Chapter 4

Automated Phylogenetic Analysis Using Best Reciprocal BLAST

Erin R. Butterfield, James C. Abbott, and Mark C. Field

Abstract

Reconstruction of the evolutionary history of specific protein-coding genes is an essential component of the biological sciences toolkit and relies on identification of orthologs (a gene in different organisms related by vertical descent from a common ancestor and usually presumed to have the same or similar function) and paralogs (a gene related to another in the same organism by descent from a single ancestral gene which may, or may not, retain the same/similar function) across a range of taxa. While obviously essential for the reconstruction of evolutionary histories, ortholog identification is of importance for protein expression, modeling for drug discovery programs, identification of critical residues and other studies. Here we describe an automated system for searching for orthologs and paralogs in eukaryotic organisms. Unlike manual methods the system is fast, requiring minimal user input while still being highly configurable.

Keywords Phylogenetics, Sequence searching, Homology, Automation, Evolution, Ortholog, Drug discovery

1 Introduction

Orthologs are genes derived from the same ancestor before speciation occurred and are often assumed to have the same function—although this is debated (Fig. 4.1a), whereas paralogs are genes which duplicated within a genome and may have distinct functions (Fig. 4.1b) [1]. Ortholog identification is required for many aspects of biological research including evolutionary analysis, drug discovery pipelines, and the study of proteins of unknown function [1–3]. Prediction of orthologs frequently relies on utilizing either precalculated databases (e.g., EggNOG [4], TriTrypDB [5]), command-line tools (e.g., OrthoFinder [6]) or manual searching, all of which have varying limitations including dataset size, sampling limitations, and difficulty identifying divergent orthologs as increases in ortholog prediction coverage can result in a decrease in prediction accuracy [7]. While manual searching provides great

Luis M. de Pablos and Javier Sotillo (eds.), *Parasite Genomics: Methods and Protocols* , Methods in Molecular Biology, vol. 2369, https://doi.org/10.1007/978-1-0716-1681-9_4,

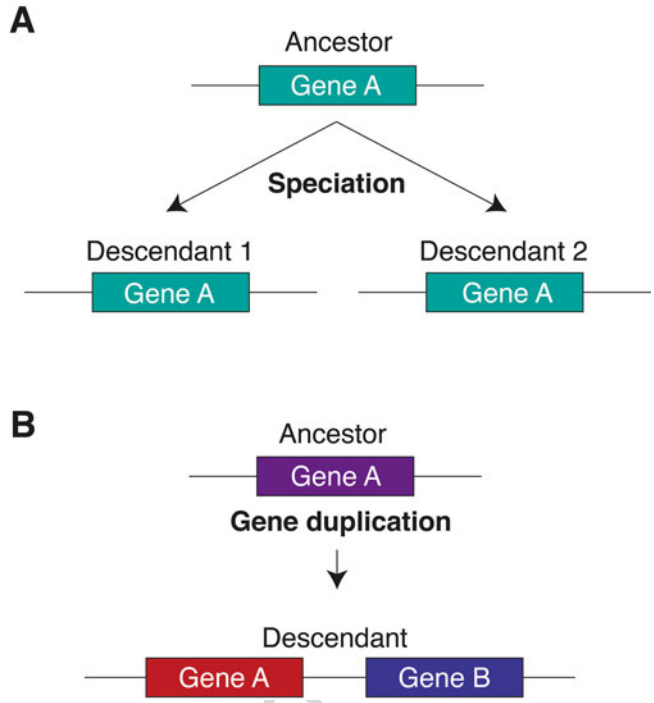

Fig. 4.1 Orthologs vs. Paralogs. (**a**) Shows the rise of orthologs, (**b**): Shows the rise of paralogs

flexibility in ortholog identification which can overcome prediction coverage and accuracy issues, it is generally unsuitable for the analysis of more than a few targets. We present here a protocol for the identification of orthologs based upon manual searches but utilizing scripts for data handling thereby decreasing the time required for manual searching while still maintaining user-oversight and flexibility. The pipeline is designed to enable maximum ortholog identification and therefore requires user over-sight for the elimination of paralogs and spurious hits.

The protocol described below utilizes best reciprocal BLAST [7–9] to identify putative orthologs. Figure 4.2 shows an overview of the protocol. Sequences of interest are searched against a user-created database containing all the organisms in which the user is interested. SeqKit [10] is first used to validate user sequences which are subjected to the required BLAST [8] type. Hits are extracted from these BLAST [8] results according to user stringency, based on their ranking per query per organism and the extent of contiguous homology with the query sequence. Hits with identical results are not included towards the hit count. Hits are used in a reciprocal BLAST [7–9] against the organism of the initial queries and resulting hits are filtered using the same criteria (rank and query coverage). If the initial sequence of interest is identified within the filtered hits the result is considered an ortholog of the query.

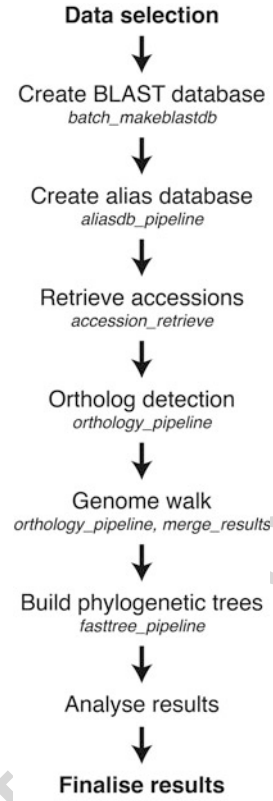

Fig. 4.2 Overview of the analysis pipeline. Italics indicates the commands to be run at each step

Depending on conservation, sequences may not detect potential orthologs across all organisms in the organism database and can require "genome walking," i.e., where orthology results from one organism are used for ortholog searching and the results of the two outputs (the original query and the new organism) are collated into a single output. A user can perform as many rounds of "genome walking" as required to achieve saturation. Following this the sequences are aligned using MUSCLE [11], edited using alncut [12] and a phylogenetic tree is built using FastTree [13]. Analysis of the tree is performed by the user and can be used to eliminate spurious hits and/or paralogs. Throughout this protocol, unless indicated otherwise, spaces should not be included in parameters and names should not start with numbers.

