Module 3 Content

Introduction

Topics to be covered:

- BLAST
- Multiple Sequence Alignment
- NCBI eUtils
- BLAST searching with Perl

Before we start, some vocabulary (from the optional Pevsner text):

- **Homologous** – "two sequences are homologous if they share a common evolutionary ancestry…sequences are either homologous or not."
- **Identity** – a quantitative value that describes the relatedness of sequences, namely how many amino acid residues or nucleotides are the same.
- **Similarity** – also a quantitative value that describes the relatedness of sequences -- here, how many have similar biochemical properties, and are structurally or functionally related, such as serine (S) and threonine (T).

BLAST

Another way (besides an Entrez query, for example) to obtain information from a sequence database is to do a sequence search. We will look at using BLAST and its variants to take a sequence as a query and identify (potentially) relevant sequences in a database. Here, relevant usually means "evolutionarily related", or in other words, "homologous."

In a nutshell, BLAST is a way to take a protein or nucleotide sequence and search a protein or nucleotide sequence database – not by keyword search, or simple character matching, or with SQL, but, and this is an important point, a different kind of search – one that is by sequence similarity that suggests (but does not prove) evolutionary relatedness. (Maybe for your course project you will need to design a new type of search to best work with your data.)

Before we begin using BLAST, we need to look at scoring matrices, which BLAST uses to find sequences to return in response to a query sequence.

Background

Scoring matrices. These matrices model evolutionary relatedness, which allows the sequence search to find not just an exact match to the sequence query but similar sequences also. This is a very important concept – it determines whether it is reasonable for an amino acid residue "L" to turn into an "I", and if so, what number can be assigned to the change to represent the likelihood of the change and the closeness of the relationship.

PAM and BLOSUM matrices. For protein searching, two important scoring matrix families are PAM and BLOSUM. As discussed in the optional Pevsner text, the PAM matrices are based on manually evaluating closely related proteins and determining how likely one amino acid changes into another (by Dayhoff and colleagues back in the 1970s). These changes were called "accepted point mutations" as in changes that were accepted by evolution, and the "likeliness" is the mutation probability.(Essentially, the number of

substitutions between a pair of amino acid residues (the letters) are counted, turned into mutation frequencies, and finally into log odds compared to background occurrences.)

A PAM1 matrix gives the probabilities of mutations when proteins have undergone a 1% change per 100 amino acids. A PAM1 matrix is appropriate for comparing very closely related sequences. PAM250 is created by multiplying PAM1 by itself 250 times (therefore not through direct observation and comparison of distantly related proteins, but computationally), and is intended for comparing more distantly related sequences. Note that the numbers in the scoring matrices used for BLAST searches are log odds, not probabilities. (Recall odds = (probability/(1-probability)).) This statement applies to both PAM and BLOSUM.

BLOSUM matrices were developed in the 1990s (by Henikoff and Henikoff) and were generated from "blocks", or groups of local multiple sequence alignments. Different BLOSUM matrices are based on different percentage identities of the sequences in the alignment. Sequences in the block that are of a greater percentage identity get grouped for the purpose of counting substitution occurrences. Sequences in such a group only contribute a fractional count toward the number of occurrences.

BLOSUM62, for example, was derived from sequences with 62% identity. This scoring matrix has been shown to perform very well for detecting distantly related proteins. BLOSUM80 would be more appropriate for more closely related sequences, while BLOSUM45 would be better for more divergent.

The following is the upper left corner of the BLOSUM62 matrix at https://www.ncbi.nlm.nih.gov/Class/FieldGuide/BLOSUM62.txt :

	A	R	N	D	C	Q	E	G	H	I	L	K
A	4	-1	-2	-2	0	-1	-1	0	-2	-1	-1	-1
R	-1	5	0	-2	-3	1	0	-2	0	-3	-2	2
N	-2	0	6	1	-3	0	0	0	1	-3	-3	0
D	-2	-2	1	6	-3	0	2	-1	-1	-3	-4	-1
C	0	-3	-3	-3	9	-3	-4	-3	-3	-1	-1	-3
Q	-1	1	0	0	-3	5	2	-2	0	-3	-2	1
E	-1	0	0	2	-4	2	5	-2	0	-3	-3	1
G	0	-2	0	-1	-3	-2	-2	6	-2	-4	-4	-2
H	-2	0	1	-1	-3	0	0	-2	8	-3	-3	-1
I	-1	-3	-3	-3	-1	-3	-3	-4	-3	4	2	-3
L	-1	-2	-3	-4	-1	-2	-3	-4	-3	2	4	-2
K	-1	2	0	-1	-3	1	1	-2	-1	-3	-2	5

These matrices are normally symmetric about the diagonal (upper left to lower right) so you will often only see half the matrix (a triangle) given. Notice the positive scores going down the diagonal which is the score assigned to keeping an amino acid residue unchanged. Most substitutions are negative, although not all. The larger the negative number, the more penalized the substitution is in the scoring.

Depending on the distance of the relationship you are interested in between your protein query and matching protein sequences, you would choose a different scoring matrix. NCBI has this page (https://www.ncbi.nlm.nih.gov/blast/html/sub_matrix.html) with some suggestions.

