1 Workflow for estimating rooted species trees using ALE

The following section provides step-by-step instructions on using ALE and some other tools to root the small tree of life demonstration dataset. It is written with the intent that someone new to this area should be able to follow along. The workflow assumes a UNIX-based system (e.g. Linux or OS X), and shell commands are written in the `Consolas` font. Experienced users will likely have favourite tools for some of these steps and should feel free to diverge from the workflow at will. All of the scripts, files and example data discussed below are available at the Github repository associated with this chapter (https://github.com/ak-andromeda/ALE_methods). The workflow is summarised in Figure 4.

1.1 Preparing data for analysis

The input data should be descriptive enough to allow ALE to determine the genome of origin for each sequence in a gene family and also preferably be human readable. By default, ALE expects the text preceding an underscore "_" in a FASTA header to contain the name of the genome. Here we provide a script that formats genomes in an ALE-friendly way. Genomes should be first downloaded from repositories in a FASTA format. Genomes should be named in this fashion: "Sulfolobus.fa" for genus only or "Arabidopisis.thaliana.fa" for genus and species. The script requires one command line argument: a species list in the form of a text file, with the genome names on separate lines. Please emulate "species_list_demo.txt" to correctly format the file. To execute the script call:

```
$python3 rf_genome_for_mcl.py species_list_demo.txt
```

1.2 Inferring pairwise alignments

Gene families are sets of genes that are believed to share a common ancestry. The history of these families is not necessarily one of direct vertical inheritance, as in the case of orthologues, but can include gene duplications and lateral acquisition. In practice, users infer

gene families based on grouping together sets of significantly similar sequences. An 'all vs all' BLAST search can be undertaken to infer homologous gene families - diamond BLAST (Buchfink et al. 2015) can be used on large datasets for increased speed. To download diamond blast see: https://github.com/bbuchfink/diamond. To perform the diamond BLAST, combine all the genomes into a single FASTA file with the following shell command:

```
cat *.fa > ALE_demo_dataset.fasta
```

Next a diamond blast database needs to be constructed. To construct the non-redundant diamond blast database execute the following bash script:

```
$bash make_diamond_db.sh
```

It must be noted that the names of the files may vary if you are undertaking this on your own datasets - so please change the input file names in the python/bash scripts accordingly. Following the construction of the diamond blast database, the file used to make the database can then be used as the query to BLAST the database, allowing an all-versus-all analysis. To undertake the all-versus-all diamond BLAST, execute the following bash script:

```
$bash diamond.sh
```

1.3 Clustering genes using MCL

The output of the all-versus-all BLAST is a text file called 'AvA_demo.txt'. Gene families can then be clustered from the blast output E-value using the Markov Cluster Algorithm (MCL) [1].To download MCL see: https://micans.org/mcl/. First the results must be extracted from the AvA_demo.txt results file using the shell command:

```
cut -f 1,2,11 AvA_demo.txt > seq.abc
```

Next, use mcxload to first load the blast results into a readable file, with the following shell command:

```
mcxload -abc seq.abc --stream-mirror --stream-neg-log10 -stream-tf
'ceil(2002)' -o seq.mci -write-tab seq.tab
```

MCL can now be used to cluster the blast results into gene families with the following shell command:

```
mcl seq.mci -I 1.4 -use-tab seq.tab
```

The above step can be repeated using different inflation values (-I parameter) to adjust gene family cluster sizes. Increasing the inflation value increases the ‘granularity’ of the gene families (higher -I value, smaller gene family clusters).

The output of the MCL clustering is a single text file: 'out.seq.mci.I14', which indicates the members of each cluster (gene family), with one cluster per line. Individual gene family FASTA files for subsequent analysis need to be assembled based on this output. We provide a python3 script to undertake this task. The script requires two command line arguments: 1) The output from the MCL analysis ('out.seq.mci.I14'), 2) The dataset used to in the all-versus-all blast ('ALE_demo_dataset.fasta').

