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 [7, 8] (resp. best hits) from known gene trees or evolutionary distances. Moreover, it includes an algorithm to compute supertrees [6] and a method to estimate rooted species trees from an ensemble of orthology/paralogy relations [11].

The library, and especially the simulator, was primarily designed to be able to validate mathematical concepts and 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.** *Submitted to Algorithms for Molecular Biology.*

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

To use the tree reconstruction method for best match inference and the C++ implementation of the quartet method [19], resp., the following software 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):

- RapidNJ [18]
- qinfer

3 Usage

AsymmeTree is divided into several subpackages and modules, an overview of which is given in

Table 1: Overview over the subpackages and modules.

Packages and Modules	**Description**
`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).
`tools`	
`Tree`	Includes the basic class `Tree`, provides functions for tree traversals, Newick parser, etc.
`PhyloTree`	Includes the class `PhyloTree` for phylogenetic trees (inherits from `Tree`), provides a Newick parser, etc.
`BuildST`	Includes the class `BuildST` that computes a supertree from a given list of tress (with overlapping labels) [6].
`DoublyLinkedList`	Implementation of a doubly-linked list.
`GraphTools`	Miscellaneous functions for graphs such as e.g. check for graph equality.
`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` [18] and midpoint rooting.
`Quartets`	Implementation of *Quartet* approach with two different methods for outgroup selection, optionally uses `qinfer`.
`LRTConstructor`	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 [17].
`cograph`	
`Cograph`	Includes the classes `Cotree` and `CotreeNode` as well as a generator for random cotrees/cographs.
`CographEditor`	Implements a heuristic for cograph editing [5].
`LinearCographDetector`	Implements an $\mathcal{O}(n+m)$ algorithm for cograph detection [4].
`paraphylo`	
`SpeciesTreeFromParalogs`	Species tree reconstruction from orthology/paralogy relations. Heuristic version of `ParaPhylo` [11].

Table 1 – *Continued on next page*

Table 1 – *Continued from previous page*

SpeciesTreeFromPO	Species tree reconstruction from ProteinOrtho [15, 16] output.
visualize	
GeneTreeVis	Visualization of a (simulated) gene tree (of type PhyloTree), experimental.

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, can be of type `tuple` (edge) for inner and loss vertices
`tstamp`	time stamp of the event (`double`)
`dist`	evolutionary distance or divergence time from the parent vertex (`double`)
`tranferred`	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.

```
>>> "(3<1>:0.534,2<2>:0.762)S<0>:0.273"
```

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.

If you intend to serialize species or gene trees, I recommend converting them into NetworkX graphs before applying Python's serialization library `pickle`. Note that the information about the ID of the root should be saved too:

```python
import pickle

# tree is of type PhyloTree
tree_nx, root_id = tree.to_nx()

pickle.dump( (tree_nx, root_id), open("tree.pickle", "wb") )
```

To load a tree that was serialized this way use:

```
import pickle
from asymmetree.tools.PhyloTree import PhyloTree

tree_nx, root_id = pickle.load( open("tree.pickle", "rb") )
tree = PhyloTree.parse_nx(tree_nx, root_id)
```

3.2 Simulator for Species and Gene Trees

The subpackage `asymmetree.simulator` contains modules for the simulation and manipulation of species trees and gene trees.

3.2.1 Species Trees

The function `build_species_tree(N)` simulates a dated species tree with `N` leaves (i.e. recent species) using the 'innovation model' described by Keller-Schmidt and Klemm [14]. The following keyword parameters (with their default value) are available:

`planted=True`	add a planted root that has the canonical root as its single neighbor, this way duplication (and loss) events can occur before the first speciation event in a subsequent gene tree simulation
`model="innovation"`	model for the species tree simulation, currently only the 'innovation model' is available
`non_binary=0.0`	probability that an inner edge is contracted, results in a non-binary tree

The time stamps of all vertices are normalized such that the root has time stamp `1.0` and all leaves have time stamp `0.0`.

