# AsymmeTree User Manual

David Schaller

sdavid@bioinf.uni-leipzig.de

# Contents

# 1 Introduction

AsymmeTree is an open-source Python library for the simulation and analysis of phylogenetic scenarios. It includes a simulator for species and gene trees with asymmetric evolution rates, tools for the inference and analysis of phylogenetic best matches [? ? ] (resp. best hits) from known gene trees or evolutionary distances. Moreover, it includes tools for the analysis of horizontal gene transfer (HGT) events, an algorithm to compute supertrees [? ] and a method to estimate rooted species trees from an ensemble of orthology/paralogy relations [? ].

The library, and especially the simulator, is primarily designed to explore and validate mathematical concepts, and to test inference methods for various steps on the way to more realistically available data, i.e., dated gene trees, additive distances of gene sets, noisy distances and finally sequences. Both nucleotide and amino acid sequence simulation with or without indels are supported. In both cases, several substitution models are available.

The software is hosted on GitHub and also available via The Python Package Index (PyPI). Please feel free to report bugs or make suggestions for improvement in the Issues section of the GitHub repository.

If you use AsymmeTree in your project or code from it, please cite:

> Peter F. Stadler, Manuela Geiß, David Schaller, Alitzel López Sánchez, Marcos González Laffitte, Dulce I. Valdivia, Marc Hellmuth, Maribel Hernández Rosales (2020). **From pairs of most similar sequences to phylogenetic best matches.** *Algorithms for Molecular Biology.* doi: 10.1186/s13015-020-00165-2. [? ].

# 2 Installation

AsymmeTree requires Python 3.5 or higher. Python 2 is not supported.

## 2.1 Easy Installation with pip

The `asymmetree` package is available on The Python Package Index (PyPI):

```
pip install asymmetree
```

For details about how to install Python packages see here.

## 2.2 Installation with the setup file

Alternatively, you can download or clone the repo, go to the root folder of package and install it using the command:

```
python setup.py install
```

## 2.3 Dependencies

AssymmeTree has several dependencies (which are installed automatically when using `pip` or the `setup.py`):

- NetworkX
- SciPy and NumPy
- Matplotlib

The simulation of phylogenetic scenarios and sequences, as well as most functions for their analysis, do not have any other dependencies.

However, to use the tree reconstruction method for best match inference and the C++ implementation of the quartet method [? ], the software RapidNJ [? ], resp., qinfer must be installed. I recommend that you compile these tools on your machine, place the binaries into a persistent location and add this location to your PATH environment variable.

# 3 Usage

AsymmeTree is divided into several subpackages and modules, an overview of which is given in Appendix A. The library interface functions are described in the following sections and can be imported directly from the respective subpackage (see examples).

The term 'color' regularly appears in the library and refers to the reconciliation of gene trees with species trees. In particular, the terms 'color' and 'species'/'genome' in which a gene resides are used interchangeably. The reason for this is that the information in which species/genomes the genes reside is usually modeled as a (vertex) coloring, such as e.g. in (colored) best match graphs [? ].

## 3.1 Tree Data Structures

The two classes `Tree` and `PhyloTree` (inherits from `Tree`) implement tree data structures which are essential for most of the modules in the package. The latter contains converters and parsers for the Newick format and a NetworkX graph format.

The vertices of a `PhyloTree` instance are of type `PhyloTreeNode` and contain the following attributes:

`ID` vertex ID (`int`)

`label` label (`str`), in gene trees: `'S'` for speciation, `'D'` for duplication, `'H'` for horizontal gene transfer, `'*'` for loss

`color` only gene trees; species in which the gene resides, i.e., ID of some vertex in a species tree, `int` for extant genes, can be of type `tuple` (of two `int`s) for inner and loss vertices

`tstamp` time stamp of the event (`double`)

`dist` evolutionary distance from the parent vertex (`double`); if no evolution rates (see below) were simulated yet, then this value corresponds to the divergence time between the vertex and its parent

`transferred` only gene trees; indicates whether the edge from the parent is the transfer edge from an HGT event; `1` if yes and `0` otherwise

Both species and gene trees can be converted into Newick format using the function `to_newick()` of the `PhyloTree` class. In case of a gene tree, the color is represented in brackets, e.g.

To suppress this, use `to_newick(color=False)`. Likewise, to suppress the distances, you can use `to_newick(distance=False)`. The function `PhyloTree.parse_newick()` can handle this customized format as well as the standard Newick format.