2 Materials

The following software are required:

1. Linux/Unix-like operating system (*see* **Note 1**).
2. Miniconda (*see* **Note 2**).
3. Bioconda [14] (*see* **Note 3**).
4. Alignment viewer (e.g., Jalview [15]) (*see* **Note 4**).
5. Phylogenetic tree viewer (e.g., FigTree) (*see* **Note 5**).

2.1 Installation of batch_brb

The pipeline is provided as a Bioconda [14] package, batch_brb which handles the installation of all dependencies (*see* **Note 6**).

1. Open Terminal and change into a directory where batch_brb should be located using the command below where **path** is the path to the directory (e.g., /Users/**user**/Documents where **user** is the username)
   *cd **path***
2. Install batch_brb and dependencies into a new Conda environment using the following command (*see* **Note 7**)
   conda create -n batch_brb batch_brb
3. Activate Conda environment using the following command (*see* **Note 8**)
   conda activate batch_brb
4. Run the setup script using the following command
   batch_brb_setup
5. Change into the batch_brb folder using the following command
   cd batch_brb
6. There should be four directories; **databases**, **jobs**, **templates**, and **documentation** which can be seen if the following command is used:
   ls

   databases is the folder where all the databases the user creates will be located. **jobs** is the folder where jobs are run and results can be found, **templates** contains all the templates required for this pipeline as both Excel (in Excel_files directory) and CSV files (in CSVs directory). **documentation** contains the documentation files and can be used as reference with this protocol.

3 Methods

3.1 Setup

This section does not need to be performed if progressing immediately from installation. These steps will only need to be performed for subsequent uses.

1. Open Terminal if not already opened.
2. If the batch_brb environment has not already been activated, activate the batch_brb Conda environment with the following command (*see* **Notes 7 and 8**)
   conda activate batch_brb
3. Change into the batch_brb directory using the below command where **path** is the path to the batch_brb directory
   *cd **path***

3.2 Data Selection

A phylogenetic analysis requires query sequences, i.e., sequences for which ortholog identity is sought and sequence data for the organisms the user is interested in. This pipeline requires all organism data be of one type, i.e., exclusively protein or nucleotide, where nucleotide must be a predicted transcriptome, coding sequence (CDS) or transcriptome data. For genome analysis, we refer the reader to [16, 17].

1. Decide which organisms to include in the analysis (*see* **Note 9**).
2. Decide which data type to use (*see* **Note 10**).
3. Download the fasta file of organism data from a repository (e.g., UniProt [18], the National Center for Biotechnology Information (NCBI) [19], Ensembl [20], EuPathDB [21], Joint Genome Institute (JGI) [22]) (*see* **Notes 11 and 12**). The user should have one fasta file per organism or strain to search.
4. Decompress file if required (*see* **Note 13**).

3.3 Create BLAST Database

To be able to search the organism sequence data, the data needs to first be converted into a BLAST database [8].

1. Place the fasta files for all organisms in the databases folder.
2. Fill in the 01_batch_makeblastdb_template.xlsx form located in templates/Excel_files folder.
   (a) **infile:** should contain the name of the fasta file (include the extension) (*see* **Notes 14 and 15**).
   (b) **db:** can be left blank (this refers to an SQLite3 database used by the software, a specific SQLite3 database can be provided by the user if required, otherwise please leave blank) (*see* **Note 15**).
   (c) Save the file as a CSV (*see* **Note 16**).
2. Place the CSV in the databases folder.
3. Change into the databases folder using the following command
   cd databases
4. Run the following command where **csv.csv** is the name of your csv file with the extension (*see* **Note 17**).

Table 4.1
Examples of batch_makeblastdb file outputs

Infile	Data type	Output file	Description	How to reference
Hsapiens.fasta	Protein	Hsapiens_database.pdb	Database files	**If _database required:** Hsapiens_database **If _database NOT required:** Hsapiens
		Hsapiens_database.phr		
		Hsapiens_database.pin		
		Hsapiens_database.pog		
		Hsapiens_database.pos		
		Hsapiens_database.pot		
		Hsapiens_database.psq		
		Hsapiens_database.ptf		
		Hsapiens_database.pto		
		Hsapiens_converted.fasta	Input fasta with converted headers	
Tborreli.fasta	Nucleotide	Tborreli_database.ndb	Database files	**If _database required:** Tborreli_database **If _database NOT required:** Tborreli
		Tborreli_database.nhr		
		Tborreli_database.nin		
		Tborreli_database.nog		
		Tborreli_database.nor		
		Tborreli_database.not		
		Tborreli_database.nsq		
		Tborreli_database.ntf		
		Tborreli_database.nto		

(continued)

Table 4.1 (continued)

Infile	Data type	Output file	Description	How to reference
		Tborreli_converted.fasta	Input fasta with converted headers	

batch_makeblastdb -csv ***csv.csv***

5. If the batch_makeblastdb command has executed successfully, there will be several new files present for each input file which end in _database (Table 4.1 for examples) and a single file which ends _converted.fasta (*see* **Notes 18 and 19**).

3.4 Create an Alias Database

To simplify searching, all organisms to be searched can be combined into a single alias database. This is a database that can include as many organisms as required provided; they are the same datatype (i.e., either all nucleotide or all protein).

1. Create a plain text file list of the names of all the databases to include, include _database in the database name (*see* **Notes 20 and 21**, Table 4.1).
2. Fill in the 02_make_aliasdb_template.xlsx located in templates/Excel_files folder.
   (a) **dblist_file:** Name of text file created in **step 1** of this section containing the names of all databases to include in the alias database (include the file extension (i.e., .txt)) (*see* **Note 15**).
   (b) **dbtype:** Either prot or nucl depending on whether the data is protein or nucleotide, respectively.
   (c) **title:** A title describing the database (can include spaces), this is used for reference and information about the database—it is not the name of the new database.
   (d) **output:** A name for the database (This is what will be used for later steps) (*see* **Note 15**).
   (e) **SQLite3_db:** The name of the SQLite3 database—leave blank if left blank in Subheading 3.3.
   (f) Save the file as a CSV (*see* **Note 16**).
3. Place the plain text file created in **step 1** and the CSV file in the databases folder.
4. Run the following command where **csv.csv** is the name of the CSV file with the extension (*see* **Note 22**).
   aliasdb_pipeline -csv ***csv.csv***
5. Successful completion of this pipeline will result in the creation of a new file called either **out.pal** or **out.nal** depending on

whether the file is protein or nucleotide data, respectively, where **out** is the name of the alias database from **step 2d** (*see* **Note 23**).