Example usage:

```
import asymmetree.simulator.TreeSimulator as ts

S = ts.simulate_species_tree(10, planted=True, non_binary_prob=0.2)
print(S.to_newick())
```

3.2.2 Gene Trees

Dated gene trees are simulated along a given species tree `S` with a variant of the Gillespie algorithm [10]. To this end, an instance of the class `GeneTreeSimulator` must be initialized with a species tree of type `PhyloTree`. The following parameters are available (keyword arguments are indicated by `=default`):

`DLH_rates`	a `tuple` of three `floats`, rates for duplication, loss and HGT events in the Gillespie algorithm
`dupl_polytomy=0.0`	allows non-binary duplication events by specifying the lambda parameter for a poisson distribution (copy number = drawn number + 2)

At the moment, loss events in a branch are suppressed whenever this branch is the last survivor in its species branch (by setting the loss rate in the branch to zero). The behaviour is intended to be made optional in future releases.

Example usage:

```
import asymmetree.simulator.TreeSimulator as ts

# S is a species tree of type PhyloTree
TGT_simulator = ts.GeneTreeSimulator(S)
TGT = TGT_simulator.simulate(DLH_rates=(1.0, 0.5, 0.1))
```

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:

```
# observable gene tree
OGT = ts.observable_tree(TGT)
```

3.2.3 Assignment of Variable Evolution Rates

The module `TreeImbalancer` contains a function to model realistic (asymmetric) evolution rates for a given gene tree (see Section **??**). Moreover, correlation in the evolution rate between genes of the same (and closely related) species is introduced (autocorrelation). The function `imbalance_tree(T, S)` takes a gene tree `T` and the **corresponding** species tree `S` as input and manipulated the branch length of the species 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_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
`gamma_param=(0.5, 1.0, 2.2)`	parameter the for Gamma distribution (`a, loc, scale`) from which rate factors for divergent are drawn, the default values are chosen to fit observed asymmetries between paralogs in yeast data [3]
`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

In is recommended to apply the imbalancing to the true gene tree that still contains loss events. Example usage:

```python
import asymmetree.simulator.TreeSimulator as ts
import asymmetree.simulator.TreeImbalancer as tm

S = ts.simulate_species_tree(10)

# true gene tree (with losses)
TGT_simulator = ts.GeneTreeSimulator(S)
TGT = TGT_simulator.simulate(DLH_rates=(0.5, 0.5, 0.5))

# imbalancing
TGT = tm.imbalance_tree(TGT, S, base_rate=1,
                        autocorr_variance=0.2,
                        gamma_param=(0.5, 1.0, 2.2),
                        weights=(1, 1, 1))

# observable gene tree
OGT = ts.observable_tree(TGT)
```

3.2.4 Distance Matrix and Noise

The additive 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 (specifying the indexing) and the distance matrix as a 2-dimensional `numpy` array.

```python
# T is an observable gene tree
leaves, D = T.distance_matrix()
leaf_index_dict = {leaf: i for i, leaf in leaves}
```

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 `"reject"`, `"DOMR"` and `"general"` (see Section ??)

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 specified as `True`, only the first disturbed matrix is returned. Otherwise both are returned in a tuple.

3.3 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 [7, 9, 8]. In contrast, two genes are orthologs if their last common ancestor was a speciation event. Orthology and reciprocal best matches are closely related [8].

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 [19].

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 `best_match_graphs(tree)` and `true_orthology_graph(tree)` return the respective graph representation as NetworkX (di)graphs:

```
from asymmetree.best_matches.TrueBMG import orthology_from_tree,
                                            best_match_graphs

# T is an observable gene tree
orthology_graph = orthology_from_tree(T)
BMG, RBMG = best_match_graphs(T)     # Best Match Graph and Reciprocal Best
                                      Match Graph as a tuple
```

If only distance data is available, best matches have to be estimated. AsymmeTree currently implements three different methods that are described by Stadler et al. [19]:

- Extended Best Hits (module `ExtBestHits`)
- Neighborjoining and midpoint rooting (module `TreeReconstruction`, requires the installation and accessibility of `RapidNJ` [18])
- 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.

