, a z-score greater than or equal to the z-score obtained in the search. This is reported as the Expectation value. Unfortunately, the z-score used in FastA and the previously discussed Monte Carlo style Z score are quite different and can not be directly compared. If the user requests pairwise alignments in the output, then the program uses full Smith-Waterman local dynamic programming, not ‘restricted to a band,’ to produce its final alignments.

BLAST — Basic Local Alignment Search Tool, developed at NCBI (Altschul et al. 1990 and 1997)

1) Normally not a good idea to use for DNA against DNA searches (not optimized);
2) Prefilters repeat and “low complexity” sequence regions by default;
4) Can find more than one region of gapped similarity;
5) Very fast heuristic and parallel implementation;
6) Restricted to precompiled, specially formatted databases;
The algorithm:

After BLAST has sorted its lookup table, it tries to find all double word hits along the same diagonal within some specified distance using what NCBI calls a Discrete Finite Automaton (DFA). These word hits of size $W$ do not have to be identical; rather, they have to be better than some threshold value $T$. To identify these double word hits, the DFA scans through all strings of words (typically $W=3$ for peptides) that score at least $T$ (usually 11 for peptides). Each double word hit that passes this step then triggers a process called ungapped extension in both directions, such that each diagonal is extended as far as it can, until the running score starts to drop below a pre-defined value $X$ within a certain range $A$. The result of this pass is called a High-Scoring segment Pair or HSP.

Those HSPs that pass this step with a score better than $S$ then begin a gapped extension step utilizing dynamic programming. Those gapped alignments with Expectation values better than the user specified cutoff are reported. The extreme value distribution of BLAST Expectation values is pre-computed against each precompiled database — this is one area that speeds up the algorithm considerably.

The math can be generalized thus: for any two sequences of length $m$ and $n$, local, best alignments are identified as HSPs. HSPs are stretches of sequence pairs that cannot be further improved by extension or trimming, as described above. For un-gapped alignments, the number of expected HSPs with a score of at least $S$ is given by the formula: $E = Kmne^{-\lambda s}$.

This is called an $E$-value for the score $S$. In a database search $n$ is the size of the database in residues, so $N=mn$ is the search space size. $K$ and $\lambda$ are be supplied by statistical theory, and, can be calculated by comparison to pre-computed, simulated distributions. These two parameters define the statistical significance of an $E$-value. The $E$-value defines the significance of the search. As mentioned above, the smaller an $E$-value is, the more likely it is significant. A value of 0.01 is a good starting point for significance in most typical searches. In other words, in order to assess whether a given alignment constitutes evidence for homology, it helps to know how strong an alignment can be expected from chance alone.

In review, both the BLAST and FastA family of programs base their Expectation “$E$” values on a more realistic ‘extreme value distribution,’ based on either real or simulated ‘not significantly similar’ database alignments, than Monte Carlo style Z scores do. Regardless, they follow Monte Carlo style Z scores fairly well. The higher the $E$ value is, the more probable that the observed match is due to chance in a search of the same size database and the lower its Z score will be. Therefore, the smaller the $E$ value, i.e. the closer it is to zero, the more significant it is and the higher its Z score will be! The $E$ value is the number that really matters.

Furthermore, all database searching, regardless of the algorithm used, is far more sensitive at the amino acid level than at the DNA level. This is because proteins have twenty match criteria versus DNA’s four and those four DNA bases can only be identical, not similar, to each other; and many DNA base changes (especially third position changes) do not change the encoded protein. All of these factors drastically increases the
‘noise’ level of a DNA against DNA search, and gives protein searches a much greater ‘look-back’ time, typically doubling it. Therefore, whenever dealing with coding sequence, it is always prudent to search at the protein level. Even though protein searching is more sensitive, the DNA databases have more data. This drawback can be overcome with programs that take a protein query and compare it to translated nucleotide databases, or the other way around, but one still needs to know if the translation is ‘real.’ This disadvantage is negligible though and can be investigated after the fact, so the general rule when dealing with coding sequence is to either search protein query against protein database, or DNA query against protein database.

So, how do you do this stuff? One way — GCG on Mendel — FSU’s biocomputing server

How do you use these techniques in ‘real life?’ The Internet through the WWW is often used. This is possible and easy and fun, but besides being a bit too easy too get sidetracked upon, the Web environment can’t readily handle large datasets. They quickly become intractable. You’ll know you’re there when you try it. What are the alternatives to Web based sequence analysis? Desktop software can be installed on your own personal computer. Free, public domain sequence analysis programs are available, but they can be somewhat complicated to install, configure, and maintain. The user must be pretty computer savvy, especially to get them to all cooperate with one another. Commercial software packages, such as MacVector, DNAsis, DNASTar, etc., are available, but license hassles, big expenses per machine, and database access all complicate matters. Therefore, non-Web, server-based solutions are often employed. These require network access to UNIX server computers. A big advantage of this solution is access to fast, powerful programs with convenient database access, all on the same server. Connections can be made from any networked computer anywhere! Again, free public domain solutions are available, but now a very cooperative systems administrator must build and maintain the system for its users. However, with a commercial biocomputing package there are minimum systems management concerns, and only one license fee for the entire institution, rather than individual licenses for every machine running the package. From an economics point of view, it’s a ‘no-brainer’ for an institution to support a commercial server based biocomputing solution.

Florida State University’s main biocomputing server for sequence analysis is a Dell PowerEdge 6650 named Mendel bought with Howard Hughes Medical Institute undergraduate education grant monies. Mendel has four 1.6 GH Intel Xeon CPUs, eight GB of RAM, and over 800 GB of storage. The machine (mendel.scs.fsu.edu) is managed by, and located in, the School of Computational Science (SCS), and runs RedHat Enterprise Linux version 4. Mendel only allows ssh, scp, and sftp connections. In order to display the X11 Windows that SeqLab requires on your local computer, you will need to allow ssh X tunneling. You’ll learn what this means today. I’ve created accounts on Mendel for everyone enrolled in the course.

The Genetics Computer Group

The Wisconsin Package for Sequence Analysis began as a service project in 1982 in Oliver Smithies’ lab in the Genetics Department at the University of Wisconsin, Madison. It spun off that effort into a University Research Park location becoming an independent private company, the Genetics Computer Group (GCG), in
1990. Then in 1997 the Oxford Molecular Group of Great Britain, a chemical informatics company, acquired GCG. The drug discovery and development firm Pharmacopeia next purchased GCG, and the other Oxford Molecular holdings, in late 2000. Then in summer 2001, it, along with Pharmacopeia’s other software holdings, were all placed under the new corporate name Accelrys, Inc., which became a subsidiary of Pharmacopeia. Most recently, in 2004, Accelrys left Pharmacopeia to become an independent entity. GCG’s birth and growth are a real success story among the death of so many dot.coms and biotech endeavors of the late ’90’s and early 2000’s. For more information on Accelrys and the Wisconsin Package’s history see the links at http://www.accelrys.com/about/history/gcg.html.

The Wisconsin Package has arguably become a global ‘industry-standard’ in sequence analysis software. It provides a comprehensive suite of nearly 150 integrated DNA and protein analysis programs, from database, pattern, and motif searching; fragment assembly; mapping; and sequence comparison; to gene finding; protein and evolutionary analysis; primer selection; and DNA and RNA secondary structure prediction. The package’s programs work together in a "toolbox" fashion. Much like a carpenter’s toolbox, where using the right tool correctly in the right order can build a house, several relatively simple programs properly used in succession can lead to sophisticated sequence analysis results with the Wisconsin Package. Furthermore, the programs are ‘internally compatible.’ By this I mean that once you learn how to use one program, how the programs ‘look and feel,’ then you pretty much know how to run all of the programs, since all ‘act’ similarly, and, most importantly, the output from many programs can be used as input for other programs. This is how you use the programs in a logical succession. The present version of the Wisconsin Package only runs on computers running the UNIX operating system, but it can be accessed from any networked terminal. More than 30,000 scientists worldwide at over 950 institutions in more than 30 countries use the package, so learning it here may be useful at other points in your future.

SeqLab is included as a part of the GCG Wisconsin Package standard distribution. This powerful X11 Windows based GUI is a ‘front-end’ to the package. It provides an intuitive alternative to the UNIX command line by allowing menu-driven access to most of GCG’s programs. SeqLab makes running the Wisconsin Package much easier by providing a common editing interface from which most programs can be launched and alignments can be manipulated. SeqLab originated way before it had anything to do with GCG. Steve Smith was working on bacterial ribosomal RNA phylogenies with Walter Gilbert and Carl Woese. Steve realized the vital need for a comprehensive multiple sequence editor. Nothing existed at the time that satisfied him, so he invented one. In addition to providing the vital editing function, it also served as a menuing system to external functions such as the PHYLIP molecular evolution programs and Clustal alignments. He called it the “Genetic Data Environment: GDE” (1994). Many people were very impressed and he made it freely available. Coincidentally GCG realized the need for some sort of a ‘point-and-click’ environment for their system. They were losing lots of business, only being able to provide a command line interface. Therefore, they started trying to develop a GUI for the Wisconsin Package and released it in 1994. They called it the “Wisconsin Package Interface,” WPI for short. Few were impressed. It only provided a menu to their programs, hardly anything more than the “-check” command line option they’ve always had. So
they did a natural and very smart thing. They hired Steve Smith away from Millipore, where he had recently moved, into their company, so that he could merge his GDE with their WPI. The late 1996 offspring was SeqLab, and, thank goodness, they threw away the acronyms (GDE + WPI = SeqLab). Steve has subsequently moved on to other endeavors, but, as ‘they’ say, “The rest is history,” and once more GCG’s customers are (generally) happy.

Using the Wisconsin Package — specifying sequences and logical terms

A central ‘idea’ of the Wisconsin Package, and one of the most difficult aspects of the Package for new users to get used to, is how to tell the programs what sequences you want to work with. GCG calls this “specifying sequences” and it’s crucial to understanding the way their programs work. Once you’ve become comfortable with these concepts, so many of the frustrations commonly encountered with the Package will disappear. So, to answer the always perplexing GCG question “What sequence(s)? . . . .” the four ways of specifying sequences, in order of increasing power and complexity follow:

1. The sequence is in a local GCG format single sequence file in your UNIX account. This sequence file can be anywhere in your account as long as you supply an appropriate path so that the program can find the file. The sequence file can have any name but it is best to use extensions that tell you what type of molecule it is, e.g. “.seq” and “.pep” (e.g. “my.pep” or “~/subdir/my.seq”). Use the program reformat to convert ‘raw’ text format files to GCG format. Several GCG From and To programs are also available for specific data format conversions, and SeqLab’s Editor Mode can directly “import” native GenBank and ABI style trace format files without the need to reformat.

