Programming for Bioinformatics | BIOL 7200

Exercise 12

Instructions for submission

- Download the file, exercise12_data.tar.gz, which contains data for use in this assignment.
- Name your package tarball: "gtusername.tar.gz"
- Upload your submission file on canvas.

Grading Rubric

This assignment will be graded out of 100.

Submission specifications

1. Your module ***does*** need to be generalizable to any SAM input files.
2. You may only use Python standard library modules
3. Your submission should consist of a tar.gz file containing a directory with your package and magop.py script.

Background

Assignment goal

Welcome to the final assignment of the semester! This week you will be combining the various components of what we have been calling your magnum opus. Once combined, your magnum opus will really justify that name. Each of the previous assignments has been designed to introduce or reinforce a new concept. This week we are not so much reinforcing a coding concept, but instead getting experience combining various components into a large and complex software package.

Over the course of the last few assignments there may have been points where you wondered what was the point in some of the things we were doing. For example, why bother using modules when everything could just go in a single script - then you wouldn't have to manage multiple files. Indeed, you have seen this semester through course materials, assignments, and code review sessions that you can often achieve the same things in several different ways. However, it's not always clear which is the "best" way.

I have often alluded to some coding constructs providing organizational benefits. For example, by organizing related functions and classes into modules, you know which file to go to if you want to edit those things. You are also able to control which things are exposed when the module is imported. Some functions are only needed within the module and there may only be one main function that you actually wrote to be used outside the module (e.g., you might want to use an isPCR function, but not all the functions used in intermediate steps of isPCR). My claims that it is somehow good to organize your code in ways which are often more complex or laborious to set up might not have been very convincing. The goal of this assignment is for you to see for yourself whether they are valuable.

What this assignment is intended to do is to give you an opportunity to experience the consequences of design choices we have made over the last few weeks. As you are piecing together you magnum opus, take some time to think critically about how the components are structured and whether each part is easy to use. Maybe experiment with refactoring components and see if the design of each part can make the combination of components simpler.

Much of learning to do software development is learning to see the future. As you gain more experience putting together complicated software, you get a feel for what the downstream consequences are of different design decisions. Often, when starting out on a new project, it is worth the effort to take a little bit more time to write something in a more complex form (e.g., in a class or module) if it makes your job easier later on. The goal of this assignment is for you to actually experience those downstream consequences.

Inferring phylogenetic relationships

This section provides some optional background on what your magnum opus will be useful for after completing this assignment. You can skip to the section "Phylogenetic inference in your magnum opus" if you just want to read details that are directly related to the assignment.

If you are unfamiliar with phylogenetics take a look at this review for an overview of some key concepts and approaches.

Phylogenetics is the study of evolutionary relationships among biological entities. Most often, humans have not been around to witness (or at least not paying close attention and writing down) the evolutionary history of the forms of life around us. Therefore, in order to learn about the evolutionary relationships among organisms, genes, or any other biological unit, we must infer those relationships using other data. Collectively, methods of inferring evolutionary relationships from data are called "phylogenetic inference".

The basic assumption underpinning all phylogenetic inference methods is that things that are more similar to one another are more closely related evolutionarily. In addition, all life arose from a single (or a small number of) common ancestor. Therefore, imagine we have three organisms, A, B, and C. If A and B are more similar to one another than they are to C, then we might infer that A and B share a more recent last common ancestor than do all three organisms. Here, "last common ancestor" is the name given to a hypothetical individual from whom a set of individuals descend. For example, the last common ancestors of a person and their first cousin is their grandparents.

So, phylogenetic inference is as simple as assessing the relative similarity among a set of things. The difficult part then is two fold: for which features should you assess the similarity, and how should you quantify the similarity? These are the difficult questions in phylogenetic inference.

Until very recently, scientists were limited in the characters they could use in phylogenetic inference to mostly use physiological characteristics or other phenotypic traits. For example, how many eyes does it have, how many limbs does it have, etc. This system worked perfectly fine for a lot of cases, particularly closely related organisms (you have likely heard of Darwin's finches, which were characterized using things like beak size and shape).

While we have learned huge amounts about life through such analyses, the use of physiological traits has weaknesses. Specifically, in order for a trait to be informative with regards to phylogeny it must be present in all organisms being considered, and it must be homologous, i.e., have changed over time from an ancestral state which was present in the last common ancestor of all the organisms being considered. The requirement that all individuals being considered have a trait makes it difficult to compare distantly related organisms (how many comparable traits can you imagine between yourself and a slug vs yourself and a cat?).

Additionally, it is more difficult to be confident that traits in distantly related organisms are homologous. For example, bats, birds, and butterflies all have wings, but their wings are not homologous; each evolved their wings independently through convergent evolution. The lack of informative traits and issues like convergent evolution reduce the power of phenotypic analyses to resolve phylogenetic relationships over large evolutionary distances.

In recent years, DNA sequencing has become increasingly affordable and accessible. DNA sequence-based phylogenetic inference offers large advantages over phenotypic trait-based analysis. One of the largest advantages is the sheer quantity of information provided by DNA sequencing. Each base in a DNA sequence can provide information about the evolutionary history of a set of organisms.

Over the last few weeks, these assignment background sections have described how 16S sequences have become commonly used for phylogenetic inference of prokaryotic organisms (*Bacteria* and *Archaea*). This week, you are actually going to use 16S sequences to perform phylogenetic inference. To do so, we need to cover the steps required to perform phylogenetic inference.

Just like for phenotypic traits, there are some core requirements that dictate which sequences we can use for phylogenetic inference. Those are that the sequence must be present in all organisms being considered, and that differences in the sequence represent changes that have occurred over the evolutionary history of the organism. This last point is especially tricky for bacteria as many bacteria undergo frequent horizontal gene transfer (they exchange DNA with other organisms to add to or replace DNA they already had). The 16S gene is an ideal sequence for phylogenetic inference because the small subunit rRNA it encodes is essential for life, which means that it is reliably present (although eukaryotes have a different version of this gene - 18S). Secondly, while horizontal gene transfer of ribosomal genes like 16S has been observed, it seems to be rare. Differences in 16S sequences are therefore more reliably (than other genes) representative of changes that have happened over the evolutionary history of organisms.

Phylogenetic inference in your magnum opus

Currently, your magnum opus is actually a few separate things. This week we are going to combine them into a single application. The purpose of this application will be to extract 16S sequences from different data types (reads and assemblies). The data produced by your application can then be used for phylogenetic inference to classify unknown prokaryotic organisms by placing them in a phylogenetic tree based on an analysis of their 16S rRNA gene sequence.

In order to infer a phylogenetic tree based on 16S sequences we need to do three things: identify sequences that are present in all the organisms being considered, compare the sequences to assess which are more different and which more similar among them, and then to use that comparison to infer relationships among the sequences.

There are a few ways we can go through that process, depending on which phylogenetic inference method we wish to use (see the review linked at the start of the section above). Sadly, the semester is almost over so we don't have time to implement a phylogenetic inference method ourselves (however, I'll provide an optional, ungraded, extra assignment describing how you can do so if you wish). Instead, we are going to prepare our data and then use tools created by others. What we are going to do ourselves is just the first step of the three steps listed above: identify sequences that are present in all the organisms being considered and output them in a format that can be used by downstream tools.

It is this task that we have been focused on for the last few assignments. We have now written Python code to extract 16S sequences form two different forms of input data: reads and assemblies. However, the sequences produced by your isPCR and reads-based workflows are not the same. Specifically, your reads-based approach currently extracts roughly the whole 16S gene, while your assembly workflow extracts just a small portion (the V4 region). In order to compare the sequences we generate from assemblies with those we generate from reads we need the sequences to be equivalent. i.e., we either need to extract the whole 16S gene from assemblies or we need to extract just the V4 region from reads.