Moreover, phylogenetic trees can easily be serialized in `json` or `pickle` (Python's serialization library) format:

By default, the serialization format is inferred from the file extension. Alternatively, it can be specified as keyword argument, e.g. `mode='json'`. To load a tree that was serialized this way, use:

An overview over selected functions of the tree classes is given in Appendix B.

The class `LCA` can be initiated with an instance of type `Tree` or its inheriting classes and then provides functions for efficient last common ancestor queries in that tree.

All such queries take instance of type `TreeNode` (or inheriting classes) as input or `ints`, and raise a `KeyError` if this node or ID, resp., is not in the tree.

## 3.2 Simulation of Species and Gene Trees

The subpackage `asymmetree.treeevolve` contains modules for the simulation of dated species and gene trees. In terms of divergence time, these trees define an ultrametric on the set of their (extant) leaves. Gene trees, furthermore, can be manipulated with a realistic rate heterogeneity among their branches resulting in general additive distances (but no longer ultrametric).

### 3.2.1 Species Trees

The function `simulate_species_tree(N)` simulates a dated species tree with `N` leaves (i.e. recent species) using the specified model. The following models are available:

`'innovation'` Innovation model ? ], if not specified the divergence time between the (planted) root and the leaves will be normalized to unity

`'yule'` standard Yule model [? ], default birth rate is 1.0

`'BDP'` constant-rate birth-death process [see e.g. ? ? ], default birth rate is 1.0 and death rate is 0.0

`'EBDP'` episodic birth-death process, algorithm of [? ]

The following keyword parameters (with their default value) are available:

`model='innovation'` model for the species tree simulation, currently only the 'innovation model' is available

`non_binary_prop=0.0` probability that an inner edge is contracted, results in a non-binary tree

`planted=True` add a planted root that has the first true speciation node as its single neighbor, this way duplication (and loss) events can occur before the first speciation event in a subsequent gene tree simulation

`remove_extinct= False` remove all branches leading to losses, only relevant for models with death events

`rescale_to_height= None` specify the divergence time between the (planted) root and the leaves i.e. the final height of the dated tree

For any model, the root of the resulting tree has the maximal time stamp and all (extant) species have time stamp 0.0. The episodes of the `'EBDP'` model must be supplied as a list of tuples/lists where each episode has the structure

Note that the first elements in this list correspond to the most recent ones, and that the first episode should have a time stamp of 0.0. Example usage:

### 3.2.2 Gene Trees

Dated gene trees are simulated along a given species tree `S` using a birth-death process [? ? ] with speciation events as additional branching events (fixed time points given by the species tree). At each time point, the total event rate is given by the sum of the event rates over all branches that are currently active (not extinct). Thus, the total event rate in general increases during the simulation if the loss rate does not dominate the rates of the branching events. To simulate gene tree, use the class `GeneTreeSimulator` or the function `simulate_dated_gene_tree(S, **kwargs)` with a species tree of type `PhyloTree`. The following parameters are available:

`dupl_rate=0.0` duplication rate (`float`)

`loss_rate=0.0` loss rate (`float`)

`hgt_rate=0.0` horizontal gene transfer rate (`float`)

`dupl_polytomy=0.0` allows non-binary duplication events by specifying the parameter $\lambda$ of a Poisson distribution (copy number = 2 + drawn number)

`prohibit_extinction= 'per_species'` avoid loss events for genes that are the last survivor in their species branch (`'per_species'`), the last survivor of the whole family (`'per_family'`); or no constraints (`False`)

For the constraints to avoid extinction, the loss rate in the respective branches are temporarily set to zero. Example usage:

The function `observable_tree(tree)` returns the observable part of a gene tree, i.e., it copies the tree, removes all branches that lead to loss events only and suppresses all inner nodes with only one child. It also removes the planted root. Example usage:

### 3.2.3 Assignment of Variable Evolution Rates

The module `EvolutionRates` contains functions to model realistic (asymmetric) evolution rates for a given gene tree. Moreover, correlation of the evolution rates between genes of the same (and closely related) species is introduced (autocorrelation, [**?** ]). The function `assign_rates(T, S)` takes a gene tree `T` and the **corresponding** species tree `S` as input, and manipulates the branch length of the gene tree. The following keyword parameters (with their default values) are available:

`base_rate=1.0` starting value for the substitution rate (per time unit) and expected value for conserved genes

`autocorr_factors=None` a dictionary containing rate factors for the edges of the species tree

