SOFTWARE

TreeSwift: a massively scalable Python package for trees

Niema Moshiri

Correspondence:
niemamoshiri@gmail.com
Bioinformatics and Systems Biology Graduate Program, UC San Diego, Gilman Drive, 92093 La Jolla, California, USA
Full list of author information is available at the end of the article

Abstract

Background: Phylogenetic trees are essential to evolutionary biology, and numerous methods exist that attempt to extract information applicable to a wide range of disciplines, such as epidemiology and metagenomics. Currently, the three main Python packages for trees are Bio.Phylo, DendroPy, and the ETE Toolkit, but as dataset sizes grow, parsing and manipulating ultra-large trees becomes impractical.

Results: This article describes TreeSwift, a Python package for traversing and manipulating trees that does not require any external dependencies, and it then compares its performance against that of Bio.Phylo, DendroPy, and the ETE Toolkit.

Conclusions: TreeSwift is a freely available, open source, and scalable Python package for traversing and manipulating trees. Source code and documentation are publicly available on GitHub (www.github.com/niemasd/TreeSwift), and it can easily be installed from PyPI (www.pypi.org/project/treeswift) using `pip`.

Keywords: phylogenetics; tree; traversal; Python; scalable

Background

Phylogenetic trees are essential to evolutionary biology, and phylogenetic methods are applicable to a wide range of disciplines, such as epidemiology [1, 2, 3] and metagenomics [4, 5, 6]. However, the datasets used by these methods are growing rapidly as sequencing costs continue to fall, emphasizing the need for scalable methods of tree traversal and manipulation.

Currently, the three main Python packages for trees are the Bio.Phylo module of Biopython [7], DendroPy [8], and the ETE Toolkit [9]. The three tools are simple to integrate into new methods, include a plethora of functions that cater to most phylogenetics needs, and are fast for reasonably-sized trees. However, as dataset sizes grow, parsing and manipulating ultra-large trees becomes impractical. I introduce TreeSwift, a scalable cross-platform Python package for traversing and manipulating trees that does not require any external dependencies, and I compare its performance against that of Bio.Phylo, DendroPy, and the ETE Toolkit.

Implementation

TreeSwift is a pure-Python package that has no external dependencies. It has been tested on Python versions 2.6–2.7 and 3.3–3.7, but it should theoretically work on all versions. It is also compiled and hosted on PyPI, meaning it can easily be installed with a single `pip` command without any need for administrative privileges or any

advanced knowledge. This is essential to contrast against the ETE Toolkit, which requires the Six and NumPy Python libraries to install if the user has administrative privileges or Anaconda/Miniconda to install if the user doesn't, and BioPython, which requires a C compiler and the NumPy Python library as well as computer fluency to compile tools from source using a `Makefile`.

The primary TreeSwift documentation can be found as an Application Programming Interface (API) on the GitHub repository. Further, the classes and functions of TreeSwift are thoroughly documented with Python "docstrings," so the user can access usage information via Python's `help` command.

A key feature of TreeSwift is its simplicity in class design in order to reduce time and memory overhead of loading, traversing, and manipulating trees. The entire package consists of just two classes: a `Node` class, which contains the data and local relationships, and a `Tree` class, which handles manipulation and traversal on the `Node` objects. A key distinction between TreeSwift and DendroPy is that DendroPy stores bipartition information to enable efficient comparisons between multiple trees that share the same set of taxa, but because TreeSwift is designed for the fast traversal and manipulation of a single tree (and not the comparison of multiple trees), TreeSwift forgoes this feature to avoid the accompanied overhead, resulting in a much lower memory footprint and faster execution of equivalent functions.

TreeSwift provides generators that iterate over the nodes of a given tree in a variety of traversals, including pre-order, in-order, post-order, level-order, ordered by distance from the root, and more. These built-in generators intend to provide developers a simple yet efficient manner in which to implement their own algorithms such that they only need to consider higher-level details of the traversal process. TreeSwift also provides the ability to compute various statistics of a given tree, such as tree height, average branch length, patristic distances between nodes in the tree, treeness [10], the Gamma statistic [11], etc.