The amplicon from the 515F and 806R primers we are using is the V4 hypervariable region of the 16S gene. Using this region alone may not provide the best resolution of differences compared to using the whole gene. However, we have been using it in these assignments as it works well enough for course phylogenetic comparisons and produces a manageable amplicon. Furthermore, the primers we are using are able to produce an amplicon for highly diverse organisms. Primers that amplify the whole gene do not have the ability to amplify organisms as diverse as the V4 primers we are using. Therefore, we will be able to classify more diverse organisms if we use the small V4 region we used for our original isPCR assignment. (You should certainly experiment with other primers as well to see how they perform if you are interested.)

For this assignment we will extract the V4 region from our mapping data. To do so you should combine the mapping and isPCR approach. Specifically, map the reads to a 16S reference and then use isPCR and the 515F and 806R primers to amplify the V4 region. I have provided you with the subsequence of each of last week's references that corresponds to the V4 amplicon plus primer annealing sites as well as an extra 5 bases on each end. The extra 5 bases are to improve our mapping results. When mapping reads to a reference with lots of mismatches, if the first or last few bases don't match between the read and reference then the mapping software soft clips those bases. This can be handled by simply including a few flanking bases in the reference so that additional matches can support the extension of the alignment between read and reference through the region we are interested in.

Once you have extracted corresponding sequence regions from all of the input samples, the next step is to figure out which positions in each sequence correspond. As you have seen while working with the read mapping data over the last week, there can be short insertions and deletions that result in 16S gene sequences being different lengths. That means that you can't simply assume that index "N" of two sequences correspond to bases you could meaningfully compare. Instead, you need to identify corresponding positions by aligning the sequences.

The Needleman-Wunsch algorithm that you implemented as part of your magnum opus is a pairwise alignment algorithm. That means it can be used to align two sequences. However, when wanting to align more than two sequences so that all sequences in the group, one must use a multiple sequence alignment algorithm. Multiple sequence alignment algorithms are much more complex than Needleman-Wunsch and so we won't be implementing one of those this semester. Instead, we can use something like `muscle` to align our sequences. `muscle` does need the sequences to all be in the same orientation though (it does not consider the reverse complement of each sequence when aligning them). We shall therefore use our Needleman-Wunsch implementation to make sure all of our sequences are in the same order. This can be achieved in just the same way that you figured out which orientation two sequences align best in the assignment where you first implemented Needleman-Wunsch (i.e., align them in with both in the forward orientation, then flip one, and keep whichever orientation had the best alignment score). The way to use that approach here is simply to pick a single sequence to act as an anchor and then orient all the other sequences by aligning them against the anchor sequence one by one.

After orienting all of the extracted 16S sequences so they are the same orientation, you can simply write those sequences to the stdout of your application. Those sequences can then be used for a downstream phylogenetic analysis (which I will describe below). Let's review a bullet point list of what I am asking you to do this week.

Assignment details

As described above, this week your task is to combine the various components of your magnum_opus into a single package and a script called `magop.py` with a command line interface that does the following:

Given zero or more assembly files and zero or more pairs of reads files, use isPCR, read mapping, and Needleman-Wunsch to extract and align the V4 region of the 16S gene using user-provided primers and reference sequences.

Some more details about what the above summary means:

- Your script should optionally accept assemblies. If assemblies are specified, your script should accept them as a space separated list or a glob (remember Bash expands globs out to a space separated list so this is really the same thing). For example `magop.py -a assembly1.fasta assembly2.fasta` or `magop.py -a data/assemblies/*`.
- Your script should optionally accept paired Illumina reads files. As with assemblies, you should accept these as a space separated list or glob. Importantly, you will need to identify which files correspond to a pair. To do so you can assume that the read file names will follow the following regex: `(sampleID)_[12].fastq`. i.e., a sample called "Ecoli" would have read files "Ecoli_1.fastq" and "Ecoli_2.fastq". It is important that you pair the reads so that you map corresponding reads to the references.
- Your script should accept primers as it has in previous weeks (i.e., a path to a fasta file containing the primers). This input should be required
- Your script should accepts reference sequences as it did for last week's assignment (i.e., path to a fasta file containing reference sequences). This input should be optional, but you might want to add a check to require it if reads are provided (this would be a sensible input check, but it's not required for the grade)
- If assemblies are provided, you should perform isPCR using the provided primers to extract amplicons. You should only keep one amplicon from each assembly (the 16S gene is often present in multiple copies, but they are usually identical or very similar)
- If reads are provided, you should map each pair of reads to the provided references and then use isPCR to extract one amplicons from each mapping consensus sequence (i.e., for each pair of reads, map and then extract the best consensus, then isPCR to extract an amplicon from that best consensus)
- If references are provided (i.e., when working with reads), you should also use isPCR to extract one amplicon from each sequence in the reference file.
- After extracting all amplicons, you should use Needleman-Wunsch to orient the amplicons consistently. Note that you should not keep the aligned sequences, just use the alignment score to determine if a sequence needs to be reverse complemented and then keep the reverse complement if so.
- Format the amplicons into FASTA format and print them to the stdout.

The usage for your script should be

```
magop.py [-a ASSEMBLY [ASSEMBLY ...]] -p PRIMERS [-r READS [READS ...]] [-s
REF_SEQS]
```

where square brackets indicate optional inputs and nested square brackets such as `[-a ASSEMBLY [ASSEMBLY ...]]` indicate that more than one input can be provided.

In order for your script to work with the diverse sequences we are using, you will need to modify your isPCR BLAST command to tolerate more mismatches. Specifically, you should add the following to your command: `-word_size 6 -penalty=-2`. Without those settings you won't get any amplicon for the *Spirochaetes* in the dataset.

Expected output

For a change this week, instead of giving you the expected direct output of your script, I'm going to give you instructions to perform phylogenetic inference using your output and then an example of the tree topology you should get. All of the functionality of your components was already assessed in previous assignments. This expected output will allow you to assess if you have put them together right.

I will show an expected output for all of the assemblies, reads, and reference sequences provided in this week's data tarball.

All of the software used in the following sections can be installed using conda/mamba.

Aligning your sequences with `muscle`

`muscle` takes input sequences in a FASTA file. Therefore, you will need to write your output sequences to a file before you can align them. Once they are written to a file, you can align them with the command

```
muscle -align <input file> -output <output alignment>
```

Inferring a tree with `iqtree2`

There are several approaches that can be used to infer phylogenetic trees. Among the most popular approaches is maximum likelihood analysis. You can read about maximum likelihood and other tree inference methods in the review linked in the assignment background. There are several excellent tools available for inferring maximum likelihood trees and we will be using one of the best command line tools: `iqtree2` (you may encounter people with strong opinions about which tree software is best. IQ-TREE is faster than its main competitor RAxML and has a nicer command line interface so I selected that for this assignment.)

`iqtree2` will write a bunch of files to your current directory by default. It also prints a lot to the stdout. For those reasons, you may want to make an output directory and use quiet mode. I'll include those in the example command, but if you don't want them just remove the `--prefix` and `--quiet` parts. I'll assume an ouput directory called "tree" but call it what you like.

```
iqtree2 -s <alignment> --prefix tree/ml_tree --quiet
```

In the output directory, the file with the extension ".treefile" is your tree.

Note that the `--prefix` option just prepends whatever you put there to the beginning of output file paths. If you prefer your outputs to have a different basename, just change the "ml_tree" part.

Rooting a tree with `gotree`

IQ-TREE will output an unrooted tree. Typically, when you imagine a phylogenetic tree you probably think of something that starts on the left of the page and then as you look from left to right you see a series of branchings that end in the leaves of the tree. Indeed, this is typically a default way in which trees are visualized. However, this representation implies a single origin point which acts as a root. In the case of phylogenetic trees, that root would ideally be a hypothetical last common ancestor of all the things in the tree. Reading the tree from left to right would then be like moving through time. Each branching would therefore represent the divergence of two branches of the lineages of the organisms in your tree at some point in the past. The issue is though, if your tree is not rooted based on knowledge you have about where the original ancestor would be in the tree then that representation is misleading. Instead of viewing the rectangular layout that you are probably most familiar with, it would be more appropriate to view an unrooted tree in a way without an obvious start or end. To see what I mean, got to IToL and click on "unrooted" in the right hand menu window.

