**Comparative Genomics 2018**

**Practical 4: Phylogenomics**

Group 11

Fuqi Xu, Milda Valiukonyte, Shuhan Xu

**Summary**

<Text>

**Exercise 1**

Ortholog clusters from practical 4 is used for this practical: see attachment cluster

**Exercise 2**

1.

Gene order output for 09.fa.txt: 09\_gene\_order

Gene order output for 17.fa.txt: 17\_gene\_order

Gene order output for 49.fa.txt: 49\_gene\_order

Gene order output for 51.fa.txt: 51\_gene\_order

2.

a.

There can be ambiguities in clustering.

For genome 09.fa.txt, no genes will appear in more than one clusters since this genome is the reference genome and is used as the query to BLAST search against other genomes. Hence each cluster contain a unique 09.fa.txt gene.

For 17.fa.txt, 49.fa.txt and 51.fa.txt, there may be genes appearing in more than one cluster since each of these genomes are used as reference database in the BLAST search. For instance, two different genes from 09.fa.txt may have the same best-scoring BLAST hit in 17.fa.txt. As a result, two clusters may have different genes from 09.fa.txt but the same gene from 17.fa.txt.

If a gene appears in more than one cluster, the script will assign to it the cluster number of the first cluster in the list in which the gene appears. Hence, if the gene appears in subsequent clusters, these clusters will not have that gene after the script is run.

b.

This script does not handle forward and reverse strandedness in the gene order list. To implement this functionality in the script, the script can add a positive sign ‘+’ in front of the cluster number if the gene appears in the forward strand and a negative sign ‘-’ if the gene appears in the reverse strand. This can be done in the last loop of the script:

for aGene in geneOrderList:

if partOfCluster.has\_key (aGene):

if aGene.endswith(‘rev’):

print ‘-’ + partOfCluster [aGene],

else:

print ‘+’ + partOfCluster [aGene],

Downstream scripts can check the sign and deal with the situation accordingly, e.g. generate reverse complement if the sign is negative.

**Exercise 3**

3., 4., 5., and 6.

We wrote a script to perform the tasks in question 3, 4, 5 and 6. This script incorporates rndseq.py

Script for generating pseudo-genomes from gene order files: see attachment generate\_pseudogenomes.py

Usage: python3 generate\_pseudogenomes.py <number of ORFs in largest proteome> <gene order file 1> <gene order file 2> <gene order file 3> …

Description: The script first generates a list of pseudo-genes (from rndseq.py code in the script) and converts it to a dictionary. Then, for every gene order file, it assigns a pseudo gene to each cluster number and concatenates the pseudo-genes into a single long sequence (pseudo-genome). Finally, it combines all the pseudo-genomes into one multi FASTA file called ‘multiFastafile’ in the working directory.

Multi FASTA file output from script: see attachment multiFastafile

**Exercise 4**

1.

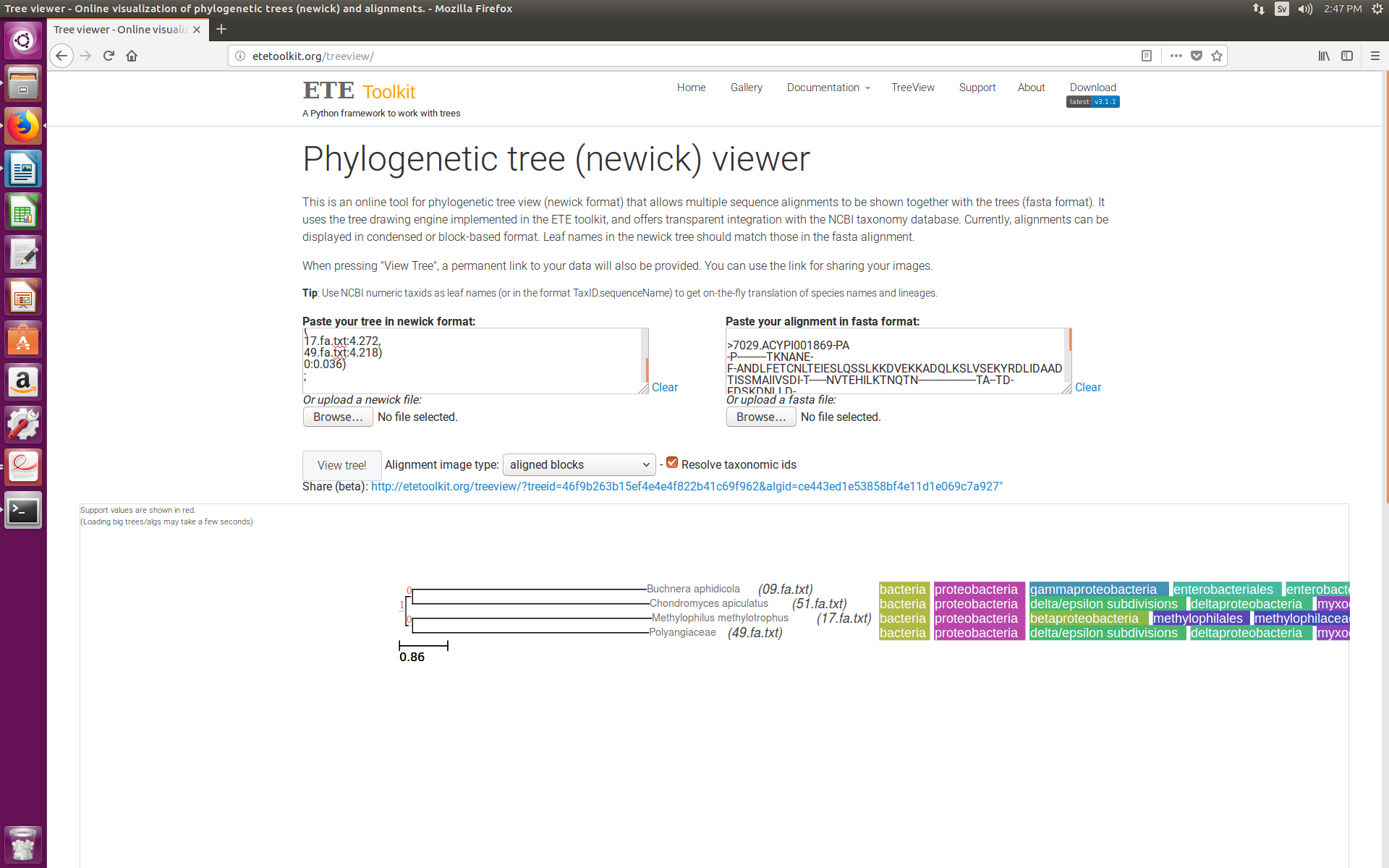
Formatted gene order file for GRIMM: see attachment grim\_genomes.txt

2.

Distance matrix from GRIMM: see attachment distance.grim

3.

Reconstructed Species Tree using distance matrix from GRIMM:



4.

Consensus Tree for 10 orthologs clusters from Practical 4:

