Should protracted speciation be incorporated in phylogenetic tree construction methods?

Richèl J.C. Bilderbeek & Rampal S. Etienne

February 2, 2016

1 Abstract

The construction of phylogenies has proven invaluable in answering evolutionary biological questions. Our current phylogenetic tools ignore the fact that speciation takes time, which has an effect unknown in phylogeny reconstruction. Here, we measure the errors that the Bayesian phylogenetic software tool BEAST2 gives when recovering simulated phylogenies, for different times-to-speciate, under a range of additional parameter settings. It has been found that branch lengths are consistently and strongly underestimated for biologically relevant parameters. This research shows that protractedness is a complexity of nature that should not be ignored and should be incorporated in our phylogenetic tools.

2 Introduction

Speciation takes time Although we know that speciation takes time, we commonly ignore this when constructing a phylogeny, by chosing a constant-rate birth-death model as a speciation model. The constant-rate birth-death model (as described in for example [6]) is among the simplest speciation models, and assumes instant speciation and extinction rates, captured by its two parameters. The constant-rate birth-death model is popular for its simplicity, yet has also served as a starting point for more elaborate specation models.

Other non-protracted speciation models Other speciation models may assume that speciation rate changes in time [REF], is dependent on the amount of species present [3], or is trait dependent [REF]. The original model by making speciation rate dependent on time, diversity or trait value. Also these extended models assume speciation is instantaneous.

Protracted speciation model There is, however, a speciation model family that does assume speciation takes time: the protracted speciation model family [5], which extends the constant-rate birth-death model by adding an additional

species state (see also figure 1). It is called a model family, as it makes no assumption about speciation and extinction rates: these may be contant or depend on time, diversity or trait value.

Use of a speciation model in inferring a phylogeny Speciation models do not exist for theoretical purposes only, but are widely used to make inferences from genetic data. There are multiple computer programs to create phylogenies and/or parameter estimates from aligned DNA sequences. One such tool is BEAST2 [1], which allow for a Bayesian approach to phylogenetics. BEAST2 supplies the user with multiple speciation models, yet all assume instantaneous speciation.

This study This simulation investigates the consequence of BEAST2 using instantaneous speciation, by creating a 'true' tree that is protracted and seeing how well BEAST2 can recover it. It is expected that for higher protractedness (thus deviation from BEAST2 its assumptions), the error will increase. It is unknown, however, if and when this error is relevant.

Preview results This study shows that [TODO: put result here]

3 Methods

3.1 Model

The speciation model used in this investigation is a constant-rate protracted speciation model.

A protracted speciation model assumes that species have at least two states: a species is either a good or incipient species. A good species is a species recognized as such, where an incipient species is not yet. Both good and incipient species can generate new incipient species, at the speciation-initiation rates b_g and b_i respectively. An incipient species can become a good species at the speciation-completion rate λ. Both good and incipient species can go extinct at rates μ_i and μ_g respectively (see also figure 1).

The simplest member of the protracted speciation famility is the constant-rate protracted speciation model, which assumes that all rates are constant in time.

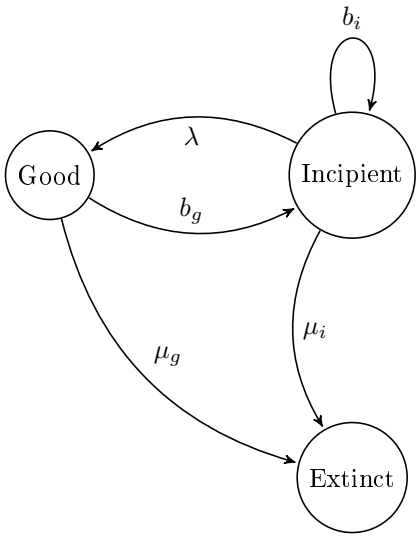

Figure 1: The states and transitions of a species. b_i: speciation-initiation rate of incipient species. b_g: speciation-initiation rate of good species. λ: speciation completion rate. μ_i: extinction rate of incipient species. μ_g: extinction rate of good species. Figure after Etienne et al, 2014, Evolution

There is no relationship known between the speciation rate (often called λ) of the constant-rate birth-death model and a combination of the speciation-initiation λ and speciation-completion rates of the protracted birth-death model (b_g and b_i). When setting $\lambda \to \infty$, the model used falls back to a constant-rate (non-protracted) birth-death model.

3.2 Workflow

3.2.1 Creating gene trees

From a parameter combination, a 'true' protracted constant-rate birth-death gene tree is simulated in the R programming language [9], using the PBD package [2].

Speciation-completion rate The speciation-completion rate λ is pivotal for this study, as when $\lambda \to \infty$ the model falls back to a birth-death model and satisfies the instantaneous speciation assumption of the tools used.

Speciation initiation rate, extinction rate and crown age The parameter values for the speciation-initiation rates b, extinction rate μ and phylogeny crown age t_c need to be balanced, as $t_c b \gg \mu$ results in overly taxa-rich phylogenies, where $t_c \mu \gg b$ results in excessively frequent extinctions. As a starting point, we used the parameters used by [4]. For simplicity, we assume species

can give rise to new species, independent of species status, thus $b_i = b_g$ [TODO: Is there reason to assume differently?]. Additionally, we assume a species can go extinct independent of species status, so$\mu_i = \mu_g$ [TODO: Is there reason to assume differently?].

3.2.2 Creating species trees

The longer speciation takes, the higher the number of incipient species will be. Because BEAST2 assumes one individual per species, we randomly sample one individual per species of all species to obtain a species tree. This is replicated n_s times. Also an outgroup is added, so that the phylogeny inferring software can root a phylogeny.

3.2.3 Creating a DNA alignment

From each species tree, n_a DNA alignments were simulated following a Jukes-Cantor nucleotide subsititution model using the phangorn R package [10].

In creating DNA sequence alignments from the phylognies, a mutation rate r and DNA sequence length l_a need to be balanced as well. As a starting point, we chose the DNA sequence length to be 1kb, as this a common gene sequence length, but we also included one and two orders of magnitude bigger. The mutation rate is chosen to match a mutation rate as in nature [TODO: find that value].

3.2.4 Creating a BEAST2 posterior

From these DNA alignments, a posterior containing phylogenies and parameter combinations were constructed using the BEAST2 software package [1]. Per alignment, n_b BEAST2 runs were performed, each with an MCMC length of l_m, to see if both runs result in similar posteriors.

The BEAST2 priors used were as follows:

For the site model, the default parameters were used: which is the Jukes-Cantor substitution model.

For the clock model, the default parameters are used, which is a strict clock prior, with a clock rate of 1.0.

For the tree prior, the 'Birth Death Model' was selected and its parameters kept at the default uniform birth-rate (range 0-10^5, initial value 1) and default uniform birth-rate (range 0-1, initial value 0.5).

3.2.5 All parameters used

Table 1 shows all parameter values used.

3.3 Analyzing the results

These posteriors were analyzed using bash (`www.gnu.org/software/bash`) scripts and the R programming language [9], using the packages rBEAST [7] to pro-

Parameters	Values
$b = b_g = b_i$	0.1, 0.5, 1.0
λ	0.1, 0.3, 1.0, 10^6
$\mu = \mu_g = \mu_i$	0.0, 0.1, 0.2, 0.4
t_c	15
r	10^{-1},10^{-2},10^{-3}
l_a	10^3,10^4,10^5
n_s	2
n_a	2
n_b	2
l_m	10^6

Table 1: Parameters used

cess BEAST2 output files, ape [8] and ggplot2 [11] for plotting and testit [12] for debugging. All the scripts can be downloaded `https://github.com/richelbilderbeek/should`.

How well do two BEAST runs repeat (from the same alignment)? Very good

How similar are the results of different alignments (of the same species tree)? Good

How similar are the results of two different species trees (from the same gene tree)? Similar

The effect of sequence length and mutation rate The effects of sequence length and mutation rate are ...

Number of taxa The number of taxa ...

Difference in nLTT plots Protracted speciation

Histogram of errors Histogram of errors

4 Results

5 Discussion

The protracted speciation model creates trees with less taxa,

A constant-rate protracted birth-death model is used, this research could easily be modified to compare other protracted birth-death models, for example,

a diversity-dependent protracted birth-death model. There has been now work done on the diversity-dependent protracted birth-deatg model yet.

This research assumes that speciation is allopatric, because in the simulation of the DNA alignments, there is no gene flow anymore between two taxa of still-the-same species.

References

[1] Remco Bouckaert, Joseph Heled, Denise Kühnert, Tim Vaughan, Chieh-Hsi Wu, Dong Xie, Marc A Suchard, Andrew Rambaut, and Alexei J Drummond. Beast 2: a software platform for bayesian evolutionary analysis. *PLoS Comput Biol*, 10(4):e1003537, 2014.

[2] Rampal S. Etienne. *PBD: Protracted Birth-Death Model of Diversification*, 2015. R package version 1.1.

[3] Rampal S Etienne, Bart Haegeman, Tanja Stadler, Tracy Aze, Paul N Pearson, Andy Purvis, and Albert B Phillimore. Diversity-dependence brings molecular phylogenies closer to agreement with the fossil record. *Proceedings of the Royal Society of London B: Biological Sciences*, page rspb20111439, 2011.

[4] Rampal S Etienne, Hélène Morlon, and Amaury Lambert. Estimating the duration of speciation from phylogenies. *Evolution*, 68(8):2430–2440, 2014.

[5] Rampal S Etienne and James Rosindell. Prolonging the past counteracts the pull of the present: protracted speciation can explain observed slowdowns in diversification. *Systematic Biology*, 61(2):204–213, 2012.

[6] Sean Nee, Robert M May, and Paul H Harvey. The reconstructed evolutionary process. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 344(1309):305–311, 1994.

[7] olli0601. *rBEAST*.

[8] E. Paradis, J. Claude, and K. Strimmer. APE: analyses of phylogenetics and evolution in R language. *Bioinformatics*, 20:289–290, 2004.

[9] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2013.

[10] K.P. Schliep. phangorn: phylogenetic analysis in r. *Bioinformatics*, 27(4):592–593, 2011.

[11] Hadley Wickham. *ggplot2: elegant graphics for data analysis*. Springer New York, 2009.

[12] Yihui Xie. *testit: A Simple Package for Testing R Packages*, 2014. R package version 0.4.

A Worked-out examples

To better understand the steps performed in this research, I show here some worked-out examples in detail.

A.1 Workflow

Every experiment has the following steps:

- Creating parameter files: see chapter A.1.1
- Checking your parameter file: see chapter A.1.2
- Do simulations: see chapter A.1.3
- Analyze results: see chapter A.1.4

Figure 2 shows the workflow graphically:

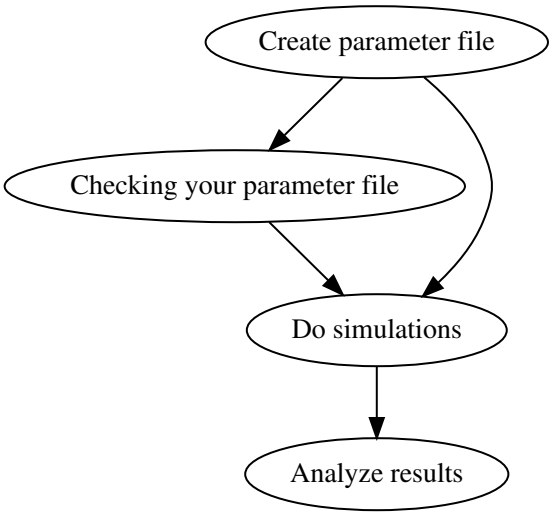

Figure 2: The chronological order of the experiment

Each step can be done from a command-line script, so that a computer cluster can conveniently work on it.

A.1.1 Creating parameter files

Parameter files can be created from the command line by supplying all the arguments to the script 'create_parameter_file.R' (see chapter C.4).

A.1.2 Checking your parameter file

You can check if you created a parameter file with the correct arguments using the R script 'check_parameter_file.R' (see chapter C.2).

A.1.3 Do simulations

A simulation (including its replicates) can be run from the command line by supplying all the arguments to the R script 'do_simulation.R' (see chapter C.5). This will result in:

- A parameter file with intermediate data added
- BEAST2 output files

A.1.4 Analyze results

After the simulations, the results are analyzed with the R script 'do_analyze.R' (see chapter C.1).

A.2 Experiments

These worked-out examples show:

- the error if protractedness is absent: example 1, see chapter A.3
- the error if protractedness strong: example 2, see chapter A.4
- comparing the errors of example 1 and 2: see chapter A.5
- the error if protractedness is absent for multiple replicates: example 3, see chapter
- the error if protractedness is strong for multiple replicates: example 4, see chapter
- comparing the errors of example 3 and 4: see chapter

An overview of all parameter settings can be seen in table 2, which can be created by calling the 'create_example_parameter_files.sh' script .

The parameter settings used in this example are identical, except for their speciation completion rate and number of replicates.

Creating these parameter files is shown in chapter A.1.1.

The experimental setup of this research has multiple steps, which we will follow closely here. These steps are described in chapter A.1.

These worked-out examples show the data produced in its raw form and does not care too much about aesthetics.

Example	1	2	3	4
Protracted?	No	Yes	No	Yes
RNG seed	1			
$b = b_g = b_i$	0.5			
λ	10^6	10^{-1}	10^6	10^{-1}
$\mu = \mu_g = \mu_i$	0.1			
t_c	5			
t_s	1		4	
r	0.01			
n_a	1		4	
l_a	1000			
n_b	1		4	
l_m	10^6			

Table 2: Parameters used in the examples

A.3 Example 1: Weak protractedness

This example answers the question: what is the base level error of the analyses in this research?

The base level error can be obtained by using parameters for a constant-rate birth-death model. All tools used assume this model, but there will be noise (thus error) added in the process.

The parameter settings of example 1 have a high speciation completion rate λ, which makes the constant-rate protracted speciation model fall back to a constant-rate birth-death model, as incipient species become good species (close to) instantaneously.

A.3.1 Gene tree

The first step is to simulate a gene tree. For our parameters, this results in the tree in figure 3

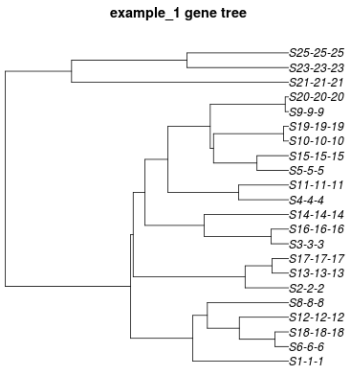

Figure 3: Example 1 gene tree. The taxon labels are S[genus]-[species]-[subspecies]'. This plot is created by the function 'plot_gene_tree'

A.3.2 Species tree

From that gene tree, we create a species tree by sampling one individual per species. Because speciation is instantenous for a constant-rate birth death model, there exist no multiple individuals per species. To being able to root our phylogenies in later steps, an outgroup is added as well, resulting in figure 4:

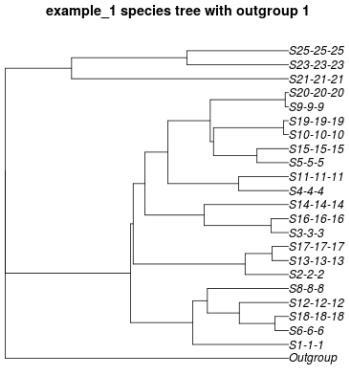

Figure 4: Example 1 species tree. This plot is created by the function 'plot_species_tree_with_outgroup'

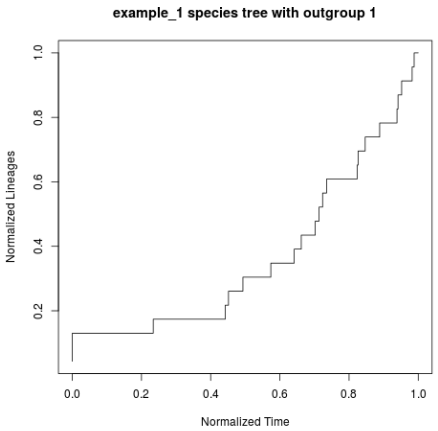

Figure 5: Example 1 species tree its nLTT plot. This plot is created by the function 'plot_species_tree_with_outgroup_nltt'

Gene tree and all species trees Doing this multiple times:

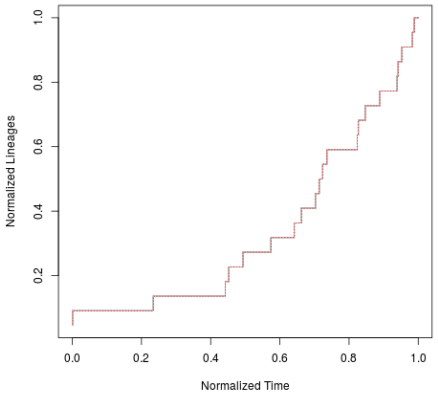

Figure 6: Example 1 gene tree (black solid line) and possible sampled species tree

A.3.3 Alignment

Knowing the evolutionary distances between species, DNA sequence alignments can be simulated fitting the tree. To do so, the parameters for sequence length and mutation rate are used. Note that this research assumes a simple Jukes-Cantor model, and does so as well in later steps.

Figure 7 shows a visualisation of the simulated alignment of our example:

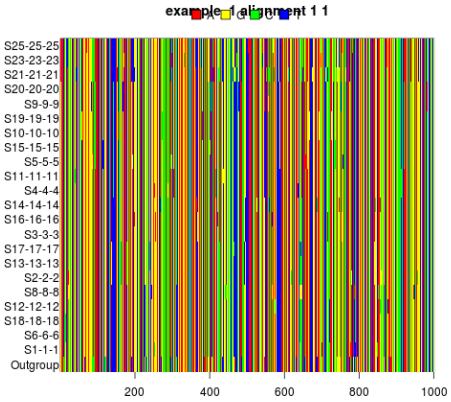

Figure 7: Example 1 alignment. This figure is created using the function 'plot_alignments'

A.3.4 Posterior

With BEAST2 we can now obtain a posterior. A posterior consists of a representative sample of all possible trees (and parameter estimates), yet with more probable trees being present more often.

After running BEAST2 on our DNA sequence, the full posterior must be verified to be eligible for further analysis. Using Tracer, we can open the .log file generated by BEAST2, which is then displayed as shown in figure 8:

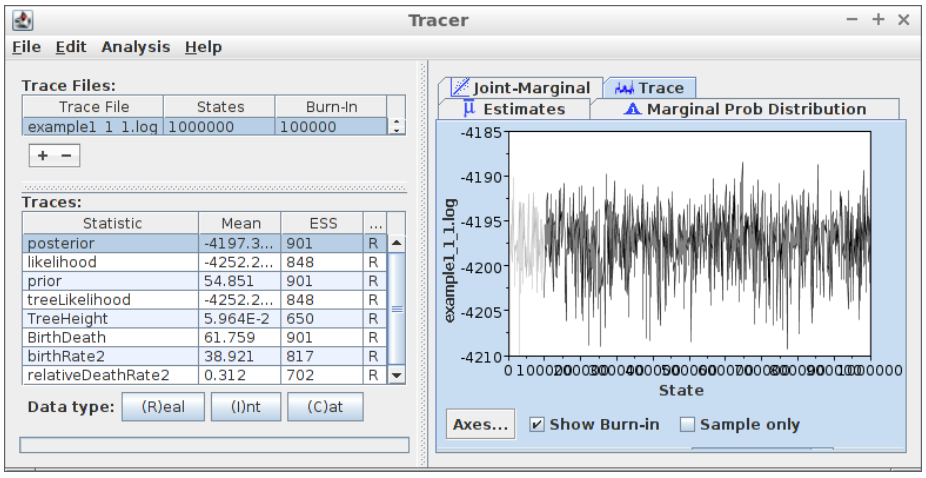

Figure 8: Example 1 its ESSes and trace log

It can be seen that the values for ESS (Effective Sample Size) are above 200

and that the trace log shows a well-mixed chain. An ESS of 200 is used as a minimum in this research.

Now that the full posterior is assumed to be correct, I will now highlight one state of it first, before going back to the full picture. In this case, I choose the last state to zoom in on. I take the last state for no specific reason and I could just as easily have picked the first or a random one. From this last state, I take the tree only. This last tree may be very different by chance, as unlikely trees are present, yet in low abundances.

The tree picked from the posterior is shown in figure 9:

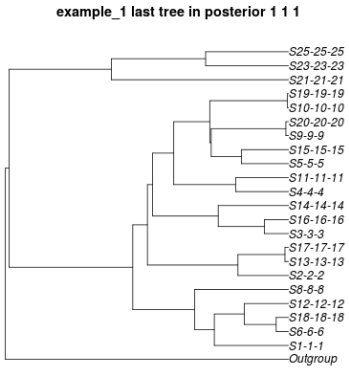

Figure 9: A random species tree from example 1 its posterior. This figure is produced by the function 'plot_posterior_samples'

The species tree picked from the posterior does not look too different compared to the original species tree (figure 4). To get a better view of their resemblance, their nLTT plots are put in the same chart in figure 10

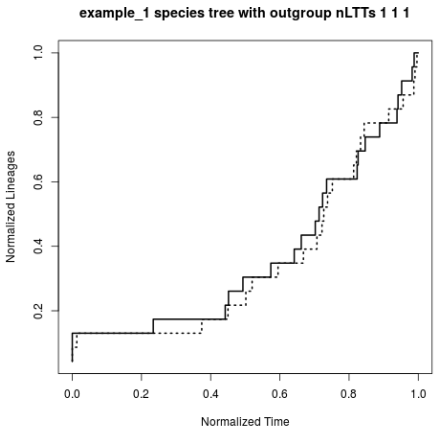

Figure 10: The nLTTs of the true species tree (solid black line) and the species tree sampled from example 1 its posterior (dotted line). This figure is produced by the function 'plot_posterior_sample_nltts'

We can observe that the lines are close, but do not match. Also this is expected, due to stochasticity in the MCMC sampling.

A posterior contains many phylogenies. Before analysing them, the tool 'densitree' can be used to view these:

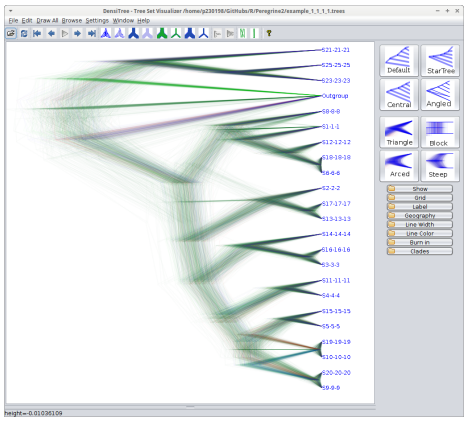

Figure 11: Example 1 its posterior its phylogenies

To derive at a quantity of the match between true/input tree and the posterior, one can sum the error between the true and posterior tree nLTT plot, see [Janzen]. For this example, the error between the true species tree and the posterior tree is 0.026. The true species tree can be compared to every tree in the posterior, calculating the nLTT statistic for each. This will results in the

following histogram:

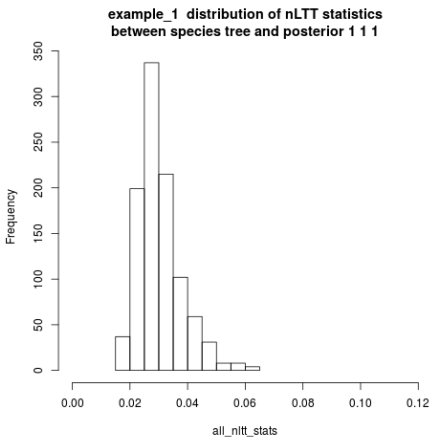

Figure 12: Histogram of the nLTT statistic between the true species tree and the trees in the posterior. This figure is produced by the function 'plot_posterior_nltt_stats_histogram'

Here we can see that this histogram looks a bit like a Poisson or Gamma distribution, with a median at $0.02 - 0.025$.

But this does not tell us where the errors have been made: near the crown, near the tips, or in between? To do so, we plot the average nLTT plot of the posterior and compare it to the true species tree its nLTT, as shown in figure 13:

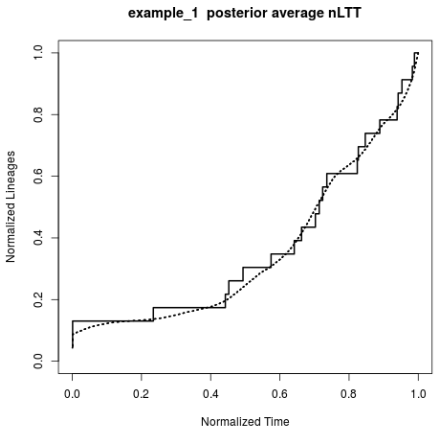

Figure 13: Average nLTT of all the trees in the posterior. Solid line: nLTT of true species tree. Dotted line: average nLTT of all posterior trees. This figure is produced using the function 'plot_posterior_average_nltts'

A.4 Example 2: Strong protractedness

This example answers the question: how do the analyses of this research look like for strong protractedness?

The pipeline is identical, except one parameter is changed: the speciation completion rate λ is set to a low value.

A.4.1 Gene tree

The gene tree simulated is shown in figure 14:

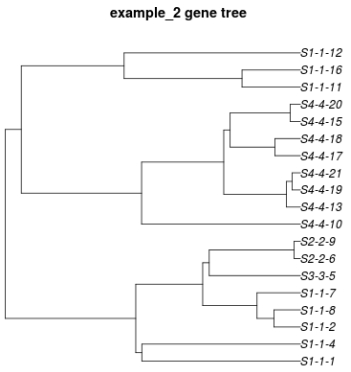

Figure 14: Example 2 gene tree. The taxon labels are S[genus]-[species]-[sub-species]'

Note that there are only four complete species here, called S1-1 to S4-4. The third label is the sub-species label. Also note that S1-1 is a polyphylic species.

A.4.2 Species tree

From that gene tree, we create a species tree by sampling one individual per species. With speciation taking a long time to complete, there are indeed many incipient species present, and only four different full species. Random picking one individual per species, and adding an outgroup, results in the species tree of figure 15:

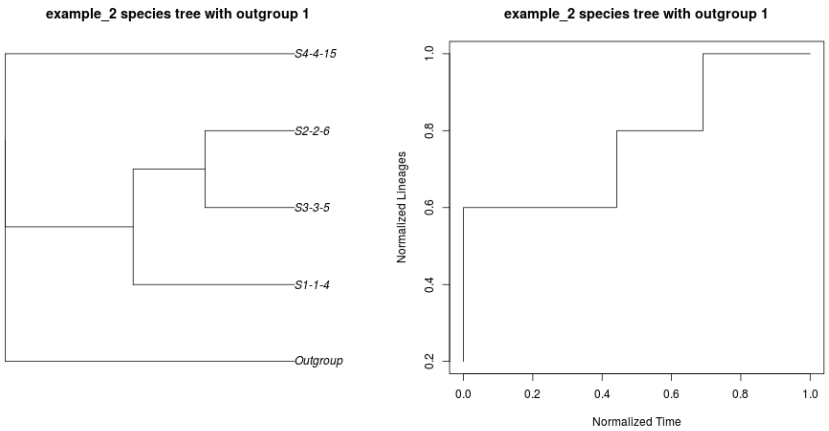

Figure 15: Example 2 species tree

Gene tree and all species trees Doing this multiple times:

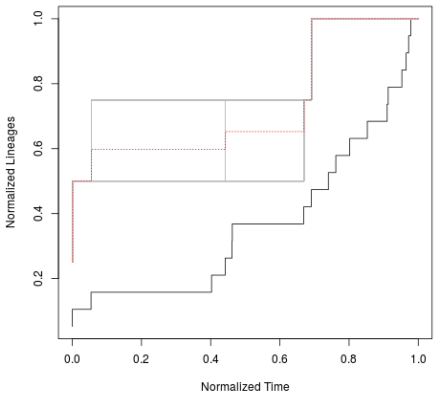

Figure 16: Example 2 gene tree (black solid line) and possible sampled species tree

A.4.3 Alignment

The DNA sequences are simulated over the species tree. Because there have been only four species and one outgroup, there will be five DNA sequences generated, as shown in figure 17:

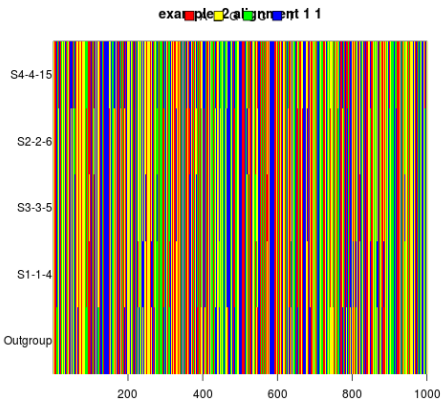

Figure 17: Example 2 alignment

A.4.4 Posterior

BEAST2 is again used to create a representative sample of all possible trees (and parameter estimates) from those alignments and checked visually by Tracer (see figure 18).

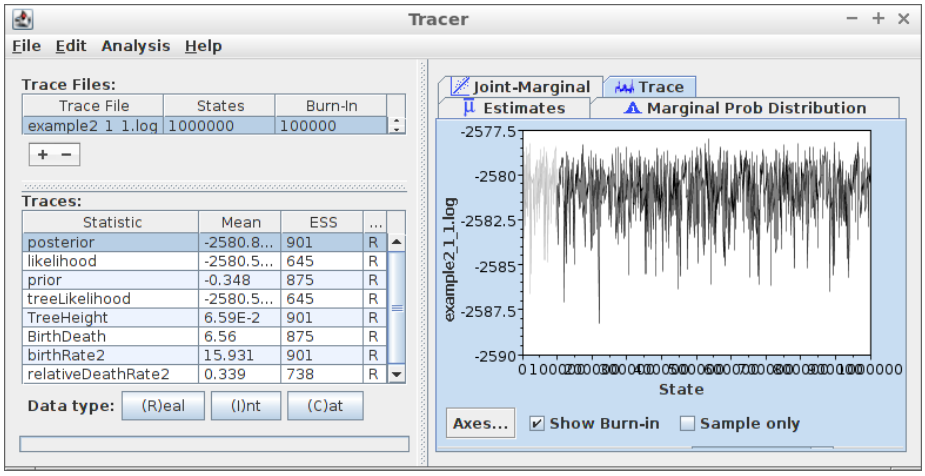

Figure 18: Example 2 its ESSes and trace log

Again all ESS values are above 200 and that the trace log shows a well-mixed chain.

Also here, we pick a random tree from the posterior (in this case, the last one):

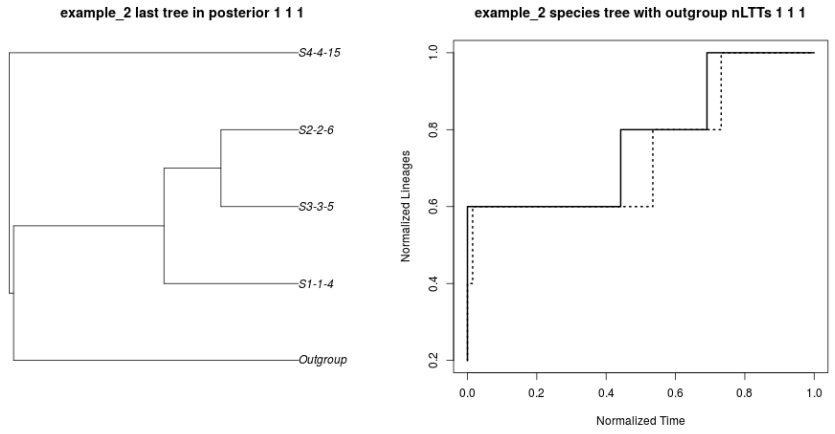

Figure 19: A random species tree (and its nLTT plot) from example 2 its posterior

We expect this tree from the posterior to match the true species tree less well, as all the tools used assume instant speciation, where we simulated tree with protracted speciation. To clearly view the difference, the nLTT plots are put in one chart, as shown in figure 20:

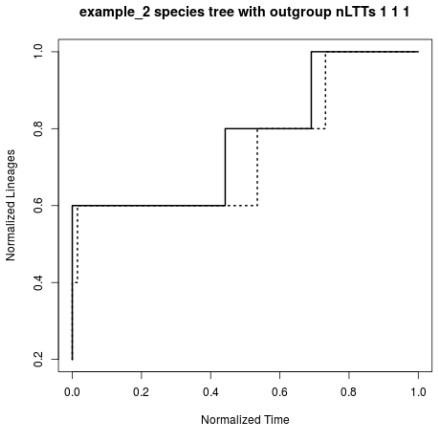

Figure 20: Both examples 2 its nLTT plots in the same chart. The solid black line is the 'true' or initial (species) tree (with outgroup), the dotted line is one of the trees in the BEAST2 posterioir

We can observe that the lines are more blocky, as there are less lineages. In this case, the surface between the nLTT statistic of the true species tree and the posterior tree is 0.071.

A posterior contains many phylogenies. Before analysing them, the tool 'densitree' can be used to view these:

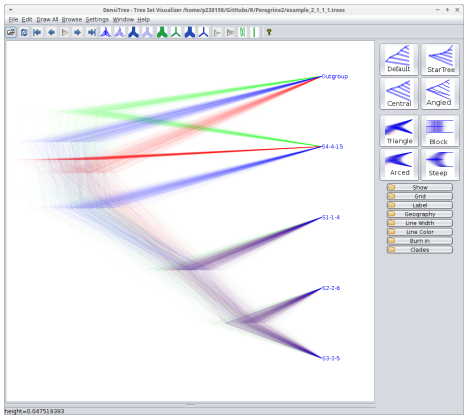

Figure 21: Example 2 its posterior its phylogenies

When scoring the nLTT statistic between the true species tree to every tree in the posterior, this will results in the histogram as shown in figure 22:

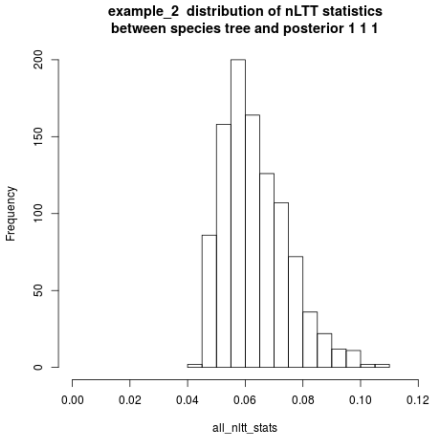

Figure 22: Histogram of the nLTT statistic between the true species tree and the trees in the posterior

This histogram looks a bit like a Gamma distribution, with a median at 0.75-0.80 [?TODO: test if this is so?].

But this does not tell us where the errors have been made: near the crown, near the tips, or in between? To do so, we plot the average nLTT plot of the posterior and compare it to the true species tree its nLTT, as shown in figure 23:

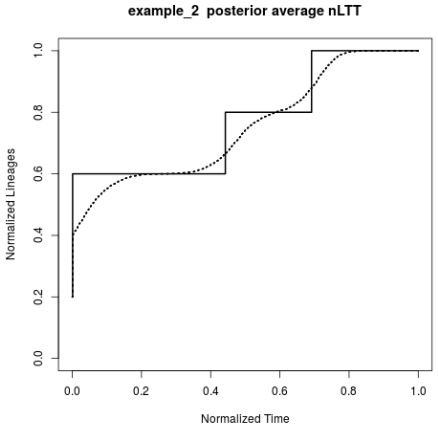

Figure 23: Average nLTT of all the trees in the posterior. Full line: nLTT of true species tree. Dotted line: average nLTT of all posterior trees

Here we can see that the posterior builds up its branches around certain timepoints: suitable phylogenies obtain their first added branch between 0.0-0.3, their second between 0.3 and 0.6, and their third between 0.6 and 0.9. This nearly always lags the true species tree.

A.5 Comparing example 1 and 2

Both examples showed a histogram of the error between true species tree and posterior trees. We expect these errors to have a different distribution: example #1 assumed instant speciation, which suits the algorithm best, where example #2 has a strong protractedness. Plotting both histograms in the same plot results in figure 24:

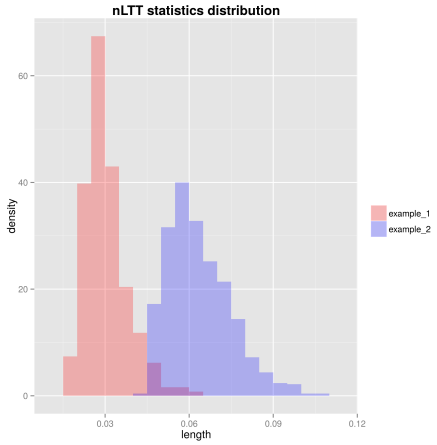

Figure 24: Histogram of the nLTT statistic of the two examples

It can be observed that these error distributions have a different median. This means that there is an observable higher error being made when the true species tree is protracted.

This visualization does not tell use how the error is made: is the error concentrated at the root, middle or tips of the tree? To locate this, the average nLTT plots of the posterior is shown in figure :

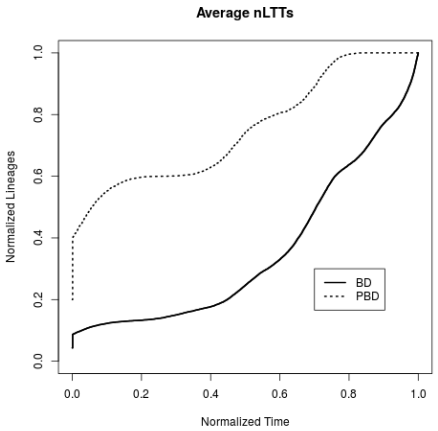

Figure 25: Average nLTTs of the posteriors of example #1 (weak protractedness, solid line) and example #2 (strong protractedness, dotted line)

A.6 Example 3: Weak protractedness with replicates

This example is an extension of the examples 1, with the differences that:

- From the (same) gene tree, multiple species trees are sampled
- Per species tree, multiple DNA alignments are simulated
- Per DNA alignment, multiple BEAST2 runs are performed

A.6.1 Gene tree

The first step is to simulate a gene tree. Because the RNG seed is at the same value as example #1, exactly the same gene tree will be obtained, as can be seen in figure 26:

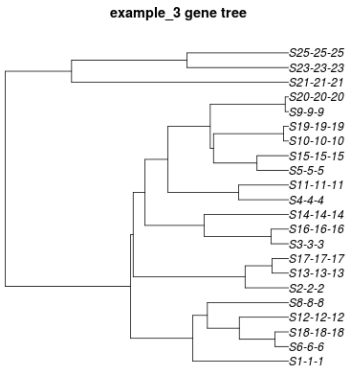

Figure 26: Example 3 gene tree. The taxon labels are S[genus]-[species]-[subspecies]'

A.6.2 Species tree

It has little use to sample multiple species trees from a gene tree, as these are identical for instant speciation models. In this example, we do sample a species trees twice from the true gene tree. This results in two identical species trees, as can be verified by figure 27:

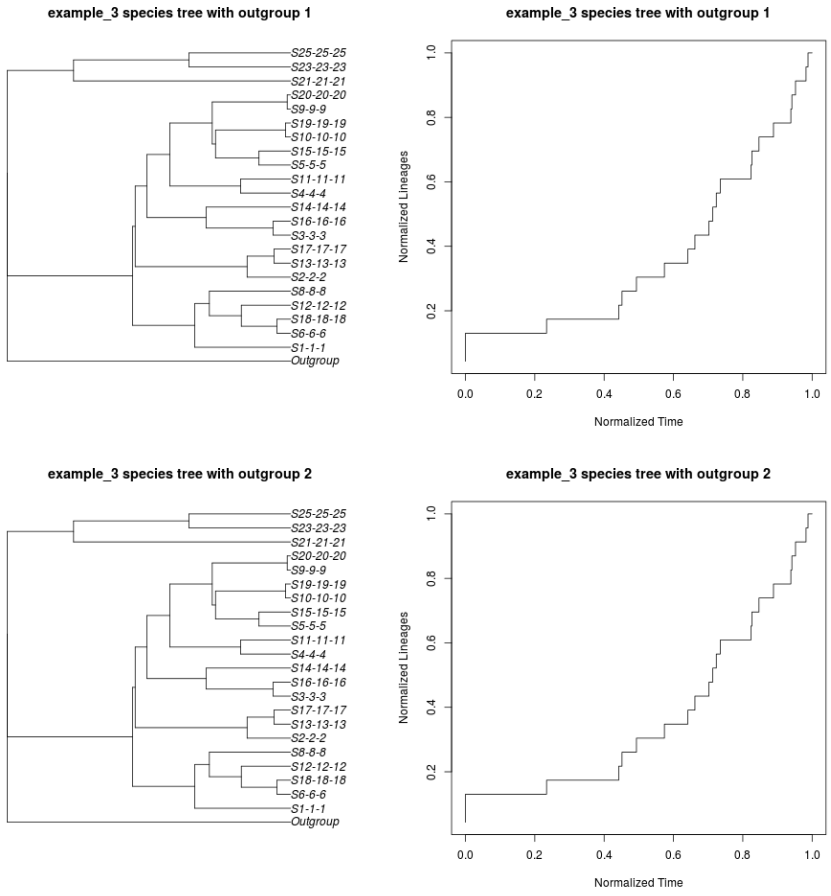

Figure 27: Example 3 species tree and nLTT plot of that species tree

Gene tree and all species trees Doing this multiple times anyways:

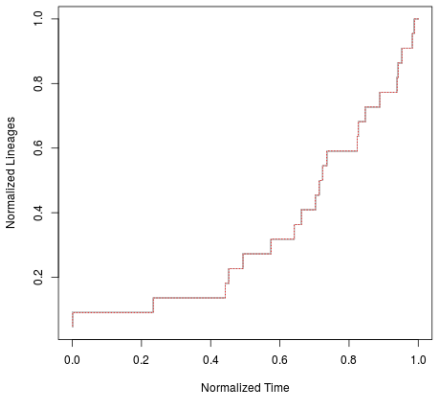

Figure 28: Example 3 gene tree (black solid line) and possible sampled species tree

A.6.3 Alignment

From these (in this case identical) species trees, multiple alignments can be simulated. In this example, for every species tree, there are two alignments simulated. Because in this example, there are two species sampled, and every species tree has two alignments simulated, this results in four alignments.

Figure 29 shows a visualisation of the simulated alignments of our example: Each alignment is different.

A.6.4 Posterior

In this example, for each alignment, we do two BEAST2 runs, instead of one. It is expected (or: it should be the case) that both runs create a similar posterior. Because this example has 2 species trees, 2 alignments per species tree and 2 BEAST2 runs per alignment, 8 different-yet-similar posteriors are expected.

Last tree The last trees picked from the eight different posterior are shown in figure 30:

The species tree picked from the posterior all are different, but not too different (figure 27).

Do posteriors contain paraphylies? Never, which is expected, as there is no protractedness

Seperate nLTT plots Each its nLTT plot is compared to the original species tree, which is shown in figure 32:

In all cases, we can observe that the lines are close, but do not match. Also this is expected, due to stochasticity in the MCMC sampling.

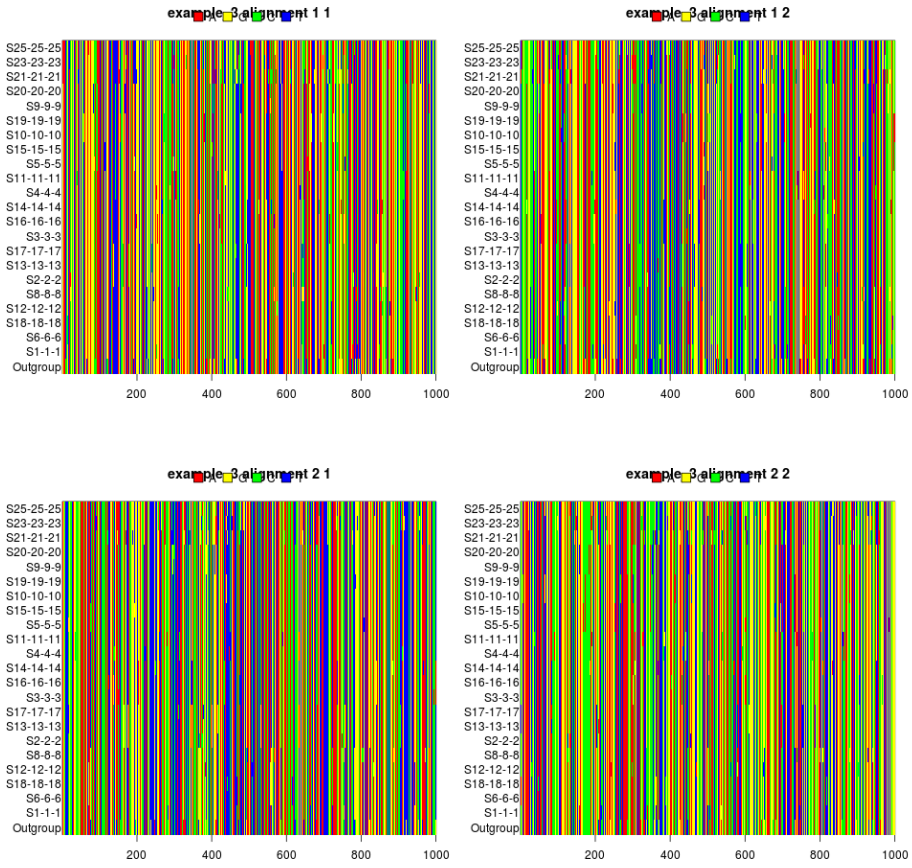

Figure 29: Example 3 alignments

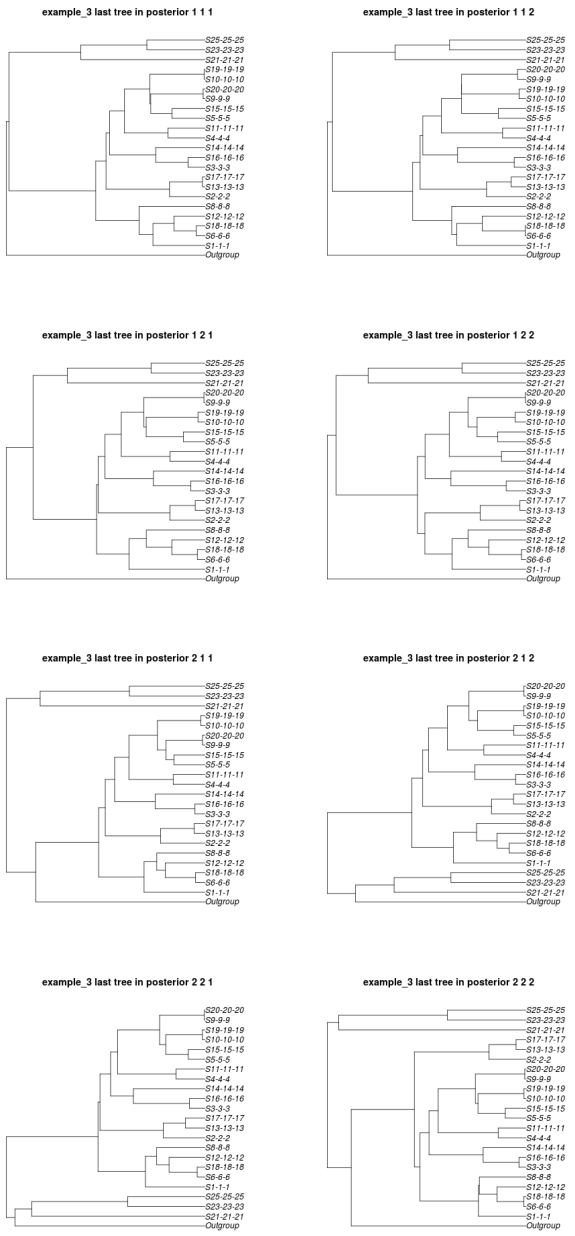

Figure 30: The random species trees from example 3 its posteriors

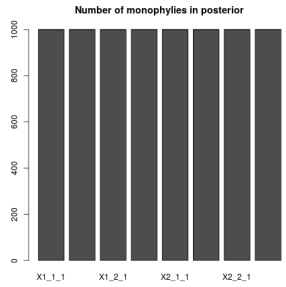

Figure 31: Count of the number of monophylies in all example 3 its BEAST2 posteriors

nLTT plots of BEAST runs To see how well the posteriors match, the nLTT plots of the BEAST runs are plotted in the same graphs:

The BEAST2 runs appear to have converged to a similar posterior.

nLTT plots of alignments Do the different alignments matter?

The different alignments do matter.

nLTT plots of species trees Do the different species tree matter?

A bit.

Errors For each tree in each of the posteriors, the error is put in a histogram as shown by figure 36:

But this does not tell us where the errors have been made: near the crown, near the tips, or in between? To do so, we plot the average nLTT plot of the posterior and compare it to the true species tree its nLTT, as shown in figure 37:

A.7 Example 4: Strong protractedness with replicates

This example is an extension of the previous examples, with the differences that:

- From the (same) gene tree, multiple species trees are sampled
- Per species tree, multiple DNA alignments are simulated
- Per DNA alignment, multiple BEAST2 runs are performed

A.7.1 Gene tree

The first step is to simulate a gene tree. Because the RNG seed is at the same value as example #1, exactly the same gene tree will be obtained, as can be seen in figure 38:

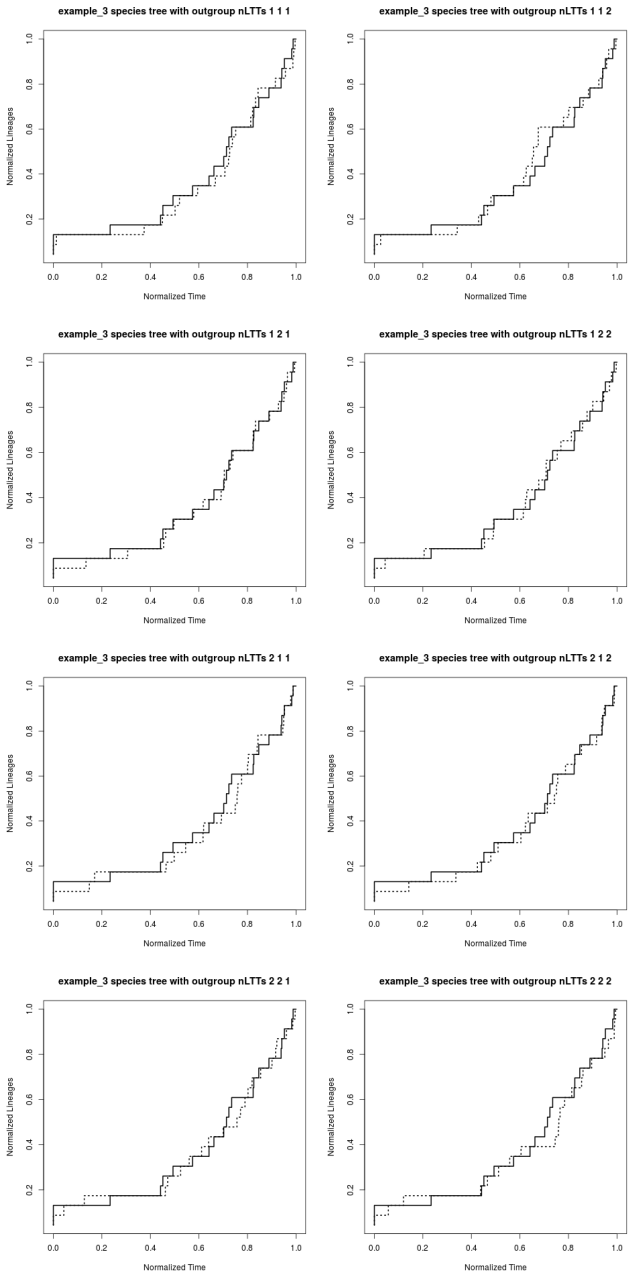

Figure 32: All example 3 its nLTT plots compare to the true species tree nLTT. The solid black line is the 'true' or initial (species) tree (with outgroup), the dotted line is one of the trees in the BEAST2 posterioir

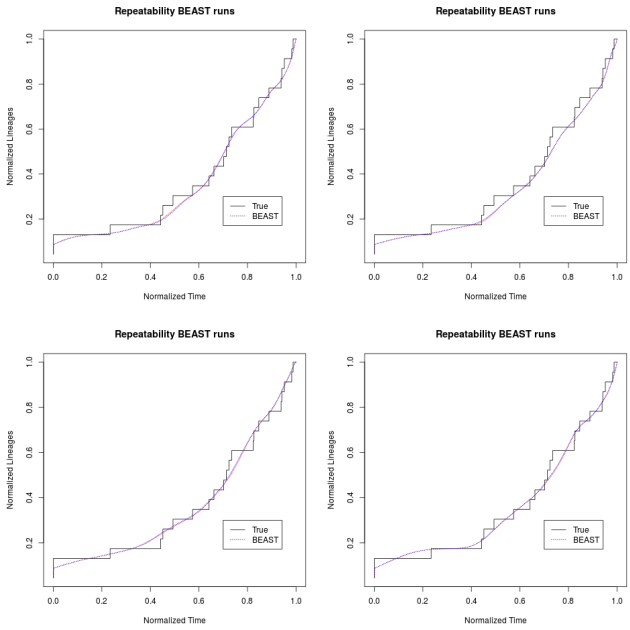

Figure 33: All example 3 its nLTT plots compare to the true species tree nLTT. The solid black line is the 'true' or initial (species) tree (with outgroup), the dotted lines are the average nLTT of the BEAST2 posterior

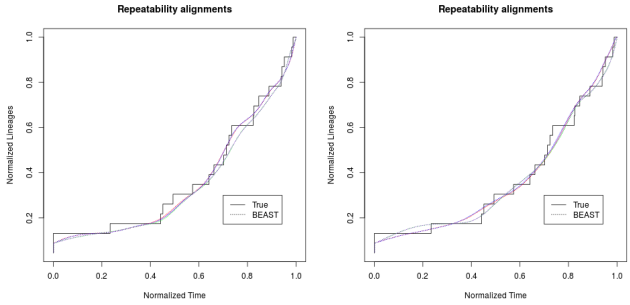

Figure 34: All example 3 its nLTT plots compare to the true species tree nLTT. The solid black line is the 'true' or initial (species) tree (with outgroup), the dotted lines are the average nLTT of the BEAST2 posterior of both alignment both its runs

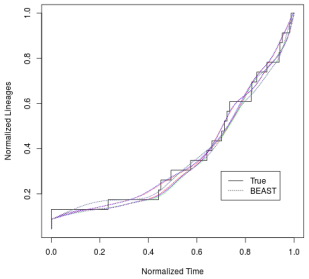

Figure 35: All example 3 its nLTT plots compare to the true species tree nLTT. The solid black line is the 'true' or initial (species) tree (with outgroup), the dotted lines are the average nLTT of the BEAST2 posterior of both species trees of both alignment both its runs

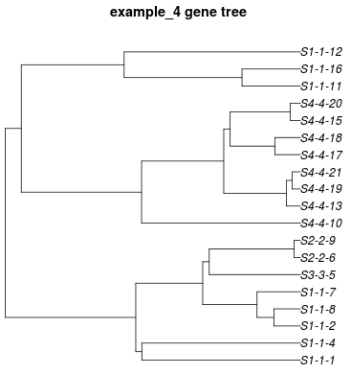

Figure 38: Example 4 gene tree. The taxon labels are S[genus]-[species]-[sub-species]'

A.7.2 Species tree

As there are multiple sub-species present for the same species, it makes sense to sample multiple species trees from a gene tree. In this example, we do sample a species trees twice from the true gene tree. This results in two different species trees, as can be verified by figure 39.

It is a bit of bad luck that the random number generator picked two very similar species trees: would, instead of

Gene tree and all species trees Doing this multiple times:

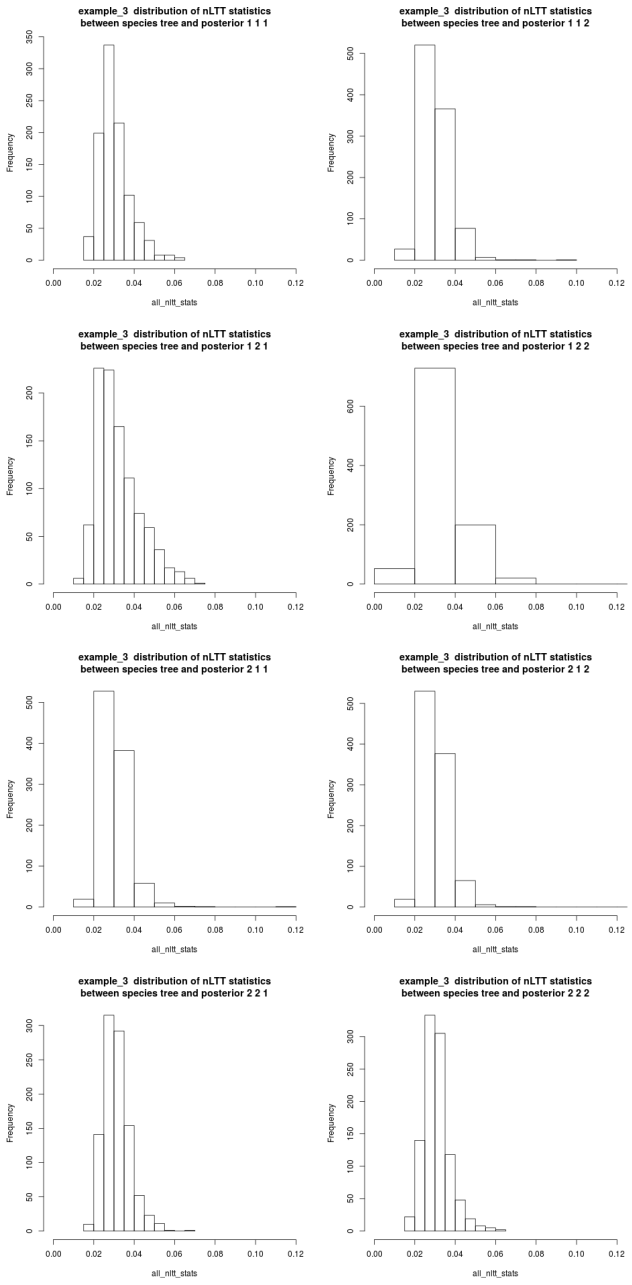

Figure 36: Histogram of the nLTT statistic between the true species tree and the trees in the posterior

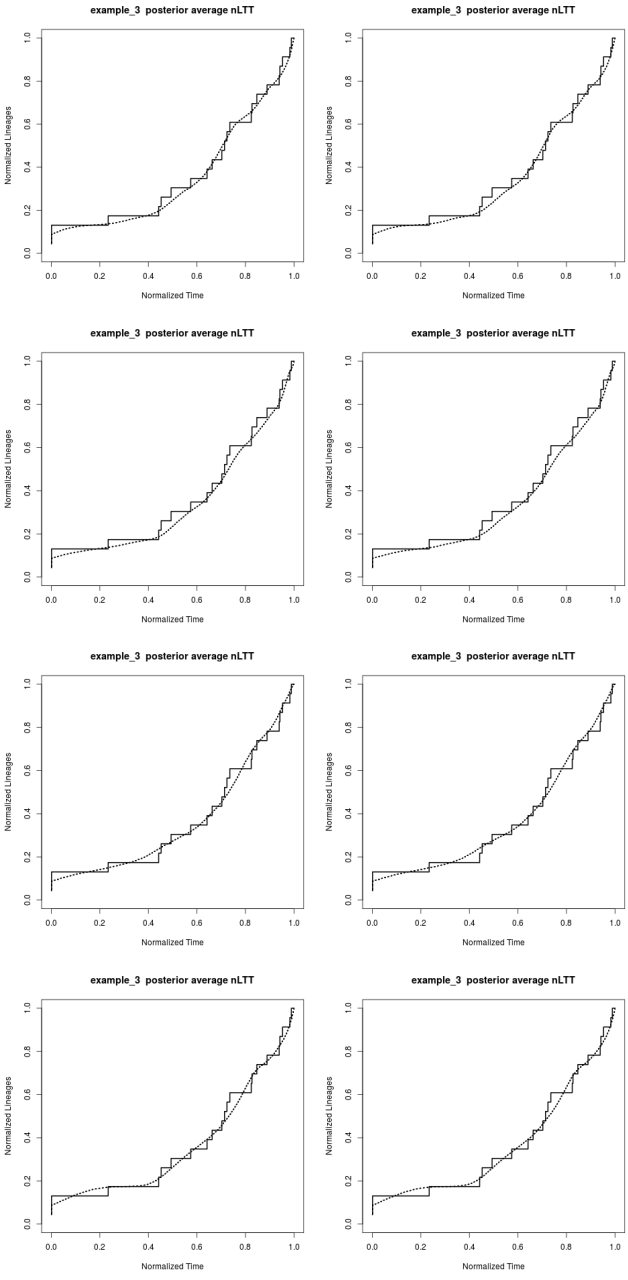

Figure 37: Average nLTT of all the trees in the posterior. Solid line: nLTT of true species tree. Dotted line: average nLTT of all posterior trees

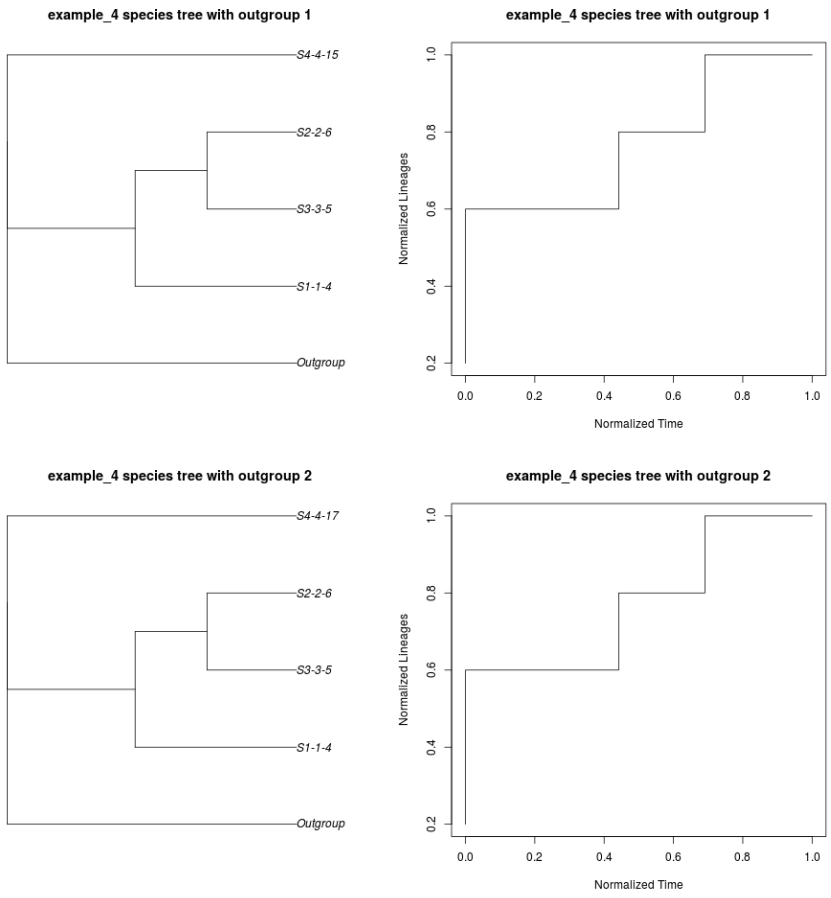

Figure 39: Example 4 species tree and nLTT plot of that species tree

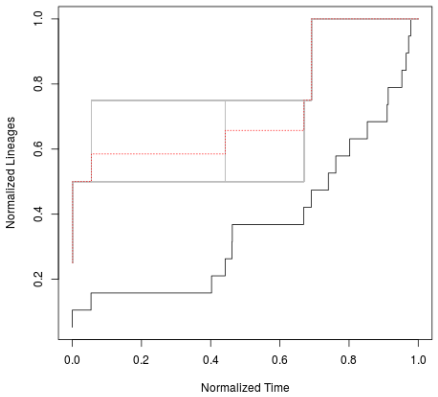

Figure 40: Example 4 gene tree (black solid line) and possible sampled species tree

A.7.3 Alignment

From these (in this case identical) species trees, multiple alignments can be simulated. In this example, for every species tree, there are two alignments simulated. Because in this example, there are two species sampled, and every species tree has two alignments simulated, this results in four alignments.

Figure 41 shows a visualisation of the simulated alignments of our example. Each alignment is different.

A.7.4 Posterior

In this example, for each alignment, we do two BEAST2 runs, instead of one. It is expected (or: it should be the case) that both runs create a similar posterior. Because this example has 2 species trees, 2 alignments per species tree and 2 BEAST2 runs per alignment, 8 different-yet-similar posteriors are expected.

Last tree The last trees picked from the eight different posterior are shown in figure 42.

The species tree picked from the posterior all are different, but not too different (figure 39).

Do posteriors contain paraphylies? Never BEAST2 creates monophyletic trees.

Seperate nLTT plots Each its nLTT plot is compared to the original species tree, which is shown in figure 44.

In all cases, we can observe that the lines are close, but do not match. Also this is expected, due to stochasticity in the MCMC sampling.

