Genesis and Gappa: Processing, Analyzing and Visualizing Phylogenetic (Placement) Data.

Lucas Czech[a,*], **Pierre Barbera**[a], **and Alexandros Stamatakis**[a,b,*]

[a]Computational Molecular Evolution Group, Heidelberg Institute for Theoretical Studies, Heidelberg, Germany; [b]Institute for Theoretical Informatics, Karlsruhe Institute of Technology, Karlsruhe, Germany; [*]To whom correspondence should be addressed. E-mail: lucas.czech@h-its.org and alexandros.stamatakis@h-its.org

The ever increasing amount of genomic and meta-genomic sequence data has transformed biology into a data-driven and compute-intensive discipline. Hence, there is a need for efficient algorithms and scalable implementations thereof for analysing such data.

We present GENESIS, a library for working with phylogenetic data, and GAPPA, an accompanying command line tool for conducting typical analyses on such data. While our tools primarily target phylogenetic trees and phylogenetic placements, they also offer a plethora of functions for handling genetic sequences, taxonomies, and other relevant data types.

The tools aim at improved usability at the production stage (conducting data analyses) as well as the development stage (rapid prototyping): The modular interface of GENESIS simplifies numerous standard high-level tasks and analyses, while allowing for low-level customization at the same time. Our implementation relies on modern, multi-threaded C++11, and is substantially more computationally efficient than analogous tools. We already employed the core GENESIS library in several of our tools and publications, thereby proving its flexibility and utility. GENESIS and GAPPA are freely available under GPLv3 at http://github.com/lczech/genesis and http://github.com/lczech/gappa.

Metagenomic Analysis Tool | Data Visualization | Phylogenetic Trees | Phylogenetic Placement | Evolutionary Placement | Software | Library

1. Introduction

When developing scientific software, there are several important, yet often times competing design objectives: (a) Most users require a fast and simple application for analysing their data, (b) some power users desire customization, e. g., via scripting etc., and (c) developers require a flexible toolkit for rapid prototyping. At the same time, with the on-going data growth, the implementation needs to be scalable and efficient with respect to both, memory, and execution times. The tools we present here aim to meet all of the above objectives. To this end, GENESIS and GAPPA are written in C++11, using a modern, modular, and function-centric software design. We evaluate the code quality, the runtime behavior, and the memory requirements for conducting typical tasks such as file parsing and data processing in the Supplementary data. We find that GENESIS has the overall best code quality score (1) compared to other scientific codes written in C or C++. It is also consistently faster than all evaluated Python and R libraries in our tests.

2. Features of Genesis

GENESIS is a highly flexible library for reading, manipulating, and evaluating phylogenetic data, and in particular phylogenetic placement data. It has a simple and straight forward API, but is also computationally highly efficient. Typical tasks such as parsing and writing files, iterating over the elements of a data structure, and other frequently used functions are mostly one-liners that integrate well with modern C++ and its standard library STL. Where possible, the library is multi-threaded, allowing for fully leveraging the computational power of multi-core systems. Hence, GENESIS allows for scalable parsing and processing of huge datasets. The functionality is divided into loosely coupled modules, which we briefly describe in the following.

Phylogenetic Trees. Phylogenetic trees are implemented via a data structure that enables fast and flexible operations, and allows to store arbitrary (meta-)data at the nodes *and* at the edges. The trees may contain multifurcations and may have a designated root node. Trees can be parsed from newick files and be written to newick, phyloxml, and nexus files, again with support for arbitrary edge and node annotations. Traversing the tree starting from an arbitrary node in, e. g., post-order, pre-order, or level-order can be accomplished via simple for loops:

```
// Read a tree from a Newick file.
Tree tree = CommonTreeNewickReader().read(
    from_file( "path/to/tree.newick" )
);

// Traverse tree in preorder, print node names.
for( auto it : preorder( tree )) {
    auto& data = it.node().data<CommonNodeData>();
    std::cout << data.name << "\n";
}
```

Functionality for manipulating trees, finding lowest common ancestors of nodes or paths between nodes, calculating distances between nodes, testing for monophyly of clades, and many other standard tasks is provided. Furthermore, functions for drawing rectangular and circular phylograms or cladograms to svg files, using individual custom edge colors and node shapes, are provided for creating publication quality figures.

Phylogenetic Placements. Phylogenetic (or evolutionary) placement is a method for classifying large numbers of (meta-)genomic sequences with respect to a given reference phylogeny (2, 3). Functions for analyzing the resulting placement data constitute a primary focus of GENESIS. It offers low-level functions for filtering, merging, and otherwise manipulating placement data, as well as high-level functions such as distance calculations (4), Edge PCA and Squash Clustering (5), and Phylogenetic *k*-means (6), among others. Advanced functions for analysing and visualizing the data are implemented as well, for instance, our adaptation of Phylofactorization to phylogenetic placement data (6, 7).