Exercise: Question – if you want to find sequences more evolutionarily distant to your query, which scoring matrix would you choose between these two: BLOSUM90 or BLOSUM45? BLOSUM90 or PAM250?

The combination of scoring matrices and sequence search is analogous to using SQL to query a relational database or to using text search to query a literature database, but it provides **the appropriate framework for asking questions that deal specifically with sequences that the other query technologies do not**. When you do your course project, consider what the best way is to allow your users to answer their questions. A hybrid approach (for example, doing a BLAST search with a text query) is also a possibility. Or providing multiple approaches (both querying and browsing) can be appropriate.

(If you would like more information, take a look at this article, “An introduction to sequence similarity (‘homology’) searching”. (Pearson, W. R. (2013). https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3820096/ .)

BLASTing and Alignment

Visit the BLAST page (https://blast.ncbi.nlm.nih.gov/Blast.cgi), and follow along below.

On the BLAST page, you'll find the different kinds of BLAST searching available at NCBI. Notice that searching on proteins and nucleic acid sequences are separate. But all of them will attempt to find similar sequences to a query sequence you enter. In the process of finding similar sequences, BLAST will produce a local sequence alignment.

Sequence alignment attempts to align two or more sequences (nucleotide or protein) such that regions of structural or functional similarity between the molecules are highlighted. By asking whether our unknown sequence is in any way similar to known sequences (and ideally to sequences of known function or structure) we can possibly identify the unknown sequence and predict its structure and function. We can also get an idea of evolutionary relatedness.

It should be pointed out that any two strings can be aligned — that is, oriented so that their common regions are matched. This is called the longest common subsequence problem, and is typically solved using a dynamic programming algorithm. The two lines below are aligned (a “global” alignment), using the arbitrary rule to match identical letters and that a letter that doesn't match is put with its next occurring, closest letter in the alphabet if possible in the other sequence without rearranging any letters. There is no scoring matrix or evolutionary relatedness here. There are likely other alignments that may be as good as this one. Some type of scoring would need to be given to the mismatches and gaps to determine which alignment is the "best". And we should have a significance indicator for the alignment, but we don’t.

```
bioinformatics databa ses
bio l o g y        basics
```

There are two common forms of sequence alignment. Global alignment (not what BLAST does) attempts to align two sequences in their entirety. The results of such a long alignment might be displayed as a dotplot. The other is a local alignment — only looking for a segment (such as a protein domain occurring in the sequence query and in the sequence database) that scores well, rather than trying to align the entire stretch of the sequences.

As mentioned above, BLAST performs a local alignment and uses scoring matrices, which have biological significance, representing evolutionary relatedness by giving a log odds score of how likely it is that one amino acid residue evolved into another one. Because using dynamic programming to align a query sequence to all sequences in a sequence database is computationally infeasible, BLAST uses a heuristic to accomplish its task.

For more information, read the following algorithm discussion about BLAST http://en.wikipedia.org/wiki/BLAST (Algorithm section only). Briefly, the following steps are performed:

- Split the query sequence into words of length W (typically 3 for proteins). For example, if a query sequence is IKLM, the words would be IKL and KLM. The value of W can be changed in the algorithm parameters.
- Identify all high scoring words that match these query words, using the selected scoring matrix to do the scoring (might be LKL, VKL, etc).
- Lookup the high scoring words to see if they are in sequences in the database.
- When a match is found in a database sequence, extend the alignment in both directions until the score drops below a threshold (configurable). This is a high-scoring segment pair (HSP).
- Find all the HSPs and determine their statistical significance.
- Report the results (the hits) and the alignments found

On the NCBI BLAST page, note that there are several major kinds of basic BLAST search. Nucleotide BLAST and Protein BLAST both directly search a sequence database that is the same type as the query (nucleotide → nucleotide, and protein → protein, respectively). Some of the other searches involve a translation.

- Blastx takes a nucleotide query and translates it into 6 different protein sequences (using all 6 reading frames of the nucleotide sequence), then searches a protein sequence database with each of the 6 protein sequence queries.
- Tblastn takes a protein sequence query and compares it to a *translated* nucleotide sequence database, where the each sequence in database has been translated into 6 different protein sequences based on the 6 different reading frames.
- Another kind, Tblastx, translates a nucleotide query into 6 protein sequences and translates each sequence in a nucleotide sequence database into 6 protein sequences, then does a protein-to-protein sequence comparison between all of these. This search is very compute intensive and is only allowed on one genome at a time. It is no longer a choice on the main BLAST page.
- There are also specialized BLASTs, set up to do well for particular use cases.

