Programming Project 3: Phylogeny

CS 181, Fall 2024

Out: Nov. 3
Due: Nov. 18, 11:59 PM

1 Task

After their visit to Sorin's Farm, the CS 181 TAs have developed a newfound curiosity for American milking goats. They decide to do some reading and find a recent study, which explores the potential of autonomous sampling in environmental DNA (eDNA) research. eDNA is an emerging and powerful method for use in animal research, conservation, and management, yet time- and resource-intensive protocols limit the scale of implementation. The potential of autonomous sampling offers great opportunity for this field, but also introduces the challenge of how to analyze large amounts of data. The CS 181 TAs want to explore the phylogenetic relationships between species identified by the eDNA, but they need your help! They missed the lecture where the UPGMA algorithm was covered, so you will need to implement the UPGMA algorithm as described in lecture. . . keep reading to find out more.

2 Phylogeny Specifications

2.1 Setup

To grab the support code, run `git pull` in your CS181 projects directory. We've provided an example shell script, `upgma.sh` and a few test cases. We encourage you to write your own test cases.

2.2 Reminders About Programming Language

Our Gradescope autograder is currently configured to accept solutions written in Python 2, Python 3, Java, R, and Julia.

Your solutions generally should not require the installation of any packages that do not come in the standard installations of your chosen programming language. However, if you are using Python, you will also have access to numpy and pandas, as well as **graphviz** for writing DOT files. Check graphviz website and user guide for more information regarding its installation and usage. If you'd like to implement the priority queue-based UPGMA algorithm discussed in class, you may also use the **heapq** in Python or manually attach the **liqueueR** package to your handin in R. You are NOT required to do this.

To facilitate anonymized & automated grading, each of your solutions must be accompanied by a shell script. Make sure each problem is able to output the correct result using the shell script provided. The shell script you turn in must explicitly call your language's compiler or interpreter. If you are using Python 3, your shell script must run your program using the command "python3" rather than "python", as "python" will run your code with Python 2.

Your shell scripts should print exactly what is shown in the examples given for each problem. If you print any extra text, you will fail our autograder and lose points. This means that if you are coding in R, you may need to print text using "cat()" instead of "print()".

Be sure to print any terminal output to stdout, which is the channel that the standard print functions write to in most programming languages. If you print to stderr, your solution will be interpreted as an error message and fail the autograder.

2.3 Shell Scripts

You must provide a shell script, `upgma.sh`, that accepts two arguments and can be run using the following command:

```
> sh upgma.sh sample.dist output.dot
```

... where `sample.dist` will be a distance matrix and `output.dot` will be the name of the DOT output file, both of which are described below.

The program you hand in should not throw exceptions for any valid inputs.

2.4 Input Format

The input to the UPGMA algorithm is a distance matrix, represented here as follows. A distance matrix file is given a `.dist` extension, and provides the distance for each pair of species.

Each line has three fields separated by spaces: two species names and a distance value. It will always be **two strings** followed by **a single floating point number**.

For example, the line `b d 3.0` means that the evolutionary distance between species `b` and species `d` is `3.0`. Thus, a sample input might be:

```
a b 5.0
b d 3.0
a d 4.0
```

You can expect the distances to be symmetric and consistent. In other words, you will never be given something like

```
first_species second_species 10.0
...
second_species first_species 5.0
```

2.5 Output Format

Your algorithm's result should be a rooted ultrametric binary tree.

There are two outputs for your program, both of which must be output simultaneously. We will use both of these for grading to allow for partial credit, but the latter should come in handy for debugging as well.

1. A printed output for the hierarchical clustering. *Whenever the algorithm decides to join two nodes with labels `x` and `y` together,* the new node's label should be `(x,y)`. They should be concatenated in **lexicographic order**, so `(y,x)` is considered invalid.

   If two pairs of nodes are tied at any given step, choose the pair whose merged labeling would come first lexicographically.

   Here is an example:

   (a) `a` and `d` gets joined into `(a,d)`.

   (b) `c` and `b` gets joined into `(b,c)`. (note the ordering)

   (c) `(a,d)` and `(b,c)` gets joined into `((a,d),(b,c))`, based on the lexicographical ordering of the strings "`(a,d)`" and "`(b,c)`".

   So for the above hypothetical example, your program should behave as follows:

```
> sh upgma.sh sample.dist tree.dot
((a,d),(b,c))
```

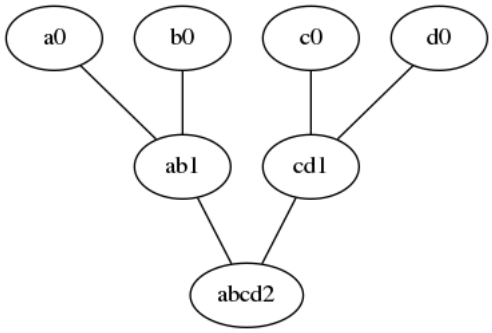
   Notice that two arguments are provided: a distance matrix and an output filename ending in `.dot`.

   Furthermore, note that the printed output should have no whitespace (e.g. no spaces after commas), and that parentheses precede all letters in lexicographic order. So, if `a` and `(b,c)` were merged at some step, the resulting node should have label `((b,c),a)`.