Fortunately for us, we do have knowledge about where the root should be in our tree. The data I provided you with has both *Bacteria* and *Archaea*. The root of those groups should be in the branch connecting them. You could therefore root explicitly by defining the *Archaea* (or the *Bacteria*) as what is called an "outgroup". However, as the branch between the *Bacteria* and *Archaea* is so long, you can actually just root on the middle of the tree (called midpoint rooting) to separate those groups, which is perfectly fine for this situation.

There are lots of tools for manipulating trees. I really like `gotree`, which provides a lot of useful tree manipulation functionality and has a nice, intuitive command line interface. `gotree` supports both outgroup and midpoint rooting. I'll provide a command to midpoint root, but check out the documentation if you are curious to try other rootings.

```
gotree reroot midpoint -i <input treefile> -o <output rooted treefile>
```

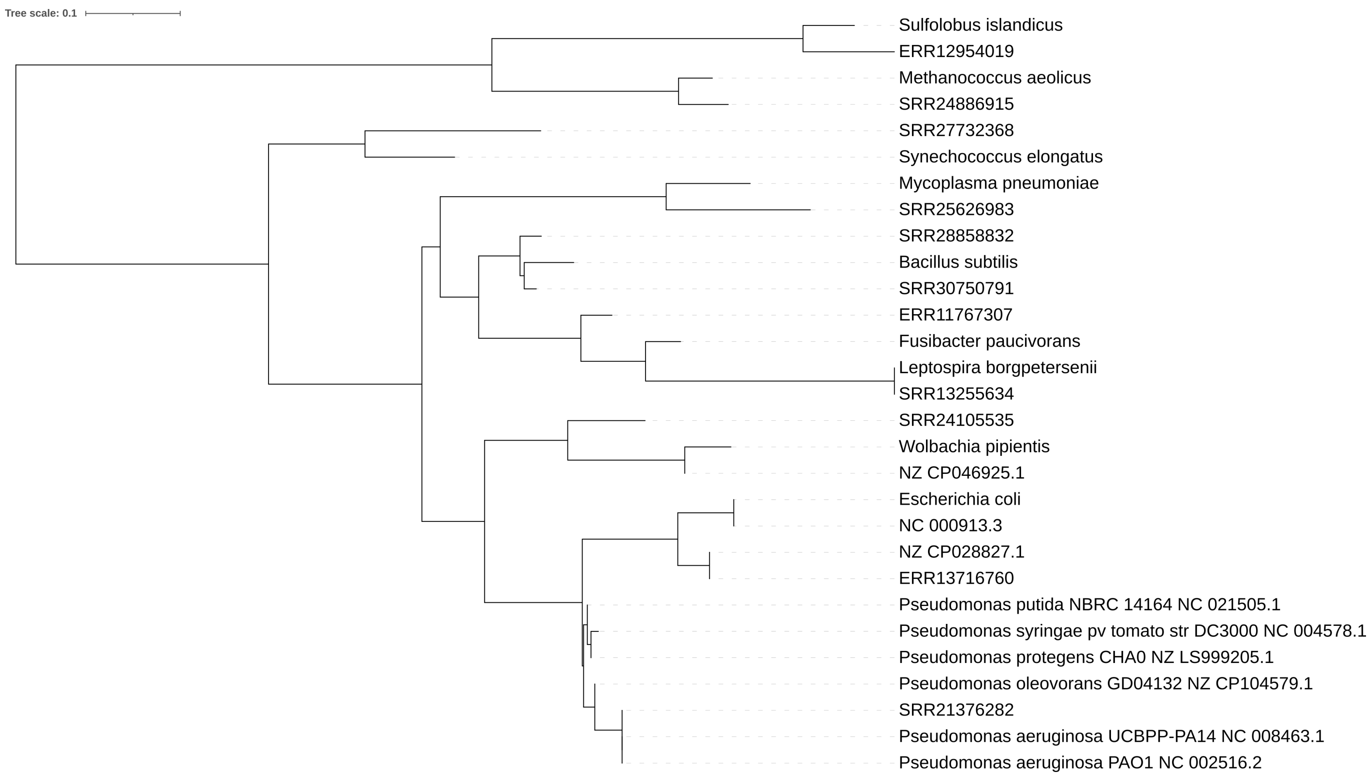
Note that `gotree` also supports reading trees from the stdin and writing them to the stdout if you prefer.

