BSC4933/ISC5224: Introduction to Bioinformatics

Laboratory Section: Wednesdays from 2:30 to 5:00 PM in Dirac 152.

Database Similarity Searching and the Dynamic Programming Algorithm

Lab Five, Wednesday, February 4, 2009

Author and Instructor: Steven M. Thompson

What’s available: algorithms, methodology, limitations, and significance. The mechanics: dot matrix analysis, pairwise dynamic programming, substitution matrices, hashing and heuristics, DNA versus protein, six frame ‘blind’ translation, and Expectations. And why homology isn’t the same as similarity.
"GCG® is the Genetics Computer Group, a product of Accelrys Inc., producer of the Wisconsin Package® for sequence analysis. © 2008 BioInfo 4U
Introduction

Given the nucleotide or amino acid sequence of a biological molecule, what can we know about that molecule? We can find biologically relevant information in sequences by searching for particular patterns that may reflect some function of the molecule. These can be catalogued motifs and domains, secondary structure predictions, physical attributes such as hydrophobicity, or even the content of DNA itself as in some of the gene finding techniques. But, what about comparisons with other sequences? Can we learn about one molecule by comparing it to another? Yes, naturally we can; inference through homology is a fundamental principle to all the biological sciences. We can learn a tremendous amount by comparing our sequence against others.

However, as we’ve seen in lecture, 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, $10^{68}$ 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 a simple overview of dynamic programming. My illustration is slightly different than that shown in lecture, but the concept is the same. Everybody has his or her own favorite way to illustrate the fundamental concept of the procedure. 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 starting on the left column on the next page:
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+1</td>
<td>0+1</td>
<td>1+1</td>
<td>0+1</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>

b) Now begin to add points based on the best path through the matrix, always working diagonally, left to right and top to bottom. The second row is completed here:

<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+1</td>
<td>0+1</td>
<td>1+1</td>
<td>0+1</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>

c) Continue adding points based on the best previous path through the matrix. The third row is completed here:

<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>0+1</td>
<td>0+1</td>
<td>1+1</td>
<td>0+2</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:

<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>
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..GGC  .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:

\[
\text{total penalty} = ( \text{[ length of gap]} \times \text{[ gap extension penalty]}) + \text{gap opening penalty}
\]

(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:
Pairwise alignment with a linear gap cost

a) First complete a match matrix using one point for matching and zero points for mismatching between bases:

<table>
<thead>
<tr>
<th></th>
<th>c</th>
<th>T</th>
<th>A</th>
<th>T</th>
<th>A</th>
<th>t</th>
<th>A</th>
<th>a</th>
<th>g</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</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>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>t</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>a</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

d) Finally, convert the score matrix into a trace-back path graph by picking the bottom-most, furthest right and highest scoring coordinate. Then choose the trace-back route that got you there, to connect the cells all the way back to the beginning using the same 'over-your-left-shoulder' rule. Only the two best trace-back routes are now highlighted with outline font in the trace-back matrix below:

<table>
<thead>
<tr>
<th></th>
<th>c</th>
<th>T</th>
<th>A</th>
<th>T</th>
<th>A</th>
<th>t</th>
<th>A</th>
<th>a</th>
<th>g</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>g</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>T</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>t</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>T</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th></th>
<th>c</th>
<th>T</th>
<th>A</th>
<th>T</th>
<th>A</th>
<th>t</th>
<th>A</th>
<th>a</th>
<th>g</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>g</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>T</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>t</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>T</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

These two trace-back routes define the following two alignments:

```
<table>
<thead>
<tr>
<th>cTATAatAag</th>
<th>cTATAatAag</th>
</tr>
</thead>
<tbody>
<tr>
<td>l l l l l l</td>
<td>l l l l l l</td>
</tr>
<tr>
<td>cg.TAatAg</td>
<td>.cgTAatAg</td>
</tr>
</tbody>
</table>
```
As we see here, there may be more than one best path through the matrix. Most software will arbitrarily (based on some internal rule) choose one of these to report as optimal. Some programs offer a HighRoad/LowRoad option to help explore this solution space. This time, starting at the top and working down as we did, then tracing back, I found two optimal alignments, each with a final score of 5, using our example’s zero/one scoring scheme. The score is the highest, bottom-right value in the trace-back path graph, the sum of six matches minus one interior gap in one path, and the sum of five matches minus no interior gaps in the other. This score is the number optimized by the algorithm, not any type of a similarity or identity percentage! This first path is the GCG Gap program HighRoad alignment found with this example’s parameter settings (note that GCG uses a score of 10 for a nucleotide base match here, 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: 0</td>
<td>Average Match: 10.000</td>
<td></td>
</tr>
<tr>
<td>Length Weight: 10</td>
<td>Average Mismatch: 0.000</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Quality: 50</th>
<th>Quality: 50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ratio: 6.250</td>
<td>Ratio: 6.250</td>
</tr>
<tr>
<td>Percent Similarity: 75.000</td>
<td>Percent Similarity: 62.500</td>
</tr>
<tr>
<td>Length: 10</td>
<td>Length: 10</td>
</tr>
<tr>
<td>Gaps: 2</td>
<td>Gaps: 0</td>
</tr>
<tr>
<td>Percent Identity: 75.000</td>
<td>Percent Identity: 62.500</td>
</tr>
</tbody>
</table>

1 cTATAtAagg 10  
| ||||| |
1 cg.TAtAaT. 8    1 .cgTAtAaT. 8

The GCG LowRoad alignment is my second, equivalent path. Notice that even though it has 62.5% identity as opposed to 75% identity in the HighRoad alignment, it has exactly the same score because of the scoring scheme we used! Another way to explore dynamic programming’s solution space and possibly discover alternative alignments is to reverse the entire process, i.e. reverse each sequences’ orientation. 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:

\[
S_{ij} = s_{ij} + \max \begin{cases} S_{i-1 \, j-1} & \text{or} \\ \max S_{i-x \, j-1} + w_{x-1} & \text{or} \\ \max S_{i-1 \, j-y} + w_{y-1} & \text{or} \\ 2 < x < i \\ 2 < y < i \end{cases}
\]
where $S_{ij}$ is the score for the alignment ending at $i$ in sequence 1 and $j$ in sequence 2,
$\text{s}_{ij}$ is the score for aligning $i$ with $j$,
$wx$ 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 ± 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 | 0 | -2 | -1 | -2 | 0 | -2 | -1 | -1 | -1 | -2 | -1 | -1 | 0 | 0 | 0 | -3 | -1 | -2 | -1 |
| B | -2 | 6 | -3 | 6 | 2 | -3 | -1 | -1 | -3 | -1 | -4 | -3 | 1 | 1 | -1 | 0 | -2 | 0 | -1 | -3 | -4 | -1 | -3 | 2 |
| C | 0 | -3 | 9 | -3 | -4 | -2 | -3 | -3 | -1 | -3 | -1 | -1 | -3 | -3 | -3 | -3 | -3 | -1 | -1 | -2 | -1 | -2 | -2 | -4 |
| D | -2 | 6 | -3 | 6 | 2 | -3 | -1 | -1 | -3 | -1 | -4 | -3 | 1 | -1 | 0 | -2 | 0 | -1 | -3 | -4 | -1 | -3 | 2 |
| E | -1 | 2 | -4 | 2 | 5 | -3 | -2 | 0 | -3 | 1 | -3 | 2 | 0 | -1 | 2 | 0 | 0 | -1 | -2 | -3 | -1 | -2 | 5 |
| F | -2 | -3 | -2 | -3 | -3 | 6 | -3 | -1 | 0 | -3 | 0 | 0 | -3 | -4 | -3 | -3 | -2 | -2 | -1 | 1 | 1 | 3 | -3 |
| G | 0 | -1 | -3 | -1 | -2 | -3 | 6 | -2 | -4 | -2 | -4 | -3 | 0 | -2 | -2 | -2 | -2 | 0 | -2 | -3 | -2 | -1 | -3 | -2 |
| H | -2 | -1 | -3 | -1 | 0 | -1 | -2 | 8 | -3 | -1 | -3 | -2 | 1 | -2 | 0 | 0 | -1 | -2 | -3 | -2 | -1 | 2 | 0 |
| I | 1 | -3 | -1 | -3 | 0 | -4 | -3 | 4 | 3 | 2 | 1 | -3 | -3 | -3 | -3 | -2 | 1 | -3 | -1 | -1 | -1 | -3 | -2 |
| K | 1 | -3 | -1 | -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 | -1 | 1 | -2 | -1 | -3 | -2 |
| M | 1 | 1 | 1 | 1 | -2 | -1 | 2 | -1 | 2 | 2 | 5 | -2 | -2 | 0 | -1 | -1 | -1 | 1 | 1 | -1 | -1 | -2 |
| N | 0 | 1 | 3 | 1 | 3 | 0 | 1 | -3 | 0 | -3 | -2 | 6 | -2 | 0 | 0 | 1 | 0 | -3 | -4 | -1 | -2 | 0 |
| P | 1 | -1 | -1 | -3 | -1 | -4 | -2 | -2 | -3 | -1 | -2 | -2 | 7 | -1 | -2 | -1 | -1 | -2 | -4 | -1 | -3 | -1 |
| Q | 1 | 0 | 2 | 1 | 1 | 0 | -1 | -1 | 0 | 1 | 1 | 5 | 1 | 0 | -1 | 2 | -2 | -1 | -1 | 2 |
| R | 1 | -2 | -3 | -2 | 0 | -3 | -2 | 0 | -3 | 1 | -2 | 0 | 0 | -1 | 5 | 1 | 0 | -1 | -2 | -3 | -1 | -2 |
| S | 1 | 0 | 1 | 0 | 0 | -2 | 0 | -1 | -2 | 0 | 0 | -1 | -1 | 1 | -1 | 0 | 0 | -1 | 4 | 1 | -2 | -3 | -2 |
| T | 0 | -1 | -1 | -1 | -1 | -2 | -2 | -1 | -1 | -1 | -1 | 0 | 0 | -1 | -1 | 1 | 1 | 5 | 0 | -2 | -1 | -2 |
| V | 0 | 1 | 3 | 1 | -3 | -2 | -1 | -3 | -3 | -2 | 1 | 1 | -3 | -2 | -1 | -3 | -2 | 0 | 4 | -3 | -1 | -2 |
| W | 3 | -4 | -2 | -4 | -3 | 1 | -2 | -2 | -3 | -3 | -2 | -1 | -4 | -4 | -2 | -3 | -3 | -2 | -3 | 11 | -1 | 2 | 3 |
| X | -1 | 1 | -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 | -2 | -2 | -2 | 1 | 2 | -1 | 7 |
| Z | -1 | 2 | -4 | 2 | 5 | -3 | -2 | 0 | -3 | 1 | -3 | -2 | 0 | 1 | 2 | 0 | 0 | -1 | -2 | -3 | -1 | -2 | 5 |

