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 and GAPPA to competing software. First, we evaluate the runtime and memory requirements for some representative and frequent tasks, by comparing GENESIS to libraries and tools that have similar scope and functionality. Second, we compare GAPPA to the GUPPY tool by benchmarking some of their common functionality. GUPPY is part of the PPLACER suite of programs [1], which were one of the first phylogenetic placement tools. Third, we evaluate the quality of the GENESIS code in comparison to other scientific software written in `C++` and `C`.

1 Runtime and Memory

We compare GENESIS and GAPPA 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 and GAPPA 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 two other libraries developed by our lab, namely LIBPLL [2] and the accompanying PLL-MODULES [3]. They are written in `C`, and are the core libraries of the recent re-implementation of RAXML-NG [4]. We however note that these libraries are independed of GENESIS and do not share any code.

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 of GENESIS for the following general tasks:

- Parsing files in `fasta` format [5]; see Figure S1.
- Parsing files in `phylip` format [6]; see Figure S2.
- Calculating base frequencies on sets of sequences; see Figure S3.
- Parsing files in `newick` format [7]; see Figure S4.
- Calculating the pairwise patristic distance matrix on a given tree with branch lengths; see Figure S5.
- Parsing files in `jplace` format [8]; see Figure S6.

Furthermore, we evaluted the runtime and memory requirements of GAPPA for the following types of typical analyses of phylogenetic placement data:

- Calculate the EDPL of the queries in a placment sample [1]; see Figure S7.
- Calculate the Kantorovich-Rubinstein (KR) distance [9] between sets of placement samples; see Figure S8.
- Calculate the Edge Principal Components (Edge PCA) [10] of sets of placement samples; see Figure S9.
- Calculate the Squash Clustering [10] of sets of placement samples; see Figure S10.

We provide all scripts used for testing the tools and creating the plots shown here at https://github.com/lczech/genesis-gappa-paper. The tests were run on a 4-core laptop with 12 GB of main memory. We note that GENESIS and GAPPA offer parallel processing and computation. They 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 and GAPPA 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.

Furthermore, GAPPA generally outperforms GUPPY in all tests, both in terms of runtime and memory usage. There is only one type of analysis (Edge PCA, Figure S9) where GUPPY has a slight speed advantage for small datasets—which however run fast anyway. For larger datasets, this advatage vanishes, and GAPPA becomes ever more efficient.

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 [11] 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 [12], MAFFT [13], MRBAYES [14], T-COFFEE [15], and SEQ-GEN [16].

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]	-
GAPPA	`C++`	0.5.1	[this article]	-
APE	`R`	5.3	[17]	7390
BIOPYTHON	`Python`	1.73	[18]	1712
DENDROPY	`Python`	4.4.0	[19]	935
ETE3	`Python`	3.1.1	[20]	644
GGTREE	`R`	1.10.5	[21]	294
GUPPY	`OCaml`	1.1.a19	[1]	476[1]
LIBPLL	`C`	2019-04-13	[2, 3]	-
SCIKIT-BIO	`Python`	0.5.5	[22]	-