`autocorr_variance=0.0` variance factor for a lognormal distribution that controls autocorrelation between genes of the same (and closely related) species, the higher the lower the autocorrelation; only relevant if `autocorr_factors` is not directly supplied

`rate_increase=('gamma', 0.5, 2.2)` distribution of the (relative) rate increase (w.r.t. the base rate) for divergent genes, i.e. to a factor 1 + x, the parameters the for default Gamma distribution are chosen to fit observed asymmetries between paralogs in yeast data [**?** ]

`CSN_weights=(1, 1, 1)` weights for choice between conservation, subfunctionalization and neofunctionalization after a duplication event

`inplace=True` manipulate edge lengths (`dist`) of the gene tree in-place, otherwise copy the tree

It is recommended to apply the rate assignment to the true gene tree that still contains loss events. Note that the rates are used to manipulate the `dist` attributes in the gene tree and not returned explicitly. Example usage:

The function `simulate_gene_trees(S)` combines the simulation of dated gene trees and the rate assignment into one step. If `N=1`, a single gene tree is returned. Otherwise, a list of gene trees is returned that shared the same rate factors for the branches in the species tree (autocorrelation factors) in the rate assignment procedure. Moreover, distribution for the base rate (assigned the planted edge of the gene tree) and for the event rates can be specified with the parameters `base_rate`, `dupl_rate`, `loss_rate` and `hgt_rate`. For available distributions and their syntax see Appendix C.

### 3.2.4 Distance Matrix and Noise

Distances derived from (real-life) gene or protein sequences are always burdened with noise. Such data can either be modeled by simulating sequences, or by disturbing the distances specified by a given tree directly. The latter alternatively is described briefly in this section.

The additive (i.e. noiseless) distance from an **observable** gene tree can be computed using the function `distance_matrix()` of a `PhyloTree` instance. It returns a tuple containing a list of leaves in the tree (corresponding to the row/column order) and the distance matrix as a 2-dimensional `numpy` array.

In the next step, noise can be introduced into a distance matrix using the `NoisyMatrix` module. Random noise can be simulated with the function `noisy_matrix(orig_matrix, sd)`. The following parameters are available (keyword arguments are indicated by their default value):

| | |
|---|---|
| `orig_matrix` | original matrix to be disturbed |
| `sd` | standard deviation of a normal distribution with mean 1 from which noise factors are drawn |
| `metric_repair='reject'` | method to ensure that the resulting distance matrix is still a metric, available are the rejection of noise steps that violate the metric property (`'reject'`), the decrease-only metric repair (`'DOMR'`) and the general metric repair (`'general'`) algorithm |

Alternatively, the function `convex_linear_comb(D1, D2)` can be used to simulate systematically biased noise by computing a linear convex combination with a disturbance matrix. The function thus takes two distance matrices (`numpy` arrays) not necessarily of the same size as input and disturbs them with one another. The contribution of the respective disturbance matrix is controlled by the keyword parameter `alpha` (default is `0.5`). If the keyword parameter `first_only` is `True`, only the first disturbed matrix is returned. Otherwise, both are returned in a tuple.

## 3.3 Simulation of Sequences

AsymmeTree supports the simulation of nucleic and amino acid sequences using time-continuous Markov models, as usually applied for this purpose [for textbooks, see e.g. **? ? ?** ]. The subpackage `asymmetree.seqevolve` contains the modules and functions for this task.

The class `Evolver` takes several model as parameters for its initialization:

- a **subtitution** model (`SubstModel`, required) for the substitution of single bases or amino acids,
- an **indel** model (`IndelModel`, optional) for the simulation of insertions and/or deletion, and
- a model for rate **heterogeneity** (`HetModel`, optional) among the sites of the sequence under evolution

### 3.3.1 Substitution Model

A substitution model usually comprises an exchangeability matrix $S$ and a vector $\pi$ containing the equilibrium frequencies of the alphabet $A$ of nucleobases, amino acids et cetera. From this, the rate matrix $Q$ can be computed as $S\Pi$ where $\Pi = \mathrm{diag}\{\pi_1, \ldots, \pi_{|A|}\}$ [? ]. The substitution probability matrix, in turn, is given by

$$P = e^{Qt}$$

which is computed efficiently by AsymmeTree using matrix diagonalization.