Other Features. Sequences and alignments can be efficiently read from and written to `fasta` and `phylip` files; in our tests, we achieved reading rates close to 400 MB/s. High-level functions for managing sequences include several methods for calculating consensus sequences, the entropy of sequence sets, and sequence re-labelling via hashes (e. g., `MD5`, `SHA1`, `SHA256`).

Taxonomies and taxonomic paths (for example, "`Eukaryota;Alveolata;Apicomplexa`") can be parsed from databases such as Silva (8, 9) or NCBI (10, 11) and stored in a hierarchical taxonomic data structure, again with the ability to store arbitrary meta-data at each taxon, and to traverse the taxonomy.

Furthermore, GENESIS supports several standard file formats such as `json`, `csv`, and `svg`. All input methods (including trees and sequence files) automatically and transparently handle `gzip`-compressed files. Moreover, a multitude of auxiliary functions and classes is provided: Matrices and dataframes to store data, histograms and statistics functions to examine such data, regression via Generalized Linear Models, color support for handling gradients in plots, etc. The full list of functionality is available via the online documentation.

Lastly, GENESIS offers a simple architecture for scripting-like development of rapid prototypes or small custom programs (e. g., convert some files or examine some data for a particular experiment).

3. Features of Gappa

The flexibility of a library such as GENESIS is primarily useful for method developers. For most users, it is however more convenient to offer a simple interface for typical, frequent tasks. To this end, we have developed the command line program GAPPA, which we present in the following.

GAPPA is short for "Genesis Applications for Phylogenetic Placement Analyses". Its original purpose was to implement and make available the methods we presented in (6) and (12). For example, it offers a method for automatically obtaining a set of reference sequences from large databases (12), which can be used to infer a reference tree for subsequent phylogenetic placement. Furthermore, it provides a set of visualization tools to display the distribution of placements on the tree, as well as visualize per-branch correlations with meta-data features of environmental samples (6). Lastly, it offers quantitative analysis methods such as Phylogenetic k-means and Placement-Factorization for environmental samples (6).

GAPPA has since been substantially extended and now also contains re-implementations of some prominent methods of GUPPY (3), as well as commands for sanitizing, filtering, and manipulating files formats such as `jplace`, `newick`, or `fasta`. It can also calculate the taxonomic assignments for a set of phylogenetic placements (previously implemented in the SATIVA pipeline (13)).

The GAPPA commands are organized into functional modules for clarity, using a consistent command interface. GAPPA can also be considered as a collection of demo programs using GENESIS, which might be helpful as a starting point for developers who intend to use our library.

As GAPPA internally relies on GENESIS, it is also efficient and scalable. In comparison to GUPPY (3), we have observed speedups of several orders of magnitude when processing large data volumes (12). Furthermore, the memory requirements are generally about 50% lower for GAPPA compared to GUPPY.

4. Conclusion

We presented GENESIS, a library for working with phylogenetic (placement) data and related data types, as well as GAPPA, a command line interface for analysis methods and common tasks related to phylogenetic placements. GENESIS and GAPPA are already used as an integral part in several of our previous publications and programs (2, 6, 12, 14, 15), proving their flexibility and usefulness.

In future GENESIS releases, we intend to offer API bindings to `Python`, thus making the functions of the library more accessible to developers. In GAPPA, we will implement additional commands, in particular for working with phylogenetic placements, as well as re-implement the remaining commands of GUPPY, in order to facilitate analysis of larger data sets.

Both GENESIS and GAPPA are freely available under GPLv3 at http://github.com/lczech/genesis and http://github.com/lczech/gappa. We maintain a Google Group for discussing phylogenetic placement and our tools at https://groups.google.com/forum/#!forum/phylogenetic-placement.

Appendices

Competing Interests. The authors declare that they have no competing interests.

Author's Contributions. LC designed and implemented GENESIS. LC and PB implemented GAPPA. All authors helped to draft the manuscript. All authors read and approved the final manuscript.

Acknowledgments. This work was financially supported by the Klaus Tschira Stiftung gGmbH Foundation in Heidelberg, Germany. We thank all members of the Exelixis Lab for their help and suggestions. The background music for this work was provided by Peter Gabriel and Phil Collins.

