PHANS-C Documentation

Release 1.0

Jaclyn Saunders

Apr 20, 2020

Contents:

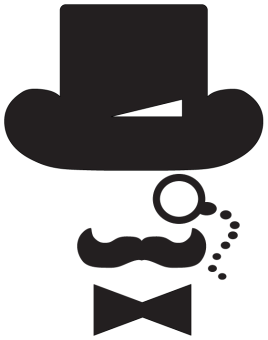
1 Overview of PHANS-C Pipeline **3**

2 Citations for using PHANS-C **7**

3 How to run PHANS-C **9**
- 3.1 Amazon AWS AMI . . . 9
- 3.2 Docker Image . . . 9
- 3.3 Local Installation . . . 10

4 PHANS-C Tutorial **13**
- 4.1 Get the tutorial folder . . . 13
- 4.2 The Tutorial . . . 13
- 4.3 **1**: Construct the Phylogenetic Reference Tree . . . 14
- 4.4 **2**: Parse and Format reads from Metagenome . . . 16
- 4.5 **3**: Align unknown reads to Reference Alignment . . . 20
- 4.6 **4**: Phylogenetic placement of unknown reads . . . 23
- 4.7 **5**: Final Summary . . . 35

5 License **37**



Welcome to the **PHANS-C Pipeline**: PHylogenetic Assignment of Next generation Sequences - in the Cloud.

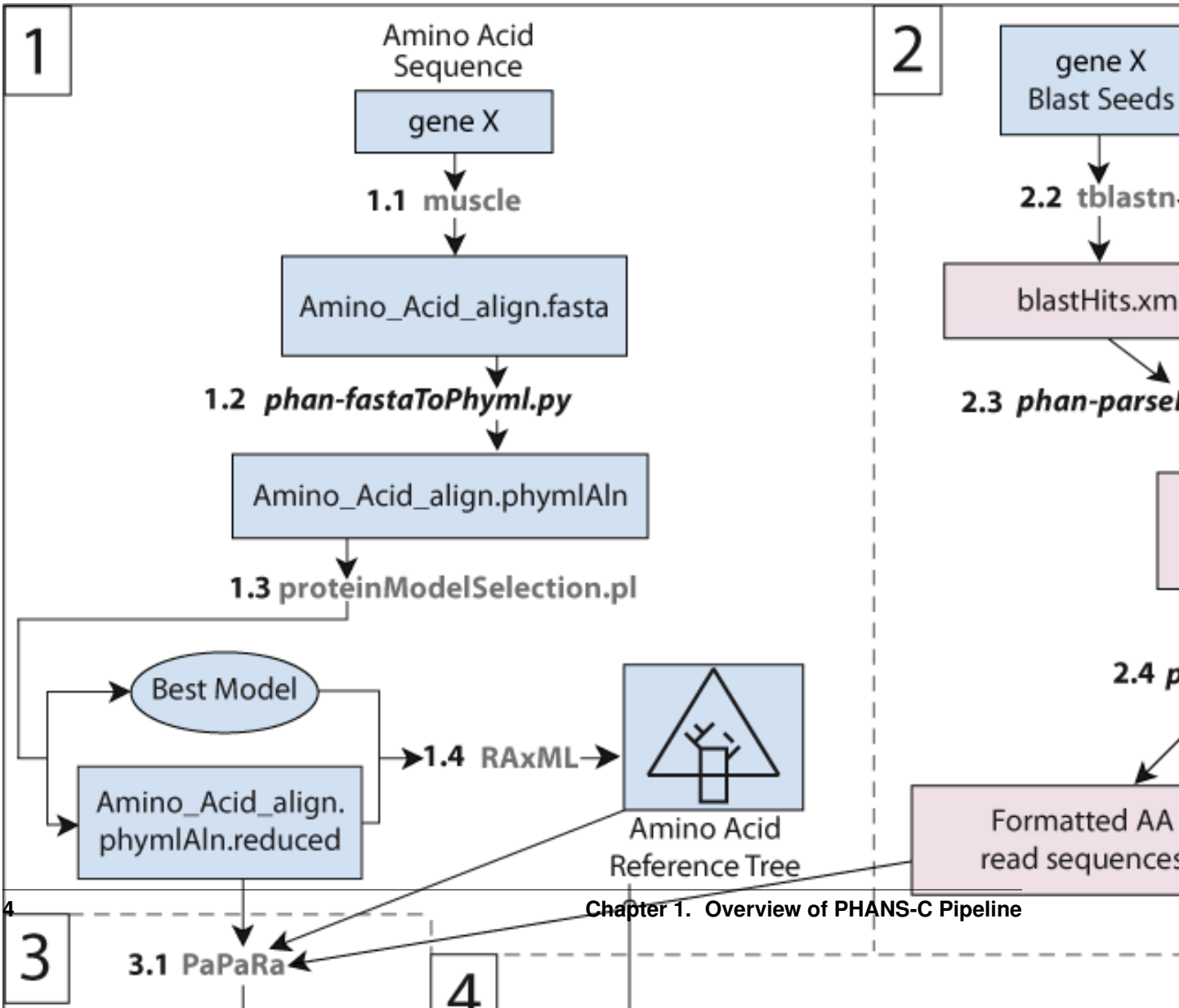
PHANS-C performs inference-based placement analysis of targeted protein coding genes in amino acid space in very large metagenomic datasets, with a focus on differentiating between closely related organisms. PHANS-C can also be used in nucleotide space, with metatranscriptomic sequencing data, and for analysis of broad taxonomic placement. The PHANS-C pipeline tools presented here are implemented in Python 3.6 on Linux based OS and work in conjunction with previously published open source software.

The complete pipeline can be found on Amazon Web Services (AWS) us-west-2 Amazon Machine Image (AMI) PHANS-C v 1.0.2 **ami-09bbe25d9d7608ea5**. This AMI has all of the PHANS-C scripts installed, along with all additional software cited and dependencies.

Original PHANS-C Pipeline code can be found at Bitbucket.

CHAPTER 1

Overview of PHANS-C Pipeline



A flowchart of the bioinformatics pipeline. Objects represent files, while connecting arrows are actions on objects either through software or manual curation. Gray text between arrowed lines represents previously existing software while bold italicized black text represents interconnecting scripts described here in the PHANS-C pipeline. The PHANS-C pipeline is broken up into four major parts: 1. Full length reference sequences for the gene of interest are aligned and a reference tree is created; 2. Short reads are parsed from a large metagenomic database using local-alignment or mapping based tools. Then the recruited reads are cleaned and formatted for phylogenetic inference; 3. The unknown reads are aligned to the reference alignment; 4. Unknown aligned reads are phylogenetically placed to their least common ancestor on the reference gene tree, reads are labeled accordingly and tabulated.

Below is a table of the various programs in the PHANS-C Pipeline along with brief descriptions of the function of the program in the pipeline as well as associated citation indicating whether the program is new and specific to this work [2] or previously available academic software.

Program	Description of Function in Pipeline
phan-makeblastdb.py	Converts a fasta file of metagenomic sequence read
tblastn	Searches a 6-frame translation of a BLAST database
phan-parseBlastXML.py	Parses information from BLAST search results and re
phan-trimFormatReads.py	Selects the best match from the BLAST query, ensur trims reads to reduce potential overhang of up/dow sequence, and reports information on recruited rea
muscle	Alignment program used for reference sequences
phan-fastaToPhyml.py	Converts a fasta formatted file to Phyml format
proteinModelSelection	RAxML-based program to find the optimal amino aci
RAxML	Maximum Likelihood phylogenetic tree constructio
PaPaRa	Phlogenetically aware alignment program for aligni
phan-spaceJoin.py	Combines read pairs which are known to be linked a reads to be analyzed for placement
RAxML-EPA	Maximum likelihood based assignment of unknown
phan-distributions.py	Produces interactive visualization of placement qua
phan-figTreeFiles.py	Reformats placement tree into newick-like format t
phan-taxaDict.py	Creates a file where individual nodes on a tree can l
phan-treeLabels.py	Labels read assignments based upon node labels de
phan-sortPlacements.py	Sorts placed reads by different metagenome sample

Citation numbers correspond to citations listed below.

CHAPTER 2

Citations for using PHANS-C

Please cite the following papers if using this pipeline:

[1] MUSCLE: Edgar, R. C. (2004). MUSCLE: multiple sequence alignment with high accuracy and high throughput. Nucleic Acids Research, 32(5), 1792-1797. doi: 10.1093/nar/gkh340

[2] PHANS-C Pipeline: Saunders, J. K., McKay, C., Rocap, G. (*submitted*). PHANS-C Pipeline: PHylogenetic Assignment of Next generation Sequences - in the Cloud.

[3] RAxML: Stamatakis, A. (2014). RAxML version 8: a tool for phylogenetic analysis and post-analysis of large phylogenies. Bioinformatics, 30(9), 1312-1313.

[4] BLAST: Altschul, S. F., Madden, T. L., Schaffer, A. A., Zhang, J., Zhang, Z., Miller, W., & Lipman, D. J. (1997). Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. Nucleic Acids Res, 25(17), 3389-3402.

[5] PaPaRa: Berger, S. A., & Stamatakis, A. (2012). PaPaRa 2.0: a vectorized algorithm for probabilistic phylogeny-aware alignment extension. Heidelberg Institute for Theoretical Studies, https://sco.h-its.org/exelixis/pubs/Exelixis-RRDR-2012-5.pdf.

[6] RAxML Evolutionary Placement Algorithm EPA: Berger, S. A., Krompass, D. & Stamatakis, A. (2011). Performance, Accuracy, and Web Server for Evolutionary Placement of Short Sequence Reads under Maximum Likelihood. Systematic Biology (60) 291-302. doi:10.1093/sysbio/syr010

[7] Biopython: Cock, P. J. A., Antao, T., Chang, J. T., Chapman, B. A., Cox, C. J., Dalke, A., ... de Hoon, M. J. L. (2009). Biopython: freely available Python tools for computational molecular biology and bioinformatics. Bioinformatics, 25(11), 1422-1423. doi: 10.1093/bioinformatics/btp163

CHAPTER 3

How to run PHANS-C

We offer 3 ways you can get and run PHANS-C.

1. Amazon AWS AMI
2. Docker Image
3. Local Installation

3.1 Amazon AWS AMI

3.1.1 Start the instance from an AMI

Spin up an EC2 instance using the Amazon AMI PHANS-C v 1.0.2 **ami-09bbe25d9d7608ea5** hosted in **us-west-2**. *Recommend using the c4.4xlarge instance type*

Will need to ssh into instance using user id "ubuntu" (as opposed to ec2-user, etc.).:

```
$ ssh -i "your-ec2-key.pem" ubuntu@##.###.###.##
```

Activate the appropriate conda environment as follows:

```
$ conda activate phansc
```

- AWS users will find the tutorial folder located in the ubuntu user's home directory.

3.2 Docker Image

3.2.1 Start the Docker container from the image:

On a machine with Docker already installed:

First download and expand the PHANS-C_Tutorial Folder, then do this:

```
$ docker run -v /Local/path/to/work/folder:/Tutorial -it rocaplab/phansc
```

The `'-v'` option allows you to designate a directory that is shared between the host machine, and the docker image.

The `'-it'` option opens the Docker image with an interactive shell.

Initiating Docker this way calls upon the latest Docker Image available for PHANS-C in Docker Hub at `rocaplab/phansc`.

