Sequence Alignment

Note: in the experiments below, you will not necessarily be given an exact recipe for how to proceed. You are encouraged to explore and find out what works and what doesn't. You're also encouraged to ask other students and team up with them. You're further encouraged to ask questions of the instructors! It's amazing how much better (efficient, robust, thoughtful) science is when people collaborate and engage in dialog.

Part 0: Files

We are going to use proteomes of some classic genomes: C. elegans, D. melanogaster, and A. thaliana. If you want to use your own favorite proteome, go ahead. Even better, do experiments with several genomes.

I like to record what I'm doing in at the command line in a lab notebook of some kind. A plain text file is ideal because plain text never goes out of date. Let's call this `lab1_notebook.md` for now, but you might want to call this something more descriptive. The first thing to write in it is the date, what versions of the files you downloaded, and where they came from.

When you write in your lab notebook, you should follow some kind of standard. I prefer doing *everything* in markdown syntax. This document is written in markdown. Do everything in markdown and if you want a pretty version of it, you can convert it easily.

Part 1: Local Alignment

The first program we are going to use for alignment is called `water`, which is named after the **Smith-Waterman** algorithm. This comes from the **EMBOSS** suite of bioinformatics programs. One of the first things you should do when using a new program is to skim the documentation. Some bioinformatics programs will have `man` pages, but not all. `water` doesn't come with a `man` page, but there is good documentation online at the EMBOSS site. Most bioinformatics programs will have a **usage statement** that gives you some help with the program. To display the usage statement, try the `-h`, `-help`, `--help` options or try using the program without any arguments. The `water` program responds to the first three, but if you give it no arguments, it will start prompting you for arguments (which can be useful or annoying depending on your mindset).

As you gain more experience with bioinformatics tools, you will find a large variety of command line syntaxes and usage statements. Why? Because most scientific programmers aren't

professional software developers. They don't always follow the rules and best practices of the Unix community (if you're a budding bioinformatician and software developer, try to follow standards rather than making things up on your own).

```
water -h
```

There are not may options. We need to provide two sequence files, penalties for gap opening and extension, and an output file. The scoring matrix appears to BLOSUM62 by default but others are available (see `/usr/share/EMBOSS/data` or `/usr/local/share/EMBOSS/data`). Let's try the interactive mode.

```
water
```

The program now asks for the name of the input sequence file. You can't tab-complete this. That's enough reason for me to NEVER USE THE INTERACTIVE MODE, but if you like it, go ahead (but it's yucky).

So where are we going to get our sequence files from? Let's grab the first protein from the A. thaliana, C. elegans, and D. melanogaster proteomes to begin. It doesn't really matter what sequences you use as the point is just to get the program to run (the helloworld equivalent). You can use `head` with a command line option for the number of lines and the `>` redirection symbol for this task. Save these files as `at1.fa`, `ce1.fa`, and `dm1.fa`. You might want to write a bit in your `lab1_notebook.md` file about how you did this. Clever/lazy people paste in parts of your `history`.

Now let's try aligning the various files to each other.

```
water at1.fa ce1.fa -gapopen 10 -gapextend 5 -outfile at_ce.water
```

Read the output of this command with `less`. Note the score, percent identity, percent similarity, and score. Is a score of 37 (or whatever) good? Hmm, that's a difficult question. What exactly does "good" mean? Should a good score mean that two sequences are evolutionarily related? That they have similar function? As a scientist, one of the first questions you should ask yourself is how likely that alignment could have occurred at random. At least you can answer that question with some degree of confidence...

Part 2: Random Expectation

In order to determine what the random background of sequence alignment looks like, we need to make some random sequences. There are several ways to go about doing this.

- Write a program that uses 5% probability for each amino acid

- Write a program that uses *realistic* probabilities
- Write a program that randomizes real sequences
- Use EMBOSS `shuffleseq` or some other program

Make some random sequences and then align them a bunch of times with `water`. This will give you an idea of what alignment score is expected at random. If you really want to explore what random looks like, you'll have to perform this experiment thousands of times (or more). So script the whole thing from sequence generation to histogram.

- Generate random sequences of some length
- Align sequences with some parameters
- Get the maximum scores
- Create a histogram of maximum scores

After performing some experiments, answer these questions to the best of your ability. You might use your `lab_notebook.md` file for these thoughts.

- Is the shape of the curve normal?
- Do you expect it to be normal?
- If not normal, how do you determine probability?
- Do you expect all protein comparisons to have the same distribution?
- What factors might change the distribution and how?
- How might real sequences be different from random?

Part 3: Karlin-Altschul Stats

Karlin-Altschul statistics tells us that the expected number of alignments that exceed some score depends on the score, lambda, the size of the sequences, and a few other parameters (see lecture notes). K-A also forbids gaps and has some other restrictions. But is that all really true? The way to find out is with some experimentation.

You might not have time to do all the experiments below. Try doing one of them to begin. Hopefully other students have done the other experiments and you can compare.

1. Write a program to examine the effect of sequence lengths
2. Write a program to examine compositional effects
3. Write a program to examine gap penalties

In your `lab_notebook.md` file, explain the intent of the experiment, how you approached it, what you found, and what that means.

