BIOINFORMATICS APPLICATIONS NOTE
Vol. 26 no. 12 2010, pages 1569–1571
doi:10.1093/bioinformatics/btq228

Phylogenetics

Advance Access publication April 25, 2010

DendroPy: a Python library for phylogenetic computing

Jeet Sukumaran* and Mark T. Holder

Department of Ecology and Evolutionary Biology, University of Kansas, Lawrence, USA

Associate Editor: David Posada

ABSTRACT

Summary: DendroPy is a cross-platform library for the Python programming language that provides for object-oriented reading, writing, simulation and manipulation of phylogenetic data, with an emphasis on phylogenetic tree operations. DendroPy uses a splits-hash mapping to perform rapid calculations of tree distances, similarities and shape under various metrics. It contains rich simulation routines to generate trees under a number of different phylogenetic and coalescent models. DendroPy's data simulation and manipulation facilities, in conjunction with its support of a broad range of phylogenetic data formats (NEXUS, Newick, PHYLIP, FASTA, NeXML, etc.), allow it to serve a useful role in various phyloinformatics and phylogeographic pipelines.

Availability: The stable release of the library is available for download and automated installation through the Python Package Index site (http://pypi.python.org/pypi/DendroPy), while the active development source code repository is available to the public from GitHub (http://github.com/jeetsukumaran/DendroPy).

Contact: jeet@ku.edu

Received on February 10, 2010; revised on April 16, 2010; accepted on April 21, 2010

1 INTRODUCTION

Here we describe DendroPy, a cross-platform library for the management, manipulation and analysis of phylogenetic tree and character data using the Python programming language. DendroPy addresses needs of researchers in phyloinformatics and population genetics. It supports reading and writing phylogenetic data in a wide variety of file formats (NEXUS, PHYLIP, FASTA, NeXML, etc.) to and from the same common object-oriented data model.

Its data model is rich and well-suited for manipulating complex datasets. Objects represent both core phylogenetic entities (such as taxa, trees, and character matrices), as well both components and collections of these objects. For example, DendroPy's representation of phylogenetic trees includes a class for the tree itself, but also classes for nodes, directed edges and support for the concept of splits (also known as bipartitions), as well as collections of these tree objects. Splits are crucial in many phylogenetic algorithms that operate on unrooted tree, such as consensus tree generation (Margush and McMorris, 1981) and the calculation of tree-to-tree distances (Robinson and Foulds, 1981). DendroPy's hashing of splits allows for constant-time lookup of an edge in a tree. In memory, trees are represented as rooted, but splits of either unrooted or rooted trees are efficiently stored by hashing. Thus, DendroPy handles both types of trees very naturally.

*To whom correspondence should be addressed.

DendroPy is designed to deal with diverse sources of datasets. Character data and trees refer to rich taxon objects, and users of the library have fine grained control of how these taxa are coordinated or kept separate across different sources of data. Thus, importing information from a wide variety of sources can be accomplished without unintentionally concatenating information. This makes DendroPy a convenient tool in the domains of bioinformatics (e.g. phylogenetic supermatrix assembly and phylogeography), in which the character data associated with trees is derived from multiple exemplars of the same biological taxon. DendroPy also provides a rich framework for the simulation of trees under a variety of branching models, such as the birth–death model (uniform or variable), the coalescent (Kingman, 1982), the censored coalescent (Rannala and Yang, 2003), etc. Below we describe the main features of DendroPy in more detailed terms, and discuss its relationship to the existing python libraries for bioinformatics.

2 DENDROPY

DendroPy is pure-Python library with no external dependencies beyond the availability of a Python 2 interpreter of version 2.4 or greater. It can easily be installed with a single command on a variety of platforms without systems administration privileges or advanced expertise. The primary documentation for DendroPy is included with the library installation as well as on the library's website in the form of a 'cookbook'. This tutorial provides practical examples illustrating the use of its classes and methods. In addition, all the major classes and methods have extensive documentation in the form of Python 'docstrings', which make the information available to the user through the native '`help`' command of Python. In this article, therefore, we only provide a brief conceptual overview of the basic data model, as well as a synopsis of the major functions of the library.

DendroPy can read and write trees in NEXUS, Newick and NeXML formats, and read and write character data in NEXUS, PHYLIP, FASTA and NeXML formats. Trees and character data can be accumulated into the same dataset object from across multiple files, and functions allow for the normalization (homogenization) of taxon references from across these files, based on common taxon labels.

