Journal of X, 2017, 1–4

doi: xx.xxxx/xxxx
Manuscript in Preparation
Paper

PAPER

TreeSwift: a massively scalable Python tree package

Niema Moshiri[1,*]

[1]Bioinformatics and Systems Biology Graduate Program, UC San Diego, 92093 USA

[*]niemamoshiri@gmail.com

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. **Findings** I therefore developed TreeSwift, a Python package for traversing and manipulating trees that does not require any external dependencies. 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`. **Conclusions** TreeSwift offers a user-friendly and massively scalable option for tree parsing and manipulation that, based on benchmark comparisons against existing methods, is ideal for algorithms performed on ultra-large trees.

Key words: phylogenetics; tree; traversal; Python; scalable

Findings

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`.

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 sin-

Compiled on: August 9, 2018.
Draft manuscript prepared by the author.

gle 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.

Data Inputs

Currently, TreeSwift supports loading trees in the Newick, Nexus, or NeXML formats via the `read_tree_newick`, `read_tree_nexus`, and `read_tree_nexml` functions, respectively. Inputs to these functions can be strings, plaintext files, or gzipped files, and TreeSwift handles the nuances of parsing them internally.

Documentation

The primary TreeSwift documentation can be found as an Application Programming Interface (API) on the GitHub repository. To aid new users, a "Cookbook" of numerous examples utilizing TreeSwift can be found on the GitHub repository as well. 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.

Installation

TreeSwift can be installed on Python 2 or 3 via `pip`:

```
$ pip install treeswift
```

As TreeSwift has no external dependencies, it can readily be integrated into any computer cluster or cloud environment with a Python interpreter.

Performance

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 its children
- **Post-Order Traversal:** Visit the children of the current node 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 unweighted path length from root
- **Root-Distance-Order Traversal:** Visit the nodes of the tree in order of weighted path length from root
- **Most Recent Common Ancestor:** Finding the most recent common ancestor (MRCA) of a set of leaves
- **Distance Matrix:** 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 performance results can be seen in Figure 1.

When compared with DendroPy, TreeSwift is significantly faster for all tested functions (an order of magnitude faster for most) with the exception of performing a root-distance-order traversal, and it has a memory footprint that is orders of magnitude lower. When compared with Bio.Phylo, TreeSwift is significantly faster for all tested functions with the exception of computing a pairwise distance matrix, and it has roughly half the 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 for most tested functions with the exception of performing a pre-order or a level-order traversal, and it has roughly half the memory footprint. Also note that the ETE Toolkit does not seem to implement a way to perform an in-order nor a root-distance-order traversal.

Importantly, TreeSwift loads trees significantly faster than all other tools, and even when TreeSwift performs a task more slowly than another tool, the difference in runtime is significantly lower than the runtime difference of simply loading the tree (a task which cannot be avoided). Further, there are cases in which the other tools simply do not scale to the largest tested size; notably, when computing the most recent common ancestor, TreeSwift was the only tool able to scale to trees with 1,000,000 leaves.

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.

Conclusion

TreeSwift is an open source Python 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 and extensive documentation of the classes within TreeSwift poise the package for community usage and contribution.

Methods

For the performance tests, TreeSwift was run on a computer with CentOS release 6.6 (Final), Intel(R) Xeon(R) CPU E5-2670 0 at 2.60GHz, and 32 GB of RAM. The processor has 12 cores, but each execution of each tool runs on a single thread. Each point is the distribution resulting from 10 iterations of the test. The code used to perform each task using each tool can be found at www.github.com/niemasd/TreeSwift-Paper.

Availability of supporting data

All simulation data used to perform the benchmarking tests can be found at www.github.com/niemasd/TreeSwift-Paper.

