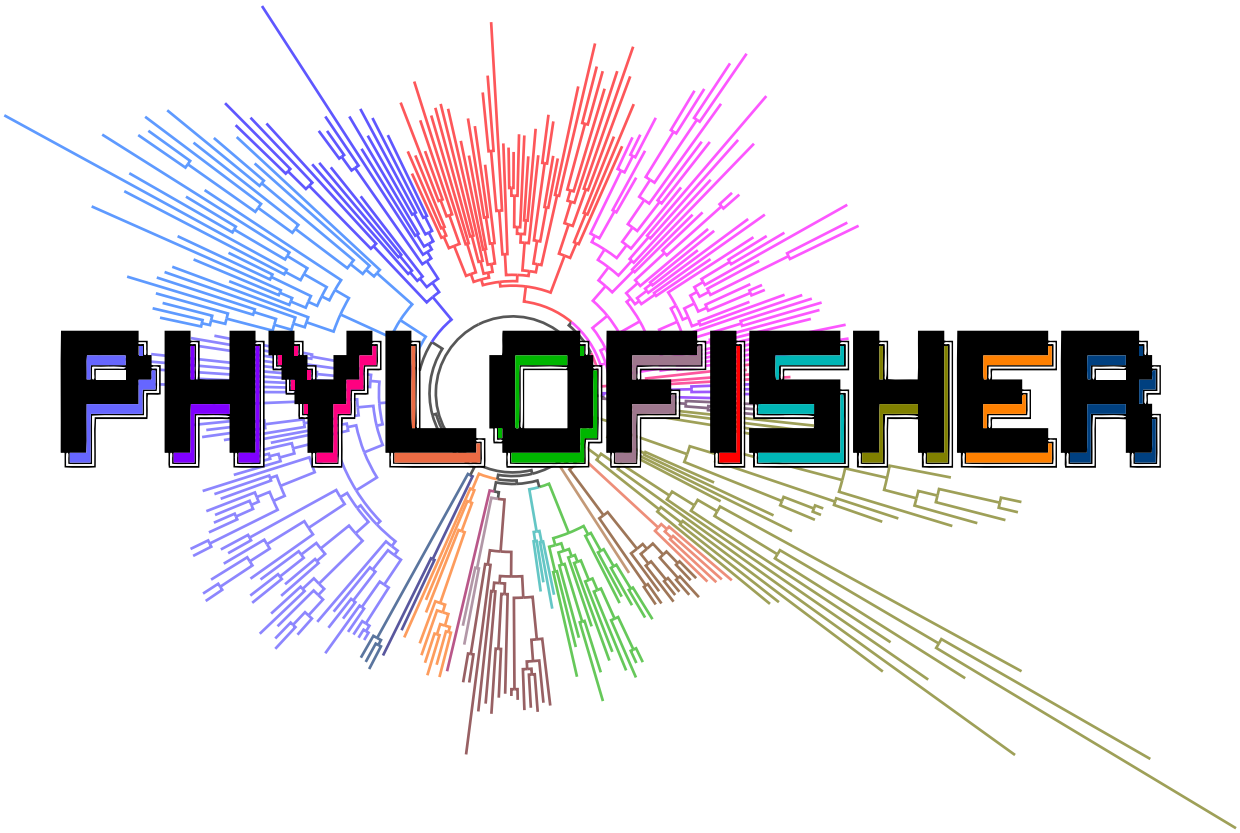

v.1.0.11 Manual

PhyloFisher: A phylogenomic package for resolving eukaryotic relationships

Alexander K. Tice [1,2,*], David Žihala [3,*], Tomáš Pánek [1,3,*], Robert E. Jones [1,2,^], Eric Salomaki [4,^], Serafim Nenarokov [4], Fabien Burki [5,6], Marek Eliáš [3], Laura Eme [7], Andrew J. Roger [8], Antonis Rokas [9] Xing-Xing Shen [10] Jürgen Strassert [5,11], Martin Kolísko [4,12#], and Matthew W. Brown [1,2,#]

1. Department of Biological Sciences, Mississippi State University, Mississippi State, MS, USA
2. Institute for Genomics, Biocomputing & Biotechnology, Mississippi State University, Mississippi State, MS, USA
3. Department of Biology and Ecology, Faculty of Science, University of Ostrava, Czech Republic
4. Institute of Parasitology, Biology Centre Czech Academy of Sciences, České Budějovice, Czech Republic
5. Department of Organismal Biology, Uppsala University, Norbyvägen, Uppsala, Sweden
6. Science for Life Laboratory, Uppsala University, Norbyvägen, Uppsala, Sweden
7. Unité d'Ecologie, Systématique et Evolution, CNRS, Université Paris-Saclay France
8. Department of Biochemistry and Molecular Biology, Dalhousie University, Halifax, Nova Scotia, Canada
9. Department of Biological Sciences, Vanderbilt University, Nashville, TN, USA
10. State Key Laboratory of Rice Biology and Ministry of Agriculture Key Lab of Molecular Biology of Crop Pathogens and Insects, Institute of Insect Sciences, Zhejiang University, Hangzhou 310058, China

11. Leibniz Institute of Freshwater Ecology and Inland Fisheries, Ecosystem Research, 12587 Berlin, Germany
12. University of South Bohemia, Faculty of Science, České Budějovice, Czech Republic

* and ^ Equally contributing author
# Corresponding authors

Contents

Introduction

What is PhyloFisher?

PhyloFisher is a software package written in Python3 that can be used for the creation, analysis, and visualization of phylogenomic datasets that consist of protein sequences from eukaryotic organisms. Unlike many existing phylogenomic pipelines, PhyloFisher comes with a manually curated database of 240 protein coding genes from 304 eukaryotic taxa. However, the tool can be used to analyze any user-provided database. PhyloFisher is also equipped with a set of utilities to aid in running routine analyses performed during construction of or on constructed phylogenomic datasets and visualization of the results. If you use a component of PhyloFisher in your research, please cite:

Tice et al. (2021). PhyloFisher: A phylogenomic package for resolving eukaryotic relationships. doi. xxxxxxx.

Compatible Operating Systems

Linux

macOS

Windows (forest.py & ParaSorter ONLY!)

Space Requirements

Allow ~4GBs for installation including provided database. ~2GBs software only.

Third Party Software Utilized by PhyloFisher

ASTRAL-III v. 5.7.3 (Zhang et al., 2018)

BLAST v. 2.9.0 (Altschul et al., 1990)

BMGE v. 1.1.2 (Criscuolo and Gribaldo, 2010)

CD-HIT v. 4.8.1 (Fu et al., 2012)

DIAMOND v. 09.24 (Buchfink et al., 2015)

dist_est v.1.0 (Susko et al., 2003)

Divver v. 1.01 (Ali et al., 2019)

ETE3 3.1.1 (Huerta-Cepas et al., 2016)

FastTree v. 2.1.11 Price et al., 2010

HMMER v. 3.2.1 (Mistry et al., 2013)

MAFFT v.7.455 (Katoh and Standley, 2013)

MAMMaL v.1.1.1 (Susko et al., 2018)

OrthoMCL v 5.0 (Chen et al., 2006)

PREQUAL v. 1.02 (Whelan et al., 2018)

RAxML v. 8.2.12 (Stamatakis, 2014)

trimAl v.1.4.rev15 (Capella-Gutiérrez et al., 2009)

PhyloFisher Package Contents

- Manually curated database of 240 orthologs and their paralogs from 304 eukaryotic taxa. These taxa cover the known diversity of eukaryotes.
- Main Workflow tools for:
  - The preparation of custom databases composed of protein coding genes in lieu of the provided database
  - Ortholog and paralog "mining" from input proteomes
  - Evaluating completeness in the database of newly input data
  - Automated removal of non-homologous sites, alignment, trimming, length filtering and single gene tree construction
  - Visualization of single gene trees and interactive selection of orthologs, paralogs, and removal of contamination
  - Generation of final statistics post ortholog selection and contamination removal
    - Can be subset by user provided thresholds of completeness or by assigned taxonomy.
  - Creation of chimeric taxa for use in phylogenomic analyses
  - Automated concatenation of phylogenomic matrices
- Utilities for:
  - Alternative genetic code prediction
  - Compositional bias testing (amino acid re-coding)
  - The removal of fast evolving sites from a phylogenomic dataset
  - The removal of fast evolving taxa from a phylogenomic dataset
  - Detection and removal of the most heterotachious sites in a phylogenomic dataset
  - Examining the number of occurrences of clades of interest in bootstrap trees
  - Matrix construction from randomly re-sampled genes from a phylogenomic dataset
  - Binning single-protein trees based on Relative Tree Certainty (RTC)
  - collapsing of multiple proteomes to produce a single "most complete" proteome with regards to the database (useful for single-cell data and to form chimeric taxa from closely related species)

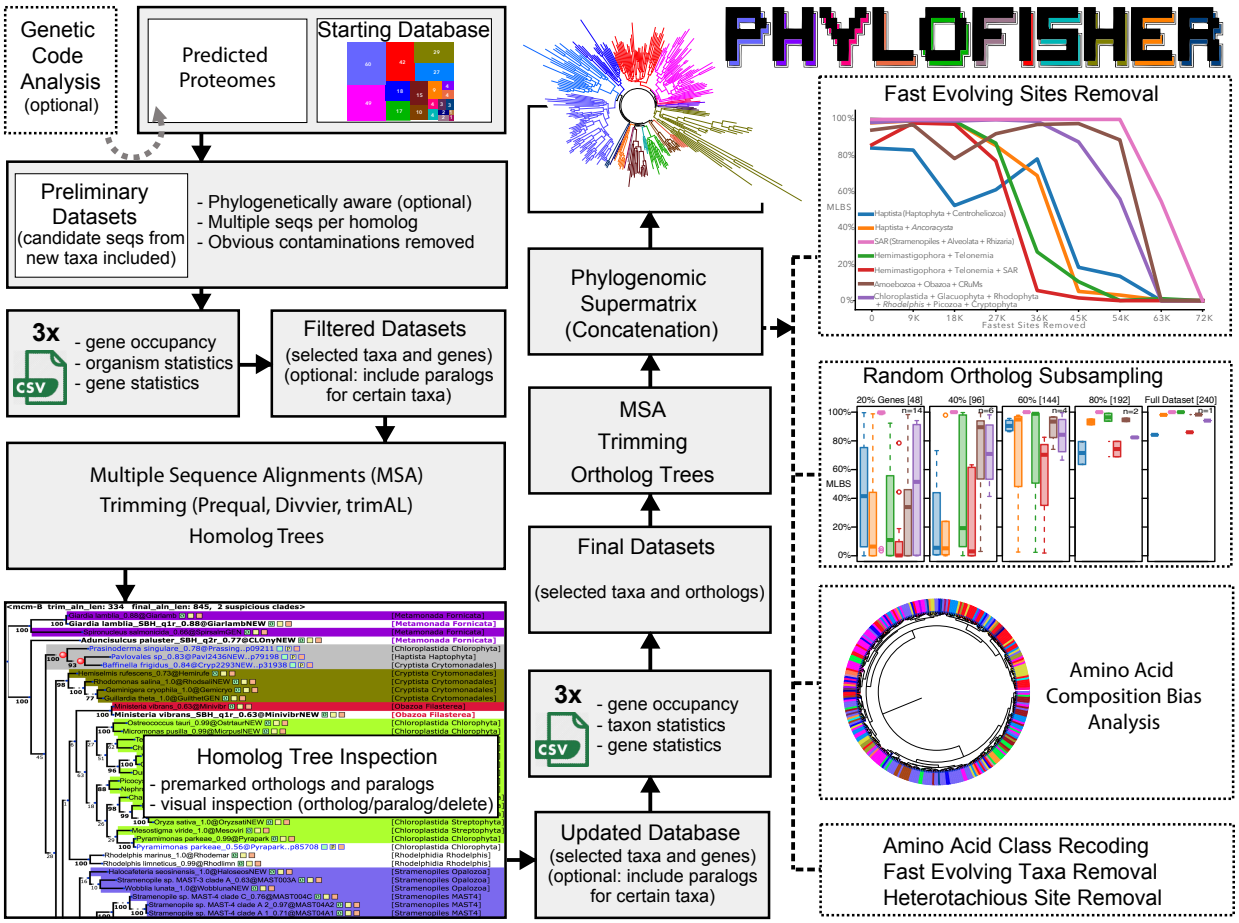

Figure 1: Overview of the PhyloFisher workflow and package contents. The PhyloFisher package consists of a manually curated database of 240 protein coding genes and their paralogs from 304 eukaryotic taxa; a series tools to perform the essential steps of phylogenomic dataset construction (homolog collection, single-protein tree construction, removal of paralogs and contaminants, matrix concatenation); and many pre and post-construction analyses necessary for a publication-quality phylogenomic study.

Terminology Used Throughout the PhyloFisher Manual

Throughout this manual different collections/types of data are referred to using specific names in an attempt to provide clarity to the exact set of data being referenced (Figure 2).

The seven collections of data referenced in the manual are:

1. **Database** - the database is a collection of manually curated sequences demarcated as either orthologs or paralogs. This can refer to either the provided database or a custom database supplied by the user. Final decisions regarding orthology and paralogy made by a user during manual curation of sequences collected from newly input proteomes are stored here. The initial database provided has been manually curated prior to its release, however the application of different phylogenetic methods, manual curation practices, and taxon selection will cause the user's database to change over time.

2. **Working Dataset** - the purpose of the working dataset is to aid in the identification of orthologs, paralogs, and contaminants from input proteomes. The working dataset is created when sequences are collected by fisher.py from input proteomes and appended to the copies of the corresponding homolog files from the database. These data will then move through the PhyloFisher workflow until

final decisions based on manual curation have been applied to the database. A working dataset is not revisited after decisions from manual curation have been made and applied to the database.

3. **Phylogenomic Dataset** - A set of orthologs and taxa in the form of individual ortholog alignments and a concatenated matrix of these alignments that have been selected from the database to perform phylogenomic analyses on.

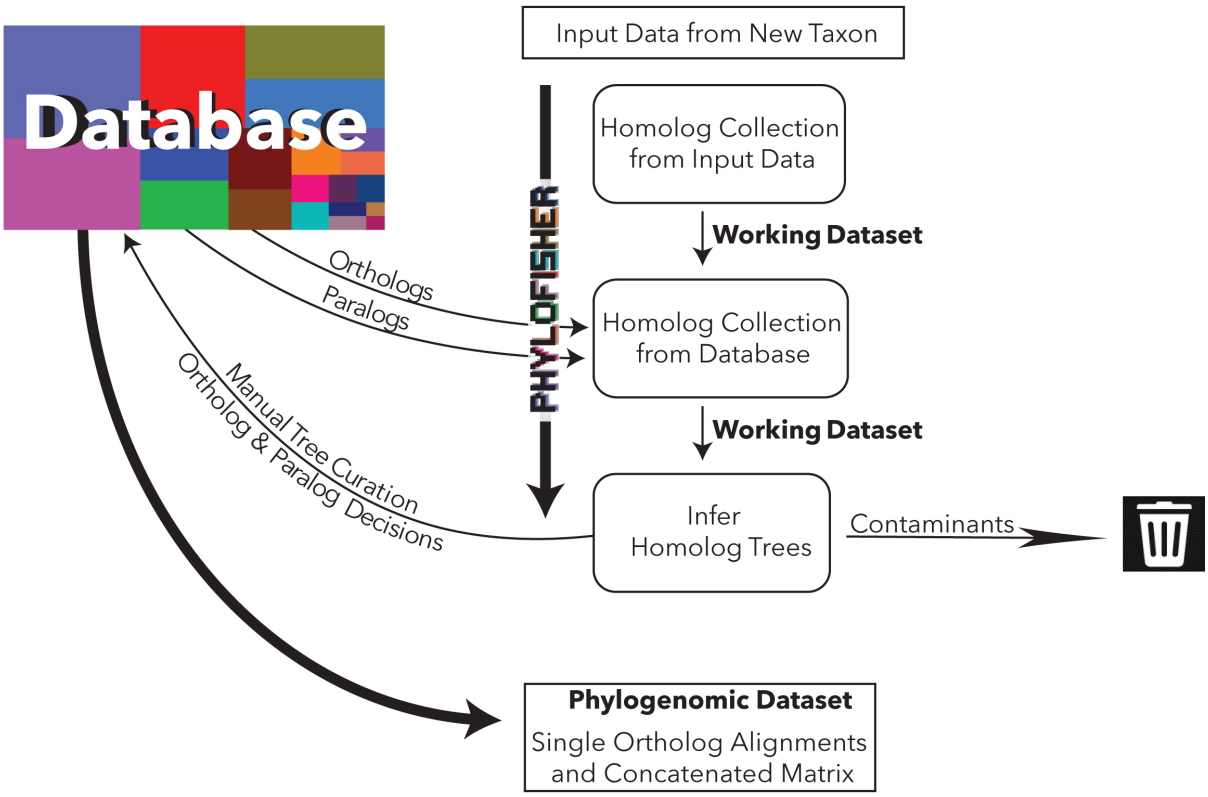

Figure 2: Flow chart of the PhyloFisher workflow documenting how collections of data are referenced throughout the manual.

