Exercise 4

NOTE this week only Bash loop solutions will be accepted. No awk loops! You may use awk to filter BLAST hits, but not for the identification of features in the BED file. Please ask if you are unclear what is permitted

Questions

1. (100 points)

In this weeks data, there are BED files which correspond to the assemblies you were working with last week. The BED files each describe the location of genes in one of the assemblies. Each organism has an assembly file (.fna) and a BED file (.bed). e.g., Wolbachia.fna is the *Wolbachia* assembly and Wolbachia.bed is the bed file describing the locations of genes in the *Wolbachia* assembly.

Your task is to extend the find_homologs.sh script you wrote previously. You may also use one of the example scripts demonstrated in class as the foundation for this assignment.

Below is some background about the task. See the "Instructions" section below for the rest of your instructions.

Background info

More about BED format

This information will help you understand the data you are working with and may be useful in writing your script.

We used a BED file in some previous assignments, but didn't discuss the information it encodes in much detail. We saw that BED format is used to encode the location of features in a sequence. Each line of the BED file refers to a different sequence, and the three mandatory columns of BED format are sufficient to describe exactly where the feature is located. The BED files you will work with today have more than 3 columns as they include information from some of the optional BED format fields. See this website for a reminder of BED format

The columns in the files you are working with today contain the following information:

- The sequence IDs in column 1 of each BED file correspond to the FASTA headers in the associated assembly. Typically, the first set of non-whitespace characters after the ">" symbol of a FASTA header are taken to be the sequence ID. After the first whitespace character, there is sometimes additional metadata, but you will see that BLAST and BED format do not pay attention to anything after the first whitespace. (For example, in a header line ">seq1 species_name something else", only "seq1" is considered to be the sequence ID.) **This is very important if the FASTA file you are working with contains more than one sequence!**
- The second and third column of the BED file gives the start and stop position in the sequnce where the gene is encoded. Each line of the BED file refers to the location of a different gene. Note that start and stop position are misnomers in that they imply direction. Instead, the "start" column simply contains the

location of the boundary of the feature that is closest to the first base of the FASTA file. This will become significant in column 6...

- Column 4 contains a name or label for the feature. This can be anything and does not necessarily need to be a gene name.

- Column 5 contains a score. In the files provided here, no score is used. Instead, as BED format forbids empty columns, a "." character is used to populate the unused column. If we did not have data in column 6, this column could be absent and we could have stopped at 4 columns. However, as we want to include the data in column 6, all prior columns need either data or ".".

- Column 6 indicates the strand on which the feature is present. Strand here refers to which of the two DNA molecules contains the feature. In the case of genes, while both DNA strands contain sequence that corresponds to the gene, only one is used during transcription. The strand column of a BED file indicates directionality of the feature. In the case of genes, this directionality indicates whether transcription proceeds left to right through the sequence (+) or right to left (-).

Note that for genes encoded on the "-" strand, transcription *begins* at the "stop" end of the BED coordinates and *ends* at the "start" end. This is not relevant to the assignment, but it is always important to keep in mind the biology being represented in the data you work with as bioinformaticians! Sometimes you must be careful when interpreting words like "start" and "stop" or other decisions in the representation of biological data.

More info about protein domains

This information is solely for those interested and can be safely skipped without impacting your ability to complete the assignment.

Many genes encode proteins. Proteins perform most of the functions that result in life. Proteins are made up of a linear strand of amino acids that typically folds into a globular structure (although linear or unstructured sections can remain). The folded protein can contain multiple regions that perform different functions. These functional regions are called domains.

Different protein domains are associated with different functions. For example, zinc finger domains bind DNA (and other things) and are often found in proteins that act as transcription factors (gene expression regulators). Histidine kinase domains are involved in transducing a signal (usually an environmental signal) sensed by another domain. However, it is the other domains of histidine kinase proteins that do the sensing. It is the combination of multiple different domains with different functions that produces the overall function of a specific protein.

Many protein domains can be found in combination with a diverse range of other domains. For example, while histidine kinases are not the most diverse, you can see in these two examples (1 and 2 - scroll down and look at the Domain architecture section) that HK domains are found in combination with different domains. Note the different domain IDs and different length of the protiens.

Without knowing what the other domains do, we can infer the function of HK genes simply by the presence of the HK domains. If you were a researcher interested in bacterial environmental responses (such as how a pathogen senses it is infecting a person and not in a pond), you might want to identify HK domain-containing proteins as likely candidates of how the organism senses the environmental signal of interest. You could then make gene deletions of each of the HK-domain genes and test the phenotype of each mutant to see if one of the genes is important.

In a study like the one described above, what we are doing here is the first step. However, the process of finding sequence matches to a query of interest in a sequence database is useful in many more situations.