Exercise: Let's get familiar with BLAST (if you are not already) by doing a search. Choose the protein BLAST search (**blastp**), paste in the sequence below (a human Cartilage Oligomeric Matrix Protein) and search the nonredundant database (nr) by selecting nr in the database field of the form (if it is not already selected as the default). The BLAST button at the bottom will start your search. Depending on how busy the system is, the search could take a while.

```
>Example sequence
MVPDTACVLL LTLAALGASG QGQSPLGSDL GPQMLRELQE TNAALQDVRD WLRQQVREIT
FLKNTVMECD ACGMQQSVRT GLPSVRPLLH CAPGFCFPGV ACIQTESGGR CGPCPAGFTG
NGSHCTDVNE CNAHPCFPRV RCINTSPGFR CEACPPGYSG PTHQGVGLAF AKANKQVCTD
INECETGQHN CVPNSVCINT RGSFQCGPCQ PGFVGDQASG CQRGAQRFCP DGSPSECHEH
ADCVLERDGS RSCVCRVGWA GNGILCGRDT DLDGFPDEKL RCPEPQCRKD NCVTVPNSGQ
EDVDRDGIGD ACDPDADGDG VPNEKDNCPL VRNPDQRNTD EDKWGDACDN CRSQKNDDQK
DTDQDGRGDA CDDDIDGDRI RNQADNCPRV PNSDQKDSDG DGIGDACDNC PQKSNPDQAD
VDHDFVGDAC DSDQDQDGDG HQDSRDNCPT VPNSAQEDSD HDGQGDACDD DDDNDGVPDS
RDNCRLVPNP GQEDADRDGV GDVCQDDFDA DKVVDKIDVC PENAEVTLTD FRAFQTVVLD
PEGDAQIDPN WVVLNQGREI VQTMNSDPGL AVGYTAFNGV DFEGTFHVNT VTDDDYAGFI
FGYQDSSSFY VVMWKQMEQT YWQANPFRAV AEPGIQLKAV KSSTGPGEQL RNALWHTGDT
ESQVRLLWKD PRNVGWKDKK SYRWFLQHRP QVGYIRVRFY EGPELVADSN VVLDTTMRGG
RLGVFCFSQE NIIWANLRYR CNDTIPEDYE THQLRQA
```

(Note: the above sequence is formatted as a FASTA sequence. In most programs that accept sequences, you can input a FASTA sequence. When doing the BLAST search, copy the entire example above, including the > line (which will be used to identify the sequence). Note: FASTA is the name of a sequence alignment algorithm, as well as a format.)

When your results are finally ready, you'll get a summary page -- notice at the top there is information about the search you just requested and several tabs.

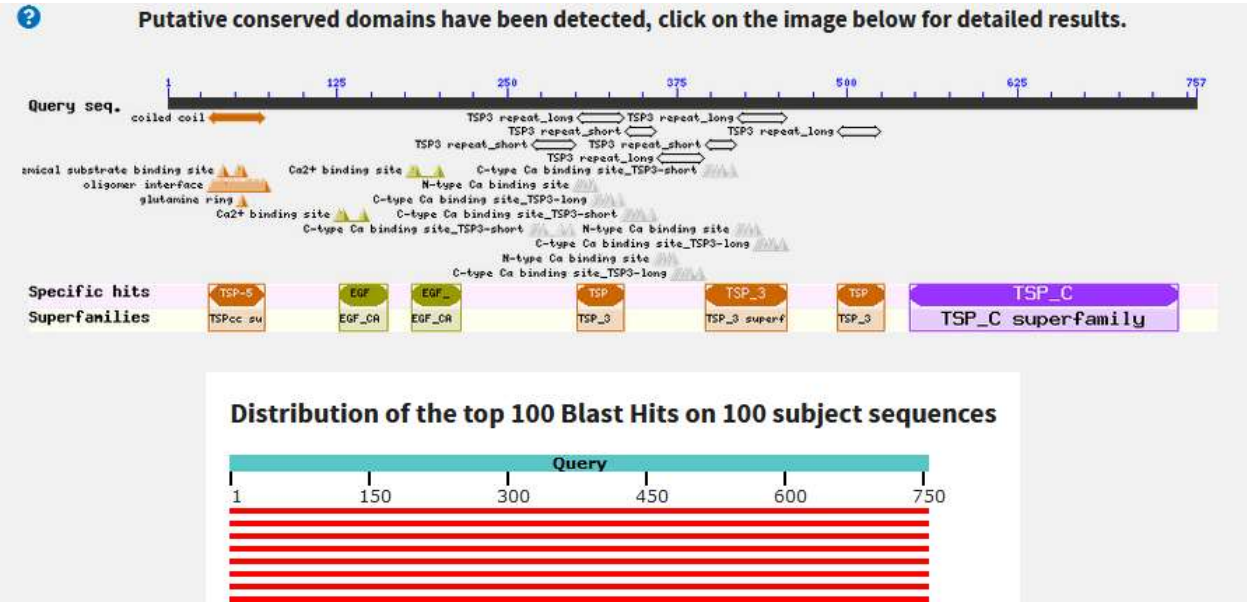
To see the alignments between the query sequence and the database sequence, click on the **Alignments** tab and scroll down to view the results. There are scores associated with the sequences matched -- of particular importance is the e-value (more about this). You will see the alignments between the query sequence and each of the sequences found by the BLAST search. In this example, the initial alignments are for sequences that are almost 100% identical. The following shows part of one alignment where some letters did not match:

Range 1: 1 to 757 GenPept Graphics ▼ Next Match ▲ Previous Match

Score	Expect	Method	Identities	Positives	Gaps
1503 bits(3892)	0.0	Compositional matrix adjust.	737/757(97%)	745/757(98%)	0/757(0%)

```
Query  1    MVPDTACVLLLTLAALGASGQGQSPLGSDLGPQMLRELQETNAALQDVRDWLRQQVREIT  60
            M PDTACVLLLTLAALGASGQ Q P GSDLGPQMLRELQETNAALQDVR+ LRQQVREIT
Sbjct  1    MAPDTACVLLLTLAALGASGQSQIPSGSDLGPQMLRELQETNAALQDVRELLRQQVREIT  60
```

To see information about conserved protein domains in the sequence, see the **Graphic Summary**. The following show the top of a Graphic Summary tab page.



