Create, Analyze, and Visualize Phylogenomic Datasets Using PhyloFisher

Robert E. Jones,[1,2] Alexander K. Tice,[1,2,3] Marek Eliáš,[4] Laura Eme,[5] Martin Kolísko,[6,7] Serafim Nenarokov,[6] Tomáš Pánek,[8] Antonis Rokas,[9] Eric Salomaki,[6,10] Jürgen F. H. Strassert,[11] Xing-Xing Shen,[12] David Žihala,[4] and Matthew W. Brown[1,2,13]

[1]Department of Biological Sciences, Mississippi State University, Starkville, Mississippi, USA
[2]Institute for Genomics, Biocomputing & Biotechnology, Mississippi State University, Starkville, Mississippi, USA
[3]Department of Biological Sciences, Texas Tech University, Lubbock, Texas, USA
[4]Department of Biology and Ecology, Faculty of Science, University of Ostrava, Ostrava, Czech Republic
[5]Unité d'Ecologie, Systématique et Evolution, CNRS, Université Paris-Saclay France, Orsay, France
[6]Institute of Parasitology, Biology Centre Czech Academy of Sciences, České Budějovice Czech Republic, USA
[7]Faculty of Science, University of South Bohemia, České Budějovice, Czech Republic
[8]Department of Zoology, Faculty of Science, Charles University, Prague, Czech Republic
[9]Department of Biological Sciences and Evolutionary Studies Initiative, Vanderbilt University, Nashville, Tennessee, USA
[10]Center for Computational Biology of Human Disease and Center for Computation and Visualization, Brown University, Providence, Rhode Island, USA
[11]Department of Evolutionary and Integrative Ecology, Leibniz Institute of Freshwater Ecology and Inland Fisheries (IGB), Berlin, Germany
[12]Institute of Insect Sciences, Zhejiang University, Hangzhou, China
[13]Corresponding author: *matthew.brown@msstate.edu*

Published in the Bioinformatics section

PhyloFisher is a software package written primarily 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, a subset of a previous phylogenetic dataset sampled from 304 eukaryotic taxa. The software package can also utilize a user-created database of eukaryotic proteins, which may be more appropriate for shallow evolutionary questions. PhyloFisher is also equipped with a set of utilities to aid in running routine analyses, such as the prediction of alternative genetic codes, removal of genes and/or taxa based on occupancy/completeness of the dataset, testing for amino acid compositional heterogeneity among sequences, removal of heterotachious and/or fast-evolving sites, removal of fast-evolving taxa, supermatrix creation from randomly resampled genes, and supermatrix creation from nucleotide sequences. © 2024 Wiley Periodicals LLC.

Basic Protocol 1: Constructing a phylogenomic dataset
Basic Protocol 2: Performing phylogenomic analyses
Support Protocol 1: Installing PhyloFisher
Support Protocol 2: Creating a custom phylogenomic database

Keywords: evolution • genomics • systematics • transcriptomics

CURRENT PROTOCOLS
A Wiley Brand
Current Protocols e969, Volume 4
Published in Wiley Online Library (wileyonlinelibrary.com).
doi: 10.1002/cpz1.969

How to cite this article:
Jones, R. E., Tice, A. K., Eliáš, M., Eme, L., Kolísko, M., Nenarokov, S., Pánek, T., Rokas, A., Salomaki, E., Strassert, J. F. H., Shen, X.-X., Žihala, D., & Brown, M. W. (2024). Create, analyze, and visualize phylogenomic datasets using phylofisher. *Current Protocols*, *4,* e969. doi: 10.1002/cpz1.969

INTRODUCTION

Single-gene molecular phylogenetic analyses revolutionized our understanding of the evolutionary history of life. However, they lacked the necessary phylogenetic signal to fully resolve especially deep branches in the tree of life (Leipe et al., 1993). In the 2000s, the field of molecular phylogenetics transformed into phylogenomics, a discipline that combines the signal from hundreds of genes (or proteins) per taxon, yielding a better understanding of deep phylogenetic relationships between lineages (Baldauf et al., 2000). However, phylogenomic databases have been difficult to manage, maintain, and update with newly available genomic data. Historically, this has been done either by manually editing the sequence files or by use of private "in-house" code, both of which drastically reduce reproducibility. In addition, as computing resources become more powerful and genomic sequencing exponentially cheaper, the amount of data included in phylogenomic studies has been consistently and rapidly increasing, making manual dataset management a difficult task. The main aim of PhyloFisher is to ease the burden of constructing, curating, maintaining, and updating phylogenomic databases in a reproducible and time-saving fashion. Both the PhyloFisher software package and associated sequence database are publicly available, easy to install, and regularly updated to ensure its proper function. PhyloFisher is also designed to open phylogenomics to researchers new to the field and to help them overcome the challenges of the complex workflow that phylogenomic analysis requires—from the selection of orthologs to post-tree analyses.

The terms database and dataset are used throughout PhyloFisher documentation. The two terms are, however, not synonymous. The database is a collection of manually curated sequences demarcated as either orthologs or paralogs. This can refer either to the provided database or to a custom database supplied by the user. Decisions made by a user regarding orthology and paralogy during manual curation of sequences from input proteomes are stored within the "database". The initial database provided was manually curated prior to its release; however, the user's application of different phylogenetic methods, manual curation practices, and taxon selection will cause the database to change over time. A dataset can refer to either a "working dataset" or a "phylogenomic 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 back to the database. A working dataset is not revisited after decisions from manual curation have been made and applied to the database. A phylogenomic dataset is 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. Figure 1 illustrates the flow of data between major steps of the workflow, and how the flow relates to a working dataset, a phylogenomic dataset, and a database.

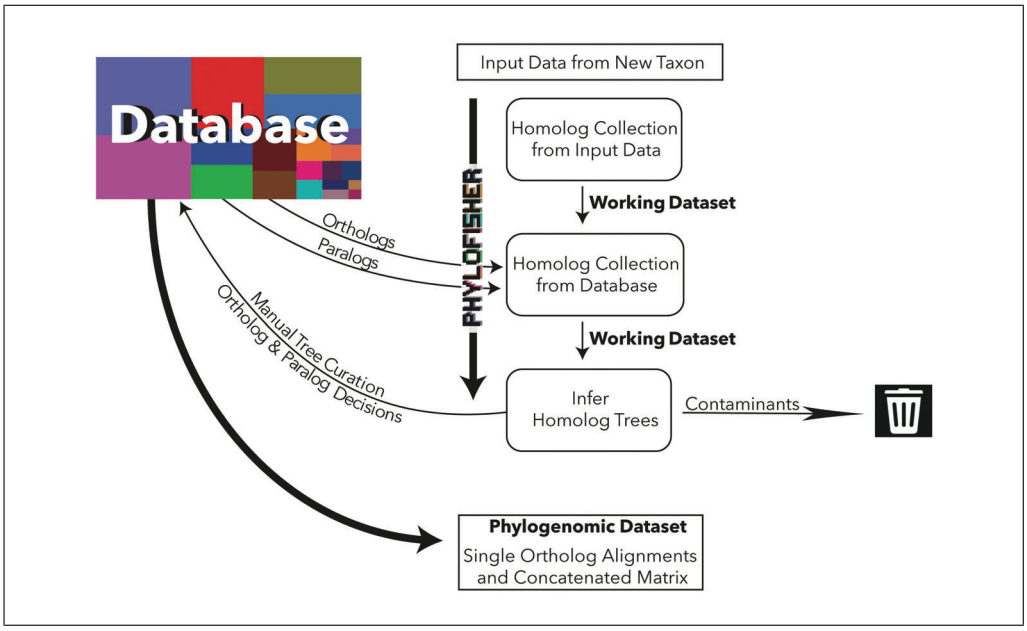

Figure 1 Flow chart of the three major PhyloFisher main workflow steps leading to the production of a phylogenomic dataset.