3.5 Retrieve Accessions

batch_brb adds a unique file identifier to all accessions during the creation of the BLAST database [8] in Subheading 3.3 which are stored in the SQLite3 database. In order to use batch_brb, the specific user accessions need to be retrieved from the SQLite3 database.

1. Create a fasta file which contains all the sequences of interest to retrieve from the same organism.
2. Fill in the 03_accession_retrieve_template.xlsx located in the templates/Excel_files folder.
   (a) **Fasta_file:** Name of the fasta file created in **step 1** of this section (include extension) (*see* **Note 15**).
   (b) **job_name:** Provide a name for the job (*see* **Note 15**).
   (c) **BLAST_database_name:** Name of the BLAST database [8] to retrieve the accessions from (do not include _database in the name (Table 4.1)—this will be automatically added as part of the workflow), this should be an organism database created in Subheading 3.3.
   (d) **SQLite3_db:** The name of the SQLite3 database—leave blank if left blank in Subheading 3.3.
   (e) **Evalue:** An Evalue cut-off to use for BLAST [8] (Optional argument, default value = 0.1) (*see* **Note 24**).
   (f) **max:** Maximum number of sequences to retrieve for BLAST [8] (Optional argument, default value = 5) (*see* **Note 24**).
   (g) **num_threads:** Number of cores to use for BLAST [8] (Optional argument, default value = 4) (*see* **Note 24**).
   (h) Save the file as a CSV (*see* **Note 16**).
3. Place the fasta file and the CSV file into the jobs folder.
4. Within terminal change into the jobs folder:
   cd ../jobs
5. Run the following command where **csv.csv** is the name of your csv file with the extension (*see* **Note 24**).
   accession_retrieve -csv ***csv.csv***
6. Once the pipeline is completed, within the jobs folder should be a new folder labeled **timestamp_job_name**, where timestamp is the time the pipeline was run and job_name is the name specified in **step 2b** of this section. Within the folder will be a file labeled **job_name_accessions.csv**, this file will contain the results of the accession retrieve. The input fasta file from

step 1 will also be located in this folder. If the pipeline was unable to retrieve the relevant accessions, there will be a file labeled **job_name_missing_BLAST_default.txt** which contains the BLAST [8] results in the default format (similar to online BLAST [8]) of the missing accessions. These results are also present in the **job_name_missing_BLAST.txt** file in the tabular format from BLAST [8] (*see* **Notes 25 and 26**).

3.6 Identify Putative Orthologs

1. Create a plain text file and list all accessions of interest from the same organism retrieved in the previous section (each organism needs their own text file) (*see* **Notes 27 and 28**).
2. Fill in the 04_orthology_pipeline_form_template.xlsx form located in the templates/Excel_files folder.
   (a) **Job_name:** provide a name for the job (*see* **Note 15**).
   (b) **Accession_list:** name of the text file that contains your list of accessions (include the extension) (*see* **Note 15**).
   (c) **FB_database:** the name of the database to search—usually the alias database created in Subheading 3.4. Do not include the database extension (e.g., do not include .pal or .nal if a protein or nucleotide alias database) (*see* **Note 29**).
   (d) **RB_database:** the name of the database the queries are from (this should only contain a single organism), do not include _database in the name (Table 4.1).
   (e) **Evalue:** An Evalue cut-off to use for BLAST (Optional argument, default value = 0.1).
   (f) **Hits:** Number of hits per organism per query to use for the orthology calculation, e.g., a value of 5 would mean the top 5 hits per organism per query are tested for orthology (*see* **Note 30**).
   (g) **Coverage:** The percentage of coverage of the query sequence, used for orthology calculation and is an integer (*see* **Note 31**).
   (h) **SQLite3_db:** The name of the SQLite3 database—leave blank if left blank in Subheading 3.3.
   (i) **Num_threads:** Number of cores to use for BLAST [8] (Optional argument, default value = 4).
   (j) **Max:** The total number of hits per query that BLAST [8] will retrieve (Optional argument, default = 150) (*see* **Note 32**).
   (k) **Trees (y/n):** Optional argument, default = n. y will launch the fasttree_pipeline and generate trees for all the queries submitted in the accession list provided more than three putative orthologs are identified.

(l) **Frequency:** Editing frequency for the fasttree_pipeline. Optional argument, default = 0.25. A value of 0.25 enables gaps to be present in 25% of sequences at a given residue. If this value or the trees or model fields are completed, the fasttree_pipeline will be performed. This field can be left blank (but if the trees and model fields are blank as well, no trees will be calculated) (*see* **Note 33**).

(m) **Model:** model to use for phylogenetic analysis. Optional argument, default is JTT [23] for protein and Jukes-Cantor (JC) [24] for nucleotide. lg [25] or wag [26] can be specified for protein and gtr can be specified for nucleotide [13] (*see* **Note 34**).

(n) Save the file as a CSV (*see* **Note 16**).

3. Transfer the text file and the CSV file into the jobs folder.

4. Run the following command where **csv.csv** is the name of the CSV file with the extension (*see* **Note 35**).
orthology_pipeline -csv ***csv.csv***

5. If the pipeline completes successfully, a folder will be located in jobs labeled **timestamp_job_name** where job_name is the job name provided in **step 2a**. Within the folder will be a file labeled **job_name_orthologs.csv**, this contains the predicted ortholog results where the first column contains the initial query accessions and each subsequent column contains the results for each organism in the search database. Within the folder **intermediary_files** are all the results produced through the pipeline, e.g., the first BLAST [8] results, etc. An additional folder labeled **Trees** will contain all the files produced from the fasttree_pipeline if this has been selected. Within this folder will be a folder for each initial query accession. Within these will be **accession_aln.fasta** (aligned fasta file), **accession_edited.fasta** (edited aligned fasta file), **accession_single.fasta** (unaligned fasta file with duplicate sequences removed), **accession_tree** (tree file), **accession.fasta** (unaligned fasta file, may contain duplicate sequences), and **accession.txt** (accession list of putative orthologs) (*see* **Note 36**).

