The Journal of Open Source Software

PhyloX: A Python package for complete phylogenetic network workflows

Remie Janssen [1]

1 National Institute for Public Health and the Environment, Bioinformatics and Computing group, Bilthoven, The Netherlands

DOI: 10.21105/joss.06427

Software

- Review
- Repository
- Archive

Editor: Frederick Boehm

Reviewers:

- @abhishektiwari
- @bgyori

Submitted: 20 January 2024
Published: 24 November 2024

Summary

PhyloX is a Python package with tools for generating, manipulating, and analysing phylogenetic networks. It uses the NetworkX package (Hagberg et al., 2008) for basic graph operations. This has the added benefit that the powerful graph tools from NetworkX can be used directly on the phylogenetic networks as well. The aim of the package is to be of general use to phylogenetic network researchers, with a current focus on I/O, random generation of networks, cherry-picking methods, rearrangement operations, and the identification of classes and properties of networks.

Phylogenetic networks

In the study of the evolutionary history of biological species and languages, it is common to represent putative histories using graphs. Traditionally, at least in biology, these graphs are most often trees, such as the well-known tree drawn by Charles Darwin in one of his notebooks. A tree like this is called a phylogenetic tree. In some cases, the evolutionary history includes complex processes like horizontal gene transfer and hybridisation. These processes cause a *reticulate* (i.e., network-like) structure in the evolutionary history, which requires phylogenetic networks to be used for representing the evolutionary histories.

A *directed phylogenetic network* (e.g., (Huson et al., 2010)) is a directed acyclic graph with four types of nodes: - a *root*: an in-degree 0, out-degree 1 node, - a labelled set of *leaves*: in-degree 1, out-degree 0 nodes, - a set of *reticulation nodes*: in-degree >1, out-degree 1 nodes, - a set of *tree nodes*: in-degree 1, out-degree >1 nodes.

A network is *binary* if each reticulation node has in-degree 2, and each tree node has in-degree 2. An *undirected phylogenetic network* is the underlying undirected graph of a directed phylogenetic network, retaining the labelling of the leaf nodes.

Network properties

When analysing or comparing phylogenetic networks or phylogenetic network methods, it can be helpful to extract some (numerical) parameters from the networks. Some of the most used properties are the *reticulation number* (the number of reticulation nodes in a binary network), the number of *blobs* (biconnected components of the network), and the *level* (the maximum reticulation number among all blobs of the network). Of course, the list of studied properties and parameters is much longer, including, for example, the recently introduced B_2-balance index of the network (François et al., 2021).

Classes of networks

In research on phylogenetic networks, it is common to restrict attention to some well-known classes of phylogenetic networks. These classes put additional restrictions on the definition of a network, for the benefit of computational efficiency, to model certain biological restraints, or for both.

Kong et al. (2022) gives a good overview of most well-known classes of directed phylogenetic networks and their biological interpretation. For example, tree-child networks are networks in which each ancestral species has a descendant among the extant taxa (the leaves) through only mutation in the network (Cardona et al., 2009). Mathematically, tree-child networks are characterised as networks in which each non-leaf node has at least one child that is not a reticulation node.

Cherry-picking

A basic structure in any network or tree is the *cherry*, a pair of leaves with a common parent. A modified version often found in phylogenetic networks is the *reticulated cherry*, an ordered pair of leaves (x, y) that are related through the three edges (p_x, x), (p_y, y), and (p_y, p_x).

A common modification to a phylogenetic network is to *pick* or (or *reduce*) a cherry or reticulated cherry. To pick a cherry (x, y), one removes the leaf x from the network together with its incoming edge and then suppresses the resulting degree 2 node if the shared parent of x and y had out-degree 2. *Suppressing* a degree 2 node consists in removing the node and its two incident edges, and replacing them all with one new edge. To pick a reticulated cherry (x, y), one removes the edge (p_y, p_x) and suppresses all resulting degree 2 nodes. The reverse action of picking a cherry is called *adding a cherry*.

These modifications are used in computational tools, for example to reconstruct networks from ancestral profiles (Bai et al., 2021; Cardona et al., 2024; Erdős et al., 2019), to check whether one tree-child network is contained in another (Janssen & Murakami, 2021), or to combine multiple trees into one network (Iersel, Janssen, Jones, Murakami, & Zeh, 2022; Linz & Semple, 2019). Their versatile use has also led to the introduction of the class of *orchard networks* (Erdős et al., 2019; Janssen & Murakami, 2021). This class contains all networks that can be reduced to a single leaf using cherry-picking operations. Networks from this class can be interpreted as trees with horizontal gene transfer arcs (Iersel, Janssen, Jones, & Murakami, 2022).