Within DendroPy's object model, each set of trees or character data is associated with a particular set of taxa. The use of rich taxon objects, rather than simple characters strings that store taxon names is crucial, because it allows DendroPy to support simple formats (in which the name of a taxon is sufficient to identify it) or formats such as NeXML (Vos, 2008) that use unique identifiers distinct from the taxon name to establish linkages between objects. Rich taxon objects also make it easier to deal with name clashes resulting from taxonomic synonymies, or maintaining correspondences between datasets, in which the taxon labels have been truncated or decorated with modifiers. DendroPy's '`DataSet`' object tracks multiple sets of taxa along with trees and character data associated with each taxon set. Explicitly reflecting the possibility of having multiple lists of taxa in the object model allows DendroPy to support matrix combination and partitioning operations in a sophisticated way.

Trees in DendroPy can be instantiated into rich objects with Node and Edge objects to reflect the topology of the tree. Efficient manipulation and querying of the tree objects are made possible by split 'masks', which use a single bit to represent the bipartition of taxa that the deletion of the branch would induce. These split masks are also stored in a hash table that maps them to the edge in question. This allows for rapid and accurate establishment of split identity across multiple trees. Through these split masks, trees that share the same set of taxa can be compared across multiple datasets using various metrics, such as weighted Robinson–Foulds distances (Robinson and Foulds, 1981), symmetrical differences (Felsenstein, 2004), etc. DendroPy also provides functions for the calculation of various statistics on a single tree, such as tree height, tree length (sum of branch lengths), the γ statistic of Pybus and Harvey (2000), goodness-of-fit to or Kullback-Leibler divergence (Kullback and Leibler, 1951) from a coalescent model, patristic distance between leaves, distances between nodes, etc.

Iteration is an central concept in Python (and many other programming languages). DendroPy provides iterators that return either nodes or edges for trees in a variety of traversal styles including pre-order, post-order or in-order traversal. Traversal can be constrained to a subset of nodes or edges through flexible filters. Expressing tree traversal via Python iterable interface allows phylogenetic operations to be expressed concisely, and without indepth knowledge of internal implementation details of DendroPy's Tree object.

Trees can also be manipulated structurally by adding or removing branches or taxa (i.e. terminal branches), or being re-rooted at different nodes. Functions are already provided to generate trees under various different branching models, including birth–death (under both constant and variable rates), the neutral coalescent, the constrained (or truncated) coalescent, etc., but the tree manipulation and growing functions provided by DendroPy allow new tree simulation functions to be written very easily and rapidly.

3 'SUMTREES' AND OTHER APPLICATIONS

In addition to various classes and functions to facilitate phylogenetic computation with Python, DendroPy also provides several 'end-user' applications and scripts that can be used even by investigators without a programming background. An example is 'SumTrees', a program to summarize non-parameteric bootstrap or Bayesian posterior probability support for splits or clades on phylogenetic trees. This program takes advantage of DendroPy's splits hash-map to rapidly enumerate splits in one or more collections of trees. The proportion of trees out of the sample from multiple files in which a particular split is found is taken to be the degree of support for that split, with a burn-in option that allows for an initial number of trees in each file to be excluded from the analysis if they are not considered to be drawn from the true support distribution. The support for the splits will be mapped onto one or more target trees either in terms of node labels or branch lengths. The target trees can be supplied by the user. If no target trees are given, then a majority-rule clade consensus tree will be constructed based on the samples given.

Other scripts include those to convert data between various standard formats, calculate the probability of trees under a coalescent model, construct a table of frequencies of splits in different empirical distributions of trees, etc. In addition to providing immediate utility, these scripts serve as canonical examples of usage of the library, and thus provide guidelines or templates for custom scripts to be developed.

While there are no tree inference functions provided by the library itself, a function is provided that delegates the tree estimation under various different criteria to PAUP* (Swofford, 2003), and retrieves the results in the form of a collection of DendroPy trees as well as a Python dictionary object representing the maximum likelihood estimate of the character substitution model.

4 INTEROPERABILITY WITH OTHER LIBRARIES

A variety of libraries supporting computing and operations with phylogenetic data exist, including a number specifically for the Python programming language: e.g., BioPython (Cock *et al.*, 2009), PyCogent (Knight *et al.*, 2007), ETE (Huerta-Cepas *et al.*, 2010) or P4 (Foster, 2010). However, despite the variety of rich, powerful and flexible phylogenetic toolkits available for the Python programming language, there remains a deficiency in terms of some important tree- or tree-shape-based calculations, analysis and comparisons, as well as tree- and tree-shape simulations, often performed in the phylogenetic context.

