Fastsubtrees: simple and efficient subtrees extractions in Python with applications to NCBI taxonomy

Aman Modi[1*] **and Giorgio Gonnella**[1,2*¶]

1 Department of Bioinformatics (IMG), University of Göttingen, Göttingen, Germany **2** Center for Bioinformatics (ZBH), University of Hamburg, Hamburg, Germany ¶ Corresponding author * These authors contributed equally.

DOI: 10.21105/joss.04755

Software

- Review
- Repository
- Archive

Editor: Kelly Rowland

Reviewers:

- @vinisalazar
- @KonradHoeffner

Submitted: 08 July 2022
Published: 13 November 2022

Summary

Tree data structures are commonly used for representing hierarchical data. A prominent example in bioinformatics are taxonomic trees, representing the biological classification of organisms, currently containing millions of nodes. An interesting operation for such trees is the extraction of the IDs and other associated values for each node in a given subtree. Here we present *fastsubtrees*, a Python package which provides a simple and efficient way to perform this operation, by storing the tree data in a suitable representation. The package provides a command line interface and a application programming interface. While the software is written in a generic way and can be applied to other trees as well, it is mainly aimed at working with the NCBI taxonomy tree.

Statement of need

Tree data structures are commonly used in different fields of computer science to represent hierarchical information (Knuth, 1997). In phylogenetics, trees are used to represent the common ancestry of organisms or macromolecules. Based on phylogenetics is the hierarchical biological classification of organisms, which is also representable as a tree. An example of this is the NCBI taxonomy database (Schoch et al., 2020).

If the taxonomic tree is annotated with data associated to taxa, hereafter called *attributes* (e.g. sequence or annotation statistics), an interesting operation is the extraction of the values distribution of an attribute in a given subtree, or in multiple subtrees, which are compared to each other. This, for instance, allows for the identification of uncommon values in some members of a taxon, which in turn can be used in a primer for new biological hypotheses.

No package currently allows for a simple and efficient extraction of a subtree, including data associated with the subtree nodes. While software exists for visualizing data associated with phylogenetic or taxonomic subtrees, e.g. AnnoTree (Mendler et al., 2019) and Treehouse (Steenwyk & Rokas, 2019), these are standalone software not simple to integrate in other applications.

The fastsubtrees Python package

Here we present the *fastsubtrees* Python package which was developed for storing tree information using a compact representation, suitable for fast extraction of any subtree. This package can be applied to any tree in which the nodes are labeled by non-negative integers. Although it was designed and tested mainly for the NCBI taxonomy tree, it is thus not limited to it.

It can be installed using `pip install fastsubtrees`. The package functionality can be used through either a command line script or an API, which can be employed from other Python programs. Manuals are provided for both kind of interfaces. A complete test suite based on *pytest* is provided. Running the complete tests, comparative benchmarks and the example application (see below) requires a database installation and setup; however, a Docker image is provided, which allows to simply run benchmarks, tests and the example application without this hassle.

The first step when using *fastsubtrees* is constructing its tree representation and storing it to file. Any source can be used as input (e.g. tabular files, or database tables) by defining a generation function (examples are provided). A separate wrapper package *ntsubtree*, installable by `pip install ntsubtree`, is provided (version 1.1; directory `ntsubtree` of the source code repository), which simplifies working with the NCBI taxonomy tree. Besides the NCBI taxonomy tree, other trees can be represented using *fastsubtrees*. For this, each node must be labeled by a unique non-negative integer ID. Furthermore, the IDs space must be compact, i.e. the numbers must not be necessarily all consecutive, but the largest node ID (*idmax*) should not be much larger than the total number of nodes, because the memory consumption is in *O(idmax)*.

In the tree representation, the node numbers and the subtree sizes under each node are stored in depth-first traversal order. Furthermore, an index table is provided which gives the ordinal position of each node in the depth-first representation. Finally, any number of numerical or string attributes can be associated to the nodes: these are stored in separate files, in the same order as the node identifiers. Thus, from the tree representation, any node and its associated data can be found in constant time and the subtree under a node can be output in time proportional to the subtree size. The representation is dynamic, i.e. it is possible to add and remove subtrees.

Subtree extraction benchmarks

Benchmarks were performed on a MacBook Pro 2021 with an Apple M1 Pro CPU and 32 GB RAM. Running times were measured as an average of 3 runs using GNU time version 1.9 (Gordon, 2018). Thereby, CPU time was computed as the sum of user and system time. For the tests Python version 3.10.2 was used. The tested version of *fastsubtrees* was 2.1.