To execute the script:

```
$python3 make_gene_families_mcl.py out.seq.mci.I14
ALE_demo_dataset.fasta
```

1.4 Constructing gene family alignments

The majority of alignment softwares require a minimum sequence number of four per gene family. We provide a python3 script that identifies viable gene families for alignment - find_viable_gene_families.py. The script will create two directories, one with gene families that

have four or more sequences, and the other includes gene families with less than four sequences. First move into the 'MCL_gene_families' folder, then execute this script in the directory with the gene family FASTA files:

```
$python3 find_viable_gene_families.py
```

Even small datasets such as the one utilised in this demonstration can yield tens of thousands of gene families, resulting in linear computations taking considerable time; to improve the speed of computations, employ parallel programming software such as GNU parallel [2], available at: https://www.gnu.org/software/parallel/. The script produces two directories – change into the 'four_plus_seq' directory. Gene families can be aligned using software such as MAFFT [3], and poorly-aligned positions removed with BMGE [4]. To align the gene families with MAFFT, execute the following shell command in the 'four_plus_seq' directory.

```
ls *.fa | parallel mafft --auto {} "›" {}.aln
```

The command runs the application in auto mode on each file in the current directory that ends with .fasta. It will run multiple jobs at a time, up to the number of cores on the computer. (That is, it runs mafft --auto on each .fasta file, in parallel). The --auto mode will select the best fitting alignment model for that particular sequence. Alignments can then be trimmed using software such as BMGE. The following shell script executes BMGE in parallel, trimming poorly aligned sites from the alignment.

```
ls *.aln | parallel java -jar BMGE.jar -i {} -t AA -m BLOSUM30 -of
trimmed_{}
```

Infer bootstrap distribution of trees for each gene family

The alignments are then used to estimate ML trees for each gene family in IQ-TREE [5], and the distribution of ultrafast bootstrap trees is used to capture gene tree uncertainty. The gene tree distributions are used to estimate conditional clade probabilities [6], and therefore gene tree

probabilities, in ALE, and so the most natural input would be an MCMC sample of gene trees (for example, sampled using PhyloBayes [7]. In practice, this can be computationally expensive for large datasets, and sets of ultrafast bootstrap trees may represent an acceptable compromise.

The distribution of gene trees corresponding to each gene family is summarized in a .ale file that can be used in subsequent ALE steps using the ALEobserve command.

The quality of input gene trees is important for subsequent analyses, and so the choice of substitution model is important [8]. The following command executes IQ-TREE with a model search over two mixture models and creates ultrafast bootstrap tree distributions. For more information on how to choose an appropriate substitution model, refer to the IQ-TREE documentation (http://www.iqtree.org/doc/).

```
iqtree2 -s <Trimmed_gene_family_ID.aln> -m MFP -madd LG+C20,LG+C60 --score-diff ALL -B 10000 -wbtl
```

To convert bootstrap tree distributions into ALE objects, use ALEobserve:

```
ALEobserve <ufbootfile>
```

To speed up the process use GNU parallel:

```
ls *.ufboot | parallel ALEobserve {}
```

1.5 Inferring the unrooted species tree

Unrooted species trees can be estimated using a variety of methods, including maximum likelihood (ML) or Bayesian analysis of a core gene concatenation, supertree and multispecies coalescent approaches. The general approach usually involves identifying a set of single-copy orthologous genes shared across all, or most, of the genomes being studied, and then estimating a species tree from these single-copy orthologues either by concatenation (sticking marker gene alignments together and treating them as one long gene), supertree (taking a

statistical summary of single gene trees inferred from each marker gene individually) or using other approaches such as inference under the multispecies coalescent [9,10]. Users will likely have their own approach to inferring unrooted species trees; for the demo dataset, we used OrthoFinder [11], which is available at https://github.com/davidemms/OrthoFinder). Orthofinder finds orthogroups in sets of genomes that can be used for concatenation and unrooted species tree inference. Note that OrthoFinder also infers a rooted species tree, and it may be interesting to compare the result to that supported by the ALE analysis.