For example, while BioPython (Cock *et al.*, 2009) has the ability to read NEXUS-formatted tree files, its emphasis is on gene and genomic sequence-based operations and it has limited functionality with respect to tree-based operations. On the other hand, both PyCogent (Knight *et al.*, 2007) and ETE (Huerta-Cepas *et al.*, 2010) have a much richer tree operation functionality, allowing for comprehensive tree manipulation operations and powerful and flexible tree visualizations. At the same time, however, these two libraries do not have many of the tree metric, analysis, comparison and simulation functionality found in DendroPy. P4 (Foster, 2010), in constrast, has some basic tree comparison functionality, but lacks the more advanced tree manipulation and visualization functionality found in PyCogent (Knight *et al.*, 2007) and ETE (Huerta-Cepas *et al.*, 2010), and lacks tree simulation functionality altogether. Similarly, there are several aspects or functionalities that are markedly absent in DendroPy that are provided by these other libraries, the most important of that are tree visualization and public sequence database querying and retrieval.

These differences do not reflect deficits or incompleteness of any of the existing libraries, but rather differences in the application goals and use-case concepts motivating the underlying design of the libraries. For example, BioPython, PyCogent and ETE are motivated by application in the phylogenomic domain, where splits comparisons across trees or operations under the coalescent framework are not usually required, while P4 emphasizes phylogenetic tree inference, where simulations under the coalescent are not usually required and tree-manipulation operations are not usually exposed to clients.

As such, these libraries serve to complement rather than replace each other, and depending on a particular task or application, one library may be more suitable than another. Recognizing this, and recognizing the usefulness of being able to use multiple libraries simultaneously, we have implemented the facility to seamlessly export DendroPy tree objects to ETE and vice versa, and are in the process of expanding this support to PyCogent and BioPython. Furthermore, DendroPy already has the facility to exchange data with other useful libraries, such as APE (Paradis *et al.*, 2004) for the R statistical programming language (R Development Core Team, 2009), and we are working on expanding this support to other R libraries, such as Geiger (Harmon *et al.*, 2009).

ACKNOWLEDGEMENTS

We wish to thank the students and faculty of KU and other institutions who provided feedback on the usage and documentation of the library, as well as the editors and anonymous reviewers whose comments helped to greatly improve this paper.

Funding: J.S. acknowledges funding support from a subaward of CIPRES grant (NSF Award # 0715370 to Warnow; subaward to M.T.H.).

Conflict of Interest: none declared.

REFERENCES

Cock,P.J. *et al*. (2009) Biopython: freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics*, **25**, 1422–1423.

Felsenstein,J. (2004) *Inferring phylogenies*. Sinauer Associates, Sunderland, MA, 580 pages.

Foster,P. (2010) P4, a python package for phylogenetics. Available at http://bmnh.org/~pf/p4.html (last accessed date February 02, 2010).

Harmon,L. *et al*. (2009) *geiger: Analysis of evolutionary diversification*. R package version 1.3-1.

Huerta-Cepas,J. *et al*. (2010) ETE: a python environment for tree exploration. *BMC Bioinformatics*, **11**, 24.

Kingman,J. (1982) The coalescent. *Stochastic Processes Appl.*, **13**, 235–248.

Knight,R. *et al*. (2007) Pycogent: a toolkit for making sense from sequence. *Genome Biol.*, **8**, R171.

Kullback,S. and Leibler,R. (1951) On information and sufficiency. *Ann. Math. Stat.*, **22**, 79–86.

Margush,T. and McMorris,F. (1981) Consensus *n*-trees. *Bull. Math. Biol.*, **43**, 239–244.

Paradis,E. *et al*. (2004) APE: analyses of phylogenetics and evolution in R language. *Bioinformatics*, **20**, 289–290.

Pybus,O. and Harvey,P. (2000) Testing macro-evolutionary models using incomplete molecular phylogenies. *Proc. R. Soc. B: Biol. Sci.*, **267**, 2267.

R Development Core Team (2009) *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Rannala,B. and Yang,Z. (2003) Bayes estimation of species divergence times and ancestral population sizes using DNA sequences from multiple loci. *Genetics*, **164**, 1645–1656.

Robinson,D.F. and Foulds,L.R. (1981) Comparison of phylogenetic trees. *Math. Biosci.*, **53**, 131–147.

Swofford,D.L. (2003) *PAUP*. Phylogenetic Analysis Using Parsimony (*and Other Methods)*. Sinauer Associates, Sunderland, MA.

Vos,R. (2008) Data standards in phylogenetics: the nexml project. In Weitzman,A.L. and Belbin,L. (eds), *Proceedings of TDWG (2008), Fremantle, Australia.*