The NCBI taxonomy tree used for the tests was downloaded on October 7, 2022 from the NCBI FTP website (NCBI, 2022). For downloading and keeping up-to-date a copy of the dump files using Python we developed the *ntdownload* package (version 1.7; directory `ntdownload` of the source code repository) installable using `pip install ntdownload`. The tree contained 2447574 nodes. The generation of the tree representation of the NCBI taxonomy tree from the dump files using the *fastsubtrees construct* command required 12.5 seconds (average of 3 runs).

An alternative to the use of *fastsubtrees* is to store the tree data in a SQL database and extract subtrees using hierarchical SQL queries. We implemented this solution in a package *ntmirror* (version 1.4; directory `ntmirror` of the repository). First, the dump data downloaded by *ntdownload* is loaded into a MariaDB database (version 10.6.10) using the script *ntmirror-dbload*, requiring 28.8 seconds (average of 3 runs). Thereafter, subtrees are extracted using the script *ntmirror-extract-subtree*, based on SQLAlchemy, implementing a hierarchical SQL query. The installation of *ntmirror* using `pip install ntmirror` requires the installation of MariaDB and its Python connector (*mariadb* package).

To select subtrees of different sizes for the benchmarks, we started from the taxonomy ID of *Escherichia coli* K12 MG1655 (511145) and climbed up the taxonomy tree, including all parent nodes in the benchmarks, up to the Bacteria node (TaxID 2), i.e. nodes 83333 (*Escherichia coli* K12), 562 (*Escherichia coli*), 561 (*Escherichia* genus), 543 (Enterobacteriaceae), 91347 (Enterobacterales), 1236 (Gammaproteobacteria) and 1224 (Proteobacteria). The running time and memory usage of *fastsubtrees* are compared with those for hierarchical SQL queries in Table 1, and show that *fastsubtrees* scales better to the extraction of large subtrees, both in terms of memory consumption (which does not change) and running time.

Subtree root ID	Subtree size	SQL CPU time (s)	*fastsubtrees* CPU time (s)	SQL real time (s)	*fastsubtrees* real time (s)	SQL memory peak (MB)	*fastsubtrees* memory peak (MB)
setup		0.57	12.47	28.83	12.48	40.7	153.7
511145	1	0.26	0.16	1.58	0.16	41.4	153.1
83333	36	0.27	0.16	2.55	0.16	41.4	153.1
562	3381	0.29	0.15	4.64	0.16	48.1	153.1
561	4436	0.29	0.15	5.73	0.16	50.4	153.1
543	22750	0.46	0.16	34.26	0.17	89.1	153.1
91347	31609	0.54	0.17	35.14	0.18	106.3	153.1
1236	123300	1.49	0.21	58.23	0.21	294.8	153.1
1224	228153	2.55	0.26	86.53	0.26	511.8	153.1
2	535272	5.59	0.39	126.48	0.40	1142.0	153.1

As an example of attributes associated to tree nodes, we computed guanine-cytosine (GC) content and genome size for each bacterial genome in the NCBI Refseq database (O'Leary et al., 2015). The results, available in the repository, contain values for 27967 genomes. The genome size attribute file generation, using the *fastsubtrees attribute* command, required 2.9 seconds. Table 2 reports the running time and memory usage for the extraction of the genome size attribute values for different subtrees.

Subtree root ID	Subtree size	CPU time (s)	Real time (s)	Memory peak (MB)	N. nodes with values	N. values
511145	1	0.47	0.47	153.1	1	9
83333	36	0.47	0.47	153.1	8	38
562	3381	0.52	0.52	153.5	165	2160
561	4436	0.50	0.50	153.5	174	2246
543	22750	0.52	0.52	153.6	839	5774
91347	31609	0.52	0.53	153.7	1150	6709
1236	123300	0.63	0.64	154.0	2830	11178
1224	228153	0.75	0.75	154.2	5099	16072
2	535272	1.08	1.08	155.2	10043	27515

Parallelizing the tree construction algorithm

As seen in the previous section, tree construction is slower than the subtree queries, thus it is interesting to analyze this operation in detail.

The tree representation for fast subtree queries requires the following arrays: *T*, node IDs in depth-first traversal order; *P*, parent of each node, in order of ID; *S*, subtree size of each node, in order of ID; *C*, coordinate in *T* of each node, in order of ID.

The construction algorithm consists in the following 3 operations:

1. *P* construction: initialization with *undef* values; iteration over the input data, 2-tuples *(node, parent)* storing *parent* at position *node* in *P*.
2. *S* construction: for each node, climb to the root, using the values in *P* and increment the subtree size of each ancestor.

3. C and T construction: first add the tree root data; then for each node, add the node to a stack and climb the tree, adding the ancestors to the stack until an ancestor which was already added to the tree is found; the first free position after this node in T is located using C the data is stored in T and C is updated; this is repeated until the stack is empty.