   This is a small example of ‘raw’ GCG single sequence format.
   Always put some documentation on top, so in the future you can figure out what it is you’re dealing with! Two periods always separate that documentation from the actual data.

   ..

   ACTGACGTCAACATACTGGGACTGAGATTTACTCGAGTTATACAAGTATACAGATTTAATAGCATGCGATCCCATGGGA

   Next, the clean GCG format single sequence file, after reformat:

   This is a small example of GCG single sequence format.
   Always put some documentation on top, so in the future you can figure out what it is you’re dealing with! The line with the two periods is converted to the checksum line.

   example.seq  Length: 77  July 21, 1999 09:30  Type: N  Check: 4099  ..

   1  ACTGACGTCA CATACTGGGA ACTGAGATTTACTCGAGTTATACAAGTATACAGATTTAATAGCATGCGATCCCATGGGA

2. The sequence is in a local GCG database in which case you ‘point’ to it by using any of the GCG database logical names. These names make sense and are either the name of the database or an abbreviation thereof. GenBank division logical names can be used for nucleotide databases, such as Bacterial. But beware, most of these divisions are based on historical taxonomy; Archaea is in Bacterial and Fungi is in Plant. A colon, (:) always sets the logical name apart from either an accession code or a
proper identifier name or a wildcard expression, and unlike the rest of UNIX, the whole thing is case insensitive. Several examples follow: GenBank:HSSRY, gb:X53772, SwissProt:SRY_Human, and uni:Q05066 all refer to SRY in *Homo sapiens*, the gene in DNA in the first two, the protein product in the last two. If you know that the database uses consistent naming conventions, then you can use a wildcard to specify all of a particular type of sequence. This works particularly well in SwissProt because of its consistent naming conventions; e.g. SW:SRY_* specifies all of the SRY sequences in SwissProt, SW:*_Human specifies all of the *Homo sapiens* sequences in SwissProt.

Because all the sequences are available in the local GCG databases, it is seldom necessary to put individual sequences in your account. In fact, the only time that you should put individual database sequences in your account is if you are somehow modifying them, such as making an alignment or engineering a vector. There’s no need to fill up your own account with data available on the same server outside your account.

**Logical terms for the Wisconsin Package at FSU** (more added periodically, especially genome databases)

### Sequence databases, nucleic acids:

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GENBANKPLUS:*</td>
<td>all of GenBank plus EST and GSS</td>
</tr>
<tr>
<td>GB:*</td>
<td>all of GenBank plus EST and GSS</td>
</tr>
<tr>
<td>GENBANK:*</td>
<td>all of GenBank except EST and GSS</td>
</tr>
<tr>
<td>GB:*</td>
<td>all of GenBank except EST and GSS</td>
</tr>
<tr>
<td>BA:*</td>
<td>GenBank bacterial</td>
</tr>
<tr>
<td>BACTERIAL:*</td>
<td>GenBank bacterial</td>
</tr>
<tr>
<td>EST:*</td>
<td>GenBank EST (Expressed Sequence Tags)</td>
</tr>
<tr>
<td>GSS:*</td>
<td>GenBank GSS (Genome Survey Sequences)</td>
</tr>
<tr>
<td>HTC:*</td>
<td>GenBank High Throughput cDNA</td>
</tr>
<tr>
<td>HTG:*</td>
<td>GenBank High Throughput Genomic</td>
</tr>
<tr>
<td>IN:*</td>
<td>GenBank invertebrate</td>
</tr>
<tr>
<td>INVERTEBRATE:*</td>
<td>GenBank invertebrate</td>
</tr>
<tr>
<td>OM:*</td>
<td>GenBank other mammalian</td>
</tr>
<tr>
<td>OTHERMAMMAL:*</td>
<td>GenBank other mammalian</td>
</tr>
<tr>
<td>OV:*</td>
<td>GenBank other vertebrate</td>
</tr>
<tr>
<td>OTHERVERTEBRATE:*</td>
<td>GenBank other vertebrate</td>
</tr>
<tr>
<td>PAT:*</td>
<td>GenBank patent</td>
</tr>
<tr>
<td>PATENT:*</td>
<td>GenBank patent</td>
</tr>
<tr>
<td>PH:*</td>
<td>GenBank phage</td>
</tr>
<tr>
<td>PHAGE:*</td>
<td>GenBank phage</td>
</tr>
<tr>
<td>PL:*</td>
<td>GenBank plant</td>
</tr>
<tr>
<td>PLANT:*</td>
<td>GenBank plant</td>
</tr>
<tr>
<td>PR:*</td>
<td>GenBank primate</td>
</tr>
<tr>
<td>PRIMATE:*</td>
<td>GenBank primate</td>
</tr>
<tr>
<td>RO:*</td>
<td>GenBank rodent</td>
</tr>
<tr>
<td>RODENT:*</td>
<td>GenBank rodent</td>
</tr>
<tr>
<td>STS:*</td>
<td>GenBank (sequence tagged sites)</td>
</tr>
<tr>
<td>SY:*</td>
<td>GenBank synthetic</td>
</tr>
<tr>
<td>SYNTHETIC:*</td>
<td>GenBank synthetic</td>
</tr>
<tr>
<td>TAGS:*</td>
<td>GenBank EST, HTC, and GSS</td>
</tr>
<tr>
<td>UN:*</td>
<td>GenBank unannotated</td>
</tr>
<tr>
<td>UNANNOTATED:*</td>
<td>GenBank unannotated</td>
</tr>
<tr>
<td>VI:*</td>
<td>GenBank viral subdivision</td>
</tr>
<tr>
<td>VIRAL:*</td>
<td>GenBank viral subdivision</td>
</tr>
<tr>
<td>REFSEQNUC:*</td>
<td>NCBI RefSeq transcriptomes</td>
</tr>
<tr>
<td>RS_RNA:*</td>
<td>NCBI RefSeq transcriptomes</td>
</tr>
</tbody>
</table>

### Genome sequence databases, nucleic acids

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOMO:*</td>
<td>NCBI human RefSeq working draft</td>
</tr>
<tr>
<td>PAN:*</td>
<td>NCBI chimpanzee RefSeq working draft</td>
</tr>
<tr>
<td>DANO*:</td>
<td>Sanger Zebrafish assembly</td>
</tr>
<tr>
<td>CELEGANS:*</td>
<td>NCBI nematidode RefSeq assembly</td>
</tr>
</tbody>
</table>

### Sequence databases, amino acids:

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIPROT:*</td>
<td>all of Swiss-Pro and all of SPTREMBL</td>
</tr>
<tr>
<td>UNI:*</td>
<td>all of Swiss-Pro and all of SPTREMBL</td>
</tr>
<tr>
<td>SWISSPROPLUS:*</td>
<td>all of Swiss-Pro and all of SPTREMBL</td>
</tr>
<tr>
<td>SWP:*</td>
<td>all of Swiss-Pro and all of SPTREMBL</td>
</tr>
<tr>
<td>SWISSPRO:*</td>
<td>all of Swiss-Pro (fully annotated)</td>
</tr>
<tr>
<td>SWISS:*</td>
<td>all of Swiss-Pro (fully annotated)</td>
</tr>
<tr>
<td>SW:*</td>
<td>all of Swiss-Pro (fully annotated)</td>
</tr>
<tr>
<td>SPTREMBL:*</td>
<td>Swiss-Pro preliminary EMBL translations</td>
</tr>
<tr>
<td>SPT:*</td>
<td>Swiss-Pro preliminary EMBL translations</td>
</tr>
<tr>
<td>GENPEPT:*</td>
<td>all of GenBank's CDS translations</td>
</tr>
<tr>
<td>GP:*</td>
<td>all of GenBank's CDS translations</td>
</tr>
<tr>
<td>REFSEQPROT:*</td>
<td>NCBI RefSeq proteomes</td>
</tr>
<tr>
<td>RS_PROT:*</td>
<td>NCBI RefSeq proteomes</td>
</tr>
</tbody>
</table>
3. The sequence is in a GCG format multiple sequence file, either an MSF (multiple sequence format) file or an RSF (rich sequence format) file. The difference is that MSF files contain only the sequence names and sequence characters, whereas RSF files contain sequence names and data, plus sequence annotation; i.e. they are “richer.” As in GCG single sequence format, it is always best to retain the suggested GCG extensions, msf or rsf, in order for you to easily recognize what type of file they are without having to look, though it is not required and they could just as well be named Joe.Blow. To specify sequences contained in a GCG multiple sequence file, supply the file name followed by a pair of braces, “{},” containing the desired sequence specification. For example, to specify all of the sequences in an alignment of elongation $1\alpha$ and Tu factors, you could use a naming system like the following: ef1a-tu.msf(*). Furthermore, one can point to individual members of the alignment or subgroups by specifying their name within the braces, e.g. EF1a-Tu.rs{eftu_ecoli} to point just to the $E\ coli$ sequence or EF1a-Tu.rs{eftu_*} to point at all of the EfTu’s as long as you use a sequence naming convention that retains this convention.

4. Finally, the most powerful method of specifying sequences, and the ‘way’ that SeqLab works, is in a GCG “list” file. This file can have any name though it is convenient to use the GCG extension “.list” to help identify them in your directory. It is merely a list of any other sequence specifications and can even contain other list files within it. List files can be created by hand with an editor and they are produced by many of the GCG programs, such as all the search programs. This is how the output from one program can become input to another. The convention to use a GCG list file in a program is to precede it with an @ sign (@). Furthermore, you can supply attribute information within list files to specify something special about the sequence. This is especially helpful with length attributes that can restrict an analysis to specific portions of a sequence and are shown in the example below:

```bash
!!SEQUENCE_LIST 1.0
An example GCG list file of many elongation $1\alpha$ and Tu factors follows. As with all GCG data files, two periods separate documentation from data. ...

my-special.pep begin:24 end:134
SwissProt:EfTu_Ecoli
Ef1a-Tu.msf(*)
/usr/accounts/test/another.rsf{ef1a_*}
@another.list
```

**Week 4 tutorial: Mendel, the Wisconsin Package, SeqLab, and finding similarity.**

Remember, do the things that are in **bold type**.