CONSTRUCTING A PHYLOGENOMIC DATASET

BASIC PROTOCOL 1

When originally conceived, PhyloFisher had the primary objective of situating novel taxa on the Tree of Eukaryotes. Although its utility extends to various other types of inquiries, we will illustrate its application within this specific context. Take, for example, a case where an isolated organism is suspected to belong to the Obazoa group—comprising Fungi, Metazoa, and their protistan counterparts (Brown et al., 2013)—based on morphology. Verification of taxon classification within Obazoa and an exploration of its phylogenetic placement within the group may be achieved through phylogenomic analysis, which entails the construction of an appropriate set of orthologs from the species under investigation. Using PhyloFisher, this step begins with a preexisting database, either the one provided with PhyloFisher or a user-constructed database. User-built databases can be constructed by using the utility `build_database.py`. The user can then enter the PhyloFisher main workflow, which has three major steps involved in dataset construction with the following scripts: `fisher.py`, `sgt_constructor.py`, and manual homolog-tree inspection via `forest.py` and `ParaSorter` (Fig. 1). `fisher.py` selects putative orthologs from input proteomes either through the default route or the phylogenetically informed route described in Tice et al. (2021). The output of `fisher.py` contains, for each protein of the initial database, putative orthologs for newly added organisms and sequences from the starting database; this is used to build an individual phylogenetic tree for each protein. These trees are then manually inspected for accuracy of ortholog assignment, and the user makes and records any necessary changes. The results of the tree inspection are then applied to the database. The final dataset to be used for phylogenomic tree construction is created from the curated set of orthologs present in the updated starting database that now contains sequences from newly added organisms (Tice et al., 2021).

Necessary Resources

Hardware

- A Unix-based high-performance computing environment with access to the Internet.
- A local computer capable of running `forest_local.py` and `ParaSorter`

Table 1 input_metadata.tsv Included in the Sample Run through PhyloFisher

Location	File name	Unique ID	Higher taxonomy	Lower taxonomy	BLAST seed	Long name	Data type	Source
./	Batrdend.faa	Batrdend	Obazoa	Fungi	Allomacr	*Batrachochytrium dendrobatidis*	Genomic	GCF_000203795.1

Table 2 Example Segment of metadata.tsv Included in the Example Database

Unique ID	Long Name	Higher Taxonomy	Lower Taxonomy	Data Type	Source
Gregniph	*Gregarina niphandrodes* (GNI3)	Alveolata	Apicomplexa	Genomic	GCF_000223845.1
Protadhe	*Protocruzia adherens*	Alveolata	Ciliata	Transcriptomic	SRR1296823
Stensten	*Stenamoeba stenopodia*	Amoebozoa	Discosea	Transcriptomic	SRR5396404
Idiovort	*Idionectes vortex*	Amoebozoa	Evosea	Transcriptomic	SRX5656095

Software

PhyloFisher (high-performance computing environment)
`ParaSorter` (locally) (*https://github.com/TheBrownLab/PhyloFisher/tree/master/parasorter*)
`forest.py` (locally)

Files

Input proteomes in FASTA format.
Sample File. Sample data can be found at *https://github.com/TheBrownLab/phylofisher-current-protocols*.

1. Download and decompress the PhyloFisher starting database in your desired location (this location can be anywhere on your computer for which you have "write" privileges).

```
$ wget https://ndownloader.figshare.com/files/29093409
$ tar -xzvf 29093409
```

2. 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 phylofisher_tutorial
$ cd phylofisher_tutorial
```

3. Create the file input_metadata.tsv (Table 1) with information about your taxa-to-be-added. The input_metadata.tsv file contains nine columns of information about input taxa that are 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 spreadsheet software such as Microsoft Excel and saved as a tab-delimited text file. Place the completed input_metadata.tsv file in the 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 (Table 2) 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. Download an example input_metadata.tsv.

```
$ wget https://raw.githubusercontent.com/TheBrownLab/phylofisher-current-protocols/main/addition/input_metadata.tsv
```

Table 3 contaminants.tsv Input of `forest.py`

Unique ID	Taxonomic term	Taxonomic level
Stensten	Fungi	Lower taxonomy

4. Create and set up the PhyloFisher configuration file.

```
$ config.py -d .. /PhyloFisherDatabase_v1.0/database/ -i input_metadata.tsv
```

5. Collect putative homologs from input taxa.

```
$ fisher.py -o fisher_out
```

6. Produce preliminary statistics about newly input data.

```
$ informant.py -i fisher_out
```

   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 or paralogs in the data (sequences that passed all criteria during `fisher.py` but will be removed during manual curation because they do not truly come from the organism's genome or are not orthologs of the gene of interest). These statistics will almost always change after manual curation. However, they can be generated quickly and are useful for a rough estimate of data completeness with regard to the starting database.

7. Construct a working dataset: Collect taxa and homologs for homolog tree construction.

```
$ working_dataset_constructor.py -i fisher_out -o working_dataset_constructor_out
```

   If you wish to exclude genes, taxa, and/or paralogs from the working dataset used for homolog tree construction (NOT RECOMMENDED) change the column values for "Homolog Tree" in gene_stats.tsv and/or org_stats.tsv to "NO" and then run. Similarly, changing the column values for "Paralogs" in "db_taxa_stats.tsv" to "NO" will lead to the inclusion of only orthologs for those taxa in homolog trees used for paralog/ortholog vetting.

8. Filter, align, and trim homolog files followed by optional homolog tree construction.

```
$ sgt_constructor.py -i working_dataset_constructor_out -o sgt_constructor_out
```

   If you would like to perform filtering, aligning, and trimming without homolog tree construction via RAxML, use the –no_tree flag.

   If you choose to build homolog 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 tree must follow {nameofyourchoosing}.{gene_name}.tre

9. Render .svg and .tsv files of homolog trees for visualization with `ParaSorter`. If working on a remote server, copy the file sgt_constructor_out_< MDY>.tar.gz file to your local machine for input into `forest.py`, using the –local_run flag.