References

1. Zapletal A, Stamatakis A (2019) softwipe. Online: https://github.com/adrianzap/softwipe. Accessed: 2019-05-20.
2. Barbera P, et al. (2018) EPA-ng: Massively Parallel Evolutionary Placement of Genetic Sequences. *Systematic Biology.*
3. Matsen FA, Kodner RB, Armbrust EV (2010) pplacer: linear time maximum-likelihood and Bayesian phylogenetic placement of sequences onto a fixed reference tree. *BMC Bioinformatics* 11(1):538.
4. Evans SN, Matsen FA (2012) The phylogenetic Kantorovich-Rubinstein metric for environmental sequence samples. *Journal of the Royal Statistical Society. Series B: Statistical Methodology* 74:569–592.
5. Matsen FA, Evans SN (2011) Edge principal components and squash clustering: using the special structure of phylogenetic placement data for sample comparison. *PLOS ONE* 8(3):1–17.
6. Czech L, Stamatakis A (2019) Scalable methods for analyzing and visualizing phylogenetic placement of metagenomic samples. *bioRxiv.*
7. Washburne AD, et al. (2017) Phylogenetic factorization of compositional data yields lineage-level associations in microbiome datasets. *PeerJ* 5:e2969.
8. Quast C, et al. (2013) The SILVA ribosomal RNA gene database project: improved data processing and web-based tools. *Nucleic Acids Research* 41(D1):D590–D596.
9. Yilmaz P, et al. (2014) The SILVA and "All-species Living Tree Project (LTP)" taxonomic frameworks. *Nucleic Acids Research* 42(D1):D643–D648.
10. Sayers EW, et al. (2009) Database resources of the National Center for Biotechnology Information. *Nucleic Acids Research* 37(Database):D5–D15.
11. Benson DA, Karsch-Mizrachi I, Lipman DJ, Ostell J, Sayers EW (2009) GenBank. *Nucleic Acids Research* 37(Database):D26–D31.
12. Czech L, Barbera P, Stamatakis A (2018) Methods for automatic reference trees and multilevel phylogenetic placement. *Bioinformatics* 35(7):1151–1158.
13. Kozlov AM, Zhang J, Yilmaz P, Glöckner FO, Stamatakis A (2016) Phylogeny-aware identification and correction of taxonomically mislabeled sequences. *Nucleic Acids Research* 44(11):5022–5033.
14. Mahé F, et al. (2017) Parasites dominate hyperdiverse soil protist communities in Neotropical rainforests. *Nature Ecology & Evolution* 1(4):0091.
15. Zhou X, et al. (2017) Quartet-based computations of internode certainty provide accurate and robust measures of phylogenetic incongruence. *bioRxiv.*

Supplementary Text: Software Comparison

Genesis and Gappa: Processing, Analyzing and Visualizing Phylogenetic (Placement) Data.

Lucas Czech, Pierre Barbera, and Alexandros Stamatakis

In this supplement, we compare GENESIS to competing software. First, we evaluate its runtime and memory requirements for some representative and frequent tasks, by comparing it to libraries that have similar scope and functionality. Second, we evaluate the quality of the code in comparison to other scientific software written in C++ and C.

1 Runtime and Memory

We compare GENESIS to several competing libraries and tools. We focus on the most widely used alternative tools. The full list of competing software to which we compared GENESIS is provided in Table S1. Most general-purpose libraries for bioinformatics tasks, manipulation and (post-)analysis of genetic sequences and phylogenetic trees are written in interpreted languages such as Python or R. In order to also compare GENESIS to software written in a similar, compiled, language, we also included other libraries developed by our lab, namely LIBPLL [1] and the accompanying PLL-MODULES [2]. They are written in C, and form the core of the recent re-implementation of RAxML-NG [3].

We compare several typical tasks for working with sequences, trees, and phylogenetic placements. As the specific scope of each library and tool differs, there is comparatively little overlap in functionality that can be used for benchmarking. Hence, we evaluated the runtime and memory requirements for the following general tasks:

- Parsing files in fasta format [4]; see Figure S1.
- Parsing files in phylip format [5]; see Figure S2.
- Calculating base frequencies on sets of sequences; see Figure S3.
- Parsing files in newick format [6]; see Figure S4.
- Calculating the pairwise patristic distance matrix on a given tree with branch lengths; see Figure S5.
- Parsing files in jplace format [7]; see Figure S6.

The tests were run on a 4-core laptop with 12 GB of main memory. We note that GENESIS offers parallel processing and computation. It can, for example read/parse multiple files simultaneously on distinct compute cores. We are not aware that any of the competing libraries and tools evaluated here transparently offer this feature. Hence, for a fair comparison, we also tested GENESIS on a single core only.

In summary, GENESIS outperforms *all* tools written in Python and R. In most cases it is also more memory-efficient. The runtime and memory requirements for LIBPLL/PLL-MODULES in comparison to GENESIS show that each of the tools performs better at specific tasks, depending on how much effort was invested into the optimization of the respective method. Based on a careful code inspection, we believe that the tasks were each software performs best are implemented close to optimal regarding runtime and/or memory. We provide all scripts used for testing the tools and creating the plots shown here at https://github.com/lczech/genesis-gappa-paper.