On the demonstration dataset, OrthoFinder identified 87 single copy orthologs. These can be aligned and trimmed to form a super matrix as stated in the next section. Frequently, no single-copy orthologues are identified as a result of large numbers of lineage-specific gene duplications (in-paralogues). One approach is simply to remove in-paralogues so as to obtain orthologues that are single copy in most species. We provide a script to do this in the Github repository. *prem3.py* generates single copy gene families from the orthologous sequence files produced by Orthofinder - *prem3.py* takes two command line arguments: 1) A species list ('species_list_demo.txt'), 2) A percentage cutoff for species representation. Species representation is defined by the number of species from the original dataset remaining once duplicate species have been removed. Adjust the percentage cutoff to determine the number of single copy families returned. Aim to increase the value as close to '100' as possible, whilst retaining a sufficient amount of data to construct a sequence alignment. To execute *prem3.py*:

```
$python3 prem3.py species_list_demo.txt 90
```

The output of the script is two directories, one containing gene families with more than or equal to species representation cut-off provided in the second command line argument, the other with all other gene families. Gene families with more than the cut-off for species representation can then be aligned and trimmed using the same scripts from the make ALE Objects section.

The aligned and trimmed single copy gene families can then be concatenated into a super matrix. We provide a Python 3 script to do this called *super_matrix_2.py*. This takes one command line argument, a species list, the same as *prem3.py* e.g. 'species_list_demo.txt'. The script will produce a number of directories. The concatenation will be in the Concatenation directory. In this directory there will also be a conservative concatenate - if a species is represented by more than 50% gaps this will not be in the conservative concatenate alignment. A list of the species removed from the conservative concatenate will be in the Results directory. Also in the results directory will be a plot of the gap content of all the species in the alignment - this is also a quick way to assess the quality of your alignment. To execute *super_matrix_2.py,* call the script in the directory with the aligned and trimmed single copy gene families:

```
$python3 super_matrix_2.py species_list_demo.txt
```

The species tree can then be inferred from the alignment under a maximum likelihood or bayesian framework using software such as IQ-TREE [5] or PhyloBayes [7]. Alternatively, instead of constructing a supermatrix, infer individual gene trees for the single copy gene families and employ a super tree approach such as ASTRAL (https://github.com/smirarab/ASTRAL). Most of these methods infer an unrooted species tree. These unrooted trees can then be manually rooted in several different candidate positions.

1.6 Computing reconciled gene trees for each candidate rooted species tree

To evaluate the likelihood of each root position, the unrooted species tree can be manually rooted in different candidate positions - the candidate root positions for the demo dataset are available at https://github.com/ak-andromeda/ALE_methods/tree/main/Demo_data. To visualise unrooted trees to help select candidate root positions, you might consider using iTOL [12] (https://itol.embl.de/) or FigTree (http://tree.bio.ed.ac.uk/software/figtree/). iTOL will introduce a piece of text ('OROOT') into the newick tree to highlight the original root if the tree

loaded was already rooted. The ‘OROOT’ can be manually removed in any text editor. Additionally, we provide a script to re-root species trees called *root_tree_in_position.py.* The script requires three command line arguments: 1) An already rooted species tree, 2) species one, and 3) species two. The script will re-root the tree between species one and species two. To execute the script issue the following command:

```
$Python3 root_tree_in_position.py sulfolobus.newick Streptomyces
Bacillus
```

Use ALEml_undated, part of the ALE distribution, to reconcile each ALE object with the candidate rooted species trees using the undated ALE model [13]. This step can be parallelized for efficiency, as even small datasets can involve thousands of gene families, each of which must be reconciled against each candidate rooted species tree. ALEml_undated takes, at minimum, two arguments, a candidate rooted species tree and an ALE file. It may be worth exploring several other features of the reconciliation model. For example, it is possible to account for varying completeness of the genomes used in the analysis by indicating the fraction of each genome estimated to be missing (fraction_missing option; see the ALE documentation for more details. In brief, a tool such as BUSCO [14] or CheckM [15] can be used to estimate genome completeness e.g 80% - thus, the fraction missing is 0.2. To add the fraction missing parameter just add: *fraction_missing=fraction_missing.txt* (where 'fraction_missing.txt' holds the respective fraction missing for each sequence e.g. 'Sulfolobus:0.20' on separate lines).