In addition, click on the **Taxonomy** tab. There you will see the species that are represented in the results of the blastp search. The following shows the beginning of an example Taxonomy tab.

Organism	Blast Name	Score	Number of Hits
root			124
. Eutheria	placentals		120
. . Boreoeutheria	placentals		119
. . . Euarchontoglires	placentals		71
. . . . Primates	primates		50
. . . . . Simiiformes	primates		47
. . . . . Catarrhini	primates		42
. . . . . Hominoidea	primates		24
. . . . . Hominidae	primates		22
. . . . . Homininae	primates		21
. . . . . Homo sapiens	primates	1540	15
. . . . . Pan paniscus	primates	1522	1
. . . . . Pan troglodytes	primates	1521	3
. . . . . Pan troglodytes verus	primates	1520	1
. . . . . Gorilla gorilla gorilla	primates	1456	1
. . . . . Pongo abelii	primates	1511	1

Try several of the other BLAST options for a protein query and use the same sequence. On the BLAST search page you can see additional parameters by clicking the `+ Algorithm Parameters` at the bottom of the page below the BLAST button.

Also note on the blastp search page that you can select comparing 2 sequences ("Align two or more sequences"). There are a number of uses for comparing two sequences, for example, finding mutations in two closely related sequences or a common domain shared by two sequences.

More about BLAST

You should understand what scoring matrices are when to use them, examples of different uses for BLAST searching, and to know what HSPs and the e-value are. For e-value you need to know how to use it to make sense of what matches might be significant. You do not need to know the math, statistics, or details of the algorithm, just the high-level information to make use of BLAST.

This video is a good introduction about Blast Expect values (part 1):
http://www.youtube.com/watch?v=nO0wJgZRZJs&list=PL8FD4CC12DABD6B39

And this is the rest about Blast expect values (part 2):
http://www.youtube.com/watch?v=Z7ek7UoP7Bg&list=PL8FD4CC12DABD6B39

The following is derived from the information at the NCBI website and contains some points I want to make sure you are aware of.

-→If an alignment requires gaps in one sequence or the other, a gap penalty will be applied that lowers the score. This consists of an opening gap penalty and an extending-the-gap penalty (which is less than the opening one): G = O + kN, where O is the opening gap penalty, N is the extending the gap penalty, and k is the length of the gap minus 1. This type of gap penalty is called an "**affine gap penalty**". An example of a gap is shown below – it is the hyphen in one sequence that skips matching a letter in the other sequence:

```
VTISCTGSSSNIGAG-NHVKWYQQL
PEVTCVVVDVSHEDPQVKFNWYVDG
```

→There are two types of scores – raw and bit scores. Raw scores are derived from the scoring matrix and the gap penalties applied, if any. Bit scores are normalized to account for the search space and scoring system.

→The statistical significance of a search is a quantitative measure of whether a particular alignment is relevant or occurred simply by chance. Local alignment scores fall into an extreme value distribution. The expected number of HSPs having a score S or better by chance is given by the equation:

$E = kmNe^{(-lambda*S)}$. (You don't need to know this equation.)

So if an alignment between a query sequence and a sequence in the database has a given score (say a bits score of 63) and an E value of $1x10^{-3}$, this means that in a database of the size of the database just searched, there would be $1x10^{-3}$ matches of the same score (63) or better by chance. Or, in other words, a score of 63 or better is expected to occur by chance 1 in $1x10^{-3}$ times in a database of this size. From the equation, a high score corresponds to a low E value and a better alignment. As E approaches 0, the probability that the alignment occurred by chance approaches 0.

Probability is related to E thusly: $P = 1 - e^{-E}$. Often in an experiment, a P value less than 0.05 is used as a cutoff to define statistical significance, but with BLAST searches **the size of the database is important**. As the number of sequences in the database increases, the chance of a random match increases, and the cutoff value must decrease. There are various guidelines for what are good E-values; they are in the realm of $1x10^{-3}$ to $1x10^{-8}$.

→ Statistical similarity is **not** enough to definitely prove or determine biological relevance (it merely suggests it). The researcher needs to apply biological criteria when evaluating an alignment. For example, the 3D structures can be compared, and the sequences can be examined in a multiple alignment. Also, if you have an alignment to a sequence that has a high E-value and you suspect that it is really a false negative, you can take this sequence and do a BLAST search with it. If this new search returns sequences with good scores and low E-values that are relevant matches, this will support the idea that the sequence with the high E-value is homologous.