Activate and log on to the computing workstation you are sitting at. On the SCS Classroom computers, or any other UNIX or Linux machine (or Mac OS X with X11 installed), launch an X enabled terminal window with the
appropriate icon. This is named “Terminal” and will be under the RedHat menu’s “System Tools” menu on the SCS Classroom machines.

Issue the following command in your new terminal window (do not type the system prompt, “$”):

```
$ ssh -X user@mendel.scs.fsu.edu
```

Replace “user” with your account name on Mendel. I’ve built these to be the same as your SCS Classroom account name. Note the “-X” is capitalized; this option is necessary to allow ‘X tunneling’ and set up your X environment. This is the only encrypted, secure way to make X connections, and is required by Mendel, if you want to use any resources that require X windows. You’ll be asked for your Mendel password. I’ve also set your Mendel password to be exactly the same as your Classroom account password. Passwords are not displayed on the screen as you type them. However, realize that this user name and password is actually separate from the one you use to get on the SCS Classroom computers, and all of your Mendel files are not shared by your Classroom account. You’ll have to use scp to transfer them back and forth between the two systems.

On pre-OS X Macs and all MS Windows machines find and use the appropriate icons to launch X windowing and ssh, either on the desktop or in menus. Note that you’ll need X11 tunnelling to be activated in the ssh application, and that you’ll need some type of X11 windows emulator, such as Xwin32 or MacX, to be installed on these machines. After ssh opens, use the appropriate menu command to connect to mendel.scs.fsu.edu.

Further details of X on Mendel beyond these key concepts will not be covered. There are too many variables depending on your local machine. If this isn’t enough, ask me for further assistance at steyet@bio.fsu.edu.

Regardless of the ssh method used to launch and connect to Mendel, you should now have an interactive command line terminal session running on Mendel in a separate window on your local machine’s desktop. Mendel’s OS checked your username and password, ran your default shell program, and any startup scripts you might have, and then it returned the system prompt. The shell program is your interface to the UNIX OS. It interprets and executes the commands that you type. Common UNIX shells include bash, Korn, the C shell, and a popular C shell derivative that Mendel users run by default called tcsh. Tcsh, like bash, enables command history recall using the keyboard arrow keys, accepts tab word completion, and allows command line editing. Upon logging in, you end up in your ‘home directory,’ that portion of Mendel’s hard drive disk space reserved just for you. You should see a screen trace similar to the following upon logging in:

```
Welcome to GCG
Version 11.1.2-UNIX
Installed on linux
Copyright (c) 1982 – 2006, Accelrys Inc.
All rights reserved.
Published research assisted by this software should cite:
GCG Version 11.1, Accelrys Inc., San Diego, CA
```
Databases available:

- GenBank                  Release  154.0   (06/2006)
- Homo (human genome)     Build    35.1    (08/2004)
- Pan (chimp genome)      Build     1.1    (11/2004)
- Danio (zebrafish genome) Version  4.0    (07/2004)
- Celegans (worm genome)  Release  97.0    (03/2003)
- GenPept                  Release  154.0   (06/2006)
- Refseq                  Release  17.0    (05/2006)
- UniProt                 Release  8.2     (06/2006)
- PROSITE                Release  19.30   (06/2006)
- Pfam                    Release  20.00   (05/2006)
- Restriction Enzymes (REBASE)      607     (06/2006)

Technical support: e-mail Steve at stevet@bio.fsu.edu or see: http://www.accelrys.com/support/ or call GCG toll-free 1-800-756-4674 and
GCG Manual PDFs in /usr/local/gcg/doc and
Online help: > genhelp or http://www.accelrys.com/support/bio/genhelp/
or http://www.scs.fsu.edu/gcg/ with restricted off-campus access
User: gcguser Password: 4mendel

The screen trace shows the version numbers for the Wisconsin Package and of all its online databases on Mendel. The system prompt, "$", displays the user and the machine name and waits to receive a command. Other UNIX systems may use different prompts depending on how the system administrator has set it up. The login process also initializes the GCG user environment, making it ready to run all of the programs.

**Using the Wisconsin Package’s GUI SeqLab**

Now that you understand some of how the Wisconsin Package ‘thinks,’ let’s take a look at the SeqLab GUI. **X windowing is required to use GCG’s SeqLab**, so I’ll mention some of its peculiarities. **X windows are only active when the mouse cursor is in that window.** To activate X items, rather than holding a mouse button down, just <click> on it. Also, X buttons are turned on when they are pushed in and shaded. Finally, always close X windows when you are through with them to conserve system memory, but **don’t close X windows with the X-server close icon in the upper right- or left-hand window corner**, rather, always, if available, use the window’s own “File” menu “Exit” choice, or “Close,” or “Cancel,” or “OK” button.

OK, now for something completely different. Use your active Mendel X-tunneled ssh session to launch SeqLab by **issuing the command “seqlab &”** (without the quotes) in your terminal window to fire up the SeqLab interface:

```
$ seqlab &
```

The ampersand, “&,” is not necessary but really helps by launching SeqLab as a background process so that you retain control of your initial terminal window where you can issue operating system commands. This
should produce two new windows, the first an introduction with an “OK” box; check “OK.” You should now be in SeqLab’s “Main List” mode with the empty default list file, “working.list,” open in the main window.

**SeqLab Preferences and Help**

Before going any further, go to the “Options” menu in SeqLab’s main window and select “Preferences . . .” A few of the options should be checked there to insure that SeqLab runs its most intuitive manner. The defaults are usually fine, but I want you to see what’s available to change. Remember, buttons are turned on when they’re pushed in and shaded.

First notice that there are three different “Preferences” settings that can be changed: “General,” “Output,” and “Fonts,” start with “General.” The “Working Dir . . .” setting will be the directory from which SeqLab was initially launched. This is where all SeqLab’s working files will be stored. It can be changed in your accounts if desired; however, leave it as is for now. Be sure that the “Start SeqLab in:” choice has “Main List” selected and that “Close the window” is selected under the “After I push the “Run” button:” choice. Next select the “Output” Preference. Be sure “Automatically display new output” is selected. Finally, take a look at the “Fonts” menu. If dealing with very large alignments, then picking a smaller Editor font size may help you see more of your alignment on the screen at once. <Click> “OK” to accept any changes.

SeqLab’s Help system is always available. Every program window and the Main window have a “Help” button. Two other Help systems are also available on the Wisconsin Package. Issuing the commands “genhelp” or “genmanual” in a command line terminal window launches a text driven Help system, and the URL http://www.scs.fsu.edu/gcg/ links to a Web version of the Help system. The Web version will request a username and password if connecting from an off-campus location, to comply with GCG’s license restrictions. Give the username “gcguser” and the password “4mendel,” if this is the case.

**Finding and loading sequences into SeqLab**

I discussed the advantages of using a centralized, local biocomputing server earlier in the introduction and in lecture last week and won’t belabor the point here. But, I will reassert one argument. That is, perhaps the biggest advantage of local sequence databases is convenience — there is no need to download and reformat data! This can be a huge time saver, particularly if you are dealing with large datasets. So how do you access the sequence databases in the Wisconsin Package? The solution was described in the introduction: GCG uses a system of ‘logical’ terms that point to the local databases. These were listed in the “logical terms” table in the introduction. Remember that these terms all support either proper sequence identifier names or accession codes or wild cards. Therefore, a term like GB:* points to all of GenBank (less the Tags category), whereas SW:*_Human points to all of the *Homo sapiens* entries in Swiss-Prot (since Swiss-Prot uses a consistent taxon naming convention), and Uni:SRY_Human only points to the human SRY protein. Within SeqLab the “Database Browser” allows you to peruse the databases using these conventions.
From the “SeqLab Main Window” in “Mode: Main List” launch the “Database Browser” found under the “Windows” menu. You should see something like the screen snapshot opposite. Notice that all of the categories seen in the “Logical Terms” table are duplicated here in the “Database Browser.” Click on any category to place the database logical term in the “Database Specification” text box and replace the wild card asterisk with your desired proper sequence identifier name or accession code, e.g. GenBank:hssry. Pressing <return> or the “Show Matching Entries” button will then return the particular entry that you are looking for. After that you can either “View Sequence” to see the record in a text window or “Add to Main Window” to place the entry in SeqLab’s Main Window, List Mode open list file, or Editor Mode as the case may be.

But what if you don’t have a clue about your sequence’s proper identifier? How can you perform the same type of text based reference search in GCG as we did with NCBI’s Entrez system? “Close” the “Database Browser” window. Go to the “Functions” “Database Reference Searching” menu and choose “LookUp. . .” to launch the Wisconsin Package’s sequence database text scanning program, thus:

LookUp is a Sequence Retrieval System (SRS) derivative (Etzold and Argos, 1993). It is incredibly useful if you will be using the Wisconsin Package at all, because it creates an output file that can be used as an input list file to other GCG programs (remember that the list file is the most powerful way to specify sequences in the Wisconsin Package and the way that SeqLab works). We’ll use it here to find human SRY sequences.

Be sure that “Search the chosen sequence libraries,” and “Uniprot” as the library to search, are both checked in your new “LookUp” window. You can search as many libraries as you want, it just makes more sense to search protein and nucleotide databases separately. SeqLab can handle both protein and DNA sequence data simultaneously; you’re not restricted to using one or the other data types at a time. However,
subsequent programs can’t analyze both data types concurrently, so you might as well keep them separate for now. We’ll use UniProt as a good example. Under the main query section of the window, type the words and symbols “sry ! ( sox | box | fragment )” (meaning ‘SRY, but not SOX or BOX or fragment,’ separate the symbols from the words with spaces) following the category “Definition” (the same as Entrez’s “Title” field) and the word “homo” in the “Organism” category.

You need to use Boolean operator symbols to connect individual query strings in LookUp because the databases are indexed using individual words for most fields. The “Organism” field is an exception; it will accept ‘Genus species’ designations as well as any single word supported level of taxonomy, e.g. “fungi.” The Boolean operators supported by LookUp are the ampersand, “&,” meaning “AND,” the pipe symbol, “|,” to denote the logical “OR,” and the exclamation point, “!,” to specify “BUT NOT.” Other LookUp query construction rules are case insensitivity, parenthesis nesting, “*” and “?” wildcard support, and automatic wildcard extension (e.g. “transcript” will find “transcriptional” and “transcript”). Press the LookUp “Help” button to read more about its rules, if you are interested. This query should find the SRY human entry in the UniProt database. Your “LookUp” window should look similar to the screenshot on the left below:

Press the “Run” button. The program will display the results of the search; look through the output and then “Close” the window. I found eight entries in UniProt that met my specifications. A screenshot of my LookUp output file follows below:
Be careful that all of the sequences included in the output from any text based searching program are appropriate. In this case the search found one applicable entry, six mutant SRY sequences, plus one molecule that interacts with SRY, but is not SRY itself. Receptors, interacting or associated molecules, improper nomenclature, and database inconsistencies can all cause these sorts of problem. If you find inappropriate sequences upon reading the LookUp output, you can either edit the output file to remove them, or “CUT” them from the SeqLab Editor display after loading the list (see below). Another option, if you use an editor, is to comment out the undesired sequences by placing an exclamation point, “!,” in front of the unwanted lines. Exclamation points work as remark delineators in GCG data files.