3.7 Genome Walk

When analyzing divergent sequences, it is not always possible to detect orthologs across the eukaryotic tree as the high divergence can eliminate detection. However, it is often possible to "Genome walk" across the tree, i.e., take ortholog predictions from a closer relative and use these as queries for ortholog prediction. Multiple rounds of this can enable ortholog detection across the eukaryotic tree.

1. Choose another organism to use as queries in ortholog prediction.
2. Copy all the accessions for this organism into a plain text file ensuring each accession is on a new line and remove any commas (“,”).
3. Save the file.
4. Fill in the 04_orthology_pipeline_form_template.xlsx located in the templates/Excel_files folder as described in Subheading 3.6.
5. Save the file as a CSV (*see* **Note 16**).
6. Transfer the text file and the CSV file into the jobs folder.
7. Run the following command where **csv.csv** is the name of the csv file with the extension.
   *orthology_pipeline -csv **csv.csv***
8. Once completed, transfer the new orthology results CSV (orthology results from **step 7** of this section) and the original orthology results CSV (orthology results from **step 4** of Subheading 3.6) into the jobs folder.
9. Run the following command where **in1** is the original orthology results (**step 4** of Subheading 3.6), **in2** is the new orthology results (**step 7** of this Subheading 3.7) and **output** is the name of the merged file to produce (*see* **Note 37**).
   *merge_results **in1 in2 output***
10. Successful completion of the merge_results script will result in the creation of a file labeled **output**.csv (where output is what was specified in **step 9**) which will contain the merged results. This can be used for further rounds of genome walking if required.
11. Repeat **steps 1–9** as many times as required but use the merged file from **step 10** as **in1** when performing the merge step in **step 9**.

3.8 Phylogenetic Trees

Phylogenetic trees are used to validate orthology predictions. There are many different algorithms for phylogenetic analysis. For high-throughput analysis, this pipeline utilizes FastTree due to its robust predictions and low computational time [13, 27]. However, under some conditions FastTree can be less accurate than other phylogenetic algorithms [13, 27, 28] which is why additional phylogenetic analyses are performed when finalizing results.

1. Within the jobs folder run the following command to make a new directory where **name** is the name of the directory to create.
   mkdir ***name***
2. Use the following command to change into the new directory where **name** is the name of the new directory.
   cd ***name***
3. Move the merged orthology results CSV file created in **step 10** of Subheading 3.7 into the new directory.
4. Run the following command where **alias_db** is the name of the alias database used for orthology searching and **csv.csv** is the name of the CSV from **step 3** (*see* **Notes 35** (end), **38–41**)
   fasttree_pipeline -db ***alias_db*** *-csv* ***csv.csv***
5. Successful completion of the fasttree_pipeline will result in a directory for each query being created. For a description of the files within each of these directories, *see* Subheading 3.6 **step 5** (end) (*see* **Note 42**).

3.9 Analyze Results

1. Open the _tree file in a tree viewer such as FigTree.
2. If the tree contains an outgroup, route the tree on the outgroup, if not root the tree on the midpoint (*see* **Note 43**). In FigTree: For outgroup, select outgroup branch > Reroot, For midpoint, Trees > Root tree > Midpoint
3. Show the local support values. In FigTree: Node Labels > Display: label
4. When analyzing the tree:
   (a) Look at the node support values (*see* **Note 44**).
   (b) How does the topology compare to the species tree? (*see* Fig. 4.3a–d and **Note 45**)
   (c) Is long branch attraction present? (*see* Fig. 4.3e and **Note 46**)
   (d) Compare the tree to the alignment. Is there anything in the alignment which could explain discrepancies in the tree? (*see* **Note 47**)
5. Open the _aln.fasta and _edited.fasta files in an alignment viewer such as Jalview [15].
6. Color the residues to show conservation. In Jalview [15]: Color > Clustalx
7. When analyzing the alignment:
   (a) Is the conservation strong or poor? (*see* **Note 48**)
   (b) Does the conservation cover the sequence or is it restricted to a domain region only?

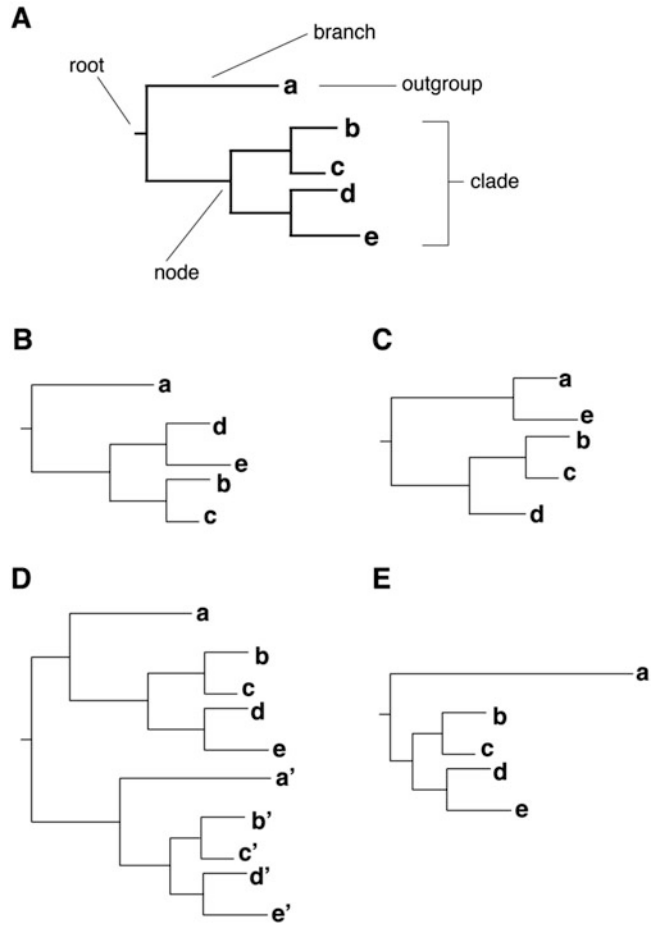

Fig. 4.3 Examples of phylogenetic trees. (**a**) Represents the species tree and is labeled to show the different components of a phylogenetic tree. (**b**) An example of rotation of branches within nodes which does not change the tree topology, while they look different, Tree A and Tree B are the same. (**c**) Rearrangement of branches, species "e" now groups with the outgroup "a". (**d**) An example of the presence of paralogs in the same tree. (**e**) Species "a" is an example of a long branch

(c) How does the edited alignment compare to the unedited alignment? Is there still good representation of the entire protein, or has the alignment been trimmed to a specific domain only?