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

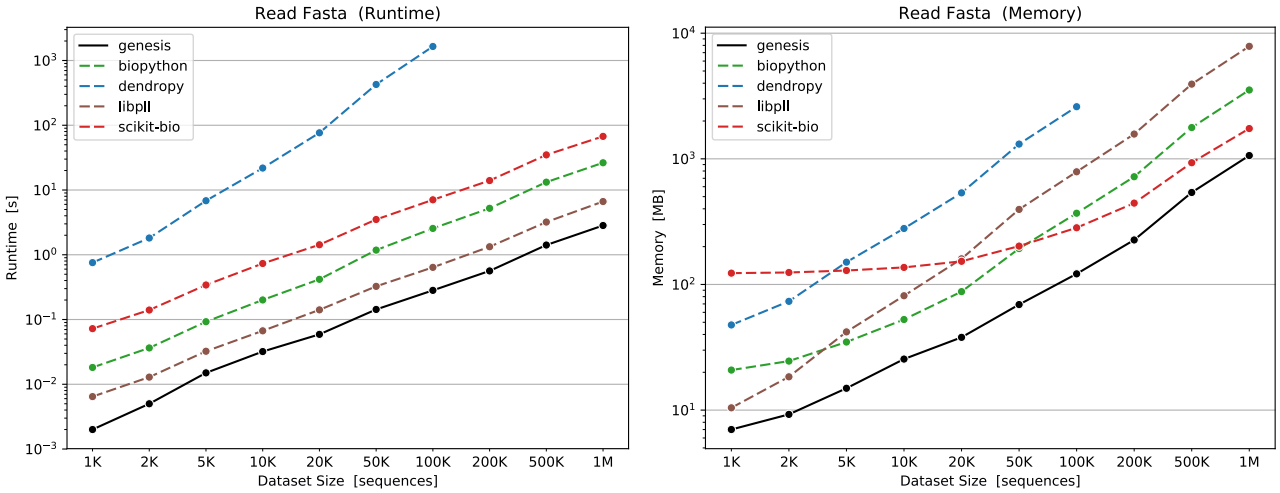

Figure S1: Runtimes and memory footprints for reading `fasta` files [5] containing 1 K to 1 M sequences. The input files consist of randomly generated sequences using the alphabet '`-ACGT`', with a length of 1000 characters per sequence. Hence, the largest file is about 1 GB in size.
In this test, GENESIS 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 $\approx$ 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.

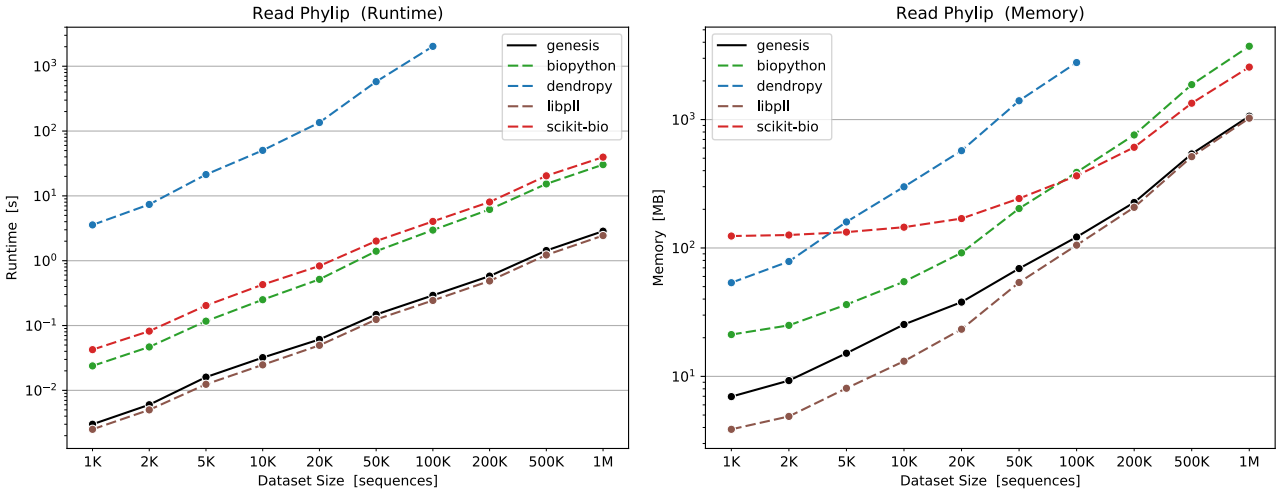

Figure S2: Runtimes and memory footprints for reading `phylip` files [6] containing 1 K to 1 M sequences. The sequences are the same as in Figure S1, but converted to a strict `phylip` format. Strict `phylip` requires exactly 10 characters for sequence names, and does not allow line breaks in the sequences. 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.