PSI-BLAST

As we've just seen, BLAST is designed to use a sequence as a query, scoring matrices to provide a measure of evolutionary distance for position changes, and a scoring system for returning likely related sequences from a sequence database. However, there are sequences that are not found by the "regular"

BLASTs. There are a number of specialized BLAST capabilities and other techniques to address this situation. You are responsible for PSI-BLAST -- you can find it on the regular protein BLAST page under the Algorithm choices.

PSI-BLAST is designed to find more distantly related proteins than blastp. It begins with a blastp search, allows the user to select good matches from the result list, and aligns those proteins (see multiple sequence alignment below) to determine a position-specific scoring matrix (PSSM) specifically for *this query*. In building the PSSM, it considers what residues are found at each position and how frequently. It will also determine if there is an invariant residue at any position. Consequently, like a regular scoring matrix, a PSSM is a matrix that indicates how likely it is for one amino acid residue to be replaced by another residue. But the PSSM is tailored to results from the query sequence, compared to the standard scoring matrices, which are based on a set of proteins with a particular level of relatedness that were analyzed to create the scoring matrix. The PSSM's size is the length of the query by 20.

PSI-BLAST is not perfect, however; its main drawback is that it sometimes latches on to spurious alignments, which causes incorrect substitution values to go in the PSSM, and the search returns hits based on this incorrect association that should not be returned.

Exercise: Go to NCBI's Protein BLAST page and run a PSI-Blast search with protein NP_006735 (choose PSI-BLAST as the Algorithm option). The first step in PSI-BLAST is just a regular protein BLAST. In the results, notice that all the proteins above the E-value threshold are checked. In this case, as of this writing, there weren't any that were below the threshold. If there are, any of these can be checked and added to the pool for calculating the PSSM if you suspect the protein is really related. You can also uncheck any sequences you think will not improve the search. Below the list of results click on the Go button to do the next iteration. Now the PSSM will be constructed and used to find more sequences. In the new results list, newly found sequences are highlighted in yellow. You can do additional iterations. Eventually the search will not find new sequences to add (it converges). Or, if the results begin to collect non-relevant results (the drawback mentioned above), the search may not converge. (You can stop at this point or try a few more iterations to see if/how the results list changes.)

Multiple Sequence Alignment

When we consider a gene or protein, one of the most fundamental questions is what other genes or proteins are related. Biological sequences often occur in families:

Paralogs — related (homologous) genes within an organism

Orthologs — related (homologous) genes in other species

Up to now, we have considered pairwise alignments and searches. In this section we will consider "multiple alignments".

Some time ago I found a site that had a nice introductory statement about multiple alignments (it was at www.techfak.uni-bielefeld.de, but is no longer there):

> "What is a multiple alignment? The short answer is this -
>
> ```
> VTISCTGSSSNIGAG-NHVKWYQQLPG
> VTISCTGTSSNIGS--ITVNWYQQLPG
> LRLSCSSSGFIFSS--YAMYWVRQAPG
> ```

```
LSLTCTVSGTSFDD--YYSTWVRQPPG
PEVTCVVVDVSHEDPQVKFNWYVDG--
ATLVCLISDFYPGA--VTVAWKADS--
AALGCLVKDYFPEP--VTVSWNSG---
VSLTCLVKGFYPSD--IAVEWESNG--
```

> That's a multiple alignment. 8 fragments from immunoglobulin sequences are displayed together. Their alignment highlights conserved residues (one of the cysteines forming the disulphide bridges, and the tryptophan are notable), conserved regions (in particular, "Q.PG" at the end of the first 4 sequences), and more sophisticated patterns, like the dominance of hydrophobic residues at fragment positions 1 and 3. The alternating hydrophobicity pattern is typical for the surface beta-strand at the beginning of each fragment. Indeed, multiple alignments are helpful for protein structure prediction.
>
> The alignment can also enable us to infer the evolutionary history of the sequences. It looks like the first 4 sequences and the last 4 sequences are derived from 2 different common ancestors, that in turn derived from a "root" ancestor. Indeed, we've got 4 fragments from the so-called variable regions, and 4 fragments from the constant regions of immunoglobulins. "

There is a good introductory bioinformatics text, “Bioinformatics and Functional Genomics”, by Jonathan Pevsner. The following are some of the reasons given in that textbook for why one would want to do a multiple sequence alignment:

1. If a protein is related to a group of proteins, the properties of the group may give insight into the structure or function of the protein.
2. Protein families have distantly related members. Multiple alignment is more sensitive than pairwise alignment in identifying these.
3. Multiple alignment of BLAST search output is useful in identifying motifs or domains. We'll talk more about motifs and domains in a later module.
4. Multiple alignment can show if there are any discrepancies in cDNA libraries.
5. When a complete genome is sequenced, analysis is needed to determine the protein families of each gene.
6. And, of particular relevance for a database class, multiple sequence alignments are used to define or organize the data in a number of databases.
7. Also, as mentioned above, alignments can give insight into evolutionary relatedness.

