CSCI1950-Z Phylogeny Assignment

February 7, 2009

1 Overview

You are presented with a suspected tree topology of the arthropod phylum by none other than Dr. Benjamin J. Raphael, Ph. D. The good doctor has meticulously observed a huge bunch of species to see if there male and female progeny have wings, no wings, or a partial winged growth. Using unknown and possible UNKNOWABLE heuristics, Ben has constructed a binary tree with these species as leaves and ancestors as internal nodes.

Ben wants to know when the winged phenotype emerged in the evolution of the species. Moreover, did this phenotype emerge/mutate more than once?

However, such plodding arithmetic such as the Sankoff and Felsenstein algorithms bores Ben. Your task is: for each internal node of the tree Ben gives you, infer the ancestral state of each node, i.e. the vector

$$\begin{pmatrix} \text{male wing value} \\ \text{female wing value} \end{pmatrix}$$

To do this you will implement both the Sankoff recurrence and the Felsenstein maximum likelihood algorithms over the fixed tree topology.

We have additionally included a DNA-based data set to show you the *power* of your algorithms, but it is a toy example not based on data.

2 Sankoff

You will implement the Sankoff algorithm, which is covered in the slides and reading. Note that our data set will send as a parameter the Fitch-equivalent matrix, but you should code for any general $m \times m$ matrix where there are m possible characters.

The method signature is

```
public void sankoff(LinkedBinaryTree<PSpeciesNode> tree, HashMap<ZCharacter, Integer>
                stateIndices, int[][] costMatrix, int sequenceLength)
```

- `tree` is the fixed tree topology, an instance of `net.datastructures.LinkedBinaryTree`
- `stateIndices` is a mapping from the character states to their matrix indices, and it is guaranteed to have all of the relevant character states and none additional
- `costMatrix` is your scoring matrix
- `sequenceLength` is the length of characters in the sequence vector (for convenience)
- The method stores sequences at the `PSpeciesNodes` (see 5.2). It does not return a value.

In order to get the correct cost for any transition, do the following array lookup:

```
costMatrix[stateIndices.get(parent)][stateIndices.get(child)]
```

3 Felsenstein

Note that this algorithm computes the node labelings such as to get maximum likelihood. The method signature is

```
public void felsenstein(LinkedBinaryTree<PSpeciesNode> tree, HashMap<ZCharacter,
      Integer> stateIndices, double[][] transitionProb, int sequenceLength)
```

- `tree` is as above
- `stateIndices` is as above
- `transitionProb` is the matrix of character state transition probabilities
- `sequenceLength` is the number of characters in the sequence vector (for convenience)

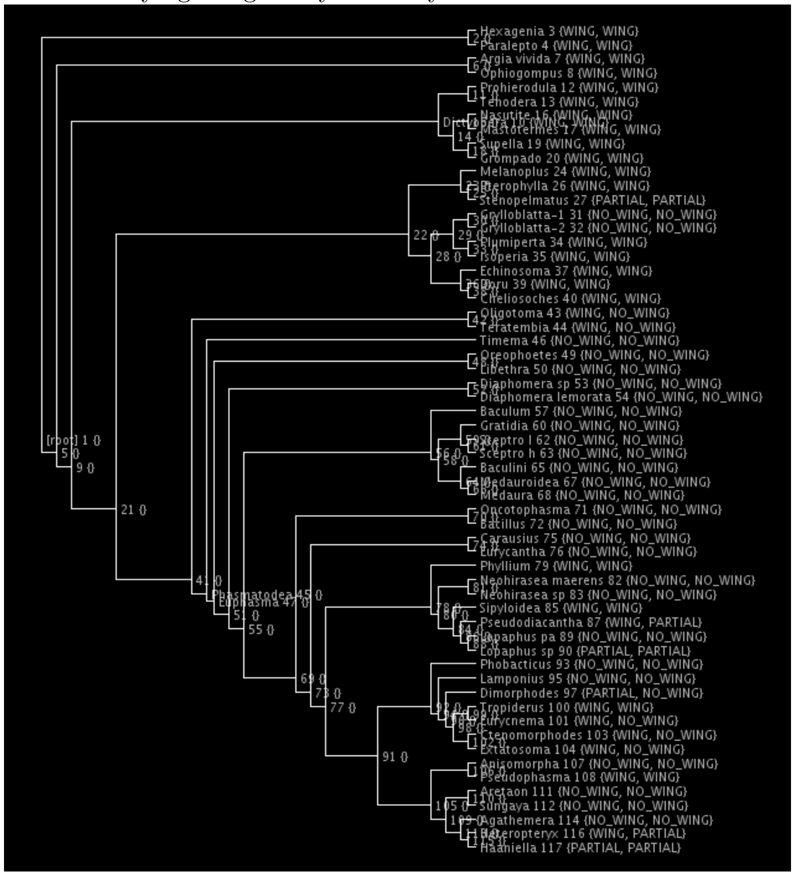
Note that we are calculating $Pr[x \mid y, t]$ slightly differently than shown in the slides for Lecture 2. You are not required to take into consideration the branch lengths t. Thus

$$Pr[x \mid y, t] = \texttt{transitionProb[stateIndices.get(y)][stateIndices.get(x)]}$$

We apologize for the `[y][x]` convention, but this maintains consistency with `[parent][child]` in Sankoff.