The following models for nucleotide (`n`) and amino acid (`a`) substitution are currently available (codon models are not supported at the moment):

| model | type | reference | required parameters (`kwargs`) |
|---|---|---|---|
| `JC96` | `n/a` | Jukes & Cantor 1969 [? ] | - |
| `K80` | `n` | Kimura 1980 [? ] | `kappa` (transition/transversion rate ratio) |
| `GTR` | `n` | general time-reversable model (GTR) 1986 [? ] | `abcdef` (list of rates (a) $C \leftrightarrow T$, (b) $A \leftrightarrow T$, (c) $G \leftrightarrow T$, (d) $A \leftrightarrow C$, (e) $C \leftrightarrow G$, (f) $A \leftrightarrow G$);<br>`f` (list of equilibrium frequencies $A/C/G/T$) |
| `DAYHOFF` | `a` | Dayhoff 1978 [? ] | - |
| `BLOSUM62` | `a` | BLOSUM62 1992 [? ] | - |
| `JTT` | `a` | Jones, Taylor & Thornton 1992 [? ] | - |
| `WAG` | `a` | Whelan & Goldman 2001 [? ] | - |
| `LG` | `a` | Le & Gascuel 2008 [? ] | - |
| `CUSTOM` | `n/a` | - | `filename` (path to a file with a model in PAML [? ] format) |

Note that a custom substitution model can be specified via `model_name='CUSTOM'`. In this case, the path to the model in PAML [? ] format must be supplied. Moreover, the model type (`n/a`) must fit this model. Example usage:

### 3.3.2 Indel Model

Insertions and deletions are modeled based on `Dawg` [? ]. An indel model requires sitewise rates for insertion `insertion_rate` and deletion `deletion_rate`. Moreover, the following parameters are available (with default values):

| | |
|---|---|
| `length_distr=('zipf', 1.821)` | distribution of indel length, default value for zipf distribution cf. [? ] |

`min_length=1` integer min. value at which the specified distribution is truncated, must be less than the expected value of the distribution, `None` means no limit

`max_length=None` integer max. value at which the specified distribution is truncated, must be greater than the expected value of the distribution, `None` means no limit

For available length distributions and their syntax see Appx. C. A zipf or negative binomial distribution are typically used for this purpose [? ? ]. Example usage:

### 3.3.3 Heterogeneity Model

Selective pressure usually varies among the sites of a sequence under evolution. To model this, rate factors $r$ for single sites or groups of sites are commonly drawn from a Gamma distribution ('+Γ') with mean 1 and parameter $\alpha$ [? ? ? ]. The rate matrix $rQ$ is then used instead of $Q$. Note that smaller values for $\alpha$ correspond to higher heterogeneity.

AsymmeTree supports two modes of the '+Γ'-model. You can specify a number of classes to which the sites are assigned randomly and uniformly distributed. Sites of the same class share a common factor $r$. The other possibility is a sitewise heterogeneity, i.e., every site has its own rate. In both cases, the rate or class membership is inherited from the parent sites during the evolution along a tree. Note that the sitewise mode is expected to have a longer running time.

An other aspect of among site heterogeneity is the modeling of invariant sites ('+$I$'), i.e., sites that never mutate at all as a result of very strong selective pressure. The (expected) proportion $p$ of invariant sites can be specified by the user, and sites are assigned as `'invariant'` with probability $p$. Note that $p > 0$ affects the overall substitution rate. In other words, the rates of the non-invariant sites are **not** adjusted to compensate the decreased number of expected substitution over all sites.

To summarize, the following parameters are available for the class `HetModel` (keyword arguments are indicated by their default value):

`alpha` parameter $\alpha$ of the Gamma distribution

`classes=5` number of classes; sites in the same class share the same rate factor

`sitewise=False` if `True`, factors are drawn sitewise; the number of classes is ignored in this case

`invariant=0.0` (expected) proportion $p$ of invariant sites

Note that the '+$I$'-model can be used without the '+Γ'-model by setting `classes=1` (the single class will have a factor of 1) and `invariant` to some proportion greater than 0. Example usage:

### 3.3.4 The Class `Evolver`

The class `Evolver` evolves a sequence according to the specified models (see previous sections) along a phylogenetic tree. In AsymmeTree, the **`dist`** attribute of the vertex $v$ in an edge $uv$ of the tree ($u$ is closer to the root) is always used as the **expected number of substitutions** along this edge. Thus, PAM distances as e.g. used optionally in [? ] are not supported. The `dist` attribute is also used as the duration of the Markov process in which insertions and deletions are drawn.