4. **Gene(s)** - The term gene is used throughout this manual to refer to a group of closely related homologs (similar to a gene family). These genes are named based on the corresponding human or *Arabidopsis thaliana* orthologs that were used as queries in BLAST searches to collect sequences from other eukaryotic organisms during the construction of the phylogenomic database developed in Tice et al., 2016. The aforementioned database was used as a starting point for construction of the PhyloFisher v1.0 database. We have chosen not to use either of the terms "gene family" or "orthogroup" because these genes only contain the most similar sequences to the initial search query rather than all potential members.

5. **Ortholog(s)** - The term ortholog, in the context of PhyloFisher, refers to a sequence(s) within the database that will ultimately be used for the construction of phylogenomic datasets. Additionally, the term ortholog is applied to the putative sequence from each newly added taxon that was selected by the fisher.py algorithm for use in phylogenomic datasets, pending decisions made during manual curation.

6. **Paralog(s)** - The term paralog, in the context of PhyloFisher, refers to sequences that are included in working datasets to enable proper ortholog selection and to reveal frequency of duplications in a particular gene and taxonomic group. The databases paralogs help users avoid the inclusion of sequences from different paralogs for different species and thus avoid the generating incorrect phylogenetic

signal in the final dataset. Sequences denoted paralogs after each round of manual curation are not to be used in the construction of phylogenomic datasets. Additionally, the term paralog refers to sequences that are collected from newly input taxa that were not selected by the fisher.py algorithm as the putative ortholog, pending decisions made during manual curation. They can also represent hidden contaminants or variants of the same contig (e.g. due to a retained intron(s)). After manual curation all sequences that are denoted as "paralogs" will be stored in the database to be utilized during manual curation of subsequent taxon additions.

7. **Homolog(s)** - The term homolog is used when referring to datasets that consist of both orthologs and paralogs (such as the working dataset), or to describe the relationship between orthologs and paralogs.

PhyloFisher's Provided Database

PhyloFisher's Provided Database Is...

The PhyloFisher package is equipped with a manually curated database of 240 protein coding genes from up to 304 eukaryotes that can be used for phylogenomic analyses of eukaryotic organisms. Details on the genes and taxa included in the database can be found currently at the below link. A metadata file (metadata.tsv) that provides information on the proteomes used in database construction is included with the database and can be found in the directory database/. In addition to a single ortholog of each of the databases's 240 genes, a database of each gene's paralogs is maintained.

PhyloFisher's Provided Database Is Not...

A set of genes that are universally present and single copy in all eukaryotic taxa included in the database.

How a User's PhyloFisher Database Changes Over Time

Regardless of whether a user chooses to utilize the provided database or their own custom database, the nature of the PhyloFisher workflow changes the chosen database overtime in a way that increases the database's ability to inform a user on levels and phylogenetic location of paralogy for each gene in the database. Typically, the addition of new taxa to phylogenomic databases, especially underrepresented lineages, often reveals previously unknown or "hidden" paralogy. This previously unknown paralogy can exist at any level from shallow and species specific to ancient paralogy that simply could not be inferred with the previous database. Depending on the level of paralogy and the user's phylogenetic question, accidental inclusion could have dramatic effects on phylogenomic inference and subsequent interpretations of evolutionary events based on the resulting phylogenomic tree. Once revealed, newly discovered paralogs are demarcated as such and maintained in the paralog portion of the database to aid in the identification of the paralog from newly added taxa in subsequent rounds of addition and prevent inclusion in future phylogenomic datasets. By continuously keeping a record of all known paralogs regardless of their emergence in evolutionary time a user's decision regarding ortholog selection will always be the most informed even if/when the depth of their phylogenetic questions change over time. Our strategy also allows for the construction of a larger/more complete phylogenomic dataset as the most well represented orthologs can be chosen for phylogenomic analyses.

Users do have the option of slimming down the database through complete deletion of taxa. User's may also choose to exclude an organism and/or the organism's paralogs from the working dataset used for single gene tree construction. While these options are provided to ease the computational burden of single gene tree construction, we highly recommend using as many taxa and their associated paralogs as feasible. Experience has shown that including more taxa will lead to more informed ortholog selection and therefore a more reliable final tree.

Ortholog Selection via the Fisher Algorithm

Ortholog selection and paralog demarcation is performed using the python script fisher.py (Figure 3) that requires two inputs: predicted proteome(s) to be added to the database and an input-metadata file containing key information about them. To remove redundancy in the input proteome (typical when proteomes are

predicted in silico from transcriptomic data), fisher.py invokes CD-HIT v.4.8.1 (Fu et al., 2012) with the option "-c 0.98", which clusters sequences globally at a similarity threshold of 98%. Next, hmmsearch from the software package HMMER v. 3.2.1 (Mistry et al., 2013) is run on the input proteome using a pre-computed profile Hidden Markov Model (HMM) of each protein alignment in the database. Up to a user-defined number of sequences are collected (default = 5) that meet the significance threshold (e-value < 1e -10). If no sequence meets the significance threshold, the script moves on to the next protein. If sequences are found (up to the user-defined number), they are prioritized based on the level of significance (i.e., the sequence with the most significant hit is preliminarily denoted as the most likely putative ortholog with other included sequences preliminarily denoted as putative paralogs). From here, the algorithm proceeds in either of two directions, depending on information given in the input metadata file. In the two routes outlined below, sequences can be added to or removed from the initial sequences collected by hmmsearch, as they meet or fail to meet the outlined criteria. The prioritization of a sequence can also change within the list based on the criteria outlined below. At the end of either route, if only one sequence remains, it is considered the putative ortholog for the input taxon. If more than one remains, the sequence with the highest priority in the list is considered the putative ortholog and the other retained sequences are considered putative paralogs. All retained are added to the corresponding single-protein alignment.

Default Route

If no related species present in the database is/are listed in the "Blast Seed" column of the input metadata file to use as specific queries, each sequence collected from the initial hmmsearch is used as the query in a search using DIAMOND v. 09.24 (Buchfink et al., 2015) against OrthoMCL v 5.0 (Chen et al., 2006) with the options "blastp -e 1e-10 –more-sensitive." Any sequence that has a significant hit (e-value < 1e -10) to a bacterial orthogroup in OrthoMCL is discarded along with any sequence that does not have a significant hit to the OrthoMCL orthogroup corresponding to the respective profile HMM used to retrieve the sequence. Retained sequences are again used as queries in a DIAMOND search using the parameters given above against the database. Query sequences whose best hit represents the conserved gene corresponding to the initial profile HMM that retrieved the query sequence is retained and added to the alignment. The remaining sequence that has the highest priority based on the initial hmmsearch is considered to be the putative ortholog and any other surviving sequences are labelled putative paralogs.

Phylogenetically Informed Route

The input metadata file gives users the option to specify taxa already in the database whose sequences will be used as queries for blast searches against the new organism's proteome; typically these should be closely related to the newly added taxon. Any number of species already present in the database may be chosen to serve as specific queries, but the algorithm will use them in the order provided, and, in cases outlined below, may not proceed to subsequent taxa. If species to be used as specific queries are listed in the input metadata file, fisher.py will initially pick the sequence representing a particular orthologous gene group ("ortholog" for simplicity) from the first organism listed, if present. If absent in the database, fisher.py will sequentially check for the ortholog from the remaining taxa listed. If the orthologs from one or more of the subsequent taxa are found, the algorithm proceeds as outlined below. If no ortholog from all listed organisms can be found, fisher.py will proceed to the default route for this particular protein. If an ortholog is present in at least one of the listed species, then its sequence is used as the query in a BLAST (Camacho et al. 2009) search against the input organism's predicted proteome. If no significant hit (e-value < 1e -10) is found, fisher.py will use the ortholog of the next listed species, if one is provided. If no other species is listed, the ortholog is not present in the database for the listed species, or if the orthologs of all listed species do not return a significant BLAST hit, the protein is skipped for the input taxon. If significant BLAST hits are found, up to a user-defined number of those (default = 5) are collected and examined further. The sequences from the original hmmsearch are reprioritized based on the level of significance of the BLAST hit. The sequence with the most significant hit becomes the priority sequence, unless it was not also collected by the initial hmmsearch: any sequence that produces a significant BLAST hit but was not collected in the initial hmmsearch is discarded. The retained sequences are then used as queries against OrthoMCL. Any sequence that has a significant hit (e-value < 1e -10) to a bacterial orthogroup in OrthoMCL is discarded along with any sequence that does not have a significant hit to the OrthoMCL orthogroup corresponding

to the respective profile HMM used to retrieve the sequence. Next, all the sequences that have passed the filtering in the previous step are compared with blast against the database of conserved marker proteins. If the query sequence's best blast hit is a sequence corresponding to the profile HMM that retrieved the query sequence in the initial search, the sequence is retained. If the sequence's best blast hit is a sequence from another alignment, the sequence is still retained, but is written out to the file "non_corresponding_hits.txt" with a note on which protein from the database represents the best blast hit. Sequences are then added to the corresponding dataset from the database and aligned using MAFFT v.7.455 (Katoh and Standley, 2013) with the parameters "–auto –reorder", trimmed with trimAl v.1.4.rev15 (Capella-Gutiérrez et al. 2009) with a gap threshold of 0.2, and subjected to phylogenetic tree reconstruction via FastTree v. 2.1.11 (Price et al., 2010) with default parameters. The resulting tree is examined using the python package ETE3 (Huerta-Cepas et al., 2016). By default sequences that branch sister to or within a clade composed of organisms with the same assigned higher taxonomy are retained while sequences that branch outside of the assigned higher taxonomy are eliminated unless no sequence branches with sequences assigned the correct higher taxonomy. This happens regardless of previous criteria. Higher taxonomy for taxa is derived either from the database's metadata file or the input metadata file. Finally, a length filtration step will remove any sequences that have more than 70% gaps in the trimmed alignment. All retained sequences are then added to their corresponding alignments from the database, with the sequence that received the highest level of priority being denoted the putative ortholog and all others denoted as putative paralogs.

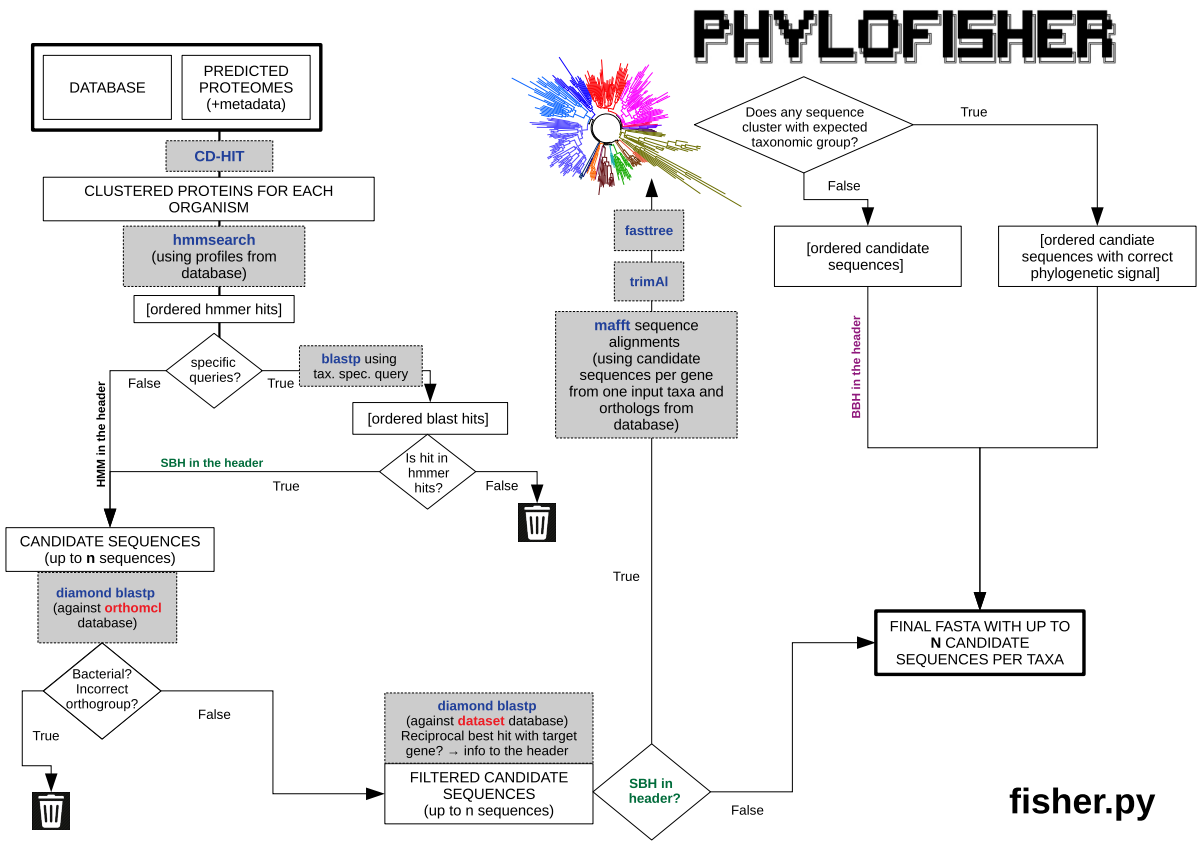

Figure 3: Flowchart of homolog collection performed by the PhyloFisher python script fisher.py. Briefly, each predicted proteome of a new taxon to be added is processed through either a default route, or a phylogenetically aware route that utilizes the manually curated orthologs from closely related taxa chosen by the user (and present in the database) as search queries against the proteome of the new taxon. Up to a user-defined number of collected sequences are reprioritized or eliminated based on a set of criteria designed to maximize correct demarcation of the desired ortholog and related paralogs while avoiding contaminating sequences. See section "Ortholog Selection via the Fisher Algorithm" of this guide for a detailed description of the logic, third-party software and associated parameters utilized.

Automated Filtering, Alignment, Trimming, and Single Gene Tree Construction

PhyloFisher also includes a python script sgt_constructor.py to automatically filter, align, trim, length filter, and construct phylogenetic trees from all homolog alignments. The script takes the output files from fisher.py (original single-protein alignments that now contain newly added sequences selected by the fisher algorithm for input taxa as well as the previously denoted paralogs from the database) and removes any dashes to produce an unaligned set of sequences for non-homologous character removal via PREQUAL v. 1.02 (Whelan et al., 2018) using the default settings. Next, a length filtering step is performed to minimize the inclusion of proteins predicted from fragments of the same gene. This is common in proteomes predicted from transcriptomes. First, sequences are aligned with the program MAFFT using the settings "–globalpair –maxiterate 1000 –unalignlevel 0.6". This is followed by assessment of alignment error and uncertainty by the program Divvier v. 1.01 (Ali et al., 2019) using the options "-partial -mincol 4 -divvygap". The resulting alignments are trimmed using BMGE v. 1.1.2 (Criscuolo and Gribaldo, 2010) with a gap-rate cutoff of 0.3. Any sequence whose length is less than half the total alignment length, after BMGE filtering, is removed. After removal of "short" sequences, files are prepared for single-protein phylogenetic tree construction by rerunning MAFFT and Divver as above, followed by trimming with the program trimAl with a gap threshold

of 0.01. Finally, single-protein tree reconstruction is performed using RAxML v. 8.2.12 (Stamatakis, 2014) with the options "-m PROTGAMMALG4XF -f a -x 123 -N 100 -p 12345". When single gene tree construction begins "sgt_constructor.py" will check to see how many gene trees are to be built and how many threads were provided by the user. If the number of gene trees to be built is greater than the number of threads provided, "sgt_constructor.py" will run as many jobs as it can using a single thread each. This process will continue until all gene trees have been built. If the total number of gene trees to be built is less than the number of threads available "sgt_constructor.py" will distribute all threads available as evenly as possible.

Getting Started with PhyloFisher

PhyloFisher can be installed and updated via Conda or PIP.

Installation via Anaconda

1. If you do not already have conda installed, please install via Anaconda.

   NOTE: Installation via Miniconda will complete. However, all non-python dependencies will not be installed. In short, installation via Miniconda will NOT work.

2. Install mamba via conda

   ```
   conda install mamba
   ```

3. Prepare a conda virtual environment via mamba :

   ```
   mamba create -n fisher
   ```

4. Activate the environment you just created:

   ```
   conda activate fisher
   ```

5. Add the following Anaconda Cloud channels to your channels:

   ```
   conda config --append channels phylofisher
   conda config --append channels bioconda
   conda config --append channels conda-forge
   ```

6. Now install PhyloFisher:

   ```
   mamba install phylofisher
   ```

   NOTE: when conda virtual environment is active all PhyloFisher python scripts are now in your path and you can access them by typing the first few letters and tab.

Installation via PIP

1. Install PhyloFisher via PIP

   ```
   pip install phylofisher
   ```

2. Install non-python dependencies

   linux: `install_deps.py`

macOS: `install_deps.py --gxx </path/to/g++/executable>`

NOTE: The gcc compiler is required for installing non-python dependencies on macOS. If you do not already have the gcc compiler it can be quickly installed via brew by running:

```
brew install gcc
```

NOTE: After PIP installation, all PhyloFisher python scripts are now in your path and you can access them by typing the first few letters and tab.