3.4 Supertree Computation

The module `BuildST` contains an implementation of the BuildST algorithm described by Deng and Fernández-Baca [6] 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 Henzinger and King [12] and Holm et al. [13]. The latter can also be used separately (see module `asymmetree.tools.hdtgraph.DynamicGraph`)

The class `BuildST` is initialized with a list of trees that are of type `Tree` (thus also `PhyloTree` is allowed). The method `run()` then return 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:

```
from asymmetree.tools.BuildST import BuildST

# tree_list is a list of Tree instances
st_builder = BuildST(tree_list)
supertree = st_builder.run()

if supertree:
  print(supertree.to_newick())
else:
  print("Could not build a supertree!")
```

3.5 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 [11] 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 Crespelle [5] is applied. For the Maximum Consistent Triple Set problem, tree different heuristics are available:

- `BPMF`: Best-Pair-Merge-First [20] (modified for weighted triples)
- `MINCUT`: Aho's `BUILD` with weighted MinCut [1, 2]
- `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:

```
from asymmetree.paraphylo.SpeciesTreeFromParalogs import TreeReconstructor

tree_reconstr = TreeReconstructor()

# ortho_relations is a list of orthology relations
for graph in ortho_relations:
    tree_reconstr.add_ortho_graph(graph)

# finally the tree can be computed
S_estimate = tr.build_species_tree(mode="BPMF")
print(S_estimate.to_newick())
```

The module `SpeciesTreeFromPO` contains functions to estimate a species tree from a `ProteinOrtho` output file. For example, the function `reconstruct_from_PO(filename, triple_mode="BPMF")` takes the filename to the `ProteinOrtho` 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.

References

[1] Aho, A. V., Sagiv, Y., Szymanski, T. G., and Ullman, J. D. Inferring a Tree from Lowest Common Ancestors with an Application to the Optimization of Relational Expressions. *SIAM Journal on Computing*, 10(3):405–421, August 1981. ISSN 0097-5397, 1095-7111. doi: 10.1137/0210030.

[2] Byrka, J., Guillemot, S., and Jansson, J. New results on optimizing rooted triplets consistency. *Discrete Applied Mathematics*, 158(11):1136–1147, June 2010. ISSN 0166218X. doi: 10.1016/j.dam.2010.03.004.

[3] Byrne, K. P. and Wolfe, K. H. Consistent Patterns of Rate Asymmetry and Gene Loss Indicate Widespread Neofunctionalization of Yeast Genes After Whole-Genome Duplication. *Genetics*, 175 (3):1341–1350, March 2007. ISSN 0016-6731, 1943-2631. doi: 10.1534/genetics.106.066951.

[4] Corneil, D. G., Perl, Y., and Stewart, L. K. A Linear Recognition Algorithm for Cographs. *SIAM Journal on Computing*, 14(4):926–934, November 1985. ISSN 0097-5397, 1095-7111. doi: 10.1137/0214065.

[5] Crespelle, C. Linear-Time Minimal Cograph Editing. 2019.

[6] Deng, Y. and Fernández-Baca, D. Fast Compatibility Testing for Rooted Phylogenetic Trees. page 12 pages, 2016. doi: 10.4230/LIPICS.CPM.2016.12.

[7] Geiß, M., Chávez, E., González Laffitte, M., López Sánchez, A., Stadler, B. M. R., Valdivia, D. I., Hellmuth, M., Hernández Rosales, M., and Stadler, P. F. Best match graphs. *Journal of Mathematical Biology*, 78(7):2015–2057, June 2019. ISSN 0303-6812, 1432-1416. doi: 10.1007/s00285-019-01332-9.