2. The tree in DOT format, saved to the specified file. In the above example, this would be `tree.dot`. To output your graph, your program should write a correct phylogenetic tree in DOT format.

```
graph mytree {
  a0 -- ab1
  b0 -- ab1
  c0 -- cd1
  d0 -- cd1
  ab1 -- abcd2
  cd1 -- abcd2
}
```

   In DOT format, an edge between two vertices is indicated by `--`. The name of the tree (`mytree` in the above example) does not matter. Nodes should be labeled by both their descendant nodes in the same order as the printed output (or the character in the distance matrix, if a leaf node) and their heights in the tree – **that is, the UPGMA height, not graph theoretic height**. This means that based on the UPGMA algorithm's clustering, you should assign each leaf node a height of 0 and each parent node a height of 1 greater than the maximum height of its children, as opposed to assigning heights based on the evolutionary distances specified in the input distance matrix. So the DOT file above corresponds to the following tree.



Note that if your tree has multiple nodes at the same height, you will need to ensure that these nodes have unique names so that the tree is constructed correctly. Nodes must be labeled with descendants followed by height.

Using DOT makes it very easy for you to visualize your output. Say, for example, that your program outputs a file called `tree.dot`. One can easily visualize this tree using `dot` program, by typing into the terminal

```
dot -Tpng tree.dot -o tree.png
```

This will produce a graphic image in PNG format, and save it to a file called `tree.png`.

Alternatively, you can use this online Graphviz visualizer.

If you're not working on a department machine, you can get the `dot` program by installing graphviz.

3 Application

COVID-19 is caused by the virus SARS-CoV-2, but you might have heard of (or remember) earlier viral outbreaks, such as the MERS epidemic from 2012 or the SARS epidemic from 2002. All of these have been caused by a family of viruses known as coronaviruses, which are named for their "crown-like" appearance owing to the spike (S) proteins on their surface. S proteins bind to specific receptor proteins and facilitate the viruses' infection of host cells. Coronaviruses are responsible for certain respiratory diseases in humans, but they also affect other organisms in a variety of ways. Given the diversity of hosts, effects, and viral species, just how related are all of these coronaviruses? In this task you will use your UPGMA algorithm to begin exploring these relationships by constructing a phylogeny.

The set-up script should have provided you with a folder, `application`, containing a set of 5 FASTA files. We've removed all labels from these files, but they contain amino acid sequences corresponding to various spike proteins. Your task will be to construct a phylogeny based on these sequences.

You are free to choose any distance metric you like or devise your own. You may also use your alignment program from PR1 or any of the alignment tools listed below to assist you with obtaining pairwise distances. Be sure to justify your decisions.

- EMBOSS; this toolkit offers several options for alignment algorithms and will present you with percent identity, percent similarity, gap frequency, and alignment scores.

- SIM; this program finds non-intersecting alignments between two sequences and will present you with percent identity, gap frequency, and alignment scores.
- LALIGN; this program offers several options for alignment algorithms and will present you with percent identity, percent similarity, and alignment scores.
- BLASTP, making sure to select "Align two or more sequences"; this program performs the BLAST algorithm (Basic Local Alignment Search Tool) and will present you with percent identity and alignment scores.

Once you have constructed your phylogeny, answer the following questions:

1. What conclusions can you draw about the relationships between samples? Discuss at least two pairs or groups of labels.

   AFTER you've finished analyzing your phylogeny, take a look at the true viral label for each sample, which is shown on the last page of this PDF.

2. Comment on the accuracy of your phylogeny. Do the relationships in your phylogeny appear sensible and consistent with the label assignments? If so, explain your reasoning. If not, provide a possible explanation that attempts to reconcile these discrepancies.
3. Justify the distance metric that you chose (and the alignment parameters, if applicable). Are there any changes that you would make to your workflow? Why or why not?
4. Comment on our decision to use spike protein sequences. Is there an alternative type of data that you would use instead? Why or why not?

You should submit a reproducible description of your workflow with enough detail for us to trace your steps from start to finish, the final phylogeny you generate, and answers to the application questions above in a file named `application.pdf`

4 README

You must include a README file. Include the following information:

- A description of any known bugs
- Anything you want the TAs to know about your project

5 Handin

To hand in this assignment, upload your shell scripts along with all files needed to run your code on Gradescope. If you have any subdirectories or folders that you would like to preserve in your handin, you will need to compress your submission into a zip file and upload the zip file to Gradescope. However, to ensure that the autograder runs your handin properly, make sure that all shell scripts are present at the root of your handin; in other words, do not place your shell scripts inside any folders.

When you hand in your solution, our autograder will immediately run on your submission. We have made the test cases provided in the project handout immediately visible to you so that you can ensure your solution runs properly with the autograder. If you fail the autograder and cannot determine why, notify a TA and we will help you to diagnose the problem. If your final submission is not compatible with our autograder, it will be very difficult for us to give you credit for this assignment.

6 Grading

We will grade your handin using pre-generated test cases with a certain number of points allocated per test case. Test edge cases extensively!

ATTENTION: DO NOT scroll to the next page of the PDF until **AFTER** you have finished implementing UPGMA and responding to the first application question.

Samples and viral labels:

A: Human MERS-CoV spike protein (2013)

B: Ebola spike protein

C: Bat coronavirus spike protein (2020)

D: SARS-CoV-1 spike protein (2003)

E: SARS-CoV-2 spike protein (2020)