Identity values range from positive 4 through 11 and 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. To get another perspective on what these scores mean, read each program’s algorithm section in the GCG Program Manual (http://gcg.sc.fsu.edu/) — statistics can be confusing, but the descriptions are written well and do help.

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, and they are fast and easy. 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 many alignment programs. To utilize this strategy, compare two sequences using the appropriate algorithm, and randomize (jumble) the sequence that you are comparing your query to at least 100 times after the initial alignment is produced. This generates scores for the initial alignment and all the jumbled alignments and a standard deviation based on the spread of the jumbled scores. 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 score 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% (and 62.5%) 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 dice depending on how many dice you roll, whether they are ‘loaded’ or not, and how often you try. 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.

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 transitive. 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 sequences 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 in your sequence match words from 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 your sequence plus 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:
1) 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);
2) 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);
3) a translated (‘on-the-fly’) version of a DNA sequence against a protein database;
4) a protein sequence against a translated (‘on-the-fly’) version of the DNA database;
5) 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 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 rescored 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.

Then 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;
3) can find more than one region of gapped similarity;
4) very fast heuristic and parallel implementation;
5) 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 precomputed 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 ungapped alignments, the number of expected HSPs with a score of at least \( S \) is given by the formula: \( E = Knme^{-\lambda s} \). 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 precomputed, simulated distributions. These two parameters define the statistical significance of an \( E \)-value. As with FastA the \( E \)-value defines the significance of the search. As mentioned above, the smaller the \( E \)-value, the more likely it’s significant.

In review, both the FastA and BLAST 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 most Monte Carlo style Z scores do, since they are often based on the Normal distribution. Regardless, they parallel 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. A value of 0.01 is usually a decent starting point for significance in most typical searches.

Furthermore, all database searching, regardless of the algorithm, 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 at least 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.

**Standard disclaimer**

I write these tutorials 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, it 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. This also makes the tutorials longer than otherwise necessary. Sorry.

I use three writing conventions in the tutorials, besides my casual style. 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 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.

**A ‘real-life’ project oriented approach: database searching and significance**

Activate and log on to the computing workstation you are sitting at and then log onto HPC with an X-tunneled ssh session. If using an terminal window on Mac OS X or UNIX/Linux issue the following command (the X has to be capitalized and replace “user” with your account name):

```
$ ssh -X user@submit.hpc.fsu.edu
```

As you’ve learned, specialized X-server graphics communications software is required to use GCG’s SeqLab. I’ll remind you of a few things about using X: X Windows are only active when the mouse cursor is in that
window, and always close X Windows when you are through with them to conserve system memory. Furthermore, to activate X items, just <click> on them, rather than holding your mouse button down. Also, X buttons are turned on when they are pushed in and shaded. Finally, don’t close X Windows with the X-server software’s 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.

Preliminary preparations

Change your directory from ‘home’ to last week’s subdirectory. List that directory and check out the files left over from last week’s tutorial. As I’ve said before, files really tend to accumulate quickly when using SeqLab. Look through them and remove any that you don’t want to save. However, be sure to save the RSF file that has the genomic sequence that you brought into SeqLab from GenTrainData last week; we’ll be using it today. Delete the entire SeqMerge project database subdirectory If you want; it’s OK with me. We won’t need it anymore. You can use the following command to delete an entire subdirectory without the need to delete all of its individual files first:

    $ rm --r subdirectory_name

But be very careful with this command; you could delete your entire account if used improperly!

Next, change directory back to your home directory, create a subdirectory for this week’s tutorial data, and then change directory into it.

After you’ve taken care of these file maintenance chores launch SeqLab with the standard command:

    $ seqlab &

Next, it would again be helpful to change your SeqLab working directory to your present location so that everything you do today will automatically be saved in your new directory rather than last week’s directory. Do this as before with SeqLab’s “Options” “Preferences…” “Working Dir…” button.

Now verify that you are in SeqLab’s “Main List” “Mode:” and start a new list to contain this week’s data. Therefore, select “New List…” from the “File” menu and give your new list an appropriate name. It’s not essential to use the file name extension “.list” but it’s a good idea. Check “OK.”

You should now be in List Mode with an empty window. Go to the “File” menu and select “Add Sequences From” “Sequence Files…” Use the “Directories” column to move from your present directory over to last week’s subdirectory and then replace the text in the “Filter” text box with a name or wildcard that will identify the RSF file that has your actual genomic sequence from last week (the one you brought in from GenTrainData). Press the “Filter” button and then select the correct entry. Press the “Add” button to add it into your new empty list file and then “Close” the “Add Sequences” window. Go to the “File” menu and press “Save List.”
Traditional database searching: the FastA family

The first widely used heuristic, hashing-style database searching algorithm was FastA (Pearson and Lipman, 1988; and Pearson, 1998; also see WordSearch, Wilbur and Lipman, 1983). This family of programs is incorporated into GCG (see the GCG online Program Manual at http://gcg.sc.fsu.edu/ for further details apart from what I described in the Introduction). In spite of the fast hashing, heuristic style algorithm incorporated, they are quite CPU intensive and can take a while to run. Therefore, you should submit them as a batch or background process when screening current databases (or, better yet, use the HPC’s MOAB scheduling system with native serial or MPI FastA). Most GCG database similarity searching programs accept an automatic batch submission option from the command line (--batch), or, even easier, default to background mode in SeqLab.

The FastA family includes several programs:

1) The original FastA program compares a sequence query and database of the same type, either protein or nucleotide.

2) FastX compares a nucleotide query against a protein database by translating the query in all six frames allowing for frame shifts.

3) TFastA and TFastX compare a protein sequence query against all six translations of a nucleotide database.

TFastA and TFastX take advantage of the size of the DNA databases, and yet still retain 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 FastX in today’s tutorial. It is especially useful for situations like we are emulating, that is, we’ve just sequenced a new stretch of genomic DNA, and we want to see where the coding regions within it lay, and what they are most similar to.