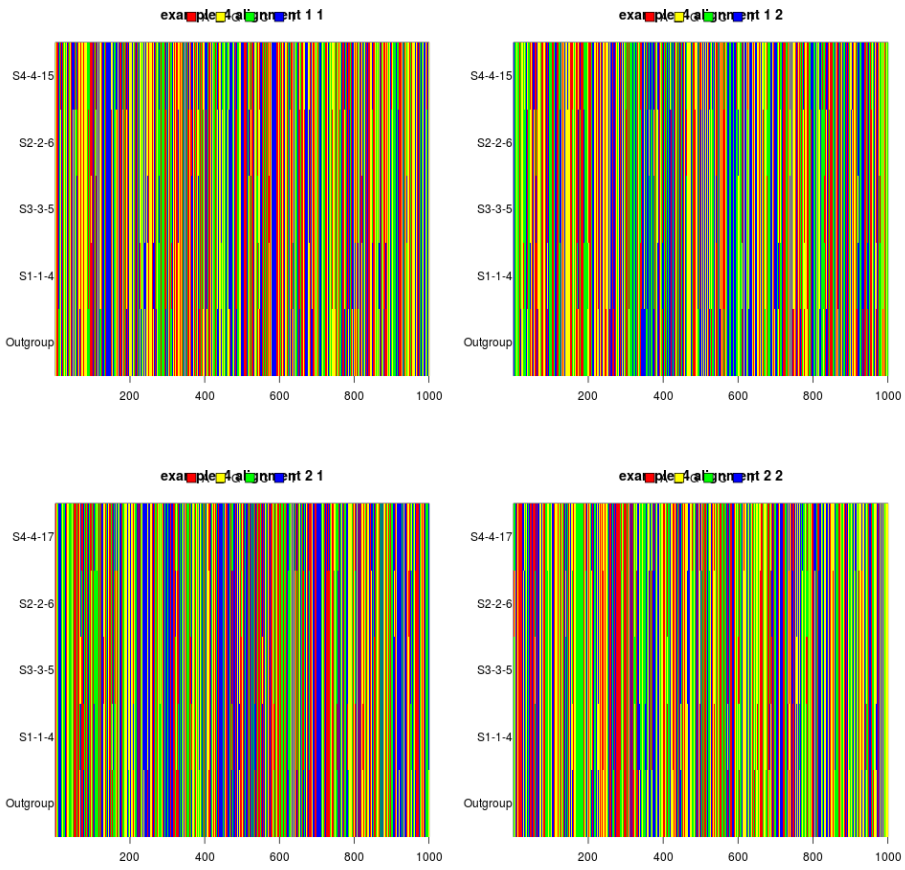

Figure 41: Example 4 alignments

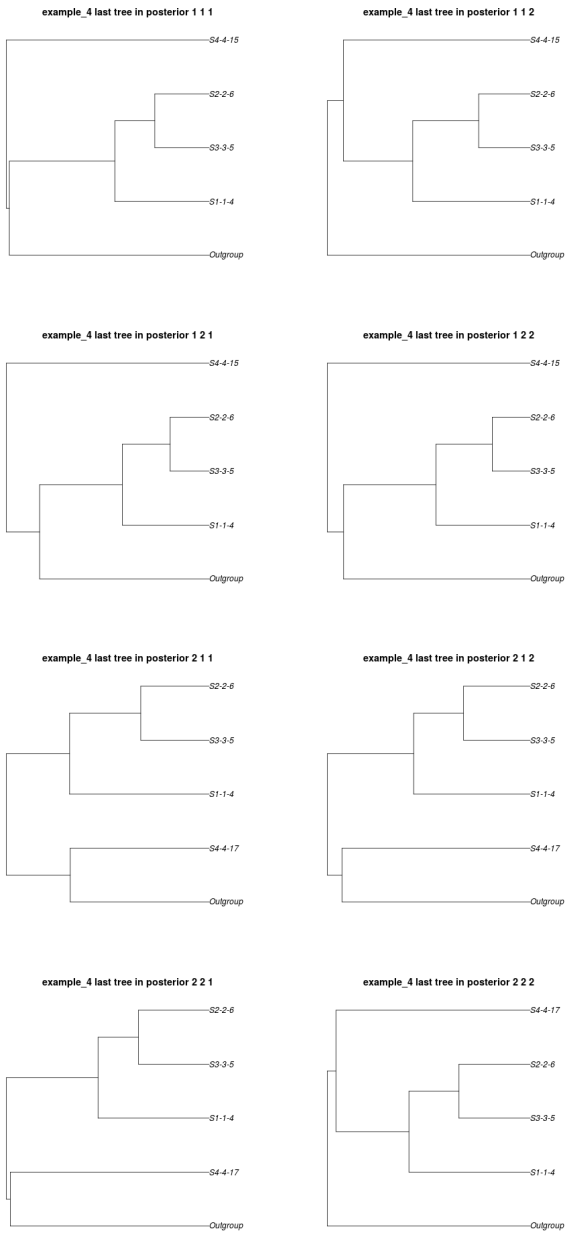

Figure 42: The random species trees from example 3 its posteriors

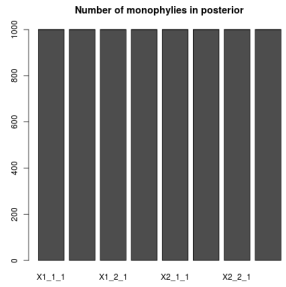

Figure 43: Count of the number of monophylies in all example 4 its BEAST2 posteriors

nLTT plots of BEAST runs To see how well the posteriors match, the nLTT plots of the BEAST runs are plotted in the same graphs:

The BEAST2 runs appear to have converged to a similar posterior.

nLTT plots of alignments Do the different alignments matter?

The different alignments do matter.

nLTT plots of species trees Do the different species tree matter?

A bit.

Error For each tree in each of the posteriors, the error is put in a histogram as shown by figure 48.

But this does not tell us where the errors have been made: near the crown, near the tips, or in between? To do so, we plot the average nLTT plot of the posterior and compare it to the true species tree its nLTT, as shown in figure 49.

A.8 Comparing example 3 and 4

Both examples showed a histogram of the error between true species tree and posterior trees. We expect these errors to have a different distribution: example 3 assumed instant speciation, which suits the algorithm best, where example 4 has a strong protractedness. Plotting both histograms in the same plot results in figure 50:

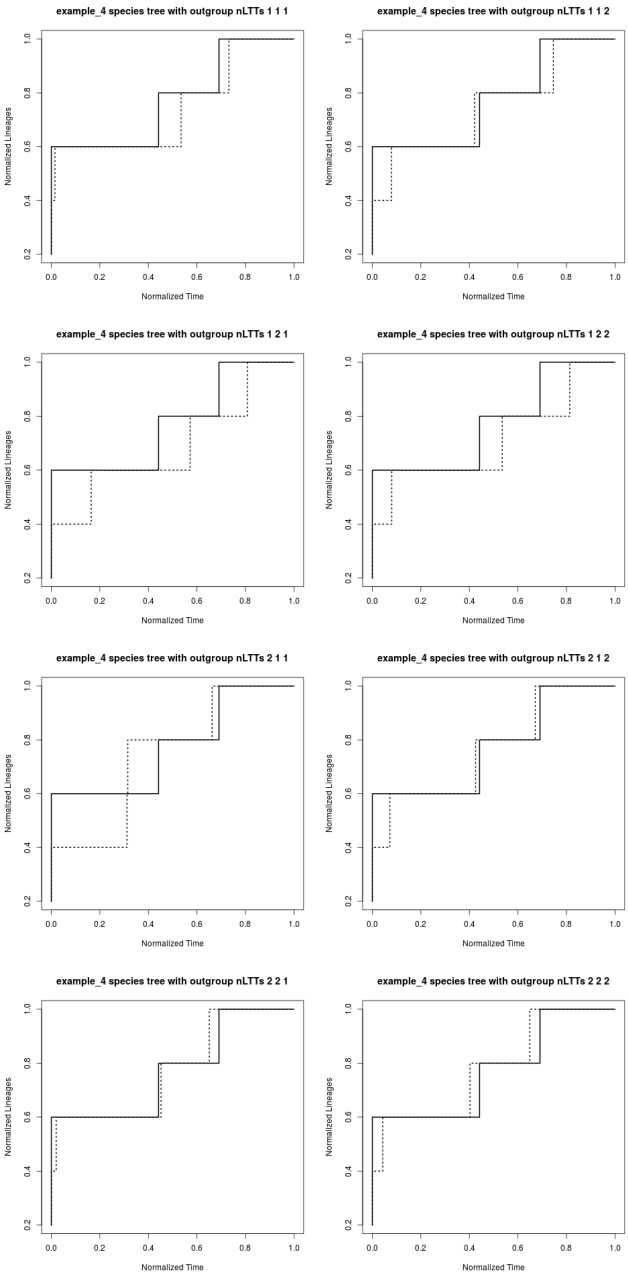

Figure 44: All example 4 its nLTT plots compare to the true species tree nLTT. The solid black line is the 'true' or initial (species) tree (with outgroup), the dotted line is one of the trees in the BEAST2 posterioir

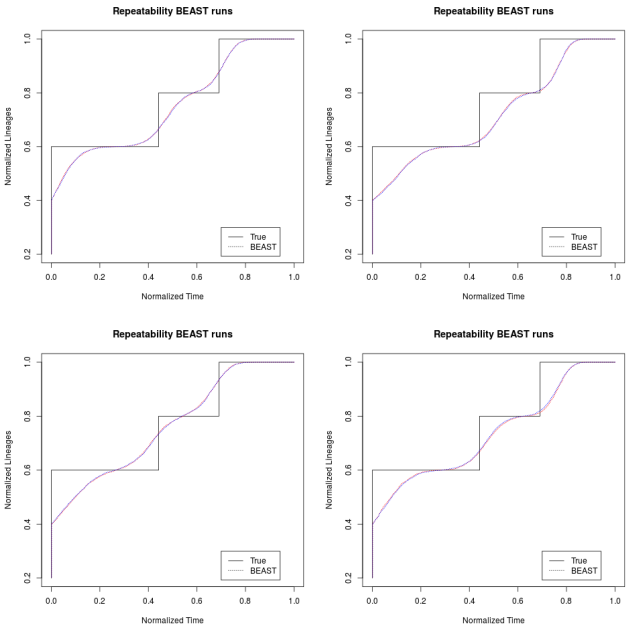

Figure 45: All example 4 its nLTT plots compare to the true species tree nLTT. The solid black line is the 'true' or initial (species) tree (with outgroup), the dotted lines are the average nLTT of the BEAST2 posterior

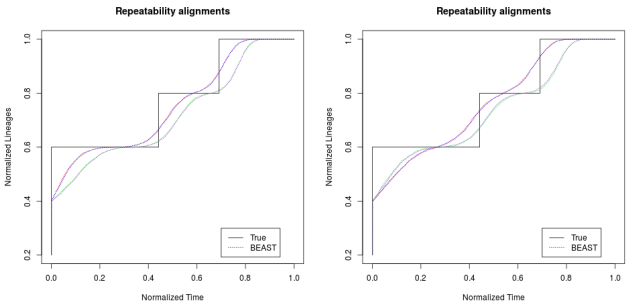

Figure 46: All example 4 its nLTT plots compare to the true species tree nLTT. The solid black line is the 'true' or initial (species) tree (with outgroup), the dotted lines are the average nLTT of the BEAST2 posterior of both alignment both its runs

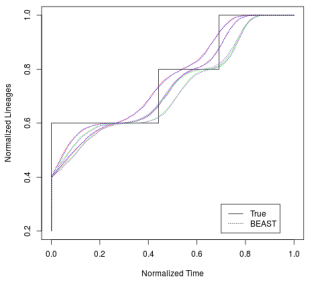

Figure 47: All example 4 its nLTT plots compare to the true species tree nLTT. The solid black line is the 'true' or initial (species) tree (with outgroup), the dotted lines are the average nLTT of the BEAST2 posterior of both species trees of both alignment both its runs

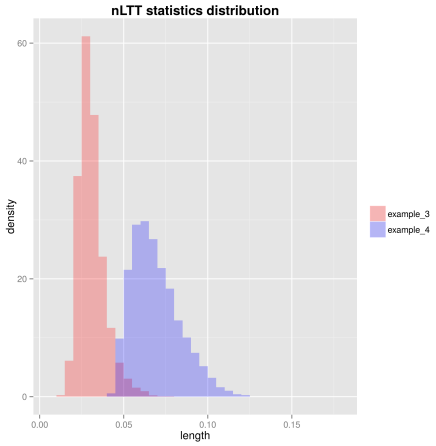

Figure 50: Histogram of the nLTT statistic of the two examples

It can be observed that these error distributions have a different median. This means that there is an observable higher error being made when the true species tree is protracted.

Because of the number of repeats, there are more data points, as all are lumped together.

This visualization does not tell use how the error is made: is the error concentrated at the root, middle or tips of the tree? To locate this, the average nLTT plots of the posterior is shown in figure 51:

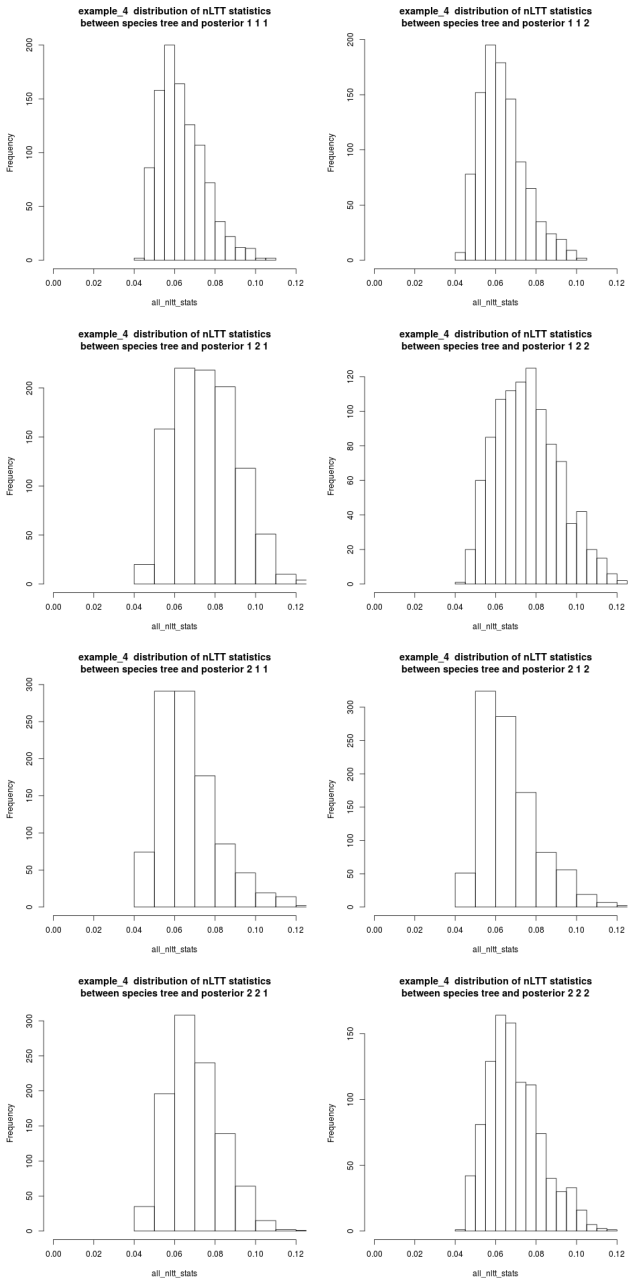

Figure 48: Histogram of the nLTT statistic between the true species tree and the trees in the posterior

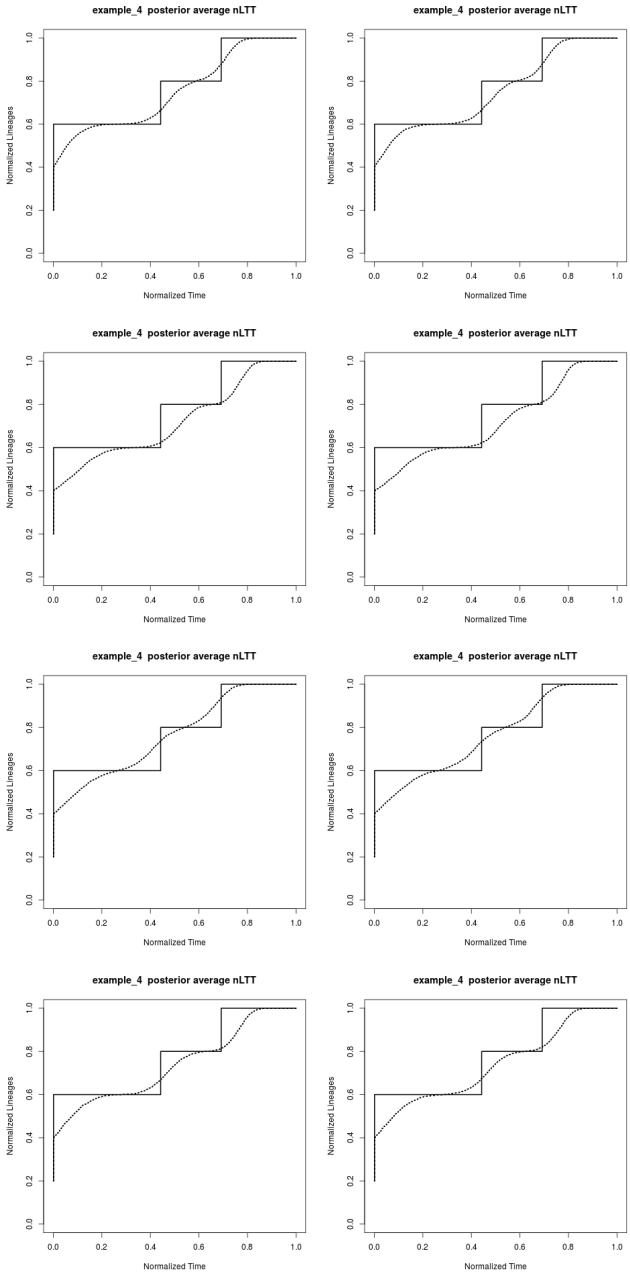

Figure 49: Average nLTT of all the trees in the posterior. Solid line: nLTT of true species tree. Dotted line: average nLTT of all posterior trees

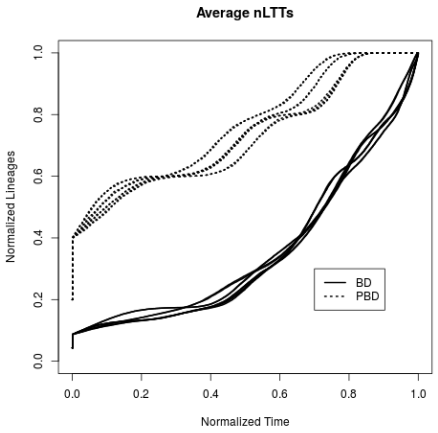

Figure 51: Average nLTTs of the posteriors of example #3 (weak protractedness, solid line) and example #4 (strong protractedness, dotted line)

B Bash scripts

Running scripts locally:

- `create_parameters_[X].sh`
- `./do_simulation_[X].sh`
- `./analyse_[X].sh`

Running scrips on cluster:

- `sbatch ./create_parameters_[X]_job.sh`
- `sbatch ./do_simulation_[X]_job.sh`
- `sbatch ./analyse_[X]_job.sh`

Formula:

$$\text{filename} = \left\{ \begin{array}{c} \text{create\_parameters} \\ \text{do\_simulation} \\ \text{analyse} \end{array} \right\} - \left\{ \begin{array}{c} \text{toy\_examples} \\ \text{examples} \\ \text{article} \end{array} \right\} - \left\{ \begin{array}{c} \text{.sh} \\ \text{job.sh} \end{array} \right\}$$

B.1 analyse_article.sh

Algorithm 1 The script

```
#!/bin/bash

# Individual file
for filename in `ls article_*.RDa`
do
  Rscript analyse.R $filename
done

# Compare BEAST2 runs: how similar are two runs?
Rscript plot_beast_repeatabilities.R

# Compare alignments: how similar are two alignments?
Rscript plot_alignment_repeatabilities.R

# Compare species trees: how similar are species trees?
Rscript plot_species_tree_repeatabilities.R

# Compare species trees: count the number of paraphylies
Rscript plot_paraphyly_count.R

# Observe the effect of sampling from a (optionally paraphyletic) gene
    tree
Rscript plot_gene_tree_to_species_trees.R
```

B.2 analyse_article_job.sh

Algorithm 2 The script

```
#!/bin/bash
#SBATCH --time=0:10:00
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --ntasks=1
#SBATCH --mem=100000
#SBATCH --job-name=analyse_article
#SBATCH --mail-type=BEGIN,END
#SBATCH --output=analyse_article_job.log
module load R
./analyse_article.sh
```

B.3 analyse_examples.sh

This bash script analyzes the the results of the worked-out examples (chapter A) and is displayed in algorithm 3:

Algorithm 3 The 'analyse_examples.sh' script

```
#!/bin/bash
Rscript analyse.R example_1.RDa
Rscript analyse.R example_2.RDa
Rscript analyse.R example_1.RDa example_2.RDa
mv multi_average_nltts.png example_12_average_nltts.png
mv multi_nltt_stats_histogram.png example_12_nltt_stats_histogram.png
Rscript analyse.R example_3.RDa
Rscript analyse.R example_4.RDa
Rscript analyse.R example_3.RDa example_4.RDa
mv multi_average_nltts.png example_34_average_nltts.png
mv multi_nltt_stats_histogram.png example_34_nltt_stats_histogram.png
```

The first line of the file indicates that it is a bash script. Then the R script file 'do_analyze.R' (see chapter C.1) is called with one or more parameter filenames.

