Exploring Trade-offs in Scalable Phylogenetic Placement Methods

Gillian Chu
gchu4@illinois.edu

Tandy Warnow
warnow@illinois.edu

May 23, 2022

Abstract

Phylogenetic placement is the problem of placing "query" sequences into an existing tree (called a "backbone tree") whose leaves are aligned sequences, and has applications to updating large trees and microbiome analysis. While substantial advances have been made in developing methods for phylogenetic placement, to date the most accurate approaches (e.g., pplacer and EPA-ng) are based on maximum likelihood, and these methods tend to have computational challenges when the backbone tree is large. Of the two, EPA-ng can scale to larger backbone tree sizes than pplacer (which seems to be limited to about 5,000-leaf backbone trees), but pplacer seems to have better accuracy than EPA-ng when it can run. Divide-and-conquer methods have been developed to address the limited scalability of pplacer, which operate by finding a small subtree of the backbone tree for the given query sequence, and then placing into that small subtree; `SCAMPP` is a recent development that shows particular benefits. Another approach, which is specific for pplacer, is `taxtastic`, which provides numeric model parameters in a form that helps pplacer run on larger datasets. In this study, we examine the potential of using both these approaches for scaling `pplacer` to large datasets, exploring the impact on accuracy as well as on running time and memory usage. We show that the combination of techniques (i.e., `pplacer-taxtastic-SCAMPP`) produces the best accuracy of all placement methods to date, with excellent speed and reduced memory usage. Finally, we explore how changing the subtree size associated with the `SCAMPP` framework changes the runtime-accuracy trade-off, and discuss avenues for future research. Our software for `pplacer-taxtastic-SCAMPP` is available at https://github.com/gillichu/PLUSplacer-taxtastic.

1 Introduction

Phylogenetic placement is the problem of placing sequences (called 'queries') into an existing phylogeny (called a 'backbone tree'). The backbone tree is assumed to have sequences at the leaves already in an alignment. There are two leading applications of phylogenetic placement methods: updating a large phylogenetic tree (which is of interest to evolutionary biologists) and doing taxon identification and abundance profiling for microbiome samples. Both applications benefit from large backbone trees (see, for example, the discussion in [15] for how increasing backbone tree size improves abundance profiling for metagenomics).

The most accurate of the current phylogenetic placement methods operate using maximum likelihood, and so require the query sequences to already be aligned to the backbone sequences; examples of such methods include pplacer [9] and EPA-ng [3]. Other types of methods have been developed for phylogenetic placement that are not based on maximum likelihood; a recent example is APPLES-2 [1], which uses branch lengths and distances computed between the query sequence and the leaves in the backbone tree to place each query sequence. APPLES-2 is much faster than pplacer and EPA-ng and has lower memory requirements, but is not as accurate as pplacer. Moreover, to date none of the alternative approaches have been as accurate as the maximum likelihood-based methods. Instead, their advantage is computational efficiency, rather than accuracy. Given our interest in accuracy, our focus is on the maximum likelihood placement methods.

Maximum likelihood-based placement pipelines operate by estimating numerical model parameters, such as branch lengths defining expected numbers of substitutions and the substitution rate matrix for the Generalized Time Reversible model [20], and then use these numerical model parameters to find the best edge into which to place each query sequence, using maximum likelihood. In fact, these methods also output the top placements, each with its probability for being the placement edge. Both pplacer and EPA-ng use RAxML [18] software for these numeric model parameter estimations (albeit different versions of RAxML).

Of these two, pplacer seems to have a slight edge for accuracy [2], but is also more computationally intensive than EPA-ng. Moreover, pplacer fails to run on many datasets (or rather produces illegal outputs) when the backbone tree exceeds about 10,000 sequences. The issue for pplacer (but not for EPA-ng), according to a recent study [21], may be numerical, so that once the backbone tree is very large, the log likelihood scores may become too large in magnitude for pplacer. Thus, pplacer fails to run on datasets with more than about 10,000 sequences (occasionally succeeding on somewhat larger datasets). While EPA-ng does not seem to have the same numerical issues as pplacer, it also has computational costs that make it unable to run on datasets with more than about 30,000 sequences [21]. Thus, both of the leading maximum likelihood methods have limitations to somewhat small backbone trees.

Two approaches have been used to address the limited scalability of pplacer. The first is a software package `taxtastic` [4], which computes the numerical model parameters on the backbone tree (required for the maximum likelihood placement of query sequences) using FastTree 2 [14] instead of RAxML. This substitution allows pplacer to run on larger datasets without producing negative infinity log likelihood values, as shown in [1, 21].

The other approach for improving the scalability of pplacer uses divide-and-conquer, so that each query sequence is placed into a subtree of the backbone tree rather than into the entire backbone tree. The techniques that have used this approach include pplacer-SCAMPP [21] and pplacer-DC [5]. The details of the two methods differ, but pplacer-SCAMPP achieved

better scalability, reduced runtime and memory usage, and better accuracy, so we focus on pplacer-SCAMPP.

The fundamental technique in pplacer-SCAMPP is as follows. Given the backbone tree and a query sequence q, the leaf $l(q)$ that is closest to q with respect to Hamming distance is selected, and then a subtree of the B nearest leaves to $l(q)$ is extracted. This subtree is called the **placement subtree**, and is the subtree into which the query sequence q is placed using pplacer. Finally, the placement edge for q in the placement subtree (and the exact location within that edge's length for q's insertion) determines where q is added into the backbone tree. An analysis of settings for B in [21] showed that setting $B = 2000$ gave generally the best accuracy: much smaller settings reduced accuracy in all settings, and sometimes using a larger setting for B greatly increased error. This approach with $B = 2,000$ allowed pplacer-SCAMPP to place into backbone trees with 200,000 leaves with high accuracy. Moreover, pplacer-SCAMPP with this setting was more accurate than APPLES-2, and even faster than APPLES-2 on the largest backbone trees when placing fragmentary sequences. Finally, for a fixed setting for B, pplacer-SCAMPP is very fast (linear time) and has low memory requirements.

Despite these advances, as far as we know, nothing is known about the relative accuracy of pplacer (used in default mode, which means using RAxML for branch length estimation) compared to pplacer-taxtastic (i.e., using taxtastic for branch length estimation). In addition, very little is known about the relative accuracy of pplacer-taxtastic compared to pplacer-SCAMPP. Finally, no study has combined these techniques to see how the two methods operate together; that is, follow the SCAMPP divide-and-conquer framework, but instead of using pplacer in default mode to place the query sequence into the placement subtree, use pplacer-taxtastic. Furthermore, pplacer-SCAMPP allows the parameter B to be provided by the user, the fact that pplacer-taxtastic can run on larger backbone trees means we could use larger settings for B (which determines the size of the placement subtree) when using pplacer-taxtastic for the placement step. This paper reports on a study evaluating these placement methods, in a sequence of experiments.

2 Study Design

2.1 Overview

We explored four phylogenetic placement pipelines, which depend on both the method used to estimate numeric parameters as well as the phylogenetic placement method. The phylogenetic placement methods we studied are: pplacer-default (also referred to simply as pplacer), pplacer-taxtastic, pplacer-SCAMPP, and pplacer-taxtastic-SCAMPP. We used two collections of simulated datasets to evaluate accuracy (since the true tree is not known for biological datasets).

Our experiments focused on scalability to large datasets and accuracy on large datasets, both when placing full-length sequences (the application relevant to adding sequences into large phylogenies) and when placing fragmentary sequences (the application most relevant to analyses of reads produced in microbiome analyses and metagenomics).

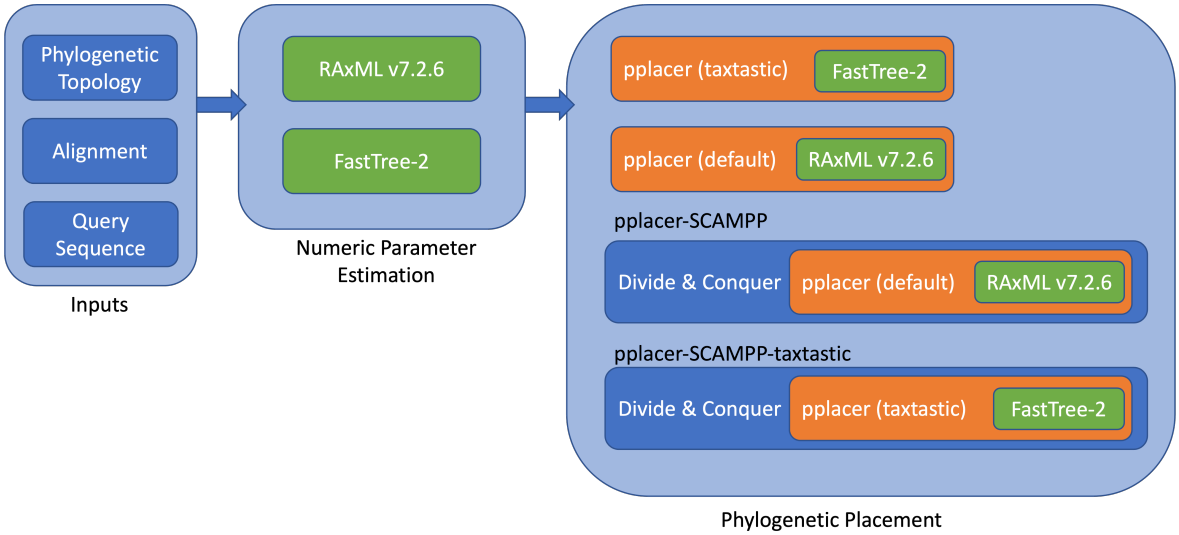

Figure 1: **(Placement pipelines):** The input is a backbone tree ("phylogenetic topology"), an alignment on the backbone sequences, and an aligned query sequence. In the first step, we estimate the numeric parameters on the backbone tree using a selected maximum likelihood code (i.e., RAxML 7.2.6 or FastTree 2.). We then use the specified phylogenetic placement method (i.e., pplacer, pplacer-SCAMPP, or pplacer-SCAMPP-taxtastic). Each of these has a specific technique that is recommended for the previous step (i.e., numeric parameter estimation).

2.2 Placement pipelines studied

The input to a phylogenetic placement pipeline is a backbone tree (in which the leaves are already aligned) and a single query sequence. (Since the query sequences are inserted independently, the extension to multiple query sequences is trivial). The pipelines we explored first estimate branch lengths and other numeric parameters for each backbone tree, then use the specified phylogenetic placement method to insert the query sequence into the tree. The phylogenetic placement methods we studied are:

- pplacer-default (also referred to simply as pplacer).
- pplacer-taxtastic
- pplacer-SCAMPP
- pplacer-taxtastic-SCAMPP

The developers of the phylogenetic placement methods each specify a particular method for numeric parameter estimation; these are indicated in Figure 1. In particular, the `taxtastic` package recommends use with `FastTree-2` numeric parameters, whereas `pplacer (default)` recommends use with `RAxML v7.2.6` (and by extension, `pplacer-SCAMPP` recommends use with `RAxML v7.2.6`).

The pplacer-default pipeline This is the use of pplacer in default mode, which operates as follows. Given a backbone alignment and tree, we use RAxML v. 7.2.6 to estimate numeric parameters on the topology. We then place the query sequence(s) into the tree using pplacer.

The pplacer-taxtastic pipeline Given a backbone alignment and tree, we use FastTree-2 to estimate the numeric parameters on the topology. Then, we use the `taxtastic` package to reformat the tree topology, numeric parameters and alignment into a reference package for use with `pplacer`. Then, `pplacer` uses this reference package to place queries into large datasets.

The pplacer-SCAMPP pipeline Given a backbone alignment and tree, we use RAxML 7.2.6 to estimate the numeric parameters on the topology. Then, given a query sequence, `pplacer-SCAMPP` uses the `SCAMPP` framework to find the nearest leaf (using Hamming distance) to the given query sequence, and uses this to extract a placement subtree of size B (with B=2,000 by default, as recommended in [21]). To place the query sequence into the subtree, `pplacer-SCAMPP` runs `pplacer` for query placement. The placement in the subtree provides the location as well as how the selected branch subdivides its branch length, which is then used to place the query sequence.

The pplacer-SCAMPP-taxtastic pipeline This is identical to pplacer-SCAMPP except for the numeric parameter estimation, as we now describe. Given a backbone alignment and tree, we use FastTree 2 to estimate the numeric parameters on the topology. Then, given a query sequence, `pplacer-SCAMPP-taxtastic` uses the `SCAMPP` framework to find the nearest leaf (using Hamming distance) to the given query sequence, and uses this to extract a

placement subtree of size B. Since pplacer-taxtastic can potentially run on larger trees than pplacer, the optimal value for B may be larger than its default setting in pplacer-SCAMPP. To place the query sequence into the subtree, `pplacer-SCAMPP-taxtastic` builds a `taxtastic` reference package on the subtree using the backbone sequences and numeric parameters, and provides `pplacer` with this `taxtastic` reference package for query placement.

2.3 Datasets

We explore the phylogenetic placement methods on four publicly available simulated datasets (`ROSE 1000M1`, `ROSE 1000M5`, `RNASim`, `nt78`). We have summarized the empirical properties of each dataset in Table 1.

ROSE 1000-sequence datasets The ROSE datasets (1000M1 and 1000M5) have 1000 sequences each and were simulated with substitutions (under the GTR+GAMMA model) with medium length indels using ROSE [19]; these were originally simulated for the SATé study [8] but have also since been used in studies of alignment accuracy [17, 16, 13] and placement accuracy [10]. We used the provided binary model trees and the true alignment. Each model condition dataset contains 20 replicates. In this study, we arbitrarily chose the first 5 replicates, where each replicate contains $1,000$ taxa. We explore our phylogenetic placement methods on the $1000M1$ and $1000M5$ datasets, where the M indicates a "medium" gap length, and 1 indicates the highest rate of evolution, whereas 5 indicates the lowest rate of evolution [8]. The rates of evolution are also reflected in the average p-distance (i.e., normalized Hamming distance) for each of these datasets.

RNASim-VS We use the RNASim-VS datasets, which are subsets of the million-taxon `RNASim` simulated dataset [11]. The RNASim simulation is under a non-homogeneous biophysical fitness model which reflects selective pressures to maintain the RNA secondary structure (as opposed to sites that evolve identically and independently down the tree) [11]. This RNASim dataset has been widely used in studies evaluating alignment accuracy [11, 16, 17, 12] and random subsets of the RNASim dataset (RNASim-VS) have also been used to study phylogenetic placement accuracy [21, 1]. This dataset was specifically used to study `APPLES`, `APPLES-2`, `pplacerDC`, `pplacer-SCAMPP`, and in this paper will be used to evaluate `pplacer-SCAMPP -taxtastic`. The RNASim Variable-Size (`RNASim-VS`) datasets provide true phylogenetic trees, multiple sequence alignments and estimated FastTree-2 [14] maximum likelihood trees. We use three subsets from the `RNASim-VS` dataset, containing $50,000$ sequences, $100,000$ and $200,000$ sequences. We refer to each subset as `RNASim-50k`, `RNASim-100k` and `RNASim-200k`. For `RNASim-50k` we arbitrarily choose to use the first 5 replicates. For the two larger datasets, we run on one replicate (arbitrarily picking `R0`).

nt78 This dataset contains $78,132$ sequences and 20 simulated replicates, and was created to study `FastTree-2` [14]. We chose the first replicate to use as it has been used in other studies as well [21]. The `nt78` dataset was simulated with `ROSE` [19] under the HKY model to resemble 16S sequences, and with different evolutionary rates for each site selected from 16 different rate categories. See the Appendix for additional information about this simulated dataset.

Dataset	seqs	align length	type	p-distance average	p-distance maximum	prop gaps	reps
nt78 [14]	78,132	1,287	simulated	.404	.639	.006	1
RNASim-50k [11]	50,000	1,620	simulated	.410	.618	.051	5
RNASim-100k [11]	100,000	1,620	simulated	.410	.618	.051	1
RNASim-200k [11]	200,000	1,620	simulated	.410	.618	.051	1
ROSE 1000M1 [7]	1,000	3,895	simulated	.697	.770	.738	5
ROSE 1000M5 [7]	1,000	1,762	simulated	.501	.607	.426	5

Table 1: **Nucleotide Dataset Statistics:** We provide details for all the datasets used in this study. All used datasets are publicly available. The first column provides the dataset name. For each dataset, we provide the number of sequences, alignment length, type of alignment, p-distance (normalized Hamming distance) average and maximum, proportion of gaps and number of replicates.

Fragmentary Protocol We evaluated the placement methods when given fragmentary sequences on the RNASim-VS and `nt78` datasets. Given a query sequence with original length l_o, a random starting position was selected by sampling from a normal distribution. Two sequences of different length were generated for each query sequence: in the low fragmentary case, sequence length was sampled using $\mathcal{N}(\mu = .25l_o, \sigma = 60)$. In the high fragmentary case, sequence length was sampled using $\mathcal{N}(\mu = .10l_o, \sigma = 10)$. Note that these distributions were chosen because 154 basepairs is close to the Illumina read sequence length, making a study of placing read-length query sequences more relevant to downstream applications of phylogenetic placement. The fragment lengths for both `RNASim-VS` and `nt78` are both reported in the context of those experiments.

Leave-one-out Experiments All experiments in this study were run as leave-one-out experiments, with results reported on a per-query basis. For all datasets, we randomly choose 200 sequences to be used as the query sequences. Thus, for a given dataset with n sequences, with one selected as a query sequence, we use each phylogenetic placement method to place the selected query sequence into the backbone tree containing the other $n - 1$ sequences.

2.4 Criteria

We evaluated phylogenetic placement accuracy using delta error, a distance metric quantifying the increase in estimation error incurred by placing the query sequence into the backbone tree [21, 2]. Suppose T is the backbone tree used, T^* is the true tree, and P is the tree produced when the query q is added into backbone tree T using the phylogenetic placement pipeline. Let $\mathcal{L}$ represent the set of leaves in T, and let $B(t)$ refer to the set of bipartitions for a tree t. Then the equation for delta error is provided below:

$$\Delta e(P) = |B(T^*) \backslash B(P)| - |B(T^*|_{\mathcal{L}}) \backslash B(T)|$$

Note that delta error is always non-negative, and that regardless of whether the estimated backbone tree used for placement contains errors, if the query sequence is placed on the same edge as in the reference tree, the delta error can still be 0.

2.5 Experiments

- Experiment 1: We compared the behavior of `pplacer-default` and `pplacer-taxtastic` when placing full-length query sequences using `ROSE-1000M1`, `ROSE-1000M5`, and `nt78` datasets.
- Experiment 2: We explored `pplacer-taxtastic` and `pplacer-SCAMPP` on `nt78` and `RNASim-50k`, using both full and fragmentary query sequences.
- Experiment 3: We evaluate the impact of changing the subtree size in `pplacer-SCAMPP -taxtastic`, on both the `nt78` and `RNASim`-VS and nt78 datasets, in comparison to `pplacer-SCAMPP` and `pplacer-taxtastic`.

All experiments were performed on the UIUC Campus Cluster, using the secondary queue with a time-limit of 4 wall-time hours per query and 64 GB of available memory.

3 Results and Discussion

3.1 Experiment 1

In Experiment 1 we explored the behavior of `pplacer (default)` and `pplacer -taxtastic` on the simulated `ROSE` and `nt78` datasets. On the 1000-sequence ROSE 1000M1 and 1000M5 datasets (Figures 2 (a) and (b), respectively), we observe that `pplacer (default)` and `pplacer-taxtastic` have very close delta error, with perhaps a slight advantage to pplacer-default. However, `pplacer-default` does not run on the nt78 dataset with 78,132 sequences (Figure 2 (c)) while `pplacer-taxtastic` does succeed in running, showing a computational advantage when using `pplacer-taxtastic`. Other trends we observe show that `pplacer-taxtastic` and `pplacer-default` have similar runtimes (with a slight advantage to `pplacer-taxtastic`) when both can run. Finally, both methods incur lower delta error on the lower rate of evolution (1000M5) than on the higher rate of evolution (1000M1). Both methods consume negligible memory. Thus, there is a clear advantage to using `pplacer-taxtastic` on large backbone trees where `pplacer (default)` fails to run.

3.2 Experiment 2

Experiment 2 explored the behavior of `pplacer -taxtastic` and `pplacer-SCAMPP` on the simulated `nt78` and `RNASim-50k` datasets, using both full and fragmentary query sequences. On the RNASim-50K datasets (Figure 3), there were no noteworthy differences in delta error except for the shortest query sequences, where `pplacer-taxtastic` had lower error than `pplacer-SCAMPP`. However, `pplacer-SCAMPP` was consistently faster and used less memory than `pplacer-taxtastic`.

On the nt78 dataset (Figure 4), we see bigger differences between the methods. Specifically, with the exception of the full-length sequences (where both do very well), `pplacer -taxtastic` has much lower delta error than `pplacer-SCAMPP`, an improvement of several orders of magnitude. On the other hand, for all query lengths, `pplacer-SCAMPP` is significantly faster and consumes less memory than `pplacer-taxtastic`.