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 as you are with BLAST. Therefore, let’s take advantage of this feature by preparing a LookUp search list of all the UniProt proteins that come from our specific project molecule’s sort of organism. Therefore, just as you did back in Lab #2, go to the “Functions” “Database Reference Searching” menu and choose “LookUp. . .” to launch the Wisconsin Package’s sequence database annotation searching program. However, this time you won’t be looking for any particular protein, rather you’ll be looking for all of the proteins from whatever group of organisms your project molecule comes from, either vertebrates (Vertebrata), or fungus (Fungi), or primitive, non-vascular plants (Viridiplantae ! Tracheophyta). Specify “UniProt” to “Search the chosen sequence libraries” and type your taxonomic designation in the “Organism” category. Do not use any other restrictions. For my search, as before, I typed in the appropriate organism Boolean clause, but this time I did not specify a particular protein, in order to find
all of the ‘primitive’ eukaryotic proteins. Press the “Run” button. The result file will be huge. That’s fine; mine contains 185,099 entries! Yours will likely be much bigger. “Close” the LookUp results file, and then press the “Add to Main List” button in the “SeqLab Output Manager” and “Close” the window afterwards. Use the “File” menu and again “Save List.” Do not select the file and attempt to switch “Mode:” to “Editor.”

Do select the name of the RSF file that holds your genomic sequence in your list and switch “Mode:” to “Editor.” If there is more than one sequence in the display, select all their names except the genomic entry and “CUT” them away. Select your genomic entry next and go to the “Functions” “Database Sequence Searching” menu and select “FastX…” (not FastX+) to start the FastX program. If a "Which selection" window pops up asking if you want to use the "selected sequences" or "selected region;" choose "selected sequences" to run the program on the full length of your genomic sequence.

“Using” all of “uniprot:*” is the default FastX “Search Set…” database. This takes advantage of the great annotation in UniProt, but UniProt is really huge, so the search would take quite while, plus there’s all sorts of things in UniProt that we’re not really interested in, and finding what we are interested in within the huge output is problematic. Therefore, we’ll change the search set specification from “uniprot:*” to the LookUp list file we just made. This will make the search run more quickly, will exclude sequences we are not interested in, and will dramatically increase its sensitivity. Use the “Search Set” button to select “uniprot:*,” in the “Build FastX’s Search Set” pop up box, and then press “Remove from Search Set.” Next, press the “Add Main List Selection…” button and then pick your new LookUp file in the “List Chooser” window that pops up to identify your new list file. Press “Add” and then “Close” the “List Chooser” window and the “Build Search Set” windows. The other parameters in the main FastX 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 “2.00” to a more reasonable “1.00,” just to reduce the output list size some.

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’re not interested in the pairwise alignments and want smaller output files. Some of the other options can be helpful depending on your specific situation and can be explored in your own research. Restricting your search by the database sequence length or by date of their deposition in the database may be handy. The program default “Save and sort by optimized score” option (–OptAll) causes the algorithm to sort its output based on a normalized derivative of the optimum score, the result of the program’s dynamic programming ‘in-a-band’ pass, and what you want, rather than the initn score, which is the longest combined word score. “Close” the “Options” window, be sure that the “FastX” program window shows “How:” “Background Job,” and then press the “Run” button.

To check on the job’s progress go to SeqLab’s “Windows” menu and choose “Job Manager.” Select the “FastX” entry to see its progress (though there is no indication of how much longer the job will take!) 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, you use the “Job Manager” to “Stop” the given job. Go on with the rest of the tutorial rather than waiting for the FastX results; depending on the HPC submit head node’s load it could be a while.

**BLAST: Internet and local server based similarity searching**

As described in the Introduction, 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 similarity database searching, yet maintains the sensitivity of older 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 BestFit type alignments of the FastA family), but it 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 (though you can build your own custom BLAST databases with FormatDB+, and if you are feeling bold, go ahead and do that and then run BLAST from the command line to specify your custom database). You can fine-tune BLAST by altering its operating parameters and taking advantage of the many options available in it. However, BLAST is not the best tool for comparing nucleotide sequences against the nucleotide database without translation, especially with short sequences. In this situation, with default parameters, it will only find nearly identical DNA sequences, but will not be able to locate sequences that are only somewhat similar. 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. In addition to this tutorial’s introduction, 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 the HPC. You’ll often have to wait in a queue though, because the server gets very busy. 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 their sequence databases every night. However, realize that this program, as with the local version of BLAST, is optimized for protein comparisons, and, unlike other GCG programs, the list generated by NetBLAST is not appropriate as input to other GCG analyses.
NetBLAST returns files in NCBI’s own format, incompatible with GCG (but see GCG’s 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 does produce 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 your genomic sequence entry name is still selected and then pick “Blast…” (not Blast+) off of the “Functions” “Database Sequence Searching” menu. As above, if a "Which selection" window pops up asking if you want to use the "selected sequences" or "selected region," choose "selected sequences." The program default on the main window is to “Search a nucleotide database” “Search Set. . .” “Using local genbank” with a nucleotide query. Using BLAST in this manner, that is a nucleotide query against a nucleotide database without translation, activates BLASTN and is the very worst way to run BLAST (but most commonly used). That is because, as explained previously, BLAST is not optimized for this type of search, and DNA just doesn’t have the ‘look-back’ time of protein regardless of what program used. The sensitivity is just not there. Therefore, we will run TBLASTX. This program translates both the query and the database sequences. We could run BLASTX by searching a local protein database, the “Search a protein database” choice, but I want to generate a nucleotide dataset to be used later on in the semester, as well as the protein dataset we generated with FastX. We will switch from the GenBank default to the mRNA section of RefSeq though, to speed things up a bit. Therefore, press the “Search Set….” button and pick “Local | Nuc | rs_rna” and then “Close” the “Search Set” window. As in the FastA programs, decreasing the Expectation cutoff value will decrease the output list size. Set “Ignore hits that might occur more than how many times by chance alone” from its default “10.0” down to “1.00” as before.

Push the “Options…” button next, where we can specify TBLASTX. Check in “Translate nucleotide query (TBLASTX)” to do this. Notice that if you are working with something other than ‘standard’ nuclear genes, you can specify alternate genetic code translation tables. Also notice that “Filter input sequences for low complexity regions” is turned on by default. This powerful option causes troublesome repeat and low information portions of the query sequence to be ignored in the search, and should generally be taken advantage of. This screening of low complexity sequences from your query minimizes search confusion due to random noise. (Programs that perform this function on protein sequences, Xnu and Seg, are available separately in GCG for prescreening protein sequences prior to other analyses besides BLAST.) Also notice the “Display alignments from how many sequences” button; this generates the –Align option, useful for suppressing unneeded segment alignments and reducing the size of the output file. The standard output file is very long because BLAST in SeqLab automatically aligns the best 250 matches so you may wish to reduce this parameter. However, beginning and ending attributes are only saved in the BLAST output list file from those segment alignments that you request. “Close” the “Options” window and then press the “Run” button in BLAST’s main window after assuring that “How:” shows “Background Job.”

Read on through the rest of the tutorial at this point, though the BLAST run may be done before your FastX run. If you end up waiting to finish the tutorial later on, then “Exit” SeqLab and save your RSF and list files,
and then log off HPC and the Classroom computer. When you return to SeqLab your results will be waiting for you and you can complete the tutorial.

When the TBLASTX and FastX search results are ready, use the “Output Manager” located under SeqLab’s “Windows” menu to display and save your BLAST and FastX output files with appropriate names. You can also use the “Job Manager” located there to check on the status of your running jobs. Just select the job to see its status.