Select your LookUp output file in the “SeqLab Output Manager.” This is a very important window and will contain all of the output from your current SeqLab session. Files may be displayed, printed, saved in other locations or with other names, and deleted from this window. Press the “Save As...” button and give the LookUp output file a more appropriate name. Be sure not to change the directory specification, only changing that portion after the last slash. Next press the “Add to Main List” button in the “SeqLab Output Manager” and “Close” the window afterwards. This will add the results of the LookUp search to your empty “working.list.” Go to the “File” menu next and press “Save List.” This is now a list file within a list file. Notice the ‘at’ sign (@) in front of it in your working list. Remember the ‘at’ sign is necessary in the Wisconsin Package for specifying a list file; <double-clicking> the entry ‘opens’ the file to show you both of its members; <double-click> it again to ‘close’ the file. My working list follows, on the left below:

Next, be sure that your LookUp output file is selected in the “SeqLab Main Window” and then switch “Mode:” to “Editor.” This will load the file into the SeqLab Editor where further analyses on any or all the sequences can be performed.

Exploring SeqLab

These sequences came from Mendel’s local GCG UniProt database (remember logical_term:ID). They are named by their official UniProt entry names (ID identifier), and appear in the Editor window with their amino acid residues color-coded. The nine color groups are based on a UPGMA clustering of the BLOSUM62 amino acid scoring matrix, and approximate physical property categories for the different amino acids. Expand the window to an appropriate size by ‘grabbing’ the bottom-right corner of its ‘frame’ and ‘pulling’ it out as far as desired. Your display should look similar to the graphic at the top of the following page:
Use the scroll bars to move around within the alignment. The horizontal scroll bar at the bottom allows you to move through the sequence linearly; the vertical scroll bar at the side allows you to scroll through additional members of a larger dataset. Clicking in the scroll area beyond the bottom scroll slider moves your view one screen at a time.

Another way to get sequences into SeqLab uses the “Add sequences from” “Sequence Files...” choice under the “File” menu. All GCG format compatible sequences or list files are accessible through this route. Select the file that you want from the “Files” box, and then check the “Add” and then “Close” buttons at the bottom of the window to put the desired sequences into your current list, if you’re in List Mode, or directly into the Editor, if you’re in “Editor Mode.” Furthermore, in “Editor Mode” two additional choices are available. You can “Import” sequences from GenBank or FastA format files or from ABI style binary trace files. And you can use the “File” menu “New Sequence” choice to create empty slots to hold brand new entries, either “DNA,” “RNA,” “Protein,” or “Text,” where you can either type in data or copy and paste it from a different window.

Since NHR2_HUMAN is not a SRY protein at all, it just interacts with SRY, <single-click> its name to select it and then use the “CUT” button to get rid of it. Also get rid of the mutant SRY sequences, so that the only sequence left in the Editor is SRY_HUMAN. Turning off “Invert” causes the letters to be colored and the background to go white. Turning “Wrap” on causes the sequence to wrap vertically in the display, so that you no longer have a horizontal scroll bar, only a vertical one. Use whichever combination of settings you prefer. I prefer the default non-wrapped, inverted display. My non-inverted, wrapped display follows:
Any portion of, or the entire dataset loaded, is available for analysis. Nearly all GCG programs are accessible through the “Functions” menu. “Plus” programs remove format and length restrictions but are more difficult to use. Select SRY_HUMAN and go to the “Functions” menu to see all that are available, but don’t run any just yet. You’ll have plenty of chances for that coming up. Select sequences in their entirety by <single-clicking> on their names or you select any position(s) within sequences by ‘capturing’ them with the mouse or by using the “Edit” menu “Select Range. . .” function. You can select a range of sequence names, if there’s more than one, by <shift><clicking> the top-most and bottom-most name desired, or <ctrl><click> sequence entry names to select noncontiguous entries. (A bug in the Linux version of SeqLab prevents this from working correctly. The ‘work-around’ is to use the right mouse button, not the left.)

Place your cursor anywhere within the sequence data. The “pos:” and “col:” indicators show you where the cursor is located on the sequence without including and with including gaps respectively. Press the <space bar> to insert gaps and move the sequence to the right. Periods (.) appear in the sequence to represent alignment gaps. Press <delete> to remove gaps. A very powerful manual alignment function can be thought of as the ‘abacus’ function. To do this select a region that you want to slide flanked by gaps and then press the <shift> key as you move the region with the right or left arrow key. You can slide residues greater distances by prefacing the command keystrokes with the number of spaces that you want them to slide. You are not allowed to delete sequence characters unless you change their “Protections.” This prevents you from accidentally changing your sequence data. <Click> on the padlock icon to produce a “Protections” window. Notice that the default protection allows you to modify “Gap Characters” and “Reversals” only. Check “All other characters” to allow you to “CUT” regions out of an alignment and/or delete individual residues and then <click> “OK” to close the window.

Change the “Display:” box from “Residue Coloring” to “Feature Coloring.” The display now shows a color schematic of the entry’s feature information based on its database Feature Table. Furthermore, many GCG analyses can produce RSF feature files adding to that annotation. Quickly <double-click> on one of the colored regions of the sequence (or use the “Features” choice under the “Windows” menu). This will produce a new window that describes the features located at the cursor.

Select the feature to show more details and to select that feature in its entirety. Your display should look similar to the following desktop snapshot:
All the features are fully editable through the “Edit” check box in this panel and new features can be added with several desired shapes and colors through the “Add” check box. “Close” the “Feature Editor” and “Sequence Features” windows, if you’ve opened them.

Next change “Display:” to “Graphic Features.” “Graphic Features” represents features using the same colors as above, but in a ‘cartoon’ fashion. The “1:1” scroll bar near the upper right-hand corner allows you to ‘zoom’ in or out on the sequences — move it to 2:1 and beyond and notice the difference in the display. Here’s a screenshot of SW:SRY_Human at a “2:1” zoom factor using SeqLab’s “Graphic Features” “Display.”

It’s a good idea to save sequences in the SeqLab Editor display multiple times as you work on a dataset, just in case there’s an interruption of service for any reason. Go to the “File” menu and choose “Save As.” Accept the default “.rsf” extension; give it any file name that you want. RSF (Rich Sequence Format) contains all the aligned sequence data as well as all the reference and feature annotation associated with each entry. It is “Richer” than most other multiple sequence formats and is SeqLab’s default format.

Quickly <double-click> on the entry’s name (or single <click> the “INFO” icon with the sequence entry name selected) to see its annotation. (This is the same information that you can get with the GCG command “typedata -ref” at the command line.) You can also change sequences’ names and add any documentation that you want in this window. Scroll down the annotation to the section flagged “DR.” This stands for ‘database reference’ and lists all the other databases that the sequence is associated with. Note that the first EMBL cross-reference is X53772, as seen to the right:
This accession code will work for GenBank too, but it’s not a link, so you can’t get there by clicking it. While we have the GenBank identifier, let’s practice loading it into the Editor from the online GCG databases. Go to the “File” “Add sequences” menu, “Databases . . .” choice to launch the “Database Browser.” Select “GenBank” and then delete the asterisk after the colon and replace it with the accession code. You can use copy and paste in X by selecting with the left mouse button and pasting with the middle. Press “Add to Main Window” to load the new DNA sequence into the Editor. It’s one we’ve seen before: HSSRY. We could have used that name as well. “Close” the “Sequence Information” and the “Database Browser” windows when you’re done loading the new entry. You now have both the human SRY protein and its corresponding genomic DNA sequence loaded into the SeqLab Editor, both from GCG’s onsite databases. Change your “Display” back to “Residue Coloring” and notice the different color schemes used for DNA and protein.

These two sequences are not aligned — there would have to be two gaps after every residue in the protein sequence to make them correspond to the gene’s codons. To see what this would look like quickly <double-click> anywhere within the DNA sequence (or use the “Features” choice under the “Windows” menu) to get the “Sequence Features” window. Change “Show:” from “Features at cursor” to “All features in current sequence.” Select “CDS” from the list to select the entire SRY coding region and then “Close” the “Sequence Features” window. Go to the “Edit” menu, “Translate . . .” choice. If you’ve also got the entry name selected, you’ll be asked “Do you want to use the . . .” “Selected sequences” or the “Selected regions;” specify “Selected regions.” Check “Align Translation” near the bottom of the “Translate” window and then check “OK.” A new protein sequence will appear below the DNA coding region with each amino acid residue aligned to its respective codon. Select it as well as the DNA sequence above it by dragging your mouse through both of them, or by <shift><clicking> them both. Press the “GROUP” icon above your dataset.

The “GROUP” function allows you to manipulate ‘families’ of sequences as a whole — any change in one will be propagated throughout them all, they’ll behave collectively. You can have as many groups as you want. We’ve just grouped the SRY DNA sequence to its corresponding protein translation; that’s indicated by the group number “1” associated with the entry names. Move to an area of the display where you can see this correspondence. Place your cursor anywhere within the DNA sequence and add some gaps with the <space> bar. Notice that as you move the DNA sequence its corresponding protein translation comes along:
Grouping becomes very important when dealing with large multiple sequence alignments. It makes manual fine-tuning much easier. Furthermore, grouping allows you to prepare mixed protein and DNA datasets with DNA sequences aligned to their corresponding proteins, as seen here with SRY_HUMAN and HSSRY. This is sometimes very desirable, especially if preparing datasets with very similar sequences for phylogenetic analyses. Phylogenetic analysis is then performed on the DNA rather than on the proteins. This is especially important when dealing with datasets that are quite similar since the proteins may not reflect many differences hidden in the DNA. Furthermore, many people prefer to run phylogenetic analyses on DNA rather than protein regardless of how similar they are — the multiple substitution models have a for DNA. In fact, many phylogenetic inference algorithms do not even take advantage of amino acid similarity when dealing with protein sequences; they only count identities! However, the more diverged a dataset becomes, the more random third and eventually first codon positions become, which introduces noise (error) into the analysis. Therefore, often third positions and sometimes first positions are excluded. Just like in most of computational molecular biology, one is always balancing signal against noise. Too much noise or too little signal both degrade the analysis to the point of nonsense. The logic to this paired protein and DNA alignment approach is as follows:

1) The easy case where you can align the DNA directly: If the DNA sequences can be aligned directly because they are quite similar, then merely create your DNA alignment. Next use the “Edit” menu “Translate” function and the “align translations” option to create aligned corresponding protein sequences. Select the region to translate based on the CDS reference in each DNA sequence’s annotation. Be careful of CDS entries that do not begin at position 1 — the GenBank CDS feature annotation “/codon_start=” identifies which position the translation begins within the first codon listed. You may also have to trim sequences down to just the relevant exons/gene, especially if they’re genomic. You’ll have to change their protections with the padlock icon if this is the case. Group each protein to its corresponding DNA sequence so that subsequent manipulations will keep them together.