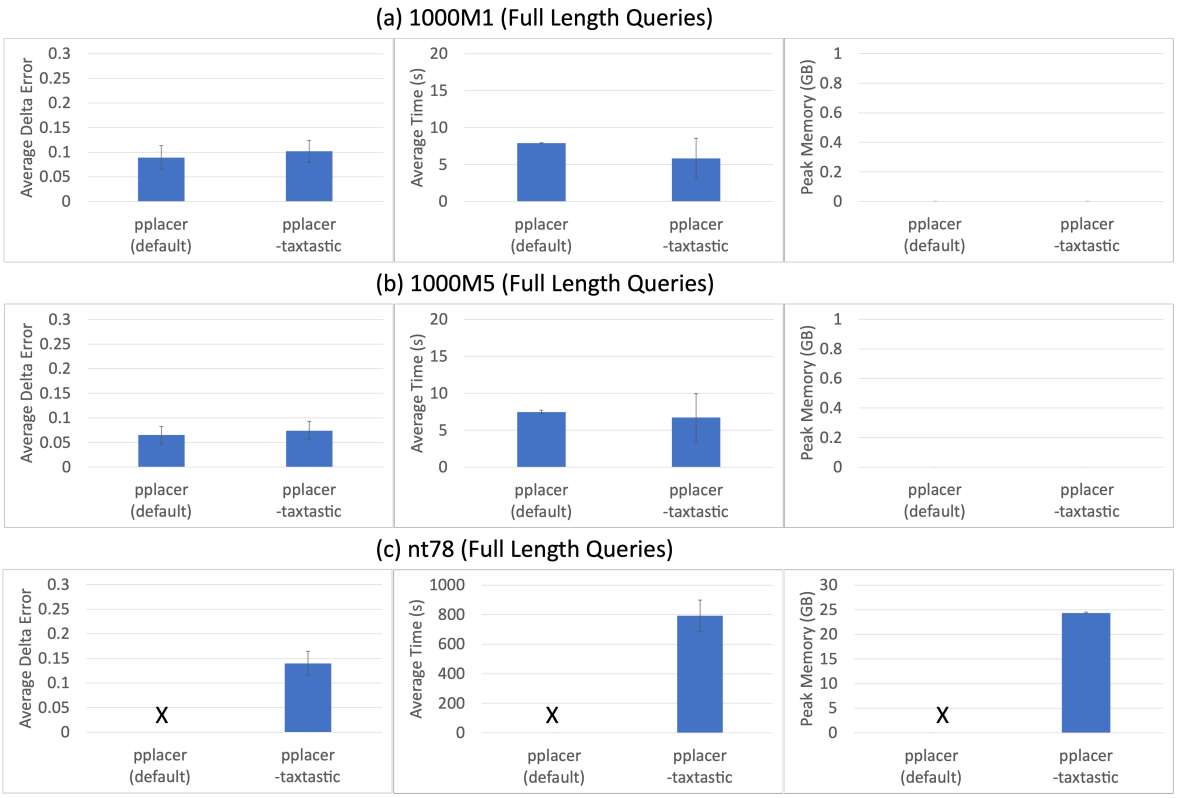

Figure 2: **(Experiment 1) Comparison between pplacer(default) and pplacer-taxtastic on the ROSE and nt78 datasets.** (a) Results on the ROSE 1000M1 datasets, (b) Results on the ROSE 1000M5 datasets, and (c) Results on one replicate of the nt78 dataset containing 78,132 sequences. The left two columns show per-query delta error results, the middle two columns show average per-query runtime to place each sequence, and the two rightmost columns illustrate per-query memory consumption. For delta error we present standard error, and for runtime and memory consumption we present standard deviation. Note that the runtime and memory usage y-axes are different between the ROSE and nt78 datasets. The ROSE 1000M1 and 1000M5 datasets contain 1,000 sequences and 5 replicates each. 1000M1 has a high rate of evolution and 1000M5 has a low rate of evolution. The "X" indicates that pplacer (default) failed with an out of memory error on these experiments. Peak memory usage on the ROSE datasets is negligible, so that panels (a) and (b) do not show visible memory usage for either method.

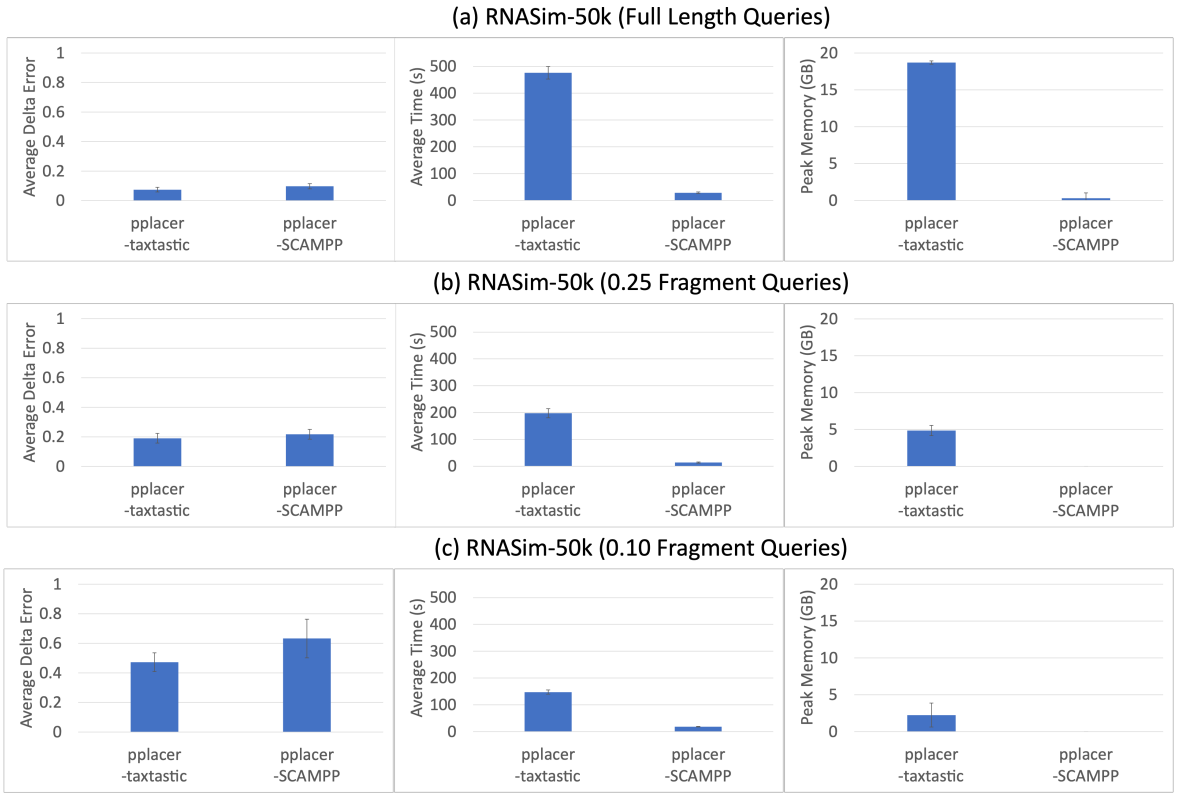

Figure 3: **(Experiment 2): Comparison between pplacer-SCAMPP and pplacer-taxtastic on RNASim-VS datasets using both full and fragmentary query sequences.** Per-query results on the fragmentary RNASim (simulated) dataset containing $50,000$ sequences. Panel (a) shows results on full length sequences, panel (b) shows results on 0.25 fragment length (385 basepairs) and panel (c) shows results on 0.10 fragment length (154 basepairs). For all panels, the first column shows per-query delta error results, the second column shows average per-query runtime to place each sequence and the third column illustrates per-query memory consumption. For delta error we present average standard error, and for runtime and memory consumption we present standard deviation. Please note that the value for B, which determines the placement subtree size for pplacer-SCAMPP ,is set to the default ($B = 2,000$).