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

I will show abridged database search output files from my example elongation factor DNA genomic sequence next. Naturally, the topmost ‘hits’ will turn out to be EF-1α proteins and genes for me; it’s the ones below the expected hits that may prove interesting for this section of the tutorial. Those results follow below, starting with TBLASTX. Especially pay attention to BLAST’s E value scores in its output file. As explained in the Introduction, these are the likelihoods (Expectations) that the observed matches could be due to chance, the smaller the E number, the more significant the match. They are much easier to interpret than the information theory bits score in the adjacent column. Here is my greatly abridged TBLASTX output from the search of RefSeqRNA. Remember, yours should be of your project molecule, not my example below:

```plaintext
!!SEQUENCE_LIST 1.0
TBLASTX 2.2.10 [Oct-19-2004]
Query= /panfs/storage.local/genacc/home/stevet/.seqlab-submit/input_12.rsf{TRBEF1AE} (1964 letters)
Database: rs_rna
1,211,414 sequences; 2,053,035,099 total letters
Searching. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .done

<table>
<thead>
<tr>
<th>Score</th>
<th>E</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1016</td>
<td>0.0</td>
<td></td>
</tr>
<tr>
<td>959</td>
<td>0.0</td>
<td></td>
</tr>
<tr>
<td>439</td>
<td>0.0</td>
<td></td>
</tr>
</tbody>
</table>
```

rs_rna:XM_814346  Begin: 1 End: 1278
rs_rna:XM_814346  Begin: 3 End: 644 Strand:-
rs_rna:XM_814346  Begin: 723 End: 1298 Strand:-
rs_rna:XM_814346  Begin: 723 End: 1298 Strand:+

rs_rna:NM_200009  Begin: 727 End: 1368
rs_rna:NM_200009  Begin: 43 End: 693
>rs_rna:XM_814346 Trypanosoma cruzi strain CL Brener.
Length = 1350
Score = 1016 bits (2213), Expect = 0.0
Identities = 424/426 (99%), Positives = 424/426 (99%)
Frame = +3 / +1

Query: 417 MGKEKVHMLNVLVGVHDAGKSTATGHLIYKCGGIDKRTIEKFEKEEAEGKSSFKYAWVL 596
MGKEKVHMLNVLVGVHDAGKSTATGHLIYKCGGIDKRTIEKFEKEEAEGKSSFKYAWVL
Sbjct: 1 MGKEKVHMLNVLVGVHDAGKSTATGHLIYKCGGIDKRTIEKFEKEEAEGKSSFKYAWVL 180

Query: 597 DKLKAEREPGITIDIALWKFESPKSVFTIIDAPCHRDIFIKNMITGTPQADAALVIASSQ 776
DKLKAEREPGITIDIALWKFESPKSVFTIIDAPCHRDIFIKNMITGTPQADAALVIASSQ
Sbjct: 181 DKLKAEREPGITIDIALWKFESPKSVFTIIDAPCHRDIFIKNMITGTPQADAALVIASSQ 360

Query: 777 GEFEGASKDGQREHALLAFTGKVQMVVCCNKMDKSVNFAQERYDEIVKEVASYLLK 956
GEFEGASKDGQREHALLAFTGKVQMVVCCNKMDKSVNFAQERYDEIVKEVASYLLK
Sbjct: 361 GEFEGASKDGQREHALLAFTGKVQMVVCCNKMDKSVNFAQERYDEIVKEVASYLLK 540

Query: 957 VGYNEKEVFRIPIISGWQDMIDKSNEMPWYKGTIPEALDMEPVRPSDKPRPLQD 1136
GYNEKEVFRIPIISGWQDMIDKSNEMPWYKGTIPEALDMEPVRPSDKPRPLQD
Sbjct: 541 VGYNEKEVFRIPIISGWQDMIDKSNEMPWYKGTIPEALDMEPVRPSDKPRPLQD 720

Query: 1137 VYKIGGGITVPVGRVTMKGPDGVVTAPANVTEVKSIESHEQQLEATPGDNVGGV 1316
VYKIGGGITVPVGRVTMKGPDGVVTAPANVTEVKSIESHEQQLEATPGDNVGGV
Sbjct: 721 VYKIGGGITVPVGRVTMKGPDGVVTAPANVTEVKSIESHEQQLEATPGDNVGGV 900

Query: 1317 KNVSVKDIRRNCGSNKNDPPKEAADFTAQVIILNHQGQIGNYAPVLDCHTCHIACKP 1496
KNVSVKDIRRNCGSNKNDPPKEAADFTAQVIILNHQGQIGNYAPVLDCHTCHIACKP
Sbjct: 901 KNVSVKDIRRNCGSNKNDPPKEAADFTAQVIILNHQGQIGNYAPVLDCHTCHIACKP 1080

Query: 1497 AEIESKIDRDSSKGELEKNSIKSGDAMVRMVQPQKPMCEVEVFNYAPLGRFAVDRMRTQT 1676
AEIESKIDRDSSKGELEKNSIKSGDAMVRMVQPQKPMCEVEVFNYAPLGRFAVDRMRTQT
The output is a perfectly suitable GCG list file, complete with beginning and ending attributes for those alignments specified and complementary strand attributes if necessary when searching a DNA database. The pairwise alignments requested are illustrated with 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, and is seen in my example above. This can be particularly helpful in
those cases where the query, as it is in our ‘project’ cases, or the database entry, is from genomic DNA and has several dispersed exons each with separate homologies.

You will often be able to see a demarcation where the Expectation values drop off between the significant hits and background noise. In my EF-1α case I expect to see the best E values for other EF-1α proteins and close homologues, and then another bracket of very good values for other sequences with GTP-binding protein P-Loop signatures, and finally, a category of not so good scores that reflect background noise. Unfortunately in my BLAST search of RefSeqRNA, EF-1α is such a well studied protein that I hit the default maximum list size of 500 before venturing beyond GTP-binding proteins. In fact I didn’t get an Expectation value greater than 10^{-120}! One thing that is certain, my genomic sequence is an EF-1α from Trypanosoma cruzi. Your searches will hopefully not be as troublesome.

Next I’ll show my abridged example FastX search output file against the UniProt LookUp list. You can see the first E value bracket that I mention above here. The scores rise more quickly above 10^{-11} where EF-1α paralogues and other GTP-binding proteins first show up. I just barely got into the neighborhood of background random sequences beyond all of the EF-1α, EF-Tu, and other GTP-binding proteins with the Expectation value cutoff set at 1.00 in this search.

This search was not as comprehensive as the BLAST search, since we didn’t look through all of UniProt, just the portion of it that contains our organisms of interest. That makes for a more sensitive search, and is why the scores rise so much more quickly here than in the TBLASTX search of RefSeqRNA.

!!SEQUENCE_LIST 1.0
(Nucleotide) FASTX of: genomic.seq from: 1 to: 1964 January 31, 2008 12:03
TO: @primitive.uniprot.list Sequences: 185,099 Symbols: 83,498,991 Word Size: 2
Databases searched:
Searching with both strands of the query.
Scoring matrix: GenRunData:blosum50.cmp
Constant pamfactor used
Gap creation penalty: 15 Gap extension penalty: 2 Frameshift penalty: 20

Histogram Key:
Each histogram symbol represents 323 search set sequences
Each inset symbol represents 11 search set sequences
z-scores computed from opt scores

<table>
<thead>
<tr>
<th>z-score obs</th>
<th>exp</th>
</tr>
</thead>
<tbody>
<tr>
<td>(&lt; 20)</td>
<td>332</td>
</tr>
<tr>
<td>22</td>
<td>32</td>
</tr>
<tr>
<td>24</td>
<td>43</td>
</tr>
<tr>
<td>26</td>
<td>62</td>
</tr>
<tr>
<td>28</td>
<td>123</td>
</tr>
<tr>
<td>30</td>
<td>333</td>
</tr>
<tr>
<td>32</td>
<td>1139</td>
</tr>
<tr>
<td>34</td>
<td>2899</td>
</tr>
<tr>
<td>36</td>
<td>5727</td>
</tr>
<tr>
<td>38</td>
<td>9914</td>
</tr>
</tbody>
</table>
The best scores are: init1 initn opt z-sc E(184529)...