Any changes inside the container's `/Tutorial` directory will instanty be reflected in the host's `/Local/path/to/work/folder` directory as well. In this way, a directory can be shared between the host and the container.

- A note on running from a Windows OS: it is recommended to specify the path in quotes, for example, `-v "C:\Local\path\to\work\folder":/Tutorial`.

This will drop you into a command prompt inside a centos based container (similar to a virtual machine). You can then navigate to the tutorial folder:

```
cd /Tutorial
```

3.3 Local Installation

The PHANS-C Pipeline scripts can be found in our Bitbucket repository and have been written and implemented using Python. However, the Pipeline depends on multiple other open source software written in multiple languages with their own dependencies and requirements. the PHANS-C scripts themselves should be installed somehere in your PATH, and made executable, so that they can be called from anywhere. The following are some very brief tips that may help you install dependencies locally.

3.3.1 Build with all dependencies using .yml file

You can build a conda environment using the provided .yml file.

```
conda env create -f phans-c_environment.yml
```

Alternatively, you can add packages individually as described below.

3.3.2 Biopython

Biopython can be found through the Biopython website. The PHANS-C Pipeline was developed with version 1.68

There are many ways to install Biopython, including MacPorts, Homebrew, Pip, and Bioconda. We installed ours with Bioconda.

```
conda install python
conda install biopython
```

3.3.3 Muscle

The PHANS-C Pipeline was created using Muscle version 3.8.31. The latest version of Muscle as well as older versions can be found at the Muscle website. We installed ours with Bioconda.

```
conda install muscle
```

3.3.4 RAxML

RAxML and the RAxML Evolutionary Placement Algorithm are located in the same executable. The PHANS-C Pipeline was created using RAxML version 8.2.8 and has been tested successfully with RAxML version 8.2.11. A typical install of RAxML may install several different binaries. Which is best may depend on your architecture. Try raxmlHPC-PTHREADS-AVX first, as this is the fastest and multi-core capable. If that doesn't work, experiment with some of the other options. The most recent version of RAxML can be found from from the Exelixis Lab GitHub repository or from the Exelixis Lab homepage. We installed ours with Bioconda.

```
conda install raxml
```

3.3.5 proteinModelSelection.pl

A perl script, proteinModelSelection.pl, for identifying the best AA substitution model for tree generation was created by the RAxML author Alexis Stamatakis and can be found at the RAxML site. You may have to replace the shebang ("#!") line at the top of the script to point to your perl location. You may also have the change the raxml executable name to match the one on your system. Remember to make it executable after you put it into your PATH.

3.3.6 BLAST

We installed BLAST with Bioconda.

```
conda install blast
```

3.3.7 PaPaRa

PaPaRa 2 can be found through the Exelixis Lab homepage or through Simon Berger's GitHub repository. The PHANS-C Pipeline uses PaPaRa version 2.5.

We had a lot of trouble getting PaPaRa working on MacOS High Sierra (10.13). The precompiled version of PaPaRa offered by the Exelixis Lab did not work for us. Compiling manually did not work at first because of boost compatibility problems. Manually installing boost, as well as installing with homebrew and macports failed. We were finally able to get PaPaRa to compile by installing a particular version of boost with conda. This was how we were successful, your mileage may vary.

```
#Assumes you have Bioconda and an environment setup and activated
conda install boost=1.64
cd /path/to/papara/source/directory
ln -s /path/to/your/conda/installation/your_env/include/boost/ ./boost
sh build_papara2.sh
cp ./papara /path/to/your/bin/dir
```

The PaPaRa precompiled binary worked our our Ubuntu Linux 18.04 system, but bear in mind it is not compiled for multiple threads. For that, you must compile your own.

3.3.8 Bokeh

Bokeh can be found on the Bokeh website. We installed ours with Bioconda.

```
conda install bokeh
```

3.3.9 Pandas

Pandas can be found on the Pandas website. We installed ours with Bioconda.

```
conda install pandas
```

3.3.10 Access the tutorial files:

The `Tutorial` folder is included in the repository.

CHAPTER 4

PHANS-C Tutorial

For those who want to hit the ground running...

You can either use a pre-made Amazon AWS AMI, Docker image, or install PHANS-C locally.

For tips getting PHANS-C running, and getting the tutorial folder, check out Get PHANS-C.

4.1 Get the tutorial folder

- AWS users will find the folder in `/home/ubuntu/Tutorial`
- Docker users will need to download the Tutorial folder, then mount it as a shared folder (see Get PHANS-C)
- Local users will find the tutorial folder in the main PHANS-C repository download.

4.2 The Tutorial

The order of the tutorial steps roughly follow the flow diagram of the pipeline found here. Open the Tutorial directory. From here on out, paths are relative to the Tutorial directory.:

If you prefer, there is a shell script which automates the following process. However, we recommend that you manually run through the steps to gain a greater understanding. If you prefer to use the shell script for the tutorial data, it can be run as follows:

```
$ sh run_tutorial.sh
```

Warning: If you installed PHANS-C and prerequisites manually, rather than using the AWS image, it is very common to have to slightly modify some scripts to get the workflow running. For example, the raxml binary can be called different things (e.g. `raxml`, `raxmlHPC`, `raxmlHPC-AVX`, `raxml-AVX2`, `raxml_8.2.11`, etc) depending on how it was installed. You may need to open the run_tutorial.sh and `ProteinModelSelection.pl` scripts and adjust the name of the raxml binary. Be on the lookout for other similar errors. Another example is the third party `ProteinModelSelection.pl` script may need to have the perl path modified on the first line of the script.

4.3 1: Construct the Phylogenetic Reference Tree

First, you will need to assemble a list of full length gene sequences which will act as known reference sequence which will be used to compare the unknown metagenomic sequences. For this tutorial, we will identify and categorize sequences of the ribosomal protein *rpsD*. Full length *rpsD* reference sequences in amino acid space can be found in the file `AA/rpsD_AA.fasta`.

4.3.1 1.1: Make reference sequence alignment

Move into the AA directory:

```
$ cd AA/
```

Generate an alignment for the reference *rpsD* sequences using the program muscle.

```
$ muscle -in rpsD_AA.fasta -out rpsD_AA_aln.fasta
```

4.3.2 1.2: Convert alignment to phymlAln format

Convert from fasta to phymlAln format for tree construction.

```
$ phan-fastaToPhyml.py -f rpsD_AA_aln.fasta
```

phan-fastaToPhyml.py command line options:

-h, --help	Show this help message and exit.
-f FILE	Specify the FASTA in-file that contains the sequence alignment.

4.3.3 1.3: Find best amino acid substitution model for reference tree

Since we are using amino acid space, you may want to test for the best amino acid substitution model. However, you may use any model available in RAxML. An excellent perl script, proteinModelSelection.pl, for identifying the best AA substitution model for tree generation was created by the RAxML author Alexis Stamatakis and can be found at the RAxML site.

The proteinModelSelect.pl script generates a lot of output files,so, to manage these create a new directory:

```
$ mkdir ProteinModel
```

Will need to move a copy of the alignment file into this directory in order to run proteinModelSelection.pl.:

```
$ cp rpsD_AA_aln.phymlAln ./ProteinModel/
```

Now, move into the ProteinModel directory.

```
$ cd ProteinModel
```

Run proteinModelSelection.pl and redirect the standard output, which is a printout of the best substitution model, to the file BestModel.txt.

```
$ proteinModelSelection.pl rpsD_AA_aln.phymlAln > BestModel.txt
```

When proteinModelSelection.pl is finished running, check for the best identified substitution model.

```
$ cat BestModel.txt
```

Which should print out `Best Model:  LG`

Also, check to see if a file `rpsD_AA_aln.phymlAln.reduced` has been generated. If there are duplicate sequences, this .reduced file is generated, and it is best to construct a tree on this alignment file. You can find what sequences were duplicates and which were removed from the alignment by checking the RAxML_info files.

```
$ more RAxML_info.LG_rpsD_AA_aln.phymlAln_EVAL
```

Move a copy of the .reduced file back to the AA directory, which should be one level above the current directory.

```
$ cp rpsD_AA_aln.phymlAln.reduced ../
```

Now, move back to the AA directory.

```
$ cd ../
```

4.3.4 1.4: Construct an amino acid reference phylogenetic tree

Build a tree using the generated alignment files and the best amino acid subtiution model identified. You can find more information about constructing trees with RAxML at the RAxML software page.

The PHANS-C Pipeline data analysis was developed using RAxML version 8.2.8. On your system, the RAxML exectuable may be named differently. For example, on our local system, ours is called *raxmlHPC-PTHREADS-AVX*. On the AWS AMI, version 8.2.8 can be accessed via the AMI executable *raxml*; however, a newer version of RAxML, version 8.2.11, is also available on the AMI and has been tested for compatibility with the PHANS-C Pipeline. RAxML version 8.2.11 can be accessed through the AMI executable *raxml_8.2.11* or via *raxml*.

```
$ raxml -m PROTCATLG -s rpsD_AA_aln.phymlAln.reduced -# 1 -T 8 -p 610 -n rpsD_AA
```

Some of the options used to construct this tree are as follows:

-m STR	Designates the model used to build the tree. Here, we are constructing an amino acid tree PROT using a gamma model of rate heterogeneity GAMMA using the best amino acid substitution model printed by proteinModelSelection.pl which is LG… PROTGAMMALG
-s FILE	Specify the input alignment file.
-T INT	Specify the number of threads. Do not use more than your computer has available.
-p INT	Specify a random seed for starting parsimony inferences.
-n STR	Specify a title for the output file names.

-# INT Specify the number of distinct starting trees.

For expediency in running this tutorial, we are using the CAT model and only using 1 starting tree.However, we would strongly encourage more stringent settings for real data, and recommend using the GAMMA model of rate heterogeneity *-m PROTGAMMALG* and at least 10 starting trees *-# 10*.

See the RAxML documentation for further explanation of options.

4.4 2: Parse and Format reads from Metagenome

4.4.1 2.1: Construct a BLAST database from the metagenomic sequence reads

You will need BLAST+ in order to build a blast database. You can find information on obtaining BLAST Get PHANS-C. The BLAST database will need to be hash indexed in order to pull metagenomic reads from the database for placement.

First, move into the blast directory:

```
$ cd ../blastDB/artificialMG/
```

An artificial metagenome, `artificialMG.fasta`, is located in this directory. Now, use this file to generate a blast database.

```
$ makeblastdb -parse_seqids -hash_index -dbtype nucl -in artificialMG.fasta -out␣
↪artificialMG
```

In order to access this blast database from any directory, we will need to add it to the blast path. If you have it installed locally, use your own custom location. If you are using the AWS image, to add this directory to the blast path for this session of the shell, enter the following in the command line:

```
$ export BLASTDB=$HOME/Tutorial/blastDB/artificialMG
```

In order to add this directory to the blast path for every session, update the shell .profile with the following:

```
$ echo 'export BLASTDB=$HOME/Tutorial/blastDB/artificialMG/' >> ~/.profile
```

4.4.2 2.2: Search the BLAST database for potential reads of interest

Use BLAST search to pull out reads from the metagenome which show similarity to your target gene of interest. In this case, parse out reads that show similarity to a set of *rpsD* sequences we are interested in. We will use a set of full length amino acid sequences as queries from the file `~/Tutorial/AA/rpsD_AA_seeds_for_blast.fasta`. NOTE: It is important to use full length gene sequences, as the trimming script used later will trim the metagenomic reads to be within the length or your starting query sequences.