Results and Discussion

To compare the performance of these tools, I tried to choose functionality that would be essential to most users and that is compatible with as many of the tools as possible: pre-order traversal (visit the current node before visiting the children), post-order traversal (visit the children before visiting the current node), in-order traversal (in fully-bifurcating trees, visit the left child, then visit the current node, then visit the right child), level-order traversal (visit the nodes of the tree in order of path length from root, where path length is defined as the number of edges between a given node and the root, ignoring edge lengths), root-distance-order traversal (visit the nodes of the tree in order of path length from root, where path length is defined as the sum of edge lengths in the path from a given node to the root), computing total branch length (i.e., the sum of the lengths of all branches in the tree), finding the most recent common ancestor (MRCA) of a set of leaves, and computing a pairwise distance matrix of the leaves of the tree. Not all tasks I chose to test were natively implemented in all of the tools, so I used example code from authors when available, wrote my own code if the task was decently trivial, or omitted the tool from the comparison entirely if there was no clear simple way to perform the task. The code used to perform each task using each tool can be found

at www.github.com/niemasd/TreeSwift-Paper. The results of the comparisons can be seen in Figure 1.

When compared against DendroPy, TreeSwift is significantly faster for all tested functions (an order of magnitude faster for most) and has a memory footprint that is over an order of magnitude lower. When compared against Bio.Phylo, TreeSwift is significantly faster when finding the MRCA of a set of leaves, is roughly twice as fast when computing total branch length and when performing pre-order and post-order traversals, is roughly equivalent in speed when performing a level-order traversal, is significantly slower when computing a pairwise distance matrix, and has a slightly lower memory footprint. Note that Bio.Phylo does not seem to implement a way to perform an in-order nor a root-distance-order traversal. When compared against the ETE Toolkit, TreeSwift is significantly faster when computing a pairwise distance matrix, is much faster when computing total branch length and when performing pre-order and post-order traversals, is roughly equivalent in speed when performing a level-order traversal and when finding the MRCA of a set of leaves, and has a slightly lower memory footprint. Note that, although the ETE Toolkit was roughly as fast as TreeSwift when finding the MRCA of a set of leaves, the tool was not able to scale to a tree with 1,000,000 leaves because of excessive memory consumption during the MRCA search. Also note that the ETE Toolkit does not seem to implement a way to perform an in-order nor a root-distance-order traversal.

It must be emphasized that these comparisons do not reflect deficits in the existing packages against which TreeSwift was compared. These other libraries allow the integration of richer datasets, such as the integration of sequence data onto the taxa, as well as other relevant features TreeSwift has no intention of replicating, as they would likely come at a performance loss. Instead, these comparisons intend to highlight differences in use-cases between the various tools, and TreeSwift is designed to complement these other packages by providing a simple yet fast tree package meant specifically for traversal-intensive tasks on ultra-large phylogenetic trees.

Conclusions

TreeSwift is an open source Python 2 and 3 tree package suitable for phylogenetic algorithms on ultra-large trees. TreeSwift can be installed easily from PyPI using the `pip` command, and the cross-platform nature of Python as well as the simplicity of the classes within TreeSwift poise the package for community contribution.

Availability and Requirements

Project name: TreeSwift
Project home page: www.github.com/niemasd/TreeSwift
Operating system(s): Platform independent
Programming language: Python (2 or 3)
Other requirements: None
License: GNU General Public License v3.0
Any restrictions to use by non-academics: None

Abbreviations

API: Application Programming Interface

Declarations

Acknowledgements
I would like to acknowledge Siavash Mirarab for his mentorship. I would also like to acknowledge Jeet Sukumaran and Mark Holder, as DendroPy provided much motivation during TreeSwift's development.

Funding
This work was supported by NIH subaward 5P30AI027767-28 to NM.

Availability of data and materials
All test datasets and scripts to perform the performance comparisons and to generate the figures can be found on GitHub (www.github.com/niemasd/TreeSwift-Paper). The source code and documentation for the tool itself can also be found on GitHub (www.github.com/niemasd/TreeSwift). TreeSwift can be installed from PyPI using the `pip` command by following the instructions on the tool's documentation page.

Author's contributions
NM completed all portions of the project.

Competing interests
The author declares that he has no competing interests.

Consent for publication
Not Applicable.

Ethics approval and consent to participate
Not Applicable.