The following parameters are available for the initialization of an `Evolver` instance (keyword arguments are indicated by their default value):

| | |
|---|---|
| `subst_model` | substitution model; instance of `SubstModel` |
| `indel_model=None` | model for insertions and deletions; instance of `IndelModel` |
| `het_model=None` | model for among site heterogeneity and invariant sites; instance of `HetModel` |
| `jump_chain=False` | if `True`, an alternative Gillespie-like [? ] algorithm is applied for the substitution process instead of the computation of $P = e^{Qt}$ |

Once the `Evolver` is initialized, its function `evolve_along_tree()` can be called to evolve a sequence along a tree. The following parameters are available for this function (keyword arguments are indicated by their default value):

| | |
|---|---|
| `tree` | phylogenetic tree; instance of `PhyloTree` |
| `start_length=200` | length of the root sequence which is randomly drawn from the equilibrium frequencies in the specified substitution model |
| `start_seq=None` | root sequence (`str`); must be compatible with the specified substitution model (`model_type='n'/'a'`); if supplied, the `start_length` attribute is ignored |

The sequences of a simulation run are returned by this function (and also available via the attribute `sequences` as long as the function has not been called again), which is a `dict` containing the nodes (inner and leaf nodes) as keys and instances of type `EvoSeq` as values. The latter can be converted into `str` using `subst_model.to_sequence(evo_seq)`.

The function `true_alignment()` can be used to compute (and optionally write into a file) the 'true' alignment of a simulation run. The following keyword parameters are available:

| | |
|---|---|
| `include_inner=True` | if `True`, include also inner nodes in the alignment; otherwise only sequences of leaf nodes are aligned |
| `write_to=None` | path and filename for the output |
| `alignment_format='phylip'` | format of the alignment file; available are `'phylip'`, `'clustal'` and `'pretty'` |

Example usage of the class `Evolver`:

### 3.4 Simulation of Genomes

The class `GenomeSimulator` combines multiple steps described in the previous section in order to conveniently simulate whole genomes/proteomes. An instance of this class is initialized with a species tree (of type `PhyloTree`), and optionally the path to an output directory (`outdir=...`) if the user wants to save the results. The gene trees and the sequences are simulated in subsequent steps using the classes' function

(i) `simulate_gene_trees(N, **kwargs)`, and

(ii) `simulate_sequences(subst_model, **kwargs)`.

The first step (i) takes the same keyword parameters as input as the function `simulate_gene_trees()` in Section 3.2.3, where `N` is the number of gene families to be simulated. Thus, rates for the three event types (`dupl_rate, loss_rate, hgt_rate`), autocorrelation (`autocorr_variance`), the distribution of base rates (`base_rate_distr`) etc. can be specified.

The second step (ii) simulates the sequences along the observable part (without loss branches) of the simulated gene trees. The function takes the following parameters as input (keyword arguments are indicated by their default value):

`subst_model` substitution model; instance of `SubstModel`

`indel_model=None` model for insertions and deletions; instance of `IndelModel`

`het_model=None` model for among site heterogeneity and invariant sites; instance of `HetModel`

`root_genome=None` list of sequences for the roots of the gene trees; must contain the same number of `str` sequences as trees were simulated in step (i) i.e. `N`; sequences must be compatible with the specified substitution model (`model_type='n'/'a'`)

`length_distr=('constant', 200)` distribution of the length of the root sequences if `root_genome` is not supplied; see Appx. C

`min_length=10` minimal length at which the distribution of lengths is truncated; must be less than the mean of this distribution

`max_length=None` maximal length at which the distribution of lengths is truncated; must be greater than the mean of this distribution

`write_fastas=True` if `True` and an output directory was specified, write the **sequences** (one file **per species**) into the directory `fasta_files` in the output directory

`write_alignments=True` if `True` and an output directory was specified, write the **true alignments** (one file **per gene tree**) into the directory `alignments` in the output directory

After step (i), the `lists` of full and observable gene trees are accessible via the attributes `true_gene_trees` and `observable_gene_trees`, respectively. Moreover, the full gene trees are serialized into the directory `true_gene_trees` if an output directory was specified. After step (ii), the `lists` of sequence `dictionaries` are accessible via the attribute `sequence_dicts`.

Example usage:

## 3.5 Best Match Inference

Phylogenetic best matches of a gene $x$ of species $X$ are defined as those genes $y$ of another species $Y \neq X$ that share the lowest common ancestor with $x$ in the gene tree among all genes in that species [? ? ? ]. In contrast, two genes are orthologs if their last common ancestor was a speciation event. Orthology and reciprocal best matches are closely related [? ].