Installation of forest.py & ParaSorter locally (including Windows)

Installation of these two tools on a users local machine is required for downstream steps in the PhyloFisher workflow. Although the entire PhyloFisher software package does not run on Windows, this minimal download will. This download will provide users (including Windows users) the graphical tree parsing aspects of the package on their local machines. This minimal download of forest.py and ParaSorter can be retrieved from the "ParaSorter" section of PhyloFisher GitHub Repository.

Further documentation on the usage of both forest.py and Parasorter can be found in steps 9 and 10 in the "Detailed Example Workflow" section of this manual respectively.

Download PhyloFisher's Provided Database or Prepare a Custom Database.

Download PhyloFisher's Provided Database

1. Retrieve the provided starting database via wget:

```
wget https://ir.library.msstate.edu/bitstream/handle/11668/19731/Tice_etal.PhyloFisherDatabase_v1.0_Apr.11.2021.tar.gz
```

2. Uncompress the .tar.gz file:

```
tar -xzvf Tice_etal.PhyloFisherDatabase_v1.0_Apr.11.2021.tar.gz
```

The uncompressed database directory contains the subdirectories and files detailed below.

- database/
  - backups/
    - {Month}_{Day}_{Year}.tar.gz (a compressed file containing backups of the two database folders orthologs/, paralogs/, the database file metadata.tsv, and tree_colors.tsv
  - datasetdb/
    - datasetdb.dmnd (a diamond database of the PhyloFisher v 1.0 orthologs)
    - datasetdb.fasta (a fasta file of the PhyloFisher v 1.0 orthologs used to construct the diamond database
  - orthologs/
    - {gene_name}.fas (240 fasta files of the PhyloFisher v 1.0 orthologs)
  - orthomcl/
    - bacterial (abbreviated names of bacterial species present in OrthoMCL v. 5.0)
    - gene_og (a tsv file detailing the names of the 240 PhyloFisher v 1.0 genes and their corresponding OrthoMCL v. 5.0 orthogroup number(s))
    - orthomcl.diamonddb.dmnd (diamond database of OrthoMCL v. 5.0)
  - paralogs/
    - {gene_name}_paralogs.fas (240 fasta files of the PhyloFisher v 1.0 paralogs)

  - profiles
    * {gene_name}.hmm (240 profile hmm files of the PhyloFisher v 1.0 orthologs)
  - proteomes/
    * {Unique_ID}.faa.tar.gz (complete proteomes of all taxa in the database)
  - metadata.tsv (tsv file of containing metadata for species in the PhyloFisher v 1.0 database. Detailed extensively in the section "Detailed Explanation of the PhyloFisherDatabase_v1.0 Metadata File" of this manual)
  - tree_colors.tsv (a comma separated file used to color taxa based on taxonomy for manual inspection of single gene trees.)

- TEST/
  - database/
    * the same contents as PhyloFisherDatabase_v1.0/database/ (detailed above) except with only one test gene, RPL5, and no backups directory.
  - analysis/
    * db_taxa_stats.tsv.bak (a prefilled db_taxa_stats.tsv to be used in the installation test.)
    * input_metadata.tsv (installation test input metadata)
    * SacccereTEST.faa (input proteome used to test for successful installation)
    * testInstall.sh (bash script that runs the PhyloFisher workflow on the test data to test for successful installation)

Prepare a Custom Database

1. Retrieve the recommended PhyloFisher directory structure, OrthoMCL v. 5.0 database, example metadata.tsv, and tree_colors.tsv file via wget:

   ```
   wget https://ir.library.msstate.edu/bitstream/handle/11668/19731/Tice_etal.PhyloFisher_FOR_CUSTOM_DATABASE_Mar.29.2021.tar.gz
   ```

2. Uncompress the file PhyloFisher_FOR_CUSTOM_DATASET_Jan.28.2021.tar.gz

   ```
   tar -xzvf Tice_etal.PhyloFisher_FOR_CUSTOM_DATABASE_Mar.29.2021.tar.gz
   ```

3. Move into the directory PhyloFisher_FOR_CUSTOM_DATABASE

   ```
   cd PhyloFisher_FOR_CUSTOM_DATABASE
   ```

4. Place your single gene files of orthologs in the directory /database/orthologs/

   - The ortholog files must be in fasta format.
   - Each ortholog file must be named with the following convention {gene_name}.fas.
     - Ex. RPL7.fas
   - Each individual taxon should have a Unique ID as the header in all ortholog files. This Unique ID must be the the same in all ortholog files.
   - Each taxon can be present only once in each ortholog file.

5. Place files of known paralogs for each gene in the directory /database/paralogs/ (OPTIONAL)

   - Each gene file must be named with the following convention {gene_name}_paralogs.fas.
     - Ex. RPL7_paralogs.fas
   - Each individual taxon should have a Unique ID as the header in all paralog files. This Unique ID must be the the same in all paralog files and the corresponding ortholog files.

- Each taxon can be present more than once in each paralog file.

Some notes about sequence headers:

- Each sequence header (sequence header = Unique ID) within a file must be unique. Sequence headers must be the same across all files for a taxon and must be the same as the Unique ID provided in the metadata.tsv file for the taxon. This is true for both ortholog and paralog fasta files.
- Sequence headers cannot contain underscores "_", at symbols "@" white spaces, or double dots "..". This is true for both ortholog and paralog fasta files.
- If you provided separate sequence files containing paralogs, each sequence header within each file will have "..p<randomfivedigitnumber>" appended to the end by build_database.py (the python script that will prepare the custom database for use in PhyloFisher).

6. Place the complete proteome of each taxon present in the ortholog files in /database/proteomes
   - All proteomes must be in fasta format
   - All proteomes must be tar and gzipped and follow the naming convention {Unique ID}.tar.gz
7. Fill out the metadata.tsv file found in PhyloFisher_FOR_CUSTOM_DATABASE/database. Deailted instructions on preparing the metadata.tsv file can be found here
8. Run build_database.py. Detailed instructions on build_database.py can be found here.

Check if your Installation was Successful (ANACONDA):

1. Move into the directory PhyloFisherDatabaset_v1.0/TEST/analysis/

   ```
   cd TEST/analysis/
   ```

2. delete the *_out directories

   ```
   rm -r *_out
   ```

3. run the test data script "testinstall.sh"

   ```
   ./testInstall.sh
   ```

4. Although "testinstall.sh" produces many files and folders all you need to do to check if installation was successful is open "forest_out/Rpl5.tsv". You should see Saccharomyces cerevisiae has been added to the bottom of the file. If this occurred and no error messages were printed to the screen during the run your installation was successful!

5. When you are done using PhyloFisher run:

   ```
   conda deactivate
   ```

   This command will deactivate the fisher conda virtual environment created here.

Check if your Installation was Successful (PIP):

1. Add the below to your ~/.bashrc (linux) or ~/.bash_profile (macOS)

   ```
   export PATH="$HOME/bin:$PATH"
   ```

2. Now source your /.bashrc (linux) or ~/.bash_profile (macOS)

   `source ~/.bashrc`(linux)
   `source ~/.bash_profile` (macOS)

3. Move into the directory TEST/analysis/

   `cd PhyloFisherDatabase_v1.0/TEST/analysis/`

4. delete the *_out directories

   `rm -r *_out`

5. run the test data script "testinstall.sh"

   `./testInstall.sh` (This should take ~60s to finish.)

6. Although "testinstall.sh" produces many files and folders all you need to do to check if installation was successful is open "forest_out/Rpl5.tsv". You should see Saccharomyces cerevisiae has been added to the bottom of the file. If this occurred and no error messages were printed to the screen during the run your installation was successful!

Detailed Explanation of the PhyloFisherDatabase_v1.0 Metadata File

The file metadata.tsv initially contains information regarding the data used to construct the database. As users add taxa to the database metadata.tsv acts as a permanent archival file and is updated after each round of addition of new taxa.

The contents of metadata.tsv can be explored using the provided utility "explore_database.py". A user should never be required to manually open this file to view its contents. However, the structure of metadata.tsv is detailed below.

- metadata.tsv is a tab delimited file with six columns:
  1. Unique ID - an abbreviated name for each taxon. We have followed an eight-letter convention. For user's making a metadata.tsv for a custom database,**Unique IDs cannot cannot contain underscores "_", at symbols "@", double dots "..".**
  2. Long Name - Full name of the input taxon. For user's making a metadata.tsv for a custom database, **the long name cannot cannot contain underscores "_", at symbols "@", double dots "..".**
  3. Higher Taxonomy - The highest taxonomic rank for the input organism desired. We have chosen to use the "supergroup" level here but any user defined rank is accepted. However, we do recommend users choose a highly inclusive taxonomic rank here (phylum or class level in Linnaean terms) to promote the best possible performance of the fisher algorithm and other downstream tools.
  4. Lower Taxonomy - a taxonomic rank above the genus level for the input taxon
     - Notes on Taxonomy: The default taxonomic terms in "metadata.tsv " can be replaced with user preferred terms. If terms are to be replaced though, we recommend replacing them with synonyms of similar rank. DO NOT MIX SYNONYMOUS TERMS (see example below). This will lead to misbehavior in the fisher algorithm and wrongly denoted suspicious clades during generation of files for visualisation and manual inspection of single gene trees. The same erratic behavior will likely occur if our chosen taxonomic ranks are split up and replaced by several less inclusive taxonomic terms.
       * Example Taxonomic Change: If the term "Chromist" is preferred over "Stramenopiles" replace all instances of "Stramenopiles" in metadata.tsv with "Chromist" before adding new taxa with "Chromist" chosen as the higher taxonomic term in input_metadata.tsv. DO NOT MIX THE TWO, erratic behavior will ensue. Go here for more information regarding input_metadata.tsv requirements related to metadata.tsv.
  5. Data Type - A place to add a note about the type of data being added (ex. transcriptomic, genomic, EST . . .)

6. Source - A place to add notes about the source of the data such as accession numbers, "in-house information", strain information etc.

NOTE: COMMA (,) MAY NOT BE USED ANYWHERE IN METADATA.TSV!

Best Practice Suggestions For Maintaining a Single Lab Database

Multiple lab members may be asked to add to the same ever-growing lab version of the database. To accomplish this, all lab members will want to set up their configuration file to add to the same database (instructions below). In order for the lab database to be as accurate as possible, only one round of addition should be run at a time and another should not be started until after apply_to_db.py (introduced below) is run to implement decisions from the previous round of manual curation on a working dataset to the database.

Quick Start (Main Workflow Only)

***Use `<script>.py -h` to see all options and their descriptions.

1. Make a project directory (named anything) and move into it

```
mkdir <project_directory>
cd <project_directory>
```

2. explore_database.py
   - Explore the contents of PhyloFisher's provided starting database to help make decisions on proteomes to add (Optional)
   - Usage: `explore_database.py [OPTIONS]`

3. Fill out "input_metadata.tsv" for each proteome to be added to the database. See Table 1 for an example and detailed explanation of input_metadata.tsv or check the example provided in the starting database directory PhyloFisherDatabase_v1.0/TEST/analysis/input_metadata.tsv

4. config.py
   - config.py allows users to configure the analysis directory
   - Usage: `config.py [OPTIONS] -d <database_folder> -i <input_file.tsv>`

5. fisher.py
   - Run fisher.py for homolog collection:
   - Usage: `fisher.py [OPTIONS]`

6. informant.py
   - Prepare preliminary statistics for and select organisms and genes prior to homolog tree construction:
   - Usage: `informant.py [OPTIONS] -i <fisher_out_dir>`

7. working_dataset_constructor.py
   - Apply your selections from the previous step:
   - Usage: `working_dataset_constructor.py [OPTIONS] -i <fisher_out_dir>`

8. sgt_constructor.py
   - Length filtration, trimming and homolog tree construction:

- Usage: `sgt_constructor.py [OPTIONS] -i <working_dataset_constructor_out_dir>`

9. forest.py
   - Tree visualization:
   - Usage: `forest.py [OPTIONS] -i <dir_containing_trees>`

10. apply_to_db.py
    - Apply parsing decisions and add new data to the database:
    - Usage: `apply_to_db.py [OPTIONS] -i <forest_out_dir>`

11. select_taxa.py (Optional)
    - Select a subset of taxa from the database for final phylogenomic analyses
    - Usage: `select_taxa.py [OPTIONS]`

12. select_orthologs.py (Optional)
    - Select a subset of orthologs from the database for final phylogenomic analyses
    - Usage: `select_orthologs.py [OPTIONS]`

13. prep_final_dataset.py
    - Collect taxa and genes for final phylogenomic analyses
    - Usage: `prep_final_dataset.py [OPTIONS]`

14. matrix_constructor.py
    - Build phylogenomic matrix (Concatenation):
    - Usage: `matrix_constructor.py [OPTIONS] -i <prep_final_dataset_out_dir>`

Detailed Example Workflow

1. Make a project directory in your desired location (this location can be anywhere on your computer and is independent of the location of the database). Once you have made the project directory move into it:

   ```
   mkdir <project_directory>
   cd <project_directory>
   ```

2. Fill out the file "input_metadata.tsv" (see PhyloFisherDatabase_v1.0/TEST/analysis/input_metadata.tsv and/or Table 1 for an example) with information about your input taxa. The input_metadata.tsv file contains nine columns (detailed below) of information about input taxa that is used throughout a PhyloFisher analysis. Each row in the file should be a new taxon to be added to the database. The input_metadata.tsv file can be most easily completed in a spreadsheet software such as Microsoft Excel and saved as a tab delimited text file. Place the completed input_metadata.tsv file in your project directory created in the previous step. After completion of a PhyloFisher run the information in input_metadata.tsv will be appended to the permanent file "metadata.tsv" for any input taxon that is to be permanently added to the database. The file "metadata.tsv" serves as a long-term record of the contents of the database as taxa are added over time.

- input_metadata.tsv is a tab separated file with nine columns (detailed below and in Table 1)
  1. Location (Required) - the absolute path to the directory where the input proteome(s) are located.
  2. File Name (Required) - the name of the input proteome file.
  3. Unique ID (Required) - an abbreviated name for the input taxon (we have followed an eight-letter convention in the provided database but any number of characters may be used. **However, Unique IDs cannot contain underscores "_", at symbols "@", double dots "..", white space, or asterisk "*" (with one exception outlined below)**.
  4. Higher Taxonomy (Required) - The highest taxonomic rank for the input organism desired. We have chosen to use the "supergroup" level here but any user defined rank is accepted. However, we do recommend users choose a highly inclusive taxonomic rank here (phylum or class level in Linnaean terms) to promote the best possible performance of the fisher algorithm and other downstream tools.
  5. Lower Taxonomy (Required) - a taxonomic rank above the genus level for the input taxon.
     - NOTES ON TAXONOMY: Aspects of the PhyloFisher workflow require that taxonomic terms for both categories provided in "input_metadata.tsv" be the same as in the permanent archival file "metadata.tsv". The only exception occurs when adding an organism with a taxonomic affiliation not represented in the database (see below). The default taxonomic terms in "metadata.tsv " can be replaced with user preferred terms. If terms are to be replaced though, we recommend replacing them with synonyms of similar rank. **DO NOT MIX SYNONYMOUS TERMS (see example below)**. This will lead to misbehavior in the fisher algorithm and wrongly denoted suspicious clades during manual inspection of single gene trees. The same erratic behavior will likely occur if our chosen taxonomic ranks are split up and replaced by several less inclusive taxonomic terms.
     - New taxonomic terms) If you are adding an organism with a taxonomic affiliation of one or both taxonomic categories not already represented in the database, add an "*" at the end (no space) of the unrepresented name in input_metadata.tsv to avoid a warning from fihser.py alerting you that the taxonomy is not in metadata.txt. If higher and lower taxonomy are unknown at the time of addition simply use some unique placeholder until proper taxonomic terms can be assigned later. **DO NOT USE TERMS YOU MAY REUSE LATER** such as "unknown" or "new". Failure to change out a term and then reusing it again later for an unrelated organism will cause erratic behavior in fisher.py and forest.py.
     - Example Taxonomic Change: If the term "Chromist" is preferred over "Stramenopiles" replace all instances of "Stramenopiles" in metadata.tsv with "Chromist" before adding new taxa with "Chromist" chosen as the higher taxonomic term in input_metadata.tsv DO NOT MIX THE TWO, erratic behavior will ensue.
  6. Blast Seed (Required) - If the word 'none' is provided the ortholog fishing algorithm will proceed through the default route. If the algorithm is to proceed through the phylogenetically aware route at least one Unique ID of a related organism already in the database must be provided. Use explore_database.py or open metadata.tsv to get a list of all Unique IDs of taxa already present in the provided database. There is no limit to the number of Unique IDs that can be provided here. Unique IDs of taxa must be separated by only a comma with no space in between. **THIS IS THE ONLY PLACE COMMAS MAY BE USED IN INPUT METADATA!**
  7. Long Name (Required) - Full name of the input taxon. **The long name cannot cannot contain underscores "_", at symbols "@", double dots ".."**.
  8. Data Type (Required) - A place to add a note about the type of data being added (ex. transcriptomic, genomic, EST . . .)
  9. Source (Required) - A place to add notes about the source of the data such as accession numbers, "in-house information", strain information etc. If a delimiter is needed here we recommend pipe "|".