Part 4: Alignment Significance

Your task is to determine the statistical significance of an alignment. The query sequence is B0213.10, which can be found in the C. elegans proteome. Search this against all A. thaliana and D. melanogaster proteins to find its homologs. Of course, you can use some other sequence and some other proteome as you like.

Before you begin, it's a good idea to get an idea how long it will take to do the experiment. Will your experiment take seconds, minutes, hours, days, or longer? Before attempting to do the whole experiment, it pays to do a small fraction first. Don't begin until you can answer the following questions?

- How many amino acids can I align per second?
- How many amino acids do I need to align to do this experiment?
- How long would it take to compare two proteomes?

Set up the experiment any way you like. You will find the Unix `time` command handy. Address the following questions in your `lab_noteboook.md` file.

- What is the best match in each genome?
- What protein is this?
- What are the alignment properties (% identity, etc)
- What is the expected score of your alignment at random?
- How different is your best score from random?
- How statistically significant is this score?
- How biologically significant is this score?

Part 5: BLASTP

BLASTP is supposed to be a lot like Smith-Waterman, but better. Perform the same experiment above, but with BLASTP. To get a usage statement for `blastp`, type its name on the command line.

```
blastp
```

There is a bewildering number of options! And you'll have to format a database with `xdformat` before you can use `blastp`. Here's how you format a protein database of the C. elegans proteome (assuming its name is `worm`).

```
xdformat -p worm
```

This creates several new files with various extensions. These extra files are the BLAST database. To search it with some other fasta file, the command line is very simple.

```
blastp worm other_file.fa
```

There are many ways to control `blastp`. But before we do that, let's redo the experiment in Part 4.

- How fast is `blastp` compared to `water`?
- Do you believe the statistics in the BLAST report?

Part 6: Orthologs

We previously aligned B0213.10 to the A. thaliana and D. melanogaster genomes. The best match is often the *ortholog* but really, the best reciprocal match is a better definition of the ortholog. That is, after finding the best match to the fly genome, one must take the fly protein and search it against the worm proteome to determine if it finds B0213.10. If if it does, the proteins are orthologous.

Let's do something ambitious. Let's find ALL the orthologous proteins between worm and fly (or any two genomes). First thing, we had better estimate how long that will take. You can do that by aligning a few sequences and timing it.

There are a lot of parameters that control the speed of BLAST. The most important of these are the seeding parameters. The default `blastp` search uses `W=3 T=11`. We can change the seeding parameters quite a bit. We will set `T=999` so that there is no **neighborhood** around each word match. In other words, only exact matches will produce a seed. Let's vary the word match from 1 to 6.

```
time blastp worm whatever T=999 W=1 > w1.blast
time blastp worm whatever T=999 W=2 > w2.blast
etc.
```

- How much faster does `blastp` run at higher word sizes?
- High speed reduces sensitivity. What is acceptable to you?

Time to set up the searches. Create BLAST databases for each proteome and then search one against the other. Do these in separate terminals so they run at the same time (or you can background them).

```
blastp worm fly mformat=2 hspmax=1 E=1e-15 W=5 warnings > wvf.blastp
blastp fly worm mformat=2 hspmax=1 E=1e-15 W=5 warnings > fvw.blastp
```

A few command line switches were added. `mformat=2` made the output tabular. `hspmax=1` made it so that only the best alignment was shown (there can be more than one unlike `water`).

`warnings` turns off some warning messages. `E=1e-15` set the E-value to something that should never occur by chance.

If the search is taking a while, skip down to Part 7 and start working on that.

Once the jobs are complete use `less` to examine the outputs of each (actually, you can do this even before the jobs are complete). Columns 1, 2, and 3 contain the names of the proteins and the E-value of the match. If it makes it easier, you can `cut` these, but you might want to look at other columns of the file.

```
cut -f 1-3 wvf
```

Use the `less` search feature (forward slash key) to find proteins in one file that match proteins in the other file. Find some putative orthologs. Answer the following questions.

- What difficulties are there in finding orthologs?
- Which BLAST parameters might you change if you did this again?
- What additional information would you like to have?
- What logic would you use to find orthologs automatically?

Part 7: Paralogs

In this part, we are going to look for the **paralogs** of a couple worm genes: T21B10.2a and B0213.10. That is, we are going to search the these proteins against the C. elegans proteome to look for sequences that arose by duplication within the same genome. A gene may have from 0 to many paralogs. A simple, but flawed way to determine which alignments represent paralogs is to use an E-value cutoff. But it's a good place to start.

```
blastp worm T21B10.2a > T21B10.2a.blastp
blastp worm B0213.10 > B0213.10.blastp
```

Inspect the BLAST report with `less`. Do all the sequences look like they are highly related? You may want to change the E-value cutoff to something higher or lower than 1e-10.

- What parameters did you choose for the search?
- How did you choose the parameters? What trade-offs were considered?
- What other properties would be useful for finding paralogs?
- How would you identify paralogs using BLAST?