Move to the directory `AA`

```
$ cd ../../AA/
```

Now use this amino acid sequence seed file to search a 6-frame translation of the metagenomic reads.

```
$ tblastn -db_gencode 11 -db ../blastDB/artificialMG/artificialMG -outfmt 5 -num_
↪threads 2 -max_target_seqs 1000000 -evalue 1 -query rpsD_AA_seeds_for_blast.fasta -
↪out rpsD_artificialMG.tblastn.xml
```

4.4.3 2.3: Parse reads from the BLAST results

Take the outfile from this BLAST search, `rpsD_artificialMG.tblastn.xml`, and parse out the desired information:

```
$ phan-parseBlastXML.py -f rpsD_artificialMG.tblastn.xml -n 0 -e ../blastDB/
↪artificialMG/artificialMG -s -l -o rpsD_artificialMG_blastResults -c 2
```

phan-parseBlastXML.py command line options:

-h, --help	Show this help message and exit.
-f FILE	Specify the blast XML format FILE that you wish to use as input.
-o FILE	Specify the outfile that you wish to use. If nothing is specified, the name and location of the infile will be used, with a "best" as the suffix.
-n INT	Specify the number of hits you want from each blast record. A 0 means return all results. Default is 1.
-s, --sort	Sort by evalue.
-e BLASTDB	Specify db to extract hit read sequences from.
-c INT	Specify number of cpu cores to use. Default=2.

phan-parseBlastXML.py output files:

`rpsD_artificialMG_blastResults.hits` - A file of all the blast hits: contains the blast information as well as the sequence of the reads. Blast results partitioned by query sequence. **This file to be passed as input to next pipeline step**

```
Each hit has a fasta title in the following tab delimited format:
Query   Hit Accession   Description Score               Evalue  Hit Number
↪           Query Length    Hit ID  Hit Length              Align Length
↪Identities             Positives               Gaps    Query From
↪Query To               Hit From                Hit To  Frame Query Frame
↪Hit



#####
#Hits below from query Acaryochloris_marina_MBIC11017_66
#####

>SMPLB_Deinococcus_geothermalis_DSM_11300_NCBI_taxonomyId_319795_37835_2
↪           SMPLB_Deinococcus_geothermalis_DSM_11300_NCBI_taxonomyId_319795_
↪37835_2 No definition line        130.0    3.53312e-10     1       202
↪SMPLB_Deinococcus_geothermalis_DSM_11300_NCBI_taxonomyId_319795_37835_2
↪142     42      25      30      0       89      130     1     126      0
↪            -2
TCGGTAGCTGGGAATATCCACGCGCTTGCCGTTGACCAGGATGTGACCGTGACCCACAAACTGCCGGGCCTGACGGCGGGTCGAGGCAAACCCCAT
>SMPLA_Deinococcus_geothermalis_DSM_11300_NCBI_taxonomyId_319795_37835_2
↪           SMPLA_Deinococcus_geothermalis_DSM_11300_NCBI_taxonomyId_319795_
↪37835_2 No definition line        130.0    3.53312e-10     2       202
↪SMPLA_Deinococcus_geothermalis_DSM_11300_NCBI_taxonomyId_319795_37835_2
↪142     42      25      30      0       89      130     1     126      0
↪            -2
...
```

`rpsD_artificialMG_blastResults` - A tab delimited file of all the unique blast hits with associated blast result information.:

```
Query   Hit Accession   Description Score               Evalue  Hit Number
↪           Query Length    Hit ID  Hit L
ength   Align Length    Identities              Positives               Gaps
↪   Query From              Query To                Hit F
rom             Hit To  Frame Query Frame Hit
Acaryochloris_marina_MBIC11017_66               SMPLB_Deinococcus_
↪geothermalis_DSM_11300_NCBI_taxonomyId_3197
```

(continues on next page)

(continued from previous page)

```
95_37835_2              SMPLB_Deinococcus_geothermalis_DSM_11300_NCBI_
↪taxonomyId_319795_37835_2 No definition
 line   130.0   3.53312e-10             1               202
↪SMPLB_Deinococcus_geothermalis_DSM_11300_NCBI_taxonom
yId_319795_37835_2              142             42              25
↪   30              0               89              130             1
↪       126             0       -2
Acaryochloris_marina_MBIC11017_66               SMPLA_Deinococcus_
↪geothermalis_DSM_11300_NCBI_taxonomyId_3197
95_37835_2              SMPLA_Deinococcus_geothermalis_DSM_11300_NCBI_
↪taxonomyId_319795_37835_2 No definition
 line   130.0   3.53312e-10             2               202
↪SMPLA_Deinococcus_geothermalis_DSM_11300_NCBI_taxonom
yId_319795_37835_2              142             42              25
↪   30              0               89              130             1
↪       126             0       -2
...
```

`rpsD_artificialMG_blastResults.hits_all.fasta` - A file that just contains all of the read sequences for the blast hits.:

```
>lcl|SMPLB_Deinococcus_geothermalis_DSM_11300_NCBI_taxonomyId_319795_37835_2
TCGGTAGCTGGGAATATCCACGCGCTTGCCGTTGACCAGGATGTGACCGTGACCCACAAACTGCCGGGCCTGACGGCGGG
TCGAGGCAAACCCCATGCGGAACACGACATTGTCGAGGCGGCGCTCCAGCAGCTGCAGGAAC
>lcl|SMPLB_Candidatus_Pelagibacter_ubique_HTCC1062_NCBI_taxonomyId_335992_
↪58376_2
TTTTTACCGTTAACTCTTACGTGACCATGGTTAATAAGTTGTCTCGCTGAAAATACTGTTGTTGAAAATCTTGCTCTATA
AACAACAGAATCTAATCTTCTTTCAAGCAAACCAATTAAATTTTCACCCGTATCCCCTTTTAACAT
...
```

4.4.4 2.4: Trim the metagenomic reads:

Pass the `.hits` output file from the previous step to `phan-trimFormatReads.py` in order to format and clean the reads for placement.:

```
$ phan-trimFormatReads.py -g rpsD -f rpsD_artificialMG_blastResults.hits -m 100 -e 1e-
↪5
```

phan-trimFormatReads.py command line options:

--version	Show program's version number and exit.
-h, --help	Show this help message and exit.
-g STR	Specify the gene name which will be used to label output files.
-f FILE	Specify the input file that is parsed from blast output... *.hits file.
-m INT	Specify the minimum length of reads that will be passed on to rest of analysis, if no min specified all reads will be passed.
-e STR	(OPTIONAL) Evalue cutoff all reads must be =< this evalue to continue, if no max specified all reads will be considered. Enter in scientific notation (1e-5).
-s INT	(OPTIONAL) All reads must be >= this score to continue, if no min specified all reads will be considered.

Functions of phan-trimFormatReads.py:

- Trims the reads so that they are in the proper reading frame which corresponds with the BLAST hit.
- Trims the reads to remove any overhang that might exist - ie. if the read captures the beginning of a gene, but also sequence upstream, the upstream sequence will be removed to improve alignment quality.
- Sets an e-value or score threshold for reads - must meet certain similarity scores in order to continue for placement analysis.
- Sets a length requirement - reads, after trimming, must be of a specific length (designated in nucleotide bases) in order to continue for placement analysis.
- Removes any reads with ambiguous nucleotide bases - creates a file with all the reads which ambiguous nucleotides.
- Removes stop codons in the amino acid translation (the stop codons do not work with downstream placement software). List of reads with stop codons produced.

phan-trimFormatReads.py output files:

`rpsD_formatted_query_seqs_NT.fasta` - A fasta file of the trimmed read sequences. Trimmed according to the blast hit information.:

```
>SMPLA_Candidatus_Pelagibacter_ubique_HTCC1062_NCBI_taxonomyId_335992_58376_1
AAAGTTAACATTGGAAGTTACGTTGTTAAAGAAGAAGATATAATTGAAATTAGAGATAAATCTAAACAGTTGGCTATAATTGATATCGCTCTAGCT
AGAAAGAGAAACACCTGAATACATTAATTTAGATGAAAAAAAT
>SMPLA_Candidatus_Pelagibacter_ubique_HTCC1062_NCBI_taxonomyId_335992_58376_2
ATGTTAAAAGGGGATACGGGTGAAAATTTAATTGGTTTGCTTGAAAGAAGATTAGATTCTGTTGTTTATAGAGCAAGATTTTCAACAACAGTATTT
GAGACAACTTATTAACCATGGTCACGTAAGAGTTAACGGTAAA
>SMPLB_Candidatus_Pelagibacter_ubique_HTCC1062_NCBI_taxonomyId_335992_79864_1
AAACATAAAATAGATAGAAGGTTAAAAATTAACCTATGGGGTAGACCTAAAAGTCCTTTTAATAAAAGGGATTATGGTCCAGGACAACACGGACAA
AAAAGGAAAACCTTCTGATTACGGAATTCAGCTACAAGCTAAACAAAAATTAAAA
...
```

`rpsD_AA_translation_stops_removed.fasta` - A fasta file of the amino acid translation of the trimmed sequence reads according to the blast hit information. Any stop codons (*) have been removed as stop codons interfere with downstream programs. A file called `rpsD_AA_translations_with_Stops.fasta` is also produced which contains any reads with stop codons.:

```
>SMPLA_Candidatus_Pelagibacter_ubique_HTCC1062_NCBI_taxonomyId_335992_58376_1
KVNIGSYVVKEEDIIEIRDKSKQLAIIDIALASKERETPEYINLDEKN
>SMPLA_Candidatus_Pelagibacter_ubique_HTCC1062_NCBI_taxonomyId_335992_58376_2
MLKGDTGENLIGLLERRLDSVVYRARFSTTVFSARQLINHGHVRVNGK
>SMPLB_Candidatus_Pelagibacter_ubique_HTCC1062_NCBI_taxonomyId_335992_79864_1
KHKIDRRLKINLWGRPKSPFNKRDYGPGQHGQGRKGKPSDYGIQLQAKQKLK
...
```

`rpsD_info_DICTIONARY_FILE_for_back_translate.csv` - A data file in .csv format which contains key sequence information.:

```
Accession, e-value, abs_frame, raw_read, fixed_read, AA_with_stop, AA_
↪removed_stop
>SMPLA_Candidatus_Pelagibacter_ubique_HTCC1062_NCBI_taxonomyId_335992_58376_
↪1, 1.15292e-24, 3,␣
↪ATTTTTTTCATCTAAATTAATGTATTCAGGTGTTTCTCTTTCTTTACTAGCTAGAGCGATATCAATTATAGCCAACTGTTTAGATTTATCTCTA
↪␣
↪AAAGTTAACATTGGAAGTTACGTTGTTAAAGAAGAAGATATAATTGAAATTAGAGATAAATCTAAACAGTTGGCTATAATTGATATCGCTCTAG
↪ 233.0, KVNIGSYVVKEEDIIEIRDKSKQLAIIDIALASKERETPEYINLDEKN,␣
↪KVNIGSYVVKEEDIIEIRDKSKQLAIIDIALASKERETPEYINLDEKN
...
```

`rpsD_unique_reads_with_ambiguous_nucleotides.fasta` - If there are reads which have ambiguous nucleotides (e.g. base = 'N'), then those reads are removed and listed here.