There are several categories of multiple sequence alignments. For this course, we are primarily interested in the fact that multiple sequence alignments are used in constructing some databases, rather than in the actual algorithms. Multiple sequence alignment is an example of how a specialized algorithm can be used in the creation of data for a database. But here is a brief explanation of different, general, approaches to multiple sequence alignments from the Pevsner text. You are responsible for the explanation but any links in #1-5 below are **optional**.

1. **Exact Approaches**. These use dynamic programming on an N-dimensional matrix (N is the number of sequences). While this approach can identify the optimal alignment in theory, it takes much too long to run. Pevsner gives the computational complexity as $O(2^N * L^N)$ for N sequences of length L. 2^N quickly becomes huge as N grows and L^N is even larger. This type of calculation could easily take more time than the lifetime of the universe with a large enough N and L. Reminder: “Big Oh” notation is used to describe the computational or size growth rate of an algorithm. Although you can’t get a specific time (for example) for an algorithm because you need to know the speed of the processor the algorithm would run on, you can compare algorithms. For example, if one algorithm’s computational complexity is O(n * logn), it will have better performance than one that is $O(n^2)$ (all else being equal). This notation can also let you know if an algorithm will scale well and is, therefore, a good candidate to use with large data. Any time you

see exponents (especially when a number is raised to the power of N or if the exponent is 2 or more), the algorithm is not scalable to very large datasets.

2. **Progressive Sequence Alignment**. Do a pairwise alignment between all sequences. Take the closest pair. Add the sequence that is closest to this pair. Add the next, etc., progressing along until all sequences have been accounted for. ClustalW (http://www.clustal.org/clustal2/) is an older program that performs this type of alignment. One problem with progressive methods occurs if an error is made early in the process. There is no way to go back and say, hey, this sequence alignment would have been a better choice back then now that M sequences have been considered.
3. **Iterative Approaches**. These algorithms attempt to correct the problem with progressive methods by beginning with a progressive approach but allowing for modifications of the alignment along the way. An iterative method may take an alignment determined with a progressive step and modify it to try to improve the score by such actions as adjusting gaps or realigning some sequences. MAFFT (http://www.ebi.ac.uk/Tools/msa/mafft/) is one iterative approach; MUSCLE (http://www.ebi.ac.uk/Tools/msa/muscle/) is another.
4. **Consistency-Based Approaches**. These methods rely on a notion of consistency across multiple sequences. Generally, the idea is that if sequences A and B align at positions A2 and B7, and if sequences B and C align at positions B7 and C3, then A2 and C3 should also align. ProbCons (http://probcons.stanford.edu/about.html) is one example. Essentially a posterior probability is calculated for an alignment given the prior knowledge of what currently is aligned. T-Coffee (http://www.ebi.ac.uk/Tools/msa/tcoffee/) uses a different approach. After calculating the pairwise alignments, it figures out the best 10 and weights all pairs based on the best. These weights are used to create a substitution matrix that shows what the more favored substitutions are (a type of consistency). T-Coffee then combines several alignment techniques to ultimately derive the final alignment.
5. **Structure-Based Approaches**. Because structure diverges more slowly than sequence, structural data (such as that from PDB) can be used. One example is the APDB-iRMSD server at T-Coffee.org (http://tcoffee.crg.cat/apps/tcoffee/do:irmsd). This server analyzes PDBs (the APDB) considering the root mean square deviation (RMSD) of the alpha carbons along the protein backbone to measure alignment goodness.

As you can see, there are many programs (and more that weren't mentioned) that generate multiple alignments (and some of the algorithms can blend more than one category together, so they may not all fit neatly into one category). Often, these programs produce quite different results. It is reasonable to pick one for regular use. However, you will likely have to experiment and adjust parameters to optimize the alignments.

There are several categories of errors associated with multiple alignments. These include:

- False negatives: performing an alignment that omits true homologs
- False positives: adding family members that are not authentic
- Alignment errors: failing to match distant subgroups
- Alignment errors: adding gaps improperly

Alignment errors may result in alignments that are meaningless biologically. In this situation, change the alignment parameters — try adjusting the gap creation and/or gap extension penalties — for example, and re-align the sequences.

Given the fact multiple sequence alignments are used to create data in databases and the other purposes identified above, you should have exposure to creating a multiple sequence alignment yourself.

Exercise: take the sequences text file uploaded to this module (mod03_sequences_for_alignment.txt) (or obtain your own protein sequences) and run a multiple alignment with one of the successors to ClustalW,

namely Clustal Omega (https://www.ebi.ac.uk/Tools/msa/clustalo/). It should only take a moment to run. Once it finishes, click on the Phylogenetic Tree tab. Which species is closest to Hummingbird? How is Bat related to the other species in this input? Are you surprised by any grouping?

(If you would like more information see: “Fast, scalable generation of high-quality protein sequence alignments using Clustal Omega”, http://msb.embopress.org/content/7/1/539). Optionally, try some of the other clustering programs.

NCBI eUtils

We return to using programmatic access to obtain data, expanding upon what we covered in the last module. eUtils is the web-based access for Entrez and consists of a suite of programs that lets you get information from NCBI using URLs.

In general, a URL can be to a script or program instead of to a static page. When you go to that URL, the program runs and it might access a database, read files, perform calculations, or do all sorts of activities before it creates a page to display in your browser or to return to your program.

As we saw last module, the Perl module LWP allows you to do GET and POST to obtain information. The URL you want to access is passed into the GET or POST, along with any parameters.

Aside: The LWP package contains a number of different modules that you may or may not want to use. For example, in addition to LWP::Simple, there is a more complex agent to access websites — LWP::UserAgent; and a robot agent in case you want to write a script that acts like a crawler — LWP::RobotUA.

As you should know by now, after you've told LWP to go to the URL of your choice, you get the response. The format of the response will depend upon how the website you went to structures the data and/or any requests you put in the URL to tell the website how you want the data (if the website lets you select a format).

You can process your results in different ways. You can use a regular expression; or the Perl functions index, substr, etc.; or an XML parser package such as we used last time.

I'll walk you through one of the examples given at NCBI (written by Oleg Khovayko). This example uses the procedural form of Perl, not the object-oriented we saw last module. (The program has been included in the zip mod03_example_files.zip that you can download from the Module.)

use LWP::Simple;

```
my $utils  = "https://www.ncbi.nlm.nih.gov/entrez/eutils";
my $db     = ask_user("Database", "Pubmed");
my $query  = ask_user("Query",    "zanzibar");
my $report = ask_user("Report",   "abstract");
```

This script uses a separate subroutine to get the values of the variables. You could just hard code these:

```
my $db = "Pubmed";
my $query = "zanzibar";
```

```
my $report = "abstract";
```

Now the code sets up the full URL. The NCBI script that will be called is esearch.fcgi, and it gets the beginning part of the URL from the previously set variable $utils. The parameters are db, retmax, usehistory, and term. A ? sets the parameters apart from the script and & separates the parameters. In the next statement the URL is not quite finished because term isn't set to anything yet.

```
my $esearch = "$utils/esearch.fcgi?" .
             "db=$db&retmax=1&usehistory=y&term";
```

Perl reminder: scalar variables have the $ prefix, and if you use the variable in a string, Perl will substitute the variable's value for the variable itself. So "db=$db" will actually be "db=PubMed". Also period (.) is the string concatenation operator.

The results are returned in the next statement. Note that the script author has chosen to finish off the URL by concatenating the query to the rest of the URL in the function call. He could have put $query in the above statement where $esearch was set, after term=.

```
my $esearch_result = get($esearch . $query);
```

When you download and execute the script, you will see the XML returned for this esearch request (text is also returned when abstracts are efetched). Here is some of it when the script was executed:

```
<eSearchResult>
        <Count>6230</Count>
        <RetMax>1</RetMax>
        <RetStart>0</RetStart>
        <QueryKey>1</QueryKey>
        <WebEnv>00i6JMeEeYlXeX-
T3vq3aKPyVGUyGHU_3nXmDvWJJKgIp04TzCKjo97y_pzA9@1EDE1E9484A94D00_0023SID</WebE
nv>
        <IdList>
                <Id>18536052</Id>
        </IdList>
```

There is a lot more not shown. The example script uses a regular expression to extract the count (it does not parse the XML as we did last week but treats it as a string), the query key, and the WebEnv, thusly:

```
$esearch_result =~
         m|<Count>(\d+)</Count>.*<QueryKey>(\d+)</QueryKey>
         .*<WebEnv>(\S+)</WebEnv>|s;
```

From your reading about Perl regular expressions last week, the above should make sense. But to review, this will find the string <Count>, then grab 1 or more numeric digits (\d+) until it sees </Count>, then skip 0 or more characters until it matches <QueryKey>. Again it will grab 1 or more numeric digits until it matches </QueryKey> and will skip 0 or more characters until it matches <WebEnv>. Finally, it will match 1 or more non-space characters (\S+) until it matches </WebEnv>. By using parentheses in the regular expression, the matched characters will be saved in the variables, $1 (for the first parenthesized item), $2 for the second, etc. You should be able to read regular expressions like this and recognize Perl's character classes for digits, etc.

Recall the following:

- \d means match a digit

- + means match one or more (so \d+ means match one or more digits)
- . means match any character
- * mean match 0 or more (so .* means match 0 or more any characters)
- \S means non-space character (note it is a capital S)

So now this information can be saved in more user-friendly variable names:

```
my $Count    = $1;
my $QueryKey = $2;
my $WebEnv   = $3;
```

This particular example has enabled the user history (usehistory=y in the parameter list to esearch.fcgi), so we can continue to get information from it, one page at a time (paging through results is a common practice). In this case, we're going to pull back abstracts we just got from our esearch query. The QueryKey and WebEnv will be used by NCBI keep track of our query from the many it receives in order to give us our information and not someone else's.

➔ Note that the search results only tell us that results were found and what the identifiers of those results are.

To obtain those results requires fetching. The easiest way to get the information is to loop (start with 0) until we reach the Count (at that point we do not fetch anything, but stop). efetch.fcgi is the eUtil script to call. The example script prints out the results and waits until the user hits enter before going through the loop again. Each iteration gets the next page of data, starting at $retstart, returning $retmax items.