B.4 analyse_examples_job.sh

Algorithm 4 The script

```
#!/bin/bash
#SBATCH --time=0:10:00
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --ntasks=1
#SBATCH --mem=100000
#SBATCH --job-name=analyse_examples
#SBATCH --mail-type=BEGIN,END
#SBATCH --output=analyse_examples_job.log
module load R
./analyse_examples.sh
```

B.5 analyse_toy_examples.sh

Algorithm 5 The script

```
#!/bin/bash
Rscript analyse.R toy_example_1.RDa
Rscript analyse.R toy_example_2.RDa
Rscript analyse.R toy_example_1.RDa toy_example_2.RDa
mv multi_average_nltts.png toy_example_12_average_nltts.png
mv multi_nltt_stats_histogram.png toy_example_12_nltt_stats_histogram.
    png
Rscript analyse.R toy_example_3.RDa
Rscript analyse.R toy_example_4.RDa
Rscript analyse.R toy_example_3.RDa toy_example_4.RDa
mv multi_average_nltts.png toy_example_34_average_nltts.png
mv multi_nltt_stats_histogram.png toy_example_34_nltt_stats_histogram.
    png
```

B.6 analyse_toy_examples_job.sh

Algorithm 6 The script

```
#!/bin/bash
#SBATCH --time=0:01:00
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --ntasks=1
#SBATCH --mem=100000
#SBATCH --job-name=analyse_toy_examples
#SBATCH --mail-type=BEGIN,END
#SBATCH --output=analyse_toy_examples_job.log
module load R
./analyse_toy_examples.sh
```

B.7 check_parameters_examples.sh

This bash script shows how to check the example parameter files:

Algorithm 7 The 'check_parameters_examples.sh' script

```
#!/bin/bash
for filename in `ls example_*.RDa`
do
  Rscript check_parameter_file.R $filename
done
```

The bash script indicates in its first line that it is a bash script. The next two lines call an R script file called 'check_parameter_file.R' (see chapter C.2) with the parameter filename.

Algorithm 8 Output of 'check_example_parameter_files.sh'

B.8 create_parameters_article.sh

Algorithm 9 The 'create_parameters_article.sh' script

```
#!/bin/bash
b_index=0
for b in 0.1 0.5 1.0
do
  lambda_index=0
  for lambda in 0.1 0.3 1.0 1000000
  do
    mu_index=0
    for mu in 0.0 0.1 0.2 0.4
    do
      r_index=0
      for r in 0.1 0.01 0.001
      do
        l_index=0
        for l in 1000 10000 100000
        do
          rng_seed=1
          crown_age=15
          n_species_trees=2
          n_alignments=2
          mcmc_length=1000000
          n_beast_runs=2
          filename='article_'$b_index'_'$lambda_index'_'$mu_index'_'
              $r_index'_'$l_index'.RDa'
          Rscript create_parameter_file.R $rng_seed $b $b $lambda $mu
              $mu $crown_age $n_species_trees $r $n_alignments $l
              $mcmc_length $n_beast_runs $filename
          l_index=$((l_index+1))
        done # l
        r_index=$((r_index+1))
      done # r
      mu_index=$((mu_index+1))
    done # mu
    lambda_index=$((lambda_index+1))
  done # lambda
  b_index=$((b_index+1))
done # b
```

The first line indicates that this file is a bash script. Then it creates all parameter files.

B.9 create_parameters_article_job.sh

Algorithm 10 The 'create_parameters_article_job.sh' script

```
#!/bin/bash
#SBATCH --time=1:00:00
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --ntasks=1
#SBATCH --mem=100000
#SBATCH --job-name=create_parameters_article
#SBATCH --mail-type=BEGIN,END
#SBATCH --output=create_parameters_article_job.log
module load R
./create_parameters_article.sh
```

A Peregrine cluster job script, calling 'create_parameters_article.sh' (chapter B.8).

On the Peregrine cluster, call this with:

```
sbatch ./create_parameters_article_job.sh
```

B.10 create_parameters_examples.sh

Algorithm 11 The 'create_parameters_examples.sh' script

```
#!/bin/bash
Rscript create_parameter_file.R 1 0.5 0.5 1000000 0.1 0.1 5 1 0.01 1
    1000 1000000 1 example_1.RDa
Rscript create_parameter_file.R 1 0.5 0.5      0.1 0.1 0.1 5 1 0.01 1
    1000 1000000 1 example_2.RDa
Rscript create_parameter_file.R 1 0.5 0.5 1000000 0.1 0.1 5 2 0.01 2
    1000 1000000 2 example_3.RDa
Rscript create_parameter_file.R 1 0.5 0.5      0.1 0.1 0.1 5 2 0.01 2
    1000 1000000 2 example_4.RDa
```

The first line indicates that this file is a bash script. The next two lines call the R script file called 'create_parameter_file.R' (see chapter TODO) with the 14 parameter arguments.

B.11 create_parameters_examples_job.sh

Algorithm 12 The script

```
#!/bin/bash
#SBATCH --time=0:10:00
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --ntasks=1
#SBATCH --mem=100000
#SBATCH --job-name=create_parameters_example
#SBATCH --mail-type=BEGIN,END
#SBATCH --output=create_parameters_examples_job.log
module load R
./create_parameters_examples.sh
```

B.12 create_parameters_toy_examples.sh

Algorithm 13 The script

```
#!/bin/bash
Rscript create_parameter_file.R 1 0.5 0.5 1000000 0.1 0.1 5 1 0.01 1
    1000 10000 1 toy_example_1.RDa
Rscript create_parameter_file.R 1 0.5 0.5      0.1 0.1 0.1 5 1 0.01 1
    1000 10000 1 toy_example_2.RDa
Rscript create_parameter_file.R 1 0.5 0.5 1000000 0.1 0.1 5 2 0.01 2
    1000 10000 2 toy_example_3.RDa
Rscript create_parameter_file.R 1 0.5 0.5      0.1 0.1 0.1 5 2 0.01 2
    1000 10000 2 toy_example_4.RDa
```

B.13 create_parameters_toy_examples_job.sh

Algorithm 14 The script

```
#!/bin/bash
#SBATCH --time=0:01:00
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --ntasks=1
#SBATCH --mem=100000
#SBATCH --job-name=create_parameters_toy_example
#SBATCH --mail-type=BEGIN,END
#SBATCH --output=create_parameters_toy_examples_job.log
module load R
./create_parameters_toy_examples.sh
```

B.14 do_simulation_article.sh

This scripts runs all simulations of the article, as shown in algorithm 15:

Algorithm 15 The 'do_simulation_article.sh' script

```
#!/bin/bash
for filename in `ls article_*.RDa`
do
  Rscript do_simulation.R $filename
done
```

The bash script indicates in its first line that it is a bash script, then call an R script file called 'do_simulation.R' (see chapter C.5) with the parameter file its name.

B.15 do_simulation_article_job.sh

This scripts sbatches all simulations of the article, as shown in algorithm 16:

Algorithm 16 The 'do_simulation_article_job.sh' script

```
#!/bin/bash
for filename in `ls article_*.RDa`
do
  sbatch ./sbatch_me.sh do_simulation.R $filename
done
```

The bash script indicates in its first line that it is a bash script, then call an R script file called 'do_simulation.R' (see chapter C.5) with the parameter file its name.

B.16 do_simulation_examples.sh

This scripts locally runs all simulations of the worked-out examples, as shown in algorithm 17:

Algorithm 17 The 'do_simulation_examples.sh' script

```
#!/bin/bash
for filename in `ls example_*.RDa`
do
  Rscript do_simulation.R $filename
done
```

The bash script indicates in its first line that it is a bash script, then call an R script file called 'do_simulation.R' (see chapter C.5) with the parameter file its name.

B.17 do_simulation_examples_job.sh

This scripts sbatches all simulations of the article, as shown in algorithm 18:

Algorithm 18 The 'do_simulation_examples_job.sh' script

```
#!/bin/bash
for filename in `ls example_*.RDa`
do
  sbatch ./sbatch_me.sh $filename
done
```

The bash script indicates in its first line that it is a bash script, then call an R script file called 'do_simulation.R' (see chapter C.5) with the parameter file its name.

B.18 do_simulation_toy_examples.sh

Algorithm 19 The script

```
#!/bin/bash
for filename in `ls toy_example_*.RDa`
do
  Rscript do_simulation.R $filename
done
```

B.19 do_simulation_toy_examples_job.sh

Algorithm 20 The script

```
#!/bin/bash
for filename in `ls toy_example_*.RDa`
do
  sbatch ./sbatch_me.sh $filename
done
```

B.20 sbatch_me.sh

This scripts runs a simulation on a computer cluster. It is shown in algorithm 21:

Algorithm 21 The 'sbatch_me.sh' script

```
#!/bin/bash
# Call with e.g. 'sbatch ./sbatch_me do_simulation.R example1.RDa'
#SBATCH --time=240:00:00
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --ntasks=1
#SBATCH --mem=100000
#SBATCH --job-name=article
#SBATCH --mail-type=BEGIN,END
module load R Beast
Rscript $1 $2
```

Algorithm 22 The 'analyse.R' script

```
args <- commandArgs(TRUE)
if (length(args) == 0) {
  print("Please supply the parameter filename(s)" )
  stop()
}

for (filename in args) {
  if (!file.exists(filename)) {
    print(paste(filename,": not found"), sep="")
    print("Please supply the filename(s) of (an) existing file(s)" )
    stop()
  }
}

if (length(args) == 1) {
  print("Analyzing one parameter file")
  source("~/GitHubs/R/Peregrine/analyse_single.R")
  analyse_single(args[1])
} else {
  print("Analyzing multiple parameter files")
  source("~/GitHubs/R/Peregrine/analyse_multi.R")
  analyse_multi(args)
}
```

The bash script indicates in its first line that it is a bash script. Then it sets up the computer cluster parameters, loads the modules for BEAST2 and R, then runs a simulation for the parameter filename given as its first argument when called.

C R scripts

Files that are run standalone and can be called from the command line.

C.1 analyse.R

Performs the analysis and creates the result graphs, as shown in algorithm 22:

This script it behavior depends on the number of arguments supplied from the command line:

- one parameter filename: create the graphs of a single run using the R function 'analyse_single' (see chapter D.6)
- multiple parameter filenames: create the graphs of mutiple runs using the R function 'analyse_multi' (see chapter D.5)

C.2 check_parameter_file.R

The R script 'check_parameter_file.R' file prints out the parameters and its code is shown in algorithm 23:

Algorithm 23 The 'check_parameter_file.R' script

```
if (length(commandArgs(TRUE)) != 1) {
  print("Please supply a parameter filename" )
  stop()
}

filename <- commandArgs(TRUE)[1]

if (!file.exists(filename)) {
  print("Please supply the filename of an existing file" )
  stop()
}

source("~/GitHubs/R/Peregrine/load_parameters_from_file.R")

file <- load_parameters_from_file(filename)
print(t(file$parameters[2,]))
```

First, 'check_parameter_file.R' checks if the parameter exists, after which it displays the parameter values.

The bash script 'check_example_parameter_files.sh' (see chapter B.7) shows how to the example parameter files are checked.

C.3 collect_parameters.R

Algorithm 24 The 'collect_parameters.R' script

```
source("~/GitHubs/R/Peregrine/load_parameters_from_file.R")

setwd("~/GitHubs/R/Peregrine")

text <- NULL

for (filename in list.files(path = ".", pattern = "*.RDa")) {
  file <- load_parameters_from_file(filename)
  text <- rbind(text,c(filename,as.numeric(file$parameters[2,])))
}

# Create table headings *after* the creation of the table (order is
    important here)
for (filename in list.files(path = ".", pattern = "*.RDa")) {
  file <- load_parameters_from_file(filename)
  #print(colnames(file$parameters[2,]))
  colnames(text) <- c("filename",colnames(file$parameters[2,]))
  break
}

write.csv(file="collect_parameters.csv", x=text)

?list.files


file <- load_parameters_from_file("example_1.RDa")
print(file$parameters[2,])
show(file$parameters[2,])
text <- rbind(text,file$parameters[2,])

nrow(file$parameters[2,])
ncol(file$parameters[2,])

as.numeric(file$parameters[2,])
```

Creates a .csv with all parameter files in the folder.

C.4 create_parameter_file.R

Algorithm 25 The 'create_parameter_file.R' script

```
argv <- commandArgs(trailingOnly = TRUE)
if (length(argv) == 0) {
  argv <- c("42","0.5","0.5","1000000","0.1","0.1","5","2","0.01","2","
      1000","1000000","2","1.RDa")
}
if (length(argv) != 14) {
  print("Please supply 14 arguments, for example:" )
  print("Rscript create_parameter_file.R 42 0.5 0.5 1000000 0.1 0.1 5 2
      0.01 2 1000 1000000 2 1.RDa" )
  stop()
}
for (i in seq(1,13)) {
  if (is.na(as.numeric(argv[i]))) {
    print("Please supply 13 values, for example:" )
    print("Rscript create_parameter_file.R 42 0.5 0.5 1000000 0.1 0.1 5
        2 0.01 2 1000 1000000 2 1.RDa" )
    stop()
  }
}
source("~/GitHubs/R/Peregrine/save_parameters_to_file.R")
save_parameters_to_file(
  rng_seed = as.numeric(argv[1]),
  species_initiation_rate_good_species = as.numeric(argv[2]),
  species_initiation_rate_incipient_species = as.numeric(argv[3]),
  speciation_completion_rate = as.numeric(argv[4]),
  extinction_rate_good_species = as.numeric(argv[5]),
  extinction_rate_incipient_species = as.numeric(argv[6]),
  age = as.numeric(argv[7]),
  n_species_trees_samples = as.numeric(argv[8]),
  mutation_rate = as.numeric(argv[9]),
  n_alignments = as.numeric(argv[10]),
  sequence_length = as.numeric(argv[11]),
  mcmc_chainlength = as.numeric(argv[12]),
  n_beast_runs = as.numeric(argv[13]),
  filename = argv[14]
)
```

Creates a parameter file from the parameters supplied. First, 'create_parameter_file.R' checks if there are the right amount of parameters, with the correct form. The last argument is the parameter file its filename. As the parameter file uses the R data format, it is fitting to use the '.RDa' extension.

Most of the work is then pushed forward to the 'save_parameters_to_file' function, which is described in chapter D.18.

The bash script 'create_example_parameter_files.sh' (see chapter B.10) shows how to create the example parameter files.