`rpsD_query_reads_processing_info.log` - A helpful basic stats file that should be reviewed before proceeding to next step of the pipeline. If there are significant reads which are removed due to ambiguous nucleotides or a high abundance of reads with stop codons, then the user should proceed with caution.:

```
The original number of blast reads prior to deduplication = 3110

The original number of unique reads recruited by blast = 106

e-value cutoff of 1e-05 used; all reads reported are <= this e-value␣
↪threshold.
Number of reads which did not make this e-value threshold and were␣
↪discarded: 23

The number of unique reads recruited from blast that also met thresholds␣
↪specified = 83

The number of unique reads containing ambiguous nucleotides = 0
The percent of unique reads with ambiguous nucleotides that were removed␣
↪from analysis = 0.00%

The number of unique reads that were less than 100 bases & removed from␣
↪analysis = 7
The percent of reads that were too short & removed from analysis = 8.43%

The number of clean unique reads without ambiguous nucleotides = 76

The number of reads with at least one stop codon = 0
Percent of unique reads with stop codons = 0.00%
```

4.5 3: Align unknown reads to Reference Alignment

4.5.1 3.1: Use the program PaPaRa to align reads

Combine the formatted unknown reads with the reference phylogeny for identification. Take the alignmment that was used to construct the phylogentic reference tree and align all the unknown clean reads (amino acid space) to the reference alignment using the software package PaPaRa.

```
$ papara -t RAxML_bestTree.rpsD_AA -s rpsD_AA_aln.phymlAln.reduced -q rpsD_AA_
↪translation_stops_removed.fasta -n rpsD_AA_artificialMG -j 1 -a -f
```

PaPaRa command line options used:

-t FILE	Specify the reference phylogenetic tree.
-s FILE	Specify the alignment used to build the reference tree.
-q FILE	Specify the file with the formatted environmental sequence reads.
-n STR	Specify label for output files.
-j INT	Specify number of threads (do not specify more threads than available on your machine).
-a	Using amino acid sequences.

PaPaRa output files:

`papara_alignment.rpsD_AA_artificialMG` - The alignment file in phymlAln format which contains the recruited reads aligned to the reference alignment.

```
140 228
Bifidobacterium_angulatum_DSM_20098_21
↪                              --MTNVQRSRRQVRLSRALGIAL----------TPKAQRIFEKRPYAPGEHG--
↪RTRRRTESDYAVRLREKQRLRAQY-
↪GLSEKQLRAVYEKGTKTSGQTGNAMLQDLEVRLDNLVLRAGFARTTAQARQFVVHRHILVDGNIVNRPSYRVKPGQTIQVKAKSQTMEPFQAAAEGVHL
↪-V-PAQVNIQYVVEFYAR--
Pirellula_sp_1_61
↪                                                    MKVTMARYTGPKARINRRLGTMLYET-----
↪AGAARA---MDRRPQPPGMHT----RGRRPSNYGAALMEKQKIKHYY-
↪GLGERQLRRYFENVGRKSGNTGELLLLMCERRLDNVVRRVGFTKTRPQARQGITHGHFRVNGVKVTKPGYMLRAGDLIEVRGRENLKNLYRGVIANSPL
↪---WVSFDSETLRATVLSLPGAVD--I-SLPVDANSVVEFLSR--
Planctomyces_brasiliensis_DSM_5305_53
↪                              ----MGRYTGPKARINRRLGSPVFES-----AGALRA---SEKRDSPPGMH---
↪QRRK-KPTIYGAALTEKQKIKYYY-
↪GFRERQLRKYFNEARRLKGNTGQNLLVLCERRLDNVVRRAGFAQTRPQARQAIVHSHFQLNGRTVNKPSIQVRAGDVITVRNRPNLHAIYKEQVGRADT
↪---FVSVDDSALKIVVSAIPTFAD--V-SLPVDVNQVVAFLSR--
...
SMPLB_Candidatus_Pelagibacter_ubique_HTCC1062_NCBI_taxonomyId_335992_32557_2  --------
↪------------------------------------------------------------------------------------------
↪-----------------------------------------------------------KEEDIIEIRDKSKQLAIIDIALA-
↪SKERETPE---YINLDEKNKKVTFVRTPKFDE----------------------
...
```

`papara_log.rpsD_AA_artificialMG` - Settings and statistics for the PaPaRa run. This information is also printed to StdOut upon running PaPaRa.

```
papara called as:
papara -t RAxML_bestTree.rpsD_AA -s rpsD_AA_aln.phymlAln.reduced -q rpsD_AA_
↪translation_stops_removed
.fasta -n rpsD_AA_artificialMG -j 16 -a -f
references container instantiated as: papara::references<pvec_pgap, sequence_
↪model::tag_aa>
edges: 125
scoring scheme: -3 -1 2 -3
papara_core version 2.4
start scoring, using 16 threads
thread 1: 0.786583 gncup/s
thread 3: 0.378382 gncup/s
thread 10: 0.200862 gncup/s
thread 6: 0.37806 gncup/s
thread 4: 0.371692 gncup/s
thread 9: 0.193493 gncup/s
thread 8: 0.378985 gncup/s
thread 12: 0.209433 gncup/s
thread 2: 0.477034 gncup/s
thread 13: 0.177332 gncup/s
thread 14: 0.282384 gncup/s
thread 0: 0.286568 gncup/s
thread 15: 0.283097 gncup/s
thread 11: 0.327415 gncup/s
thread 5: 0.594323 gncup/s
thread 7: 0.489285 gncup/s
scoring finished: 0.023149
```

(continues on next page)

(continued from previous page)

```
generating best scoring alignments
SUCCESS 0.147885
```

4.5.2 3.2: Join reads to increase phylogenetic signal

Next generation sequencing technologies often use mate pairing or paired ends - geneating 2 reads per strand of genomic DNA. In order to enhance the phylogenetic signal of the unknown reads for placement, and reduce errors of duplicate counting, the script spaceJoin will combine the alignments of individual read pairs into one read for placement. Since the exact length between the read pairs is unknown, an assumption is made that the alignment of the individual reads is accurrate. Read pairs should be designated as _1 and _2 in fasta file used for building the BLAST database, and the optional suffix _0 can represent read pairs that have already been joined.

```
$ phan-spaceJoin.py -p papara_alignment.rpsD_AA_artificialMG -r 64 -o rpsD_AA_cut40 -
↪l 40
```

phan-spaceJoin.py command line options:

--version	Show program's version number and exit.
-h, --help	Show this help message and exit.
-p FILE	Specify the output file from PaPaRa that has the reference aln and metagenome reads aln combined ("papara_alignment.*").
-r INT	Specify the number of sequences in the reference alignment (# of sequences in reference tree).
-o FILE	Specify the your desired output alignmemt file name.
-l INT	Specify the minimum length of each read for placement; default set to 45 bases/residues.

phan-spaceJoin.py output files:

`Log_spaceJoin_rpsD_AA_cut40_joinedReadsForPlacement.txt` - A log file that outputs statistics about the joining of read pairs. Reads that are joined are relabeles as "_3".

```
The following of the input alignment are the following:

Pre-Joind BOTH READS: 0
BOTH READS: 29
ONLY READ 1: 12
ONLY READ 2: 6


********************************************************************************
The following are the set of reads in the output file that are longer than 40
LONG Pre-Joined BOTH READS: 0
LONG BOTH READS: 29
LONG ONLY READ 1: 12
LONG ONLY READ 2: 4

Perecent of reads from total dataset that are longer than 40: 97.37%

Percent of long reads with pairs: 78.38%
```

(continues on next page)

(continued from previous page)

```
*******************************************************************************
Reads that are written to output file

Reads with mate pairs joined:
SMPLA_Vibrio_sp_MED222_NCBI_taxonomyId_314290_82469_3, SMPLB_Vibrio_sp_
↪MED222_NCBI_taxonomyId_314290_82469_3, SMPLA_Deinococcus_geothermalis_DSM_
↪11300_NCBI_taxonomyId_319795_37835_3, SMPLA_Candidatus_Pelagibacter_ubique_
↪HTCC1062_NCBI_taxonomyId_335992_19173_3,
...
```

`rpsD_AA_cut40_joinedReadsForPlacement.phymlAln` - A new alignment file with the joined reads.

```
109 228
Bifidobacterium_angulatum_DSM_20098_21
↪                                        --MTNVQRSRRQVRLSRALGIAL----------
↪TPKAQRIFEKRPYAPGEHG--RTRRRTESDYAVRLREKQRLRAQY-
↪GLSEKQLRAVYEKGTKTSGQTGNAMLQDLEVRLDNLVLRAGFARTTAQARQFVVHRHILVDGNIVNRPSYRVKPGQTIQVKAKSQTMEPFQAAA
↪-V-PAQVNIQYVVEFYAR--
Pirellula_sp_1_61
↪
↪MKVTMARYTGPKARINRRLGTMLYET-----AGAARA---MDRRPQPPGMHT----
↪RGRRPSNYGAALMEKQKIKHYY-
↪GLGERQLRRYFENVGRKSGNTGELLLLMCERRLDNVVRRVGFTKTRPQARQGITHGHFRVNGVKVTKPGYMLRAGDLIEVRGRENLKNLYRGVI
↪---WVSFDSETLRATVLSLPGAVD--I-SLPVDANSVVEFLSR--
...
SMPLA_Escherichia_coli_W3110_NCBI_taxonomyId_316407_83762_3
↪                       ------------------------------------------------------
↪-------------------------------
↪FRNYYKEAARLKGNTGENLLALLEGRLDNVVYRMGFGATRAEARQLVSHKAIMVNGRVVNIASYQVSPNDVVSIREKAKKQSRVKAALELAEQR
↪-------------------------------------------
SMPLB_Candidatus_Pelagibacter_ubique_HTCC1062_NCBI_taxonomyId_335992_8052_3
↪     ----MTKRISAKHKIDRRLKINL--------WGRPKS--PFNKRDYGPGQHG--
↪QGRKGKPSDYGIQLQAKQKLKGYYGNINERQFRNIYKKAT-----------------------------------
↪-------------------------------------------------------------------------
↪------------------------
...
```

4.6 4: Phylogenetic placement of unknown reads

4.6.1 4.1: Use the Evolutionary Placement Algorithm (EPA) to assign unknown reads within tree

The formatted unknown metagenomic reads are combined with the reference phylogeny using RAxML EPA.

```
$ raxml -f v -m PROTCATLG -s rpsD_AA_cut40_joinedReadsForPlacement.phymlAln -r RAxML_
↪bestTree.rpsD_AA -n rpsD_AA_placements -T 8
```

RAxML-EPA command line options used:

-f v Use EPA to place environmental sequence reads onto a reference tree.

-m PROTGAMMALG Use the PROTGAMMALG model.

-s FILE Alignment file with joined environmental reads aligned to reference alignment.

-r FILE	Specify the original reference tree.
-n STR	Label for output files.
-T INT	Specify number of threads (do not specify more threads than available on your machine).

RAxML-EPA output files:

RAxML-EPA produces numerous files - details of which can be found in the RAxML manual.

`RAxML_classification.rpsD_AA_placements` - Read placement/classification results file.

```
SMPLB_Bacteroides_plebeius_DSM_17135_NCBI_taxonomyId_484018_1483_1 I9 1 0.
↪00000100000050002909
SMPLB_Shewanella_baltica_OS155_NCBI_taxonomyId_325240_2956_1 I96 1 0.
↪00000100000050002909
SMPLA_Bacteroides_plebeius_DSM_17135_NCBI_taxonomyId_484018_1483_1 I9 1 0.
↪00000100000050002909
SMPLA_Shewanella_baltica_OS155_NCBI_taxonomyId_325240_2956_1 I96 1 0.
↪00000100000050002909
...
```

`RAxML_classificationLikelihoodWeights.rpsD_AA_placements` - Read placement/classification results using likelihood weights.