Location	File Name	Unique ID	Higher Taxonomy	Lower Taxonomy	Blast Seed	Long Name	Data Type	Source
toadd/	Cuneruss.faa	Cuneruss	Amoebozoa	Discosea	Dictdisc	Cunea russae	Transcriptomic	SRR123

Table 1: Example line of "input_metadata.tsv". Here a proteome predicted from transcriptomic data for *Cunea russae* (Amoebozoa, Discosea) SRA accession SRR123 will be added to the database. The file is located in the directory "toadd/" and is named Cuneruss.faa. This proteome will have the Unique ID "Cunruss." Orthologs in the starting dataset from *Dictyostelium discoideum* (Unique ID=Dictdisc) will be used as queries in the BLAST searches against the *C. russae* predicted proteome.

3. Create and set up the PhyloFisher configuration file.

```
config.py [OPTIONS] -d <path_to_database> -i <path_to_input_metadata.tsv>
```

Required arguments:

-d, --database_folder <database_dir> Path to database directory

-i, --input_file <input.tsv> Path to input_metadata.tsv

Optional arguments:

--orthomcl <omcl_data> Path to orthomcl if not in default location

--tree_colors <tree_colors.tsv> Path to alternative single gene tree color configuration file

-h, --help Show this help message and exit

Default config.py output:

- a file called "config.ini" that contains the paths specified in input.

NOTE: A path to the OrthoMCL diamond database only needs to be provided to config.py via the –orthomcl option if the database is located outside of PhyloFisherDatabase_v1.0/database/orthomcl.

NOTE: By default the downstream python script "forest.py" will use the color configuration file "tree_colors.tsv" located in PhyloFisherDataset_v1.0/database/. If a user would like to create a separate color configuration file for a particular study they should provide the path to the alternate color configuration file to config.py. See STEP 8 for instructions on altering color configuration files.

4. Collect putative homologs from input taxa.

```
fisher.py [OPTIONS]
```

Optional arguments:

-t, --threads <N> Number of threads, where N is an integer

- Default: 1

-n, --max_hits <N> Max number of hits to check, where N is an integer.

- Default: 5

--keep_tmp Keep temporary files.

--all_BBH Keep all significant BLAST hits regardless of phylogenetic affiliation as determined with FastTree through the phylogenetically aware route.

--add <input_metadata> Input metadata (different from original the one in config.ini) that contains new input proteomes. Must be used with –add_to option below.

--add_to <fisher_out> Previous fisher.py output directory to add to. Must be used with the --add option above.

-o, --output <out_dir> Path to user-defined output directory

- Default: ./fisher_out_<M.D.Y>

-h, --help Show this help message and exit.

Default fisher.py output:

- a directory called "fisher_out_<M.D.Y>" that contains:
  - fasta files of orthologs from the database for which putative homologs were collected from newly input taxa and appended to the end of the fasta file. These fasta files make up the working dataset. NOTE: if no putative homologs were found in any newly input proteome for a particular ortholog in the database, that ortholog will not be present in this directory. All files will have the extension ".fas".
  - a tab delimited file called "orgininal_names.tsv" with the original sequence headers from the input proteomes that were clustered by CD-HIT in fisher.py in the first column and a new "cleaner" name assigned by fisher.py made up of the chosen Unique ID and a numbered 1 to N.
- The file "noncorresponding_hits.txt" that contains:
  - information regarding collected sequences whose best BLAST hit was not the corresponding ortholog in the database. (See fisher algorithm details for more information on this criterion). This file will NOT appear if the best BLAST hit of each retained sequence is the corresponding ortholog.

NOTE: If you plan to collect homologs from more taxa prior to single gene tree construction stop here for now.

5. Running more rounds of addition prior to single gene tree construction (OPTIONAL).

```
fisher.py [OPTIONS] --add <Add_More_Taxa.tsv> --add_to <previous_fisher_out_dir>
```

Use –add option in combination with the --add_to option of fisher.py to include homologs from more taxa in another round of addition to the working dataset. Create a new file identical to input_metadata.tsv in structure except with a different name (Ex. Add_More_Taxa.tsv). Put this new file in the project directory created in STEP 1. The information in this new text file will be appended to input_metadata.tsv.

NOTE: We recommend checking that the information contained in the new input file was added to the end of "input_metadata.tsv" after the new addition run is complete. If so, delete the new input file after addition is complete to avoid unnecessary clutter in the directory.

6. Produce preliminary statistics about newly input data.

At this step the statistics provided by informant.py should be considered preliminary. The number of orthologs shown for a newly input taxon can be inflated due to contamination in the data or if fisher.py harvested only a paralog for a particular gene (If only one sequence passes all criteria it is automatically marked the putative ortholog but is not always the true ortholog). These statistics will almost always change after manual curation. However, they can be generated quickly and are useful for a rough estimate of data quality.

```
informant.py [OPTIONS] -i <input_directory>
```

Required arguments:

-i, --input <input_dir> Path to fisher.py output directory

Optional arguments:

--orthologs_only Paralogs will NOT be included from any taxa in the database in downstream single gene tree construction.

-h, --help Show this help message and exit

Default informant.py output (found in the fisher.py output directory provided to informant.py via the -i/--input options):

- A directory called "informant_stats" that contains five files:
  - db_taxa_stats.tsv - a comma separated file with taxa as rows and seven columns:
    1. Unique ID - short name of organism in the database
    2. Long Name - full name of organism in the database
    3. Higher Taxonomy - highest taxonomic level assigned to the taxon in metadata.tsv
    4. Lower Taxonomy - lower taxonomic level assigned to the taxon in metadata.tsv
    5. Genes collected out of "N"- number of genes present in taxa already in the database where N is the number of genes found by fisher.py for all newly added taxa
    6. SGT - will allow for the inclusion (YES) or exclusion (NO) of taxa in single gene tree construction on a per taxon basis
    7. Paralogs - will allow for the inclusion (YES) or exclusion (NO) of paralogs on a per taxon basis for single gene tree construction. If no paralogs exist for an organism in the database this column will be labelled "none". If the --orthologs_only flag was used, columns for all taxa that have paralogs in the database will be marked "NO."
  - genes_stats.tsv - a comma separated file with gene names from the database as rows and four columns. Only genes that had a sequence from at least one newly added taxon appended to the alignment are included.
    1. Gene Name - gene name in the database
    2. Number of Taxa - number of taxa which the gene is present
    3. Percent of Total Taxa (out of "N") - percentage of taxa which have the gene where N is the total number of taxa.
    4. SGT - include gene in single gene trees. To be used by working_dataset_constructor.py downstream.
  - new_taxa_stats.tsv - a comma separated file with taxa as rows and ten columns:
    1. Unique ID - short name of organism in the database
    2. Long Name - full name of organism in the database
    3. Higher Taxonomy - highest taxonomic level assigned to the taxon in metadata.tsv
    4. Lower Taxonomy - lower taxonomic level assigned to the taxon in metadata.tsv
    5. Genes collected out of "N"- number of genes present in newly added taxa where N is the number of genes found by fisher.py for all newly added taxa
    6. Sequences collected - number of total sequences collected by fisher.py for a taxon.
    7. #SBH - number of collected sequences for a taxon where the specific query used produced a significant hit from the input proteome and the collected sequence WAS sister to a sequence already present in the database of the same Higher Taxonomy during FastTree analysis performed automatically by fisher.py
    8. #BBH - number of collected sequences for a taxon where the specific query used produced a significant hit and the collected sequence WAS NOT sister to a sequence already present in the database of the same higher taxonomy during FastTree analysis performed automatically by fisher.py
    9. #HMM - number of collected sequences for a taxon that were selected by the default HMM route.
    10. SGT - will allow for the inclusion (YES) or exclusion (NO) of taxa in single gene tree construction on a per taxon basis

- occupancy.tsv - a comma separated file with taxa as rows and genes in the starting dataset as columns. A "0" (zero) indicates a gene is absent in a taxon. Any non-zero number is the number of sequences collected for that "gene" for a taxon.
- occupancy_with_routes.tsv - a comma separated file with taxa as rows and genes in the starting dataset as columns. A "0" (zero) indicates a gene is absent in a taxon. Any non-zero number is the number of sequences collected for that "gene" for a taxon. The abbreviation for the route through fisher.py that each gene was collected (either SBH, BBH, HMM) by is appended to

7. Collect taxa, and homologs for single gene tree construction.

   NOTE: If you wish to exclude genes, taxa, and/or paralogs from the working dataset used for single gene tree construction (NOT RECOMMENDED) change the column values for "Include in Single Gene Tree Construction" in gene_stats.tsv and/or org_stats.tsv respectively to "NO" and then run:

   ```
   working_dataset_constructor.py [OPTIONS] -i <input_directory>
   ```

   Required arguments:

   -i, --input <input_dir> Path to output directory of fisher.py that contains the output of informant.py

   Optional arguments:

   -o, --output <out_dir> Path to user-defined output directory
   - Default: ./working_dataset_constructor_out_<M.D.Y>

   -h, --help Show this help message and exit.

   NOTE: The <input_dir> should be the directory created by fisher.py "fisher_out_<M.D.Y>" which contains the gene files to be subset and the output directory created by informant.py.

   Default working_dataset_constructor.py output:

   - a directory "working_dataset_constructor_output_<M.D.Y>" that contains:
     - fasta files of genes to be included in single gene tree construction. This will be either all or the chosen subset of the gene files contained in the input directory. Each gene file contains only taxa to be included in single gene tree construction.

8. Filter align and trim single gene files followed by optional single gene tree construction

   Trees should be constructed from the files located in the output directory of working_dataset_constructor.py.

   NOTE: If you would like to perform filtering, alignment, and trimming but no single gene tree construction via RAxML use the --no_tree flag.

   NOTE: If you choose to build single gene trees without using "sgt_constructor.py" you must use a maximum likelihood program. Downstream quality control steps are not set up to interpret Bayesian posterior probability values. Your naming convention for each maximum likelihood single gene tree must follow {nameofyourchoosing}.{gene_name}.tre. "nameofyourchoosing" can be anything you wish but it CANNOT contain a period "." in it.

   To do this run:

   ```
   sgt_constructor.py [OPTIONS] -i <input_directory>
   ```

   Required arguments:

   -i, --input <input_dir> Path to output of working_dataset_constructor.py

Optional arguments:

-t, --threads <N> Number of threads
- Default: 1

--no_trees Do not build single gene trees.

--trees_only <in_dir> Only build single gene trees. No other operations performed.

-o, --output <out_dir> Path to user-defined output directory
- Default: ./sgt_constructor_out_<M.D.Y>

-h, --help Show this help message and exit

Default sgt_constructor.py output:

- a directory "prequal" that contains the files:
  - {gene_name}.aa - unaligned gene file used as PREQUAL input
  - {gene_name}.aa.filtered - output of PREQUAL. Used as input for MAFFT in subsequent length filtration step.
  - {gene_name}.aa.filtered.PP - output of PREQUAL.##
  - {gene_name}.aa.warning - output of PREQUAL.##
- a directory "length_filtration" that contains:
  - a directory "mafft" that contains the file:
    - {gene_name}.aln - MAFFT output. Aligned gene file in fasta format. Used as input for Divvier.
  - a directory "divvier" that contains the files:
    - {gene_name}.aln.divvy.fas - output of Divvier.
    - {gene_name}.aln.PP - output of Divvier.$$
  - a directory "BMGE" that contains the files:
    - {gene_name}.pre_bmge - modified version of "gene_name.aln.divvy.fas" with the character "X" replaced by the character "-". Used as input for BMGE.
    - {gene_name}.bmge - output of BMGE. Used as input for the length filtration step.
    - {gene_name}.length_filtered - length filtered fasta file used as input for a second run of MAFFT.
- a directory "mafft" that contains the file:
  - {gene_name}.aln2 - output of the second run of MAFFT and input for a second run of Divvier.
- a directory "divvier" that contains the files:
  - {gene_name}.aln2.divvy.fas - output of the second run of Divvier and input for timAl.
  - {gene_name}.aln2.PP - output of Divvier.$$.
- a directory "trimAl" that contains the files:
  - {gene_name}.final - output of trimAl. Trimmed alignment in fasta format. Used as input for RAxML.
  - {gene_name}.final.reduced - output of trimAl. Trimmed alignment in phylip format.
  - a directory "RAxML" that contains the files:
    - RAxML_bootstrap.{gene_name}.tre&&
    - RAxML_bestTree.{gene_name}.tre&&
    - RAxML_bipartitionsBranchLabels.{gene_name}.tre&&
    - RAxML_bipartitions.{gene_name}.tre&&
    - RAxML_info.{gene_name}.tre&&
  - a directory "trees" that contains:
    - RAxML_bipartitions.gene_name.tre - copies of from the "RAxML" directory. Used as the input for forest.py.

        * {gene_name}.final - the alignments that produced each tree in fasta format named copied from the "trimAl" directory. Used as the input for forest.py.
        * {gene_name}.trimmed - the trimmed alignments used in length filtration copied from the BMGE directory. Used as the input for forest.py.
- a directory "logs" that contains:
    - a directory prequal the file:
        * gene.log - the log file for gene run through trimAl.
    - a directory length_filter_mafft file:
        * gene.log - the log file for gene run through the MAFFT step of length filtration.
    - a directory length_filter_divvier file:
        * gene.log - the log file for gene run through Divvier step of length filtration.
    - a directory x_to_dash file:
        * gene.log - the log file for gene run through removal of X's from gene files, as part of length filtration.
    - a directory length_filter_bmge file:
        * gene.log - the log file for gene run through BMGE step of length filtration.
    - a directory length_filtration file:
        * gene.log - the log file for the length filtration of gene.
    - a directory mafft file:
        * gene.log - the log file for gene run through MAFFT.
    - a directory divvier file:
        * gene.log - the log file for gene run through Divvier.
    - a directory trimal file:
        * gene.log - the log file for gene run through trimAL
    - a directory raxml file:
        * gene.log - the log file for gene run through RAxML
    - a file tar_local_dir.log - the log file for the compression of the local output directory, sgt_constructor_out_<M.D.Y>-local.

- These are standard PREQUAL output files for each gene that PhyloFisher has appended the corresponding gene name to. See the PREQUAL documentation for a thorough explanation of their contents.

$$ - These are standard Divvier output files for each gene that PhyloFisher has appended the corresponding gene name to. See the Divvier documentation for a thorough explanation of their contents.

&& - These are standard RAxML output files for each gene that PhyloFisher has appended the corresponding gene name to. See the RAxML documentation for a thorough explanation of their contents.

NOTE: For a detailed explanation of the methodology implemented in "sgt_constructor.py" see Automated Filtering, Alignment, Trimming, and Single Gene Tree Construction.

NOTE: A file sgt_constructor_out_<M.D.Y>.tar.gz is also created by sgt_constructor.py. This file is compressed file that contains the trees directory outlined above along with the metadata.tsv, input_metadata.tsv, and tree_colors files. This is created to ease the movement of all required data over to a local machine to render the svg files used in the next step. This is often necessary due to the lack of graphics capabilities of headless servers.

NOTE: If sgt_constructor.py dies in the middle of a run, simply provide the sgt_constructor output directory from the previous run to sgt_constructor.py via the -o flag in addition to the previous command and the script will pick up where it left off.

9. Render .svg and .tsv files of single gene trees for visualization with ParaSorter.

   If working on a remote server, copy the file sgt_constructor_out_<M.D.Y>.tar.gz file to your local machine for input into forest.py.

```
forest.py [OPTIONS] -i <input>
```

   Required arguments:

   -i, --input <input_dir> Path to sgt_constructor_out_<M.D.Y>/trees if run remotely or sgt_constructor_out_<M.D.Y>.tar.gz if run locally

   Optional arguments:

   -a, --contaminants <contams.tsv> Path to file containing known contaminants to be removed Table 2.

   -b, --backpropagate Path to file containing known contaminants to be backpropagated Table 2.

   -t, --threads <N> Number of threads

   - Default: 1

   -o, --output <out_dir> Path to user-defined output directory

   - Default: ./forest_out_<M.D.Y>

   --local_run To be used when sgt_constructor_out.tar.gz has been downloaded from a server for single gene tree visualizations to be rendered locally.

   -h, --help Show this help message and exit.

