Programming for Bioinformatics | BIOL 7200

Exercise 11

Instructions for submission

- Download the file, exercise11_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 your package.

Background

The challenge of working with diverse sequences

As I mentioned last week, 16S rRNA gene sequences are used to classify prokaryotic organisms because the sequence is sufficiently conserved as to be reliably present in those organisms, but also sufficiently diverse as to allow us to tell the organisms apart. If you want a sense of just how diverse the 16S gene can be, use your Needleman-Wunsch code to align the *Sulfolobus islandicus* and *Escherichia coli* sequences I have provided this week in the data/refs/16S.fna file.

The challenge of working with diverse sequences in situations like these is this: imagine you have isolated some kind of bacterium but don't know what it is yet and have sequenced its genome. How do you identify which part of the genome corresponds to the 16S gene? If you already have an example of a 16S gene from a highly related organism then it is simple: you just find the part of the genome that is very similar to the 16S gene you already have. However, as the 16S gene can be quite diverse (see the *S. islandicus* and *E. coli* comparison suggested above), it can be quite difficult to tell first which region even corresponds to the 16S gene, and second, what the exact boundaries of the gene are.

This has actually been a difficult question that has taken many people a lot of work to figure out. It is also a question at the very core of bioinformatics. One of the earliest applications of bioinformatics was concerned with comparing sequences in different organisms. That involves two steps: Identifying which portion of each organism corresponds to the shared part, and then figuring out how to measure differences. Before whole genome sequencing, the task of extracting the shared region was achieved using PCR followed by Sanger sequencing. This approach amplifies just a target region and sequences the amplicon. The trick is figuring out primer sequences that can amplify the target sequence in all target organisms.

Some of you may have had the opportunity to design PCR primers in your career. The process is to find two sequences that are roughly the right distance apart (i.e., create your desired amplicon size), which are either

side of the DNA you are interested in amplifying, and which are conserved in all the organisms you want to work with (i.e., the primers need to anneal in all the target organisms - not too many mismatches). These sequences should ideally not occur anywhere else in the genome so that you can get a single amplicon and not have to deal with off-target amplicons (primers will amplify anything they anneal to if they follow the rules we applied in our isPCR implementation). In the case of designing 16S primers, the hard part is finding sequences that are conserved enough for primers to anneal in every extant prokaryotic organism.

If you are interested in reading more about how the use of 16S sequences revolutionized not just Microbiology, but all of Biology, I strongly recommend this historical perspective article. Crucially, while the original proposal that 16S sequences could be used to distinguish and classify organisms is approaching 50 years old, our understanding of exactly how to do so is perhaps still developing. Indeed, the primers you have been using for amplifying 16S sequences were recent revisions of what were previously thought to be primers that would amplify all 16S sequences. Instead, both the 515F and the 806R primer were found to not amplify large numbers of organisms and updated within the last decade or so. Perhaps in the future it will be determined that we are missing yet more of the incredible diversity of microorganisms that surround us.

When it comes to extracting 16S sequences from assemblies, we are able to use the primers that have been determined to produce the most reliable amplification of phylogenetically diverse sequences. However, we can't simply PCR sequencing reads. We therefore need to take a different approach. We have begun implementing a read mapping approach. However, this approach is not without challenges.

As described above, designing primers for PCR requires that you find short sequences which are highly conserved. These short, conserved sequences are used as primer sequences so that the primers will anneal to the selected locations. The DNA between those annealing locations can then be amplified. A major advantage of PCR-based approaches for extracting 16S gene sequences is that PCR is completely insensitive to the sequence between the primers. PCR will amplify any DNA regardless of what the sequence is. That means that designing good PCR primers only requires sequence conservation in two very small areas. Everything else is unimportant.

Read mapping, conversely, requires decent sequence conservation across the entire mapping region. Because read mapping relies on the identification of similarity between reads and a reference sequence, any part of the reference which is sufficiently different from the reads will not have reads mapped to it. What this means for 16S sequences is that read mapping only works if the reads come from an organism that is quite closely related to the source of your reference sequence.