(d) Investigate any sequences which are significantly shorter than the rest of the alignment in the edited alignment (*see* **Note 49**).

8. If sequences need to be removed from the alignment, the tree will need to be recomputed.

(a) Remove required sequences from the unaligned fasta files and save.

(b) Within the jobs folder run the following command to make a new directory where **name** is the name of the directory to create
mkdir ***name***

(c) Use the following command to change into the new directory where **name** is the name of the new directory.
cd ***name***

(d) Move the new unaligned fasta files into the new directory

(e) Run the following command (*see* **Notes 38–41**).
fasttree_pipeline

9. Repeat **steps 1–8** as many times as required until the user is satisfied; no false hits are present.

3.10 Finalize and Validate Results

Orthology results should be validated through multiple methods where possible to increase confidence. At the very least results should be submitted to other phylogenetic algorithms, e.g., PhyML [29] and MrBayes [30]. It is also useful to compare domain predictions and domain organization between candidate orthologs. For proteins performing the same function, it would be expected AU3 that they have the same (or highly similar) domain predictions and organizations: however, divergence can sometimes lead to a failure to predict a domain. Domain predictions can also be useful for identifying gain or loss of features. It is also useful to compare orthogonal data, e.g., localization of different orthologs (ideally experimentally confirmed rather than predicted). Orthogonal approaches for validation can increase confidence in functional orthology results particularly if conservation is poor between orthologs.

4 Notes

1. This pipeline has been tested on MacOS Mojave 10.14.3, MacOS Catalina 10.15.7, CentOS 6 and Red Hat Enterprise Linux 7.
2. Miniconda can be downloaded from Anaconda: https://docs.conda.io/en/latest/miniconda.html
3. Bioconda [14] installation instructions can be found here: https://bioconda.github.io/user/install.html
   If Miniconda is already installed, then only the channels should need to be setup.
4. Jalview [15] can be downloaded from: http://www.jalview.org/getdown/release/ or installed using Bioconda [14].

5. FigTree can be downloaded from: https://github.com/rambaut/figtree/releases or installed using Bioconda [14].
6. batch_brb can be installed from Bioconda [14], source code is available on GitHub:
   https://github.com/erin-r-butterfield/batch_brb and Zenodo: https://doi.org/10.5281/zenodo.4282534
7. Depending on the operating system Conda may first need to be activated using the following command where **path_to_conda** is the path where Conda is installed.
   *source /**path_to_conda**/bin/activate*

   Once Conda is activated "(base)" should be visible next to the prompt.
8. Once activated the environment name "*(batch_brb)*" should be visible next to the prompt.
9. Several considerations should be made when choosing organisms to include. Firstly, is the goal to analyze a specific lineage or pan-eukaryotic? Secondly, even sampling across the desired dataset is important and include at least two organisms from each group, this can give confidence to negative results. Thirdly, understand the quality of the datasets and specifically if the entire coding complement is adequately covered [9]. Finally, how divergent are the genes of interest? A large search database can compromise identification of divergent candidates due to the Expect value (Evalue) being dependent on database size and requiring increasing levels of conservation to secure a significant result [31].
10. Protein sequences enable detection of greater divergence due to codon degeneracy and the increase in character states (20 vs 4) decreasing random matching [9].
11. Genome annotations can change; therefore, it is important to log information regarding data source, relevant publications, date of download, and data type. This information will also be required for publication.
12. Valid input fasta file header formats:
    >..|accession information (e.g., UniProt [18])
    >ENA|accession|information (e.g., the European Nucleotide Archive (ENA) [32])
    >jgi|organism|accession (e.g., JGI [22])
    >...|accession information
    >accession information (e.g., NCBI [19], Ensembl [20])
    >accession | information (e.g., EuPathDB [21])
    provided the accession is less than or equal to 44 characters, if greater than this the user will need to truncate the accession.
13. Files with the extension .tar.gz can be decompressed using the following command where **file** is the filename.

*tar -zxvf **file**.tar.gz*

14. If all files in the databases folder need to be converted to BLAST databases, the **ls** command can be used to get all the names and the results can be copied into the **infile** column.
*ls **

15. Do not include spaces (use _ instead, replace in the filename as well) and do not start with a number.

16. If using MacOS Numbers, ensure to remove columns which contain no headings or data before converting to CSV.

17. The batch_makeblastdb command first checks a valid fasta file has been submitted and which data type (protein or nucleotide). The headers of the sequences in the fasta file are converted into the required format for the pipeline and a five character code followed by an underscore is appended as a prefix to the accessions. A single code is used for all sequences in the fasta file. The details about the file, unique code, and all the accessions are added to the SQLite3 database and the fasta file is converted into a BLAST [8] database using the BLAST makeblastdb command [33].

18. Some common errors:
    (a) **Names are already present in database**; ensure the file has not already been used. If it has and it needs to be replaced, the **delete_db** pipeline can be used (*see* **Note 19**); otherwise, change the required names.
    (b) **lock file detected, sleeping**...; This occurs most frequently if the batch_makeblastdb command was terminated prematurely by the user. Provided no other script is running at the time (this is important if many users are using the software at once (e.g., in a cluster environment) as the lock_file is to protect the SQLite3 database from corruption), end the current script (*Ctrl* + *c*), delete the lock_file (*rm lock_file*), and repeat the script.
    (c) **BLAST Database creation error: Near line x, the local id is too long. Its length is y but the maximum allowed local id length is 50. Please find and correct all local ids that are too long;** The accessions are too long in the input fasta file (they must be less than or equal to 44 characters due to the addition of a unique id), please truncate the accession.