2 Code Quality

Code quality is an important property of scientific software, as 'good' code quality facilitates detecting bugs and ensures long-term maintainability of the software; see [8] for a recent analysis and discussion on the state of software for evolutionary biology. We used the prototype implementation of SOFTWIPE (https://github.com/adrianzap/softwipe; also developed in our lab) to assess the relative code quality of GENESIS. SOFTWIPE is a meta-tool for C and C++ software that employs other tools such as CLANG-TIDY and CPPCHECK to obtain scores for a number of different code quality characteristics/indicators: It checks for compiler warnings, memory leaks, undefined behaviour, usage of assertions, cyclomatic complexity (modularity of the code), code duplication, etc.

The result of SOFTWIPE is a ranking of the tested codes from best to worst, scoring each code relative to the others for each code quality indicator, as well as an average "overall" ranking (average over all code quality indicators). At the time of writing this paper, 15 different software codes in C and/or C++ formed part of SOFTWIPE code quality benchmark, including highly cited tools such as INDELible [9], MAFFT [10], MrBayes [11], T-COFFEE [12], and Seq-Gen [13].

Overall, GENESIS currently achieves the highest score of all tested codes, with 9.6/10 points, with top scores (10/10) in compiler and sanitizer warnings, usage of assertions, and (low) cyclomatic complexity. See https://github.com/adrianzap/softwipe/wiki for details and the up-to-date benchmark results.

Table S1: Evaluated libraries and tools for the runtime and memory tests. The table lists the respective programming language, the version we used in our evaluation, the main reference(s), and the accumulated number of citations, using Google Scholar (https://scholar.google.com), accessed on 2019-05-05.

Tool	Language	Version	Reference	Citations
GENESIS	C++	0.22.1	[this article]	-
APE	R	5.3	[14, 15]	7390
BIOPYTHON	Python	1.73	[16]	1712
DENDROPY	Python	4.4.0	[17]	935
ETE3	Python	3.1.1	[18]	644
GGTREE	R	1.10.5	[19]	294
GUPPY	OCaml	1.1.a19	[20]	476[1]
LIBPLL	C	2019-04-13	[1, 2]	-
SCIKIT-BIO	Python	0.5.5	[21]	-

[1] GUPPY is part of the PPLACER suite of programs; the number of citations hence also includes those of PPLACER itself.