Rearranging networks

For phylogenetic inference problems, it is often necessary to use heuristics that search through a space of networks. Such a space of networks takes the shape of a graph, whose objects are all networks with a common set of leaf labels (the sampled taxa) and sometimes also a set number of reticulations. The edges of the graph correspond to small changes made to a network: there is an edge between two networks if one can make a modification to one of the networks to arrive at the second network.

The modifications that are allowed are well-defined as types of *rearrangement* operations. These operations can be *horizontal*, keeping the reticulation number the same, or *vertical*, changing the reticulation number. Most horizontal operations are a variation on or restriction of the *rooted subtree prune and regraft* (*rSPR*) operation (Bordewich et al., 2017; Gambette et al., 2017).

The names for vertical moves have not been standardised, but they generally do the same. A vertical move that removes a reticulation removes an incoming edge of a reticulation node, and then suppresses the resulting degree 2 nodes. A vertical move that adds a reticulation does the reverse: it *subdivides* two edges of the network, and adds a new edge between the two new degree 2 nodes (e.g. (Bordewich et al., 2017)).

As mentioned, rearrangement moves can be used to traverse a space of networks. This is used, for example, to sample posterior distributions in Bayesian analyses Zhang et al. (2018) and to find networks that score high on a maximum likelihood criterium (Wen, Yu, Hahn, et al., 2016; Yu et al., 2014).

Generating networks

To test phylogenetic network methods, one either needs to source or create a test set of networks. Creating them is often the simpler option, so methods to randomly generate phylogenetic networks are ready at hand. Moreover, these methods are often based on evolutionary models that are defined on a high level, i.e., with explicit events for processes such as speciation, extinction, and hybridisation.

The paper (Janssen & Liu, 2021) contains a comparison of several 'generators', including several previously existing ones (e.g., (Pons et al., 2019) and (Zhang et al., 2018)) and a new extension of a tree generator to networks.

Representing networks

Because phylogenetic networks are graphs, a common representation is as a list of edges. Another commonly used representation is the extended Newick format (Cardona et al., 2008). The extended Newick notation has a further extension (Rich Newick format) that adds numerical parameters to the edges of the network, such as the branch length and the inheritance probability (for incoming edges of a reticulation node) (Barnett, 2012; Wen et al., 2018).

PhyloX Functionality

PhyloX is equipped to handle all the aspects of phylogenetic networks mentioned in the previous section. It is written primarily for explorative research into algorithmic aspects of phylogenetic networks, although application-focused implementations can also be realised with it. An example is the software (Julien et al., 2023) for the paper (Bernardini et al., 2023), which uses cherry-picking methods in combination with machine learning to efficiently combine a large number of trees into a phylogenetic network. This software shares some of its basic code with the cherrypicking module and the generators module of PhyloX.

I/O

PhyloX handles all stages of a phylogenetic workflow involving networks. This starts and ends with the input/output of networks. The DiNetwork class, which is used to represent phylogenetic networks in PhyloX, inherits from the DiGraph class of NetworkX (Hagberg et al., 2008). Hence, `phylox.DiNetwork` objects can simply be created using the API of `networkx.DiGraph` and adding labels to the leaves:

```
from phylox import DiNetwork
from phylox.constants import LABEL_ATTR

network = DiNetwork()
network.add_edges_from(((0,1),(1,2),(1,3)))
network.nodes[2][LABEL_ATTR] = "leaf1"
network.nodes[3][LABEL_ATTR] = "leaf2"
```

The same can be achieved with a modified initialisation of DiNetwork:

```
from phylox import DiNetwork

network = DiNetwork(
```

```
    edges=((0,1),(1,2),(1,3)),
    labels=[(2,"leaf1"), (3,"leaf2")]
)
```

Alternatively, the network can be initialised from a Newick string with

```
from phylox import DiNetwork

network = DiNetwork.from_newick("((leaf1,leaf2));")
```

NetworkX also provides functionality to output networks in several formats. For example, it is possible to output the list of edges or to create a drawing of the network. Of course, output as Newick string is also available with PhyloX (with `network.newick()` for a network called `network` as in the example code blocks above). This outputs all edge information in rich Newick format by default, but can also be forced to output an extended Newick string without edge information.

Generating networks

Networks can also be generated randomly in PhyloX, which can be utilised to create test sets for new methods. The implemented generators are based on the code from (Janssen & Liu, 2021). These include generators based on evolutionary models, such as the LGT generator and the ZODS generator based on (Pons et al., 2019) and (Zhang et al., 2018), but also a [Metropolis-Hastings sampler] enabling uniform sampling from classes of networks.

