Algorithms for Bioinformatics: Project 3
Python module for Automatic Sequence Analysis

Miguel Sozinho Ramalho, UP201403027

June 18^{th}, 2019

1 Introduction

The current report describes the architecture and underlying concepts of a Python3 module whose purpose is the implementation of a *bioinformatics* pipeline. The module should be able to handle biological sequences, perform BLAST[1] operations on a FASTA[2] database, calculate the result of Multiple Sequences Alignment (MSA), produce distance matrices between a list of biological sequences and use this matrix to perform Unweighted Pair Group Method with Arithmetic mean (UPGMA[3]) clustering and generate phylogenetic trees, and also produce draw and unweighted graphs from those distance matrices by specifying a maximum distance threshold for two sequences (that ultimately represent a species) to be connected.

This module is a new version of the previous practical work and so all the base concepts of biological sequences and the base functionality of reading/handling them is assumed to be common ground upon which the new version has been built, as well as the Dynamic Programming techniques used to produce global and local alignments.

A Jupyter Notebook (run_me.ipynb) was created to exemplify the use of the developed functionality (a static version is available as run_me.html) along with unit tests that guarantee the correctness of the software through time and horizontal changes. Documentation was also made and is available in the *docs* folder (open the *docs/_build/html/index.html* file or type **help(bioseq.moduleName)** after installation).

2 Implementation and Results

This project was particularly challenging due to a concern for simple and fast implementation for algorithms, such as the UPGMA. The visualisation of phylogenetic trees and distance graphs was also a novelty and required the use of some new libraries (Phylo[4] and Networkx[5]). It should be noted that all external libraries used are included in the requirements.txt file which can be used to prepare the python environment with the command **pip install -r requirements.txt**. TQDM[6] was used to provide progress bar feedback in typically long operations. Some further considerations for each building block of the pipeline are included below:

[1] `https://blast.ncbi.nlm.nih.gov/Blast.cgi`
[2] `https://en.wikipedia.org/wiki/FASTA`
[3] `https://en.wikipedia.org/wiki/UPGMA`
[4] `https://biopython.org/wiki/Phylo`
[5] `https://networkx.github.io/`
[6] `https://github.com/tqdm/tqdm`

2.0.1 BLASTA

- The database can be loaded from a FASTA file
- When performing the search, sequences matching the query (same species) are discarded
- A "blast" object can be built with a given database and then reused so that the process of building the database is not executed every time a search must be done
- The process of expanding the matching subsequences is done first forward and then backward without code duplication as the backward steps are done forward on the reversed sequences
- A custom threshold can be specified (instead of the static "half" rule) to stop the expanding process
- All sequences in the database are scored against the query and then the top n are returned (by default n is 10), this is considered to be better than an iterative search of the best match (which inevitably has to score all functions in each iteration)
- The score used between sequences was that of the ratio of matches instead of the number of matches, as pragmatically this led to a better result segregation

2.1 MSA

- For cases where the global alignment returns multiple results only the first is considered, this was not considered harmful for the results as the other alignments had the same score and because there is no a priori manner of knowing which alignment is better (time constraints and result interpretation complexity also motivated this decision)

2.2 Phylogenetic Trees (ptrees)

- The input for the ptree generation can either be previously aligned sequences or not yet aligned ones, using the same function call (this is automatically detected)
- The Phylo module is used to display the ASCII version of the ptrees as well as the visual ones. This required the implementation of a method that would convert our ptrees into an XML version that Phylo could read[7], but that can also useful for other applications

3 Results

As apendix, there are some pictures outlying the pipeline blocks that are considered interesting or relevant for demonstration purposes. All the used parameters are dynamic so only their default values are included (no default makes sense for the cut value but these can be seen in the run_me instance [.py, .html, .ipynb]) in Table 1.

4 Conclusions

All the initial goals were met and a lot was learned both from a bioinformatics perspective as well as algorithmic and knowledge on available libraries.

[7] `https://biopython.org/wiki/PhyloXML`

Section	Parameter	Default Value	Details
Blast	top	10	how many to return
Blast	threshold	0.5	when to stop expanding subsequence
Graph	cut	NA	include only distances below this value

Table 1: Parameter default values

Appendix

```
# blosum and gap penalty of -1
p = PTree(blosum, -1)
```

```
mx = p.distance_matrix(blast_seqs)
print(mx)
```

```
100%|██████████| 55/55 [04:07<00:00,  5.31s/it]
```

```
0  8 17 19 44 51 50 50 47 49 15
0  0  9 11 36 43 42 42 39 41  7
0  0  0 14 36 43 42 42 39 41 12
0  0  0  0 35 42 41 41 38 40 14
0  0  0  0  0 47 46 46 43 46 37
0  0  0  0  0  0  3  5  6  9 45
0  0  0  0  0  0  0  2  3  6 44
0  0  0  0  0  0  0  0  3  6 44
0  0  0  0  0  0  0  0  0  3 41
0  0  0  0  0  0  0  0  0  0 43
0  0  0  0  0  0  0  0  0  0  0
```