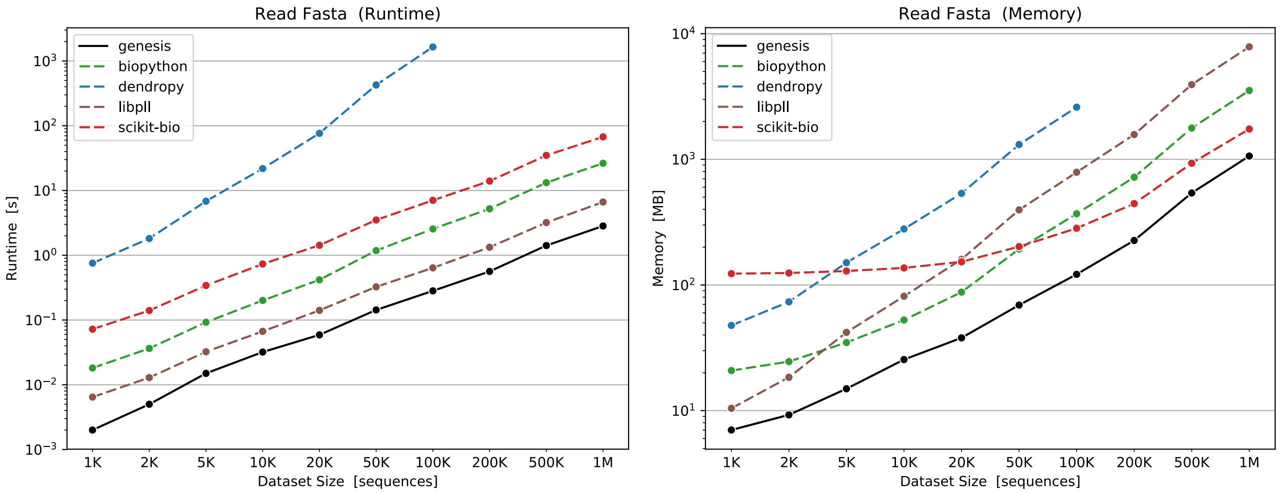

Figure S1: Runtimes and memory footprints for reading **fasta** files [4] containing 1 K to 1 M sequences. The input files consist of randomly generated sequences using the alphabet '**-ACGT**', with a length of 1000 characters per sequence. Hence, the largest file is about 1 GB in size.
In this test, GENESIS is by far the fastest and most memory-efficient software. In all cases, it parses files more than two times faster than the second fastest software, LIBPLL. Furthermore, the memory usage of LIBPLL for **fasta** files seems to be problematic: It uses about 8 times as much memory as the file size, for example, 7.8 GB for the 1 GB file. On the other hand, GENESIS has almost no overhead for keeping the files in memory, as can be seen by the 10^3 MB ≈ 1 GB memory usage for 1 M sequences. Asymptotically, the memory usage of SCIKIT-BIO is the second best after GENESIS: It starts at around 120 MB, probably due to constant memory for its **Python** environment, but then levels out for larger files. However, SCIKIT-BIO is also one of the slowest tools, and at least 20 times slower than GENESIS in all cases.
Note that we did not evaluate DENDROPY for files larger than 100 K sequences (corresponding to a 100 MB file), because the runtime of 30 min for that file is already impractical for reasonable usage.

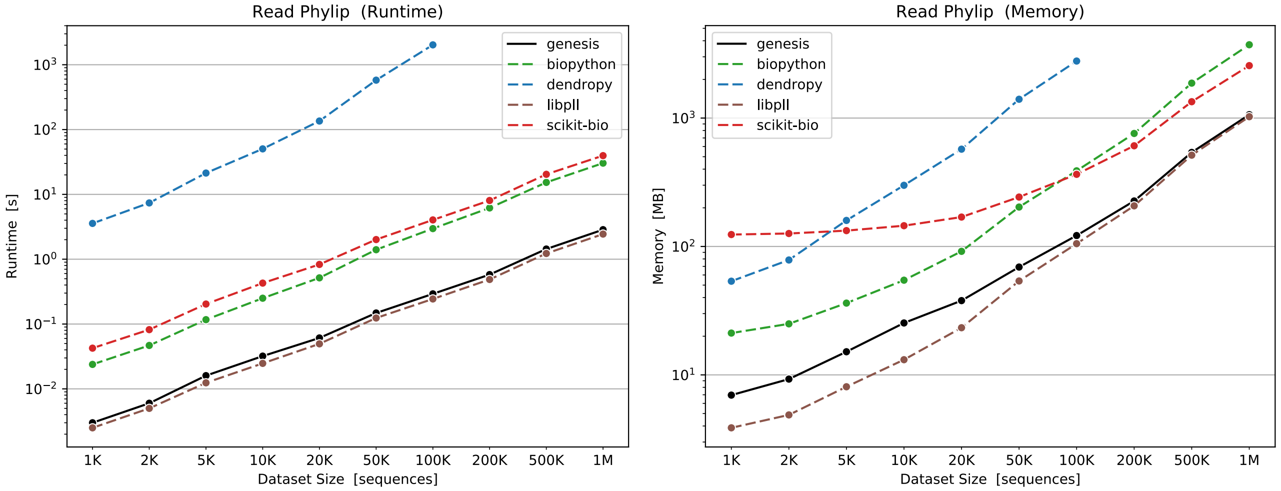

Figure S2: Runtimes and memory footprints for reading **phylip** files [5] containing 1 K to 1 M sequences. The sequences are the same as in Figure S1, but converted to a strict **phylip** format. Strict **phylip** requires exactly 10 characters for sequence names, and does not allow line breaks in the sequences. This was required by some of the tools, which can not handle "relaxed" **phylip** containing longer sequence names and/or line breaks. Although GENESIS supports all **phylip** variants, we generally advise against using the **phylip** format for maximum portability across tools.
In this test, GENESIS is second best after LIBPLL in both runtime and memory usage, but only by a small margin. Using the **phylip** format, LIBPLL also does not exhibit the memory issue that it shows for **fasta**, c. f. Figure S1. We again limited the test of DENDROPY to reasonable runtimes, as explained in Figure S1.