   Default forest.py output:

   - a directory "/forest_out_<M.D.Y>" that contains:
     - an svg file for each single gene tree
     - a tsv file for each single gene tree with the status of each sequence assigned by fisher.py and, if the contaminant or contaminant+backpropogate flags were invoked, forest.py. Each sequence is marked either an ortholog "O", a paralog "P", or delete "D"

   If a user has *a priori* knowledge of eukaryotic contamination in their data they can provide forest.py with a file via the --contaminants flag. The file should be tab delimited and have three columns. In the first column is the Unique ID of the contaminated input proteome in the second column taxonomic term of the contaminant. Any of the three taxonomic levels used in PhyloFisher (higher taxonomy, lower taxonomy, Unique ID) can be used to identify contamination. Finally in the third column is the name of the taxonomic level chosen for the contaminant Table 2. When this file is provided, forest.py will pre-mark instances of this branching pattern for deletion regardless of MLBS for the relationship.

Dermalge	Chloroplastida	Higher Taxonomy

Table 2: Example input file to premark or backpropagate known eukaryotic contaminants. Here *Dermamoeba algensis* (Unique ID = Dermalge) is contaminated with a eukaryote that is a member of Cholorplastida. Cholorplastida's rank in PhyloFisher's default metadata file is "Higher Taxonomy".

   If during manual curation of single gene trees a user discovers previously unknown pervasive contamination, rather than having to manually mark each instance for deletion, the user can provide forest.py with a file in the same format as above only with both the --contaminations and --backpropagate flags given. When the --backpropagate is added all decisions made in previously

manually curated gene trees will be maintained rather than rewriting the original .tsv files with the only difference being the defined contaminant is marked for deletion.

NOTE: Only sequences from newly added organisms can be pre-marked for contamination or have contamination back propagated. Contamination from previously added organisms must be marked manually if found after their initial addition.

The file "tree_colors.tsv" is used by "forest.py" to color taxa by taxonomic rank when rendering .svg files of single gene trees for visualization via the included tool ParaSorter. A default tree_colors.tsv is supplied with the database. The default file is configured to have all taxa in the database colored by the higher taxonomic rank assigned to them in metadata.tsv Table 3.

Taxonomy	Color
Amoebozoa	Red
Ancyromonadida	LightSalmon
CRuMs	Salmon
Discoba	Gold
Malawimonadida	Orange
Metamonada	DarkViolet
Obazoa	Crimson
Rhizaria	Magenta
Rhodophyta	MediumSpringGreen
Stramenopiles	MediumSlateBlue
Haptista	MediumSeaGreen
Cryptista	Olive
Hemimastigophora	LavenderBlush
Telonemia	MistyRose
Chloroplastida	GreenYellow
Alveolata	DeepSkyBlue
Glaucophyta	LimeGreen
Rhodelphidia	SpringGreen
Alveida	Khaki
Picozoa	Lime

Table 3: Default tree_colors.tsv file. Color and the taxonomic rank that taxa are colored by can be changed by the user.

The color and the taxonomic rank that taxa are colored by can be changed by the user in the default file or users can provide config.py with an alternative tree_colors.tsv file. Taxa can be colored by any of the three possible taxonomic designations used in PhyloFisher; higher taxonomy, lower taxonomy, or by the taxon's Unique ID. A lower taxonomic rank and the color assigned to it will take precedence over the higher taxonomic rank and its associated color that the lower taxonomic rank is a part of.

For example, *Dermamoeba algensis* is assigned the higher taxonomic rank Amoebozoa, the lower taxonomic rank Discoba, and has the Unique ID "Dermalge" in metadata.tsv. By default, all members of Amoebozoa are colored red. However, if a user were to add "Discoba Brown" to "tree_colors.tsv" then all members of Discoba will be colored brown while all members of the other two sub lineages of Amoebozoa will remain red. If a user also added "Dermalge Teal" to "tree_colors.tsv" all members of Discoba will be colored brown except *D. algensis* which will be colored teal while all other amoebozoans will remain red.

If a taxon was added that has a taxonomy not previously represented in the file metadata.tsv users will need to add one of the taxonomic ranks they assigned to the taxon in "input_metadata.tsv" or the organism's Unique ID for the taxon to be colored by and a corresponding color to tree_colors.tsv. Users need to do this after the taxon has been added to the database so during subsequent rounds of addition and single gene tree inspection the new taxon will be colored accordingly.Example colors can be found here.

10. Manual inspection of single gene trees.

    To manually inspect a single gene tree run:

    `ParaSorter`

    Once ParaSorter has opened:

    In the upper left corner click "Open tree"
    Choose an svg file created by forest.py in the last step.
    Click "Import tsv" and choose the corresponding gene's tsv file

    To the right of each sequence in the single gene tree displayed there are three boxes Figure 4. If selected, the leftmost box (green in color) will display a capital letter "O" for "ortholog", the middle box (yellow in color) will display a capital letter "P" for "paralog", and the rightmost box (red in color) will display a capital letter "D" for "delete." These boxes are used to display and change a sequence's designation in the database. To change a sequences designation simply click the box that corresponds to the desired assignment. These designations will be applied to the starting database in the next step.

    Information provided in sequence headers (phylogenetic tree leaves) and single gene tree files to better inform ortholog, paralog, and contamination selection during visual inspection Figure 4:

    - Full genus and species name of taxon
    - Route through fisher.py that produced sequence:
      - Specific Best Hit (SBH) – appended if a specific query produced a significant hit and the sequence **WAS** sister to a sequence within the database of the same taxonomy during FastTree analysis performed automatically by fisher.py
      - Best Blast Hit (BBH) - appended if a specific query produced a significant hit and the sequence **WAS NOT** sister to a sequence within the database of the same taxonomy during FastTree analysis performed automatically by fisher.py
      - HMMER route (HMM) – appended if the sequence was selected by the default route using hmmer search
    - Priority in final set of collected sequences:
      - Denoted "qn" where "n" is an integer representing the sequence's order of priority out of "n" sequences collected in the initial search using the profile HMMs (ex. q1, q2 . . ., qn).
    - If the collected sequence's best BLAST hit is or is not a sequence in the corresponding ortholog alignment from the database.
      - c - collected sequence's best BLAST hit **IS** a sequence in the corresponding ortholog alignment from the database.
      - n - collected sequence's best BLAST hit **IS NOT** a sequence in the corresponding ortholog alignment from the database.
    - Proportional length of the sequence in the trimmed alignment used for length filtering
    - Short name of taxon in dataset
    - Broad and fine scale taxonomic designation
    - Other notes:

- Displayed at the top of each tree is the name of the gene, the length of trimmed alignment used to filter sequences, the length of the alignment used to produce the tree and the number of "suspicious clades" detected.
  - A clade is marked as suspicious if it is supported by a maximum likelihood bootstrap value of 70 or greater and contains sequences of mixed higher taxonomy. These clades are highlighted with a grey background and each node that meets the above criteria is marked with a red sphere.
- Newly added sequences that were chosen as the putative ortholog by fisher.py are written in bold black font while sequences chosen as paralogs are written in regular black font.
- Sequences that are newly added are not highlighted in color based on taxonomy.
- Orthologs already in the database are highlighted in color based on taxonomy and are written in regular black font.
- Paralogs already in the database are not highlighted in color based on taxonomy and are written in regular blue font.
- Sequences pre-marked for deletion are not highlighted in color based on taxonomy and are written in regular red font. Only present if known contaminants were provided to forest.py via the contaminants and backpropagate flags.

See Figure 4 for an example ParaSorter display and sequence header with explanation:

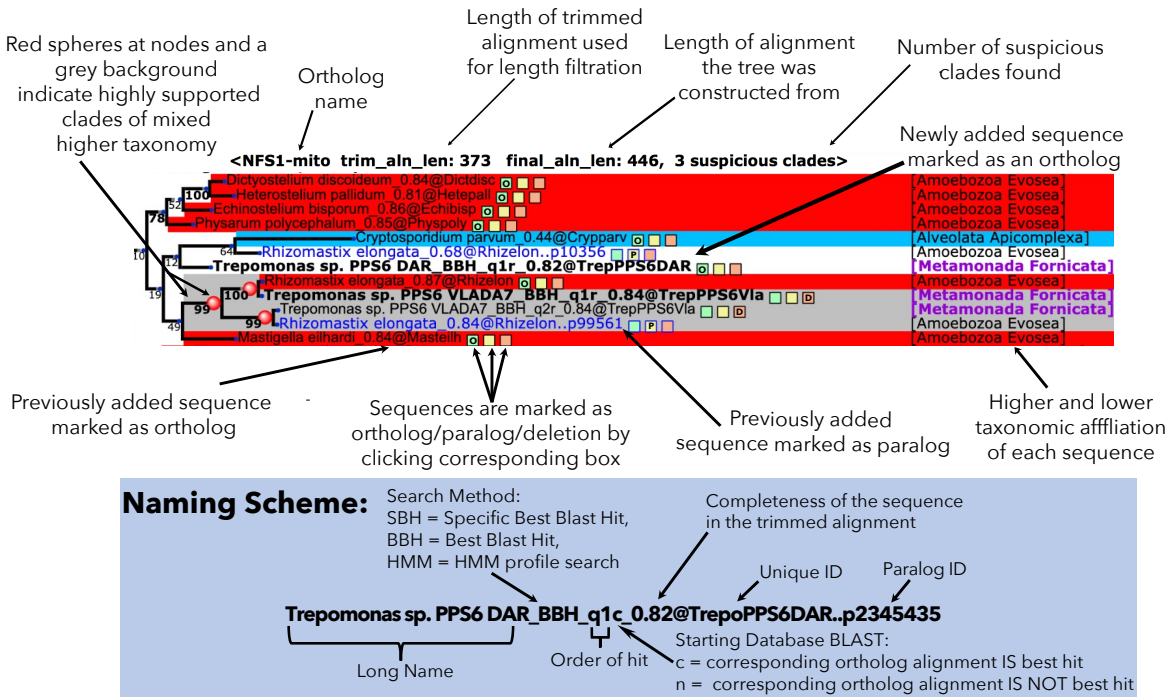

Figure 4: Example and explanation of a single gene tree visualized with ParaSorter (above) and of the PhyloFisher sequence header naming scheme (below).

After all decisions have been made click "Save to tsv." The default name ParaSorter will suggest for the new file will be {gene_name}_parsed.tsv. The file must have "{gene_name}_parsed.tsv" naming convention for subsequent steps of the workflow to perform correctly.

11. To backup the database, add new taxa to the database, and implement all decisions made during manual curation with ParaSorter run:

```
apply_to_db.py [OPTIONS] -i <input_directory> -fi <fisher_output_dir>
```

Required arguments:

-i, --input <in_dir> Path to input directory containing parsed tsv labeled {gene_name}_parsed.tsv

-fi, --fisher_dir <fisher_output_dir> Path to initial fisher.py output directory

Optional arguments:

-t, --threads <N> Number of threads

--to_exclude to_exclude.txt Path to .txt file containing Unique IDs of taxa to exclude from dataset addition with one taxon per line

- Example: Unique ID1
  Unique ID2

-h, --help Show this help message and exit.

apply_to_db.py will first backup the current database/ directory and place the backup in PhyloFisherDatabase_v1.0/database/backups. Next, apply_to_db.py will append orthologs from input taxa to their corresponding gene files located in /database/orthologs/, append paralogs for input taxa to their corresponding gene files located in /database/paralogs/, and remove sequences marked for deletion. apply_to_db.py will update the binary matrix file "db_ortholog_occupancy.tsv" for newly added taxa based on decisions made during manual curation. The file "metadata.tsv" will also be updated with information provided for newly added taxa in "input_metadata.tsv." Next, apply_to_db.py will rebuild the database so sequences from newly added taxa are present in the profile HMMs, the diamond database. This will allow orthologs from the newly added taxa to be used as specific queries in subsequent runs of addition. Finally, apply_to_db.py will copy the input proteome of all new taxa that are added to the database to /database/proteomes/.

12. Select a subset of taxa to include in a phylogenomic dataset (OPTIONAL)

In order to select taxa to be included in the final phylogenomic dataset users can run the script "select_taxa.py". This will generate a .tsv file (contents explained below) that will serve to select taxa to include and is used as input for prep_final_dataset.py downstream.

NOTE: If all taxa are to be used in the final phylogenomic analyses this step is not necessary. Skip to step 13 (ALSO OPTIONAL) (to select orthologs to include in the final phylogenomic analyses) or directly to step 14 (if all taxa and all orthologs are to be used in the final phylogenomic analyses).

To select taxa for the final phylogenomic analyses run:

```
select_taxa.py [OPTIONS]
```

Optional arguments:

--to_exclude taxa_to_exclude.txt A text file of groups or individual taxa to exclude from the final phylogenomic matrix

--to_include taxa_to_include.txt a text file that will re-include groups or individual taxa.

--chimeras chimeras.tsv A .tsv containing a Unique ID, higher and lower taxonomic designations, and the Unique IDs of the taxa to collapse, for each chimera one per line.

-h, --help Show this help message and exit

NOTE: Detailed explanation of to the --to_exclude and --to_include options and their input files:

By default all taxa will be included. These options were designed to decrease the amount of manual manipulation of "taxa_subset.tsv" generated in this first run of "select_taxa.py." Manual manipulation will be necessary if many taxa from the database are to be excluded from the final phylogenomic analysis. Both options take a file created by the user that has one column with an organism's Unique ID or a taxonomic rank (higher or lower) from the metadata to be excluded or included in downstream steps. The options can be used individually or in conjunction with one another to implement decisions on taxa selection in an automated fashion. For example, if all taxa in the eukaryotic assemblage "Amoebozoa" are to be excluded from downstream phylogenomic analyses provide the --exclude_taxa option with a file (named anything) that contains "Amoebozoa" as the first entry of the column. This will result in all amoebozoans being marked as "no" in the "Include in Subset" column of "taxa_subset.tsv" when it is generated. If a user wanted all amoebozoans excluded except *Dictyostelium discoideum* (Unique ID = Dictdisc) provide the --exclude_taxa option with a file (named anything) that contains "Amoebozoa" as the first entry of the column as before and the --to_include option with a file (named anything) with "Dictdisc" as the first entry of the column. This will result in all amoebozoans being marked as "no" in the "Include in Subset" column of "taxa_subset.tsv" except *D. discoideum* which will be marked "yes."

NOTE: --to_include is not necessary it only serves to ease the burden of manual taxa selection in cases as outlined above. All taxa are marked to include by default so it is NOT necessary to provide --to_include with a list of all taxa you wish to include in downstream steps.

NOTE) Decisions provided to --to_include will override decisions provided to --to_exclude so be cautious.

Alternatively, all changes can be made manually by opening "select_taxa.tsv" and changing taxa designations from "yes" to "no."

NOTE) An example input file for the –chimeras option of select_taxa.py is given below in Table 4 below.

BrevCHIM	Obazoa	Breviatea	Brevanat	Lenilimo	Pygsbifo

Table 4: Example input .tsv file for the --chimera option of select_taxa.py. Here a chimera will be made from three breviates in the database. The chimera will have the Unique ID "BrevCHIM", the higher taxonomy "Obazoa", the lower taxonomy "Breviatea" and will be made from Brevanat (*Breviata anathema*), Lenilimo (*Lenisia limosa*), Pygsbifo (*Pygsuia biforma*).

Default select_taxa.py output:

- a .tsv file "select_taxa.tsv" with five columns:
  1. Unique ID
  2. Higher Taxonomy
  3. Lower Taxonomy
  4. Completeness (percentage of orthologs) from the database present in the taxon
  5. Include in Subset
- a plot showing completeness (number of orthologs) in each taxon. Groups of taxa are colored by level of completeness in increments of 10% (Figure 5).

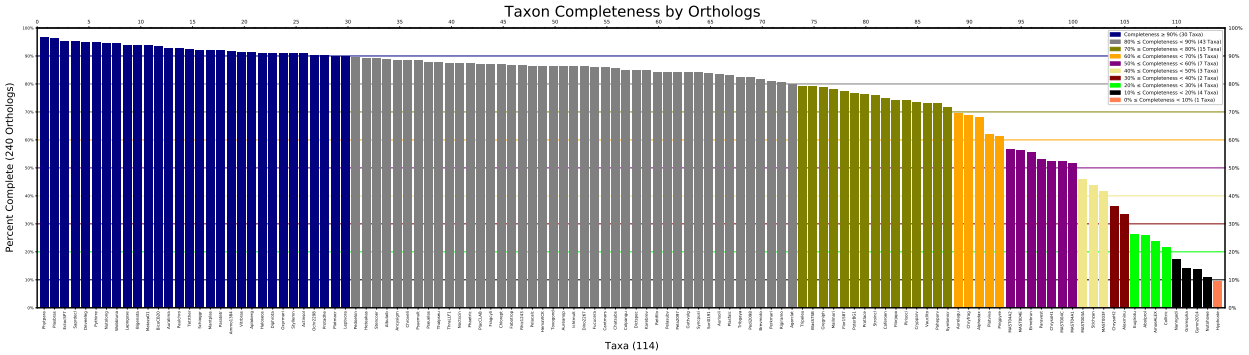

Figure 5: Plot produced by select_taxa.py. Taxa Unique IDs are shown at the bottom of the figure across the x-axis. The percentage of total orthologs present in a taxon (completeness) is shown on the y-axis. The total number of taxa is shown across the top of the figure on the x-axis. The bars are colored by level of completeness in increments of 10%. Legend explaining color coding is shown in the top right of the figure.