[8] Geiß, M., Laffitte, M. E. G., Sánchez, A. L., Valdivia, D. I., Hellmuth, M., Rosales, M. H., and Stadler, P. F. Best match graphs and reconciliation of gene trees with species trees. *Journal of Mathematical Biology*, January 2020. ISSN 0303-6812, 1432-1416. doi: 10.1007/s00285-020-01469-y.

[9] Geiß, M., Stadler, P. F., and Hellmuth, M. Reciprocal best match graphs. *Journal of Mathematical Biology*, 80(3):865–953, February 2020. ISSN 0303-6812, 1432-1416. doi: 10.1007/s00285-019-01444-2.

[10] Gillespie, D. T. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *Journal of Computational Physics*, 22(4):403–434, December 1976. ISSN 00219991. doi: 10.1016/0021-9991(76)90041-3.

[11] Hellmuth, M., Wieseke, N., Lechner, M., Lenhof, H.-P., Middendorf, M., and Stadler, P. F. Phylogenomics with paralogs. *Proceedings of the National Academy of Sciences*, 112(7):2058–2063, February 2015. ISSN 0027-8424, 1091-6490. doi: 10.1073/pnas.1412770112.

[12] Henzinger, M. R. and King, V. Randomized dynamic graph algorithms with polylogarithmic time per operation. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing - STOC '95*, pages 519–527, Las Vegas, Nevada, United States, 1995. ACM Press. ISBN 978-0-89791-718-6. doi: 10.1145/225058.225269.

[13] Holm, J., de Lichtenberg, K., and Thorup, M. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *Journal of the ACM*, 48(4):723–760, July 2001. ISSN 00045411. doi: 10.1145/502090.502095.

[14] Keller-Schmidt, S. and Klemm, K. A model of macroevolution as a branching process based on innovations. *Advances in Complex Systems*, 15(07):1250043, October 2012. ISSN 0219-5259, 1793-6802. doi: 10.1142/S0219525912500439.

[15] Lechner, M., Findeiß, S., Steiner, L., Marz, M., Stadler, P. F., and Prohaska, S. J. Proteinortho: Detection of (Co-)orthologs in large-scale analysis. *BMC Bioinformatics*, 12(1), December 2011. ISSN 1471-2105. doi: 10.1186/1471-2105-12-124.

[16] Lechner, M., Hernandez-Rosales, M., Doerr, D., Wieseke, N., Thévenin, A., Stoye, J., Hartmann, R. K., Prohaska, S. J., and Stadler, P. F. Orthology Detection Combining Clustering and Synteny for Very Large Datasets. *PLoS ONE*, 9(8):e105015, August 2014. ISSN 1932-6203. doi: 10.1371/journal.pone.0105015.

[17] Schaller, D., Geiß, M., Stadler, P. F., and Hellmuth, M. Complete Characterization of Incorrect Orthology Assignments in Best Match Graphs. *arXiv:2006.02249 [cs, math, q-bio]*, June 2020.

[18] Simonsen, M., Mailund, T., and Pedersen, C. N. S. Rapid Neighbour-Joining. In Crandall, K. A. and Lagergren, J., editors, *Algorithms in Bioinformatics*, volume 5251, pages 113–122. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. ISBN 978-3-540-87360-0 978-3-540-87361-7. doi: 10.1007/978-3-540-87361-7_10.

[19] Stadler, P. F., Geiß, M., Schaller, D., Sánchez, A. L., González, M. E., Valdivia, D. I., Hellmuth, M., and Rosales, M. H. From Best Hits to Best Matches. *arXiv:2001.00958 [q-bio]*, January 2020.

[20] Wu, B. Y. Constructing the Maximum Consensus Tree from Rooted Triples. *Journal of Combinatorial Optimization*, 8(1):29–39, March 2004. ISSN 1382-6905. doi: 10.1023/B:JOCO.0000021936.04215.68.