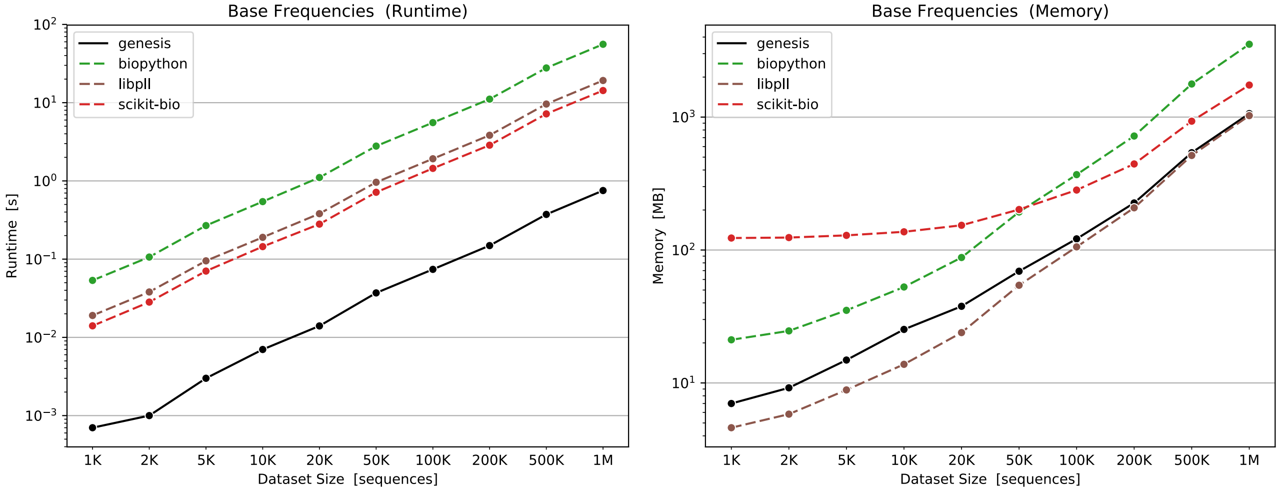

Figure S3: Runtimes and memory footprints for calculating base frequencies on a given set of sequences. Apart from merely reading and parsing files, we also tested some fundamental functionality for working with sequence data. Unfortunately, the range of functions offered by each software largely differs, and we did not find any function or procedure that is provided by all of them. We therefore evaluate the simple calculation of character frequencies/occurences in the sequences of Figure S1 and Figure S2. The parsing of the sequences is *not* included in the time measurement. We only include results for those libraries that do offer a function for counting characters in either single sequences or in an entire set of sequences.

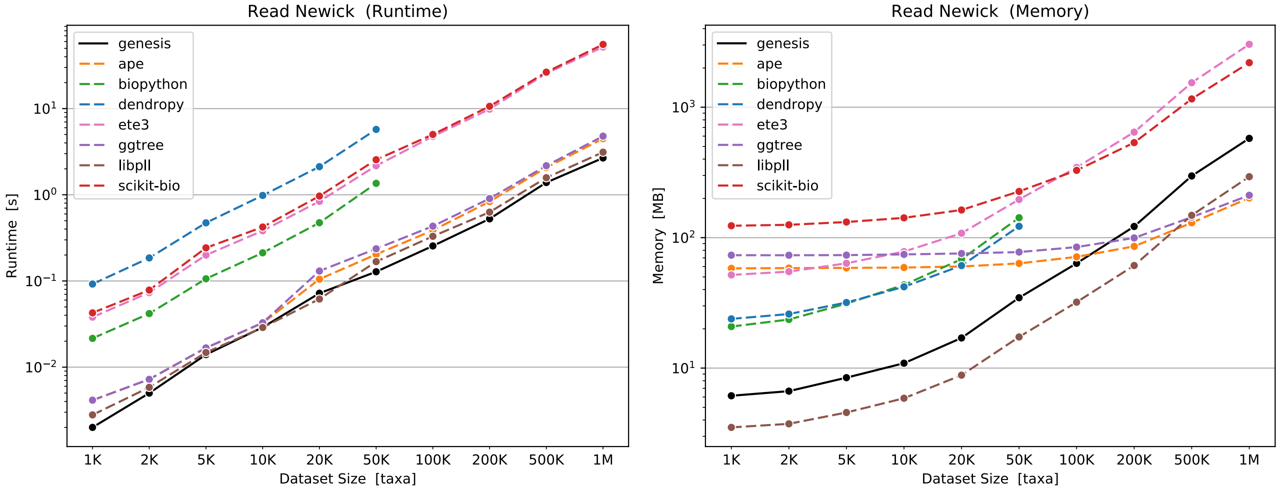

Figure S4: Runtimes and memory footprints for reading `newick` files [6] with 1 K to 1 M taxa (tip/leaf nodes) and a randomly generated topology. The `newick` format is the only one supported by all libraries that we tested. In most cases, GENESIS is fastest, closely followed by LIBPLL, which does however exhibit a more efficient memory usage. The memory usage for large trees is best for the `R`-based libraries APE and GGTREE, as they store most of the tree data in a memory-efficient table (instead of per-node storage that most other tools use). ETE3 exhibits the worst memory efficiency, with 3 GB for a 25 MB `newick` file containing a tree with 1 M taxa. We did not include measurements of DendroPy and Biopython for trees above 50 K taxa. These tools were able to *read* these trees, but not *work* with them: They store and traverse trees recursively, so that even a simple counting of the number of taxa in the tree exceeded the `Python` recursion depth, essentially rendering the tools inapplicable to such large trees.