13. Select orthologs to be included in the final phylogenomic dataset (OPTIONAL).

    NOTE: If all orthologs are to be included in the final phylogenomic dataset this step is not necessary. Skip to step 14.

    To select orthologs to be included in the final phylogenomic dataset run:

    ```
    select_orthologs.py [OPTIONS]
    ```

    Optional arguments:

    --out_group out_group.txt Path to text file containing out group taxa Unique IDs.
    -n, --gene_number <N> Number of genes for analysis
    -c, --percent_complete <N> Threshold for percent completeness
    -h, --help Show this help message and exit

    select_orthologs.py output:

    - a tsv file "select_orthologs.tsv" with three either three or five columns:
      1. Ortholog - ortholog name
      2. Total Completeness - percentage of all taxa to be included in the final phylogenomic dataset the ortholog is present in.
      3. In-Group Completeness (only present if --out_group option is utilized) -percentage of In-group taxa only to be included in the final phylogenomic dataset the ortholog is present in.
      4. Out-Group Completeness (only present if --out_group option is utilized) - percentage of Out-Group taxa only to be included in the final phylogenomic dataset the ortholog is present in.
      5. Include in Subset
    - a plot showing completeness (number of taxa) of each gene. Groups of genes are colored by level of completeness in increments of 10% (Figure 6).

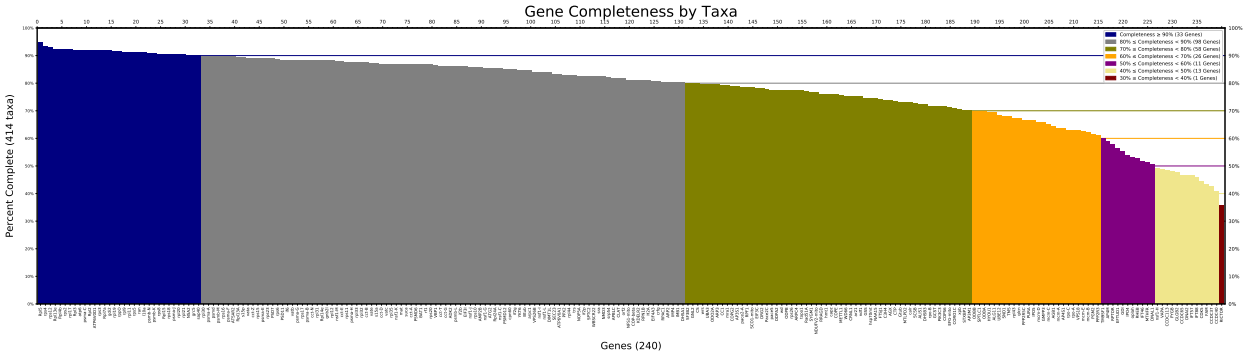

Figure 6: Plot produced by select_orthologs.py. Ortholog names are shown at the bottom of the figure across the x-axis. The percentage of total taxa present in an ortholog (completeness) is shown on the y-axis. The total number of genes is shown across the top of the figure on the x-axis. The bars are colored by level of completeness in increments of 10%. Legend explaining color coding is shown in the top right of the figure.

14. Collect orthologs and taxa to be included in the final phylogenomic dataset

    To do this run:

    ```
    prep_final_dataset.py [OPTIONS]
    ```

    Optional arguments:

    -o, --output <out_dir> Path to user-defined output directory
    - default ./prep_final_dataset_<M.D.Y>

    -h, --help Show this help message and exit

    Default prep_final_dataset.py output:

    - a directory called prep_final_dataset_<M.D.Y> that contains:
      - a fasta file of each ortholog selected to be in the final phylogenomic dataset each containing only taxa selected to be in the final phylogenomic dataset.

15. Trim, align, and concatenate orthologs into a super-matrix.

    To do this run:

    ```
    matrix_constructor.py [OPTIONS] -i <input_directory>
    ```

    Required arguments:

    -i, --input <in_dir> Path to prep_final_dataset_<M.D.Y>

    Optional arguments:

    -f, --out_format <format> Desired format of the output matrix
    - Options: fasta, phylip, phylip-relaxed, or nexus.
    - Default: fasta

    -if, --in_format <format> format of the input files.
    - Options: fasta, phylip, phylip-relaxed, or nexus.
    - Default: fasta

    -c, --concatenation_only Only concatenate alignments. Filtering, alignment, and trimming are not performed automatically.

-t, --threads <N> Desired number of threads to be utilized.

- Default: 1

-o, --output <out_dir> Path to user-defined output directory

- Default: ./matrix_constructor_out_<M.D.Y>

-p, --prefix <prefix> Prefix of input files

- Default: NONE
- Example: path/to/input/prefix*

-s, --suffix <suffix> Suffix of input files

- Default: NONE
- Example: path/to/input/*suffix

-h, --help Show this help message and exit

Default matrix_constructor.py output:

- a directory called "matrix_constructor_out_<M.D.Y>" containing:
  - a directory "prequal" that contains:
    - {gene_name}.aa - unaligned gene file used as PREQUAL input
    - {gene_name}.aa.filtered - output of PREQUAL. Used as input for MAFFT in subsequent length filtration step.
    - {gene_name}.aa.filtered.PP - output of PREQUAL.##
    - {gene_name}.aa.warning - output of PREQUAL.##
  - a directory "mafft" that contains:
    - {gene_name}.aln - output of of MAFFT and input for Divvier.
    - {gene_name}.aln.PP - output of PREQUAL.##
  - a directory "divvier" that contains:
    - {gene_name}.aln.partial.fas - output of Divvier and input for timAl.
  - a directory "trimAl" that contains:
    - {gene_name}.gt80trimAl.fas - output of trimAl. Trimmed alignments, in FASTA format, that will be used for concatenation.
  - indices.tsv - a tab separated file with three columns outlining the single gene boundaries in the supermatrix:
    1. Gene - name of the gene
    2. Start - first position of the gene within the super matrix
    3. Stop - last position of the gene within the super matrix
  - matrix.<fas | nex | phy> - the concatenated super matrix of all genes in the provided input directory in the specified file format.
  - matrix_constructor_stats.tsv - a tab separated file with two columns:
    1. Taxon -Unique ID of taxon in the database
    2. Percent Missing Data - percentage of unoccupied sites within the concatenated matrix.

- These are standard PREQUAL output files for each gene that PhyloFisher has appended the corresponding gene name to. See the PREQUAL documentation for a thorough explanation of their contents.

UTILITIES

Overview

While the above processes will allow users to develop a phylogenomic dataset and infer single gene and phylogenomic trees using whatever best practices are available to them, many more analyses are often used to examine the effects of aspects of the data. Phylogenomic analyses are often accompanied by additional analyses in which users manipulate their phylogenomic dataset or examine the dataset in a different fashion to reveal artifacts in their data. Here we provide users with simple programs to perform some of the most often employed tactics.

aa_comp_calculator.py: Calculates amino acid composition and uses euclidean distances to hierarchically cluster these data, in order to examine if amino acid composition may bias the groupings that were inferred in a phylogenomic tree. See Brown et al., 2018 for an example.

astral_runner.py: Generates input files and infers a coalescent-based species tree given a set of single ortholog trees and bootstrap trees using ASTRAL-III (Zhang et al., 2018).

backup_restoration.py: Restores a previous version of the database from the directory backups/.

bipartition_examiner.py: Calculates the observed occurrences of clades of interest in bootstrap trees.

build_database.py: used to format custom databases for use in the PhyloFisher workflow. This utility is also used to rename taxa in either the provided database or a custom database.

explore_database.py: Used to examine the composition of the database using taxonomic terms as search queries.

fast_site_remover.py: The fastest evolving sites are expected to be the most prone to phylogenetic signal saturation and systematic model misspecification in phylogenomic analyses. This tool will remove the fastest evolving sites within the phylogenomic supermatrix in a stepwise fashion, leading to a user defined set of new matrices with these sites removed.

fast_taxa_remover.py: Removes the fastest evolving taxa, based on branch length. This tool will remove the fastest evolving taxa within the phylogenomic supermatrix in a stepwise fashion, leading to a user defined set of new matrices with these taxa removed.

genetic_code_examiner.py: Checks stop-to-sense and sense-to-sense codon reassignment signal in transcriptome/genome data. This script is using the advantage of the phylogenetically broad database accompanying our software. The first step is the creation of multiple sequence alignments from all fasta files containing manually curated orthologous sequences. This step can take some time but it is necessary only for the first run of genetic_code.py. In the next step, tblastn searches with orthologs from selected related organism/s in the database are performed against the given transcriptome/genome (with 1e-30 default e-value). For all genes, the best scoring hit is investigated for "good quality positions". Such positions are located at least 6 amino acids from the beginning or end of the blast alignment and the number of low scoring mismatches (normally denoted by spaces in blast middle line) in close proximity to these positions (+- 3 amino acids) is less than 3. Corresponding positions from the query are then analyzed in previously prepared multiple sequence alignments and information about well-conserved amino acids (in more than 70% of organisms, default) is collected and connected to the underlying codon from transcriptome/genome. Codons which show evidence for signals different from the standard genetic code signal are then visualized in the form of bar plots (occurrence of conserved amino acids). Thanks to the evolutionary well-conserved nature of proteins in our database, realigning all sequences again with provided input nucleotide data is not necessary. This script performs well with genomic and transcriptomic data. Analysis of one transcriptome/genome should usually take less than 5 minutes on an average personal computer. It has to be mentioned that alternative genetic code signal from multiple sequence alignments is only one way to analyze this phenomenon and tRNAs should be investigated as well if possible.

heterotachy.py: Within-site rate variation (heterotachy) (Lopez et al., 2002) has been shown to cause artificial relationships in molecular phylogenetic reconstruction (Inagaki et al., 2004). This tool will remove the most heterotachious sites within a phylogenomic supermatrix in a stepwise fashion, leading to a user defined set of new matrices with these sites removed.

mammal_modeler.py: Generates a MAMMaL site heterogeneous model from a user input tree and supermatrix with estimated frequencies for a user defined number of classes. This program creates a set of temporary files from the user provided input that MAMMaL is able to handle. The output is a heterogeneous model in Nexus format that is usable in IQtree using options (-m LG+ESmodel+G -mdef esmodel.nex). This program outputs by default a 61 class mixture model with 60 site frequency classes and the overall amino acid frequencies (+F) of the phylogenomic dataset. Also please note that when using this program we hard-code the "not using likelihood weighting" option. This is because using likelihood weighting will cause issues in the calculation of likelihood weights in sparse phylogenomic matrices. The problem is that there may be pairs of sequences that have no sites in common. Specifically, the proportion of times an amino acid occurs for the pair of sequences becomes NA because the denominator is 0 (calculation is [p_{aa;sj}] in Eqn (5) of the Susko et al., 2018) (Ed Susko personal communication).

purge.py: This tool is used for deleting taxa and/or taxonomic groups from the database and metadata permanently.

random_resampler.py: This tool randomly resamples the gene set into a set of new matrices that are subsamples of the super matrix. It constructs supermatrices from randomly sampled genes with user defined options such as the confidence interval sampling all genes in a random fashion and the percentage of subsampling a user requires per sampling step. This method was used in Brown et al., 2018 as an example.

rtc_binner.py: Calculates the relative tree certainty score (RTC) in RAxML Stamatakis, 2014 of each single ortholog tree and bins them based on their RTC scoring into top 25%, 50%, and top 75% sets. Supermatrices are constructed from these bins of orthologs.

SR4_class_recoder.py: To minimize phylogenetic saturation this tool recodes input supermatrix into the four-character state scheme of SR4 (Susko and Roger, 2007), based on amino acid classification.

taxon_collapser.py: Allows users to combine multiple operational taxonomic units into one single taxon. For example if a user has multiple single cell libraries from a taxon or multiple strains of the same species (or genus etc.), a user may decide to collapse all these strains/libraries into a single taxon.

Example Usages of Utilities:

Calculate the amino acid composition of an input matrix.

```
aa_comp_calculator.py [OPTIONS] -i <input_matrix>
```

Required arguments:

-i, --input matrix Path to input matrix for analysis

Optional arguments:

-o, --output <out_dir> Path to user-defined output directory

- Default: ./aa_comp_calculator_out_<M.D.Y>

-h, --help Show this help message and exit

Default aa_comp_calculator.py output:

- a directory “aa_comp_calculator_output_<M.D.Y>” that contains:
  - a tab separated file “aa_comp.tsv” with 21 columns:
    1. “Taxon” - short name of taxon in dataset
    2. Each of the following 20 columns are labeled with the IUPAC single letter abbreviation for each standard amino acid. The values in rows are the proportion of the corresponding amino acid in the sequence data for a taxon in the input file.
  - AA_Composition_Hierarchical_Clustering.pdf” - a dendrogram built using pairwise distances calculated by Ward’s criterion between taxa after hierarchical clustering based on amino acid composition in pdf format Figure 7.

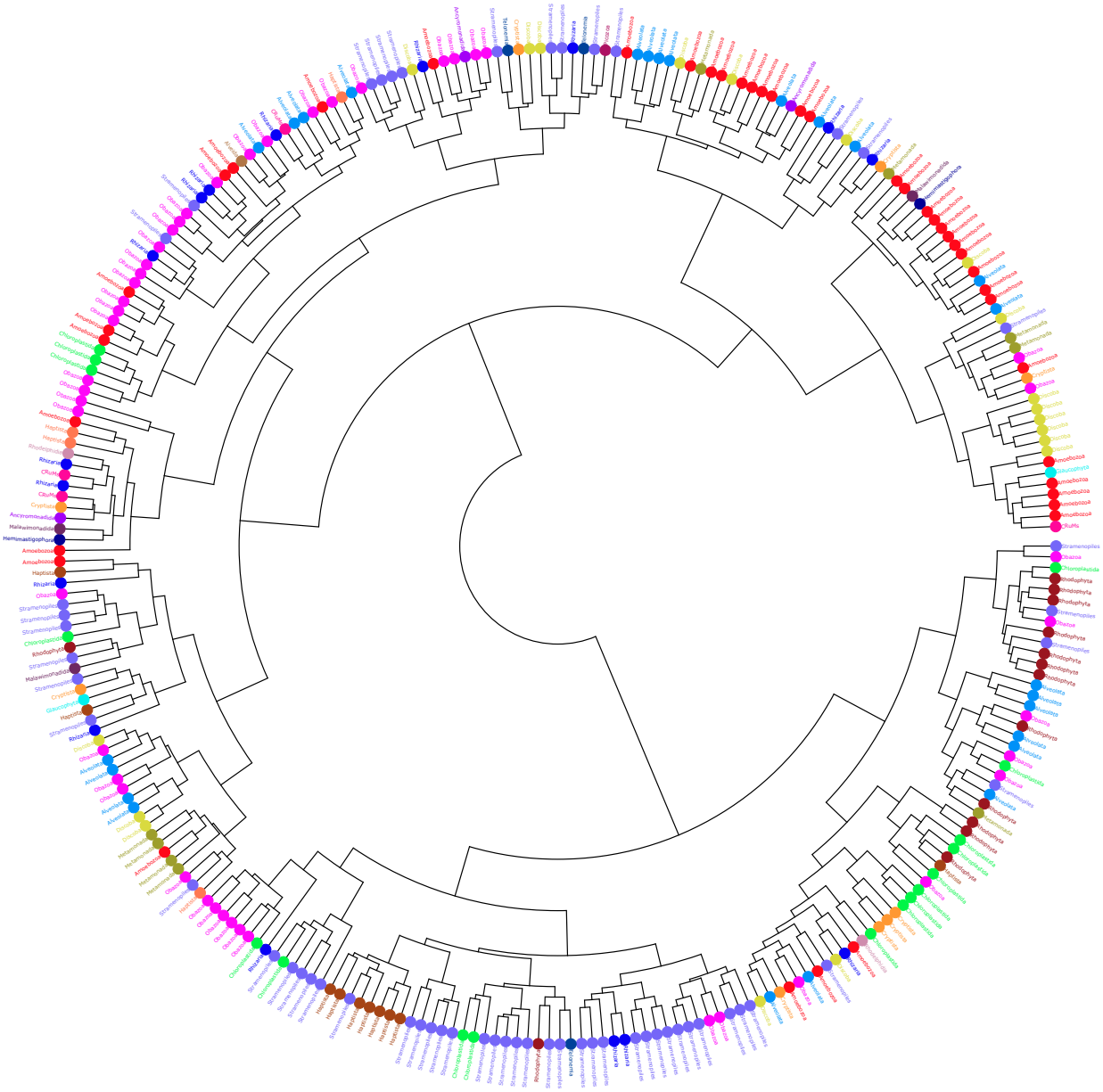

Figure 7: Plot produced by aa_comp_calculator. The plot is built using pairwise distances calculated by Ward's criterion between taxa after hierarchical clustering based on amino acid composition as performed in Brown et al., 2018. Leaves are labeled and colored according to higher taxonomy of taxa in the provided alignment.