Fortunately, we can get around the requirement that the reads must have fairly high sequence identity to the reference. A simple way to do so is to map to multiple references which are each quite different from one another. Mapping software is able to identify multiple candidate mapping locations for each read. It then uses a scoring system to assess which locations are good enough to report, and it compares among the good ones to identify the best. In the last assignment, we saw these "good but not the best" mappings as secondary alignments. If we ignore the secondary alignments then we end up with only the primary read mappings, which ***hopefully*** correspond to the single reference with the highest similarity to the sequenced organisms.

For this week's assignment, your task is to add to your sam module to implement functionality to map reads, identify which reference produced the best mapping, and extract the consensus sequence of the reads to determine the 16S sequence in the sample.

This week's data

I have prepared a set of reference sequences for you to use in "data/refs/16S.fna". These sequences are taken from across the prokaryotic tree of life and include representatives from many major clades of bacteria and archaea. For each reference I have included a FASTA header of the genus and species name followed by a space, followed by the domain and phylum name. These headers are designed so that the genus_species will be present in SAM and other outputs, but you can check the reference FASTA file if you want to know the higher classifications.

I have also provided you with a selection of reads. Each pair of reads files comes from a different bacterial or archaeal isolate. You can map any of these pairs of reads files to the provided reference and experiment with mapping settings to see the challenge described above in action. I have selected reads and references so that each read does have a closely related organism represented in the references file.

Note that these reads were all downloaded from the SRA or ENA databases and then filtered to only keep reads that correspond to the 16S region. I did this to keep this week's dataset at a manageable size and it will not impact your results, but will improve running times and storage requirements. If you would like to test that for yourself you can download the original reads dataset using the accessions in the filenames.

Mapping settings

We don't have time in this course to cover read mapping algorithms in any detail. However, the Github page for the mapping software I am asking you to use this week has a summary of its algorithm as well as a link to the publication where you can find more information about exactly how it maps reads.

If you watched the videos I shared with you early in the course that describe how BLAST works, then you can think of minimap2 as working in a similar way. This is not really correct, but unless you need to know exactly how minimap2 works for your own research, this approximate conceptual framework will serve you just fine for this assignment. Specifically, a not quite right but good enough description of what minimap2 does with short reads (such as Illumina sequencing reads like ours) is:

1. First index the reference sequence into a kmer table. "kmer" simply means a sequence of nucleotides of length k. That means this step finds all the unique sequences of length k in the reference and creates a dict with key of the kmer and value of the list of indices at which that kmer was found.
2. Next the first read is indexed into kmers and each kmer is checked against the reference kmer table.
3. Any matches between these sets of kmers are candidate mapping locations.
4. These kmer matches are then extended by checking if the read base on either side of the kmer matches the reference base either side of the kmer. Any matches or mismatches are scored and this extension proceeds until the score drops below some threshold.
5. After extension is complete, if the alignment was good enough, the alignment is compared to other good alignments and the best is returned as the primary mapping of that read.
6. Steps 2-5 are then repeated for the next read and so on until all reads have been considered.

When using BLAST, we have used different `-task` settings to control things like "word size" which is another name for kmer length. We could also control things like match, mismatch, and gap penalties. You have seen how important those penalties are in your Needleman-Wunsch implementation. We can also control those settings when performing read mapping.

If you imagine (or look at) an alignment of two sequences of high similarity and compare that to an alignment of two sequences of low similarity, you might have an idea of what we might need to do to the settings listed

above in order to increase the number of reads that will map between distantly related reads and reference sequence. Specifically, distantly related sequences have more mismatches, more gaps, and shorter stretches of identical bases (shared kmers). That means that if we want to map reads to a distantly related reference we will want to use a short kmer size to increase the number of alignment seeds and to make sure there are few regions where no reads map to the reference. We will also want to reduce the penalty for gaps and mismatches so that seeds are more readily extended through regions of low sequence identity.