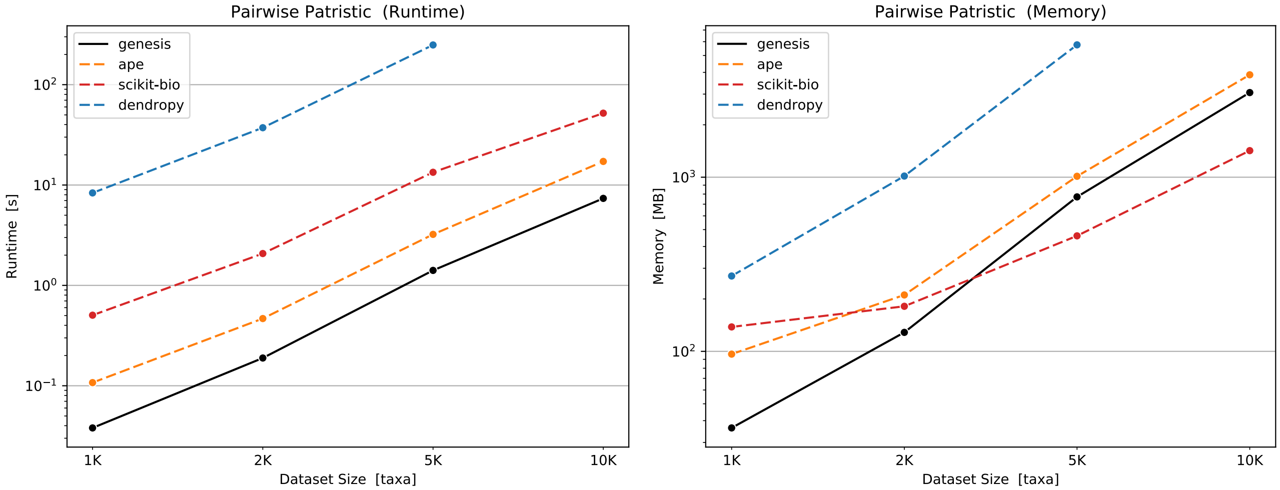

Figure S5: Runtimes and memory footprints for calculating the pairwise patristic distances on a given tree. Similar to Figure S3, we evaluate a simple function that operates on trees: The calculation of the pairwise patristic (branch length) distance matrix between all taxa of the tree. Unfortunately, we could again not identify a basic function that is available in all libraries. For example, ETE3 does have a function for calculating distances between two given taxa—but applying this to all pairs of taxa to obtain the full matrix was prohibitively slow, so we did not include it here. The tree parsing times are *not* included in the time measurements.
Again, GENESIS outperforms all other libraries. We note that GENESIS calculates the matrix for all nodes of the tree, including inner nodes, while the other libraries shown only return the distance matrix for the taxa (leaf nodes) of the tree, which is only a quarter of the size. This explains the better memory footprint of SCIKIT-BIO for larger trees. We could not test all tree sizes here, as the resulting matrices would have exceeded the available memory. This is also the reason why DendroPy was not able to calculate the matrix for 10 K taxa on the given hardware.