For a given rooted species tree, the ALEml_undated jobs can be distributed on a multicore processor using GNU parallel as described above:

```
ls *.ale | parallel ALEml_undated between_bac_arc.newick {}
```

The parameters of the reconciliation model used by ALEml_undated can be modified using various command line arguments (see https://github.com/ssolo/ALE for more details). For example, D, T or L rates can be individually set to zero using the delta, tau and lambda arguments (e.g. tau=0 results in a reconciliation without any gene transfers, delta=0 a reconciliation without any gene duplications); otherwise, the values of these parameters are optimised by maximum likelihood. The demonstration dataset includes six taxa, and so there are 9 possible root positions on the unrooted tree (Figure 3). We ran the above command 9 times, resulting in a set of ALE output files for each gene family under each candidate root. To avoid confusion, we ran the reconciliations associated with each root position in a different directory, with the directory names corresponding to the root branch. Other arrangements are also possible, but the scripts included in the workflow assume that you organised the data in this way.

1.7 Comparing support for candidate root positions

Gene family likelihoods can be used to compare support for different rooted species trees in the same way as site likelihoods can be used to compare support for different gene trees in traditional phylogenetics. The program CONSEL [16] implements statistical tests to choose among trees given sets of site (or gene family) likelihoods. We provide a python script to construct an input file for CONSEL based on sets of gene family likelihoods calculated for the same gene families under several different candidate roots. The script requires the name of directories containing the reconciliation files for each root to be listed. Please note that the naming of each '.uml_rec' file should be consistent for each root - if you have tested 4 roots, in directories named root_1, root_2, root_3 and root_4, the reconciliation output for the ALE object: 1.ale, should be named 1.ale.uml_rec in each of the four directories. 1.ale.uml_rec will have different values for DTL under each candidate root. Use the *write_consel_file.py* script to construct a table of likelihoods:

```
$python3 write_consel_file_p3.py root1 root2 root3 root4 >
likelihoods_table
```

The order of the roots is not preserved in the likelihood table. Open the likelihood table in a text editor and note the order of the roots in the likelihood table , as these are assigned a representative number i.e. the first root in the likelihood table is referred to as item ‘1’ in subsequent results output.

Next, use CONSEL (available at: http://stat.sys.i.kyoto-u.ac.jp/prog/consel/) to perform an AU test to establish a confidence set of roots that cannot be rejected given the model and data. CONSEL requires three commands to be executed in succession:

```
$makermt likelihoods_table
$consel likelihoods_table
$consel/bin/catpv likelihoods_table > au_test_out
```

Assess the likelihood of each root in the 'au_test_out' file. The ‘au_test_out’ file includes a table ranking the roots by likelihood, and a number of statistical tests. The AU value column provides a *P*-value. At the conventional threshold of 0.05, candidate roots with $P < 0.05$ are rejected, while roots with $P > 0.05$ are part of a confidence set of roots that cannot be rejected. See 'au_test_out' as an example on the GitHub page.

1.8 Evaluating the nature of the root signal (robustness checks)

In traditional phylogenetics, site stripping – the removal of fast-evolving [17] or compositionally-biased [18,19] sites – is often used to assess the robustness of phylogenetic signal. The rationale is that the information in these sites can be misleading (or at least difficult to model adequately), for example favouring long branch or compositional attraction over real evolutionary signal. Relationships in the overall tree that derive their support from fast-evolving or compositionally-biased sites are then considered less reliable than relationships that are robust to the removal of these sites. An analogous situation exists in the context of root or

topology inference at the phylogenomic level: gene families differ in terms of size, evolutionary rate, and rates of DTL events, and some families may be easier to model than others given the methods available. In a recent study [20], some of us explored the concept of gene family filtering: sorting families by some quality metric, and then filtering out an increasing proportion of families and evaluating the impact on support for competing relationships. By analogy to the traditional case, relationships that derive their support from particular subsets of the data (e.g., gene families with very high loss or transfer rates) might be considered less reliable than those which obtain support across the full distribution of families. As with traditional site filtering, this procedure is perhaps best viewed as a way to explore the data. It is not clear that filtering “outlier” sites or gene families will always improve inference, and there are no clear-cut thresholds at which, for example, fast sites or frequently-transferred genes can be definitively considered unreliable.