`RAxML_entropy.rpsD_AA_placements` - Entropy results using likelihood weights.

`RAxML_originalLabelledTree.rpsD_AA_placements` - Original reference tree with labeled node IDs.

`RAxML_labelledTree.rpsD_AA_placements` - Reference tree with branch labels and read placements.

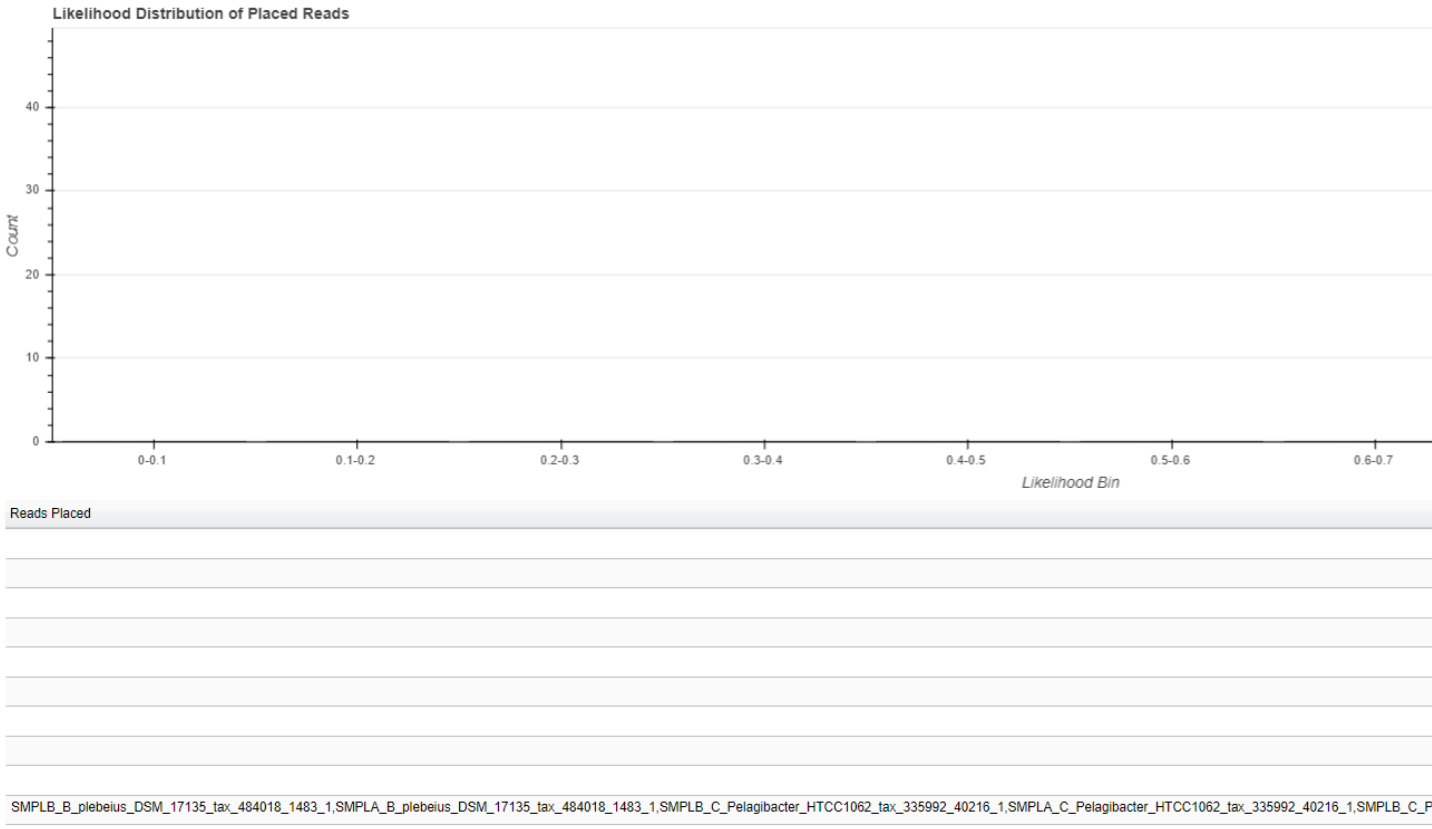
`RAxML_portableTree.rpsD_AA_placements.jplace` - Labelled reference tree with branch labels in portable pplacer/EPA format (without query sequences). This file can be used along with the package genesis for viewing/manipulating placement data. genesis is not included within the AWS AMI or the PHANS-C Pipeline.

4.6.2 4.2: Visualize distribution of likelihood scores (or branch-lengths)

All of the reads may not be placed with high quality scores. In order to evaluate read placement quality, you can view an interactive visualization of the distribution of either likelihood scores for the final placement of a read or the branch lengths. The script "phan-distributions.py" will generate a stand-alone interactive html file where you can inspect the distribution of reads (see figure below for example). Below the chart of the distribution is a table of the reads sorted in order of the score bins. By clicking on a bar in the chart, it will highlight the corresponding row in the table. You can then use keyboard shortcuts to copy the row of values (the reads with scores in that bin) and paste them into a text editor.

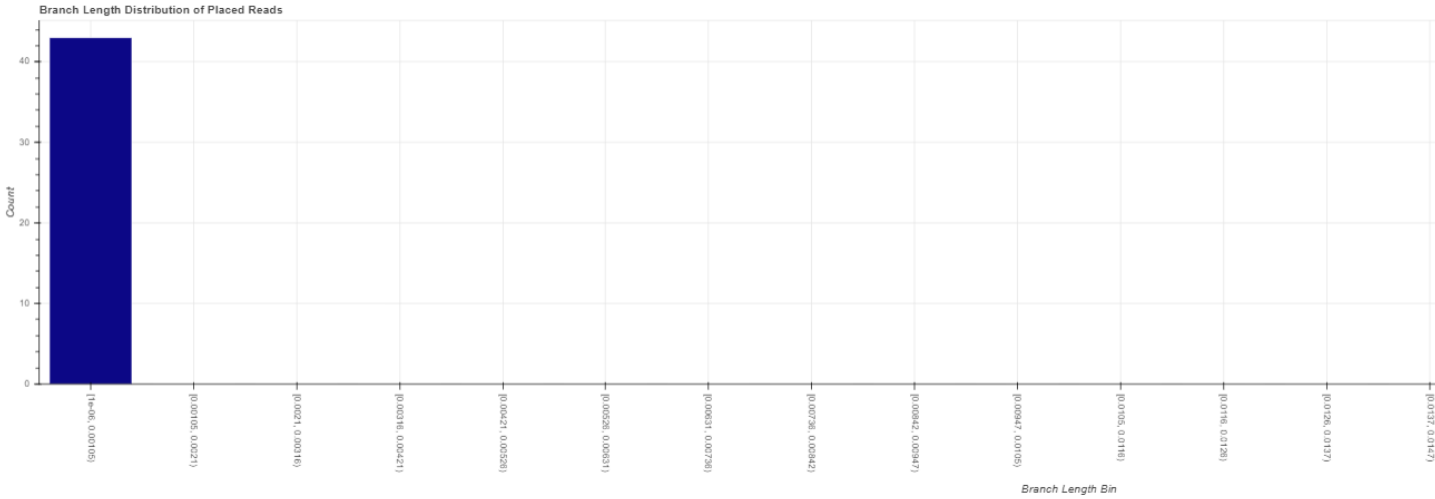
The following will generate a distribution of the likelihood scores in a stand-alone .html file:

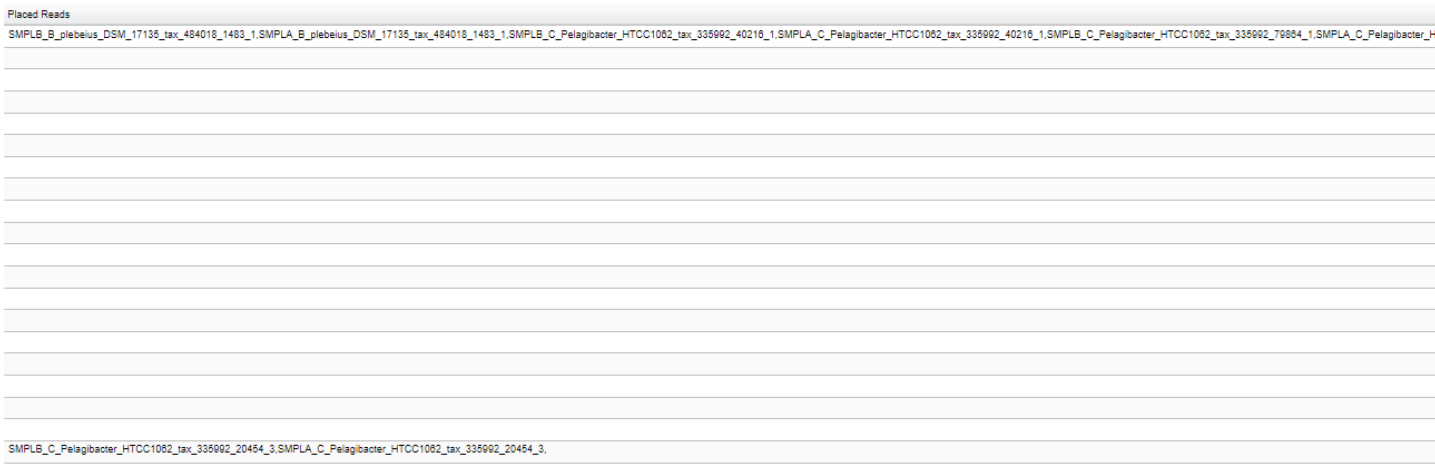
```
$ phan-distributions.py -i RAxML_classificationLikelihoodWeights.rpsD_AA_placements -
↪l -o rpsD_likelihoods
```