19. To delete a database, fill in the delete_database_template.xlsx located in the templates/Excel_files folder:
    (a) **BLAST_db**; Name of the BLAST [8] database to delete (do not include _database in the name (Table 4.1)).
    (b) **SQLite3_db**; Name of SQLite3 database, leave blank if left blank in Subheading 3.3.

(c) Save as a CSV into the databases folder (*see* **Note 16**).

(d) Change into the databases folder.
cd ../databases

(e) Run the following command where **csv.csv** is the name of the CSV file.
delete_db -csv ***csv.csv***

This will delete the database and remove information about this database from the SQLite3 database.

20. Ensure all database names end in _database (Table 4.1); a simple way to do this is to run the following command where **out** is the name of the output to create provided all databases in the databases folder are to be included.
*ls *_converted.fasta | sed 's/_converted.fasta/_database/g' > **out**.txt*

21. Ensure to include the organism the sequences of interest come from in the alias database, this will provide useful information regarding possible paralogs and also ensure these sequences are included in the fasttree_pipeline later.

22. This script will check the new alias database has not previously been used, create an alias database using the BLAST blastdb_aliastool [33], and add the newly formed database information to the SQLite3 database.

23. Some common errors:
    (a) **BLAST database creation error: BLASTDB alias file creation failed. Some referenced files may be missing**; ensure no errors occurred during the batch_makeblastdb pipeline in Subheading 3.3, ensure _database (Table 4.1) has been added to the end of all database names.
    (b) **Names are already present in database**; *see* **Notes 18a and 19.**
    (c) **lock file detected, sleeping**. . .; *see* **Note 18b.**

24. The accession_retrieve pipeline determines the md5 checksum for the input sequences and compares these to the md5 checksums in the SQLite3 database. Any md5 checksums that match will be retrieved. These are filtered to those present in the BLAST [8] database specified by the user (this is why a single organism database should be used for retrieval, rather than an alias database). For sequences where the relevant accession cannot be retrieved, the pipeline will run BLAST [8] so the user can determine which is the relevant accession.

25. The output of the BLAST [8] results in the tabular format are: query id (qseqid), subject id (sseqid), percentage identity (pident), length, number of mismatches (mismatch), number of gap openings (gapopen), alignment start for query sequence

(qstart), alignment end for query sequence (qend), alignment start for subject sequence (sstart), alignment end for subject sequence (send), Expect value (evalue), bitscore, number of gaps (gaps), total query coverage (qcovs), query coverage per high scoring pair (qcovhsp), query length (qlen), and subject length (slen) [34]. Column names are not provided, names in brackets refer to identifiers from [34].

26. Common errors:
    (a) The accessions retrieved are not from the organism of the sequences of interest; this occurs because the sequences are identical. The user can ensure the correct organism accession is retrieved by listing the specific organism database rather than an alias database.
    (b) **Error: Cannot find Fasta_file**; Ensure fasta file is located in jobs directory (it will have moved if the job is being rerun), ensure file name is correctly spelt and contains extension.
27. Each accession should be on a new line. Commas (",") should be removed (if using results from previous ortholog predictions).
28. Accessions need to be the version stored within the SQLite3 database. For the first round of orthology, searching these accessions will need to be retrieved from the SQLite3 database using the accession_retrieve pipeline in Subheading 3.5. For genome walking (Subheading 3.7), the accessions listed in the orthology results from Subheading 3.6 will be the SQLite3 database accessions, so these will not need to be retrieved.
29. If not using an alias database, include _database in the database name (Table 4.1).
30. It is advisable to keep this number low, as the greater the number the greater the false-positive rate. A value of 5 is normally used for divergent sequences. A value of 2 or 3 is normally used for more conserved sequences.
31. The lower the value, the greater the false-positive rate. A value of 30 is normally used for divergent sequences. A value of 50 is normally used for more conserved sequences.
32. If searching a database with many organisms, this may need to be increased.
33. Alignments are edited before performing phylogenetic analyses. While gaps can be useful when looking at alignments, i.e., for the identification of indels, poorly conserved regions of an alignment can decrease the alignment signal-to-noise ratio which can affect tree quality although this can be less of an issue for short alignments [35].
34. If specifying a model, ensure to use lower case.

35. The orthology_pipeline retrieves sequences from the organism database for the accessions provided and performs a BLAST [8] against the search database. Results are filtered to take the top x hits per organism per query where x is user supplied (identical results do not count towards the hit count). These are further filtered to those hits which cover y percentage of the query where y is user supplied. The sequences for this list of hits are retrieved and BLAST [8] is performed against the organism database. These results are filtered to take the top x hits per query where x is user supplied (identical results do not count towards the hit count). These are further filtered to those which cover y percentage of the query where y is user supplied. The two results sets are compared, where they match the hits are considered an ortholog (i.e., if A detects B in first BLAST [8] and B detects A in reverse BLAST [8], then A and B are called orthologs). If the tree pipeline is selected, the sequences of all predicted orthologs per query will be retrieved, aligned using MUSCLE [11], edited using alncut [12] and a FastTree [13] phylogenetic tree will be built.

36. Some common errors:
    (a) **Error: [blastdbcmd] Entry not found:** ***accession***

    Error: [blastdbcmd] Entry or entries not found in BLAST database; ensure the SQLite3 database accessions are used, obtained with the accession_retrieve pipeline (Subheading 3.5).

    (b) **Can't find (first/reverse) BLAST database**; ensure the database names are correct.

    (c) **mv: rename Accession_list to ../Accession_list: No such file or directory**; ensure Accession_list (from **step 2b**) contains the file extension.

    (d) Didn't get the anticipated results; stringency may be too high, lower the alignment coverage and potentially increase the hit number. Alternatively, perform "genome walking" (Subheading 3.7).

37. merge_results will combine the results of two orthology searches. The script will map the query accessions from the round of genome walking back to the organism results in the first orthology search to determine the query accessions of the original orthology search. This mapping is used to add the results of the genome walk. Instances where the result for an organism is detected by two different queries will result in the new results being added to both queries, i.e., if A and B from the first orthology results both predict C as an ortholog then the orthology results using C as query will be mapped back to both A and B.