To facilitate exploration of the data in this way, we provide a script (*DTL_ratio_analysis_ML_diff.py)* that evaluates the impact of gene families with high rates of duplication, transfer and loss (DTL). The script sequentially removes gene families from the analysis based on their DTL rates, and re-calculates the summed likelihood of each root and the difference in likelihood from the maximum likelihood root. Thus, from these analyses we can visualise the extent to which particular kinds of gene families (for example, those with particularly high or low DTL rates) drive support for a given root in the full analysis.

DTL_ratio_analysis_ML_diff.py creates 3 plots based on either the duplication, transfer or loss rates or the ratio of loss/speciation, duplication/speciation or transfer/speciation rates . Only one of the above metrics can be tested at one time. The first plot shows the summed likelihood of the different roots following the sequential removal of families based on the chosen metric, the second plot shows the likelihood difference of each root from the maximum likelihood root, and the third plot shows the same as the second but for only gene families with high species representation. The script requires two text files named "roots_to_test.txt", and

"species_list.txt". The “roots_to_test.txt” file should contain the names of the directories of the rec_files. The species_list.txt file should contain all the species in the dataset and should match the formatting of species tree used in the reconciliation analysis. The script also requires two command line arguments: The name of the directory of the maximum-likelihood root determined by an AU test, and the metric you want to remove gene families by D,T,L, LS, DS or TS. The script also remakes the plots only using high species representation reconciliation files. The species representation cutoff is currently set to 50% - this can be manually changed by altering the make_high_rep_df() function. Run the script in the same directory as ‘write_consel_file.py’. To execute the script for e.g. command to assess the LS ratio under Sulfolobus as the most likely root:

```
$python3 DTL_ratio_analysis_ML_diff.py Sulfolobus LS
```

Plots and data produced will be saved to a new directory e.g. ‘LS_ratio_results’. The directory name will change with the metric used. Plots will be saved as both png and svg figures. The likelihood difference from the ML root following the removal of families based on DTL rates is presented in Figure 5. As previously mentioned, only one of the metrics – that is either, Duplication (D), Transfer (T), and Loss (L) – can be analyzed at one time, and Figure 5 is a combination of results from each of these runs.

1.9 Gene content evolution on the most likely rooted species tree

Once the most likely root has been identified, this technique allows users to quantify the relative contributions of duplication, transfer, loss and origination in the gene content evolution (Figure 6). These predicted values can be calculated for every branch of the species tree, enabling insights into how gene content evolves over time. Here we provide a script that calculates the predicted total number of duplication, transfer, loss and origination events for

all branches of the species tree and estimates the number of genes present in the ancestral genome at each internal tree node.

```
$python3 branchwise_number_of_events.py > dtloc.tsv
```

The ALE output also provides estimates of the gene families present at each node. Therefore we can model the presence and absence of gene families at internal nodes - reconstructing ancestral genomes. We provide the following script to reconstruct all the genes present at internal nodes. There are three command line arguments. The first is the percentage the gene family appears at that branch in the reconciliations i.e. 0.5 = occurs in 50% of the reconciliations. If you would like to perform a more stringent ancestral reconstruction, increase the percentage. The second and third arguments is the range of the internal nodes. For e.g. in this dataset the internal nodes are labelled 6,7,8,9,10 (Can be seen in the output of *branchwise_number_of_events.py).* Thus, the command line arguments are 6 and 10. All nodes between 6-10 will be reconstructed.

```
$python3 Ancestral_reconstruction_copy_number.py 0.5 6 10
```

Sequences of interest can then be functionally annotated with GO-terms and the KEGG database, using eggNOG-mapper [21] and BlastKOALA [22] respectively.