```
$ forest.py --local_run -i sgt_constructor_out-local.tar.gz -o forest_out
```

   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 (Table 3). The first column contains the Unique ID of the contaminated input proteome and the second column contains the 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, the third column contains the name of the taxonomic level

chosen for the contaminant. When this file is provided, `forest.py` will pre-mark instances of this branching pattern for deletion regardless of maximum likelihood bootstrap support (MLBS) for the relationship.

If a user discovers pervasive contamination during manual curation of homolog trees, 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 –contaminants and –backpropagate flags given. When the –backpropagate is added, all decisions made previously in 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.

Only sequences from newly added organisms can be pre-marked for contamination or have contamination backpropagated. Contamination discovered in previously added organisms must be marked manually if found after their initial addition.

10. Homolog Tree Inspection

The PhyloFisher workflow strongly encourages manual inspection of homolog trees to ensure that only sequences with an orthologous relationship make it into the final phylogenomic dataset and that any sequences derived from contaminants are removed. While this process is an arduous portion of the workflow, tools are provided within PhyloFisher to help alleviate some of this burden. Below, we also provide guidelines for ortholog selection during the construction of the PhyloFisher v. 1.0.0-provided starting database.

To manually inspect a homolog tree, open the application `ParaSorter`, which can be downloaded from the PhyloFisher GitHub repository.

Once `ParaSorter` has opened:

In the upper left corner, click “Open tree” and choose a .svg file created by `forest.py` in the last step.
Click “Import tsv” and choose the corresponding gene’s .tsv file.

Figure 2 illustrates a segment of a homolog tree visualized with `ParaSorter` (top) and of the PhyloFisher naming scheme for phylogenetic tree leaves (bottom). To the right of each sequence in the tree are three boxes. If selected, the leftmost box (green) will display a capital letter “O” for “ortholog”, the middle box (yellow) will display a capital letter “P” for “paralog”, and the rightmost box (red) will display a capital letter “D” for “delete.” These boxes are used to display and change a sequence’s current or future designation in the database. To change a sequence’s designation, simply click the box that corresponds to the desired assignment. These designations will be applied to the starting database in the next step.

Below is an overview of information provided in sequence names (phylogenetic tree leaves) and homolog tree files to better inform ortholog, paralog, and contamination selection during manual inspection (Fig. 2):

A) User-provided “Long Name” of the taxon.
B) Route through `fisher.py` that produced sequence:
   a. Specific Best Hit (SBH) – appended if a specific query produced a significant hit and the sequence was sister to another sequence from the same taxonomic group in the phylogeny automatically inferred by `fisher.py` using FastTree (Price et al., 2010).
   b. Best Blast Hit (BBH) – appended if a specific query produced a significant hit and the sequence was *not* sister to another sequence from the same taxonomic group in the phylogeny automatically inferred by `fisher.py` using FastTree.

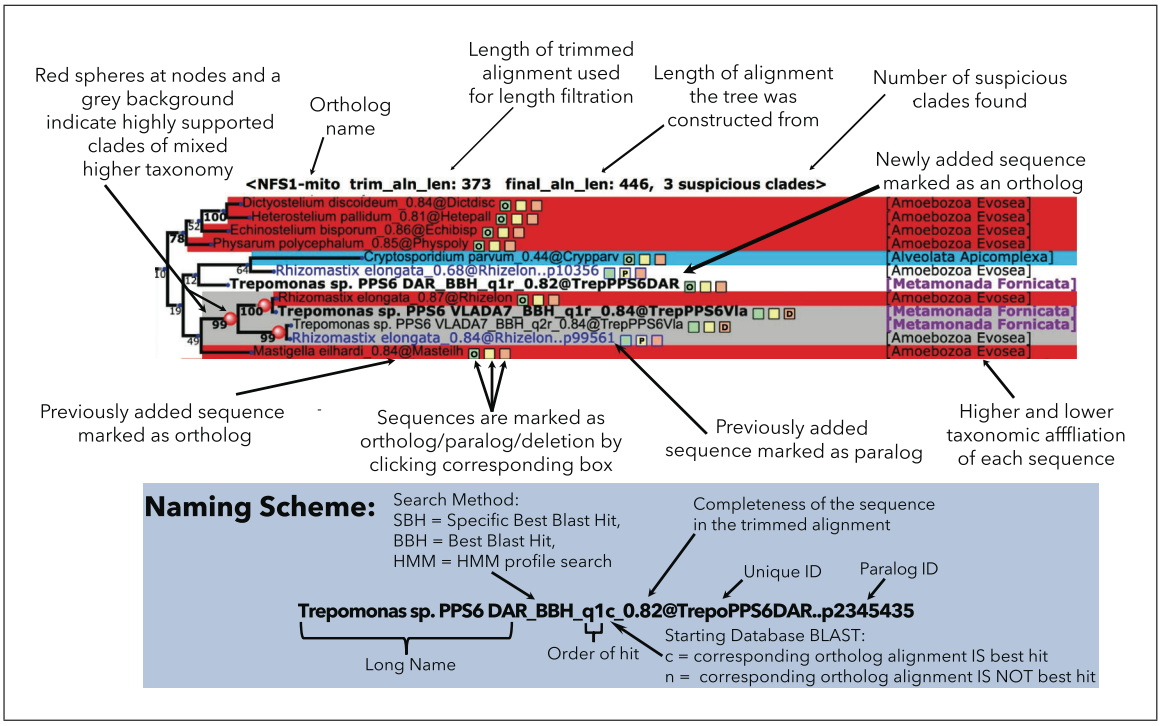

Figure 2 Example segment of a homolog tree visualized with `ParaSorter` (above) and of the PhyloFisher sequence header naming scheme (below).

c. HMMER route (HMM) – appended if the sequence was selected by the default route using a hmmer search (Mistry et al., 2013).

C) Priority in final set of collected sequences:
   a. 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 (e.g., q1, q2, …, qn).
   b. If the collected sequence's best BLAST hit is or is not a sequence in the corresponding ortholog alignment from the database.
      i. c - collected sequence's best BLAST hit IS a sequence in the corresponding ortholog alignment from the database.
      ii. n - collected sequence's best BLAST hit IS NOT a sequence in the corresponding ortholog alignment from the database.

D) The proportional length of the sequence in the trimmed alignment used for length filtering.

E) Unique ID of taxon in dataset.

F) Higher and Lower taxonomic designations.

G) Other valuable information provided:
   a. 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.
   b. A clade is marked as suspicious if it is supported by an MLBS value of 70 or greater and contains sequences from multiple higher taxonomic groups. These clades are highlighted with a grey background and each node that meets the above criteria is marked with a red sphere.
   c. 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.

d. Sequences that are newly added are not highlighted in color based on taxonomy.
e. Orthologs already in the database are highlighted in color based on taxonomy and are written in regular black font.
f. Paralogs already in the database are not highlighted in color based on taxonomy and are written in regular blue font.
g. Sequences pre-marked for deletion are not highlighted in color based on taxonomy and are written in regular red font. Red sequences will only be presented if known contaminants were provided to `forest.py` via the –contaminants and –backpropagate flags.

Here we provide the logic used for ortholog selection during the construction of PhyloFisher v. 1.0.0 provided starting database as described in Salomaki et al. (2020) and Tice et al. (2021). However, many of these guidelines are flexible for users with knowledge of systematics and molecular evolution to adjust based on their preferences.

A) In cases of in-paralogy, defined as instances of a duplication event that occur in a single organism, the longest sequence was retained as the ortholog (Fig. 3a).
B) In cases of mid or deep paralogy, clades with 50% or greater MLBS and sequences from at least one taxon present on both sides of the initial bifurcation were considered to represent gene duplication events.
   a. In the case of such duplication events, sequences on the side of the duplication that contained the most sequence data (measured as the sum total of retained sequence length) were retained as the orthologs (Fig. 3b).
   b. In instances where the sequence data retention was nearly equal on both sides of the bifurcation, sequences from the side which had the most species represented were retained as the orthologs (Fig. 3c).
   c. Figure 3d shows a case were paralogy clearly exists for taxon A and B. While the tree topology suggests that taxon G is also paralogous, as it branches with taxa A and B, it can be retained as an ortholog because its position is not highly supported and there is no strong evidence for its position with the paralogous sequences from A and B.

After all decisions have been made, click "Save to tsv" on the top left-hand side of the `ParaSorter` display. The default name `ParaSorter` will suggest {gene_name}_parsed.tsv as the new name of the file. The file must be named {gene_name}_parsed.tsv for subsequent steps of the workflow to operate correctly.

11. Apply tree parsing decision to the database.