UNIPROT_TREMBL:O00819_TRYCR Begin: 1 End: 449
! O00819 trypanosoma cruzi. elongatio... 2984 2984 2984 2732.0 4.3e-145

UNIPROT_TREMBL:Q5XXD2_TRYCR Begin: 1 End: 449
! Q5xxd2 trypanosoma cruzi. elongatio... 2960 2960 2960 2710.0 7.1e-144

UNIPROT_TREMBL:Q4DYF9_TRYCR Begin: 1 End: 449
! Q4dyf9 trypanosoma cruzi. elongatio... 2950 2950 2950 2700.9 2.3e-143

UNIPROT_TREMBL:Q4CXI1_TRYCR Begin: 1 End: 449
	
UNIPROT_TREMBL:Q5EVA3_9STRA Begin: 1 End: 414
! Q5eva3 mallomonas rasilis. elongati... 1329 2094 2118 1941.1 4.8e-101

UNIPROT_TREMBL:Q77478_PLABE Begin: 1 End: 443
! Q77478 plasmodium berghei. elongati... 1292 2082 2101 1925.2 3.7e-100

UNIPROT_TREMBL:Q86LK4_9EUKA Begin: 2 End: 398
! Q86lk4 streblomastix strix. elongat... 2099 2099 2099 1923.9 4.3e-100

UNIPROT_TREMBL:Q96978_9CILI Begin: 1 End: 408
! O96978 naxella sp. translation elon... 1751 1751 2091 1916.5 1.1e-99

UNIPROT_TREMBL:Q5EVA2_9STRA Begin: 1 End: 414
! Q5eva2 plectospira myriandra. elong... 1672 1672 1672 1534.7 2.1e-78
! Q7rj38 plasmodium yoelii yoelii. el... 111 111 153 143.2 0.59
UNIPROT_TREMBL:Q7QWN8_GIALA Begin: 2 End: 467 ! vs rev query
! Q7qw8 giardia lamblia atcc 50803. ... 59 88 152 143.2 0.59
UNIPROT_TREMBL:Q81335_PLAF7 Begin: 230 End: 401
! Q8i335 plasmodium falciparum (isola... 111 164 152 141.5 0.73
UNIPROT_TREMBL:Q54Z32_DICDI Begin: 48 End: 245 ! vs rev query
! Q54z32 dictyostelium discoideum ax4... 83 83 142 141.0 0.78
UNIPROT_TREMBL:Q8I335_PLAF7 Begin: 230 End: 401
! Q8i335 plasmodium berghei. hypothet... 102 147 144 140.8 0.8
UNIPROT_TREMBL:Q54Z32_DICDI Begin: 48 End: 245 ! vs rev query
! Q54z32 dictyostelium discoideum ax4... 83 83 142 141.0 0.78
UNIPROT_TREMBL:Q4TB90_PLABE Begin: 94 End: 298
! Q4tb90 plasmodium berghei. hypothe... 102 147 144 140.8 0.8
UNIPROT_TREMBL:Q54Z32_DICDI Begin: 48 End: 245 ! vs rev query
! Q54z32 dictyostelium discoideum ax4... 83 83 142 141.0 0.78
UNIPROT_TREMBL:Q7R371_GIALA Begin: 6 End: 227 ! vs rev query
! Q7r371 giardia lamblia atcc 50803. ... 96 125 141 140.4 0.84
UNIPROT_TREMBL:Q8I335_PLAF7 Begin: 230 End: 401
! Q8i335 plasmodium berghei. hypothet... 102 147 144 140.8 0.8
UNIPROT_TREMBL:Q54Z32_DICDI Begin: 48 End: 245 ! vs rev query
! Q54z32 dictyostelium discoideum ax4... 83 83 142 141.0 0.78
UNIPROT_TREMBL:Q4TB90_PLABE Begin: 94 End: 298
! Q4tb90 plasmodium berghei. hypothe... 102 147 144 140.8 0.8
UNIPROT_TREMBL:Q54Z32_DICDI Begin: 48 End: 245 ! vs rev query
! Q54z32 dictyostelium discoideum ax4... 83 83 142 141.0 0.78
UNIPROT_TREMBL:O96305_9EUKA Begin: 81 End: 182
! O96305 monocercomonas atcc50210. tr... 72 133 140 140.2 0.87
UNIPROT_TREMBL:Q26878_TRYCR Begin: 21 End: 195
! Q26878 trypanosoma cruzi. mucin-lik... 43 43 139 139.7 0.93
\End of List

trbef1ae
UNIPROT_TREMBL:O00819_TRYCR
ID O00819_TRYCR PRELIMINARY; PRT; 449 AA.
AC O00819;
DT 01-JUL-1997, integrated into UniProtKB/TrEMBL.
DT 01-JUL-1997, sequence version 1.
DT 02-MAY-2006, entry version 29.
DE Elongation factor 1-alpha. . . .
SCORES Init1: 2984 Initn: 2984 Opt: 2984 z-score: 2732.0 E(): 4.3e-145
>>UNIPROT_TREMBL:O00819_TRYCR                             (449 aa)
initn: 2984 init1: 2984 opt: 2984 Z score: 2732.0 expect(): 4.3e-145
Smith-Waterman score: 2984; 100.0% identity in 449 aa overlap
(417-1763:1-449)

420 450 480 510 540 570
trbef1ae MGKEKVHMNLVVVGHVDAGKSTATGHLIYKCGGIDKRTIEKFEKEAAEIGKSSFKAYWVL
        ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
O00819_TRYCR MGKEKVHMNLVVVGHVDAGKSTATGHLIYKCGGIDKRTIEKFEKEAAEIGKSSFKAYWVL
         10  20     30   40     50     60
600 630 660 690 720 750
trbef1ae DKLKAEREPGITIDIALWKFESPKSVFTIDAPGRDFIKNMTGTPQADAvLVIASSQ
        |||||||||||||||||||||||||||||||||
O00819_TRYCR DKLKAEREPGITIDIALWKFESPKSVFTIDAPGRDFIKNMTGTPQADAvLVIASSQ
         70  80    90   100    110    120
780 810 840 870 900 930
trbef1ae GEFEAGISKDGQTREHALLAFTLGVKQMVVCCNKMDDKSVNFAQERYDEVKESAYLKK
        ||||||||||||||||||||||||||||||||||||||
O00819_TRYCR GEFEAGISKDGQTREHALLAFTLGVKQMVVCCNKMDDKSVNFAQERYDEVKESAYLKK
         130 140  150   160    170    180
960 990 1020 1050 1080 1110
trbef1ae VGYNVEKVRFIPISGWQDGNMKDSNQMFPWYKGPTLEALDMLEPPVRPSTKDPLRLPLQD
        ||||||||||||||||||||||||||||||||
O00819_TRYCR VGYNVEKVRFIPISGWQDGNMKDSNQMFPWYKGPTLEALDMLEPPVRPSTKDPLRLPLQD
         190 200  210   220    230    240
1140 1170 1200 1230 1260 1290
trbef1ae VYKIGGTTVPQVRVTGTMKPVDVTVTPAPVTSVEKSIEMHHEQLAETPGDNVGFNV
        ||||||||||||||||||||||||||||||||||||| |
O00819_TRYCR VYKIGGTTVPQVRVTGTMKPVDVTVTPAPVTSVEKSIEMHHEQLAETPGDNVGFNV
         250 260  270   280    290    300
1320 1350 1380 1410 1440 1470
trbef1ae KNVSVDIRGNNVCGNSKDPPKEAADFTAQVIIILNHGPQIGNYAPVLDCHTCHIACKF
        |||||||||||||||||||||||||||||||||||||
O00819_TRYCR KNVSVDIRGNNVCGNSKDPPKEAADFTAQVIIILNHGPQIGNYAPVLDCHTCHIACKF

27
UNIPROT_SPROT:EF1C_PORPU

ID   EF1C_PORPU     STANDARD;      PRT;   449 AA.
AC   P50256;
DT   01-OCT-1996, integrated into UniProtKB/Swiss-Prot.
DT   01-OCT-1996, sequence version 1.
DT   21-MAR-2006, entry version 37.
DE   Elongation factor 1-alpha C (EF-1-alpha). . . .

SCORES   Init1: 1204  Initn: 2281  Opt: 2267  z-score: 2076.8 E(): 1.3e-108
>>UNIPROT_SPROT:EF1C_PORPU                                (449 aa)
initn: 2281 init1: 1204 opt: 2267 z-score: 2076.8 expect(): 1.3e-108
Smith-Waterman score: 2267;    75.3% identity in 449 aa overlap
(417-1751:1-448)
The FastX output file is also an acceptable GCG list file with database sequence begin and end attributes that can serve as input to other GCG programs. These may include strand attributes where necessary when using DNA, to designate the reverse complement DNA strand. Pay particular attention to reverse query strand comments in the list denoted by “! vs rev query” (especially prevalent in the p21 RAS project system), as seen in three of mine in the screen trace above. The file shows a histogram of the extreme value 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, “::.” Forward slashes, “/,” indicate frame shifts on the DNA strands necessary for the alignment to complete. The numbering coordinates in this section of the output can be used to go back to the original nucleotide entry and ascertain translated areas that may correspond to exons of genomic sequence.

Notice that every ten query amino acids here increase in numbering by thirty — the triplet bases of a codon.