Instructions

Modify the identify_homologs.sh script you wrote last week to carry out the following steps:

1. Find putative homologs of query amino acid sequences in FASTA format (same as last week's instruction - i.e., what your current script does)
2. Use Bash loops and Bash conditional statements (no `awk`) to identify which genes in a BED file contain the identified homologous histidine kinase domains. (Hint: if a gene contains a domain, then the location of the domain will be entirely within the boundaries of the gene, inclusive of the gene start and stop position)
3. Write an output file containing the unique gene names which your script identified as containing predicted HK domains.

Usage of your script should be: `homolog_identify.sh <query.faa> <subject.fna> <bedfile> <outfile>`

Format of output file: 1 gene name per line, no other information. Below is the output I got for *Wolbachia* with the script I wrote. If you don't get this output for *Wolbachia*, then you have made a mistake

```
GQX67_RS05910
GQX67_RS06230
```

N.B.:

- If you are on Windows using WSL note that you'll have **much** faster runtimes if you do your development with all of your script and data files stored on your linux filesystem (e.g., somewhere in your home ~ directory) instead of somewhere in your Windows filesystem (if your path starts with "/mnt/" you are in the wrong place!). I timed my example solution taking almost 10X as long in my Windows filesystem.
- **Do not** use `-task tblastn-fast`. Just use the default task (i.e., no task flag which defaults to `-task tblastn`). While tblastn-fast it is described as producing comparable results more quickly, you will find that using it in our case will result in fewer homolog predictions. It could certainly be the case that the genes that are excluded when using tblastn-fast are not truly homologs of our queries. However, for our purposes the larger set of predictions is fine. We are defining the criteria for predicted homology ourselves as 30% identity over 90% query length. Furthermore, by all students using the same `-task` in your BLAST command, there will only be one correct answer to this assignment which makes grading and discussing the assignment much more straightforward.
- All of the sequences in HK_domain.faa are Histidine Kinase domains. i.e., many of your query sequences are homologous to one another. That means it is likely that some of your BLAST hits will be overlapping. Therefore, you will probably need to check for duplicate genes in your results
- not all bacterial gene IDs are nice gene names. Some are locus IDs referring to a place in the genome (e.g., "N16961_RS00025"). This sort of name often means that the gene at that locus has not been characterized in a publication, but does not necessarily mean nobody knows what sort of thing it does.

As described in the BED format description above, genes are transcribed in a directional manner. The wikipedia page linked here has a good description of the relationship between DNA sequence and RNA sequence orientation.

Transcription can either proceed in the same direction that the sequence is represented in a FASTA file, or in the opposite direction. This is represented in BED format using + or -, respectively. A plus simply means that the coding strand of the gene is the sequence shown in the FASTA file. - means that the coding strand is the one not being shown in the FASTA file.

Following transcription of the gene from DNA into RNA, the RNA is then translated into amino acid sequence. Translation occurs 5' to 3' (5 prime to 3 prime) along the RNA strand, which is the same direction the RNA was transcribed.

That means that a gene whose orientation is "+" in a BED file produces an RNA strand with sequence with the same orientation as the DNA in the file. That same gene encodes a protein whose first amino acid was encoded by the first 3 bases at the left-most end of the gene sequence. i.e., + strand genes read left to right for the DNA, RNA, and protein sequence.

The consequence of the relationships between directionality of the 3 sequences involved (DNA, RNA, and protein) is that there is a fixed relationship between the DNA encoding a gene and the sequence of amino acids it will produce. It can only produce a protein in one direction, not both.

As we are performing a tBLASTn of protein sequence against DNA, we end up with a start and stop position for our match which indicates the direction in the DNA that the protein would have to be encoded. We also have a BED file indicating the direction in which we *know* genes are encoded in each genome. If any of our protein matches are in the opposite direction to the gene that contains the matched region, then we know that the gene **could not** encode that protein. Make sure you take that into account in your script.

Ungraded extra challenges

To be clear: these are challenges you can set yourself as a learning exercise, but will not impact your grade. These challenges involve concepts which will likely be useful to you in the future.

1. I have setup a leaderboard that will track the running time of student's scripts. Think about how you can improve the efficiency of your script. While each command takes very little time to execute, if you use certain coding constructs like loops within loops, run times can become quite long. Sometime it is necessary to use constructs like that, but there may still be approaches you can take to minimize run times without compromising your output.

   If you are only going to run a script once, it is usually not worth optimizing run times. However, it is good to learn some tricks that you can commonly employ to optimize speed when you do need to. It can also be a fun puzzle.

   You can time how long a command takes to run using `time`. i.e., `time <pipeline or loop etc>`. The "real" result is the total run time. Use the leaderboard to see how your run time compares to other students in the class.

2. Last week's extra credit question specified that you were not allowed to use loops. Now that we have covered loops, go back and try it using the concepts we have now covered.