2) The way more difficult case where you need to use the protein sequences to create the alignment because the DNA cannot be directly aligned: In this case, load the protein sequences first, create their alignment, and then load the corresponding DNA sequences. Find the DNA sequence accession codes in the annotation of the protein sequence entries. Next, translate the unaligned DNA sequences into new protein sequences with the Edit-Translate function using the “align translations” option and Group these to their corresponding DNA sequences, just as above. However, this time the DNA along with their translated sequences are not aligned as a set, just the other protein set is aligned. Also, Group all of the aligned protein dataset together, separately from the DNA/aligned translation set. Then rearrange your display to place the DNA, its aligned translation, and the original aligned protein sequence side-by-side. Now comes the manual part — unfortunately GCG does not include an automated way to achieve this in SeqLab — manually slide one set to match the other. Use the “CUT” and “PASTE” buttons to move the sequences around. When pasting realize that the “Sequence clipboard” contains complete sequence entries, whereas the “Text clipboard” only contains sequence data, amino acid residues or DNA bases as
the case may be. The translated sequence entries can be “CUT” away after they’re aligned to the rest of the set. Merge the newly aligned sequences into the existing alignment Group as you go and then start on the next one. It sounds difficult, but since you’re matching up two identical protein sequences, the DNA translation and the original aligned protein, it’s really not too bad. The Group function keeps everything together the way it should be so that you don’t lose your original alignment as you space residues apart to match them up to their respective codons. Some codons may become spaced apart in this process and will have to be adjusted afterwards. As usual, save your work often.

**Similarity searching and significance testing with the Wisconsin Package**

Now that you’ve seen a bit of how GCG works, let’s move on to ask an evolutionary question: where did the SRY gene come from? We know that it belongs to a whole class of transcription factors known as the SOX type HMG group, but what SOX gene led to the evolution of SRY in only the Mammalia lineage, and where did SOX come from? We saw some good hypotheses in the first lab of our series when we surveyed the Web for information related to the molecular biology of sex determination in humans, but let’s see what we can find with database searching techniques.

To begin let’s prepare another GCG list file; this time of all the RefSeq protein entries that are from chordates, but are not mammalian. This file will help us get directly to the base of the question without having to sort through all the mammalian SRY orthologues and paralogues. So go to the “Functions” menu “Database Reference Searching” choice, and launch “LookUp. . .” for the second time today. Switch “Search the chosen sequence libraries” from “Uniprot” to “RefSeq Protein” (a.k.a. RS_Prot or RefProt) and then type “chordata ! mammalia” in the “Organism” field. Press “Run.” This will find all of the entries in RefSeq Protein that are chordates; however, the Boolean ‘but not’ exclamation point will exclude any that are mammals. The results will quickly return; I found around 60,000 entries. “Close” the list file window and then use the “Output Manager” to save the file with a name that makes sense. Press “Add to Main List” and then “Close” the “Output Manager.”

**Traditional database searching: FastA**

The first widely used heuristic, hashing-style database searching algorithm was FastA (Pearson and Lipman, 1988; and Pearson, 1998; preceded by WordSearch, Wilbur and Lipman, 1983). This family of programs is incorporated into GCG (see the GCG Program Manual, take advantage of the Help buttons in SeqLab programs, or use the “genmanual” command for details). In spite of the fast hashing, heuristic style algorithms incorporated, they are cpu intensive, and work best when submitted as a batch or background process against current databases. All of the GCG database searches accept an automatic batch submission option from the command line, or default to background mode in SeqLab.

The FastA family includes several programs: the original FastA program compares a sequence query and database of the same type, either protein or nucleotide. FastX compares a nucleotide query against a protein
TFastA and TFastX compare a protein sequence query against all six translations of a nucleotide database. This takes advantage of the size of the DNA databases, and yet still retains the vastly increased sensitivity level of protein searches. However, TFastX is even better than that — it allows for frame shifts due to sequencing errors. These types of errors are especially prevalent in the tags databases (EST's [expressed sequence tags], GSS’s [genome survey sequences], and HTC’s [high-throughput cDNA’s]) — be warned.

We’ll be running FastA today. Select the “SRY_HUMAN” protein sequence entry name (only) in your editor display. Then go to the “Functions” “Database Sequence Searching” menu and select “FastA. . .” (not FastA+) to start the FastA program. If a “Which selection” window pops up asking if you want to use the "selected sequences" or "selected region," choose "selected sequences." A great feature of the FastA family of database search programs is you can search any valid GCG sequence set specification. You are not restricted to specific prebuilt databases. The default database to search, “Search Set. . .” “Using uniprot:*” is all of the sequences in the UniProt protein database, but we’re going to change it here to the list file we just made of all the non-mammalian chordate RefSeq proteins. Therefore, push the “Search Set. . .” button, select “uniprot:*” in the "Build FastA’s Search Set" box that pops up, and then “Remove from Search Set." Next, press the “Add Main List Selection. . .” button and then pick your new non-mammalian chordate LookUp file in the “List Chooser” window that popped up to identify your new list file. Press “Add” and then “Close” the “List Chooser” window. Then “Close” the “Build FastA’s Search Set” window. The other parameters in the main FastA window are fine at their default settings, though you may want to decrease the cutoff Expectation value, “List scores until E() reaches,” from its default of 10 to something more reasonable like 1.0 to reduce the output list size. Press the “Options. . .” button to check out the optional parameters. Scroll down the window and notice the “Show sequence alignments in the output file” button. This toggles the command line option –NoAlign off and on to suppress the pairwise alignment section. This can be helpful if you are not interested in the pairwise alignments and will produce smaller output files. Some of the other options, such as restricting your search by the database sequence length, or by date of their deposition in the database, may be handier depending on your specific situation. “Close” the “Options” window, be sure that the “FastA” program window shows “How:” “Background Job,” and then press the “Run” button.

To check on the progress of the job go to SeqLab’s “Windows” menu and choose “Job Manager.” Scroll down the list and select the “FastA” entry to be sure that it is running, and then close the window. Be sure not to submit the same job multiple times, and if you see that you have accidentally done so, use the “Job Manager” to “Stop” the given job. The FastA results will not be immediately available: go on with the rest of the tutorial now rather than waiting for it.

BLAST: Internet and local server based similarity searching

BLAST (Altschul, et al., 1990 and 1997) is a heuristic algorithm for searching sequence databases developed by the National Center for Biotechnology Information (NCBI), a division of the National Library of Medicine.
(NLM), at the National Institute of Health (NIH), the same people responsible for maintaining GenBank and for providing worldwide access to sequence analysis resources. The acronym stands for Basic Local Alignment Search Tool. The original BLAST algorithm only looked for ungapped segments; however, the current version (Altschul, et al., 1997) adds a dynamic programming step to produce gapped alignments. As with the FastA family, BLAST ranks matches statistically and provides Expectation values for each to help evaluate significance. It is very fast, almost an order of magnitude over traditional sequence database similarity searches, yet maintains the sensitivity of other methods for local similarity in protein sequences. Another advantage of BLAST is it not only shows you the best alignment for each similar sequence found (as in the pairwise alignments of FastA) but also shows the next best alignments for each up to a certain preset cutoff point. This combines some of the power of dot-matrix type analyses and the interpretative ease of traditional sequence alignments. A disadvantage of BLAST is it requires precompiled special databases and will not accept the general type of GCG sequence specification that the FastA programs will. You can fine-tune BLAST by altering its operating parameters and taking advantage of the many options available in it. But whatever you do, BLAST is not the best tool for comparing nucleotide sequences against the nucleotide database without translation, especially with short sequences. In this situation, especially with default parameters, it will only find nearly identical DNA sequences, and will not be able to locate sequences that are only somewhat similar at all. Therefore, if you are dealing with a non-protein-coding, non-translated locus and are forced to compare a DNA query against a DNA database without translation, use FastA instead of BLAST; it is the far more appropriate tool. NCBI’s BLAST tutorial, and GCG’s BLAST documentation, are both good sources of further information on the BLAST family of programs.

GCG accesses NCBI’s BLAST server with NetBLAST, a client-server system such that NCBI’s database and computers perform the analysis, not Mendel. The server gets quite busy though, so you may have to wait in a queue. It uses the same fast heuristic, statistical hashing algorithm as GCG’s local BLAST program, but runs on a very fast parallel computer system located at NCBI in Bethesda, MD, so that typical searches run in just a couple of minutes, after the waiting queue. Furthermore, the BLAST server at NCBI provides the most up to date search available because NCBI updates GenBank and GenPept every night. However, realize that NetBLAST, unlike other GCG programs, generates an output list that is not appropriate as input to other GCG analyses. NetBLAST returns files in NCBI’s own format, incompatible with GCG (but see NetFetch). For that reason I will be showing local BLAST here, though the same procedures and logic apply to NetBLAST.

GCG’s local BLAST program produces an output file in valid GCG “list file” format so that it can be fed directly to other GCG programs. To launch GCG’s local BLAST program, be sure that the “SRY_HUMAN” sequence entry name is still selected and then pick “Blast...” (not Blast+) off of the “Functions” “Database Sequence Searching” menu. The program default is to “Search a protein database” “Search Set...” “Using local uniprot” when the query is a protein. This would search the local precompiled UniProt BLAST database. However, we’re going to query a different database to help address that ‘where did SRY come from’ question. We can’t search the non-mammalian chordate file though, as we did with FastA, because BLAST requires preformatted databases, but we can search the complete genome of a non-mammalian chordate, the
Zebrafish, *Danio*, since I have built a BLAST style database for it. Therefore, switch to “Search a nucleotide database” and then press the “Search Set. . .” button and then pick “danio” there. “Close” the “Choose a Search Set for Blast” window.