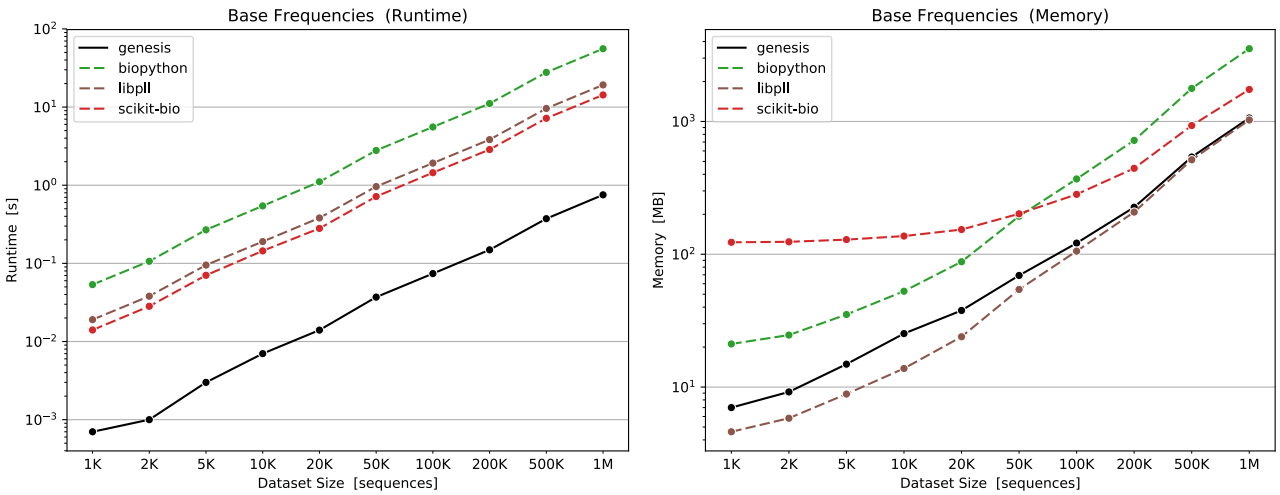

Figure S3: Runtimes and memory footprints for calculating base frequencies on a given set of sequences. Apart from merely reading and parsing files, we also tested some fundamental functionality for working with sequence data. Unfortunately, the range of functions offered by each software largely differs, 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.

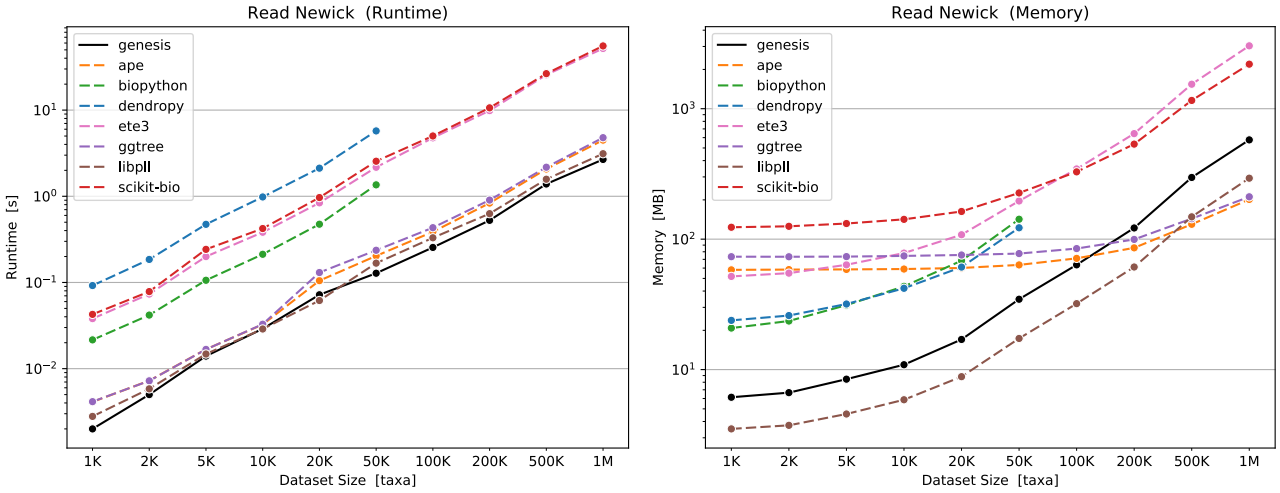

Figure S4: Runtimes and memory footprints for reading `newick` files [7] with 1 K to 1 M taxa (tip/leaf nodes) and a randomly generated topology. The `newick` format is the only one supported by all libraries that we tested. In most cases, GENESIS is fastest, closely followed by LIBPLL, which does however 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.