The histogram of the score distribution 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 default FastX Expectation cutoff of 2.0 assures this. The inset shows a zoom-in blowup of the highly significant score end of the graph — these are the best alignments found by the program, not the worst! The histogram can be suppressed with the –NoHistogram option if desired.

Notice that the entries are sorted by a “z” score parameter. This z-score is based on a normalization of the opt scores and their distribution from the rest of the database. It is different than the more traditional Monte Carlo style normal 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.) Both help describe the statistical significance of an alignment.

As in BLAST reports, the Expectation function, $E()$, is by far the most important column. It works the same as in BLAST — here it 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 BLAST E-values, the smaller the number, the better. As a conservative rule-of-thumb, for a search against a protein database of around 100,000 sequences, 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 only perhaps homologous, although these guidelines can be skewed by compositional biases. Furthermore, Expectation values need to be considered in light of each search, completely dependent on the size and content of the database, and on how often you do the search.

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

It’s easy to know that our sequence aligns well with itself and with other close homologues; we know these alignments are significant. They don’t cause anybody any problems; they’re obvious. Therefore, for the remainder of this section we’ll use sequences where the similarity isn’t so obvious. You’ll find two interesting sequences from your search that are not recognizably from the same gene family as your project molecule, and that have ‘barely decent’ Expectation scores. We’re interested in how some of the not so similar, as Russell Doolittle calls them ‘twilight zone,’ sequences, align, and the significance of those alignments. Therefore, choose two completely different, ‘twilight zone’ entries, from your FastX program run (the TBLASTX run was against a nucleotide database, and sequences from it will require translation for analysis — we’ll deal with this dataset later in the semester). This may be a little hard to do with the project molecule searches since they all belong to huge gene families that are very well studied, so it’s likely your searches just barely got beyond true homologues, if they did at all, but do what you can. Try to avoid using obvious homologues based on their definition title unless absolutely necessary, and try for two different types of proteins. There are no ‘correct’ answers here; we just want to see some interesting comparisons. Barring all else, just pick a couple of the bottommost entries. Write down your choices. We will experiment with various methods for analyzing the significance of these sequences’ similarities. Relevant lines showing the two choices from my FastX search are shown here to illustrate the comparison concept.

I chose the poorest score from my FastX search with the *Trypanosoma cruzi* EF-1α genomic sequence against my custom ‘primitive’ LookUp list as my first comparison. The entry is from the same organism as the query, *Trypanosoma cruzi*, but with a near random Expectation score, $9.3 \times 10^{-1}$, it is difficult to know whether this “mucin-like” protein has an EF-1α homologous domain or not. It is clearly a ‘twilight zone’ entry or worse:
I chose a slightly more significant comparison for my second analysis. This one comes from the so-called ‘slime mold,’ *Dictyostelium discoideum*. So-called because it is neither slime, nor a mold, and is actually a type of amoeba that lives in the soil and has a somewhat complicated unicellular and multicellular life cycle. Its FastX Expectation value was $1.5 \times 10^{-1}$.

```
UNIPROT_TREMBL:Q54NB8_DICDI Begin: 124 End: 456
! Q54nb8 dictyostelium discoideum ax4... 82 116 165 153.7 0.15
```

Load your two sequence choices into SeqLab by going to the "File" "Add sequences from" “Databases...” menu. Type the name of the entry, including its database logical name and colon separator (logical_name:entry_name), in the “Database Specification:” box of the “SeqLab Database Browser” and then press the “Add to Main Window” button. “Close” the browser box after adding the two sequences into the Editor. <Double-click> on each new entry’s name or use the “INFO” icon with the sequence’s name selected to read about each new sequence.

Your genomic sequence is nucleic acid and the new entries are protein, so an additional step is necessary before we can directly compare them. We need to translate the protein coding region in our genomic sequence that corresponds to the region discovered by the search algorithms. Sometimes you can use the following method, but not here since we searched with an unknown genomic query:

The easiest way to translate an annotated DNA sequence is to use the sequence’s CDS (coding sequence) feature annotation, if it has any and it makes sense. If it does, then <double-click> anywhere within the sequence to launch the “Sequence Features” window, and then switch the features being displayed from “Show:” “Features at the cursor” to “All features in current sequence.” This will allow you to scroll through the entire sequence’s feature list and select any that are relevant. Often with genomic DNA you’ll have to select every CDS of each exon associated with a particular gene. (Do not select “mRNA” or “exon” features — UTR’s and/or splicing variants may mess you up.) Be wary of translations that do not begin at position one. These are flagged in the entry’s Features annotation with “/codon_start=2” or “3” and are often seen in cases where the actual CDS begins before the sequence begins and ends after the sequence ends: “CDS <1 . . >418.” In these cases you’ll have to use the “Edit” menu “Select Range.” function to only select the region that you want to translate by providing the correct beginning and ending numbers.

If a nucleotide entry is not annotated with CDS feature data, such as our genomic sequences, and as is the case with most of the tags database, then you have to translate the entry using some other criteria. BLAST and Fast searches list beginning and ending attributes in the list portion of the file (unless suppressed) and indicate whether the similarity was found on the forward or reverse strand when using DNA, so you can translate DNA sequences found by those searches using that information to trim away everything not needed. You need to use the “PROTECTIONS” and “CUT” icons and perhaps the “Edit” “Reverse...” “Reverse and Complement” function to achieve this. Regardless of how you identify the region(s) to be translated, the “Edit” menu “Translate. . .” function can be used to translate the selected region(s).
However, since we’re pretending our genomic sequence is brand new off of an automated sequencing system, it has no annotation, and we have no idea what part of it to translate other than some hints from our similarity searches. That’s what this lab is all about!

Therefore, let’s just translate it blindly, just like the search programs did. I can guarantee you that we don’t need to worry about the reverse strand at least — since I designed all of the genomic sequences to be in the forward direction only. Select your genomic sequence (only). Go to the “Edit” menu, choose “Translate...” and switch “Reading Frame:” to “All Three.” Leave the other parameters as they are and then press “OK” to translate all three forward reading frames. The new protein sequences will appear in your SeqLab Editor display below everything else. Stop codons are represented as asterisks. You should have six sequences total in your Editor display at this point, your genomic DNA sequence, its three forward translations, and the two ‘interesting’ protein search results to compare against those translations. Now we’re ready to start some in-depth pairwise comparisons.

The dot plot

Dot matrix analysis, also known as doing a ‘dot plot,’ 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, which generates the data, and DotPlot, which actually draws the matrix. Unfortunately these are another two of the programs that no longer work under CentOS 5 on HPC. Therefore, we’ll achieve a similar objective with the EMBOSS program DotMatcher. You’ll generate dot plots of all your project genomic query sequence translations to both ‘twilight zone,’ near neighbors (as described above) using these methods.

You’ll run the program six times, comparing each translation frame from your genomic sequence to your two ‘interesting’ database sequences. Start the process by selecting each translated genomic sequence and each new entry, two at a time per program run, in the SeqLab main Editor display. “CUT” and “PASTE” (the ‘pasted’ entry will go right below any other sequence entry name that you have selected) your sequences to move them side-by-side on top of one another so that you can easily select both at the same time (or use <ctrl><right-click> to select non-adjacent entries). In general, put the longer sequence along the horizontal axis of a dot plot by having it first in the SeqLab display. Dot plots just look better that way, though it’s certainly not necessary. Next go to the “Extensions” menu and select “EMBOSS programs” “DotMatcher...” to get the DotMatcher program window. Notice that “X11” is checked by default so that the output will automatically be drawn on your X11 display after the “Run” button is pushed. The program defaults are “23” points within a window of “10” residues. That means wherever the sum of the BLOSUM62 match scores (others can be optionally specified) within the window exceeds 23, a line is drawn along that window. The window is slid over one position at a time, and repeated through all possible diagonals.

As in all windowing algorithms, you want to use a window size approximately the same as the size of the feature that you’re trying to recognize. Unfortunately, in many cases you won’t know what that is. Leave the
window at its default setting of “10” 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.

However, to create the most informative dot plot, rerun the program increasing or decreasing the stringency and/or window size of the comparisons until you get the best balance of signal against noise. Experiment. Remember that subsequent runs after the first can be launched from the “Windows” menu ‘shortcut’ listing. Close and delete each run’s log file with the “Output Manager” after each program run.