C.5 do_simulation.R

Performs the simulation for a parameter file, as shown in algorithm 26:

Algorithm 26 The 'do_simulation.R' script

```
if (length(commandArgs(TRUE)) != 1) {
  print("Please supply a parameter filename" )
  stop()
}

filename <- commandArgs(TRUE)[1]

if (!file.exists(filename)) {
  print("Please supply the filename of an existing file" )
  stop()
}

source("~/GitHubs/R/Peregrine/load_parameters_from_file.R")
print(t(load_parameters_from_file(filename)$parameters[2,]))
source("~/GitHubs/R/Peregrine/add_pbd_output.R")
print("Adding PBD output")
add_pbd_output(filename)
source("~/GitHubs/R/Peregrine/add_species_trees_with_outgroup.R")
print("Adding species trees with outgroup")
add_species_trees_with_outgroup(filename)
source("~/GitHubs/R/Peregrine/add_alignments.R")
print("Adding alignment(s)")
add_alignments(filename)
source("~/GitHubs/R/Peregrine/add_posteriors.R")
print("Creating BEAST2 posteriors")
add_posteriors(filename)
```

'do_simulation.R' first checks if the supplied function argument is an existing filename. Then, it runs all the steps from the experiment, by calling these functions:

- 'add_pbd_output': from the parameters, create a gene tree (see chapter D.2)
- 'add_species_trees_with_outgroup': from the gene tree, create one or more species trees with an added outgroup (see chapter D.4)
- 'add_alignments': from each species tree (with outgroup), create one or more simulated DNA alignments (see chapter D.1)
- 'add_posteriors': for each DNA alignment, create one or more BEAST2 posterior files (see chapter D.3)

The 'do_simulation_examples.sh' bash script (see chapter B.16) shows how to call 'do_simulation' for the worked-out examples.

D R functions

Functions that are called.

The functions used are listed here alphabetically. The chronological order of these function calls is shown in figure 52:

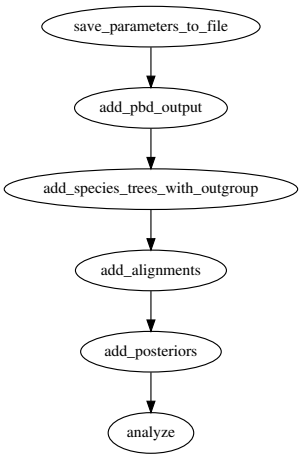

Figure 52: The chronological order of the functions called

D.1 add_alignments

Create a simulated DNA alignment from a species tree (with outgroup). See algorithm 27.

D.2 add_pbd_output

Creates a gene tree from a parameter file. See algorithm 28.

D.3 add_posteriors

For each DNA alignment, create one or more BEAST2 posterior files. See algorithm 29.

D.4 add_species_trees_with_outgroup

Creates one or more species trees from a gene tree. Also adds an outgroup to each species tree. See algorithm 30.

D.5 analyse_multi

R function that is called to creates the result graphs for a multiple parameter filenames. This is useful to do comparisons: the first filename is assumed to be a non-protracted parameter file. See algorithm 31.

It forwards the heavy lifting to other R functions.

Algorithm 27 The 'add_alignments' function

```
source("~/GitHubs/R/Peregrine/is_valid_file.R")
source("~/GitHubs/R/Peregrine/read_file.R")
source("~/GitHubs/R/Phylogenies/convert_phylogeny_to_alignment.R")
library(testit)

add_alignments <- function(filename)
{
  assert(is_valid_file(filename))
  file <- read_file(filename)
  if(is.na(file$species_trees_with_outgroup[1])) {
    print(paste("file ",filename," needs a species_trees_with_outgroup",
        sep=""))
    return ()
  }
  parameters <- file$parameters
  rng_seed <- as.numeric(parameters$rng_seed[2]) # the extinction rate
      of incipient species
  mutation_rate <- as.numeric(parameters$mutation_rate[2])
  n_alignments <- as.numeric(parameters$n_alignments[2])
  assert(n_alignments > 0)
  sequence_length <- as.numeric(parameters$sequence_length[2])
  n_species_trees_samples <- as.numeric(parameters$n_species_trees_
      samples[2])
  assert(length(file$alignments) == n_alignments * n_species_trees_
      samples)
  for (i in seq(1,n_species_trees_samples)) {
    species_tree <- file$species_trees_with_outgroup[[i]][[1]]
    if (length(species_tree) == 1 && is.na(species_tree)) {
      print(paste("species_trees_with_outgroup[[", i, "]] is NA.
          Terminating 'add_alignments'",sep=""))
      return
    }
    for (j in seq(1,n_alignments)) {
      index <- 1 + (j - 1) + ( (i - 1) * n_species_trees_samples)
      if (class(file$alignments[[index]][[1]]) == "DNAbin") {
        print(paste("    * Already stored alignment #", j, " for species
            tree #",i," at index #", index, sep=""))
        next
      }
      new_seed <- rng_seed - 1 + j + ((i -1) * n_species_trees_samples)
      set.seed(new_seed)
      alignment <- convert_phylogeny_to_alignment(
        phylogeny = species_tree,
        sequence_length = sequence_length,
        mutation_rate = mutation_rate
      )
      file$alignments[[index]] <- list(alignment)
      saveRDS(file,file=filename)
      print(paste("    * Created and saved alignments[",index,"]",sep="")
          )
    }
  }
  print(paste("file ",filename," has gotten its ", n_alignments, "
      alignments (per species tree)",sep=""))
}
```

Algorithm 28 The 'add_pbd_output' function

```
source("~/GitHubs/R/Peregrine/is_valid_file.R")
source("~/GitHubs/R/Peregrine/read_file.R")
source("~/GitHubs/R/Phylogenies/is_pbd_sim_output.R")
library(PBD)
library(testit)

add_pbd_output <- function(filename) {
  assert(is_valid_file(filename))
  file <- read_file(filename)
  if(is_pbd_sim_output(file$pbd_output)) {
    print(paste("file ",filename," already has a pbd_output",sep=""))
    return ()
  }
  parameters <- file$parameters
  rng_seed <- as.numeric(parameters$rng_seed[2])
  species_initiation_rate_good_species <- as.numeric(parameters$species
      _initiation_rate_good_species[2])
  species_initiation_rate_incipient_species <- as.numeric(parameters$
      species_initiation_rate_incipient_species[2])
  speciation_completion_rate <- as.numeric(parameters$speciation_
      completion_rate[2])
  extinction_rate_good_species <- as.numeric(parameters$extinction_rate_
      good_species[2])
  extinction_rate_incipient_species <- as.numeric(parameters$extinction_
      rate_incipient_species[2])
  age <- as.numeric(parameters$age[2])
  set.seed(rng_seed)
  #pbd_output <- pbd_sim(c(
  file$pbd_output <- pbd_sim(c(
    species_initiation_rate_good_species,
    speciation_completion_rate,
    species_initiation_rate_incipient_species,
    extinction_rate_good_species,
    extinction_rate_incipient_species
  ),age=as.numeric(parameters$age[2]),soc=2,plot=FALSE)
  #phylogeny <- pbd_output$tree
  #file$pbd_output <- pbd_output
  saveRDS(file,file=filename)
  print(paste("Added pbd_output to file ",filename,sep=""))
}
```

Algorithm 29 The 'add_posteriors' function

```
source("~/GitHubs/R/Peregrine/is_valid_file.R")
source("~/GitHubs/R/Peregrine/read_file.R")
source("~/GitHubs/R/Phylogenies/is_beast_posterior.R")
source("~/GitHubs/R/Phylogenies/convert_alignment_to_beast_posterior.R")
library(testit)
library(tools) #For file_path_sans_ext

add_posteriors <- function(filename)
{
  assert(is_valid_file(filename))
  file <- read_file(filename)
  parameters <- file$parameters
  rng_seed <- as.numeric(parameters$rng_seed[2])
  mcmc_chainlength <- as.numeric(parameters$mcmc_chainlength[2])
  n_alignments <- as.numeric(parameters$n_alignments[2])
  n_beast_runs <- as.numeric(parameters$n_beast_runs[2])
  n_species_trees_samples <- as.numeric(file$parameters$n_species_trees_
      samples[2])
  for (i in seq(1,n_species_trees_samples)) {
    for (j in seq(1,n_alignments)) {
      alignment_index <- 1 + (j - 1) + ((i - 1) * n_species_trees_
          samples)
      assert(alignment_index >= 1 && alignment_index <= length(file$
          alignments))
      alignment <- file$alignments[[alignment_index]][[1]]
      assert(is_alignment(alignment))
      for (k in seq(1,n_beast_runs)) {
        posterior_index <- 1 + (k - 1) + ((j - 1) * n_alignments) + ((i
            - 1) * n_alignments * n_species_trees_samples)
        assert(posterior_index >= 1 && posterior_index <= length(file$
            posteriors))
        if(is_beast_posterior(file$posteriors[[posterior_index]][[1]]))
            {
          print(paste("   * Posterior #", k, " for alignment #",j," for
              species tree #",i," at posterior_index #", posterior_
              index, " already has a posterior", sep=""))
          next
        }
        new_seed <- rng_seed + k
        print(paste("   * Setting seed to ", new_seed, sep=""))
        set.seed(new_seed)
        basefilename <- paste(basename(file_path_sans_ext(filename)),"_"
            ,i,"_",j,"_",k,sep="")
        posterior <- convert_alignment_to_beast_posterior(
          alignment = alignment,
          base_filename = basefilename,
          mcmc_chainlength = mcmc_chainlength,
          rng_seed = new_seed
        )
        print(paste("   * Storing posterior #", k, " for alignment #",j,
            " for species tree #",i," at posterior_index #", posterior_
            index, sep=""))
        file$posteriors[[posterior_index]] <- list(posterior)
      }
    }
  }
  saveRDS(file, file = filename)
  print(paste("file ", filename, " has gotten its posteriors", sep = "")
      )
}
```

Algorithm 30 The 'add_species_trees_with_outgroup' function

```
source("~/GitHubs/R/Peregrine/is_valid_file.R")
source("~/GitHubs/R/Peregrine/read_file.R")
source("~/GitHubs/R/Phylogenies/add_outgroup_to_phylogeny.R")
source("~/GitHubs/R/Phylogenies/sample_species_trees_from_pbd_sim_output
    .R")
library(testit)

add_species_trees_with_outgroup <- function(filename) {
  assert(is_valid_file(filename))
  file <- read_file(filename)
  if(is.na(file$pbd_output[1])) {
    print(paste("file ",filename," needs a pbd_output",sep=""))
    return ()
  }
  parameters <- file$parameters
  n_species_trees_samples <- as.numeric(parameters$n_species_trees_
      samples[2])
  rng_seed <- as.numeric(parameters$rng_seed[2])
  print(paste("Adding species_trees_with_outgroup to file ",filename,sep
      =""))

  for (i in seq(1:n_species_trees_samples)) {
    if (!is.na(file$species_trees_with_outgroup[i]))
    {
      print(paste(" * species_trees_with_outgroup[",i,"] already exists"
          ,sep=""))
      next
    }
    print(paste("   * Setting seed to ", (rng_seed + i), sep=""))
    set.seed(rng_seed + i) # Each species tree is generated from its own
        RNG seed
    species_tree <- sample_species_trees_from_pbd_sim_output(n = 1,file$
        pbd_output)[[1]]
    species_tree_with_outgroup <- add_outgroup_to_phylogeny(species_tree
        ,stem_length = 0)
    assert(class(species_tree_with_outgroup) == "phylo")
    file$species_trees_with_outgroup[[i]] <- list(species_tree_with_
        outgroup)
    saveRDS(file,file=filename)
  }
  print(paste("Added species_trees_with_outgroup to file ",filename,sep=
      ""))
}
```

Algorithm 31 The 'analyse_multi' function

```
source("~/GitHubs/R/Peregrine/plot_multi_average_nltts.R")
source("~/GitHubs/R/Peregrine/plot_multi_nltt_stats_histogram.R")

analyse_multi <- function(filenames) {
  plot_multi_average_nltts(filenames)
  plot_multi_nltt_stats_histogram(filenames)
}
```

Algorithm 32 The 'analyse_single' function

```
source("~/GitHubs/R/Peregrine/plot_gene_tree.R")
source("~/GitHubs/R/Peregrine/plot_species_tree_with_outgroup.R")
source("~/GitHubs/R/Peregrine/plot_species_tree_with_outgroup_nltt.R")
source("~/GitHubs/R/Peregrine/plot_alignments.R")
source("~/GitHubs/R/Peregrine/plot_posterior_average_nltts.R")
source("~/GitHubs/R/Peregrine/plot_posterior_samples.R")
source("~/GitHubs/R/Peregrine/plot_posterior_sample_nltts.R")
source("~/GitHubs/R/Peregrine/plot_posterior_nltt_stats_histogram.R")

analyse_single <- function(filename) {
  if (!file.exists(filename)) {
    print(paste(filename,": not found"))
    stop()
  }
  plot_gene_tree(filename)
  plot_species_tree_with_outgroup(filename)
  plot_species_tree_with_outgroup_nltt(filename)
  plot_alignments(filename)
  plot_posterior_average_nltts(filename)
  plot_posterior_samples(filename)
  plot_posterior_sample_nltts(filename)
  plot_posterior_nltt_stats_histogram(filename)
}
```

D.6 analyse_single

R function that is called to creates the result graphs for a single parameter filename. See algorithm 32.

It forwards the heavy lifting to other R functions.

D.7 is_valid_file

The function 'is_valid_file' checks if a parameter file is valid, as can be seen in algorithm 33.

It checks the parameter file its form and its parameter values.

D.8 plot_alignments

This R function plots the alignments of a parameter file.

Figure 7 is produced by this function.

D.9 plot_gene_tree

This R function plots the gene tree of a parameter file.

There is always exactly one gene tree per parameter file.

Figure 3 is produced by this function.

Algorithm 33 The 'is_valid_file' function

```
source("~/GitHubs/R/Peregrine/read_file.R")

is_valid_file <- function(filename) {
  if (!file.exists(filename)) return (FALSE)
  file <- read_file(filename)
  if (mode(file) != "list") return (FALSE)
  if (is.null(file$parameters)) return (FALSE)
  if (is.null(file$pbd_output)) return (FALSE)
  if (is.null(file$species_trees_with_outgroup)) return (FALSE)
  if (is.null(file$alignments)) return (FALSE)
  if (is.null(file$posteriors)) return (FALSE)
  parameters <- file$parameters
  if (as.numeric(parameters$species_initiation_rate_good_species[2]) <
      0.0) return (FALSE)
  if (as.numeric(parameters$species_initiation_rate_incipient_species
      [2]) < 0.0) return (FALSE)
  if (as.numeric(parameters$speciation_completion_rate[2]) < 0.0) return
       (FALSE)
  if (as.numeric(parameters$extinction_rate_good_species[2]) < 0.0)
      return (FALSE)
  if (as.numeric(parameters$extinction_rate_incipient_species[2]) < 0.0)
       return (FALSE)
  if (as.numeric(parameters$age[2]) <= 0.0) return (FALSE)
  if (as.numeric(parameters$n_species_trees_samples[2]) < 1) return (
      FALSE)
  if (as.numeric(parameters$mutation_rate[2]) <= 0.0) return (FALSE)
  if (as.numeric(parameters$n_alignments[2]) < 1) return (FALSE)
  if (as.numeric(parameters$sequence_length[2]) < 1) return (FALSE)
  if (as.numeric(parameters$n_beast_runs[2]) < 1) return (FALSE)
  if (as.numeric(parameters$mcmc_chainlength[2]) < 1) return (FALSE)
  return (TRUE)
}
```

Algorithm 34 The 'plot_alignments' function

```
library(testit)
source("~/GitHubs/R/Peregrine/is_valid_file.R")
source("~/GitHubs/R/FileIo/get_base_filename.R")
source("~/GitHubs/R/Peregrine/read_file.R")

plot_alignments <- function(filename) {
  assert(is_valid_file(filename))
  base_filename <- get_base_filename(filename)
  file <- read_file(filename)
  n_species_trees_samples <- as.numeric(file$parameters$n_species_trees_
      samples[2])
  n_alignments <- as.numeric(file$parameters$n_alignments[2])
  for (i in seq(1,n_species_trees_samples)) {
    for (j in seq(1,n_alignments)) {
      alignment_index <- 1 + j - 1 + ((i - 1) * n_species_trees_samples)
      assert(alignment_index >= 1)
      assert(alignment_index <= length(file$alignments))
      png(paste(base_filename,"_alignment_",i,"_",j,".png",sep=""))
      image(file$alignments[[alignment_index]][[1]], main=paste(base_
          filename,"alignment",i,j))
      dev.off()
    }
  }
}
```

Algorithm 35 The 'plot_gene_tree' function

```
library(ape)
library(testit)
source("~/GitHubs/R/Peregrine/is_valid_file.R")
source("~/GitHubs/R/FileIo/get_base_filename.R")
source("~/GitHubs/R/Peregrine/read_file.R")

plot_gene_tree <- function(filename) {
  assert(is_valid_file(filename))
  base_filename <- get_base_filename(filename)
  file <- read_file(filename)
  png(paste(base_filename,"_gene_tree.png",sep=""))
  plot(file$pbd_output[[1]], main = paste(base_filename," gene tree",sep
      =""))
  dev.off()
}
```

D.10 plot_multi_average_nltts

This R function plots the alignments of a parameter file.

Figure 25 is produced by this function.

D.11 plot_multi_nltt_stats_histogram

This R function plots the alignments of a parameter file.

Figure 24 is produced by this function.