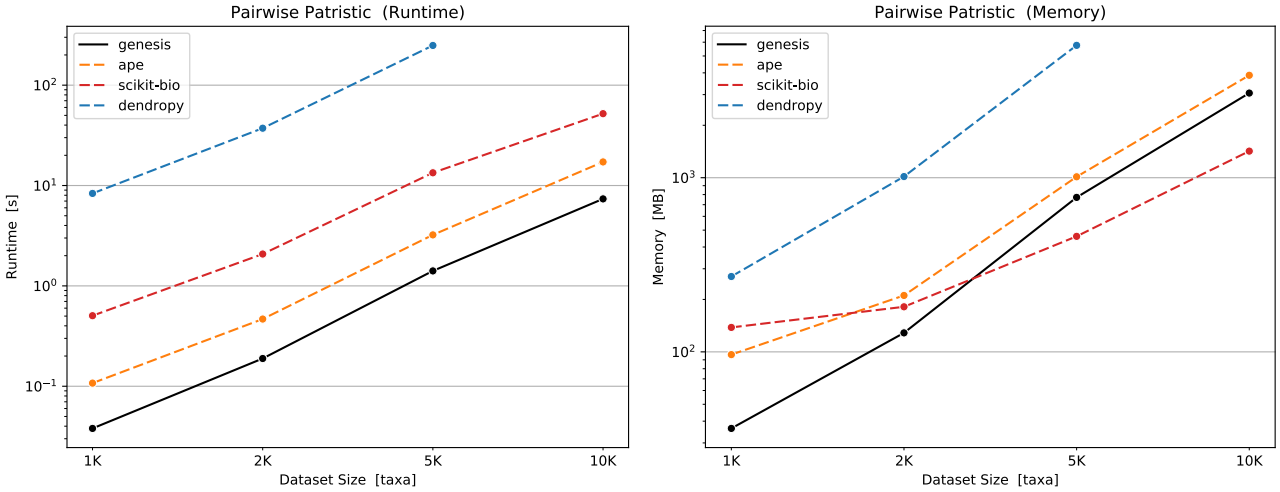

Figure S5: Runtimes and memory footprints for calculating the pairwise patristic distances on a given tree. Similar to Figure S3, we evaluate a simple function that operates on trees: The calculation of the pairwise patristic (branch length) distance matrix between all taxa of the tree. Unfortunately, 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.

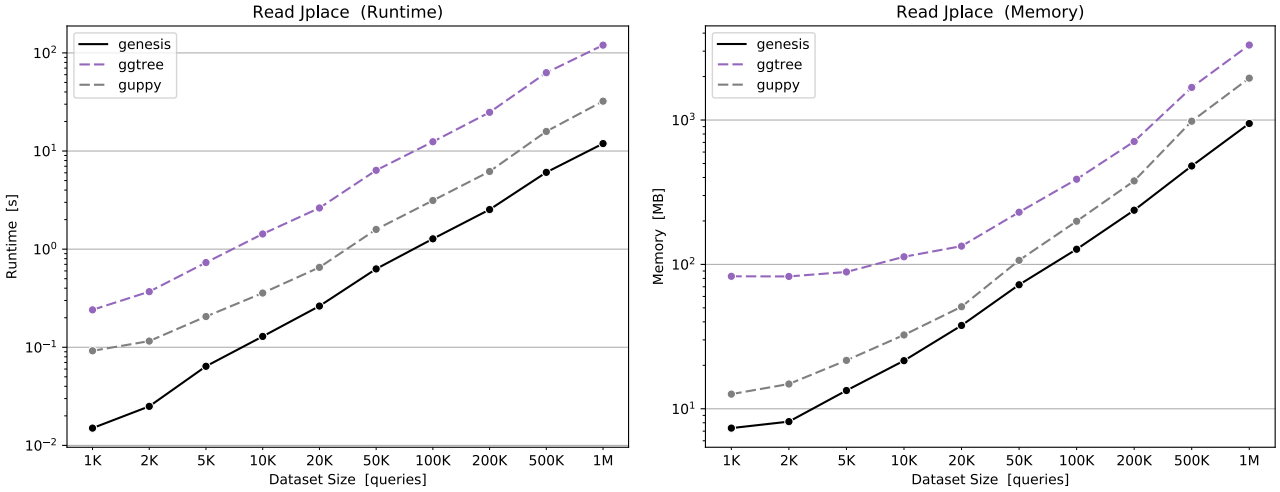

Figure S6: Runtimes and memory footprints for reading `jplace` files with 1 K to 1 M placed sequences ("queries"). The `jplace` format [8] is a standard file format for storing phylogenetic placements of sequences on a given, fixed reference tree. We conducted this test, because the manipulation, analysis, 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.