```
$ apply_to_db.py -i forest_out -fi fisher_out
```

`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 in PhyloFisherDatabase_v1.0/database/orthologs/, append paralogs for input taxa to their corresponding gene files located in PhyloFisherDatabase_v1.0/database/paralogs/, and remove sequences from the database that were marked for deletion (this does not impact proteome files stored in PhyloFisherDatabase_v1.0/database/proteomes/). The file metadata.tsv (Table 2) will also be updated with information provided for newly added taxa in input_metadata.tsv (Table 1). Next, `apply_to_db.py` will rebuild the database so sequences from newly added taxa are present in the profile HMMs and the diamond database. This will allow orthologs from the newly added taxa to be used as specific queries in subsequent addition runs. Finally, `apply_to_db.py` will copy the input proteome of all new taxa added to the database to PhyloFisherDatabase_v1.0/database/proteomes/.

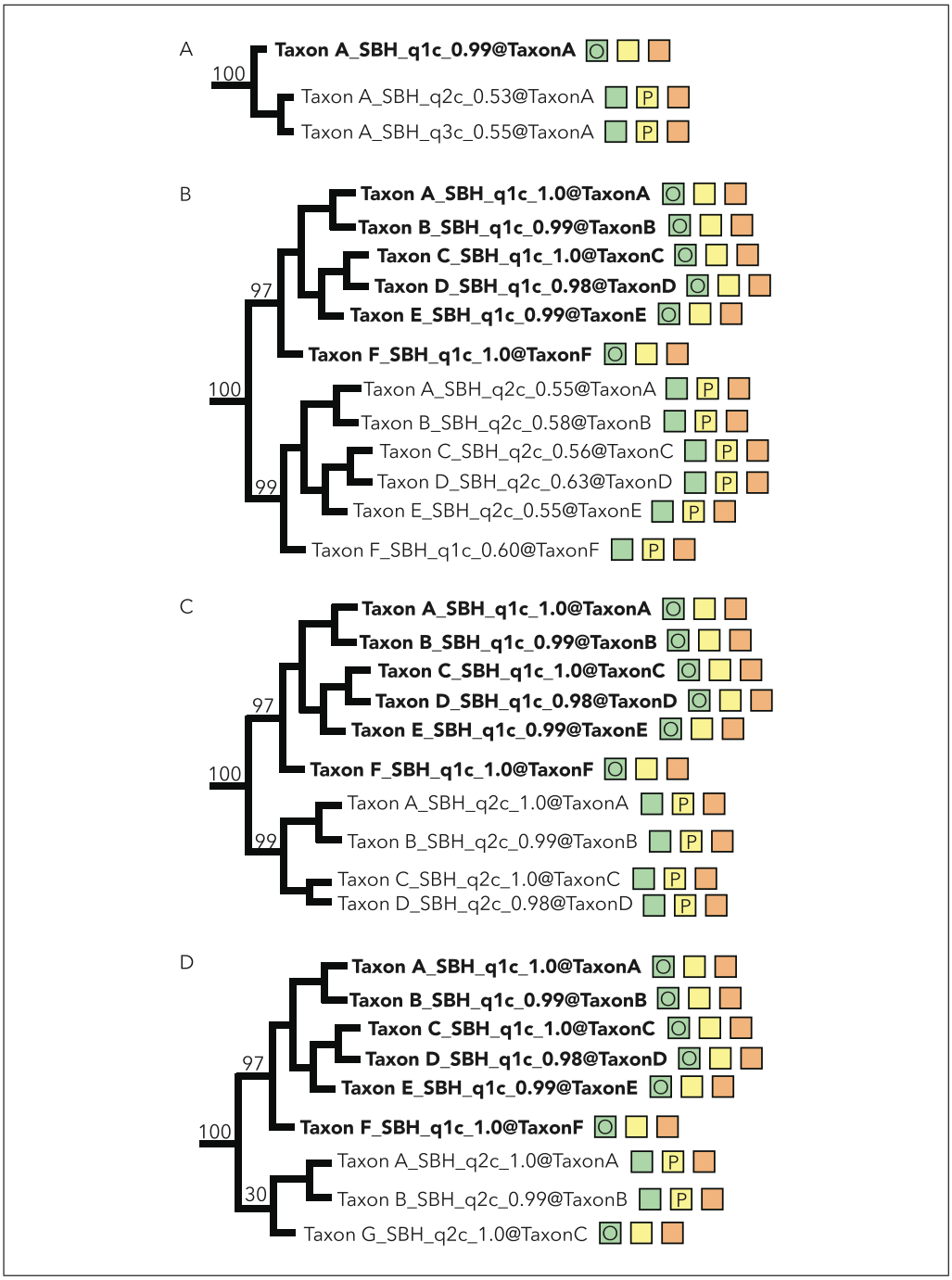

Figure 3 Diagrammatic subsections of phylogenetic trees showing how ortholog and paralog assignments were made during construction of the PhyloFisher 1.0.0 database. (**a**) In cases where a duplication event occurred in a single organism, the longest sequence was retained as the ortholog. (**b**) In cases where two clades supported by >50% MLBS that contain overlapping taxa in each, but on one side taxa are represented more completely in the sequence alignment, the sequences on the side of the duplication that contained the most sequence data (measured as the sum total of retained sequence length shown as a proportion to the immediate left of the "@" symbol in the sequence header) were retained as the orthologs. (**c**) In cases where two clades supported by >50% MLBS that contain overlapping taxa, and the sequence data retention was nearly equal for taxa on both sides of the bifurcation, sequences from the side which had with the most species represented were retained as the orthologs. (**d**) Illustration of cases where paralogy clearly exists for taxons A and B. While tree topology suggests that taxon G is also paralog as it branches with taxa A and B, it can be retained as ortholog since its position is not highly supported and there is no strong evidence for its position with the paralogous sequences from A and B.

12. (OPTIONAL) Select taxa to compose final phylogenomic matrix.