The subpackage `asymmetree.best_matches` contains functions to compute both relations from a given gene tree or to estimate them from distance data on a set of genes [? ]. If the true (observable) gene tree is known (as e.g. the case in simulations), best matches and orthologs can be computed using the module `TrueBMG`. The functions `bmg_from_tree()` and `orthology_from_tree()` return the respective graph representation as NetworkX (Di)Graphs:

If only distance data is available, best matches have to be estimated. AsymmeTree currently implements three different methods that are described by ? ]:

- Extended Best Hits (module `ExtBestHits`)
- Neighborjoining [? ] and midpoint rooting (module `TreeReconstruction`, requires the installation and accessibility of `RapidNJ` [? ])
- Quartet method (module `Quartets`, Python implementation and wrapper for the C++ tool `qinfer`)

Please see the file `examples/best_match_infer.py` in the GitHub repo for an example usage of these modules following the simulation of gene tree scenarios.

## 3.6 Analysis of Horizontal Gene Transfer

The subpackage `hgt` contains several functions for the analysis of horizontal gene transfer events in the simulated scenarios. In particular, transfer edges, the directed and undirected Fitch graph can be extracted, as well as the pairs of genes that diverged later than the respective species in which they reside, i.e. the so-called later-divergence-time (LDT) graph. The latter situation is indicative for the presence of HGT events in the scenario.

An edge $(u, v)$ in a gene tree is a *"true"* transfer edge if an HGT event happened on the path from $u$ to $v$. In the simulated trees, this is indicated by the attribute `transferred` of `PhyloTreeNode` $v$ which is set to `1`. An edge $(u, v)$ in the gene tree is an *rs*-transfer edge (named after the *relaxed scenario* concept) if the `colors` of $u$ and $v$ are incomparable in the corresponding species tree `S`. True and rs-transfer edges may not be equivalent, e.g. when a transfer from branch $A$ to $B$ is followed by a transfer from $B$ to $A$ and this gene lineage does not survive in branch $B$.

The directed Fitch graph of a tree $T$ has as vertex set the leaves of $T$ and a directed edge $(x, y)$ when the path from the last common ancestor of $x$ and $y$ to the leaf $y$ contains a transfer edge [**?** ]. The undirected Fitch graph of a tree $T$ also has as vertex set the leaves of $T$ and an undirected edge $xy$ when the path from $x$ to $y$ contains a transfer edge [**?** ].

As mentioned above, the situation in which two genes diverged later that their corresponding species witnesses HGT events. The graph that contains edges for any such gene pair has been termed the later-divergence-time (LDT) graph.

In order to reduce runtime, precomputed instances of the class `LCA` for `T` and `S` can be supplied using the keyword parameters `lca_T` and `lca_S`, respectively. Otherwise, such instances are initiated within the function.

### 3.7 Supertree Computation

The module `BuildST` contains an implementation of the BuildST algorithm described by **?** ] to compute a supertree from a given list of tree based on the leaf labels. The algorithm uses the dynamic graph data structure described by **?** ] and **?** ]. The latter can also be used separately:

The class `BuildST` is initialized with a list of trees that are of type `Tree` (thus also `PhyloTree` is allowed). The method `run()` then returns a supertree if the trees in the list are compatible *and* they overlap in their sets of leaf labels. More precisely, the graph on the set of input trees, in which two trees are connected by an edge if and only if they have at least one leaf label in common, must be connected. Example usage:

### 3.8 Cograph Editing and ParaPhylo

The subpackages `asymmetree.cograph` and `asymmetree.proteinortho` contain heuristics for cograph editing and a method to compute rooted species tree from orthology/paralogy relations. The latter is a reimplementation of ParaPhylo [**?** ] which uses heuristics for the NP-hard steps instead of exact ILP solutions. For cograph editing, the $\mathcal{O}(n^2)$ algorithm (where $n$ is the number of vertices in a connected graph) by **?** ] is applied. For the Maximum Consistent Triple Set problem, tree different heuristics are available:

| | |
|---|---|
| `BPMF` | Best-Pair-Merge-First [**?** ] (modified for weighted triples) |
| `MINCUT` | Aho's `BUILD` with weighted MinCut [**? ?** ] |
| `GREEDY` | a greedy approch based on Aho's `BUILD` |

The class `TreeReconstructor` in the module `SpeciesTreeFromParalogs` computes a species tree after it is provides with one or more NetworkX graphs that represent (estimated) orthology relations. To this end, the nodes in these graph must have a `'color'` attribute, since these will be the leaf labels in the reconstructed species tree. Example usage:

The module `SpeciesTreeFromProteinOrtho` contains functions to estimate a species tree from a `ProteinOrtho` [? ] output file. For example, the function `reconstruct_from_proteinortho(filename, triple_mode='BPMF')` takes the filename to the output file and optionally the triple heuristic as input, and returns a tuple consisting of the estimated species tree (`PhyloTree`) and a Newick representation (`str`) containing support values for the inner nodes.