38. Editing frequency can be altered by using the below argument where **frequency** is a value between 0 and 1. The default value is 0.25 which enables gaps to be present in 25% of sequences at a given residue.
    -f frequency

39. The model can be altered by using the below argument where **model** is either lg [25] or wag [26] for protein or gtr for nucleotide [13]. The default is JTT [23] for protein and JC [24] for nucleotide.
    -m model

40. This script requires there be greater than three sequences present for each query in order to build an alignment and tree.

41. This script also works with fasta files or text files of accession lists. For text files the **-db** argument is required. The **-csv** argument is not required for either text or fasta files. Each file will produce its own alignment and tree, so only include the accessions/sequences that should appear on the same tree.

42. Some common errors:
    (a) **Database required if text files supplied**; the **-db** argument is required for both text files and a CSV file, ensure this has been included.
    (b) **Not enough sequences to build tree**; this pipeline requires there to be at least three sequences for a tree to be built.
    (c) **Error: Inappropriate model type chosen**. . .; This occurs if a nucleotide model was chosen for protein sequences or vice versa. If this occurs, the default algorithm will be selected (JTT [23] for protein, JC [24] for nucleotide).
    (d) The pipeline is making folders/ trees, etc. of everything in the jobs folder; ensure a new folder is created which contains only the files the user wants to use to build a tree. Make sure the terminal is within this folder before launching fasttree_pipeline

43. Where possible the tree should be rooted by an outgroup—this can either be an organism that is known to be at the base of the tree or alternatively a gene sequence which shares ancestry with the gene of interest. Where this is not possible trees can be rooted by midpoint, this takes the two longest branches of the tree and places the root at the midpoint between. Assuming equal evolutionary rates across the tree, this will normally result in the tree following the species tree [7].

44. The closer the value is to 1, the more confident the topology. Nodes with values $\geq$0.95 show strong support, 0.85–0.94 show good support, 0.75–0.84 show moderate support and $\leq$0.74 show poor support.

45. While the tree topology will not always follow the species tree, it can be used to give an indication as to whether false positives or paralogs have been included [7]. Figure 4.3 demonstrates various examples of how the tree can differ from the species tree (A). Branches within a node are of equal distance from other branches within the same node. Therefore, rotation of branches within a node does not change how the tree reads, i.e., Tree A and Tree B in Fig. 4.3 are the same as "d" and "e" are the same evolutionary distance from "b" and "c". In Fig. 4.3c, species "e" has moved to group with species "a", this may suggest that "e" is a mishit as the sequence is now grouping with the outgroup—however, this requires further investigation to ensure there are no features in the alignment which could support this move and knowing whether these organisms "normally" attract each other when building phylogenetic trees. Figure 4.3d represents paralog presence on the tree, further investigation is required to determine whether these sequences should continue to be considered orthologous or should be split into two separate trees.
46. Sequences which evolve rapidly relative to other sequences within the tree will be given long branches (Fig. 4.3e species "a" is an example of a long branch) and can be falsely grouped together. This is referred to as long branch attraction. These sequences can be removed individually to test their placement, if they move around it suggests long branch attraction. There are many methods for dealing with long branch attraction including removal of these sequences, increased taxa sampling, and selection of a more appropriate evolutionary model [36, 37].
47. Topology of the tree should be checked against the alignment to:
    (a) Ensure the alignment is of sufficient quality to build the tree.
    (b) Ensure there are no features of the alignment which could explain oddities in the tree (e.g., short sequences, lack of domains etc.).
48. Tree building algorithms require a phylogenetic signal to be able to build a tree. Alignments with poor conservation have a high signal-to-noise ratio and can produce poor quality trees. Equally, alignments with too much conservation can also produce poor quality trees due to the lack of phylogenetic signal. Applying more or less stringent editing, respectively, can improve these [35].
49. The sequence origin should be investigated to determine whether the sequence is complete (i.e., ensure the entire sequence has been predicted). If the sequence is complete

and still considerably shorter than the remainder of the alignment, the sequence should usually be removed.

Acknowledgments

Development of the software described and provided here has been supported by a Wellcome Trust grant for establishing the Wellcome Centre for Anti-Infectives Research (*W*CAIR) at the University of Dundee (to Paul Wyatt (PI), MCF, and others). We thank Tim Butterfield, Frederik Dröst, and Michele Tinti for comments on the code. We also thank Ricardo Canavate del Pino and Ning Zhang for help with testing.

References

1. Stamboulian M, Guerrero RF, Hahn MW et al (2020) The ortholog conjecture revisited: the value of orthologs and paralogs in function prediction. Bioinformatics 36(Supplement_1): i219–i226. https://doi.org/10.1093/bioinformatics/btaa468
2. Baragaña B, Forte B, Choi R et al (2019) Lysyl-tRNA synthetase as a drug target in malaria and cryptosporidiosis. Proc Natl Acad Sci U S A 116(14):7015–7020. https://doi.org/10.1073/pnas.1814685116
3. Klinger CM, Ramirez-Macias I, Herman EK et al (2016) Resolving the homology—function relationship through comparative genomics of membrane-trafficking machinery and parasite cell biology. Mol Biochem Parasitol 209:88–103. https://doi.org/10.1016/j.molbiopara.2016.07.003
4. Huerta-Cepas J, Szklarczyk D, Forslund K et al (2016) EGGNOG 4.5: a hierarchical orthology framework with improved functional annotations for eukaryotic, prokaryotic and viral sequences. Nucleic Acids Res 44(D1): D286–D293. https://doi.org/10.1093/nar/gkv1248
5. Aslett M, Aurrecoechea C, Berriman M et al (2009) TriTrypDB: a functional genomic resource for the Trypanosomatidae. Nucleic Acids Res 38(Database issue):D457–D462. https://doi.org/10.1093/nar/gkp851
6. Emms DM, Kelly S (2015) OrthoFinder: solving fundamental biases in whole genome comparisons dramatically improves orthogroup inference accuracy. Genome Biol 16 (1):157–157. https://doi.org/10.1186/s13059-015-0721-2
7. Altenhoff AM, Glover NM, Dessimoz C (eds) (2019) Inferring orthology and paralogy (vol. 1910). Evolutionary genomics. Methods in molecular biology. Springer, New York
8. Altschul SF, Madden TL, Schäffer AA et al (1997) Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. Nucleic Acids Res 25(17):3389–3402. https://doi.org/10.1093/nar/25.17.3389
9. Klute MJ, Melançon P, Dacks JB (2011) Evolution and diversity of the Golgi. Cold Spring Harb Perspect Biol 3:a007849
10. Shen W, Le S, Li Y et al (2016) SeqKit: a cross-platform and ultrafast toolkit for FASTA/Q file manipulation. PLoS One 11(10):e0163962. https://doi.org/10.1371/journal.pone.0163962
11. Edgar RC (2004) MUSCLE: multiple sequence alignment with high accuracy and high throughput. Nucleic Acids Res 32 (5):1792–1797. https://doi.org/10.1093/nar/gkh340
12. Lawrence TJ, Kauffman KT, Amrine KCH et al (2015) FAST: FAST analysis of sequences toolbox. Front Genet 6:172. https://doi.org/10.3389/fgene.2015.00172
13. Price MN, Dehal PS, Arkin AP (2010) FastTree 2—approximately maximum-likelihood trees for large alignments. PLoS One 5(3): e9490. https://doi.org/10.1371/journal.pone.0009490
14. Grüning B, Dale R, Sjödin A et al (2018) Bioconda: sustainable and comprehensive software distribution for the life sciences. Nat Methods 15(7):475–476. https://doi.org/10.1038/s41592-018-0046-7
15. Waterhouse AM, Procter JB, Martin DM et al (2009) Jalview version 2—a multiple sequence alignment editor and analysis workbench.