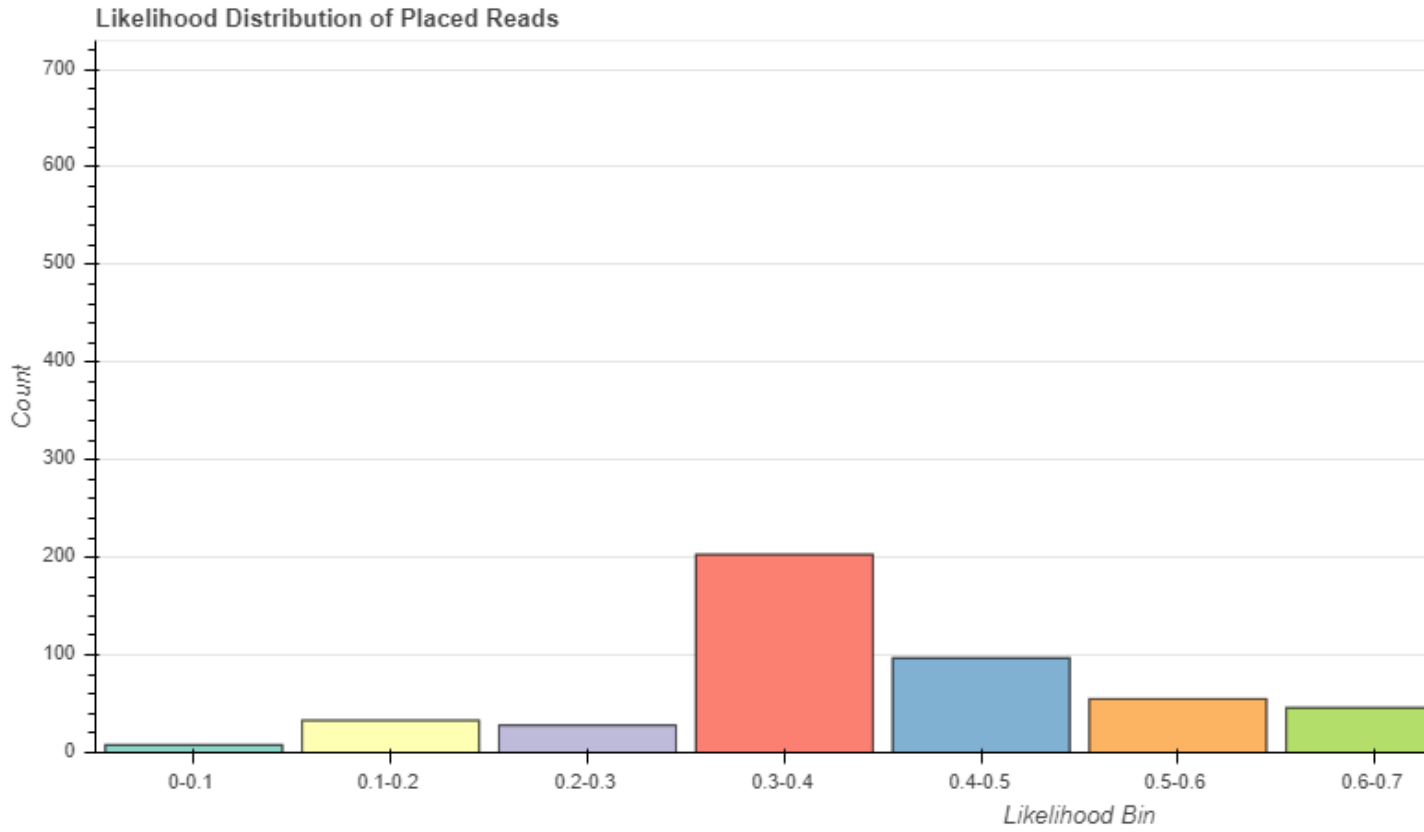
The following will generate a distribution of branch lengths in a stand-alone .html file:

```
$ phan-distributions.py -i RAxML_classification.rpsD_AA_placements -b -o rpsD_
↪branchLengths
```

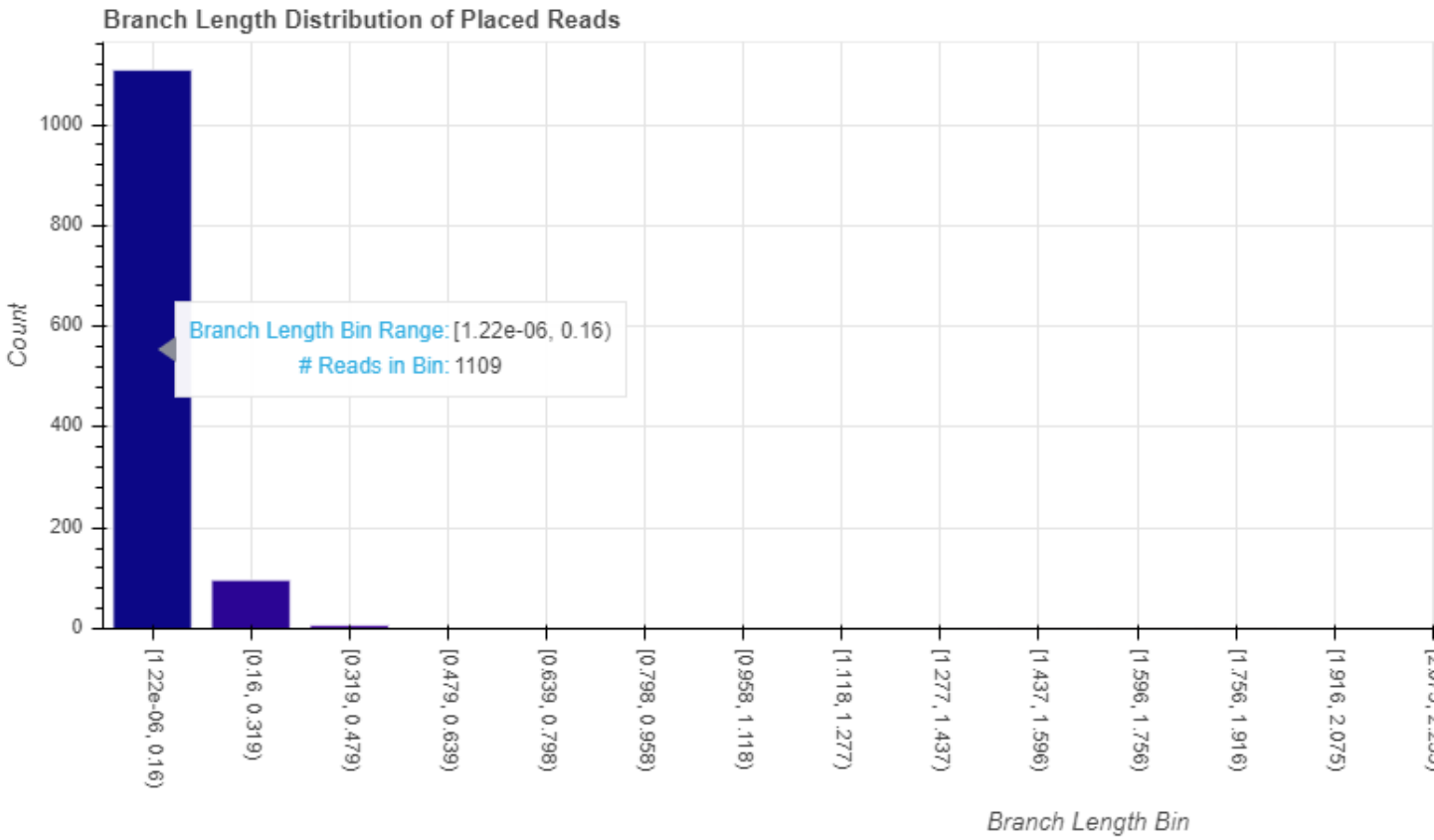




While these figures are extremely simple for this tutorial, they become more useful with complex data. For example, the following are the distributions for the more complicated dataset analyzed and described in the manuscript.



Reads Placed
SUP05_isolate_Vega_2549783_1,SUP05_isolate_Vega_2590939_1,Burkholderia_sp_383_224064_1,Nitrosococcus_oceani_ATCC_19707_1056567_1,Burk
SUP05_isolate_Vega_2586142_1,Prochlorococcus_marinus_str_MIT_9301_628085_1,Prochlorococcus_marinus_str_MIT_9301_629063_1,SUP05_isolate_
SUP05_isolate_Vega_2524183_1,SUP05_isolate_Vega_2544157_1,SUP05_isolate_Vega_2561573_1,SUP05_isolate_Vega_2557280_1,SUP05_isolate_Ve
Candidatus_Pelagibacter_ubique_HTCC1002_1631006_1,SUP05_isolate_Vega_2567443_1,Cand_Pelagibacter_ubique_HTCC1062_2122039_1,Candidatu
Prochlorococcus_MIT9311_2890236_1,SUP05_isolate_Vega_2636733_1,Prochlorococcus_MIT9311_2882882_1,Prochlorococcus_marinus_str_MIT_9313_
SUP05_isolate_Vega_2538691_1,SUP05_isolate_Vega_2514932_1,SUP05_isolate_Vega_2631509_1,SUP05_isolate_Vega_2655356_1,SUP05_isolate_Ve
Prochlorococcus_MIT9322_2951266_1,Prochlorococcus_MIT9322_2955994_1,SUP05_isolate_Vega_2557066_1,SUP05_isolate_Vega_2579108_1,SUP05_
Planktomarina_temperata_RCA23_Roseobacter_2670693_1,Prochlorococcus_PAC1_2974742_1,Prochlorococcus_EQPAC1_2766587_1,Prochlorococcus_
Prochlorococcus_PAC1_2967802_1,SUP05_isolate_Vega_2538124_2,Prochlorococcus_marinus_str_MIT_9313_779935_2,Prochlorococcus_marinus_str_M
Prochlorococcus_marinus_str_AS9601_69815_1,Prochlorococcus_marinus_str_AS9601_133907_1,Oenococcus_oeni_PSU-1_1356586_1,Candidatus_Pela



Placed Reads
Prochlorococcus_marinus_str_AS9601_69815_1,Prochlorococcus_marinus_str_AS9601_133907_1,Oenococcus_oeni_PSU-1_1356586_1,Candidatus_Pela
SUP05_isolate_Vega_2538691_1,SUP05_isolate_Vega_2567443_1,SUP05_isolate_Vega_2514932_1,SUP05_isolate_Vega_2524183_1,SUP05_isolate_Ve
SUP05_isolate_Vega_2544157_1,SUP05_isolate_Vega_2575141_1,SUP05_isolate_Vega_2634211_1,SUP05_isolate_Vega_2514595_2,SUP05_isolate_Ve

4.6.3 4.3: View the placements and tree with FigTree

The script "phan-figTreeFiles.py" will reformat the placement tree into a newick-like format that can be viewed with the phylogenetic tree viewer FigTree. A basic annotation file will also be generated that can be loaded into FigTree for additional tree visualizations (used in the color mapping of reads).

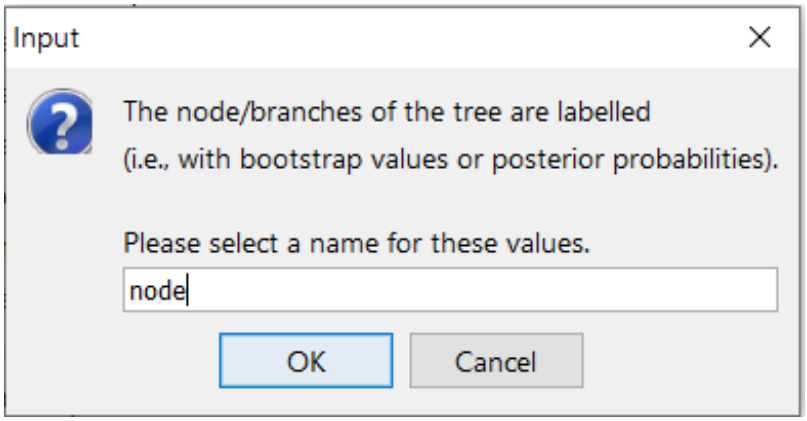
```
$ phan-figTreeFiles.py -i RAxML_labelledTree.rpsD_AA_placements -o rpsD
```

phan-figTreeFiles.py options:

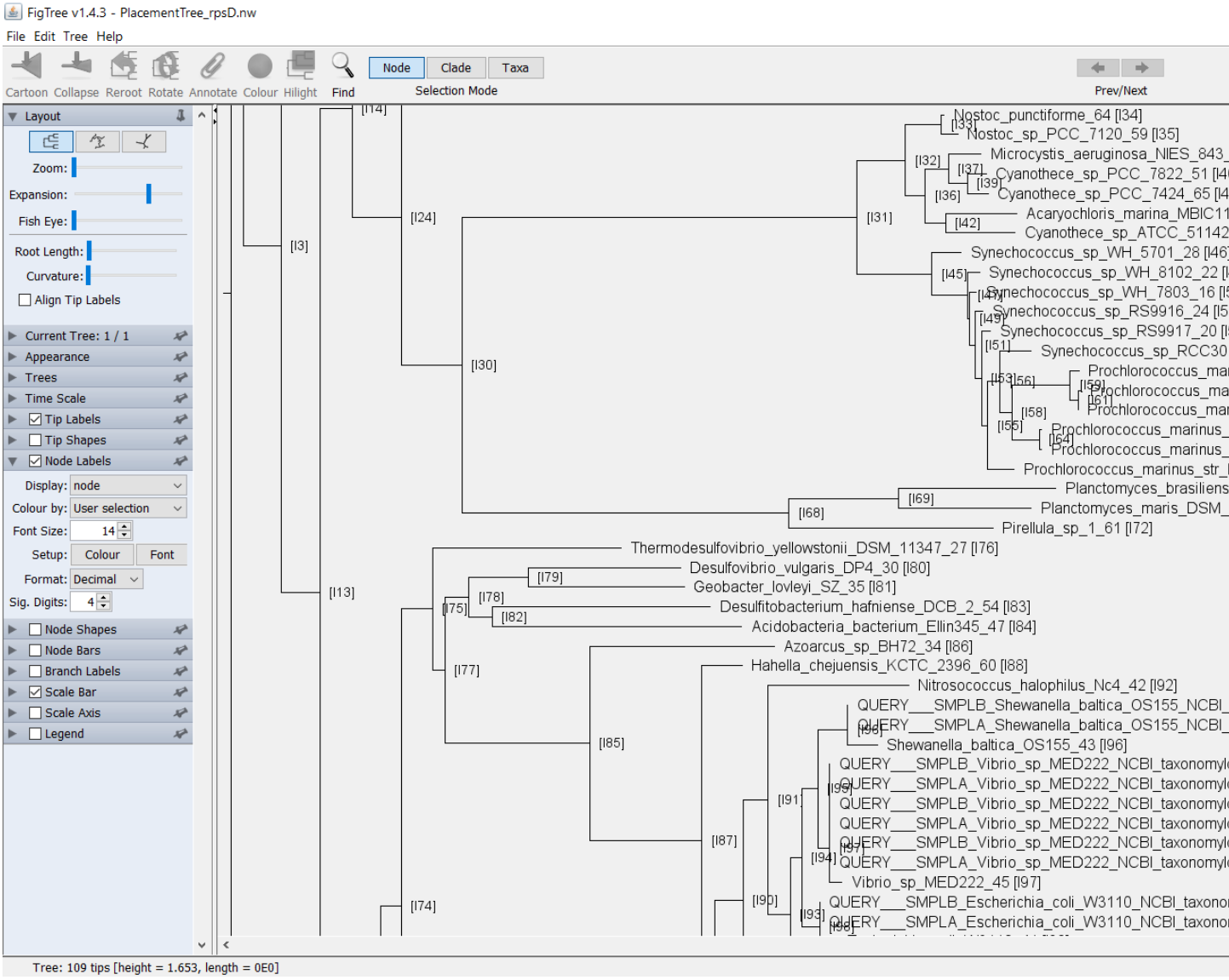
-i Specify the tree you wish to reformat. In the example, using the labelled tree output from RAxML-EPA, but the original tree output can also be used as input.

-o specify the label you would like to use in your output file names.

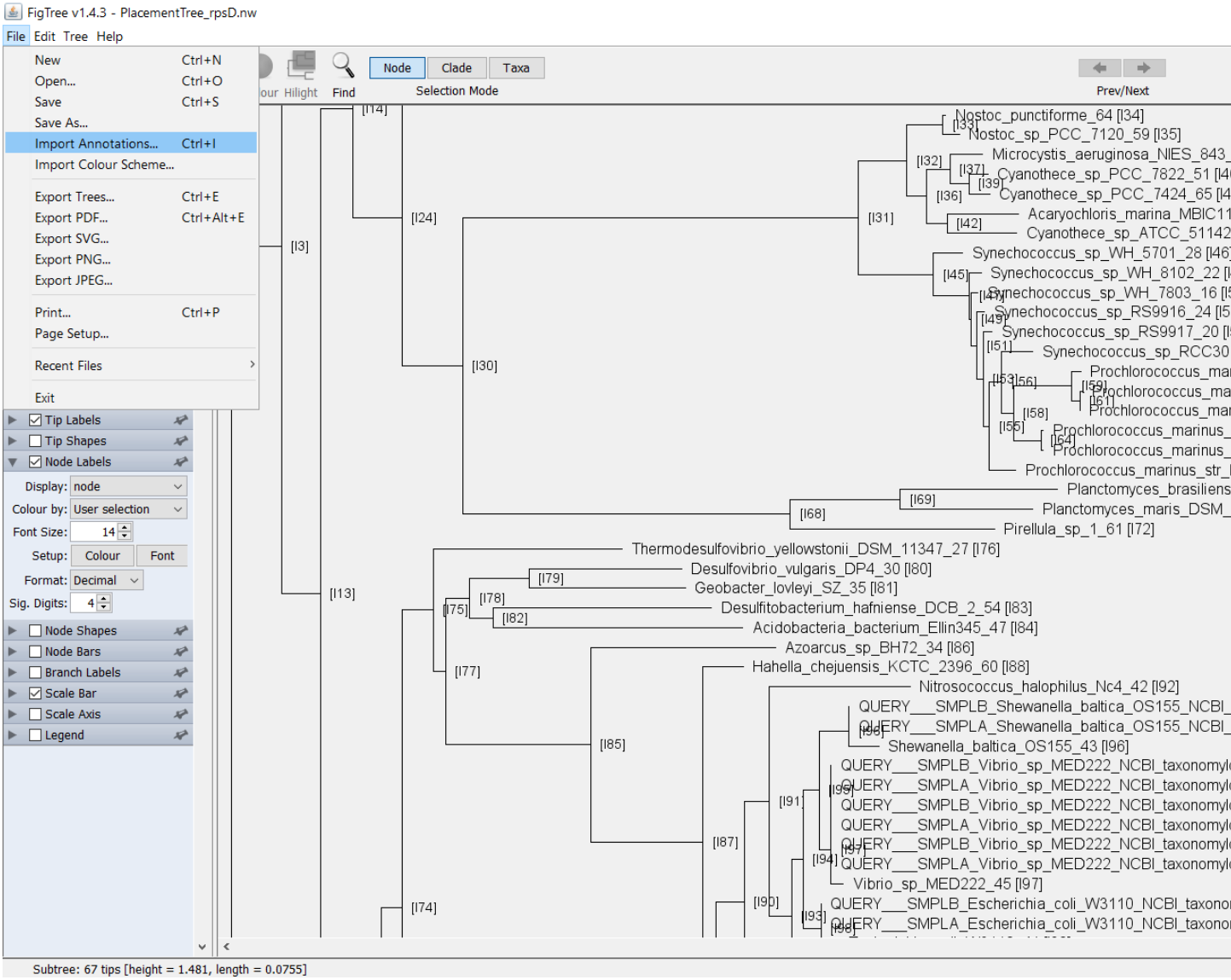
You can now open the output file in the FigTree GUI tree viewer. Open the file `PlacementTree_rpsD.nw` with FigTree. When the "Input" window pops up, change the field to "node".



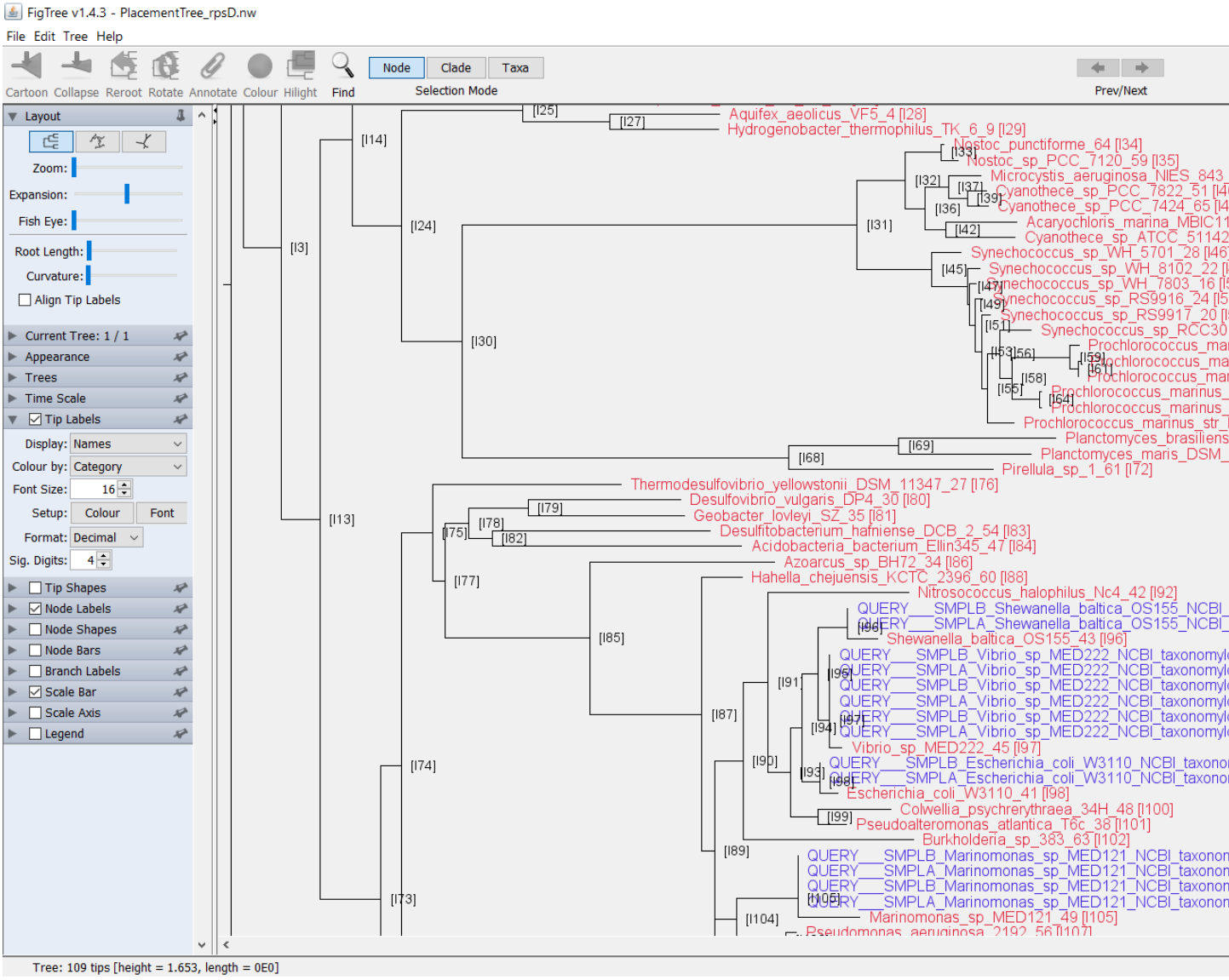
Now you can see the Tree in FigTree. Notice that the names of the terminal nodes now have the node number appended to the end of the label as [I#]. You can now add in internal node labels as well by checking the box for "Node Labels" in the left pane. If you select the arrow, the field will drop down with more details. Click on the drop-down selector next to "Display" and select the field "node". You should now be able to see all the internal nodes labeled throughout the tree.