The latter makes use of a large part of the functionality of PhyloX, especially when sampling orchard networks: after generating or choosing a starting network, the `phylox.generators.mcmc.sample_mcmc_networks` randomly traverses the space of phylogenetic networks using the rearrangement module, and rejects proposals if the resulting network is not orchard using the cherry-picking module.

```
from phylox.generators.randomTC import generate_network_random_tree_child_sequence
from phylox.generators.mcmc import sample_mcmc_networks
from phylox.classes import is_orchard
from phylox.rearrangement.move import MoveType

# Generate an arbitrary orchard network with 10 leaves and 5 reticulations
start_network = generate_network_random_tree_child_sequence(10, 5, seed=4321)
# Generate 100 orchard networks with 10 leaves and 5 reticulations
sampled_networks = sample_mcmc_networks(
    start_network,
    {MoveType.TAIL: 0.5, MoveType.HEAD: 0.5},
    number_of_samples=100,
    burn_in=5,
    restriction_map=is_orchard,
    add_root_if_necessary=True,
    correct_symmetries=False,
    seed=1234,
)
# Write the sampled networks to a file
with open("sampled_networks.nwk", "w") as f:
    for network in sampled_networks:
        f.write(network.newick() + "\n")
```

For this sampler to work correctly, the space of networks that is sampled from needs to be connected. That is, it has to be possible to transform each network into each other network in the space using the selected rearrangement moves. In the example above, this means that the space of orchard networks with 10 leaves and 5 reticulations needs to be connected under tail moves and head moves (i.e., rSPR moves).

This is something the user needs to check or prove themselves, as it is not viable to check this computationally. Fortunately, such connectivity results have been studied in detail (Erdős et al., 2021; Iersel, Janssen, Jones, & Murakami, 2022; Janssen, 2021; Klawitter, 2020). For example, the result needed to prove that this example is correct can be found in (Iersel, Janssen, Jones, & Murakami, 2022).

Comparing networks

Based on all the properties above, PhyloX provides a toolkit to compare networks. For example, it can be used to determine whether two networks are isomorphic (i.e., the same); whether they have the same properties: level, number of blobs, reticulation number, and number of (reticulated) cherries; whether one is contained in the other if both are tree-child; and whether they are similar with respect to a rearrangement distance.

Statement of Need

Currently, no Python package enables a full workflow for analysing properties and methods of phylogenetic networks. Isolated scripts for this purpose do appear on GitHub or as pseudocode regularly, most often as part of publications studying one method or one property (Janssen et al., 2020; Janssen, 2021; Janssen & Murakami, 2020; Pons et al., 2019; Zhang et al., 2018). Combining such scripts requires substantial work, for example because the phylogenetic networks themselves are represented by different Python classes with their own methods.

This package, PhyloX, aims to bring these scripts together: it standardises implementations of several basic objects related to phylogenetic networks, such as the networks themselves, the labelling of the nodes, and rearrangement moves. It currently implements a limited but important set of basic functions: I/O for networks (e.g., lists of edges and extended Newick format), network generation for test sets, comparing networks resulting from reconstruction methods, and computing several well-used network properties such as the reticulation number, the level, and the number of cherries.

Related packages

As mentioned above, there are currently no Python packages that enable a complete workflow for phylogenetic networks. However, some Python packages are available that enable part of this workflow or a very similar one. In this section, we compare the functionality of several of these packages to PhyloX, focussing only on usability for phylogenetic networks.

PhyloNetwork

Like PhyloX, PhyloNetwork is a Python package based on NetworkX. It has a richer implementation for phylogenetic trees than PhyloX. For example, it includes more tree-specific rearrangement moves, the calculation of node properties such as the latest common ancestor (LCA), and some presets for drawing networks.

However, it has very few methods for phylogenetic networks, and most of those methods are also included in PhyloX. Another advantage of using PhyloX over PhyloNetwork is the inclusion of explicit random seeds. This is an important factor for the reproducibility of research.

Note that code from PhyloNetwork and PhyloX may be easy to combine, as both use NetworkX to implement the phylogenetic network class.

Biopython - Phylo

This phylogenetics module, Phylo (Talevich et al., 2012), of the Biopython package (Cock et al., 2009) is built for phylogenetic analyses in Python. However, it is set up for phylogenetic trees only. The encoding of trees as sets of clades does not easily allow extension to networks, which makes it unsuitable to use for these phylogenetic network methods.