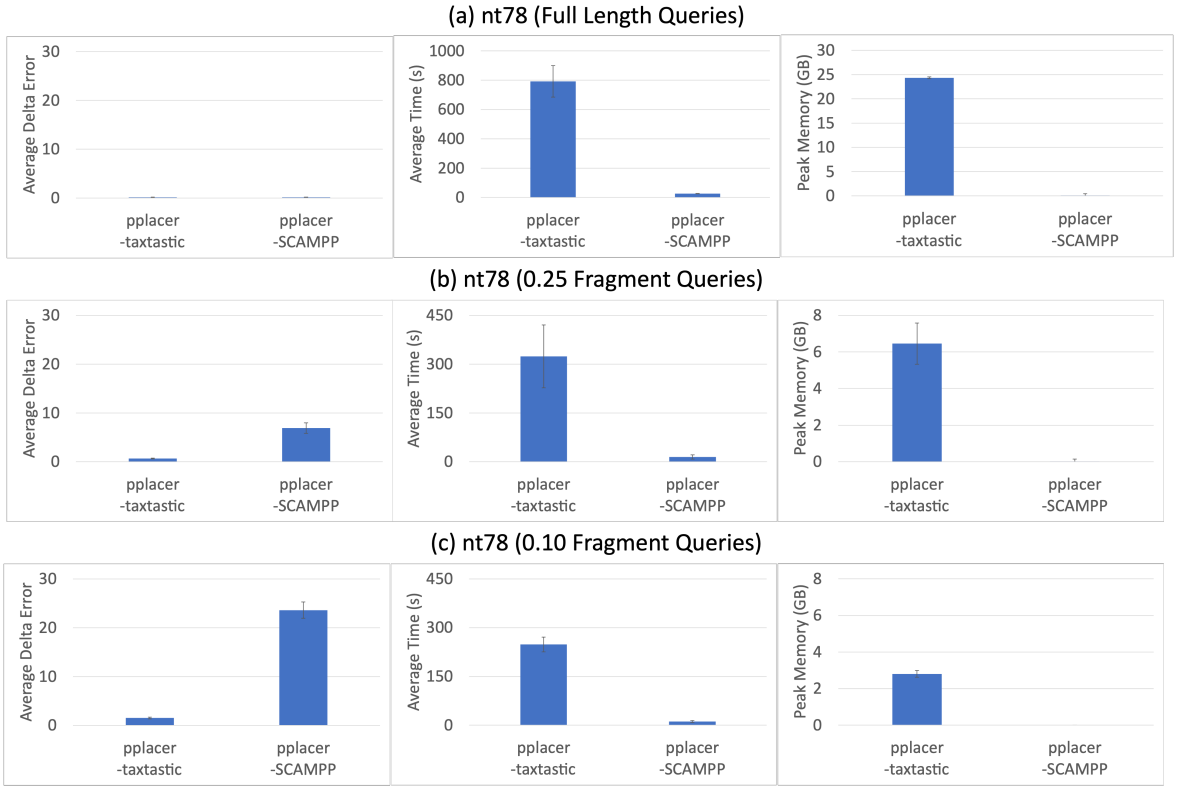

Figure 4: **(Experiment 2): Comparison between pplacer-SCAMPP and pplacer-taxtastic on the nt78 datasets when placing full and fragmentary query sequences.** Per-query results on the nt78 (simulated) dataset containing 78,132 sequences. Panel (a) shows results on full length sequences, panel (b) shows results on 0.25 fragment length (322 basepairs), and panel (c) shows results on 0.10 fragment length (127 basepairs). For all panels, the first column shows per-query delta error results, the second column shows average per-query runtime to place each sequence and the third column illustrates per-query memory consumption. For delta error we present average standard error, and for runtime and memory consumption we present standard deviation. **Note that the y-axes are different for runtime and memory usage between the full-length and fragmentary datasets.**

Method	delta error	runtime (s)	peak memory (GB)
pplacer-SCAMPP-taxtastic (2k)	6.83	27.50	0.09
pplacer-SCAMPP-taxtastic (10k)	3.86	79.43	0.63
pplacer-SCAMPP-taxtastic (20k)	2.30	164.88	1.79
pplacer-SCAMPP-taxtastic (30k)	1.25	358.25	2.68
pplacer-SCAMPP-taxtastic (40k)	0.90	482.88	3.53
pplacer-SCAMPP	6.91	14.65	0.02
pplacer-taxtastic	0.69	323.86	6.46

Table 2: **Summary of results from Experiment 3, when placing** 0.25 **fragment query sequences into the nt78 dataset (see Figure 5).**

3.3 Experiment 3

Experiment 3 evaluates `pplacer -SCAMPP -tax` using several different subtree sizes on the `nt78` and `RNASim` datasets. Figure 5 (see also Table 1) illustrates `pplacer -SCAMPP -tax`'s behavior as the setting for B changes (which changes the placement subtree sizes in the SCAMPP framework). We observe the accuracy-runtime trade-off as we increase the placement subtree size on both fragment query lengths. As `pplacer -SCAMPP -tax` incurs less delta error as the placement subtree size increases, but the runtime and memory consumption also increase. As expected, `pplacer -SCAMPP -tax` greater delta error on the shorter fragment lengths, but the runtime and memory consumption on 0.10 fragment query sequences is less than on the longer 0.25 fragment query sequences.

We see that `pplacer-SCAMPP-taxtastic (20k)` uses a fraction of the runtime used by `pplacer-taxtastic`. All `pplacer-SCAMPP-taxtastic` variants tested definitively improve on the peak memory usage when compared to `pplacer-taxtastic`. Additionally, we see that as the subtrees grow larger, the runtime and memory increase but the error decreases. From Table 2, we can observe that `pplacer-taxtastic` still performs best in terms of the delta error. However, `pplacer-SCAMPP-taxtastic (40k)` achieves similar delta error accuracy, while consuming slightly less time and significantly less memory.

In Figure 6, we explored the same per-query results on large RNASim datasets. In particular, we looked at `RNASim-100k` and `RNASim-200k` with 100,000 and 200,000 sequences respectively. On each dataset, we compared variants of `pplacer-SCAMPP -tax` using different sizes of subtrees, `pplacer-SCAMPP` and `pplacer-taxtastic`. However, as we can see from Panel (a), `pplacer-SCAMPP` suffers slightly higher delta error while `pplacer-taxtastic` achieves slightly lower delta error on `RNASim-100k`. We observe that as the size of the placement subtree increases, the average delta error for `pplacer-SCAMPP-tax` decreases and the runtime increases.