Expected output tree

After midpoint rooting the tree using the above tools, you should have a tree that looks like this (note your branch lengths may not be identical to these as maxmimum likelihood methods often use non-deterministic steps)

```
(((SRR24886915:0.052625178,Methanococcus_aeolicus:0.0358153276):0.197569359,
(ERR12954019:0.0966964968,Sulfolobus_islandicus:0.0543698655):0.329315489):0.503820
6648500001,
(((((((SRR13255634:0.000001,Leptospira_borgpetersenii:0.000001):0.2633529694,Fusiba
cter_paucivorans:0.0370652287):0.0684704067,ERR11767307:0.0329521578):0.1081518529,
((SRR30750791:0.012819385,Bacillus_subtilis:0.052394945):0.0044592618,SRR28858832:0
.0225816465):0.0437371911):0.0406996379,
(SRR25626983:0.1526125852,Mycoplasma_pneumoniae:0.0889670182):0.2391305027):0.01940
07195,
(((((Pseudomonas_protegens_CHA0_NZ_LS999205.1:0.0000026856,Pseudomonas_syringae_pv_
tomato_str_DC3000_NC_004578.1:0.0078434758):0.0038960055,Pseudomonas_putida_NBRC_14
164_NC_021505.1:0.0000020031):0.0039693457,
(((Pseudomonas_aeruginosa_PAO1_NC_002516.2:0,Pseudomonas_aeruginosa_UCBPP-
PA14_NC_008463.1:0):0.000001,SRR21376282:0.000001):0.0288975852,Pseudomonas_oleovor
ans_GD04132_NZ_CP104579.1:0.000001):0.0118877822):0.0013653507,
((ERR13716760:0.000001,NZ_CP028827.1:0.000001):0.0335799097,
(NC_000913.3:0.000001,Escherichia_coli:0.000001):0.0592373699):0.1010885427):0.1033
991177,
((NZ_CP046925.1:0.0000024848,Wolbachia_pipientis:0.048922924):0.1238701325,SRR24105
535:0.0819209235):0.0879869042):0.0663416244):0.1622957982,
(Synechococcus_elongatus:0.0948611532,SRR27732368:0.1860209665):0.1020748106):0.267
46026604999995);
```

And if you view it on IToL (or similar tree viewing software) it should look something like this


Tree scale: 0.1
Sulfolobus islandicus
ERR12954019
Methanococcus aeolicus
SRR24886915
SRR27732368
Synechococcus elongatus
Mycoplasma pneumoniae
SRR25626983
SRR28858832
Bacillus subtilis
SRR30750791
ERR11767307
Fusibacter paucivorans
Leptospira borgpetersenii
SRR13255634
SRR24105535
Wolbachia pipientis
NZ CP046925.1
Escherichia coli
NC 000913.3
NZ CP028827.1
ERR13716760
Pseudomonas putida NBRC 14164 NC 021505.1
Pseudomonas syringae pv tomato str DC3000 NC 004578.1
Pseudomonas protegens CHA0 NZ LS999205.1
Pseudomonas oleovorans GD04132 NZ CP104579.1
SRR21376282
Pseudomonas aeruginosa UCBPP-PA14 NC 008463.1
Pseudomonas aeruginosa PAO1 NC 002516.2