DendroPy

Like Biopython's phylogenetics package, the DendroPy package focuses on phylogenetic trees (Sukumaran & Holder, 2010). Unlike Biopython, the implementation of the trees in DendroPy is graph-based, making it more suited for analyses of phylogenetic networks. This could still require large changes, as some properties of trees are built into the code on a fairly fundamental level, such as each node having (at most one) parent node.

Availability

The code of PhyloX is available as an open-source project on GitHub under the BSD 3-Clause licence. The package is also available via PyPI, so it can be installed via pip (or pip in conda), and updates to the release branch are automatically converted into new versions of the package. The releases are recorded in Zenodo, so persistent identifiers can be used to cite specific releases of the software. When citing this software, please make sure to also cite the original source of the code, which is mentioned in the documentation of each method or class.

Acknowledgements

Most of the code has been written in the form of separate scripts during the author's PhD project, which was conducted under Leo van Iersel's Vidi grant: 639.072.6

Anyone willing to contribute is very welcome to do so via pull requests and issues on GitHub!

References

Bai, A., Erdős, P. L., Semple, C., & Steel, M. (2021). Defining phylogenetic networks using ancestral profiles. *Mathematical Biosciences*, *332*, 108537. https://doi.org/10.1016/j.mbs.2021.108537

Barnett, R. (2012). Rich newick format. In *Rich Newick Format - Phylonet - Rice University Campus Wiki*. https://wiki.rice.edu/confluence/display/PHYLONET/Rich+Newick+Format

Bernardini, G., Iersel, L. van, Julien, E., & Stougie, L. (2023). Constructing phylogenetic networks via cherry picking and machine learning. *Algorithms for Molecular Biology*, *18*(1), 13. https://doi.org/10.1186/s13015-023-00233-3

Bordewich, M., Linz, S., & Semple, C. (2017). Lost in space? Generalising subtree prune and regraft to spaces of phylogenetic networks. *Journal of Theoretical Biology*, *423*, 1–12. https://doi.org/10.1016/j.jtbi.2017.03.032

Cardona, G., Pons, J. C., Ribas, G., & Coronado, T. M. (2024). Comparison of orchard networks using their extended μ-representation. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*. https://doi.org/10.1109/TCBB.2024.3361390

Cardona, G., Rossello, F., & Valiente, G. (2009). Comparison of tree-child phylogenetic networks. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, *6*(4), 552–569. https://doi.org/10.1109/TCBB.2007.70270

Cardona, G., Rosselló, F., & Valiente, G. (2008). Extended newick: It is time for a standard representation of phylogenetic networks. *BMC Bioinformatics*, *9*, 1–8. https://doi.org/10.1186/1471-2105-9-532

Cock, P. J. A., Antao, T., Chang, J. T., Chapman, B. A., Cox, C. J., Dalke, A., Friedberg, I., Hamelryck, T., Kauff, F., Wilczynski, B., & Hoon, M. J. L. de. (2009). Biopython: freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics*, *25*(11), 1422–1423. https://doi.org/10.1093/bioinformatics/btp163

Erdős, P. L., Francis, A., & Mezei, T. R. (2021). Rooted NNI moves and distance-1 tail moves on tree-based phylogenetic networks. *Discrete Applied Mathematics*, *294*, 205–213. https://doi.org/10.1016/j.dam.2021.02.016

Erdős, P. L., Semple, C., & Steel, M. (2019). A class of phylogenetic networks reconstructable from ancestral profiles. *Mathematical Biosciences*, *313*, 33–40. https://doi.org/10.1016/j.mbs.2019.04.009

François, B., Cardona, G., & Celine, S. (2021). Revisiting shao and sokal's b 2 index of phylogenetic balance. *Journal of Mathematical Biology*, *83*(5). https://doi.org/10.1007/s00285-021-01662-7

Gambette, P., Iersel, L. van, Jones, M., Lafond, M., Pardi, F., & Scornavacca, C. (2017). Rearrangement moves on rooted phylogenetic networks. *PLoS Computational Biology*, *13*(8), e1005611. https://doi.org/10.1371/journal.pcbi.1005611

Hagberg, A. A., Schult, D. A., & Swart, P. J. (2008). Exploring network structure, dynamics, and function using NetworkX. In G. Varoquaux, T. Vaught, & J. Millman (Eds.), *Proceedings of the 7th python in science conference* (pp. 11–15). https://doi.org/10.25080/TCWV9851

Huson, D. H., Rupp, R., & Scornavacca, C. (2010). *Phylogenetic networks: Concepts, algorithms and applications*. Cambridge University Press. https://doi.org/10.1017/CBO9780511974076