In Panel (b) of Figure 6, we see that all methods which can run perform quite similarly in terms of delta error. Notably, unfortunately `pplacer-taxtastic` was completely unable to run due to an out of memory error. In terms of runtime and memory consumption, we see that `pplacer-SCAMPP` actually takes more time (perhaps due to the memory constraints). Consistently, however, we see that `pplacer-SCAMPP-tax` increases in runtime and memory consumption as the size of the placement subtree increases.

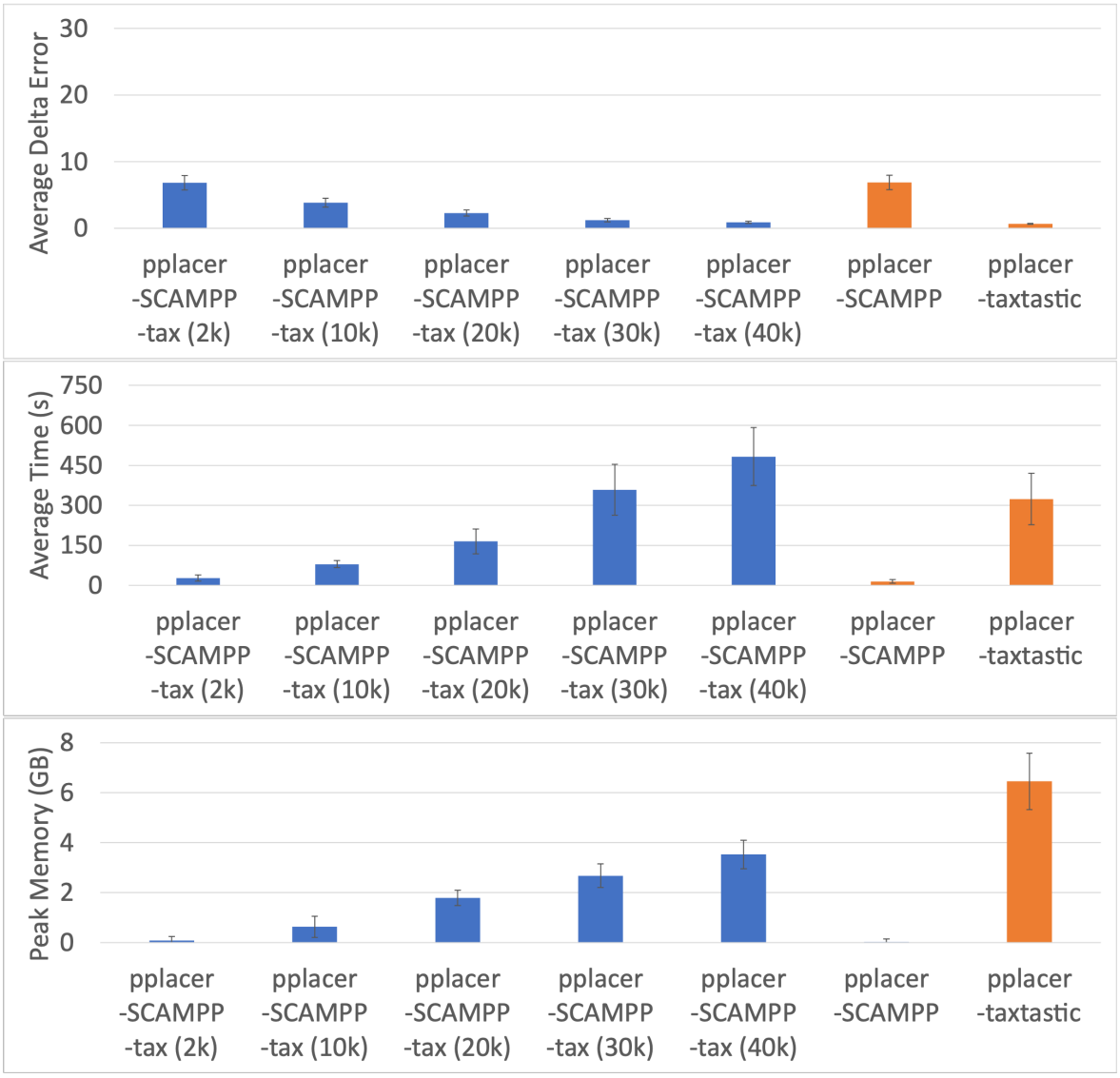

Figure 5: **(Experiment 3): Comparison between pplacer-SCAMPP-taxtastic, pplacer-SCAMPP and pplacer-taxtastic when varying subtree placement size and placing fragmentary query sequences into the nt78 tree.** Per-query results on the nt78 (simulated) dataset containing 78,132 sequences. Panel (a) shows results on 0.25 fragment length (322 basepairs) and panel (b) shows results on 0.10 fragment length (127 basepairs). For all panels, the first column shows per-query delta error results, the second column shows average per-query runtime to place each sequence and the third column illustrates per-query memory consumption. For delta error we present average standard error, and for runtime and memory consumption we present standard deviation. Please note that the y-axes are different between the full-length and fragmentary datasets. Along the x-axis the methods are sorted by the placement subtree size B, given parenthetically. Specific numbers are shown for the 0.10 fragmentary case in Table 2.