4 Winged Data Set

Ben's problem comes from a real study of insects. `ant data` displays the data set. The leaves of the data set are labeled `<Name> <unique integer>` {`<CHAR1>, <CHAR2>`} where `CHAR1` = male phenotype, `CHAR2` = female phenotype. `PARTIAL` indicates partial wing growth. Note that the male and female phenotypes may differ, and this may be part of your analysis. Most of the internal nodes are not labeled – except for `<unique integer>` – because the species are inferred and not known. However, you may want to refer to nodes by their identifying integer in your analysis.


Hexagenia 3 {WING, WING}
Paralepto 4 {WING, WING}
Argia vivida 7 {WING, WING}
Ophiogompus 8 {WING, WING}
Prohierodula 12 {WING, WING}
Tenodera 13 {WING, WING}
Nasutite 16 {WING, WING}
Dictyoptera 10 {WING, WING}
Mastotermes 17 {WING, WING}
14 {}
Supella 19 {WING, WING}
Grompado 20 {WING, WING}
Melanoplus 24 {WING, WING}
Pterophylla 26 {WING, WING}
Stenopelmatus 27 {PARTIAL, PARTIAL}
Grylloblatta-1 31 {NO_WING, NO_WING}
Grylloblatta-2 32 {NO_WING, NO_WING}
22 {}
29 {}
Plumiperta 34 {WING, WING}
28 {}
Isoperia 35 {WING, WING}
Echinosoma 37 {WING, WING}
Doru 39 {WING, WING}
Cheliosoches 40 {WING, WING}
Oligotoma 43 {WING, NO_WING}
Teratembia 44 {WING, NO_WING}
Timema 46 {NO_WING, NO_WING}
Oreophoetes 49 {NO_WING, NO_WING}
Libethra 50 {NO_WING, NO_WING}
Diaphomera sp 53 {NO_WING, NO_WING}
Diaphomera lemorata 54 {NO_WING, NO_WING}
Baculum 57 {NO_WING, NO_WING}
Gratidia 60 {NO_WING, NO_WING}
Sceptro l 62 {NO_WING, NO_WING}
[root] 1 {}
5 {}
56 {}
Sceptro h 63 {NO_WING, NO_WING}
9 {}
58 {}
Baculini 65 {NO_WING, NO_WING}
Medauroidea 67 {NO_WING, NO_WING}
Medaura 68 {NO_WING, NO_WING}
21 {}
Oncotophasma 71 {NO_WING, NO_WING}
Bacillus 72 {NO_WING, NO_WING}
Carausius 75 {NO_WING, NO_WING}
Eurycantha 76 {NO_WING, NO_WING}
Phyllium 79 {WING, WING}
41 {}
Phasmatodea 45 {}
Neohirasea maerens 82 {NO_WING, NO_WING}
Euphasma 47 {}
Neohirasea sp 83 {NO_WING, NO_WING}
51 {}
78 {}
Sipyloidea 85 {WING, WING}
55 {}
80 {}
Pseudodiacantha 87 {WING, PARTIAL}
84 {}
Lopaphus pa 89 {NO_WING, NO_WING}
Lopaphus sp 90 {PARTIAL, PARTIAL}
Phobacticus 93 {NO_WING, NO_WING}
69 {}
Lamponius 95 {NO_WING, NO_WING}
73 {}
Dimorphodes 97 {PARTIAL, NO_WING}
77 {}
92 {}
Tropiderus 100 {WING, WING}
Eurycnema 101 {WING, NO_WING}
98 {}
Ctenomorphodes 103 {WING, NO_WING}
102 {}
Extatosoma 104 {WING, NO_WING}
91 {}
Anisomorpha 107 {NO_WING, NO_WING}
106 {}
Pseudophasma 108 {WING, WING}
Aretaon 111 {NO_WING, NO_WING}
110 {}
105 {}
Sungaya 112 {NO_WING, NO_WING}
109 {}
Agathemera 114 {NO_WING, NO_WING}
Heteropteryx 116 {WING, PARTIAL}
115 {}
Haaniella 117 {PARTIAL, PARTIAL}

Summary: run `ant data`

5 Implementation Detail

5.1 Your Code

You need only write your algorithms to `src/cs195z/phylogeny/PStudentTreeAlgorithms.java`. You may write anything you want to that directory. No modifications to the support code are necessary, but the source is included.