```
$ select_taxa.py
```

By default, all taxa will be included. The –to_include and –to_exclude options were designed to decrease the amount of manual manipulation of select_taxa.tsv generated by `select_taxa.py`. 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 –to_exclude option with a file (named anything) that contains "Amoebozoa" as the first column entry. This will result in all amoebozoans being marked as "no" in the "Include in Subset" column of select_taxa.tsv when it is generated. If a user wanted all amoebozoans excluded except *Dictyostelium discoideum* (Unique ID = Dictdisc), provide the –to_exclude 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 column entry . This will result in all amoebozoans being marked as "no" in the "Include in Subset" column of select_taxa.tsv, but *D. discoideum* will be marked "yes."

–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, it is not necessary to provide –to_include with a list of all taxa you wish to include in downstream steps.

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."

13. (OPTIONAL) Select genes to compose final phylogenomic matrix.

```
$ select_orthologs.py
```

By default, all genes will be included. The options –gene_number and –percent_complete filter the genes to be included based on completeness. Completeness is the percentage of taxa containing a gene. For example, if the gene ADK2 is present in 80 of 100 taxa, the completeness of ADK2 is 80%. –percent_complete allows for genes greater than or equal to the given threshold to be included. The –gene_number option receives an integer. Genes are sorted based on completeness and the highest n, with n being the integer given to –gene_number, number of genes and marked "yes" and the remaining genes "no".

If `select_taxa.py` is run prior to `select_orthologs.py`, completeness is calculated based on only those taxa to be included in the final phylogenomic matrix rather than all taxa in the database.

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

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

```
$ prep_final_dataset.py -o prep_final_dataset_out
```

At this step, FASTA-formatted files for the orthologs selected in step 13 are created containing the taxa selected in step 12.

15. Align, trim, and concatenate orthologs into a super-matrix.

```
$ matrix_constructor.py -i prep_final_dataset_out -o matrix_constructor_out
```

Common errors and their solutions are detailed in Table 9.

PERFORMING PHYLOGENOMIC ANALYSES

Here we demonstrate how to perform phylogenetic analyses using a concatenation-based approach with IQ-TREE2.

Concatenation of individually aligned and trimmed loci results in a large super-matrix that contains a multiple sequence alignment generated from stitching together all trimmed orthologs from a given taxon one after another, creating a super-matrix file. Each character of a column in the super-matrix is presumed homologous, meaning they share common evolutionary ancestry, amongst all taxa, and it accounts for missing data within an ortholog or the complete absence of the ortholog for an individual taxon. Typically, we use this super-matrix file as a large single protein alignment; however, PhyloFisher also provides ortholog boundaries in a tab-delimited file allowing users to use individual protein substitution models per ortholog (i.e., partitioned models). This is output by `matrix_constructor.py` as *indices.tsv*. When inferring phylogenomic trees, accounting for site specific amino acid replacement patterns is critical for accurate phylogenetic inferences in the face of phylogenetic signal saturation and long-branch attraction artifacts (Wang et al., 2018; Yang, 1994; Lartillot & Philippe, 2004; Le et al., 2008). For ML analyses we utilize profile mixture classes (Le et al., 2008) which are variants of the CAT model (Lartillot & Philippe, 2004). In our examination of model parameters in use with the PhyloFisher super-matrix in Tice et al. (2021), our preferred approach first uses the LG+C20+G+F model, which is the LG protein substitution matrix (Le et al., 2008) applied to the 20-profile mixture classes (C20) of site rate heterogeneity plus another class for the empirical AA profile (counted from the current data) and gamma rate heterogeneity across sites (Yang, 1994) with a default of four rate categories. Under this LG+C20+G+F model, we infer an ML tree, which is then used as a preliminary guide tree (-ft) to infer another ML tree under a model with 60 profile mixtures (C60) with the LG model applied to these profiles plus the empirical AA profile (+F) of the data also with four gamma categories employing the posterior mean site frequency (PMSF) method as a computationally efficient strategy in IQ-TREE2. This algorithm also generates a conditional mean amino acid frequency profile for each site output as (*.sitefreq). This approach permits full nonparametric bootstrap analyses under this complex site-heterogeneous model (LG+C60+G+F+PMSF) on large concatenated super-matrices, which would otherwise be computationally challenging due to high requirements in RAM and computational time (Wang et al., 2018).

Necessary Resources

Hardware

A Unix-based high-performance computing environment

Software

IQ-TREE2

Sample File. Sample data can be found at *https://github.com/TheBrownLab/phylofisher-current-protocols*.

Concatenation-based analysis using IQ-TREE2:

1. Generate a LG+C20+F+G ML tree from the super-matrix
2. Applying LG matrix for 20 classes plus the 21st class of empirical AA profile (counted from the current data) and gamma rate heterogeneity. The `-mem` and `-T` options should be adjusted based on available resources.

```
$ iqtree2 -T 60 -m LG+C20+F+G -s matrix.fas -pre PF.tutorial.LGC20GF -mem 800G
```

3. Use the LG+C20+F+G ML tree from the super-matrix as a guide tree for LG+C60+F+G+PMSF to generate an ML tree and a site frequencies file (.sitefreq).

```
$ iqtree2 -T 60 -ft PF.tutorial.LGC20GF.treefile -m LG+C60+F+G -s matrix.fas
-pre PF.tutorial.LGC60GCF-PMSF -mem 800G
```

4. Make a directory to run Real Bootstrap replicated and move into it.

```
$ mkdir REALBS
$ cd REALBS
```

5. Make 100 Real Bootstrap replicate submission scripts.

```
$ iqtree2 -T 1 -m LG+C60+F+G -s matrix.fas -fs ../PF.tutorial.LGC60GCF-
PMSF.sitefreq -pre PF.tutorial.LGC60GCF-PMSF.realbs_100 -mem 80G --bonly 1
```

6. Write Bootstrap values to tree.

```
$ raxmlHPC-PTHREADS-AVX2 -f b -t PF.tutorial.LGC20GF.LGC60GCF-PMSF.treefile -z boot.tre
-n PF.tutorial.LGC20GF.LGC60GCF-PMSF.100MLBS -N 2 -m PROTGAMMALGF -s ../matrix.fas
```

SUPPORT PROTOCOL 1

INSTALLING PHYLOFISHER

Bioinformatic software comes in multiple different languages and forms, each of which has a unique method of installation and dependency handling. Academic software bears its own set of unique challenges due to the need for reproducibility. In domains outside of academics, system administrators are responsible for installing and maintaining software. However, academic scientists must often handle these tasks on their own. To overcome such difficulties, we utilize the Conda package manager (*https://conda.io*) to install PhyloFisher. The PhyloFisher Conda package is housed in the Bioconda channel (Grüning et al., 2018). Here we present how to install PhyloFisher via the Conda package manager.

Necessary Resources

Hardware

A Unix-based computing environment with access to the internet.

1. Install mamba via mamba-forge. Mamba greatly reduces environment solve time when creating conda environments.

```
$ curl -L -O "https://github.com/conda-forge/miniforge/releases/latest/download/Mambaforge-$(uname)-$(uname-m).sh"
$ bash Mambaforge-$(uname)-$(uname -m).sh
```

2. Create a conda virtual environment.

```
$ mamba create -n fisher
```

3. Activate the conda virtual environment.

```
$ conda activate fisher
```

4. Add the Bioconda & Conda-Forge Anaconda Cloud channels to your channels.

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

5. Install PhyloFisher.