Given n nodes with a maximum ID m (not much larger than n), and a tree height h (in the worst case $h = m$, however, in general it is much smaller), the construction operations require the following time. Operation 1 requires iterating over all $O(n)$ input tuples and initializing P requires $O(m)$. Thus the total time is in $O(m)$. Operation 2 requires climbing the tree from each node, thus $O(h)$ time for each node, in total $O(n * h)$. Operation 3 requires climbing the tree again, but this is stopped whenever nodes are found which were already added; thus the total time is in this case $O(n)$.

Since the first and third operations are performed in linear time, they are faster than the second. Benchmarks showed that this is indeed the bottleneck of the construction. Thus an attempt was made to speed up this operation, by parallelizing it. For this, the node IDs are divided into x slices, each assigned to a different sub-process (not thread, because of the Python Global Interpreter Lock). Each sub-process then counts nodes in each subtree belonging to the slice only. These counts could be stored in a shared subtree sizes table, however this requires a semaphore, and it was very slow in our tests. Thus, we implemented a version, in which each sub-process stores its counts in a local table. The results for each sub-process are summed up after completion, to obtain the S table.

Despite the distribution of the work among processes, benchmarks did not show a significant performance improvement in the parallel version for the construction of the NCBI taxonomy tree, likely because of the overhead of subprocess starting, data initialization and results consolidation.

Example application: Genomes Attributes Viewer

To provide an example of usage of *fastsubtrees* we implemented an interactive web application, Genomes Attributes Viewer, based on the *dash* library version 2.0.0 ("Dash Python User Guide," 2022) and installable by `pip install genomes-attributes-viewer`. The application (version 1.3; directory *genomes_attributes_viewer* of the repository) allows displaying diagrams of the value distribution of GC content and genome size values in any subtree of the NCBI taxonomy tree (we have included in the example only values for bacterial genomes, see above). Attribute value distributions for multiple subtrees can be thereby compared graphically.

Author Contributions

Aman Modi: software development, test suite, example application, documentation and benchmarks.

Giorgio Gonnella: funding acquisition, conceptualization, software development, test suite, documentation, Docker image, benchmarks, project supervision and manuscript redaction.

Funding

Giorgio Gonnella has been supported by the DFG Grant GO 3192/1-1 ''Automated characterization of microbial genomes and metagenomes by collection and verification of association rules''. The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Acknowledgements

The authors would like to thank Serena Lam (Department of Bioinformatics, University of Göttingen) for language style suggestions and grammar corrections.

References

Dash Python user guide. (2022). In *Dash Project Website*. plotly. https://dash.plotly.com/

Gordon, A. (2018). GNU time. In *GNU Foundation Website*. GNU Foundation. https://www.gnu.org/software/time/

Knuth, D. E. (1997). *The art of computer programming, volume I: Fundamental algorithms, 3rd edition* (p. 308). Addison-Wesley. ISBN: 0-201-89683-4

Mendler, K., Chen, H., Parks, D. H., Lobb, B., Hug, L. A., & Doxey, A. C. (2019). AnnoTree: visualization and exploration of a functionally annotated microbial tree of life. *Nucleic Acids Research*, *47*(9), 4442–4448. https://doi.org/10.1093/nar/gkz246

NCBI. (2022). NCBI taxonomy FTP site. In *NCBI*. NCBI. https://ftp.ncbi.nlm.nih.gov/pub/taxonomy/

O'Leary, N. A., Wright, M. W., Brister, J. R., Ciufo, S., Haddad, D., McVeigh, R., Rajput, B., Robbertse, B., Smith-White, B., Ako-Adjei, D., Astashyn, A., Badretdin, A., Bao, Y., Blinkova, O., Brover, V., Chetvernin, V., Choi, J., Cox, E., Ermolaeva, O., … Pruitt, K. D. (2015). Reference sequence (RefSeq) database at NCBI: Current status, taxonomic expansion, and functional annotation. *Nucleic Acids Res*, *44*(D1), D733–45.

Schoch, C. L., Ciufo, S., Domrachev, M., Hotton, C. L., Kannan, S., Khovanskaya, R., Leipe, D., Mcveigh, R., O'Neill, K., Robbertse, B., Sharma, S., Soussov, V., Sullivan, J. P., Sun, L., Turner, S., & Karsch-Mizrachi, I. (2020). NCBI taxonomy: A comprehensive update on curation, resources and tools. *Database (Oxford)*, *2020*. https://doi.org/10.1093/database/baaa062

Steenwyk, J. L., & Rokas, A. (2019). Treehouse: A user-friendly application to obtain subtrees from large phylogenies. *BMC Research Notes*, *12*(1), 541. https://doi.org/10.1186/s13104-019-4577-5