Figure 1: Distance Matrix

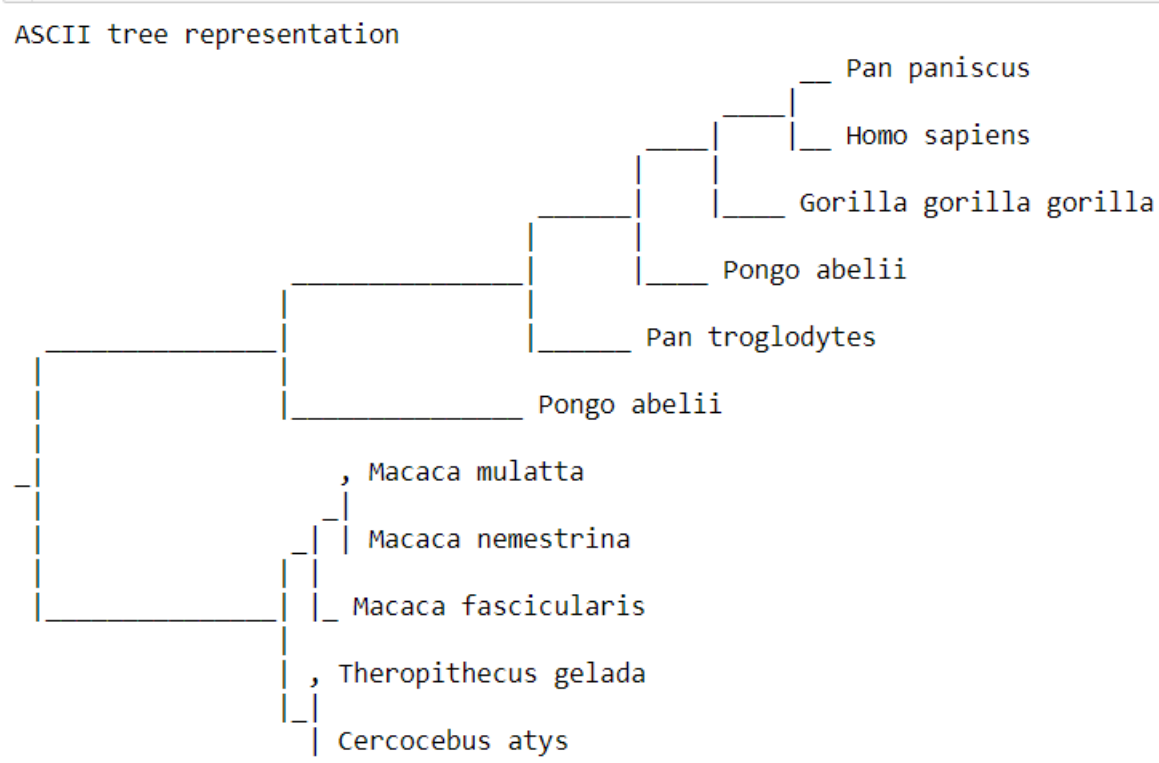

Figure 2: ASCII output of the phylogenetic tree

```
print("drawing the tree with Phylo:")
p.draw()
```

```
drawing the tree with Phylo:
```

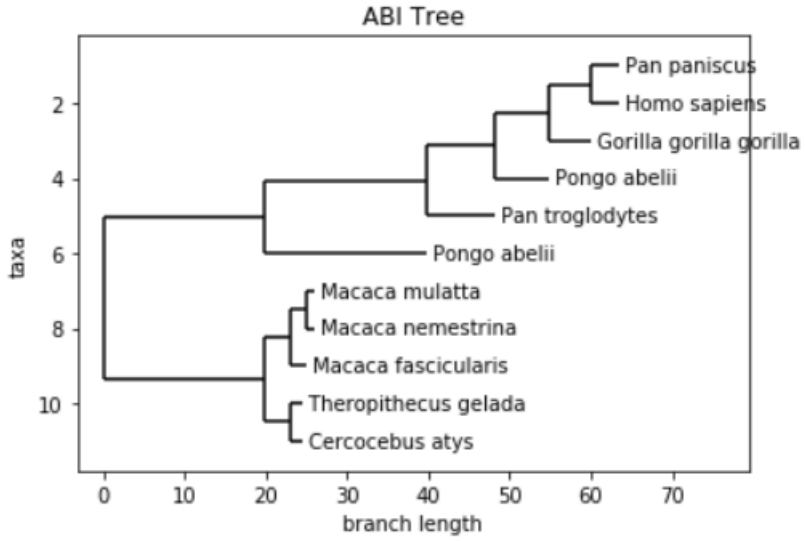

Figure 3: Dynamic visualization of a phylogenetic tree

```
mx.graph(cut=40)
```

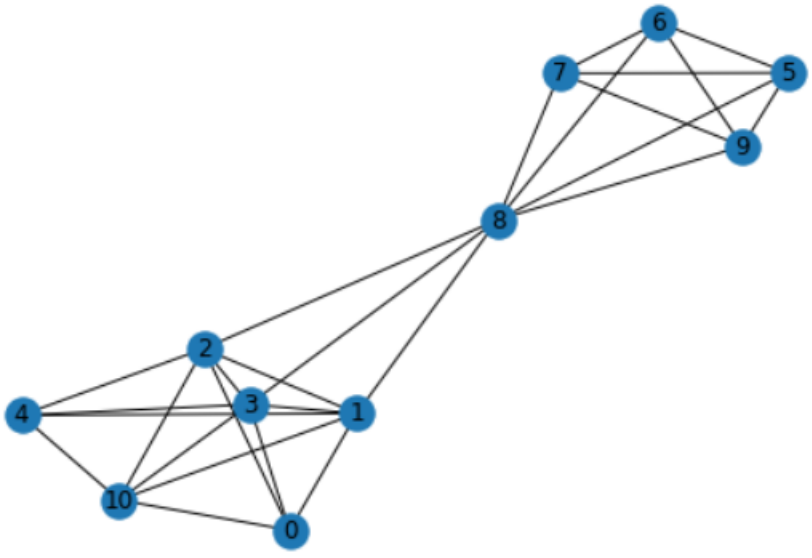

Figure 4: Example of one of the generated graphs