Generate input files and infer a coalescent-based species tree with ASTRAL-III.

```
astral_runner.py [OPTIONS] -i <input_directories>
```

Required arguments:

-i, --input <input_directory> Path to directory containing single gene trees and their corresponding bootstrap value files.

Optional arguments:

-o, --output <out_dir> Path to user-defined output directory
- Default: ./astral_runner_<M.D.Y>

-p, --prefix <prefix> Prefix of input files
- Default: NONE
- Example: path/to/input/prefix*

-s, --suffix <suffix> Suffix of input files
- Default: NONE
- Example: path/to/input/*suffix

-h, --help Show this help message and exit.

Default astral_runner.py output:

- a file all_sgt.tre that contains:
  - all single gene trees provided in Newick format
- a file bs_files.txt that contains:
  - paths to all ML bootstrap files
- a file astral_BS.out that contains:
  - 100 bootstrapped replicate trees
  - A greedy consensus of the 100 bootstrapped replicate trees
  - The "main" ASTRAL tree

NOTE: Users may choose to use the trees generated by sgt_constructor.py using the ortholog files generated by the matrix_constructor.py. For example, "sgt_constructor.py –trees_only -i matrix_constructor_out_<M.D.Y>/trimal/ -o <above, directory of your choice>/". If users construct trees via this method, example run is "rtc_binner.py -i <directory of your choice>/RAxML/ -st gt80trimal.tre -s gt80trimal".

Restore the database from a backup.

```
backup_restoration.py [OPTIONS] -d <path/to/database/>
```

Required arguments:

-d, --database <db_dir> Path to database directory.

Optional arguments:

--list_backups List available backups to restore from.

--restore <N> Number of backup to restore from.

-h, --help Show this help message and exit.

Calculate and plot the observed occurrences of clades of interest in bootstrap trees.

```
bipartition_examiner.py [OPTIONS] -b <input_MLBS_files> -g groups.txt
```

Required arguments:

-b, --bs_files <infile> A file that contains paths to sets of bootstrap tree files. One file name per line (Table 5). Results from files will be plotted in the order they were provided

-g, --groups <infile> A file containing taxonomic groups/relationships of interest one per line (Table 6)

Optional arguments:

-o, --output <out_dir> Path to user-defined output directory

- Default: ./bipartition_examiner_out_<M.D.Y>

--bar_plot Plot categorical data as a barplot

- Default: Plot series data as a line graph.

-h, --help Show this help message and exit.

bipartion_examiner.py output:

- bipartion_examiner.pdf - a plot (line graph or histogram) with MLBS values on the y-axis file names on the x-axis. Groups of interest are color coded.
- bipartion_examiner.tsv - a .tsv file that contains the values plotted in bipartion_examiner.pdf

/home/myworkingdirectory/MLbootstrap.experiment1
/home/myworkingdirectory/MLbootstrap.experiment2
/home/myworkingdirectory/MLbootstrap.experiment3

Table 5: Example input file provided to bipartition_examiner.py via the -b/--bs_files options providing the location of input sets of bootstrap trees. The results will be plotted in the order the files were provided.

Amoebozoa
Amoebozoa+Obazoa+CRuMs
Fungi+Metazoa
Homosapi+Gallgall
Fungi:Sacccere,Candalbi,Aspefumi,Coprcine

Table 6: Example input file provided to bipartition_examiner.py via the -g/--groups options that lists the groups of interest to be examined in the sets of bootstrap trees. If the data has been processed through the PhyloFisher workflow and a metadata.tsv exists detailing the higher and lower taxonomy of each taxon in the dataset then these taxonomic terms can be utilized here along with individual Unique IDs to examine support for relationships of interest. **In summary, lines 1-4 are only if the data has been processed through the PF workflow and a metadata file exists. If bipartion_examiner.py is being used as a stand-alone tool each group of interest will have to be defined as in line 5.** Here the group label (Fungi) is followed by a colon and sequence headers (ex. Sacccere) of all taxa in the group are listed and separated by commas.

Construct a custom database or update taxonomy in a database.

```
build_database.py [OPTIONS]
```

Optional arguments"

-t, --threads <N> Number of threads

- Default:1

-n, --no_og_file Do not make Gene OG file

-o, --og_threshold 0.X (0-1) proportion of sequences that must hit an OrthoMCL orthogroup for the group to be assigned.

- Default: 0.1 (10%)

--rename <to_rename.tsv> Rename taxa in the database. Input is a tab-delimted file (.tsv) containing the Old Unique ID, New Unique ID, and New Long Name (Table 7).

-h, --help Show this help message and exit

NOTE: build_database.py must be run within PhyloFisherDatabase_v1.0/database

build_database.py output:

- a directory "profiles" that contains:
  - profile HMMs of all ortholog files from the custom database
- a directory "datasetdb" that contains:
  - a diamond blast database of the orthologs from the custom database
- a directory "paralogs" - This empty directory is created if no paralogs directory exists initially
- a tab separated file "gene_og" with two columns:
  1. Name of gene from the custom database
  2. OrthoMCL orthogroup identification number(s) assigned to a gene from the custom database separated by commas.

What occurred:

- The script build_database.py will:
  - align the provided set of orthologs using MAFFT and create profile HMMs for each gene alignment using the "hmmbuild" utility from the HMMER3 package. These profiles will be used in the ortholog "fishing" algorithms implemented in fisher.py.
  - build a diamond blast database from the set of provided orthologs for use in the ortholog "fishing" algorithms implemented in fisher.py.
  - assign OrthoMCL orthogroup number(s) to each ortholog for use in the ortholog "fishing" algorithms implemented in fisher.py.
    - OrthoMCL orthogroup numbers are assigned by using all sequences in a provided gene file as queries in a BLAST search against the OrthoMCL v. 5.0 database. If a user defined percentage (default = 10%) of sequences hit an OrthoMCL orthogroup with a significance threshold of evalue < 1e -10 then that Orthogroup is assigned to the gene.
    - More than one OrthoMCL orthogroup numbers can be assigned to one gene.
    - If the provided gene alignment is assigned "no group" in OrthoMCL the gene cannot be used in the PhyloFisher workflow.
    - If the gene is assigned a bacterial OrthoMCL orthogroup the gene cannot be used in the PhyloFisher workflow.

NOTE: OrthoMCL orthogroup assignment hinges on integrity of ortholog choices in the starting ortholog files provided. If paralogs are unknowingly present in the provided ortholog alignments the paralogs will likely be prioritized by the fisher algorithm. To investigate the level of paralogy of genes in a custom database, we strongly recommend users re-add all taxa in their custom database using the main workflow of PhyloFisher. After an initial run through the main PhyloFisher workflow that includes manual curation, we recommend users rerun build_dataset.py to update profile HMMs, and blast databases to promote highest level of accuracy by the fisher algorithm in subsequent runs.

Old Unique ID	New Unique ID	New Full Name
Pleucart	Chrycart	Chrysotila carterae

Table 7: Example input file for the --rename flag of build_database.py. Here *Pleurochrysis carterae* (Unique ID = Pleucart) is being renamed to Chrysotila carterae (Unique ID = Chrycart) in the database.

Examine the composition of the database using taxonomic terms as search queries.

```
explore_database.py [OPTIONS]
```

Optional arguments:

-d, --database <db_dir> Path to the database directory if config.py has not been run.

-t, --higher_taxonomy Show higher taxonomy and number of taxa in each group in the database.

-l, --lower_taxonomy Show lower taxonomy and number of taxa in each group in the database.

-r, --get_higher <HigherTax> Return table with all taxa assigned the given higher taxonomy that displays the UniqueID, Long Name, Higher and Lower taxonomy, the number of orthologs, and the number of paralogs present in the database.

-w, --get_lower <LowerTax> Return table with all taxa assigned the given lower taxonomy that displays the UniqueID, Long Name, Higher and Lower Taxonomy, the number of orthologs, and the number of paralogs present in the database.

-o, --get_org <UniqueID> For the given Unique ID returns the Long Name, Higher and Lower Taxonomic Designation, Data Type, Orthologs (number present in the database for the taxon), Paralogs (number present in the database for the taxon), and the Accession.

Remove the fastest evolving sites within a phylogenomic supermatrix in a stepwise fashion.

```
fast_site_remover.py [OPTIONS] -m <input_matrix> -tr <input_tree>
```

Required arguments:

-m, --matrix <matrix.fas|nex|phy> Path to matrix

-tr, --tree <input.tre> Path to tree in Newick format

Optional arguments:

-s, --step_size <N> Size of removal step (i.e., 3000 sites removed) to exhaustion

- Default: 3000

-f, --out_format <format> Desired format of the output matrices
- Options: fasta, nexus, phylip (names truncated at 10 characters), or phylip-relaxed (names are not truncated)
- Default: fasta

-o, --output <out_dir> Path to user-defined output directory
- Default: ./fast_site_removal_out_<M.D.Y> with sub-directories:
  - steps_N

-h, --help Show this help message and exit

fast_site_remover.py output:

- a directory called ./fast_site_removal_out_<M.D.Y> with:
  - sub-directories steps_<N> (N=step size) that contain:
    - alignment files of the input matrix with N fastest sites removed iteratively until exhaustion
  - a file DE.dat[&&]
  - a file rate[&&]
  - dist_est.ctl[&&]

[&&] These are standard dist_est input/output files. See the dist_est documentation for a more detailed explanation of their contents.

Remove the fastest evolving taxa from a matrix based on branch length.

```
fast_taxa_remover.py [OPTIONS] -m <input_matrix> -tr <input_tree>
```

Required arguments:

-m, --matrix <matrix.fas|nex|phy> Path to input matrix

-tr, --tree <tree> Path to input tree

-i, --iterations <N> Number of iterations

-or, --ortholog_files Path to directory containing the individual ortholog files. This will be the path to prep_final_dataset_<M.D.Y> if used within the main PhyloFisher workflow.

Optional arguments:

-in_format <format> Input matrix format
- Options: fasta, phylip (names truncated at 10 characters), phylip-relaxed (names are not truncated), or nexus
- Default: fasta

-out_format <format> Desired output format.
- Options: fasta, phylip (names truncated at 10 characters), phylip-relaxed (names are not truncated), or nexus
- Default: fasta

-s, --step_size <N> Number taxa removed per iteration
- Default: 1

-o, --output <out_dir> Path to user-defined output directory
- Default: ./fast_taxa_removal_out_<M.D.Y>

-t, --threads <N> Desired number of threads to be utilized.
- Default: 1

-h, --help Show this help message and exit

fast_taxa_remover.py output:

- a directory called ./fast_taxa_remover_out_<M.D.Y> with sub-directories:
  - steps_<N> (N=step size) that contain:
    * a directory "prequal" that contains:
      · {gene_name}.aa - unaligned gene file used as PREQUAL input
      · {gene_name}.aa.filtered - output of PREQUAL. Used as input for MAFFT in subsequent length filtration step.
      · {gene_name}.aa.filtered.PP - output of PREQUAL.##
      · {gene_name}.aa.warning - output of PREQUAL.##
    * a directory "mafft" that contains:
      · {gene_name}.aln - output of MAFFT and input for Divvier.
    * a directory "divvier" that contains:
      · {gene_name}.aln.partial.fas - output of Divvier and input for timAl.
      · {gene_name}.aln.PP - output of Divvier.$$
    * a directory "trimAl" that contains:
      · {gene_name}.gt80trimAl - output of trimAl. Trimmed alignments that will be used for concatenation.
    * indices.tsv - a tab separated file with three columns outlining the single gene boundaries in the supermatrix. These columns are:
      1. Gene - name of the gene
      2. Start - first position of the gene within the super matrix
      3. Stop - last position of the gene within the super matrix
    * matrix.<fas|nex|phy> - the concatenated super matrix of all genes in the provided input directory in the specified file format.
    * matrix_constructor_stats.tsv - a tab separated file with two columns:
      1. Taxon -Unique ID of taxon in the database
      2. Percent Missing Data - percentage of unoccupied sites within the concatenated matrix.

- These are standard PREQUAL output files for each gene that PhyloFisher has appended the corresponding gene name to. See the PREQUAL documentation for a thorough explanation of their contents.

$$ - These are standard Divvier output files for each gene that PhyloFisher has appended the corresponding gene name to. See the Divvier documentation for a thorough explanation of their contents.

Explore potential alternative nuclear genetic codes for input taxa

```
genetic_code_examiner.py [OPTIONS] -i <input_file> -q <query_file>
```

Required arguments:

-i, --input <infile.fas> Fasta file with nucleotide sequences

-q, --queries <Unique ID1,Unique ID2,...> Comma separated Unique IDs of related organisms in the database which should be used as queries.

Optional arguments:

-t, --threads <N> Number of threads

- Default: 1

--prepare_alignments Prepare alignments for genetic code analysis. MUST BE USED ON THE FIRST RUN!

-c, --conserved <N> Proportion (0-1) of taxa from the database that have the same amino acid at a position.

- Default: 0.7 (70%)

-e, --blast_evalue <1e-X> E-value threshold for blast searches.

- Default: 1e-30

-a, --all_codons Plot conserved positions for all codons

-o, --output <out_dir> Path to user-defined output directory

- Default: ./genetic_code_out_<M.D.Y>

-h, --help Show this help message and exit

NOTE: Users should have their config.ini file in the directory in which they wish to run genetic_code_examiner.py. Users may need to provide updated paths in this file to account for a different directory location.

Default genetic_code_examiner.py output:

- a file inputname_genecode.pdf - Each bar chart in the file corresponds to a separate plot for codons with a suspicious genetic code signal (signal for coding schemes different from the standard nuclear genetic code) if any exist (Figure 8).

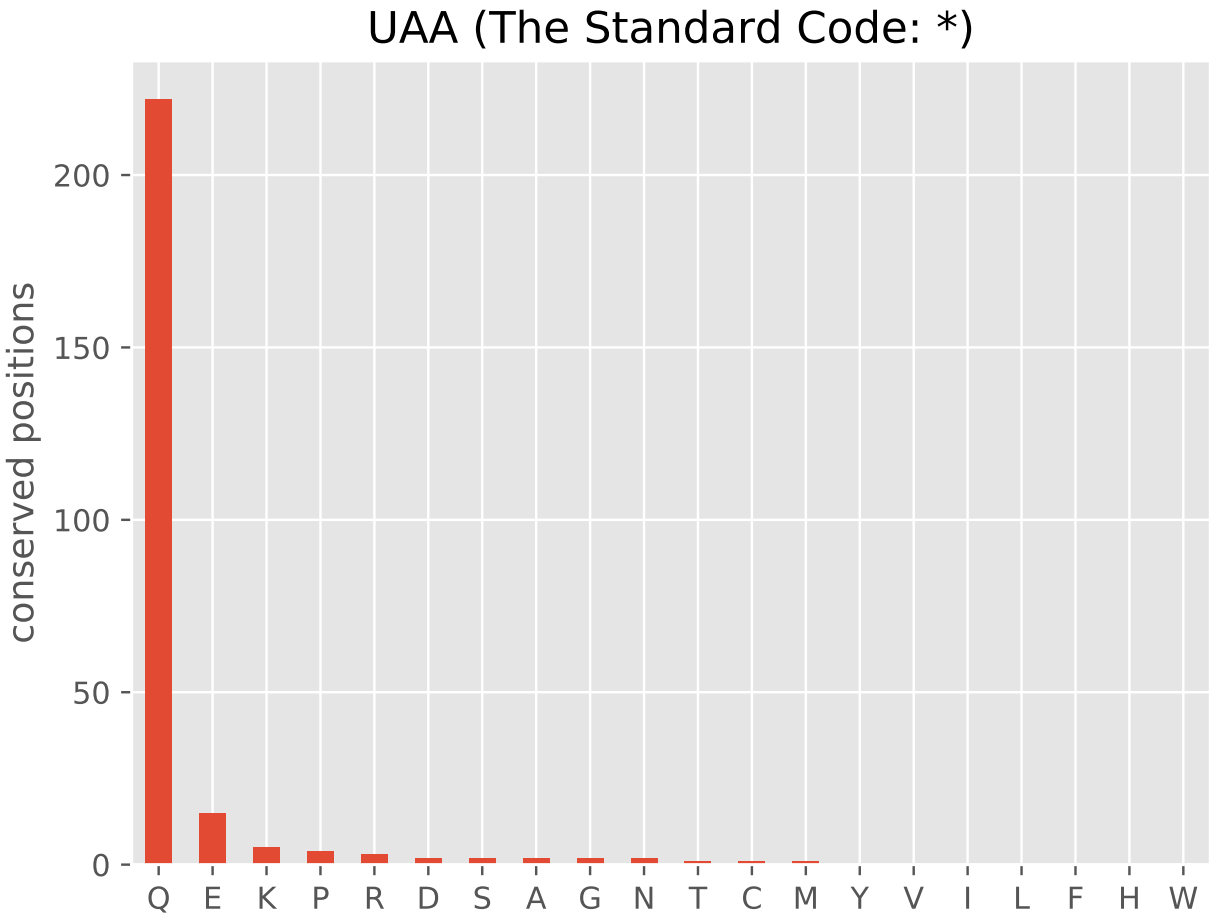

Figure 8: Plot produced by genetic_code_examiner.py. The y-axis shows the number of taxa from the database with the amino acid labeled on the x-axis conserved for a given site and the input taxon has the codon in the title at the same site. This plot provides evidence that the codon UAA that acts as a stop codon (*) in the standard nuclear genetic code has been reassigned to code for the amino acid glutamine (Q) in the nuclear genome of *Tetrahymena thermophila*.