My first example compares the *Trypanosoma cruzi* EF-1α genomic frame one translation against Q26878_TRYCR, the *Trypanosoma cruzi* “mucin-like” protein with the $9.3 \times 10^{-1}$ BLAST Expectation score. The default stringency of 23 points within a 10 residue window showed a few small diagonals, but no background noise at all, so I reran it a few times deciding that 10 points within the 10 residue window gave the optimal balance and the clearest picture. My dot plot graphic is quite clean and is shown below:

Running the comparison at an appropriate stringency produces a relatively clean plot with little confusing noise, but ‘just’ enough to see all the signal. You’ll learn that a tremendous amount of bioinformatics is concerned with discerning signal among abundant noise. Three obvious columns of diagonals, indicating strong regions of nested, overlapping, direct repeats, are seen about two thirds of the way through the EF1α frame one sequence, from about residue 210 through 330, to most of the length of the mucin-like protein, around residue 40 through 170. What’s going on here? Remember, this sequence pair’s overall Expectation value was $9.3 \times 10^{-1}$, not very good at all — are these columnar ‘blobs’ on the dot plot significant; do they have anything to do with homology?

The second comparison, the frame two translation against the mucin-like sequence, produced a dot plot with even more columns. I kept my 10 points within a 10 residue window stringency for this comparison. The graphic follows at the top of the next page.
As I said above, columns (or rows) of multiple diagonals in a dot plot always indicate direct repeat sequences. Dot matrix techniques are about the best available for recognizing repeats in biological sequences.

The frame three dot plot at 10 in 10 shows a somewhat similar picture, but with fewer and smaller columns:

How about my other comparisons, the *Trypanosoma cruzi* EF-1α genomic translations against the *Dictyostelium discoideum* Q54NB8_DICDI protein? This one had a FastX Expectation value of $1.5 \times 10^{-1}$, and is supposedly an “RNA-binding region-containing protein (RNP-1).” Does it have any real homology to my Trypanosome sequence? They both can bind nucleotides. I tried the frame one translation first. Again, the default 23 in 10 window threshold seems a bit stringent; 15 in 10 worked best for me. The resulting graphic is shown at the top of the following page.
This time nothing obvious is apparent. A few tiny diagonals hint of some similarities and repeats, but not much seems to be happening here, nothing jumps right out at you. Some diagonals may correspond to nucleotide-binding motifs, but I doubt it. I don’t think that they are anything other than background noise. The dot plots from the other two frames were equally disappointing — just lots of random noise — so I won’t bother showing them. I’m guessing that the $1.5 \times 10^{-1}$ Expectation value seen by FastX between these sequences has nothing to do with homology and is not at all significant.

To contrast these previous examples where there is no obvious alignment, other than repeats, with an example where there is a real alignment, I’ll show you the dot plot from the Trypanosome EF-1α genomic frame one translation to an entry found by FastX with an Expectation value that is only barely better than the Dictyostelium comparison. Here’s that entry from the FastX output:

```
UNIPROT_TREMBL:Q4DJX6_TRYCR    Begin: 81  End: 308
! Q4djx6 trypanosoma cruzi. selenocys...  112   141   164  155.7  0.12
```

This one is another Trypanosome protein, but is supposedly a selenocysteine-tRNA-specific elongation factor. Even though its Expectation value, $1.2 \times 10^{-1}$, is almost the same as the previous example, here the dot plot tells a very different story. That graphic, plotted at 15 points within a 10 residue window, is shown here:
You may have some problem seeing it, but there truly is an alignment here. A gapped diagonal begins at the first position of both sequences and runs through the first two-thirds of the genomic translation and the first half of the selenocysteine-tRNA-specific elongation factor, along the first 300 residues or so of both sequences. This undoubtedly corresponds to the actual translational frame of the Trypanosome genomic sequence encoding an EF-1α domain. There are also lots of scattered, small direct repeats throughout the dot plot indicated by columns and rows of diagonals, but this is most likely just noise. Sometimes interpreting a dot plot can be quite an accomplishment itself — just remember that diagonals are regions of similarity between the two sequences, and any diagonal off the main center line indicates regions that do not correspond in linear placement between the two sequences yet are still similar.

Do any of your dot plots show direct repeats? Do any of your comparisons show frame jumps or surprising similarities where you wouldn’t have expected them based on Expectation scores? These type of observations point out the critical need to go beyond just one type of analysis and investigate any particular question in several different ways. When running your dot plots, take notes of those particular regions in the proteins that have the longest running similarity. Don’t worry about being too accurate with these coordinates, just get them within about 25 residues or so, but we will need these numbers in the next section. Get at least the one best region from each of your two ‘interesting’ comparisons.

**Pairwise dynamic programming alignment algorithms: global versus local**

You need to understand the difference between these algorithms. Global pairwise alignment schemes, *ala* Needleman and Wunsch (1970), align the full length of two sequences, and local pairwise alignment algorithms, usually *ala* Smith and Waterman (1981), or heuristics-based methods thereof, align the ‘best’ local portions of two sequences. Usually programs using these algorithms operate on sequences of the same type, either protein or DNA, but some, such as GCG’s FrameAlign, align DNA to protein, and can be global or local, depending on the options that you set. 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 global alignment is what you want; if you only suspect an area of one is similar to an area of another, then you should use local alignment. To force a local alignment program to be even more local, you can specify a more stringent scoring matrix, such as the PAM30 or BLOSUM90 matrices. If you suspect that a DNA sequencing error is affecting the alignment, then a program like FrameAlign can be used.

BestFit and Gap are the two pairwise alignment programs, local and global respectively, in the GCG package. However, these are two more of those programs that no longer operate under CenOS 5. They both allow you to estimate significance with a Monte Carlo –Randomizations option, as described in the Introduction, but since they no longer work on our system, we won’t be able to use them. Instead we’ll use Pearson’s jumble implementation in version 35 of his FastA package (1998). Previously Pearson had a separate program, PRSS, to do pairwise jumbled comparisons, but with version 35, he incorporated the routine into all of his search programs anytime you compare your query to only one other sequence, and he dropped the separate
PRSS program. We'll use his native implementation of the Smith-Waterman search program SSearch35 for the job. GCG also has an SSearch implementation, but it comes from an earlier version of the FastA package, so it does not have the built-in, pairwise jumble routine. However, unlike the jumble approach described in my introduction, SSearch35 uses the actual extreme value distribution of the jumbles to calculate a real Expectation value, and does not display the type of Z-score statistic described previously. This is good and bad — good because it's based on real statistics, bad because Z-scores are very easy to understand.

Before beginning though, study your dot plot notes from before. This approach works best when applied to local areas where you already know some similarity exists, and you wish to further test that similarity. Otherwise you could be throwing noise into the analysis, and perhaps artificially driving down the significance of the comparison. Therefore, restrict the next set of analyses to those regions of similarity identified by the dot plots. However, remember that dot plots show us all the regions that are similar, whereas dynamic programming only gives us one optimal solution.

Unfortunately SeqLab's Editor will not allow us to choose two different ranges on two different sequences for analysis. Some things are still simpler from the command line! Plus, the native FastA suite requires FastA format input. Therefore, create FastA format sequences of all portions of each sequence you want to further compare. As I said above, do this for at least two different comparisons. However, some weird bug prevents SeqConv+ from working correctly in this fashion — the entire sequence is reformatted, rather than just the designated portion. Therefore, trim your sequences of interest down to just the portions of interest first. If your genomic translation needs to have a couple of different subsequences created, make copies of it first. Make copies by selecting the sequence's name and then pressing the “COPY” and “PASTE” buttons. Next change the protections on all of the sequences by selecting your entire dataset and then pressing the “PROTECT” button, pushing in all of the buttons and accepting “OK.” Now trim sequences down by selecting their name first, then by using the “Edit” menu “Select Range . . .” choice to select the portions that you want to cut away, and then pressing the “CUT” button, designating “Selected regions.” You'll be left with just those portions of sequence that we want to further compare.

Next, select the new partial sequence's name, one in turn, and use “Functions” “Improting/Exporting” “SeqConv+” to create “FastA” format sequences of them. Realize that SeqConv+ does not open the Output Manager window for you when used to create a non-GCG format, so you should use your terminal window to manually change the names of the output files from the default “seqconv+.fa” (or similar) to something more appropriate, e.g. from my last dot plot example above:

```
$ mv seqconv+.fa Q4DJX6_TRYCR.region.fsa
```

We won't need SeqLab any more today in the tutorial. However, you may want to use the “Output Manager” first to delete many of the output files produced during this tutorial, just be sure not to delete your BLAST and FastX search output lists — we will be needing them later on in the semester! After this file cleanup action
exit SeqLab with the “File” menu “Exit” choice; save your RSF and list file with appropriate responses. Designate an RSF file name that make sense to you; SeqLab will close.