I have empirically determined some mapping settings that seem to work pretty well for the dataset I have provided you. You are welcome to play around with those settings to see if you can get even better mapping or just to become more familiar with how the settings impact read mapping. The settings I settled on are a kmer size of ten (`-k 10` down from a default of 21) and a mismatch penalty of zero (`-B 0` down from a default of 8). I did not find that reducing the gap penalties improved the output, but there might be a configuration that produces better results than the settings I chose. You can check the default values by specifying short-read mode when calling the help message with `minimap2 -x sr --help`. If you don't include `-x sr` then the default values for long reads will be printed instead.

The command you should use for mapping is `minimap2 -ax sr -B 0 -k 10 ref.fasta read1.fastq read2.fastq`. If you find mapping settings that perform better please let me know, but to make grading easier I will require you use these settings so that everyone should produce the same output. Note that the provided command will output to stdout so you should pipe or redirect the output to a file when you run that command.

Assignment details

You may use any number of functions and classes in addition to those specified in the assignment document if you wish. You may not use external Python packages beyond those in the standard library.

1. Write a `SAM` class that stores all of the reads from a SAM file and implements the following methods:
    1. `from_sam`: a `@classmethod` to create an instance of `SAM` from a SAM file where the `SAM` instance also creates a `Read` instance for each (non-header) entry in the SAM file. Only primary mappings of reads should be stored (i.e., no unmapped reads or supplemental/secondary mappings). This method should take the path to a SAM file as input and return an instance of your `SAM` class.
    2. `reads_at_pos`: given a sequence name (corresponding to the RNAME field in the SAM file) and a 1-base position, return a list of all `Read` instances that map to that position in the reference sequence. Note bases corresponding to an "M" or a "D" in the CIGAR string should be considered mapped.
    3. `pileup_at_pos`: given a sequence name (corresponding to the RNAME field in the SAM file) and a 1-base position, return `tuple` of a `list` of the base calls and a `list` of quality scores for those base calls at a given position in the reference. The order of the two lists must be such that the first quality score corresponds to the first base etc. The return type for this method should be `tuple[list[str], list[str]]`, e.g., `(["A", "A", "T"], ["F", "F", "&"])`
    4. `consensus_at_pos`: given a sequence name (corresponding to the RNAME field in the SAM file) and a 1-base position, return the majority base call at a position in the reference as a `str` (this should use a simple majority of >50% to determine consensus).
    5. `consensus`: given a sequence name (corresponding to the RNAME field in the SAM file) return the majority base call for all positions in the specified reference as a `str`. If the sequence name does not exist in the reference or if no reads mapped to it, return an empty `str`.

6. `best_consensus`: return the majority base call for all positions in the reference with the best mapping as a `str`. Best mapping here can be measured as simply the number of positions to which reads map - the more positions are covered, the better the mapping.

For this assignment if there is a tie between any base calls (even between a base and a deletion or one base vs a multi-base insertion), simply put an "N" in that position. N is a character which indicates that the base is unknown as it represents any **N**ucleotide.

2. Write a script called "map_consensus.py" with command line interface and any necessary functions or classes that you need to do the following:
    - Take the following as input:
        - The path to a reference file in FASTA format
        - The path to a read file in FASTQ format
        - The path to a second read file in FASTQ format
        - Optionally, the name of a sequence in the reference FASTA file. This sequence name will have no spaces; it should correspond to the first word after ">" in one of the FASTA headers in the reference sequence file.
    - Perform the following functions:
        - Map the provided reads to the provided reference sequence
        - Print a consensus sequence to the terminal in FASTA format according to the following rules:
            - If the user specified a sequence name in the reference, print the consensus of reads that mapped to the named sequence
            - If the user did not specify a sequence name, print the consensus of reads that mapped to the reference with the best mapping
    - The usage of your script should be `map_consensus.py -1 <READ1> -2 <READ2> -r <REF_SEQS> [-s <SEQ_NAME>]`