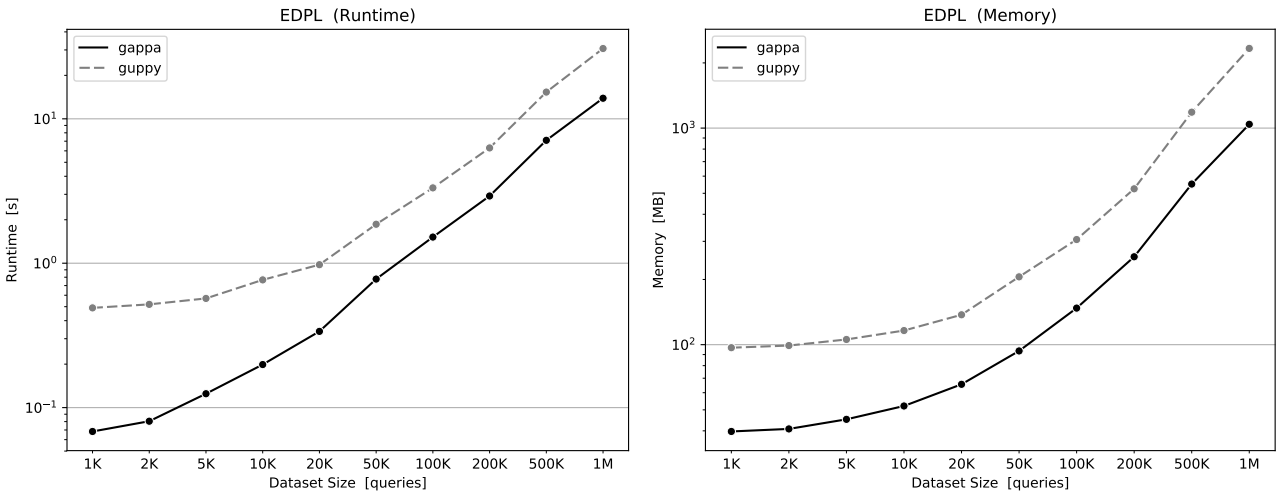

Figure S7: Runtimes and memory footprints for calculating the Expected Distance between Placement Locations (EDPL) [1] of the queries in a `jplace` file. We again use the randomly generated `jplace` files (so-called "samples") of Figure S6 with 1 K to 1 M placed sequences ("queries") here. The EDPL is a metric that measures the uncertainty of each query in a sample, by assessing the distribution of the different placement positions of each query along the branches of the tree. The most time- and memory-consuming part of this analysis is the file reading itself, while the overhead for the actual computation is fairly small. Hence, the plot exhibits similar runtimes and memory footprints as Figure S6. For larger samples, both the runtime and the memory consumption of GAPPA are about 2 times better compared to GUPPY.