```
$ mamba install phylofisher
```

SUPPORT PROTOCOL 2

PREPARING A CUSTOM STARTING DATABASE

Due to the pan-eukaryotic composition of the provided starting database and the marker genes it contains, the provided database will not be suitable for all protein-based

phylogenomic studies. Therefore, users may benefit from creating their own, more appropriate, starting database. Here we detail the necessary steps to construct a custom starting database to be used within the PhyloFisher workflow.

Necessary Resources

Hardware

A Unix-based high-performance computing environment with access to the internet

Software

PhyloFisher (high-performance computing environment)

Files

Amino acid sequences for the genes and taxa to be included in the starting database in FASTA format

1. Retrieve the required PhyloFisher directory structure, OrthoMCL v. 5.0 database, example metadata.tsv (Table 2), and tree_colors.tsv file via wget.

```
$ wget https://ndownloader.figshare.com/files/29093325
```

2. Decompress the file 29093325.

```
$ tar -xzvf 29093325
```

3. Move into the directory PhyloFisher_FOR_CUSTOM_DATABASE.

```
$ cd PhyloFisher_FOR_CUSTOM_DATABASE
```

4. Populate database/orthologs/

```
$ cp <source> 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 and nothing else (see step 7 below for rules regarding Unique ID structure). This Unique ID must be the same in all ortholog files.

Each taxon can be present only once in each ortholog file.

5. (OPTIONAL) Populate database/paralogs/

Place files of known paralogs for each gene in the directory database/paralogs/

```
$ cp <source> database/paralogs
```

Each gene file must be named with the following convention {gene_name}_paralogs.fas (i.e., RPL7_paralogs.fas).

Each individual taxon should have a Unique ID as the header in all paralog files. This Unique ID must be the same in all paralog files and the corresponding ortholog files.

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

6. (OPTIONAL) Populate database/proteomes/

```
$ cp <source> database/proteomes
```

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 compressed with both tar and gzip and follow the naming convention {Unique ID}.faa.tar.gz (i.e., Homosapi.faa.tar.gz).

7. Fill out the metadata.tsv file.

   The file metadata.tsv (Table 2) 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`. Once created, a user should never be required to open this file to view its contents manually. However, the structure of metadata.tsv is detailed below.

   `metadata.tsv` is a tab-delimited file with six columns:

   i) Unique ID - An abbreviated name for each taxon. We followed an eight-letter convention in the provided starting database. Unique IDs cannot contain underscores "_", at symbols "@", or double dots "..".
   ii) Long Name - Full name of the input taxon. The long name cannot contain underscores "_", at symbols "@", double dots "..".
   iii) 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 that 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.
   iv) Lower Taxonomy - A taxonomic rank at or above the genus level for the input taxon
   v) Data Type - A place to add a note about the type of data being added (e.g., transcriptomic, genomic, EST…
   vi) Source - A place to add notes about the source of the data such as accession numbers, "in-house information", strain information, etc.

   A comma (,) may not be used anywhere in metadata.tsv (Table 2).

8. Build Database

   ```
   $ build_database.py
   ```

   In this step, `build_database.py` will first align the provided set of orthologs using MAFFT (Katoh & Standley, 2013) and create profile HMMs for each gene alignment using the hmmbuild utility from the HMMER3 package (Mistry et al., 2013). Next, a DIAMOND BLAST database (Buchfink et al., 2021) will be built from the set of provided orthologs for use in the ortholog "fishing" algorithms implemented in `fisher.py`. Finally, `build_database.py` assigns OrthoMCL (Li et al., 2003) 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 e-value <1e-10, then that Orthogroup is assigned to the gene. More than one OrthoMCL orthogroup number 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. Also, if the gene is assigned a bacterial OrthoMCL orthogroup the gene cannot be used in the PhyloFisher workflow.

   OrthoMCL orthogroup assignment hinges on the integrity of ortholog choices in the starting ortholog files provided. If paralogs are unknowingly present in the provided ortholog

alignments, they could be prioritized by the fisher algorithm. To investigate the level of paralogy of genes in a custom database, we strongly recommend that 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, `build_dataset.py` *will update profile HMMs and blast databases to promote the highest level of accuracy by the fisher algorithm in subsequent runs.*

COMMENTARY

Background Information

PhyloFisher is not without limitations. For instance, it does not work with nucleotides as the initial input. However, nucleotide sequences can be utilized downstream to build nucleotide supermatrices with `nucl_matrix_constructor.py` (Jones et al., 2023). If a user begins with nucleotide data, amino acid sequences must first be inferred using an amino acid inference software like TransDecoder (Haas & Papanicolaou, 2017). Once amino acid sequences are inferred, the user can enter the PhyloFisher workflow. Also, due to the highly conserved nature of the genes composing the starting database, we do not recommend using the provided PhyloFisher database for investigating divergences less than 100 mya. In these cases, it is recommended that users build their own starting database with more appropriate genes for the time scale in question. This can be achieved by utilizing the script, `build_database.py`, which is a utility script included within PhyloFisher.

The homolog tree parsing step of the PhyloFisher workflow allows for an increase in accuracy compared to the default designation provided by `fisher.py`. This step requires manual curation by the user, which can be time intensive. However, this strategy is highly encouraged in the PhyloFisher main workflow, since programmatically assigning paralog and ortholog designations can be difficult due to the complex nature of homolog relations.

The PhyloFisher starting database performs similarly to thousands of genes. Therefore, utilizing the database requires less computational time compared to databases with many more genes. The database contains sequence data for 304 eukaryotic taxa from various public sources and is comprised of 240 "housekeeping" genes, a subset of a previous phylogenomic dataset, BORDOR, that was developed in Tice et al. (2016). The provided script `explore_database.py` can be used to parse the "metadata.tsv" file, which contains information about data that comprises the provided starting database, such as the species name, taxonomic information, source information, and number of the 240 marker genes identified for each taxon (Tice et al., 2021).

Critical Parameters

Deciding between a phylogenetically informed and default algorithm is important. In cases where novel lineages of eukaryotes or organisms with completely unknown phylogenetic positions are added, it is better to use the default algorithm, as choosing the wrong seed query can lead to distorted results and inevitable inclusion of paralogs.

The BLAST query field in input_metadata.tsv (Table 1) used by `fisher.py` is an important parameter for accurate results. The best practice is to use the most closely related organism available in the starting database. A poor choice in BLAST query can lead to the collection of contaminating sequences if present or paralogs being inaccurately classified as orthologs by the `fisher.py` algorithm.

All significant BLAST hits from `fisher.py` can be kept by utilizing the –all_bbh flag. This leads to an increase in candidate orthologs. If contamination is present in the input sequences, it is probable that this contamination will be picked up and inaccurately included in the candidate orthologs. While the inclusion of contamination and paralogs can be dealt with in single homolog tree inspection, it is important to keep in mind the ramifications of considering all best BLAST hits.

Paralogs from the starting database can be included or excluded by editing the db_taxa_stats.tsv (Table 4) file as output by `informant.py`. The inclusion of paralogs from the starting database in homolog tree construction can allow for easier classification of orthologs and paralogs during manual inspection.

`sgt_constructor.py` can be circumnavigated, and alternative single homolog tree-building strategies can be implemented. Users can develop their own pipelines to trim, align, and build trees from the amino acid sequence files output by `working_dataset_constructor.py`.

Table 4 db_taxa_stats.tsv Output by `informant.py`

Unique ID	Higher taxonomy	Lower taxonomy	Genes out of 5	Long name	SGT	Paralogs
Allomacr	Obazoa	Fungi	5	*Allomyces macrogynus*	yes	yes
Ancysigm	Ancyromonadida	Ancyromonas	5	*Ancyromonas sigmoides* B70	yes	none
Bigenata	Rhizaria	Cercozoa	5	*Bigelowiella natans*	yes	yes
Carpmemb	Metamonada	Fornicata	5	*Carpediemonas membranifera*	yes	yes

Table 5 Sources and Solutions to Potential Errors

Problem	Possible cause	Solution
The file, config.ini, is missing	The configuration step of PhyloFisher has not been completed	Run `config.py` with the required arguments
`fisher.py` input file contains nucleotide sequences	A FASTA file of nucleotide sequences has been provided to `fisher.py`	Infer amino acid sequence from nucleotide FASTA file and rerun `fisher.py` with inferred nucleotide sequences as the input
`fisher.py` input file does not exist	Incorrect path is present in input_metadata.tsv (Table 1)	Correct the file path in input_metadata.tsv (Table 1)
Unique ID already in metadata, from `fisher.py`	The Unique ID provided in input_metadata.tsv (Table 1) already exists in the starting database	Choose a Unique ID not already in use by taxa in the starting database
BLAST query not in the starting database, from `fisher.py`	The blast query provided in input_metadata.tsv (Table 1) does not exist in the starting database.	Provide the Unique ID of a taxon in the starting database to be used as the blast query.
Too many values to unpack, from `fisher.py`	Extra column(s) present after handling input_metadata.tsv (Table 1)	Remove extra column(s) from input_metadata.tsv. (Table 1). Each row should have nine columns.
Too many values to unpack, from `informant.py`	Extra column(s) present after handling db_taxa_stats.tsv (Table 4), gene_stats.tsv (Table 6), or new_taxa_stats.tsv (Table 7). [*Place Table 6 near here]; [*Place Table 5 near here];	Remove extra column(s) from db_taxa_stats.tsv (Table 4), gene_stats.tsv (Table 6), or new_taxa_stats.tsv (Table 7). Each row should have seven, four, and eleven, columns, respectively.
Too many values to unpack, from `select_taxa.py`	Extra column(s) present after handling select_taxa.tsv (Table 8) [*Place Table 7 near here];	Remove extra column(s) from select_taxa.tsv. (Table 8). Each row should have six columns.
Too many values to unpack, from `select_orthologs.py`	Extra column(s) present after handling select_orthologs.tsv (Table 9) [*Place Table 8 near here];	Remove extra column(s) from select_orthologs.tsv. (Table 9). Each row should have three columns.

Table 6 gene_stats.tsv Output by `informant.py`

Gene Name	Number of Taxa	Percent of Total Taxa (out of 32)	SGT
CCT-B	32	100.0	yes
CCT-A	31	96.88	yes
RPS3	31	96.88	yes
RPL12	30	93.75	yes
CDK5	26	81.25	yes

Table 7 new_taxa_stats.tsv Output by `informant.py`

Unique ID	Long Name	Higher Taxonomy	Lower Taxonomy	Sequences Collected	Genes out of 5	#SBH	#BBH	#HMM	SGT
Batrdend	*Batrachochytrium dendrobatidis*	Obazoa	Fungi	6	5	3	3	0	yes

Table 8 select_taxa.tsv Output by `select_taxa.py`

UniqueID	Long name	Higher taxonomy	Lower taxonomy	Completeness	Include in subset
Gregniph	*Gregarina niphandrode*s (GNI3)	Alveolata	Apicomplexa	100.0	yes
Protadhe	*Protocruzia adherens*	Alveolata	Ciliata	100.0	yes
Stensten	*Stenamoeba stenopodia*	Amoebozoa	Discosea	100.0	yes
Idiovort	*Idionectes vortex*	Amoebozoa	Evosea	100.0	yes

Table 9 select_orthologs.tsv Output by `select_orthologs.py`

Orthologs	Completeness	Include in subset
CCT-B	1.0	yes
CCT-A	0.969	yes
RPS3	0.969	yes
RPL12	0.938	yes
CDK5	0.812	yes

User-developed pipelines could be a modified version of the pipeline implemented in `sgt_constructor.py`, such as differing trimming and aligning parameters. A pipeline with major differences such as different programs to align, trim, and build trees can also be used. For example, a user may prefer IQ-TREE2 (Nguyen et al., 2015) over RAxML (Stamatakis, 2014) for homolog tree construction. Differing tree-building methodologies can impact the topology of the resulting homolog tree. This, in turn, could impact ortholog and paralog designation during manual inspection.

Ortholog/paralog selection in homolog tree inspection is a crucial step in the main PhyloFisher workflow. Sequences classified as orthologs will be included in downstream phylogenomic analysis. Both paralogs and orthologs will go on to aid homolog tree inspectors in the classification of new sequences.

Troubleshooting

If you encounter problems while attempting to install or run PhyloFisher, please raise them on the "Issues" page of the PhyloFisher GitHub repository at *https://github.com/TheBrownLab/PhyloFisher/issues*. The following are some common issues that may arise when running PhyloFisher.

Installation issues

We recommend using the PhyloFisher Bioconda package, which will make installation and future updates much easier to manage. Alternatively, a docker image is also available and can be utilized on clusters with singularity (Kurtzer et al., 2017) installed.

Issues with input files

Ensure input proteomes are in FASTA format and do not include non-standard amino acids or non-ASCII characters in the sequences.

Time Considerations

Basic Protocol 1 – Constructing a phylogenomic dataset using the sample database and 5 threads takes around 90 min. Two scripts account for the bulk of this time used. The scripts `fisher.py` and `sgt_constructor.py`, utilizing 5 threads, take approximately 2 and 90 min to complete, respectively. A run-through of the PhyloFisher main workflow that adds 15 taxa to a database with 240 genes and 304 taxa, takes `fisher.py` and `sgt_constructor.py` approximately 1 hr and 8 days, respectively, when utilizing 80 threads.

Basic Protocol 2 – Performing phylogenomic analyses, as detailed in Basic Protocol 2, requires users to run IQ-TREE2. Running IQ-TREE2, using LG+C20+F+G as the model of evolution, with an input matrix containing 70428 sites, 57 taxa, and 64 threads takes approximately 8 hr and 30 min to run. Under LG+C60+F+G+PMSF with

40 threads, IQ-TREE2 takes approximately 8 hr to complete. Each bootstrap replicate using 1 thread runs for approximately 5 days.

Support Protocol 1 – Installing mamba via mamba-forge takes approximately 1 min. The installation of PhyloFisher via mamba takes approximately 3 min to complete.

Support Protocol 2 – The script, `build_database.py`, is used to build custom starting databases. The script, with 5 threads, takes approximately 2 min to complete on the sample database, which contains 5 genes and 32 taxa. For a full-size database with 240 genes and 304 taxa, `build_database.py` completes in about 90 min with 5 threads.