References
1. Ragonnet-Cronin, M., Hodcroft, E., Hué, S., Fearnhill, E., Delpech, V., Brown, A.J., Lycett, S.: Automated analysis of phylogenetic clusters. BMC Bioinformatics **14**(1), 317 (2013). doi:10.1186/1471-2105-14-317
2. Rose, R., Lamers, S.L., Dollar, J.J., Grabowski, M.K., Hodcroft, E.B., Ragonnet-Cronin, M., Wertheim, J.O., Redd, A.D., German, D., Laeyendecker, O.: Identifying Transmission Clusters with Cluster Picker and HIV-TRACE. AIDS Research and Human Retroviruses **33**(3), 211–218 (2017). doi:10.1089/aid.2016.0205
3. Moshiri, N.: TreeCluster: Massively scalable transmission clustering using phylogenetic trees. bioRxiv (2018). doi:10.1101/261354
4. Kembel, S.W., Eisen, J.A., Pollard, K.S., Green, J.L.: The phylogenetic diversity of metagenomes. PLoS ONE **6**(8), 23214 (2011). doi:10.1371/journal.pone.0023214. arXiv:1208.5792v1
5. Darling, A.E., Jospin, G., Lowe, E., Matsen, F.A., Bik, H.M., Eisen, J.A.: PhyloSift: phylogenetic analysis of genomes and metagenomes. PeerJ **2**, 243 (2014). doi:10.7717/peerj.243
6. Filipski, A., Tamura, K., Billing-Ross, P., Murillo, O., Kumar, S.: Phylogenetic placement of metagenomic reads using the minimum evolution principle. BMC Genomics **16**(Supplement 1), 13 (2015). doi:10.1186/1471-2164-16-S1-S13
7. 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., De Hoon, M.J.L.: Biopython: Freely available Python tools for computational molecular biology and bioinformatics. Bioinformatics **25**(11), 1422–1423 (2009). doi:10.1093/bioinformatics/btp163. arXiv:1011.1669v3
8. Sukumaran, J., Holder, M.T.: DendroPy: A Python library for phylogenetic computing. Bioinformatics **26**(12), 1569–1571 (2010). doi:10.1093/bioinformatics/btq228
9. Huerta-Cepas, J., Serra, F., Bork, P.: ETE 3: Reconstruction, Analysis, and Visualization of Phylogenomic Data. Molecular Biology and Evolution **33**(6), 1635–1638 (2016). doi:10.1093/molbev/msw046
10. Phillips, M.J., Penny, D.: The root of the mammalian tree inferred from whole mitochondrial genomes. Molecular Phylogenetics and Evolution **28**(2), 171–185 (2003). doi:10.1016/S1055-7903(03)00057-5
11. Pybus, O.G., Harvey, P.H.: Testing macro-evolutionary models using incomplete molecular phylogenies. Proceedings of the Royal Society B: Biological Sciences **267**(1459), 2267–2272 (2000). doi:10.1098/rspb.2000.1278

Figures
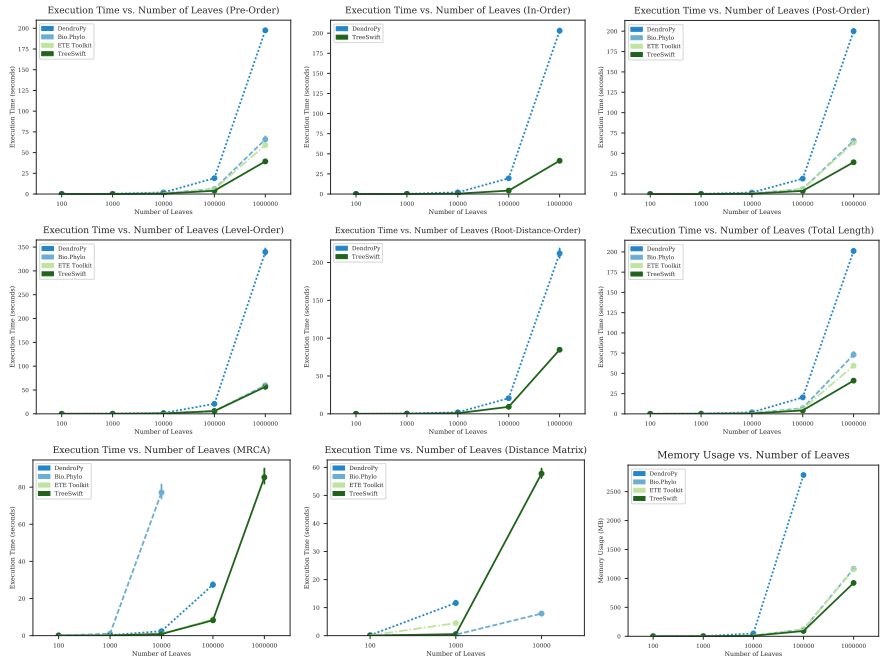
Figure 1 Tree Package Comparison. Runtimes of DendroPy, Bio.Phylo, the ETE Toolkit, and TreeSwift for a wide range of typical tree operations using trees of various sizes, as well as memory consumption after loading a tree. The truncation of a given tool's plot implies lack of scalability beyond that point (either due to time or memory consumption), and the entire lack of a given tool implies lack of implementation of the tested functionality.