Bioinformatics 25(9):1189–1191. https://doi.org/10.1093/bioinformatics/btp033

16. Barlow LD (2018) AMOEBAE. https://github.com/laelbarlow/amoebae
17. Larson RT, Dacks JB, Barlow LD (2019) Recent gene duplications dominate evolutionary dynamics of adaptor protein complex subunits in embryophytes. Traffic 20 (12):961–973. https://doi.org/10.1111/tra.12698
18. The UniProt Consortium (2019) UniProt: a worldwide hub of protein knowledge. Nucleic Acids Res 47(D1):D506–D515. https://doi.org/10.1093/nar/gky1049
19. NCBI Resource Coordinators (2018) Database resources of the National Center for Biotechnology Information. Nucleic Acids Res 46 (D1):D8–D13. https://doi.org/10.1093/nar/gkx1095
20. Yates AD, Achuthan P, Akanni W et al (2020) Ensembl 2020. Nucleic Acids Res 48(D1): D682–D688. https://doi.org/10.1093/nar/gkz966
21. Aurrecoechea C, Barreto A, Basenko EY et al (2017) EuPathDB: the eukaryotic pathogen genomics database resource. Nucleic Acids Res 45(D1):D581–D591. https://doi.org/10.1093/nar/gkw1105
22. Nordberg H, Cantor M, Dusheyko S et al (2014) The genome portal of the Department of Energy Joint Genome Institute: 2014 updates. Nucleic Acids Res 42(D1): D26–D31. https://doi.org/10.1093/nar/gkt1069
23. Jones DT, Taylor WR, Thornton JM (1992) The rapid generation of mutation data matrices from protein sequences. Bioinformatics 8 (3):275–282. https://doi.org/10.1093/bioinformatics/8.3.275
24. Jukes TH, Cantor CR (eds) (1969) Evolution of protein molecules, Mammalian protein metabolism, vol 3. Academic, New York
25. Le SQ, Gascuel O (2008) An improved general amino acid replacement matrix. Mol Biol Evol 25(7):1307–1320. https://doi.org/10.1093/molbev/msn067
26. Whelan S, Goldman N (2001) A general empirical model of protein evolution derived from multiple protein families using a maximum-likelihood approach. Mol Biol Evol 18(5):691–699. https://doi.org/10.1093/oxfordjournals.molbev.a003851
27. Liu K, Linder CR, Warnow T (2011) RAxML and FastTree: comparing two methods for large-scale maximum likelihood phylogeny estimation. PLoS One 6(11):e27731. https://doi.org/10.1371/journal.pone.0027731
28. Smirnov V, Warnow T (2021) Phylogeny estimation given sequence length heterogeneity. Syst Biol 70(2):268–282. https://doi.org/10.1093/sysbio/syaa058
29. Guindon S, Dufayard J-F, Lefort V et al (2010) New algorithms and methods to estimate maximum-likelihood phylogenies: assessing the performance of PhyML 3.0. Syst Biol 59 (3):307–321. https://doi.org/10.1093/sysbio/syq010
30. Ronquist F, Teslenko M, van der Mark P et al (2012) MrBayes 3.2: efficient Bayesian phylogenetic inference and model choice across a large model space. Syst Biol 61(3):539–542. https://doi.org/10.1093/sysbio/sys029
31. Kerfeld CA, Scott KM (2011) Using BLAST to teach "E-value-tionary" concepts. PLoS Biol 9 (2):e1001014. https://doi.org/10.1371/journal.pbio.1001014
32. Amid C, Alako BTF, Balavenkataraman Kadhirvelu V et al (2020) The European nucleotide archive in 2019. Nucleic Acids Res 48(D1): D70–D76. https://doi.org/10.1093/nar/gkz1063
33. Camacho C, Coulouris G, Avagyan V et al (2009) BLAST+: architecture and applications. BMC Bioinform 10:421. https://doi.org/10.1186/1471-2105-10-421
34. Bethesda (MD): National Center for Biotechnology Information (US) (2008) Appendices. https://www.ncbi.nlm.nih.gov/books/NBK279684/
35. Talavera G, Castresana J (2007) Improvement of phylogenies after removing divergent and ambiguously aligned blocks from protein sequence alignments. Syst Biol 56 (4):564–577. https://doi.org/10.1080/10635150701472164
36. Brinkmann H, van der Giezen M, Zhou Y et al (2005) An empirical assessment of long-branch attraction artefacts in deep eukaryotic phylogenomics. Syst Biol 54(5):743–757. https://doi.org/10.1080/10635150500234609
37. Bergsten J (2005) A review of long-branch attraction. Cladistics 21(2):163–193