➔ Notice the fetch includes passing in the QueryKey and WebEnv from the search results.

```
for($retstart = 0; $retstart < $Count; $retstart += $retmax) {
  my $efetch = "$utils/efetch.fcgi?" .
        "rettype=$report&retmode=text&retstart=$retstart&retmax=$retmax&" .
        "db=$db&query_key=$QueryKey&WebEnv=$WebEnv";

  my $efetch_result = get($efetch);

  print "————-\nEFETCH RESULT(".
     ($retstart + 1) . ".." . ($retstart + $retmax) . "): ".
     "[$efetch_result]\n——-PRESS ENTER!!!———-\n";
  <>; # reads in the enter
}
```

EXERCISE: download mod03_example_files.zip, unzip and run eutils_example.pl. What sort of output are you seeing?

This example could be improved upon — for one thing, no error checking is done to make sure the response from the get actually returned anything.

For more information, see this the short video tutorial: https://www.youtube.com/watch?v=BCG-M5k-gvE&index=3&list=PL8FD4CC12DABD6B39

More info on eUtils (optional): https://www.ncbi.nlm.nih.gov/books/NBK25500/
The demonstration program discussed above (may need to change http to https in the script): https://eutils.ncbi.nlm.nih.gov/entrez/query/static/eutils_example.pl

BLAST Search Using Perl

Another useful Perl program from NCBI is web_blast.pl (see https://www.ncbi.nlm.nih.gov/books/NBK62345/ "Contents of the /blast/documents subdirectory" section), which has also been included in the Module 3 example scripts.

Because you can invoke Perl scripts from other Perl scripts (such as with the Perl `system` function), you can set up a BLAST search in one script and invoke web_blast.pl as-is to do the search and get the results. If you use web_blast.pl, please follow NCBI's guidelines for using their server as described in the code comments. In other words, only send a request once every 5 seconds or so.

web_blast.pl uses LWP and allows you to use different BLAST programs and different databases. It uses POST to setup the request and invoke the BLAST.cgi program at NCBI, and GET to retrieve status and results.

➔ When a BLAST search is performed, the script needs to poll NCBI to determine when the search is done, at which point the results can be obtained.

A few code snippets from the script web_blast.pl are below. They should make sense to you by now. Here is the POST request:

```
$args = "CMD=Put&PROGRAM=$program&DATABASE=$database&QUERY=".$encoded_query;

$req = new HTTP::Request POST =>
                   'https://www.ncbi.nlm.nih.gov/blast/Blast.cgi';
$req->content_type('application/x-www-form-urlencoded');
$req->content($args);
```

And the first part of the polling loop, which waits for the search to finish:

```
# poll for results
while (true){
    sleep 5;

    $req = new HTTP::Request GET =>
"https://www.ncbi.nlm.nih.gov/blast/Blast.cgi?CMD=Get&FORMAT_OBJECT=SearchInf
o&RID=$rid";
    $response = $ua->request($req);
```

Here is part of the if test that checks for the done status, which means the results are ready to get:

```
    if ($response->content =~ /\s+Status=READY/m)
        {
        if ($response->content =~ /\s+ThereAreHits=yes/m)
            {
            #  print STDERR "Search complete, retrieving results...\n";
            last;
            }
```

And finally, get results:

```
# retrieve and display results
$req = new HTTP::Request GET =>
"https://www.ncbi.nlm.nih.gov/blast/Blast.cgi?CMD=Get&FORMAT_TYPE=XML&NCBI_GI
=yes&RID=$rid";
$response = $ua->request($req);

print $response->content;
```

web_blast.pl follows the same pattern of "do the search, poll search status, obtain results" as the eUtils script example above. This is a common pattern. Often, the next step after polling is to go through the results list and obtain the data for items of interest in the list with additional web services calls.

EXERCISE: run web_blast.pl using blastp, the non-redundant data (indicated as "nr") and use the included protein.fasta file as input. Be patient. This could take a while. If you are sitting looking at a blank screen waiting, you can edit web_blast.pl to add some print statements to enable you to watch what it is doing.

- ⇨ **Note** that if you end up using web_blast.pl for your own purposes, it is best to write the output to a file and figure out how to parse the response working with the file. Once your parsing is accurate, then you can switch to parsing the string that is returned by the web service call and skip writing the data to the file. This strategy is a good one to follow with all web services and will save you a lot of time.

I should warn you that NCBI will sometime penalize you if you use its services a lot. If you run web_blast.pl one time, you may have a 30 second wait one time and a 300 second wait another time.

Sometimes NCBI can take a ridiculous amount of time to respond, like over an hour. If you end up waiting more than 10 minutes, it usually is best to ^C the script and try again. This is another reason to always save your response to a file and work with the file until your script is running smoothly, and only then hook up the parsing and other actions with the web service call.

Wrap Up

In this module we covered several topics related to sequence searching:

1. Sequence alignment and homology
2. Scoring matrices
3. BLAST searching (various types)
4. Multiple sequence alignments. We'll see some examples later where multiple sequence alignments are used in building entries that appear in a database. And it's important to realize that MSA algorithms are heuristics and not exhaustive; consequently, different algorithms and settings may give different alignments.
5. eUtils
6. BLAST searching with Perl