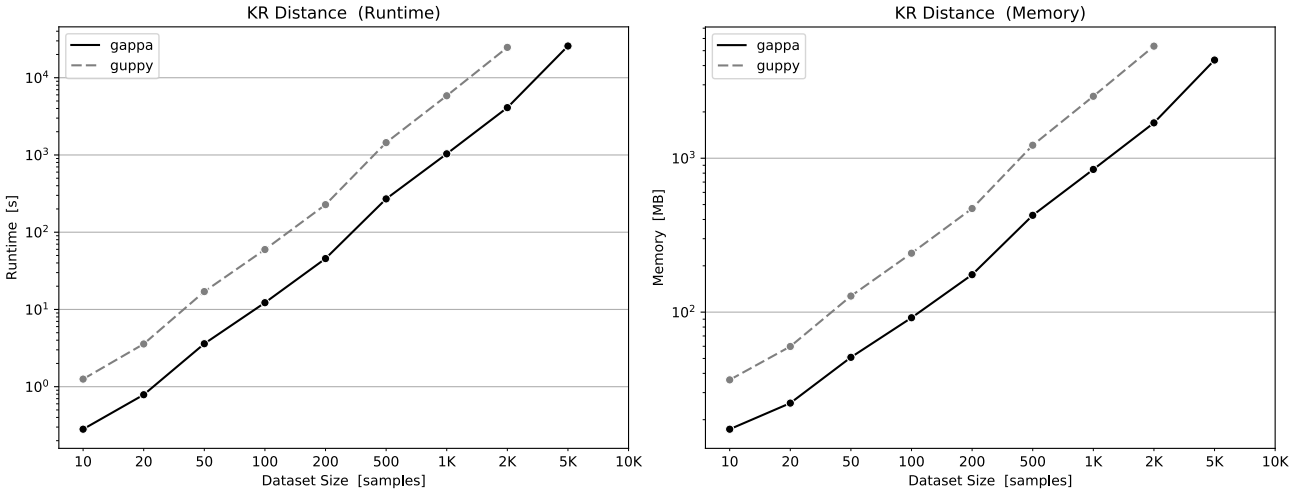

Figure S8: Runtimes and memory footprints for calculating the pairwise Kantorovich-Rubinstein (KR) distance [9] between sets of placement samples. The KR distance is a pairwise metric between placement samples (i. e., between two `jplace` files); hence, we here scale the analysis with the number of samples (`jplace` files). Each sample contains 1000 randomly placed sequences (queries), with each having up to 5 random placement positions (branches). We again used the 1 K taxon tree of Figure S4. The largest analysis with 10K samples corresponds to ≈ 3 GB of `jplace` files; we however could not compute some of the large analyses, as they would have exceeded the available main memory needed for the input data and the resulting pairwise distance matrix. Here, GAPPA is about 5 times faster, and 3 times more memory efficient than GUPPY.