Remove the most heterotachious sites from a phylogenomic supermatrix in a stepwise fashion.

```
heterotachy.py -tr <input_tree> -m <input_matrix> [OPTIONS]
```

Required arguments:

-tr, --tree <input.tre> Path to tree in Newick format

-m, --matrix <matrix.fas|nex|phy> Path to supermatrix

Optional arguments:

-s, --step_size <N> Size of removal step (i.e., 1000 sites removed) to exhaustion

- Default: 3000

-f, --out_format <format> Desired format of the output matrices

- Options: fasta, nexus, phylip (names truncated at 10 characters) or phylip-relaxed (names are not truncated)
- Default: fasta

-o, --output <out_dir> Path to user-defined output directory

- Default: ./heterotachy_out_<M.D.Y>

-h, --help Show this help message and exit

Default heterotachy.py output

- a directory heterotachy_out_<M.D.Y> that contains:
  - slow.tre - tree file in Newick format containing a tree pruned from the input tree and contains only taxa determined to be slow-evolving.
  - fast.tre - tree file in Newick format containing a tree pruned from the input tree and contains only taxa determined to be fast-evolving.
  - slow.phy - phylip formatted file containing only slow-evolving taxa.
  - fast.phy - phylip formatted file containing only fast-evolving taxa
  - slow.dist_est.ctl - control file to be used by dist_est for slow-evolving taxa
  - fast.dist_est.ctl - control file to be used by dist_est for fast-evolving taxa
  - slow.DE.dat[&&] - an output of the dist_est operation on slow.phy and slow.tre.
  - slow.rate_est.dat[&&] - an output of the dist_est operation on slow.phy and slow.tre.
  - fast.DE.dat[&&] - an output of the dist_est operation on fast.phy and fast.tre.
  - fast.rate_est.dat[&&] - an output of the dist_est operation on fast.phy and fast.tre.
  - a subdirectory steps_<N> (N=step size) that contains the files:
    - step<0-N>.phy - phylip formatted matrix files with N less sites than the previous step to exhaustion

[&&] These are standard dist_est input/output files. See the dist_est documentation for a more detailed explanation of their contents.

Generate a MAMMaL site heterogeneous model for a user defined number of classes

```
mammal_modeler.py [OPTIONS] -m <input_matrix>
```

Required arguments:

-tr, --tree <input.tre> Path to tree in Newick format

-m, --matrix <matrix.fas|nex|phy> Path to supermatrix

Optional arguments:

-if, --in_format <format> Input format of matrix

- Options: fasta, nexus, phylip (names truncated at 10 characters) or phylip-relaxed (names are not truncated)
- Default: fasta

-c, --classes <N> The number of frequency classes in the mixture model

- Options: 10, 20, 30, 40, 50, or 60
- Default: 60

-o, --output <out_dir> Path to user-defined output directory

- Default: ./mammal_modeler_out_<M.D.Y>

-h, --help Show this help message and exit.

Default mammal_modeler.py output:

- a directory mammal_modeler_out_<M.D.Y> that contains:
  - a file esmodel.nex - nexus file that contains the MAMMaL mixture model. This file can be used to fit a mixture model in IQtree using the options " -m LG+ESmodel+G -mdef esmodel.nex".
    - NOTE: Do not use +F (i.e., LG+ESmodel+G+F) when using this model. Instead use (LG+ESmodel+G). The overall amino acid frequencies are already presented in the esmodel.nex file
  - a file estimated_frequencies - a tab-delimited output file that each row gives the amino acid frequencies for a class. This file is output by MAMMaL, but is not used further here. See the MAMMaL documentation for more information.

Delete taxa and/or taxonomic groups from the database and metadata.tsv.

```
purge.py [OPTIONS] -i <input_directory>
```

Required arguments:

-i, --input <to_purge.txt> Path to text file containing Unique IDs and/or Taxonomic designations of organisms for deletion.

-d, --database <input_dir> Path to database to purge.

Optional arguments:

-h, --help Show this help message and exit

Construct supermatrices from randomly sampled genes.

```
random_resampler.py [OPTIONS] -i <input_directory>
```

Required arguments:

-i, --input <input_dir> Path to input directory containing gene files in fasta format

Optional arguments:

-if, --in_format <format> Format of the input single gene alignments.
- Options: fasta, phylip (names truncated at 10 characters), phylip-relaxed (names are not truncated), or nexus.
- Default: fasta

-of, --out_format <format> Desired format of the output steps.
- Options: fasta, nexus, phylip (names truncated at 10 characters), or phylip-relaxed (names are not truncated)
- Default: fasta

-ci, --confidence_interval <0.N> Confidence interval to use to calculate the number of replicates required.
- Default: 0.95

-ps, --percent_sampling <N> Percent sampling step size
- Default: 20%
- The default 20% sampling results in a sampling series of 20%, 40%, 60%, and 80%

-o, --output <out_dir> Path to user-defined output directory
- Default: ./random_sample_iteration_out_<M.D.Y>_ps<percentage increment>_ci<confidence interval>

-p, --prefix <prefix> Prefix of input files
- Default: NONE
- Example: path/to/input/prefix*

-s, --suffix <suffix> Suffix of input files
- Default: NONE
- Example: path/to/input/*suffix

-h, --help Show this help message and exit.

Default random_resampler.py output:

- a directory ./random_resampler_out_<M.D.Y>ps<percentage> that contains:
  - a set of supermatrices made from the randomly sampled genes. Each matrix has a file name structured in the following way: <percentage of genes from total sampled>rep<replicate number>.<fas|phy|nex>
  - a set of files "<percentage of genes from total sampled>_Percent.tsv" where column headers correspond to replicate matrices created and the rows below are genes within a particular replicate matrix.
  - a set of files "<percentage of genes from total sampled>_Percent.tgz" that contain all replicates for the respective increment and the corresponding .tsv file.

Bin orthologs based on relative tree certainty score and construct supermatrices from the bins.

```
rtc_binner.py [OPTIONS] -i <input_dir>
```

Required arguments:

-i, --input <input_dir> Path to directory containing single gene trees built from only orthologs, corresponding bootstrap value files, and corresponding alignments.

Optional arguments:

--in_format <format> Format of the input files.

- Options: fasta, nexus, phylip (names truncated at 10 characters), or phylip-relaxed (names are not truncated)
- Default: fasta

--out_format <format> Desired format of the output files.

- Options: fasta, nexus, phylip (names truncated at 10 characters), or phylip-relaxed (names are not truncated)
- Default: fasta

-o, --output <out_dir> Path to user-defined output directory

- Default: ./rtc_binner_out_<M.D.Y>

-p, --prefix <prefix> Prefix of input files

- Default: NONE
- Example: path/to/input/prefix*

-s, --suffix <suffix> Suffix of input files.

- Default: NONE
- Example: path/to/input/*suffix

-h, --help Show this help message and exit.

rtc_binner.py output:

- a directory ./rtc_binner_out_<M.D.Y> with 7 subdirectories:
  - matrix_constructor25/ that contains:
    - supermatrix (matrix, stats, and indices) of the top 25% RTC scored orthologs
  - matrix_constructor50/ that contains:
    - supermatrix (matrix, stats, and indices) of the top 50% RTC scored orthologs
  - matrix_constructor75/ that contains:
    - supermatrix (matrix, stats, and indices) of the top 75% RTC scored orthologs
  - rtc25/ that contains:
    - alignments of the top 25% RTC scored orthologs
  - rtc50/ that contains:
    - alignments of the top 50% RTC scored orthologs
  - rtc75/ that contains:
    - alignments of the top 75% RTC scored orthologs

Recode an input matrix based on SR4 amino acid classification

```
SR4_class_recoder.py [OPTIONS] -i <input_matrix>
```

Required arguments:

-i, --input <matrix.fas|nex|phy> Path to input matrix

Optional arguments:

--in_format <format> Input file format if not fasta

- Options: fasta, phylip (names truncated at 10 characters), phylip-relaxed (names are not truncated), or nexus
- Default: fasta

o, --output <out_dir> Path to user-defined output directory

- Default: ./SR4_class_recoder_out_<M.D.Y>

-h, --help Show this help message and exit

SR4_class_recoder.py output:

- the input matrix in fasta format with amino acids re-coded to four amino acid classes represented by four nucleotide characters.

Permanently combine taxa that have been added to the database

```
taxon_collapser.py [OPTIONS] -i <input.tsv>
```

Required arguments:

-i, --input to_collapse.tsv A .tsv containing a Unique ID, higher and lower taxonomic designations, long name, and the Unique IDs of the taxa to collapse, for each chimera one per line.

Optional arguments:

-h, --help Show this help message and exit.

taxon_collapser.py output:

- a new line in metadata.tsv with the information provided in the input .tsv.
- protein sequences for the new taxon in the corresponding ortholog and paralog fasta files in the database

Frequently Asked Questions

How do I cite PhyloFisher?

Tice et al. (2021). PhyloFisher: A phylogenomic package for resolving eukaryotic relationships. doi.

What operating systems does PhyloFisher run on?

All aspects of PhyloFisher can be installed and run on Linux and MacOS systems. Only ParaSorter can also be run on Windows.

How do I report an issue with PhyloFisher?

Please report any errors on the PhyloFisher GitHub Repository

Can I run PhyloFisher on my laptop?

All aspects of PhyloFisher will run on a laptop computer. However, more powerful computational resources are recommended for single gene tree construction and phylogenomic tree construction using a large database such as the one provided.

How long does a typical PhyloFisher run take?

The length of time an entire run takes is dependent on the computational resources available.

Can I use my own starting database?

Yes. PhyloFisher is compatible with phylogenomic databases other than the one provided. Go here for further instructions.

How do I change the default color selection used for single gene tree visualization?

Simply change the color for taxa in tree_colors.tsv. New colors can be selected from here.

Can I change the designation of orthologs and paralogs in the provided database?

Yes. The designation of sequences as orthologs or paralogs in the provided database is not fixed. These can be changed during manual inspection of single gene trees and the changes will be applied to the database in downstream steps.

Can I delete taxa or genes from the provided database?

Yes. We have provided purge.py for permanent removal of taxa from the database. Permanent removal of genes would require re-building a custom database from only the desired genes. This is possible and instructions for constructing a custom database are provided here. However, an easier solution could be to simply not include the genes in final phylogenomic datasets via select_orthologs.py.

Do I have to use the provided script for single gene tree construction?

No. If you prefer to use alternate strategies from trimming and single gene tree construction other than those implemented in "sgt_constructor.py" you can re enter the PhyloFisher workflow after you have constructed single gene trees with your preferred methodology. However, you must use a maximum likelihood approach for single gene tree construction in order for downstream aspects of PhyloFisher to work properly.

References

Ali, R. H., Bogusz, M., & Whelan, S. (2019). Identifying clusters of high confidence homologies in multiple sequence alignments. *Molecular Biology and Evolution*, *36*(10), 2340–2351. https://doi.org/10.1093/molbev/msz142

Altschul, S. F., Gish, W., Miller, W., Myers, E. W., & Lipman, D. J. (1990). Basic local alignment search tool. *J. Mol. Biol.*, *215*(3), 403–410. https://doi.org/10.1016/S0022-2836(05)80360-2

Brown, M. W., Heiss, A. A., Kamikawa, R., Inagaki, Y., Yabuki, A., Tice, A. K., Shiratori, T., Ishida, K.-I., Hashimoto, T., Simpson, A. G. B., & Roger, A. J. (2018). Phylogenomics places orphan protistan lineages in a novel eukaryotic super-group. *Genome Biology and Evolution*, *10*(2), 427–433. https://doi.org/10.1093/gbe/evy014

Buchfink, B., Xie, C., & Huson, D. H. (2015). Fast and sensitive protein alignment using DIAMOND. *Nature Methods*, *12*(1), 59–60. https://doi.org/10.1038/nmeth.3176

Capella-Gutiérrez, S., Silla-Martínez, J. M., & Gabaldón, T. (2009). trimAl: A tool for automated alignment trimming in large-scale phylogenetic analyses. *Bioinformatics*, *25*(15), 1972–1973. https://doi.org/10.1093/bioinformatics/btp348

Chen, F., Mackey, A. J., Stoeckert, C. J., Jr, & Roos, D. S. (2006). OrthoMCL-DB: Querying a comprehensive multi-species collection of ortholog groups. *Nucleic Acids Research*, *34*, D363–D368. https://doi.org/10.1093/nar/gkj123

Criscuolo, A., & Gribaldo, S. (2010). BMGE (block mapping and gathering with entropy): A new software for selection of phylogenetic informative regions from multiple sequence alignments. *BMC Evolutionary Biology*, *10*(1), 210. https://doi.org/10.1186/1471-2148-10-210

Fu, L., Niu, B., Zhu, Z., Wu, S., & Li, W. (2012). CD-HIT: Accelerated for clustering the next-generation sequencing data. *Bioinformatics*, *28*(23), 3150–3152. https://doi.org/10.1093/bioinformatics/bts565

Huerta-Cepas, J., Serra, F., & Bork, P. (2016). ETE 3: Reconstruction, Analysis, and Visualization of Phylogenomic Data. *Molecular Biology and Evolution*, *33*(6), 1635–1638. https://doi.org/10.1093/molbev/msw046

Inagaki, Y., Susko, E., Fast, N. M., & Roger, A. J. (2004). Covarion Shifts Cause a Long-Branch Attraction Artifact That Unites Microsporidia and Archaebacteria in EF-1 Phylogenies. *Molecular Biology and Evolution*, *21*(7), 1340–1349. https://doi.org/10.1093/molbev/msh130

Katoh, K., & Standley, D. M. (2013). MAFFT multiple sequence alignment software version 7: Improvements in performance and usability. *Molecular Biology and Evolution*, *30*(4), 772–780. https://doi.org/10.1093/molbev/mst010

Lopez, P., Casane, D., & Philippe, H. (2002). Heterotachy, an Important Process of Protein Evolution. *Molecular Biology and Evolution*, *19*(1), 1–7. https://doi.org/10.1093/oxfordjournals.molbev.a003973

Mistry, J., Finn, R. D., Eddy, S. R., Bateman, A., & Punta, M. (2013). Challenges in homology search: HMMER3 and convergent evolution of coiled-coil regions. *Nucleic Acids Research*, *41*(12), e121–e121. https://doi.org/10.1093/nar/gkt263

Price, M. N., Dehal, P. S., & Arkin, A. P. (2010). FastTree 2 - approximately maximum-likelihood trees for large alignments. *PLoS ONE*, *5*(3), 1–10. https://search.ebscohost.com/login.aspx?direct=true&db=a9h&AN=56440094&site=eds-live&scope=site&custid=magn1307

Stamatakis, A. (2014). RAxML version 8: A tool for phylogenetic analysis and post-analysis of large phylogenies. *Bioinformatics*, *30*(9), 1312–1313. https://doi.org/10.1093/bioinformatics/btu033

Susko, E., Field, C., Blouin, C., & Roger, A. J. (2003). Estimation of rates-across-sites distributions in phylogenetic substitution models. *Systematic Biology*, *52*(5), 594–603. https://doi.org/10.1080/10635150390235395

Susko, E., Lincker, L., & Roger, A. J. (2018). Accelerated estimation of frequency classes in site-heterogeneous profile mixture models. *Molecular Biology and Evolution*, *35*(5), 1266–1283. https://doi.org/10.1093/molbev/msy026

Susko, E., & Roger, A. J. (2007). On Reduced Amino Acid Alphabets for Phylogenetic Inference. *Molecular Biology and Evolution*, *24*(9), 2139–2150. https://doi.org/10.1093/molbev/msm144

Tice, A. K., Shadwick, L. L., Fiore-Donno, A. M., Geisen, S., Kang, S., Schuler, G. A., Spiegel, F. W., Wilkinson, K. A., Bonkowski, M., Dumack, K., Lahr, D. J. G., Voelcker, E., Clauß, S., Zhang, J., &

Brown, M. W. (2016). Expansion of the molecular and morphological diversity of acanthamoebidae (centramoebida, amoebozoa) and identification of a novel life cycle type within the group. *Biology Direct*, *11*(1), 69. https://doi.org/10.1186/s13062-016-0171-0

Whelan, S., Irisarri, I., & Burki, F. (2018). PREQUAL: Detecting non-homologous characters in sets of unaligned homologous sequences. *Bioinformatics*, *34*(22), 3929–3930. https://doi.org/10.1093/bioinformatics/bty448

Zhang, C., Rabiee, M., Sayyari, E., & Mirarab, S. (2018). ASTRAL-III: Polynomial time species tree reconstruction from partially resolved gene trees. *BMC Bioinformatics*, *19*(6), 153. https://doi.org/10.1186/s12859-018-2129-y