# A Subpackages and Modules

| **Packages and Modules** | **Description** |
|---|---|
| **`datastructures`** | |
| `Tree` | Includes the basic class `Tree`, provides functions for tree traversals, Newick parser, etc. The class `LCA` provides efficient last common ancestor queries. |
| `PhyloTree` | Includes the class `PhyloTree` for phylogenetic trees (inherits from `Tree`), provides a Newick parser, etc. |
| `LinkedList` | Implementation of a linked list. |
| `DoublyLinkedList` | Implementation of a doubly-linked list. |
| `AVLTree` | Implementation of an ordered set (`TreeSet`) and an ordered dictionary (`TreeDict`) as a balanced binary search tree (AVL tree). |
| `Partition` | Dynamic partition that supports efficient merge operations. |
| `hdtgraph.DynamicGraph` | Dynamic graph data structure described by **?** ]. |
| **`treeevolve`** | |
| `SpeciesTree` | Simulator for dated species trees with different models. |
| `GeneTree` | Simulator for dated gene trees, construction of the observable gene tree. |
| `EvolutionRates` | Simulation of evolution rate asymmetries, autocorrelation between ancestors and descendants as well as correlation between genes in the same species. |
| `Scenario` | Wrapper class for species and gene tree scenarios, computation of the (R)BMG as well as event counts and some statistics. |
| `NoisyMatrix` | Generation of a noisy matrix (random perturbation or wrong topology noise). |
| **`seqevolve`** | |
| `Evolver` | Includes the class `Evolver` for the simulation of sequences along a phylogenetic tree. |
| `EvolvingSequence` | Data structure for sequences that are under evolution based on a doubly-linked list. |
| `SubstModel` | Substitution model for the sequence simulation. |
| `IndelModel` | Insertion/deletion (indel) model. |
| `HetModel` | Model for rate heterogeneity among the sites of an evolving sequence and for invariant sites. |
| `Alignment` | Construction of the true multiple sequence alignment after the simulation of sequences along a tree. |
| `EmpiricalModels` | Contains rate matrices and equilibrium frequencies for the empirical substitution models, taken from [**?** ]. |
| **`genome`** | |
| `GenomeSimulator` | Includes the class `GenomeSimulator` for the simulation of multiple gene families along a common species tree. |
| **`tools`** | |
| `Build` | Includes the classes `Build` and `Build2` for triple consistency tests and tree construction [**? ?** ]. |
| `BuildST` | Includes the class `BuildST` that computes a supertree from a given list of tress (with overlapping labels) [**?** ]. |

*Table continued on next page*

*Table continued from previous page*

| | |
|---|---|
| `GraphTools` | Miscellaneous functions for graphs, e.g. check for graph equality. |
| `DistanceCalculation` | Calculation of maximum likelihood distances of pairs of aligned sequences. |
| `Partitioning` | Implementation of (bi)partitioning heuristics such as Karger's algorithm. |
| `Sampling` | Includes the class `Sampler` which support drawing numbers from various distributions. |
| **`best_matches`** | |
| `TrueBMG` | Computation of the true (R)BMG from a gene tree as well as the true orthology relation. |
| `ExtBestHits` | Implementation of the *Extended Best Hits* method, optionally uses `qinfer`. |
| `TreeReconstruction` | Reconstruction of the gene tree with `RapidNJ` [? ] and midpoint rooting. |
| `Quartets` | Implementation of *Quartet* approach with two different methods for outgroup selection, optionally uses `qinfer`. |
| `LeastResolvedTree` | Construction of a least resolved tree (LRT) from a BMG via *informative triples* (optionally uses minimal edge cuts) or from a leaf-colored tree. |
| `Augmentation` | Augmentation of the least resolved tree (w.r.t. some BMG) in order to identify all unambiguously false orthology assignments [? ]. |
| **`cograph`** | |
| `Cograph` | Includes the classes `Cotree` and `CotreeNode` as well as a generator for random cotrees/cographs. The class `LinearCographDetector` implements an $\mathcal{O}(\|V\|+\|E\|)$ algorithm for cograph detection and cotree construction [? ]. |
| `CographEditor` | Implements a heuristic for cograph editing [? ]. |
| **`hgt`** | |
| `Fitch` | Extraction of transfer edges from a gene tree (together with a species tree). Construction of the directed and undirected Fitch graph. |
| `TimeComparison` | Comparison of the divergence time of genes with the divergence time of their respective species. |
| **`paraphylo`** | |
| `SpeciesTreeFromParalogs` | Species tree reconstruction from orthology/paralogy relations. Heuristic version of `ParaPhylo` [? ]. |
| `SpeciesTreeFrom`<br>`ProteinOrtho` | Species tree reconstruction from `ProteinOrtho` [? ? ] output. |
| **`visualize`** | |
| `GeneTreeVis` | Visualization of simulated gene trees (of type `PhyloTree`), experimental. |