Iersel, L. van, Janssen, R., Jones, M., & Murakami, Y. (2022). Orchard networks are trees with additional horizontal arcs. *Bulletin of Mathematical Biology*, *84*(8), 76. https://doi.org/10.1007/s11538-022-01037-z

Iersel, L. van, Janssen, R., Jones, M., Murakami, Y., & Zeh, N. (2022). A practical fixed-parameter algorithm for constructing tree-child networks from multiple binary trees. *Algorithmica*, *84*(4), 917–960. https://doi.org/10.1007/s00453-021-00914-8

Janssen, R. (2021). *Rearranging phylogenetic networks* [PhD thesis, Delft University of Technology]. https://doi.org/10.4233/uuid:1b713961-4e6d-4bb5-a7d0-37279084ee57

Janssen, R., Jones, M., & Murakami, Y. (2020). Combining networks using cherry picking sequences. *International Conference on Algorithms for Computational Biology*, 77–92. https://doi.org/10.1007/978-3-030-42266-0_7

Janssen, R., & Liu, P. (2021). Comparing the topology of phylogenetic network generators. *Journal of Bioinformatics and Computational Biology*, *19*(06), 2140012. https://doi.org/10.1142/S0219720021400126

Janssen, R., & Murakami, Y. (2020). Linear time algorithm for tree-child network containment. *International Conference on Algorithms for Computational Biology*, 93–107. https://doi.org/10.1007/978-3-030-42266-0_8

Janssen, R., & Murakami, Y. (2021). On cherry-picking and network containment. *Theoretical Computer Science*, *856*, 121–150. https://doi.org/10.1016/j.tcs.2020.12.031

Julien, E., Bernardini, G., Stougie, L., & Iersel, L. van. (2023). *Source code for the paper "constructing phylogenetic networks via cherry picking and machine learning"*. 4TU.ResearchData. https://doi.org/10.4121/c679cd3c-0815-4021-a727-bcb8b9174b27.v1

Klawitter, J. (2020). *Spaces of phylogenetic networks* [PhD thesis, University of Auckland]. http://hdl.handle.net/2292/50188

Kong, S., Pons, J. C., Kubatko, L., & Wicke, K. (2022). Classes of explicit phylogenetic networks and their biological and mathematical significance. *Journal of Mathematical Biology*, *84*(6), 47. https://doi.org/10.1007/s00285-022-01746-y

Linz, S., & Semple, C. (2019). Attaching leaves and picking cherries to characterise the hybridisation number for a set of phylogenies. *Advances in Applied Mathematics*, *105*, 102–129. https://doi.org/10.1016/j.aam.2019.01.004

Pons, J. C., Scornavacca, C., & Cardona, G. (2019). Generation of level-k LGT networks. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, *17*(1), 158–164. https://doi.org/10.1109/TCBB.2019.2895344

Sukumaran, J., & Holder, M. T. (2010). DendroPy: A python library for phylogenetic computing. *Bioinformatics*, *26*(12), 1569–1571. https://doi.org/10.1093/bioinformatics/btq228

Talevich, E., Invergo, B. M., Cock, P. J., & Chapman, B. A. (2012). Bio. Phylo: A unified toolkit for processing, analyzing and visualizing phylogenetic trees in biopython. *BMC Bioinformatics*, *13*, 1–9. https://doi.org/10.1186/1471-2105-13-209

Wen, D., Yu, Y., Hahn, M. W., & Nakhleh, L. (2016). Reticulate evolutionary history and extensive introgression in mosquito species revealed by phylogenetic network analysis. *Molecular Ecology*, *25*(11), 2361–2372. https://doi.org/10.1111/mec.13544

Wen, D., Yu, Y., & Nakhleh, L. (2016). Bayesian inference of reticulate phylogenies under the multispecies network coalescent. *PLoS Genetics*, *12*(5), e1006006. https://doi.org/10.1073/pnas.1407950111

Wen, D., Yu, Y., Zhu, J., & Nakhleh, L. (2018). Inferring phylogenetic networks using PhyloNet. *Systematic Biology*, *67*(4), 735–740. https://doi.org/10.1093/sysbio/syy015

Yu, Y., Dong, J., Liu, K. J., & Nakhleh, L. (2014). Maximum likelihood inference of reticulate evolutionary histories. *Proceedings of the National Academy of Sciences*, *111*(46), 16448–16453. https://doi.org/10.1073/pnas.1407950111

Zhang, C., Ogilvie, H. A., Drummond, A. J., & Stadler, T. (2018). Bayesian inference of species networks from multilocus sequence data. *Molecular Biology and Evolution*, *35*(2), 504–517. https://doi.org/10.1093/molbev/msx307