Acknowledgments

This work was supported in part by the United States National Science Foundation (NSF) Division of Environmental Biology (DEB) grants 1456054 and 2100888 (*http://www.nsf.gov*) awarded to M.W.B. X.X.S. was supported by the Fundamental Research Funds for the Central Universities (226-2023-00021). D.Z. was supported by European Union project LERCO (No. CZ.10.03.01/00/22_003/0000003) via the Operational Programme Just Transition. T.P. was supported by Charles University Research Centre program No. 204069. The development of PhyloFisher in the M.E. and M.K. labs was supported by the project CePaViP (ERD funds, project no. CZ.02.1.01/0.0/0.0/16_019/0000759).

Author Contributions

Robert Jones: Conceptualization; formal analysis; investigation; methodology; software; writing—original draft; writing—review and editing. **Alexander Tice:** Conceptualization; data curation; formal analysis; investigation; methodology; software; validation; visualization; writing—original draft; writing—review and editing. **Marek Eliáš:** Conceptualization; funding acquisition; writing—review and editing. **Laura Eme:** Conceptualization; writing—review and editing. **Martin Kolisko:** Conceptualization; funding acquisition; methodology; supervision; writing—review and editing. **Serafim Nenarokov:** Software; writing—review and editing. **Tomáš Pánek:** Conceptualization; writing—review and editing. **Antonis Rokas:** Writing—review and editing. **Jürgen F. H. Strassert:** Writing—review and editing. **Eric Salomaki:** Writing—review and editing. **Xing-Xing Shen:** Writing—review and editing. **David Žihala:** Software; writing—review and editing. **Matthew Brown:** Conceptualization; funding acquisition; investigation; methodology; project administration; resources; supervision; validation; writing—review and editing.

Conflict of Interest

The authors declare no conflict of interest.

Data Availability Statement

PhyloFisher is licensed and freely distributed under the MIT License. The PhyloFisher source code is available through the GitHub repository, *https://github.com/TheBrownLab/PhyloFisher*, and is maintained on Bioconda and as a Docker container at *https://quay.io/biocontainers/phylofisher*. The PhyloFisher starting database is available at *https://ndownloader.figshare.com/files/29093409*. All other data and code described in this manuscript are provided in a dedicated GitHub repository at *https://github.com/TheBrownLab/phylofisher-current-protocols*.

Literature Cited

Baldauf, S. L., Roger, A. J., Wenk-Siefert, I., & Doolittle, W. F. (2000). A kingdom-level phylogeny of eukaryotes based on combined protein data. *Science*, *290*(5493), 972–977. https://doi.org/10.1126/science.290.5493.972

Brown, M. W., Sharpe, S. C., Silberman, J. D., Heiss, A. A., Lang, B. F., Simpson, A. G., & Roger, A. J. (2013). Phylogenomics demonstrates that breviate flagellates are related to opisthokonts and apusomonads. *Proceedings of the Royal Society B: Biological Sciences*, *280*(1769), 20131755. https://doi.org/10.1098/rspb.2013.1755

Buchfink, B., Reuter, K., & Drost, H. G. (2021). Sensitive protein alignments at tree-of-life scale using DIAMOND. *Nature Methods*, *18*(4), 366–368. https://doi.org/10.1038/s41592-021-01101-x

Grüning, B., Dale, R., Sjödin, A., Chapman, B. A., Rowe, J., Tomkins-Tinch, C. H., Valieris, R., Köster, J., & The Bioconda Team. (2018). Bioconda: Sustainable and comprehensive software distribution for the life sciences. *Nature Methods*, *15*(7), 475–476. https://doi.org/10.1038/s41592-018-0046-7

Haas, B., & Papanicolaou, A. (2017). *TransDecoder*. https://Github.Com/TransDecoder/TransDecoder

Jones, R. E., Jones, E. P., Tice, A. K., & Brown, M. W. (2023). A PhyloFisher utility for nucleotide-based phylogenomic matrix construction; nucl_matrix_constructor.py. *bioRxiv*. 11.30.569490. https://doi.org/10.1101/2023.11.30.569490

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

Kurtzer, G. M., Sochat, V., & Bauer, M. W. (2017). Singularity: Scientific containers for mobility of compute. *PLOS ONE*, *12*(5), e0177459. https://doi.org/10.1371/journal.pone.0177459

Lartillot, N., & Philippe, H. (2004). A Bayesian mixture model for across-site heterogeneities in the amino-acid replacement process. *Molecular Biology and Evolution*, *21*(6), 1095–1109. 10.1093/MOLBEV/MSH112

Le, S. Q., Lartillot, N., & Gascuel, O. (2008). Phylogenetic mixture models for proteins. *Philosophical Transactions of the Royal Society B: Biological Sciences*, *363*(1512), 3965–3976. https://doi.org/10.1098/rstb.2008.0180

Leipe, D. D., Gunderson, J. H., Nerad, T. A., & Sogin, M. L. (1993). Small subunit ribosomal RNA+ of Hexamita inflata and the quest for the first branch in the eukaryotic tree. *Molecular and Biochemical Parasitology*, *59*(1), 41–48. https://doi.org/10.1016/0166-6851(93)90005-I

Li, L., Stoeckert, C. J. Jr, & Roos, D. S. (2003). OrthoMCL: Identification of ortholog groups for eukaryotic genomes. *Genome Research*, *13*(9), 2178–2189. https://doi.org/10.1101/gr.1224503

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

Nguyen, L. T., Schmidt, H. A., von Haeseler, A., & Minh, B. Q. (2015). IQ-TREE: A fast and effective stochastic algorithm for estimating maximum-likelihood phylogenies. *Molecular Biology and Evolution*, *32*(1), 268–274. https://doi.org/10.1093/molbev/msu300

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://doi.org/10.1371/journal.pone.0009490

Salomaki, E., Eme, L., Brown, M. W., & Kolisko, M. (2020). Releasing uncurated datasets is essential for reproducible phylogenomics. *Nature Ecology and Evolution*, *4*(11), 1435–1437. https://doi.org/10.1038/s41559-020-01296-w

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

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

Tice, A. K., Žihala, D., Pánek, T., Jones, R. E., Salomaki, E. D., Nenarokov, S., Burki, F., Eliáš, M., Eme, L., Roger, A. J., Rokas, A., Shen, X. X., Strassert, J. F. H., Kolísko, M., & Brown, M. W. (2021). PhyloFisher: A phylogenomic package for resolving eukaryotic relationships. *PLoS Biology*, *19*(8), e3001365. https://doi.org/10.1371/journal.pbio.3001365

Wang, H-C., Minh, B. Q., Susko, E., & Roger, A. J. (2018). Modeling site heterogeneity with posterior mean site frequency profiles accelerates accurate phylogenomic estimation. *Systematic Biology*, *67*(2), 216–235. https://doi.org/10.1093/sysbio/syx068

Yang, Z. (1994). Maximum likelihood phylogenetic estimation from DNA sequences with variable rates over sites: Approximate methods. *Journal of Molecular Evolution*, *39*, 306–314. https://doi.org/10.1007/BF00160154

Internet Resources

https://thebrownlab.github.io/phylofisher-pages/

The most up-to-date documentation can be found here, the PhyloFisher GitHub Pages website