5.2 CS195Z Support Code

Five classes make up the `cs195.support` package. Javadocs are available on the website's "Assignments" page.

- `PDataSet` generates the data set.
- `PSpeciesNode` is the main class that you are going to interact with! It stores the sequence represented by each species. It also includes separate storage arrays for Sankoff costs and Felsenstein likelihoods. All of the results of your algorithm will reside in this class. The leaf nodes will start with a character sequence that you `getSequence()`; at internal nodes you will either `add` to the back or `set` the sequence. It is **imperative** that you familiarize yourself with the Javadoc of this class first.
- `PTreeAlgorithms` is the interface for your code.
- `PTreeVisualizer` starts the visualizer for your tree after you label it.
- `ZCharacter` is an enum of biological characters. The ones used in the Ben's problem data set are `WING, NO_WING, PARTIAL`, but any of these characters are valid for a sequence.

5.3 Other Code

5.3.1 NDS4 Data Structures

The `LinkedBinaryTree` originates in the NDS4 package. Please refer to the link to the Javadoc on the Assignments page. Use the methods that will make it easiest to code! Each position in the tree holds an element of type `PSpeciesNode`. You do not need to modify the tree, but you will use it to get references to the correct node elements. Your implementation will leave its results in the `PSpeciesNodes`.

5.3.2 A Tree Viewer (ATV)

You need not worry about ATV, which is a wonderful tree viewer with lots of features. The support code automatically translates your `LinkedBinaryTree` into the format for ATV.

6 Running It!

6.1 Code Acquisition

The code and ant script is in the `/course/cs195z/pub` directory on the CS servers. We recommend the following, first making yourself a cs195z directory in the place of your choice:

```
mkdir cs195z
cp -r /course/cs195z/pub/* cs195z/
```

Once you have the files, you can develop on any system you want. If you object to acquiring the files from the CS department, apstewar can email them to you (but they will not be posted to the website).

The project requires `cs195zlib.jar` in your classpath. If you use the included ant script then you do not need to worry about the classpath. You can omit copying `cs195zlib.jar` if you work on the department servers, otherwise just copy it and it will work.

6.2 Compiling and Running

An Ant build script is included to compile and then run the visualizer. In the directory with the `build.xml` file, run one of the following commands:

- `ant data` Displays the input data set. You can do this first for a flavor of the problem. If you are on the department servers, this will work even if your code won't compile.
- `ant sankoff` Runs the Sankoff algorithm then displays it.
- `ant felsenstein` Runs the Felsenstein algorithm then displays it.
- `ant dataDNA`,`ant sankoffDNA`, and `ant felsensteinDNA` test the additional DNA data set.

7 Submission

The project is due by 11:59pm on Monday, February 23.

You will submit it by emailing it to `cs195ztas@cs.brown.edu` – please do not use the headta alias as it will inundate the staff. Include the following:

1. The minimal source code `.java` to complete the project. No need to send libraries/support code.
2. A README file with (1) a detailed paragraph answering Ben's question (**see 1, bold**) on the winged datas et, (2) 2-3 sentences about your code structure, (3) 3-4 sentences comparing the results of the Sankoff and Felsenstein runs with explanation and (4) optionally any bugs or other concerns.

8 Evaluation

Evaluation is performed on the following four bases:

1. Correctness (50%) – The algorithm works on arbitrary (but valid) test data sets and provides in each case a minimum parsimony score or maximum likelihood assignment solution (objective).
2. Logical adherence to algorithm (30%) – We should be able to clearly identify how your code performs precisely the expected algorithm. We take into consideration appropriate commenting, good structure, and no introduced inefficiencies.
3. Generality (15%) – Your implementation of the algorithm should work for the general case of input and all valid edge cases.
4. README (5%) – Must exist and follow the above specification.

Note that performance is not part of the evaluation, but poor performance could be a result of a problem with 2.

9 Errata

- There could be multiple labelings that achieve min-parsimony score or max-likelihood, so just choose one. We will take this into account.
- All the trees you are responsible for have internal nodes with two children and leaves with zero children.
- You don't have to do null checking ad nauseum - it is guaranteed that your HashMaps will take into account all of the ZCharacters you need and none you don't, etc.
- Please, please implement the traceback without using pointers.
- Good luck!