Your code should be written to assume that the user has all dependency software in a directory in their $PATH. i.e., don't hard code the path to anything. You may assume the user has the following dependencies installed:

- seqtk
- blast
- minimap2
- samtools

Your tarball should contain a directory with the following structure:

```
.
├── magnumopus
│   ├── __init__.py
│   ├── sam.py
│   └── other modules if you want
└── map_consensus.py
```

Expected output

The sample with accession ERR11767307 should map best to the *Fusibacter paucivorans* reference and produce the following consensus sequence. (You can choose a different FASTA header naming format if you like)

```
>Fusibacter_paucivorans_consensus
AGAGTTTGATCCTGGCTCAGGATGAACGCTGGCGGCGTGCCTAACACATGCAAGTTGAGCGATTTACTTCGGTAAAGAGCGGCGGACG
GGTGAGTAACGCGTGGGTAACCTACCCTGTACACACGGATAACATACCGAAAGGTATGCTAATACGGGATAATATATTTGAGAGGCAT
CTCTTGAATATCAAAGGTGAGCCAGTACAGGATGGACCCGCGTCTGATTAGCTAGTTGGTAAGGTAACGGCTTACCAAGGCGACGATC
AGTAGCCGACCTGAGAGGGTGATCGGCCACATTGGAACTGAGACACGGTCCAAACTCCTACGGGAGGCAGCAGTGGGGAATATTGCAC
AATGGGCGAAAGCCTGATGCAGCAACGCCGCGTGAGTGATGAAGGCCTTCGGGTCGTAAAACTCTGTCCTCAAGGAAGATAATGACGG
TACTTGAGGAGGAAGCCCCGGCTAACTACGTGCCAGCAGCCGCGGTAATACGTAGGGGGCTAGCGTTATCCGGATTTACTGGGCGTAA
AGGGTGCGTAGGCGGTCTTTCAAGTCAGGAGTGAAAGGCTACGGCTCAACCGTAGTAAGCTCTTGAAACTGGGAGACTTGAGTGCAGG
AGAGGAGAGTGGAATTCCTAGTGTAGCGGTGAAATGCGTAGATATTAGGAGGAACACCAGTTGCGAAGGCGGCTCTCTGGACTGTAAC
TGACGCTGAGGCACGAAAGCGTGGGGAGCAAACAGGATTAGATACCCTGGTAGTCCACGCTGTAAACGATGAGTACTAGGTGTCGGGG
GTTACCCNTCGGTGCCGCAGCTAACGCATTAAGTACTCCGCCTGGGAAGTACGCTCGCAAGAGTGAAACTCAAAGGAATTGACGGGGA
CCCGCACAAGTAGCGGAGCATGTGGTTTAATTCGAAGCAACGCGAAGAACCTTACCTAAGCTTGACATCCCAATGACATCTCCTTAAN
GGAGAGTTCCCTTCGGGGACATTGGTGACAGGTGGTGCATGGTTGTCGTCAGCTCGTGTCGTGAGATGTTGGGTTAAGTCCCGCAACG
AGCGCAACCCTTGTCTTTAGTTGCCATCATTAAGTTGGGCACTCTAGAGAGACTGCCAGGGATAACCTGGAGGAAGGTGGGGATGACG
TCAAATCATCATGCCCCTTATGCTTAGGGCTACACACGTGCTACAATGGGTAGTACAGAGGGTTGCCAAGCCGTAAGGTGGAGCTAAT
CCNCTTAAAGCTACTCTCAGTTCGGATTGTAGGCTGAAACTCGCCTACATGAAGCTGGAGTTACTAGTAATCGCAGATCAGAATGCTG
CGGTGAATGCGTTCCCGGGTCTTGTACACACCGCCCGTCACACCACGGGAGTTGGAGACGCCCGAAGCCGATTATCTAACCTTTTGGA
AGAAGTCGTCGAAGGTGGAATCAATAACTGGGGTGAAGTCGTAACAAGGTAGCCGTATCGGAAGGTGCGGCTGGATCACCTCCTT
```