D.12 plot_posterior_average_nltts

For each posterior, plots the average nLTT of all phylogenies, including the true tree its nLTT.

Figure 13 is produced by this function.

D.13 plot_posterior_nltt_stats_histogram

R function that creates a histogram out of the nLTT stats and puts it in a histogram.

Figure 12 is produced by this function.

D.14 plot_posterior_samples

R function that plots a single tree in the posterior.

Figure 9 is produced by this function.

D.15 plot_posterior_sample_nltts

R function that plots the nLTT plot of a single tree in the posterior.

Figure 10 is produced by this function.

D.16 plot_species_tree_with_outgroup

This R function that plots the species trees of a parameter file.

There are $n_{species\_trees}$ species trees per parameter file, each sampled randomly from the same species tree.

Figure 4 is produced by this function.

D.17 plot_species_tree_with_outgroup_nltt

This R function that plots the species trees their nLTT plots of a parameter file.

There are $n_{species\_trees}$ species trees per parameter file, each sampled randomly from the same species tree.

Figure 5 is produced by this function.

Algorithm 36 The 'plot_multi_average_nltts' function

```
source("~/GitHubs/R/FileIo/get_base_filename.R")
source("~/GitHubs/R/Peregrine/is_valid_file.R")
source("~/GitHubs/R/MyFavoritePackages/olli_rBEAST/R/fun.beast2output.R"
    )
source("~/GitHubs/R/Phylogenies/get_average_nltt.R")
library(testit)

plot_multi_average_nltts <- function(filenames) {
  png(paste("multi_average_nltts.png",sep=""))

  for (h in c(1,length(filenames))) {
    filename <- filenames[h]
    assert(is_valid_file(filename))
    file <- read_file(filename)

    n_species_trees_samples <- as.numeric(file$parameters$n_species_
        trees_samples[2])
    n_alignments <- as.numeric(file$parameters$n_alignments[2])
    n_beast_runs <- as.numeric(file$parameters$n_beast_runs[2])
    for (i in seq(1,n_species_trees_samples)) {
      for (j in seq(1,n_alignments)) {
        for (k in seq(1,n_beast_runs)) {
          base_filename <- get_base_filename(filename)
          trees_filename <- paste(base_filename,"_",i,"_",j,"_",k,".
              trees",sep="")
          assert(file.exists(trees_filename))
          all_trees <- beast2out.read.trees(trees_filename)
          linetype <- ifelse(h == 1, 1, 3)
          if (h == 1 && i == 1 && j == 1 && k == 1) {
            get_average_nltt(all_trees,replot = FALSE,lty=1,lwd = 2,
                main="Average nLTTs")
          } else {
            get_average_nltt(all_trees,replot = TRUE, lty = linetype,lwd
                = 2)
          }
        }
      }
    }
  }
  legend(0.7,0.3, # Top left
    c('BD','PBD'), # puts text in the legend
    lty=c(1,3), # gives the legend appropriate symbols (lines)
    lwd=c(2,2)
  )
  dev.off()
}
```

Algorithm 37 The 'plot_multi_nltt_stats_histogram' function

```
source("~/GitHubs/R/Peregrine/is_valid_file.R")
source("~/GitHubs/R/FileIo/get_base_filename.R")
source("~/GitHubs/R/Peregrine/load_parameters_from_file.R")
source("~/GitHubs/R/MyFavoritePackages/olli_rBEAST/R/fun.beast2output.R"
    )
library(ape)
library(ggplot2)
library(gridExtra)
library(nLTT)
library(testit)

plot_multi_nltt_stats_histogram <- function(filenames) {

  data <- data.frame()

  for (filename in filenames) {
    assert(is_valid_file(filename))
    file <- load_parameters_from_file(filename)
    n_species_trees_samples <- as.numeric(file$parameters$n_species_
        trees_samples[2])
    n_alignments <- as.numeric(file$parameters$n_alignments[2])
    n_beast_runs <- as.numeric(file$parameters$n_beast_runs[2])
    for (i in seq(1,n_species_trees_samples)) {
      for (j in seq(1,n_alignments)) {
        for (k in seq(1,n_beast_runs)) {
          base_filename <- get_base_filename(filename)
          trees_filename <- paste(base_filename,"_",i,"_",j,"_",k,".
              trees",sep="")
          all_trees <- beast2out.read.trees(trees_filename)
          all_nltt_stats <- NULL
          for (tree in all_trees) {
            all_nltt_stats <- c(all_nltt_stats,nLTTstat(file$species_
                trees_with_outgroup[[1]][[1]],tree))
          }
          this_data <- data.frame(length = all_nltt_stats)
          this_data$description <- get_base_filename(filename)
          data <- rbind(data,this_data)
        }
      }
    }
  }

  myplot <- ggplot(
    data, aes(length, fill = description)
  ) + geom_histogram(
    alpha = 0.25,
    aes(y = ..density..),
    position = 'identity',
    binwidth = 0.005
  ) + ggtitle("nLTT statistics distribution") +
    theme(plot.title = element_text(lineheight=.8, face="bold")) +
    scale_fill_manual(" ",values=c("red","blue"))
  grid.arrange(myplot)
  ggsave("multi_nltt_stats_histogram.png")
}
```

Algorithm 38 The 'plot_posterior_average_nltts' function

```
library(testit)
source("~/GitHubs/R/Peregrine/is_valid_file.R")
source("~/GitHubs/R/FileIo/get_base_filename.R")
source("~/GitHubs/R/Peregrine/read_file.R")
source("~/GitHubs/R/Phylogenies/get_average_nltt.R")
source("~/GitHubs/R/MyFavoritePackages/olli_rBEAST/R/fun.beast2output.R"
    )

plot_posterior_average_nltts <- function(filename) {
  assert(is_valid_file(filename))
  base_filename <- get_base_filename(filename)
  file <- read_file(filename)
  n_species_trees_samples <- as.numeric(file$parameters$n_species_trees_
      samples[2])
  n_alignments <- as.numeric(file$parameters$n_alignments[2])
  n_beast_runs <- as.numeric(file$parameters$n_beast_runs[2])
  for (i in seq(1,n_species_trees_samples)) {
    for (j in seq(1,n_alignments)) {
      for (k in seq(1,n_beast_runs)) {
        trees_filename <- paste(base_filename,"_",i,"_",j,"_",k,".trees"
            ,sep="")
        all_trees <- beast2out.read.trees(trees_filename)
        png(paste(base_filename,"_posterior_average_nltt_",i,"_",j,"_",k
            ,".png",sep=""))
        get_average_nltt(all_trees,replot = FALSE,lty=3,lwd = 2, main=
            paste(base_filename," posterior average nLTT"))
        get_average_nltt(
          list(file$species_trees_with_outgroup[[1]][[1]],
          file$species_trees_with_outgroup[[1]][[1]]),
          replot = TRUE,lty=1,lwd = 2)
        dev.off()
      }
    }
  }
}
```

Algorithm 39 The 'plot_nltt_stats_histogram' function

```
library(nLTT)
library(testit)
source("~/GitHubs/R/Peregrine/is_valid_file.R")
source("~/GitHubs/R/FileIo/get_base_filename.R")
source("~/GitHubs/R/Peregrine/read_file.R")
source("~/GitHubs/R/MyFavoritePackages/olli_rBEAST/R/fun.beast2output.R"
    )

plot_posterior_nltt_stats_histogram <- function(filename) {
  assert(is_valid_file(filename))
  base_filename <- get_base_filename(filename)
  file <- read_file(filename)
  n_species_trees_samples <- as.numeric(file$parameters$n_species_trees_
      samples[2])
  n_alignments <- as.numeric(file$parameters$n_alignments[2])
  n_beast_runs <- as.numeric(file$parameters$n_beast_runs[2])
  for (i in seq(1,n_species_trees_samples)) {
    for (j in seq(1,n_alignments)) {
      for (k in seq(1,n_beast_runs)) {
        trees_filename <- paste(base_filename,"_",i,"_",j,"_",k,".trees"
            ,sep="")
        all_trees <- beast2out.read.trees(trees_filename)
        last_tree <- tail(all_trees,n=1)[[1]]
        all_nltt_stats <- NULL
        for (tree in all_trees) {
          all_nltt_stats <- c(all_nltt_stats,nLTTstat(file$species_trees
              _with_outgroup[[1]][[1]],tree))
        }
        png(paste(base_filename,"_nltt_stats_",i,"_",j,"_",k,".png",sep=
            ""))
        hist(all_nltt_stats,xlim=c(0,0.12),main=paste(base_filename,"
            distribution of nLTT statistics\nbetween species tree and
            posterior",i,j,k))
        dev.off()
      }
    }
  }
}
```

Algorithm 40 The 'plot_posterior_samples' function

```
library(testit)
source("~/GitHubs/R/Peregrine/is_valid_file.R")
source("~/GitHubs/R/FileIo/get_base_filename.R")
source("~/GitHubs/R/Peregrine/read_file.R")
source("~/GitHubs/R/MyFavoritePackages/olli_rBEAST/R/fun.beast2output.R"
    )

plot_posterior_samples <- function(filename) {
  assert(is_valid_file(filename))
  base_filename <- get_base_filename(filename)
  file <- read_file(filename)
  n_species_trees_samples <- as.numeric(file$parameters$n_species_trees_
      samples[2])
  n_alignments <- as.numeric(file$parameters$n_alignments[2])
  n_beast_runs <- as.numeric(file$parameters$n_beast_runs[2])
  for (i in seq(1,n_species_trees_samples)) {
    for (j in seq(1,n_alignments)) {
      for (k in seq(1,n_beast_runs)) {
        trees_filename <- paste(base_filename,"_",i,"_",j,"_",k,".trees"
            ,sep="")
        all_trees <- beast2out.read.trees(trees_filename)
        last_tree <- tail(all_trees,n=1)[[1]]
        png(paste(base_filename,"_posterior_sample_",i,"_",j,"_",k,".png
            ",sep=""))
        plot(last_tree,main=paste(base_filename,"last tree in posterior"
            ,i,j,k))
        dev.off()
      }
    }
  }
}
```

Algorithm 41 The 'plot_posterior_sample_nltts' function

```
library(testit)
source("~/GitHubs/R/Peregrine/is_valid_file.R")
source("~/GitHubs/R/FileIo/get_base_filename.R")
source("~/GitHubs/R/Peregrine/read_file.R")
source("~/GitHubs/R/MyFavoritePackages/olli_rBEAST/R/fun.beast2output.R"
    )
plot_posterior_sample_nltts <- function(filename) {
  assert(is_valid_file(filename))
  base_filename <- get_base_filename(filename)
  file <- read_file(filename)
  n_species_trees_samples <- as.numeric(file$parameters$n_species_trees_
      samples[2])
  n_alignments <- as.numeric(file$parameters$n_alignments[2])
  n_beast_runs <- as.numeric(file$parameters$n_beast_runs[2])
  for (i in seq(1,n_species_trees_samples)) {
    for (j in seq(1,n_alignments)) {
      for (k in seq(1,n_beast_runs)) {
        trees_filename <- paste(base_filename,"_",i,"_",j,"_",k,".trees"
            ,sep="")
        all_trees <- beast2out.read.trees(trees_filename)
        last_tree <- tail(all_trees,n=1)[[1]]
        png(paste(base_filename,"_posterior_sample_nltt_",i,"_",j,"_",k,
            ".png",sep=""))
        nLTT.plot(file$species_trees_with_outgroup[[1]][[1]],
          main=paste(base_filename,"species tree with outgroup nLTTs",i,
              j,k), lwd = 2
        )
        nLTT.lines(last_tree, lwd = 2, lty = 3)
        dev.off()
      }
    }
  }
}
```

Algorithm 42 The 'plot_species_tree_with_outgroup' function

```
library(ape)
library(testit)
source("~/GitHubs/R/Peregrine/is_valid_file.R")
source("~/GitHubs/R/FileIo/get_base_filename.R")
source("~/GitHubs/R/Peregrine/read_file.R")

plot_species_tree_with_outgroup <- function(filename) {
  assert(is_valid_file(filename))
  base_filename <- get_base_filename(filename)
  file <- read_file(filename)
  n_species_trees_samples <- as.numeric(file$parameters$n_species_trees_
      samples[2])
  for (i in seq(1,n_species_trees_samples)) {
    png(paste(base_filename,"_species_tree_with_outgroup_",i,".png",sep=
        ""))
    plot(file$species_trees_with_outgroup[[i]][[1]], main = paste(base_
        filename,"species tree with outgroup",i))
    dev.off()
  }
}
```

Algorithm 43 The 'plot_species_tree_with_outgroup_nltt' function

```
library(nLTT)
library(testit)
source("~/GitHubs/R/Peregrine/is_valid_file.R")
source("~/GitHubs/R/FileIo/get_base_filename.R")
source("~/GitHubs/R/Peregrine/read_file.R")

plot_species_tree_with_outgroup_nltt <- function(filename) {
  assert(is_valid_file(filename))
  base_filename <- get_base_filename(filename)
  file <- read_file(filename)
  n_species_trees_samples <- as.numeric(file$parameters$n_species_trees_
      samples[2])
  for (i in seq(1,n_species_trees_samples)) {
    png(paste(base_filename,"_species_tree_with_outgroup_nltt_",i,".png"
        ,sep=""))
    nLTT.plot(file$species_trees_with_outgroup[[i]][[1]], main = paste(
        base_filename,"species tree with outgroup",i))
    dev.off()
  }
}
```

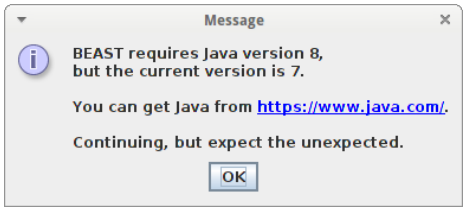

Figure 53: BEAST2 error

D.18 save_parameters_to_file

The function 'save_parameters_to_file' saves the parameters to file, as can be seen in algorithm 44.

At the core of this function is the creation of 'my_table', which stores all the parameters. After saving, it calls 'is_valid_file' to check the parameter file. Finally it checks if saving and loading of the created parameter file works as expected.

E Error

E.1 BEAST requires Java version 8

Sometimes my Java version degrades to version 7. BEAST2 will show figure 53:

On the command-line do:

```
sudo apt-get install oracle-java8-set-default
```

Algorithm 44 The 'save_parameters_to_file' function

```
library(testit)
source("~/GitHubs/R/Peregrine/is_valid_file.R")

save_parameters_to_file <- function(
  rng_seed,
  species_initiation_rate_good_species,
  species_initiation_rate_incipient_species,
  speciation_completion_rate,
  extinction_rate_good_species,
  extinction_rate_incipient_species,
  age, n_species_trees_samples,
  mutation_rate, n_alignments,
  sequence_length, mcmc_chainlength,
  n_beast_runs, filename
) {
  my_table <- data.frame( row.names = c("Description","Value"))
  my_table[, "rng_seed"] <- c("Random number generate seed",rng_seed)
  my_table[, "species_initiation_rate_good_species"] <- c("b_g",species_
      initiation_rate_good_species)
  my_table[, "species_initiation_rate_incipient_species"] <- c("b_i",
      species_initiation_rate_incipient_species)
  my_table[, "speciation_completion_rate"] <- c("lambda",speciation_
      completion_rate)
  my_table[, "extinction_rate_good_species"] <- c("mu_g",extinction_rate
      _good_species)
  my_table[, "extinction_rate_incipient_species"] <- c("mu_i",extinction
      _rate_incipient_species)
  my_table[, "age"] <- c("Phylogenetic tree crown age",age)
  my_table[, "n_species_trees_samples"] <- c("species trees sampled",n_
      species_trees_samples)
  my_table[, "mutation_rate"] <- c("DNA mutation rate",mutation_rate)
  my_table[, "n_alignments"] <- c("Number of DNA alignments per species
      tree",n_alignments)
  my_table[, "sequence_length"] <- c("DNA sequence length",sequence_
      length)
  my_table[, "mcmc_chainlength"] <- c("MCMC chain length",mcmc_
      chainlength)
  my_table[, "n_beast_runs"] <- c("Number of BEAST2 runs per alignment",
      n_beast_runs)
  my_table[, "version"] <- c("Parameter file version","0.1")
  # Create the slots for the results
  my_list <- list(
    my_table, #parameters
    NA, # pbd_output
    rep(x = NA, times = n_species_trees_samples), # species_trees_with_
        outgroup
    rep(x = NA, times = n_species_trees_samples * n_alignments), #
        alignments
    rep(x = NA, times = n_species_trees_samples * n_alignments * n_beast
        _runs) # posteriors
  )
  names(my_list) <- c("parameters", "pbd_output", "species_trees_with_
      outgroup", "alignments", "posteriors")
  assert(length(my_list$pbd_output) == 1)
  assert(length(my_list$species_trees_with_outgroup) == n_species_trees_
      samples)
  assert(length(my_list$alignments) == n_species_trees_samples * n_
      alignments)
  assert(length(my_list$posteriors) == n_species_trees_samples * n_
      alignments * n_beast_runs)
  saveRDS(my_list,file=filename)
  assert(is_valid_file(filename))
}
```