You can add further customization to the tree by using the default annotations file that is produced by `phan-figTreeFiles.py` called `AnnotationsFile_PlacementTree_rpsD.tab`. You can manually add additional labels to this file in order to add further customizations. In order to load this annotations file, in FigTree go to File > Import_Annotations and load the file `AnnotationsFile_PlacementTree_rpsD.tab`.



In order to visualize the annotations, select the arrow next to “Tip Labels” in the left pane. From the drop down selector next to “Colour by:” select “Category”. In order to adjust the colors further, next to “Setup:” click on the box “Colour”.



This is a very short overview on working with annotations in FigTree. Please see the FigTree documentation and discussion threads for further information.

4.6.4 4.4: Create annotation key for read placements

Generate tree file dictionary:

The RAxML-EPA output file `RAxML_classification.rpsD_AA_placements` classifies the placement reads by labeled node ID. In order to make the classifications more readily interpretable, a dictionary file is created based upon the tree file `RAxML_originalLabelledTree.rpsD_AA_placements` which acts as a key to convert labeled node number into the reference sequence name for all extant nodes.

```
$ phan-taxaDictionary.py -O RAxML_originalLabelledTree.rpsD_AA_placements -N rpsD_
↪taxa_dictionary.txt
```

If the output is improperly named (see note about `-N` below) you may have to rename the output file.

```
$ mv rpsD_taxa_dictionary.txt_Taxa_Dictionary.txt rpsD_taxa_dictionary.txt
```

phan-taxaDictionary.py command line options:

--version	Show program's version number and exit.
-h, --help	Show this help message and exit.
-O FILE	Specify the original tree file output from RAxML-EPA that is the labeled tree by node of Ref taxa only.
-N STR	Specify the output file name for the taxa dictionary. Note that the -N flag may not be honored, if so you must hand rename the file.

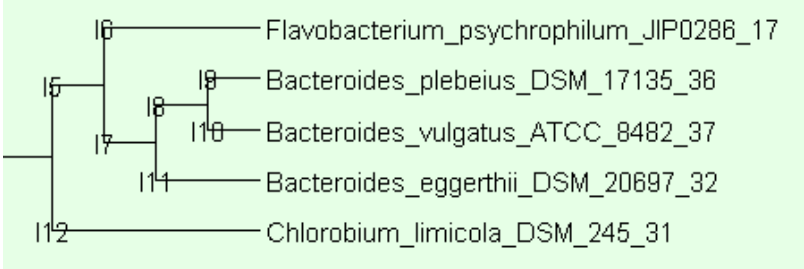
phan-taxaDictionary.py output file:

`rpsD_taxa_dictionary.txt` - The dictionary output file.

```
...
I6              Flavobacterium_psychrophilum_JIP0286_17
I9              Bacteroides_plebeius_DSM_17135_36
I10             Bacteroides_vulgatus_ATCC_8482_37
I8              I8
I11             Bacteroides_eggerthii_DSM_20697_32
I7              I7
I5              I5
I12             Chlorobium_limicola_DSM_245_31
...
```

Update tree file dictionary:

If we look at this tree, we can see that some of the internal nodes can also be relabeled from node numbers to more meaningful text labels. *Add in FigTree info Some tree viewing software may require you to add the '.tree' suffix to the end of the file name in order to open it. Below is a zoomed in section of the tree showing the portion of the reference tree containing *Bacteroides*.*



The internal nodes I8 and I7 are internal nodes containing *Bacteroides*. It is possible to manually update the dictionary file to reflect this, so that any placements made to these internal nodes will be labeled as *Bacteroides*.

Manually update `rpsD_taxa_dictionary.txt`:

```
...
I6              Flavobacterium_psychrophilum_JIP0286_17
I9              Bacteroides_plebeius_DSM_17135_36
I10             Bacteroides_vulgatus_ATCC_8482_37
I8              Bacteroides
I11             Bacteroides_eggerthii_DSM_20697_32
I7              Bacteroides
I5              I5
```

(continues on next page)

(continued from previous page)

```
I12             Chlorobium_limicola_DSM_245_31
...
```

4.6.5 4.5: Reannotate read placements with meaningful labels

Use the dictionary file, `rpsD_taxa_dictionary.txt` to relabel the nodes from the RAxML-EPA classification file.

```
$ phan-treeLabels.py -D rpsD_taxa_dictionary.txt -C RAxML_classification.rpsD_AA_
↪placements
```

phan-treeLabels.py command line options:

--version	Show program's version number and exit.
-h, --help	Show this help message and exit.
-D FILE	Specify the dictionary file that contains the node location '[I#]' with the associated reference taxa name.
-C FILE	Specify the output file from RAxML-EPA that is a list of the Query sequences assigned to their locations on the reference tree

phan-treeLabels.py output file:

`RAxML_classification.rpsD_AA_placements_placement_list.csv` - The reads placed by relabeled tree nodes in .csv format.

```
Tree_node, Read_ID
Bacteroides_plebeius_DSM_17135_36,SMPLB_Bacteroides_plebeius_DSM_17135_NCBI_
↪taxonomyId_484018_1483_1
Shewanella_baltica_OS155_43,SMPLB_Shewanella_baltica_OS155_NCBI_taxonomyId_325240_
↪2956_1
Bacteroides_plebeius_DSM_17135_36,SMPLA_Bacteroides_plebeius_DSM_17135_NCBI_
↪taxonomyId_484018_1483_1
...
```

4.6.6 4.6: Tabulate reads & sort by metagenome

Finally, you can group and count all reads placed to unique tree labeled nodes. If nodes were labeled with the same text string (e.g. internal nodes I7 & I8 relabeled as *Bacteroides*, then the count will be a combination of both of these labels).

If the original BLAST database contained sequences from multiple metagenomic samples, then this script can optionally parse out the placements by metagenome. In order to utilize this feature, the reads must all be labeled with unique metagenome sample ID pre-fixes (e.g. read *SMPLA_Bacteroides_plebeius_DSM_17135_NCBI_taxonomyId_484018_1483_1* is from metagenome **SMPLA**). A key file of the metagenomes must also be included which consists of a text file with each unique metagenomic sample ID listed per line. For the tutorial, one has been provided: `Sample_list_IDs.txt`

```
SMPLA
SMPLB
```

Since the tutorial consists of a database with two metagenomes, we will separate them out.

```
$ phan-sortPlacements.py -C RAxML_classification.rpsD_AA_placements_placement_list.
↪csv -L Sample_list_IDs.txt
```

phan-sortPlacements.py command line options:

--version	Show program's version number and exit.
-h, --help	Show this help message and exit.
-L FILE	Specify the metagenomic sample unique identifiers - one ID per line without special characters.
-C FILE	Specify the reads listed according to their node which has already been relabeled by phan-taxaDict.py.

phan-sortPlacements.py output files:

Two files are generated for every metagenomic sample. Here, we have files generate for SMPLA and SMPLB metagenomes.

`RAxML_classification.rpsD_AA_placements_placement_list_SMPLA_counts.csv` - A .csv file of the counts of reads by unique node label.

```
Tree_node, Total Reads
Escherichia_coli_W3110_41,1
Bacillus_selenitireducens_MLS10_29,2
Deinococcus_geothermalis_DSM_11300_33,3
Cand_Pelagibacter_ubique_HTCC1062_1,9
Bacteroides_plebeius_DSM_17135_36,1
Vibrio_sp_MED222_45,3
Shewanella_baltica_OS155_43,1
Marinomonas_sp_MED121_49,2
...
```

`RAxML_classification.rpsD_AA_placements_placement_list_SMPLA_readsPlaced.csv` - A .csv file of all the reads placed to each unique node label. The first entry on each line is the unique node label and all the following elements are the individual reads which placed to that node.

```
Tree_node, Reads_placed
Escherichia_coli_W3110_41,SMPLA_Escherichia_coli_W3110_NCBI_taxonomyId_316407_83762_3
Bacillus_selenitireducens_MLS10_29,SMPLA_Bacillus_selenitireducens_MLS10_NCBI_
↪taxonomyId_439292_28809_3,SMPLA_Bacillus_selenitireducens_MLS10_NCBI_taxonomyId_
↪439292_52168_3
Deinococcus_geothermalis_DSM_11300_33,SMPLA_Deinococcus_geothermalis_DSM_11300_NCBI_
↪taxonomyId_319795_37835_3,SMPLA_Deinococcus_geothermalis_DSM_11300_NCBI_taxonomyId_
↪319795_2 6433_3,SMPLA_Deinococcus_geothermalis_DSM_11300_NCBI_taxonomyId_319795_
↪93937_3
...
```

The related files, `RAxML_classification.rpsD_AA_placements_placement_list_SMPLB_counts.csv` and `RAxML_classification.rpsD_AA_placements_placement_list_SMPLB_readsPlaced.csv` are also generated for reads from metagenomic sample SMPLB.

4.7 5: Final Summary

The final result of running the PHANS-C Pipeline is a list of short sequencing reads that have been characterized according to full length references. If you ran the *run_tutorial.sh* script, then you will find a directory called */Results*

which has the highlights produced from this script. In order to generate these results manually (if you followed this tutorial manually, step-by-step), run the following to generate this same directory:

```
$ mkdir ../Results
$ mkdir ../Results/Quantitation
$ mkdir ../Results/EPA-Out
$ mkdir ../Results/Vis-Distributions
$ cp RAxML_labelledTree.rpsD_AA_placements RAxML_originalLabelledTree.rpsD_AA_
↪placements RAxML_classification.rpsD_AA_placements RAxML_
↪classificationLikelihoodWeights.rpsD_AA_placements ../Results/EPA-Out/
$ cp rpsD_branchLengths_distribution_branch-lengths.html rpsD_likelihoods_
↪distribution_likelihoodWeights.html ../Results/Vis-Distributions/
$ cp AnnotationsFile_PlacementTree_rpsD_figTree.tab PlacementTree_rpsD_figTree.nw ../
↪Results/
$ cp RAxML_classification.rpsD_AA_placements_placement_list.csv RAxML_classification.
↪rpsD_AA_placements_placement_list_SMPLA_counts.csv RAxML_classification.rpsD_AA_
↪placements_placement_list_SMPLA_readsPlaced.csv RAxML_classification.rpsD_AA_
↪placements_placement_list_SMPLB_counts.csv RAxML_classification.rpsD_AA_placements_
↪placement_list_SMPLB_readsPlaced.csv ../Results/Quantitation/
```

The overall results are broken up into 3 subdirectories. */Results/EPA-Out* are copies of the main output files from the EPA placement algorithm - to view the placement files in FigTree, you can use *$ phan-figTreeFiles.py* to generate a newick-like file that can be loaded by FigTree (See Section 4.3 for details). */Results/Vis-Distributions* are copies of the interactive visualizations of the read placement metrics. And */Results/Quantitation* lists copies of .csv files which list counts of reads placed to nodes on the reference tree.

CHAPTER 5

License

This material is based upon work supported by the National Science Foundation under Grant Numbers OCE-1138368 and OCE-1356779 as well as a NASA Graduate Research Fellowship and NASA Postdoctoral Research Fellowship to JKS. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation or the National Aeronautics and Space Administration.