# B Tree functions

The following table contains an overview over selected functions of the classes that inherit for `Tree`.

| |
|---|
| **`Tree`** (corresponding node class: `TreeNode`) |

*Table continued on next page*

*Table continued from previous page*

| | |
|---|---|
| `leaves()` | Generator for the leaf nodes. |
| `preorder()` | Generator for preorder traversal. |
| `postorder()` | Generator for postorder traversal. |
| `inner_vertices()` | Generator for inner nodes/vertices. |
| `edges()` | Generator for the edges of the tree. |
| `euler_generator()` | Generator for an Euler tour. |
| `supply_leaves()` | Add a `list` of leaf nodes in the subtree of each node as attribute `leaves` to each respective node, and return the full list (the root's list). |
| `contract(edges)` | Contract all edges in the collection `edges`. |
| `get_triples()` | Return a list of all triples that are displayed by the tree. |
| `to_newick()` | Return a `str` representation of the tree in Newick format. Inheriting classes implement their own version of this function. |
| `random_tree(N, binary=False)` | Return a random tree with `N` that is optionally forced to be binary. Stepwise, a new child is attached to a randomly selected node until `N` are reached. |
| `PhyloTree` (corresponding node class: `PhyloTreeNode`) | |
| `sorted_nodes( oldest_to_youngest=True)` | Return a list of nodes sorted by timestamp (default is from oldest, which should be the root, to youngest). |
| `distance_matrix( leaf_order=None)` | Return a list of nodes and a distance matrix on the leaves, optionally takes a list of the leaves that defines their indices in the matrix. |
| `parse_newick(newick)` | Parse a Newick `str`. |
| `to_nx()` | Return a NetworkX DiGraph version of the tree and its `root`. |
| `parse_nx(G, root)` | Convert a tree encoded as a NetworkX DiGraph (together with the root) back into a `PhyloTree`. |
| `serialize(filename, mode=None)` | Serialize a tree in JSON or pickle format specified by `mode`. Default is `None`, in which case the mode is inferred from the filename ending. |
| `load(filename, mode=None)` | Load a tree from a file in JSON or pickle format specified by `mode`. Default is `None`, in which case the mode is inferred from the filename ending. |
| `copy()` | Return a copy of the tree. |
| `Cotree` (corresponding node class: `CotreeNode`) | |
| `to_cograph()` | Return the corresponding cograph as a NetworkX `Graph`. |
| `cotree()` | Convert a cograph into a cotree. |
| `complement( inplace=False)` | Return the cotree of the complement cograph. |
| `copy()` | Return a copy of the cotree. |
| `random_cotree( N, force_series_root=False)` | Returns a random cotree with `N` leaves. Optionally forced to be connected (= root is a series node). |

# C Distributions for sampling

The following distributions are available for sampling:

| distribution | syntax | parameters |
|---|---|---|

| | | |
|---|---|---|
| constant | `x`<br>`('constant', x)` | $x$ must be a number |
| uniform (continuous) | `('uniform', a, b)` | $a <= b$ must be numbers |
| uniform (discrete) | `('discrete_uniform', a, b)` | $a <= b$ must be integers |
| gamma | `('gamma', shape, scale)` | shape and scale must be floats $> 0$ |
| gamma (mean) | `('gamma_mean', mean)` | mean must be a number $> 0$, shape$= 1$ and scale=mean/shape |
| exponential | `('exponential', rate)` | rate must be a float $\geq 0 > 0$ |
| Zipf | `('zipf', a)` | $a > 1$ must be a float value |
| negative binomial | `('negative_binomial', r, q)` | $r \geq 1$ must be an integer,<br>$0 < q < 1$ a float value |