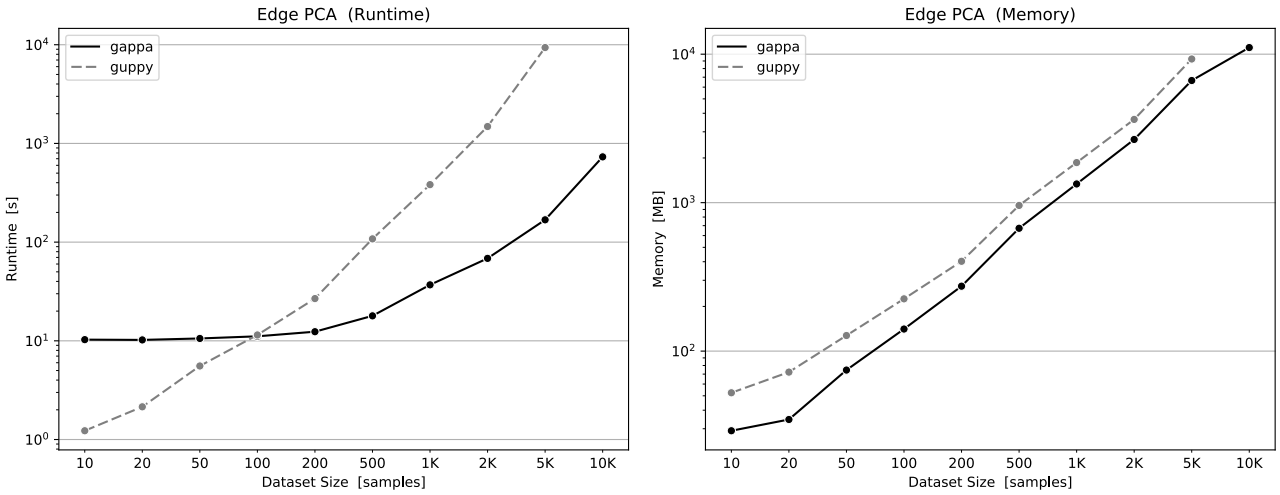

Figure S9: Runtimes and memory footprints for calculating the Edge Principal Components (Edge PCA) [10] of sets of placement samples. The Edge PCA is a typical analysis method for placement data that identifies which taxa of the reference tree are mostly responsible for differences between the composition of distinct samples. Hence, we here again scale the test with the number of samples, again using the random data from Figure S8. The PCA implementation in GAPPA differs from the one in GUPPY in that it scales worse with tree size. As the tree size is constant in the test data (1K taxa), we here get a "constant" offset of $\approx 10\,\mathrm{s}$ for GAPPA even for a small number of samples. This however is alleviated by the generally better runtime of GAPPA that pays off for larger numbers of samples. As the reference trees used in phylogenetic placement usually are around a few thousand taxa is size, this downside of GAPPA should rarely be an issue in practice. GAPPA uses about 70% the memory commpared to GUPPY, and yields growing runtime advantages for larger numbers of samples; for 5K samples, it is 55 times as fast. We could not test GUPPY with 10K samples due to the limited main memory of 12 GB.

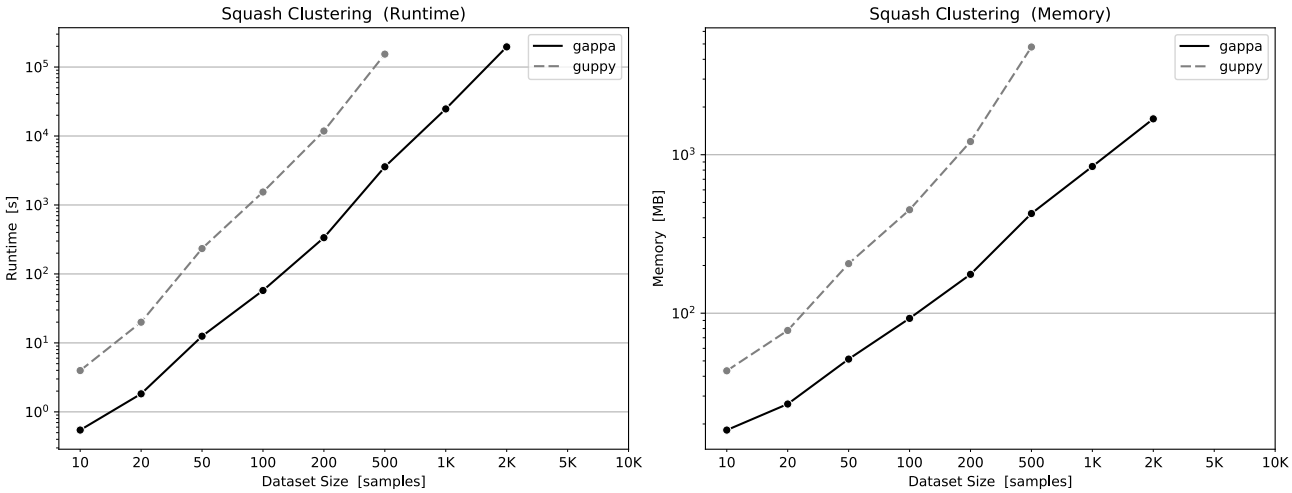

Figure S10: Runtimes and memory footprints for calculating Squash Clustering [10] of sets of placement samples. Squash Clustering is another typical analysis method for placement data that yields a cluster tree of samples by grouping samples that are similar to each other based on their KR distance. We again scale the test with the number of samples, and use the random data from Figure S8. This is by far the most time-demanding type of analysis tested here. The largest tests (with 500 and 2K samples, respectively) correspond to runtimes of ≈ 2 days; we did not run tests with more samples due to these long runtimes. As mentioned, we here limited GAPPA to run on 1 core only; using more cores yields almost linear speedups. Here, GAPPA gains larger improvements for larger numbers of samples compared to GUPPY; for 500 samples, it is 43 times as fast (on a single core) and 11 times more memory efficient.

References

[1] 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.

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

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

[4] 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*.

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

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

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

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

[9] 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.

[10] 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.

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

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

[13] 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.

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

[15] 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.

[16] 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.

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

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

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

[20] 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.

[21] 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.

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