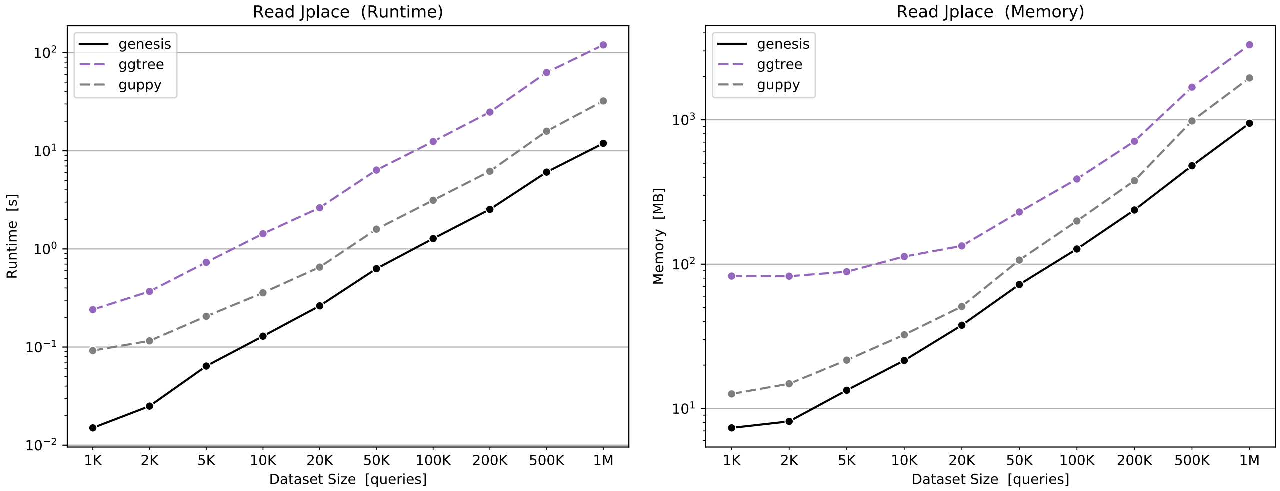

Figure S6: Runtimes and memory footprints for reading `jplace` files with 1 K to 1 M placed sequences ("queries"). The `jplace` format [7] is a standard file format for storing phylogenetic placements of sequences on a given, fixed reference tree. We conducted this test, because the manipulation, analysis, and visualization of phylogenetic placement data constitutes one of the main use cases of GENESIS and GAPPA. Here, we used the 1 K taxon tree of Figure S4 and randomly placed sequences on its branches, with each sequence having up to 5 random placement positions (branches). We include a comparison with GUPPY, which is a command line tool for the analysis of phylogenetic placements, and the tool on which many ideas and concepts of our GAPPA tool are based upon. As GUPPY is not a library, we conduct the test by measuring the runtime and memory usage of its `info` command. The command simply reads a `jplace` file and prints the number of queries stored in the file. GENESIS consistently outperforms both alternative tools that can parse `jplace` files, and is about one order of magnitude faster than GGTREE.

References

[1] Flouri T, Darriba D, et al. (2019) libpll-2. https://github.com/xflouris/libpll-2 Accessed: 2019-05-08.

[2] Darriba D, Kozlov A, Barbera P, Morel B, Stamatakis A (2019) pll-modules. https://github.com/ddarriba/pll-modules Accessed: 2019-05-08.

[3] Kozlov AM, Darriba D, Flouri T, Morel B, Stamatakis A (2019) RAxML-NG: A fast, scalable, and user-friendly tool for maximum likelihood phylogenetic inference. *Bioinformatics.*

[4] Pearson WR, Lipman DJ (1988) Improved tools for biological sequence comparison. *Proceedings of the National Academy of Sciences* 85(8):2444–2448.

[5] Felsenstein J (1981) Evolutionary trees from DNA sequences: A maximum likelihood approach. *Journal of Molecular Evolution* 17(6):368–376.

[6] Archie J, et al. (1986) The Newick tree format.

[7] Matsen FA, Hoffman NG, Gallagher A, Stamatakis A (2012) A format for phylogenetic placements. *PLoS ONE* 7(2):1–4.

[8] Darriba D, Flouri T, Stamatakis A (2018) The State of Software for Evolutionary Biology. *Molecular Biology and Evolution* p. msy014.

[9] Fletcher W, Yang Z (2009) INDELible: A Flexible Simulator of Biological Sequence Evolution. *Molecular Biology and Evolution* 26(8):1879–1888.

[10] Katoh K, Misawa K, Kuma K, Miyata T (2002) MAFFT: a novel method for rapid multiple sequence alignment based on fast Fourier transform. *Nucleic Acids Research* 30(14):3059–3066.

[11] Huelsenbeck JP, Ronquist F (2001) MRBAYES: Bayesian inference of phylogenetic trees, Technical Report 8.

[12] Notredame C, Higgins DG, Heringa J (2000) T-coffee: a novel method for fast and accurate multiple sequence alignment. *Journal of Molecular Biology* 302(1):205–217.

[13] Rambaut A, Grassly NC (1997) Seq-Gen: an application for the Monte Carlo simulation of DNA sequence evolution along phylogenetic trees. *Bioinformatics* 13(3):235–238.

[14] Paradis E, Claude J, Strimmer K (2003) APE: Analyses of Phylogenetics and Evolution in R language. *Bioinformatics* 20(2):289–290.

[15] Paradis E, Schliep K (2018) ape 5.0: an environment for modern phylogenetics and evolutionary analyses in R. *Bioinformatics* 35(3):526–528.

[16] Cock PJA, et al. (2009) Biopython: freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics* 25(11):1422–1423.

[17] Sukumaran J, Holder MT (2010) DendroPy: a Python library for phylogenetic computing. *Bioinformatics* 26(12):1569–1571.

[18] Huerta-Cepas J, Serra F, Bork P (2016) ETE 3: Reconstruction, Analysis, and Visualization of Phylogenomic Data. *Molecular Biology and Evolution* 33(6):1635–1638.

[19] Yu G, Smith DK, Zhu H, Guan Y, Lam TTY (2017) ggtree: an r package for visualization and annotation of phylogenetic trees with their covariates and other associated data. *Methods in Ecology and Evolution* 8(1):28–36.

[20] Matsen FA, Kodner RB, Armbrust EV (2010) pplacer: linear time maximum-likelihood and Bayesian phylogenetic placement of sequences onto a fixed reference tree. *BMC Bioinformatics* 11(1):538.

[21] Knight R, et al. (2019) scikit-bio. http://scikit-bio.org/ Accessed: 2019-05-08.