2 *References*

1. Enright, A. J., Van Dongen, S. & Ouzounis, C. A. An efficient algorithm for large-scale detection of protein families. *Nucleic Acids Research* vol. 30 1575–1584 (2002).
2. Tange, O. GNU Parallel 2018. (2018) doi:10.5281/ZENODO.1146014.
3. Katoh, K. & Toh, H. Recent developments in the MAFFT multiple sequence alignment program. *Brief. Bioinform.* (2008) doi:10.1093/bib/bbn013.
4. Criscuolo, A. & Gribaldo, S. BMGE (Block Mapping and Gathering with Entropy): A new software for selection of phylogenetic informative regions from multiple sequence alignments. *BMC Evol. Biol.* (2010) doi:10.1186/1471-2148-10-210.
5. Nguyen, L. T., Schmidt, H. A., Von Haeseler, A. & Minh, B. Q. IQ-TREE: A fast and effective stochastic algorithm for estimating maximum-likelihood phylogenies. *Mol. Biol. Evol.* (2015) doi:10.1093/molbev/msu300.
6. Larget, B. The estimation of tree posterior probabilities using conditional clade probability distributions. *Syst. Biol.* **62**, 501–511 (2013).
7. Lartillot, N., Rodrigue, N., Stubbs, D. & Richer, J. Phylobayes mpi: Phylogenetic reconstruction with infinite mixtures of profiles in a parallel environment. *Syst. Biol.* (2013) doi:10.1093/sysbio/syt022.
8. Yang, Z. & Rannala, B. Molecular phylogenetics: Principles and practice. *Nature Reviews Genetics* vol. 13 303–314 (2012).
9. Ren, F., Tanaka, H. & Yang, Z. A likelihood look at the supermatrix-supertree controversy. *Gene* **441**, 119–125 (2009).
10. Bravo, G. A. *et al.* Embracing heterogeneity: Coalescing the tree of life and the future of phylogenomics. *PeerJ* **2019**, e6399 (2019).
11. Emms, D. M. & Kelly, S. OrthoFinder: Phylogenetic orthology inference for comparative genomics. *Genome Biol.* (2019) doi:10.1186/s13059-019-1832-y.
12. Letunic, I. & Bork, P. Interactive Tree Of Life (iTOL): An online tool for phylogenetic

tree display and annotation. *Bioinformatics* (2007) doi:10.1093/bioinformatics/btl529.

13. Szöllosi, G. J., Tannier, E., Lartillot, N. & Daubin, V. Lateral gene transfer from the dead. *Syst. Biol.* **62**, 386–397 (2013).
14. Simão, F. A., Waterhouse, R. M., Ioannidis, P., Kriventseva, E. V. & Zdobnov, E. M. BUSCO: Assessing genome assembly and annotation completeness with single-copy orthologs. *Bioinformatics* (2015) doi:10.1093/bioinformatics/btv351.
15. Parks, D. H., Imelfort, M., Skennerton, C. T., Hugenholtz, P. & Tyson, G. W. CheckM: Assessing the quality of microbial genomes recovered from isolates, single cells, and metagenomes. *Genome Res.* **25**, 1043–1055 (2015).
16. Shimodaira, H. & Hasegawa, M. CONSEL: for assessing the confidence of phylogenetic tree selection. *Bioinformatics* (2001) doi:10.1093/bioinformatics/17.12.1246.
17. Kostka, M., Uzlikova, M., Cepicka, I. & Flegr, J. SlowFaster, a user-friendly program for slow-fast analysis and its application on phylogeny of Blastocystis. *BMC Bioinformatics* **9**, 1–6 (2008).
18. Viklund, J., Ettema, T. J. G. & Andersson, S. G. E. Independent genome reduction and phylogenetic reclassification of the oceanic SAR11 clade. *Mol. Biol. Evol.* **29**, 599–615 (2012).
19. Muñoz-Gómez, S. A. *et al.* An updated phylogeny of the alphaproteobacteria reveals that the parasitic rickettsiales and holosporales have independent origins. *Elife* **8**, (2019).
20. Coleman, G. A. *et al.* A rooted phylogeny resolves early bacterial evolution. *Science (80-.).* **372**, (2021).
21. Huerta-Cepas, J. *et al.* Fast genome-wide functional annotation through orthology assignment by eggNOG-mapper. *Mol. Biol. Evol.* **34**, 2115–2122 (2017).
22. Kanehisa, M., Sato, Y. & Morishima, K. BlastKOALA and GhostKOALA: KEGG Tools for Functional Characterization of Genome and Metagenome Sequences. *J. Mol. Biol.* **428**, 726–731 (2016).