Next, use your ssh terminal window to manually launch the SSearch35 program. We’ll need to specify the FastA format sequence files of the regions identified in your the dot plots from the genomic translations and UniProt sequences. I’ll provide sample command lines here for my two ‘best’ comparisons; they’re quite straightforward. Use my examples to come up with your own commands. My examples follow; first the genomic frame 1 translation and selenocysteine-tRNA-specific elongation factor comparison. Specify appropriate output filenames that make sense to you, and tell the program to “Display alignments.” Press <enter/return> to accept the rest of the program defaults:

$ ssearch35 frame1.region2.fsa Q4DJX6_TRYCR.region.fsa

# ssearch35 frame1.region.fsa q4djx6.region.fsa
SSSEARCH searches a sequence data bank
version 35.03 Feb. 18, 2008
Please cite:

Query: frame1.region.fsa
1>>>TRBEF1AE_FRAME1  300 aa - 300 aa
Library: q4djx6.region.fsa     300 residues in     1
300 residues in     1 sequences

300 residues in     1 sequences
Statistics: (shuffled [500]) MLE statistics: Lambda= 0.1702;  K=0.02114
Algorithm: Smith-Waterman (PGopt) (6.0 Mar 2007)
Parameters: BL50 matrix (15:-5), open/ext: -10/-2
Scan time:  0.280
Enter filename for results []: ssearch2.pair
How many scores would you like to see? [20]

The best scores are: s-w bits E(1)
Q4DJX6_TRYCR Q4djx6 trypanosoma cruzi. selenocyste ( 300)  211 57.4 4.8e-13
More scores? [0]
Display alignments also? (y/n) [n] y
number of alignments [20]?

300 residues in 1 query sequences
300 residues in 1 library sequences
Scomplib [35.03]
Function used was SSEARCH [version 35.03 Feb. 18, 2008]

And the genomic frame 1 translation against the mucin-like protein comparison:

\[ ssearch35 \text{ frame1.region1.fsa} \ Q26878\_TRYCR\_region.fsa \]

I won’t show the screen trace from this command. Do this with at least your top two choices so that you end up with a minimum of two SSearch35 output files that align the ‘best’ portions of your genomic translations with those ‘best’ portions of your two ‘interesting’ comparisons.

The SSearch35 output file from my comparison of the first 300 residues of the frame one genomic translation against the first 300 residues of selenocysteine-tRNA-specific elongation factor, is shown below. Notice that the similarity is spread pretty much over the entire length analyzed. This is always a good indication of significance. Also notice Smith-Waterman score, 211, Bit score, 57.4, and Expectation value, \( 4.8 \times 10^{-13} \). This E value is quite low, in stark contrast to its FastX database scan Expectation value of \( 1.2 \times 10^{-1} \). Therefore, the interpretation is that the similarity is extremely significant in spite of the two sequences only having 25% identity. This clearly shows how percent identities and/or similarities can be misleading.

\[ more ssearch2.pair \]

Query: TRBEF1AE\_FRAME1, 300 aa
Library: q4djx6.region.fsa 300 residues in 1 sequences

Statistics: (shuffled [500]) MLE statistics: Lambda= 0.1702; K=0.02114
Algorithm: Smith-Waterman (PGopt) (6.0 Mar 2007)
Parameters: BL50 matrix (15:-5), open/ext: -10/-2

The best scores are:

\[ Q4DJX6\_TRYCR \ Q4djx6 \ trypanosoma \ cruzi. \ selenocysteine \ (300) \ 211 \ 57.4 \ 4.8e-13 \]

\[ s-w \ bits \ E(1) \]

>>Q4DJX6\_TRYCR Q4djx6 trypanosoma cruzi. selenocysteine (300 aa)

Smith-Waterman score: 211; 25.0% identity (54.7% similar) in 316 aa overlap (2-300:3-294)
Contrast those results with my second SSearch35 analysis in which I investigate the significance of those three columns of direct repeats, residues 210 through 330 of the frame one genomic translation, against the full length of the mucin-like protein. Do you have any preconceived idea as to whether any of these repeats are significant? Remember, the FastX database search Expectation value of this alignment was only $9.3 \times 10^{-1}$. What’s its pairwise SSearch35 Expectation value going to be? The results follow:

```
ssearch35 trbef1ae.region1.fsa q26878_trycr.fsa
SSearch searches a sequence data bank version 35.03 Feb. 18, 2008
Please cite:
Query: TRBEF1AE_REGION1, 121 aa
<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Library:</td>
</tr>
<tr>
<td>Library:</td>
</tr>
<tr>
<td>Library:</td>
</tr>
<tr>
<td>Library:</td>
</tr>
<tr>
<td>Library:</td>
</tr>
<tr>
<td>Library:</td>
</tr>
<tr>
<td>Library:</td>
</tr>
<tr>
<td>Library:</td>
</tr>
<tr>
<td>Library:</td>
</tr>
<tr>
<td>Library:</td>
</tr>
<tr>
<td>Library:</td>
</tr>
<tr>
<td>Library:</td>
</tr>
<tr>
<td>Library:</td>
</tr>
</tbody>
</table>
```

300 residues in 1 query sequences
300 residues in 1 library sequences
Scomplib [35.03]
Total Scan time: 0.280 Total Display time: 0.010

Function used was SSEARCH [version 35.03 Feb. 18, 2008]
This alignment’s percent identity, 21%, isn’t a whole lot worse than the previous one, but it is localized to a very small portion of the length specified. That is because the algorithm only found the one best alignment of the many repeats in the region. Furthermore, its pairwise SSear35 Expectation value is 3.4 x 10⁻¹, way worse than the previous, and pretty much near the bottom of Doolittle’s ‘Twilight Zone.’ What’s up with that?

It agrees with the FastX Expectation value of 9.3 x 10⁻¹ though, and it reinforces the notion that percent identities or similarities don’t really mean anything as far as significance goes.

Often a seemingly decent alignment will not be very significant upon further inspection — do not blindly accept the output of any computer program! Comparisons can turn out to be insignificant in spite of what seems to be, upon first inspection, a very good alignment and a pretty high percent identity. Therefore, always investigate further for similarities can be strictly artifactual, here an abundance of proline, proline, multi-tryptophan repeats in this region in the mucin-like sequence. A Monte Carlo style test is often prudent and may suggest that 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 GCG outside of BLAST for prefiltering your protein sequences. This can be particularly valuable 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.

As described in the beginning of this section, if you suspect a frame shift sequencing error in a DNA sequence being considered, a very powerful pairwise alignment program, FrameAlign, is available, but we will not be running it here. The lab’s been long enough as it is! FrameAlign uses dynamic programming to align a protein to a DNA sequence with the allowance of frame shifts, though it does not support the randomization
option that Gap and BestFit do. Frame shifts appear in the output alignment as gaps that are not multiples of three.

Log out of your current UNIX session on HPC and off the workstation that you are sitting at.

**Homework assignment**

Use the report form to tell me the name of each of the (at least) two ‘interesting’ entries that you decided to further evaluate that your project genomic sequence ‘discovered’ with FastX, and then describe some of the results of those evaluations. These results include:

1) The Expectation score of each entry aligned to your project genomic sequence as reported by FastX. Remember, these sequences should be from near the bottom of your search list, not from the top of the list. I know how two nearly identical sequences align!

2) Which translational frames and approximate regions of your project genomic sequence and of the ‘interesting’ comparison sequences produced the most favorable alignments as seen with dot matrix analysis?

3) Of those most favorable alignments seen in your dot matrix analyses, how many did you analyze further with SSearch35 Monte Carlo randomizations and what were the Expectation values from those SSearch35 analyses? What is your conclusion as to whether or not they exhibit ‘real’ homology?

**Conclusion**

This has been a very long tutorial; I apologize for that. However, sequence database similarity searching is one of the most commonly misunderstood areas in computational molecular biology and bioinformatics. There is a tremendous amount of confusion in the field and anything that can be done to try and clear up some of the mess is entirely worthwhile. One point that remains to be made is that the previous techniques were performed largely using default parameters. 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**