As in the FastA programs, decreasing the Expectation cutoff value will decrease the output list size. Go ahead and set “Ignore hits that might occur more than how many times by chance alone” to “1.0” as we did with FastA. Push the “Options. . .” button to get a chance to review them. Notice that “Filter input sequences for complex / repeat regions” is turned on by default. This activates a very powerful option that should generally be taken advantage of. This option, the –Filter=xs switch, causes troublesome repeat and low information portions of the query sequence to be ignored in the search. This screening of low complexity sequences from your query minimizes search confusion due to random noise. The programs that perform this function, Xnu and Seg, are available separately in GCG for prescreening your sequences prior to other analyses besides BLAST. “Close” the “Options” window and then press the “Run” button in BLAST’s main window. This run will take a while; go ahead with the next part of tutorial.

(Also notice the “Display alignments from how many sequences” Options button; this generates the –Align command line option, useful for suppressing unneeded segment alignments and hence reduce the size of the output file. The standard output file is very long because BLAST in SeqLab automatically aligns the best 250 matches unless you reduce this parameter. However, unlike the FastA programs, beginning and ending attributes are only saved in the BLAST output list file from those segment alignments that you request.)

The BLAST and FastA search results will automatically display once they are done. This even happens if you log off Mendel and come back later. They will continue to run and then can be seen and used at some later point. Use the “Output Manager” located under SeqLab’s “Windows” menu when they are done to display and save your output files with appropriate names.

**What Next? Comparisons, interpretations, and further analyses**

I’ll show abridged versions of the two database search output files next. Naturally, the topmost ‘hits’ will be clear-cut SRY homologues; it’s the ones below the expected hits that may prove interesting for this section of the tutorial. Especially pay attention to the E values in the output files. As explained in the Introduction, these are the likelihoods (expectations) that the observed matches could be due to chance. The smaller the E value is, the more significant the match is. Here’s my abridged example FastA search output file against all the non-mammalian chordates in the RefSeq Protein database:

```plaintext
!!SEQUENCE_LIST 1.0
(Peptide) FASTA of: sry_human from: 1 to: 204 September 22, 2006 16:30
ID   SRY_HUMAN      STANDARD;      PRT;   204 AA.

DE   Sex-determining region Y protein (Testis-determining factor). . . .
TO: @nonmammal-chordates.refprot.list Sequences: 58,691 Symbols: 27,981,254
Word Size: 2
```
Databases searched:
REFSEQ_PROT, Release 17.0, Released on 21May2006, Formatted on 8Jun2006

Scoring matrix: GenRunData:blosum50.cmp
Variable pamfactor used
Gap creation penalty: 12  Gap extension penalty: 2

Histogram Key:
Each histogram symbol represents 92 search set sequences
Each inset symbol represents 3 search set sequences
z-scores computed from opt scores

<table>
<thead>
<tr>
<th>z-score</th>
<th>obs</th>
<th>exp</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; 20</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>22</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>24</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>26</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>30</td>
<td>39</td>
<td>81</td>
</tr>
<tr>
<td>32</td>
<td>307</td>
<td>312</td>
</tr>
<tr>
<td>34</td>
<td>950</td>
<td>846</td>
</tr>
<tr>
<td>36</td>
<td>1960</td>
<td>1738</td>
</tr>
<tr>
<td>38</td>
<td>3824</td>
<td>2872</td>
</tr>
<tr>
<td>40</td>
<td>4219</td>
<td>4006</td>
</tr>
<tr>
<td>42</td>
<td>4875</td>
<td>4897</td>
</tr>
<tr>
<td>44</td>
<td>5496</td>
<td>5402</td>
</tr>
<tr>
<td>46</td>
<td>5209</td>
<td>5502</td>
</tr>
<tr>
<td>48</td>
<td>5116</td>
<td>5267</td>
</tr>
<tr>
<td>50</td>
<td>4678</td>
<td>4807</td>
</tr>
<tr>
<td>52</td>
<td>3943</td>
<td>4226</td>
</tr>
<tr>
<td>54</td>
<td>3430</td>
<td>3610</td>
</tr>
<tr>
<td>56</td>
<td>2846</td>
<td>3015</td>
</tr>
<tr>
<td>58</td>
<td>2467</td>
<td>2475</td>
</tr>
<tr>
<td>60</td>
<td>1992</td>
<td>2005</td>
</tr>
<tr>
<td>62</td>
<td>1547</td>
<td>1608</td>
</tr>
<tr>
<td>64</td>
<td>1275</td>
<td>1278</td>
</tr>
<tr>
<td>66</td>
<td>952</td>
<td>1010</td>
</tr>
<tr>
<td>68</td>
<td>791</td>
<td>795</td>
</tr>
<tr>
<td>70</td>
<td>604</td>
<td>623</td>
</tr>
<tr>
<td>72</td>
<td>479</td>
<td>487</td>
</tr>
<tr>
<td>74</td>
<td>346</td>
<td>379</td>
</tr>
<tr>
<td>76</td>
<td>231</td>
<td>295</td>
</tr>
<tr>
<td>78</td>
<td>217</td>
<td>230</td>
</tr>
<tr>
<td>80</td>
<td>184</td>
<td>178</td>
</tr>
<tr>
<td>82</td>
<td>150</td>
<td>136</td>
</tr>
<tr>
<td>84</td>
<td>102</td>
<td>108</td>
</tr>
<tr>
<td>86</td>
<td>94</td>
<td>84</td>
</tr>
<tr>
<td>88</td>
<td>78</td>
<td>65</td>
</tr>
<tr>
<td>90</td>
<td>39</td>
<td>50</td>
</tr>
<tr>
<td>92</td>
<td>30</td>
<td>39</td>
</tr>
<tr>
<td>94</td>
<td>28</td>
<td>30</td>
</tr>
<tr>
<td>96</td>
<td>17</td>
<td>23</td>
</tr>
<tr>
<td>98</td>
<td>16</td>
<td>18</td>
</tr>
<tr>
<td>100</td>
<td>6</td>
<td>14</td>
</tr>
<tr>
<td>102</td>
<td>5</td>
<td>11</td>
</tr>
<tr>
<td>104</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>106</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>108</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>110</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>112</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>114</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>116</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>118</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>&gt;120</td>
<td>120</td>
<td>1</td>
</tr>
</tbody>
</table>

Joining threshold: 36, opt. threshold: 24, opt. width:  16, reg.-scaled
The best scores are: init1 initn opt z-sc E(58572)...

RS_PROT:XP_684859  Begin: 24  End: 182
! PREDICTED: similar to Sox3 [Danio r...  411 411 477 528.2 7.6e-23
RS_PROT:NP_001001811  Begin: 24  End: 182
! SRY-box containing gene 3 [Danio re...  411 411 477 528.2 7.6e-23
RS_PROT:NP_001032751  Begin: 24  End: 117
! transcription factor sox1b [Danio r...  424 446 439 486.2 1.7e-20

RS_PROT:NP_001009888  Begin: 6  End: 88
! SRY (sex determining region Y)-box ...  390 390 410 456.6 7.4e-19

RS_PROT:NP_990092  Begin: 6  End: 95
! SRY (sex determining region Y)

RS_PROT:NP_989640  Begin: 93  End: 219
! SRY (sex determining region Y)

RS_PROT:NP_571950  Begin: 99  End: 238
! SRY-box containing gene 10 [Danio r...  322 322 325 360.4 1.7e-13

RS_PROT:XP_701062  Begin: 82  End: 170
! PREDICTED: similar to HMG box trans...  320 320 328 365.4 8.9e-14

RS_PROT:XP_683765  Begin: 437  End: 589
! PREDICTED: similar to rtSox23 [Dani...  289 289 304 336.0 3.9e-12

RS_PROT:NP_989226  Begin: 96  End: 175
! hypothetical protein LOC394834 [Xen...   117   117   132   155.9 0.042

RS_PROT:XP_688200  Begin: 420  End: 497
! PREDICTED: similar to high-mobility...   74    74  131 148.3 0.11

RS_PROT:NP_001004674  Begin: 86  End: 165
! hypothetical protein LOC447936 [Dan...  133 133 144 168.8 0.0079

RS_PROT:XP_416622  Begin: 377  End: 451
! PREDICTED: similar to HMG-BOX trans...  133 133 153 168.7 0.008

RS_PROT:NP_990817  Begin: 83  End: 170
! non-histone chromosomal protein [Ga...  123 123 142 166.8 0.01

RS_PROT:XP_686304  Begin: 95  End: 154
! PREDICTED: similar to high mobility...  136 136 141 166.0 0.011

RS_PROT:NP_989941  Begin: 57  End: 136
! hypothetical protein LOC394538 [Xen...  81 81 119 139.3 0.35

RS_PROT:XP_421454  Begin: 221  End: 300
! PREDICTED: similar to HMG domain pr...  71 71 121 139.0 0.36

RS_PROT:XP_693421  Begin: 32  End: 113
! PREDICTED: similar to high-mobility...

RS_PROT:NP_001005436  Begin: 68  End: 147
! SWI/SNF related, matrix associated,... 73 99 117 135.3 0.58
RS_PROT:NP_001006335  Begin: 68  End: 147
! SWI/SNF related, matrix associated,... 73 73 116 134.5 0.64
RS_PROT:NP_958455  Begin: 66  End: 145
! SWI/SNF related, matrix associated,... 98 98 116 134.4 0.65
RS_PROT:NP_001006760  Begin: 102  End: 169
! high-mobility group 20A [Xenopus tr... 61 61 113 132.3 0.85

|RS_PROT:NP_990092|
LOCUS       NP_990092                240 aa            linear  VRT 08-JUL-2005
DEFINITION  SRY (sex determining region Y)-box 14 [Gallus gallus].
ACCESSION   NP_990092
VERSION     NP_990092.1  GI:45382127
DBSOURCE    REFSEQ: accession NM_204761.1
KEYWORDS    . . . .