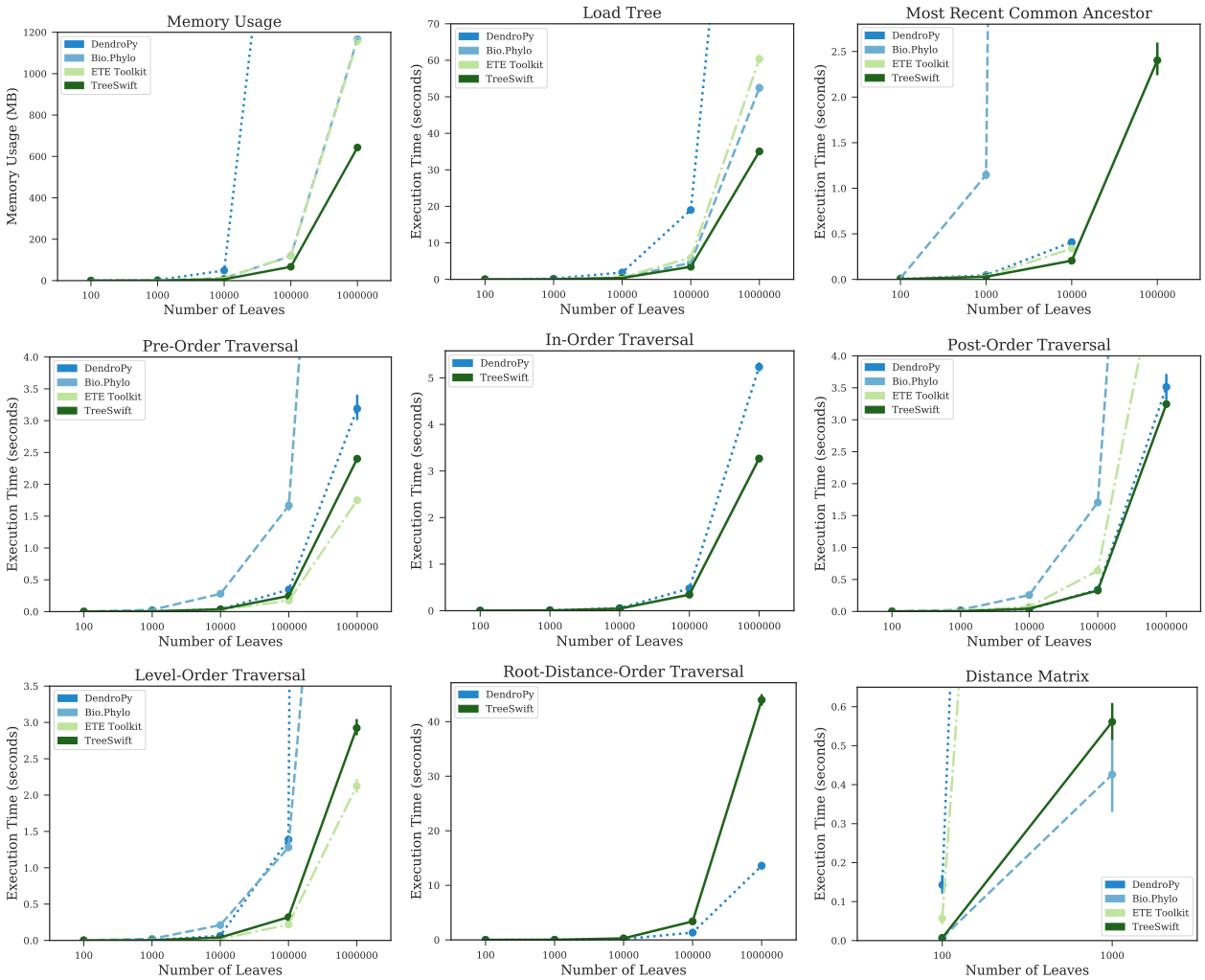

Figure 1. Runtimes of DendroPy, Bio.Phylo, the ETE Toolkit, and TreeSwift for a wide range of typical tree operations using trees of various sizes, as well as memory consumption after loading a tree. The truncation of a given tool's plot implies lack of scalability beyond that point (either due to time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

Availability of source code and requirements

- Project name: TreeSwift
- Project home page: www.github.com/niemasd/TreeSwift
- Operating system(s): Platform independent
- Programming language: Python
- Other requirements: N/A
- License: GNU General Public License v3.0

The code used to perform the performance benchmarking can be found at www.github.com/niemasd/TreeSwift-Paper

Abbreviations

- API: Application Programming Interface
- MRCA: Most Recent Common Ancestor

Competing interests

The author(s) declare that they have no competing interests.

Authors' contributions

NM performed the entirety of this work.

Funding

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

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.

References

1. Ragonnet-Cronin M, Hodcroft E, Hué S, Fearnhill E, Delpech V, Brown AJ, et al. Automated analysis of phylogenetic clusters. BMC Bioinformatics 2013;14(1):317. http://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-14-317.
2. Rose R, Lamers SL, Dollar JJ, Grabowski MK, Hodcroft EB, Ragonnet-Cronin M, et al. Identifying Transmission Clusters with Cluster Picker and HIV-TRACE. AIDS Research and Human Retroviruses 2017;33(3):211–218. http://online.liebertpub.com/doi/10.1089/aid.2016.0205.
3. Moshiri N. TreeCluster: Massively scalable transmission clustering using phylogenetic trees. bioRxiv 2018;http://dx.doi.org/10.1101/261354.
4. Kembel SW, Eisen JA, Pollard KS, Green JL. The phylogenetic diversity of metagenomes. PLoS ONE 2011;6(8):e23214. http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0023214.
5. Darling AE, Jospin G, Lowe E, Matsen FA, Bik HM, Eisen JA. PhyloSift: phylogenetic analysis of genomes and metagenomes. PeerJ 2014;2:e243. https://peerj.com/articles/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 2015;16(Supplement 1):S13. https://bmcgenomics.biomedcentral.com/articles/10.1186/1471-2164-16-S1-S13.
7. Cock PJA, Antao T, Chang JT, Chapman BA, Cox CJ, Dalke A, et al. Biopython: Freely available Python tools for computational molecular biology and bioinformatics. Bioinformatics 2009;25(11):1422–1423. https://academic.oup.com/bioinformatics/article/25/11/1422/330687.
8. Sukumaran J, Holder MT. DendroPy: A Python library for phylogenetic computing. Bioinformatics 2010;26(12):1569–1571. https://academic.oup.com/bioinformatics/article/26/12/1569/287181.
9. Huerta-Cepas J, Serra F, Bork P. ETE 3: Reconstruction, Analysis, and Visualization of Phylogenomic Data. Molecular Biology and Evolution 2016;33(6):1635–1638.
10. Phillips MJ, Penny D. The root of the mammalian tree inferred from whole mitochondrial genomes. Molecular Phylogenetics and Evolution 2003;28(2):171–185.
11. Pybus OG, Harvey PH. Testing macro-evolutionary models using incomplete molecular phylogenies. Proceedings of the Royal Society B: Biological Sciences 2000;267(1459):2267–2272. http://rspb.royalsocietypublishing.org/content/267/1459/2267.