SCORES   Init1: 376   Initn: 376   Opt: 402   z-score: 448.1 E(): 2.2e-18
>>RS_PROT:NP_990092  (240 aa)
initn: 376 init1: 376 opt: 402 Z-score: 448.1 expect(): 2.2e-18
Smith-Waterman score: 402; 62.2% identity in 90 aa overlap
(58-146:6-230)

 העובדה היא...
The FastA output file is a GCG list file with sequence attribute information that can serve as input to other GCG programs. These include beginning and ending points for the observed similarity (and reverse complementary strand attributes where necessary with DNA). The file shows a histogram of the score distribution, then a sorted list of the top scores, and finally, if alignments are not suppressed with the –NoAlign option, a specified number, or the default forty, BestFit style alignments from the score list. These pairwise alignments show gaps as hyphens, “-“ identities as vertical bars, “|,” and conservative replacement positions as colons, “:” (and forward slashes, “/,” indicate frame shifts with TFastX alignments). The score distribution histogram can be helpful to get a feeling for the statistical significance of the search and in ascertaining whether you ran your search list large enough. For the search statistics to be valid, the expected extreme value distribution, as indicated by the line of asterisks, should approximate the actual distribution, as shown by the equal signs. Normally you want your list size big enough to include some of the population of random low scores to help you ascertain the significance of the alignments. The inset shows a zoom-in blowup of significant scores — these are the best alignments found by the program.

The Expectation function, \( E() \), is by far the most important column. It is very similar to the E value in BLAST reports and describes the number of search set sequences that would be needed to obtain a \( z \)-score greater than or equal to the \( z \)-score obtained in any particular search purely by chance; in-other-words, just like with BLAST \( E \)-values, the smaller the number, the better. As a conservative rule-of-thumb for a most protein database searches, as long as optimization is not turned off, \( E() \) scores of much less than 0.01 are probably homologous, and scores from 0.01 to 1 may be homologous, whereas scores between 1 to 10 are most likely not homologous, although these guidelines can be skewed by compositional biases. You will often be able to see a demarcation where the Expectation values drop off between the significant hits and background noise. In the SRY case I expected to see the best E values for other SRY proteins, if there are any in the database queried, and then another bracket of very good values for the other SOX proteins, and then another category of just decent scores for other HMG proteins, and finally, if I ran the list big enough, a category of not so good scores that reflect background noise.

Another thing to notice in the output is that the entries are sorted by a \( z \) score parameter based on a normalization of the ‘opt’ scores and their distribution from the rest of the database. This \( z \)-score is a bit different than the more traditional Monte Carlo style distribution Z score that I described in the introduction. Here it is calculated from a simple linear regression against the natural log of the search set sequence length. (See William R. Pearson, Protein Science 4; 1145-1160 [1995] for an explanation of how this \( z \)-score is calculated.) Either type help describe the statistical significance of an alignment.

Here is my abridged BLASTP output from the search of the Danio genome:

```
!!SEQUENCE_LIST 1.0
TBLASTN 2.2.10 [Oct-19-2004]
```

Query= UNIPROT_SPROT:SRY_HUMAN  
(204 letters)  

Database: danio  
32,622 sequences; 1,673,938,027 total letters  
Searching. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . done  

<table>
<thead>
<tr>
<th>Score</th>
<th>E Value</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>143</td>
<td>4e-33</td>
<td></td>
</tr>
<tr>
<td>143</td>
<td>4e-33</td>
<td></td>
</tr>
<tr>
<td>133</td>
<td>4e-30</td>
<td></td>
</tr>
<tr>
<td>132</td>
<td>9e-30</td>
<td></td>
</tr>
<tr>
<td>132</td>
<td>9e-30</td>
<td></td>
</tr>
<tr>
<td>127</td>
<td>2e-28</td>
<td></td>
</tr>
<tr>
<td>126</td>
<td>5e-28</td>
<td></td>
</tr>
<tr>
<td>98</td>
<td>1e-19</td>
<td></td>
</tr>
<tr>
<td>98</td>
<td>1e-19</td>
<td></td>
</tr>
<tr>
<td>74</td>
<td>4e-12</td>
<td></td>
</tr>
<tr>
<td>62</td>
<td>1e-08</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>2e-07</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>3e-04</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>0.002</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>0.002</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>0.002</td>
<td></td>
</tr>
</tbody>
</table>
Breakup of NW_644032 from : 300001 to : ...
End of List

>danio:NW_633677_w1 Breakup of NW_633677 from : 1 to : 110000
Length = 110000
Score =  41.6 bits (96), Expect = 0.015
Identities = 19/50 (38%), Positives = 29/50 (58%)
Frame = +2

Query: 57     QDVRKRPMAFIVWSDQRKRMALENPRMRNSEEKSQQLGYQWKM 106
++D++RPMAF+++S+ R + +P N +S SK LG W L EK
Sbjct: 102852 KDHRRPKMAFMIFSRRHVLQHPRQNDRTVSKILGEWYALGPKAE 102901

>danio:NW_634150_w1000001 Breakup of NW_634150 from : 1000001 to : 1110000
Length = 110000
Score = 40.4 bits (93), Expect = 0.034
Identities = 17/58 (29%), Positives = 34/58 (58%)
Frame = -1

Query: 65  NAFIVWSRDQRRKMALENPRMRNSEISKQLGYQWKMLTEAEKWPFFQEAQKLQAMHRE 122
   + F ++ QR + +NP + +++K+LG W L+++EK PF A KL+ +++
Sbjct: 32219 SGFFLFCAKQRPIIKAQNPSLGIGDVAKKLGGMWNNLSDEKQFLSNADKLDRYQK 32046

Database: danio
   Posted date:  Jan 21, 2007  4:15 PM
   Number of letters in database: 1,673,938,027
   Number of sequences in database:  32,622

Lambda   K      H
0.317    0.127    0.390

Gapped Lambda   K      H
0.267   0.0410    0.140

Matrix: BLOSUM62
Gap Penalties: Existence: 11, Extension: 1
Number of Hits to DB: 413,006,215
Number of Sequences: 32622
Number of extensions: 6051226
Number of successful extensions: 24681
Number of sequences better than 1.0: 42
Number of HSP's better than 1.0 without gapping: 1912
Number of HSP's successfully gapped in prelim test: 713
Number of HSP's that attempted gapping in prelim test: 22152
Number of HSP's gapped (non-prelim): 4545
length of query: 204
length of database: 557,979,342
effective HSP length: 121
effective length of query: 83
effective length of database: 554,032,080
effective search space: 4598466240
effective search space used: 4598466240
frameshift window, decay const: 40, 0.1
T: 13
A: 40
X1: 16 ( 7.3 bits)
X2: 38 (14.6 bits)
X3: 64 (24.7 bits)
S1: 41 (21.7 bits)
S2: 80 (35.4 bits)

The output is a perfectly suitable GCG list file, complete with beginning and ending attributes for those alignments specified (and complementary strand attributes when necessary with DNA). Again, the score to pay attention to is the Expectation value — it is easier to interpret than the information bits score in the adjacent column. The pairwise alignments requested have identity positions highlighted by amino acid single letter symbols and similarity positions identified by plus signs. BLAST can find more than one segment of alignment on the same sequence entry. This can be particularly helpful in those cases where the query, or the database entry, is from genomic DNA with several dispersed exons, or where the query is modular with multiple separate domains.

**Interpreting database search results — what is significant? Beyond mere expectations . . .**

We should know how our sequence aligns to itself and other close homologues; we know these alignments are significant. These types of alignments don't cause anybody any problems; they're obvious. Therefore, we will use a sequence where the similarity isn't so obvious for this section of the tutorial. We are interested
here in how a not so similar, as Russell Doolittle calls it a ‘twilight zone,’ sequence, aligns to SRY, and the significance of that alignment. Find an interesting sequence, that has a mediocre, ‘twilight zone’ FastA score from your search result, if it has finished, or use my example, or use a different example from my FastA screen trace. Don’t use one of the Danio sequences from the BLAST search; it’d be too difficult to work with at this point. I chose an entry with an E value of $7.5 \times 10^{-2}$. There are no ‘correct’ answers here; we just want to see an interesting comparison. Write down your choice. We will experiment with another method for analyzing the significance of this sequence’s similarity to SRY. My example’s relevant lines from the FastA output follow below. It turns out to be a high-mobility protein, complete with a HMG box, from Xenopus:

```
RS_PROT:NP_001025555  Begin: 361  End: 420
! high-mobility group protein 2-like ...  75  75  133  151.2  0.075
```

Load your sequence choice into SeqLab by going to the “File” “Add sequences from” “Databases…” menu. Use the same procedure as we used earlier — the “Database Specification:” box of the “SeqLab Database Browser,” this time to specify the RefSeq Protein database and your chosen sequence, and then press the “Add to Main Window” button. “Close” the browser box after adding the sequence into the Editor. 

**Dot matrix methods: Compare and DotPlot — GCG’s implementation**

Dot matrix analysis is one of the few ways to identify other elements beyond what dynamic programming algorithms show to be similar between two sequences. GCG implements dot matrix methods with two programs. Compare generates the data that serves as input to DotPlot, which actually draws the matrix. Compare SRY_Human to its ‘twilight zone,’ neighbor (as described above) using these methods. (In general, put the longer sequence along the horizontal axis of the final dotplot by having it first in the SeqLab display. “CUT” and “PASTE” the sequences to arrange — the cut sequence will paste right below any sequence entry name that you have selected. Dotplots just look better that way, though it is not necessary.)

Start the program by selecting the “SRY_HUMAN” entry and your new ‘twilight zone’ entry in the SeqLab main Editor display. Use `<ctrl>`+`<right mouse click>` to select non-contiguous entries. Next go to the “Functions” menu and select “Pairwise Comparison” “Compare…” to produce a Compare program window. Notice that “DotPlot…” is checked by default so that the output from Compare will automatically be passed to DotPlot. The graphic will be drawn after the “Run” button is punched.

This will run the program at the GCG protein stringency default of 10 points within a window of 30 residues. That means wherever the average of BLOSUM62 match scores within the window is equal to or exceeds 10, a point will be drawn at the middle of the window, then the window is slid over one position at which point the process is repeated. Just as in all windowing algorithms, you want to use a window size approximately the same size as the feature that you’re trying to recognize. You can leave the window at its default setting of 30 for these runs, unless one of your sequences is so short that size of window would cover much of the
sequence, in which case you should reduce the window size appropriately. If necessary clean up the graph by rerunning the program increasing the stringency of the comparisons until the number of points generated is of the same order of magnitude as the length of the longest sequence being compared. This and changing the window size are both done through the “Options” menu. Remember that subsequent runs after the first can be launched from the “Windows” menu ‘shortcut’ listing.

Below, in my example, the *Homo sapiens* SRY protein compared to the *Xenopus* high-mobility protein, I found the default stringency of 10 points within a window of 30 residues resulted in 465 points — close enough to both sequence’s length and of the correct magnitude. When run at this default stringency the dotplot looks like the graphic shown below:

Notice that running the comparison at an appropriate stringency, in this case the default, produces a relatively clean plot with little confusing noise. There is a strong diagonal near the end of the *Xenopus* protein, from around position 360 through position 420, that clearly shows an alignment across a big chunk of the SRY sequence, starting around residue 50 and running through about residue 130. This corresponds to the Begin: 361 End: 420 attributes listed by FastA. There are also scattered regions of direct repeat through out the dotplot. Sometimes interpreting a dotplot can be a major accomplishment in itself — just remember that diagonals are regions of similarity between the two sequences and that any diagonal off the main center line is indicative of regions that do not correspond in linear placement between the two sequences yet are still similar. Direct repeats always appear as columns or rows of multiple diagonals in a dotplot. Dot matrix techniques are about the best available for recognizing repeats in biological sequences. If you used a different sequence than I did, take some notes about where the similarities lie on both sequences and see how it compares to what FastA identified.

The pairwise dynamic programming alignment algorithms. Use the right one for the right job — Gap, BestFit, and FrameAlign.

You need to understand the difference between these algorithms! Gap is a ‘global’ alignment scheme and BestFit is a ‘local’ algorithm, both between two sequences of the same type, whereas FrameAlign can be global or local depending on the options that you set but it always aligns DNA to protein. Using one versus
the other implies that you are looking for distinctly different relationships. Know what they mean. If you already know that the full length of two sequences of the same type are pretty close, that they probably belong to the same family, then Gap is the program for you. It will align the full length of both sequences. If you only suspect an area of one is similar to an area of another, then you should use BestFit. To force BestFit to be even more local, you can specify a more stringent alternative symbol comparison table, such as the PAM120 or BLOSUM100 matrices. If you suspect that a DNA sequencing error is affecting the alignment, then FrameAlign is the program to use. All three programs can generate ‘gapped’ output files in standard GCG sequence formats as an option; this can be handy as direct input to other GCG routines — particularly multiple sequence analysis programs.

BestFit and Gap both allow you to estimate significance with a Monte Carlo –Randomizations=100 option, as described in the Introduction. Let’s see it with BestFit to illustrate the previous SRY comparison. This approach works best when applied to local areas where you already know some similarity exists, and you wish to further test that similarity. Therefore, it should be restricted to just those regions of similarity previously identified, otherwise you are just throwing noise into the analysis, but we’re not going to worry about that detail today. Also remember that dotplots show us all the regions that are similar, whereas dynamic programming only gives us one optimal solution.

Insure that the pair of sequences to analyze are still selected and return to the “Functions” “Pairwise Comparison” menu, only this time choose “BestFit…” Press the “Options” button there to take advantage of –Randomizations. Don’t mess with the top several options, but do check the box next to “Generate statistics from randomized alignments” and change the “Number of randomizations” up to “100.” “Close” the “Options” window. Press “Run” in the BestFit window and in a moment your output will appear. My BestFit comparison output file, the Xenopus HMG protein against the Human SRY protein, follows below:

```
BESTFIT of: NP_001025555  check: 2491  from: 1  to: 554
LOCUS       NP_001025555             554 aa            linear   VRT 03-APR-2006
DEFINITION  high-mobility group protein 2-like 1 [Xenopus tropicalis].

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

to: sry_human  check: 3683  from: 1  to: 204
Symbol comparison table: /usr/local/gcg/share/matrix/blosum62.cmp
CompCheck: 1102

Gap Weight:      8      Average Match:  2.778
Length Weight:      2   Average Mismatch: -2.248
Quality:    100      Length:     57
Ratio:  1.754      Gaps:      0
Percent Similarity: 49.123   Percent Identity: 38.596
Average quality based on 100 randomizations: 38.3 +/- 5.2

Match display thresholds for the alignment(s):
|= IDENTITY
: = 2
. = 1
```

45
Notice that the 49% similarity, 39% identity, is spread fairly evenly over the 60 residue long homology ‘patch’ discovered. This doesn’t seem like that big of a deal, yet the alignment receives a relatively high original quality, 100, versus a pretty low randomized quality, 38.3 with a standard deviation of 5.2. Therefore, the Z score calculates to be 11.9. The interpretation is that the similarity is very significant, in spite of a just significant, FastA expectation value of 7.5 \times 10^{-2}. This is corroborated by the fact that both sequences bind the same structural conformation of DNA, both have the same structural fold; they are clearly homologous.

So, a mediocre Expectation score can actually be very significant. The corollary is also sometimes true, usually due to composition effects. Always investigate further; similarities may strictly be artifacts. Good alignments with decently high percent identities can turn out to be entirely insignificant. A Monte Carlo style Z-test below around 3.5, near the bottom of Doolittle’s “Twilight Zone,” suggests that a similarity is not at all significant, that it is merely the result of compositional bias. As mentioned previously, the programs Xnu and Seg are available in the Wisconsin Package outside of BLAST for pre-filtering your sequences. This is particularly prudent in situations with molecules where you know that a lot of repeat and/or low complexity sequence composition has the potential to confound search algorithms. A seemingly decent alignment can turn out not be significant at all — do not blindly accept the output of any computer program!

Even worse, Expectation values are not ‘hard’ numbers; they are probabilities, and probabilities are always situational depending on the search that is being performed, due to the size and content of the databases being searched and the frequency of searching. Expectation values have to be considered in this light.

FrameAlign techniques

Before leaving today I want to show you one more thing. As mentioned previously, if you need to align DNA to protein, and especially if you suspect a frame shift sequencing error in the DNA sequence being considered, a very powerful pairwise alignment program, FrameAlign, is available. FrameAlign uses dynamic programming to align a protein to a DNA sequence with the allowance of frame shifts. Frame shift errors will appear in the pair output alignment as gaps that are not multiples of three.

Run FrameAlign on the best match from the Danio genome against SRY_Human. First load the top BLAST hit from Danio, “danio:NW_633572_w700001” into the SeqLab Editor using the Database Browser window as we’ve already discussed. Select both it and “SRY_HUMAN.” Now go to the “Functions” “Pairwise Comparison” “FrameAlign…” menu and press the “Run” button. No options are needed for this run, but
notice that the program defaults to local alignment, so if you need to align the full length of a protein sequence against a DNA template use the global option instead. Other helpful options write the output into separate gapped GCG single sequence format. My FrameAlign output follows below:

Local alignment of: NW_633572_w700001  check: 3142  from: 1  to: 110000  
to: sry_human  check: 3683  from: 1  to: 204

Scoring matrix: /usr/local/gcg/share/matrix/blosum62.cmp  
CompCheck: 1102  
Translation table: /usr/local/gcg/share/trans/translate.txt

Gap Weight: 8  Average Match: 2.778  
Length Weight: 2  Average Mismatch: -2.248  
Frameshift Weight: 0

Quality: 342  Length: 276  
Ratio: 3.717  Gaps: 0  
Percent Similarity: 73.913  Percent Identity: 68.478

Match display thresholds for the alignment(s):

| = IDENTITY  
: = 2  
. = 1

NW_633572_w700001 x sry_human  January 21, 2007 19:59 ..

So obviously Danio has an extremely significant match to the SRY gene in its genome, as reflected by the less than 10^{-30} BLAST E value, and seen again here with FrameAlign. In fact the BLAST report listed at least two with similar expectation values. It's more than likely some SOX orthologue, not a true SRY, but only more research could tell for sure.

That's enough for now.
Finishing up

You may want to use the “Output Manager” to delete many of the output files produced during this tutorial. They won’t be needed any further in the semester. However, do not delete the FastA output file. Account maintenance is your responsibility — only you can maintain your file and directory structure in your new Mendel account. If you can’t find your files or figure out what is what, you only have yourself to blame.

This should be just about enough for today. Exit SeqLab with the “File” menu “Exit” choice. You will be asked to “Save” the RSF file and the working list, if you’ve made any changes in either. Accept the suggested changes and designate a RSF file name that makes sense to you. Check “OK,” and the windows will go away; SeqLab will close. Log out of your current UNIX session on Mendel and your local workstation.

Homework Assignment

At some point this week log back onto Mendel with an X-tunneled ssh connection and use LookUp to find a UniProt protein sequence that relates to your semester project, preferably one coded for by the genomic sequence that you researched in last week’s assignment.

Load that sequence into SeqLab and then use it to run a BLAST search of the RefSeq protein database. Set a quite restrictive Expectation score cutoff value, say 0.0001. When the search finishes, save the resulting file with a name that you’ll recognize. You’ll be using that file in next week’s assignment.

Also note the names of the sequences and of the organisms that they come from in the resulting file. Does the same organism appear more than once? If so, you are most likely dealing with a sequence that has paralogous members, in other words, it has undergone gene duplication in that organism or in the lineage leading to it. You should have also discovered this last week in your Web searching. If this is the case, then you should try to decide whether you want your semester project to focus on a comparison of the members of that gene family, i.e. the paralogues, or on a particular member of that gene family across multiple organisms, i.e. the orthologues. Making this decision early will greatly help you in your project by reducing confusion.

After you’ve considered these things, create a file that describes the gene and/or protein that you will most likely be using for your project, and whether you will be comparing it within the context of a single organism’s gene family, or between different organisms. Either way, also list the organism, for a paralogous set, or the organisms, for an orthologous set that you will be using. Finally, tell us something about the Expectation scores in the BLAST search output. Did they gradually change from 0 to 0.0001, or were there fairly distinct categories of scores, with a group of most similar, then less similar, and perhaps another group of only somewhat similar sequences? If dealing with a paralogous dataset, you’ll often see the first two categories with the paralogues belonging to that second group. If dealing with proteins that have multiple domains, you’ll often see that third category of scores. Be sure to put your name, the date, and the course title in the file, and submit it to the Course Blackboard Digital Drop Box sometime before next week’s lab.
Conclusion

This has been a very long tutorial, sorry. We needed to introduce you to GCG and database searching methods both in one lab. That made it unavoidably long. However, you should now have a basic understanding of how the GCG Wisconsin Package for sequence analysis works and of how to use their SeqLab GUI. We will be using SeqLab all semester long — get used to it. Furthermore, sequence database similarity searching is a hugely misunderstood area of computational molecular biology, so it is worth the investment of time. A point that remains to be made is the previous techniques were performed largely using GCG’s suggested defaults. This usually will work just fine for you, but it is a good idea to think about what these default values imply and adjust them accordingly, especially if the results seem inappropriate after running through a first pass with the default parameters intact.

References