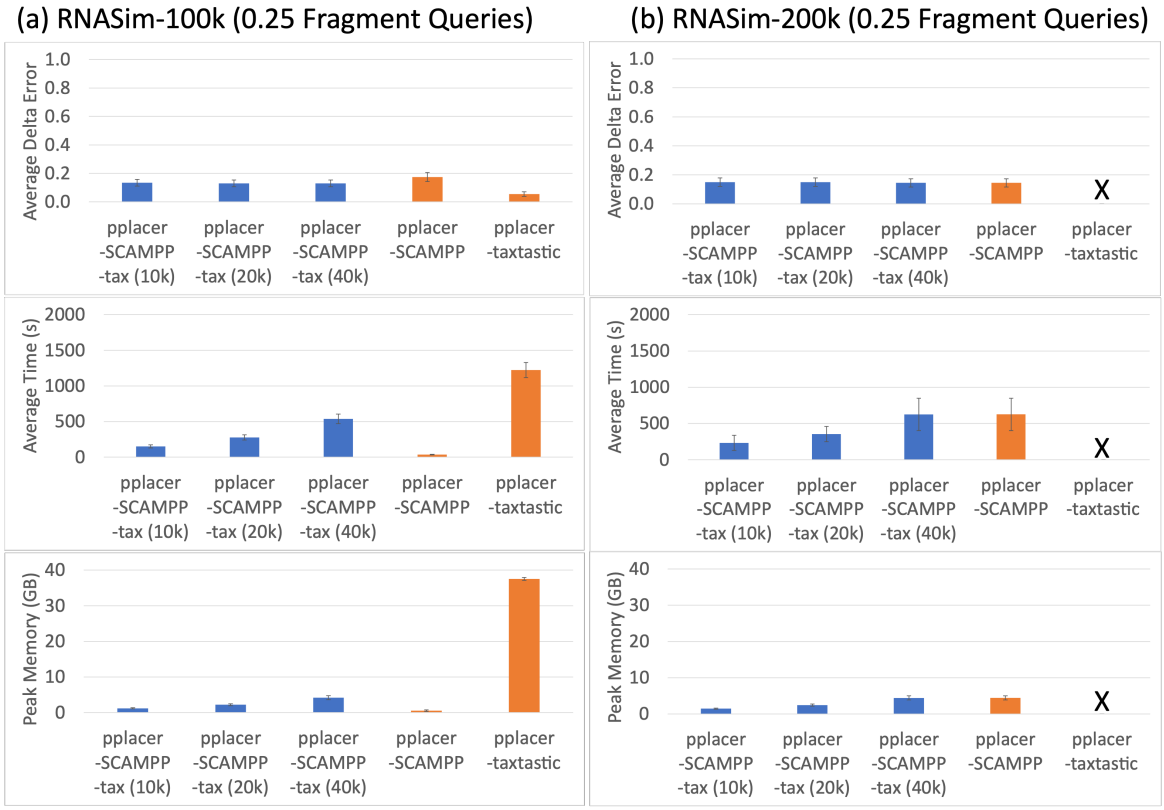

Figure 6: **(Experiment 3): Comparison between pplacer-SCAMPP-taxtastic, pplacer-SCAMPP and pplacer-taxtastic when varying subtree placement size and placing fragmentary query sequences in the RNASim-100K and RNASim-200K datasets.** Per-query results placing 0.25 fragmentary query sequences into the RNASim-100k and RNASim-200k (simulated) datasets. Panel (a) shows results on the 100,000-leaf backbone tree and panel (b) shows results on the 200,000-leaf backbone tree. For both panels, the first row shows per-query delta error results, the second row shows average-per-query runtime to place each sequence and the third row illustrates per-query memory consumption. For delta error we present average standard error, and for runtime and memory consumption we present standard deviation. pplacer-taxtastic failed with an out of memory error (64 GB provided to each placement job) for each of the 200 query sequences for the RNASim-200k dataset.

4 Conclusion

This study set out to understand two different techniques for scaling `pplacer`, a highly accurate phylogenetic placement method, to large backbone trees. The first of these techniques is `taxtastic`, which provides the numeric model parameters needed by `pplacer`, and the second of these techniques is `SCAMPP`, which uses divide-and-conquer. We provided conclusive evidence that both techniques are able to improve scalability (though `SCAMPP` is superior in this regard) and that `SCAMPP` also reduces runtime and memory usage (but this is not true for `taxtastic`). The two techniques have different impacts on placement accuracy, with `taxtastic` typically not changing accuracy but `SCAMPP` sometimes reducing accuracy compared to `pplacer-taxtastic` (when used with its default setting for $B = 2000$, which allows it only to place into subtrees of size 2000). This shows the benefit of being able to place into the full backbone tree (achieved by `pplacer-taxtastic`) compared to only placing into a (relatively small) placement tree when using `pplacer-SCAMPP`. However, we note that the restriction to placement subtrees with only 2000 leaves in `pplacer-SCAMPP` is due to `pplacer`'s tendency to fail or have poor accuracy on larger backbone trees, due to numerical issues. We also note that `pplacer-taxtastic` does not have this limitation, as our study shows it can be used with very large backbone trees (seemingly up to $100,000$ sequences).

These observations (and most importantly on the ability to use `pplacer-SCAMPP` on larger placement subtrees) led to the design of a combined method, `pplacer-SCAMPP -taxtastic`, which follows the `SCAMPP` pipeline but replaces default `pplacer` by `pplacer-taxtastic`. Furthermore, because `pplacer-taxtastic` can be used with large backbone trees, this replacement allows `pplacer-SCAMPP-taxtastic` to be used with larger settings for the placement subtree size B.

Our study shows that `pplacer-SCAMPP -taxtastic` provides the best features of both techniques for improving scalability of pplacer, achieving the same scalability as `pplacer-SCAMPP` (which has been shown to scale to backbone trees with 200,000 sequences, matching the scalability of APPLES-2), and matching (and in some cases improving) on the placement accuracy of `pplacer-SCAMPP` when the parameter B is increased. Moreover, this approach allows the user to select the value for B based on preferences: when accuracy is more important, increasing B (perhaps as a function of the backbone tree size) can be helpful, but when speed and memory usage is more important, keeping B small (e.g., $B = 2000$) can be most desirable.

This study suggests several avenues for further research. Perhaps the most intriguing question is why `taxtastic` benefits `pplacer`: it improves scalability to larger backbone trees (though it still fails on the largest backbone trees we examined, with 200,000 sequences), and in some cases improves accuracy. Understanding the conditions where it improves accuracy is certainly of interest, but understanding why it consistently improves scalability is also important. We hypothesize that `taxtastic` helps pplacer overcome its numerical instability noted in [21], which is why it is able to scale to larger backbone trees; this may also help it to improve accuracy on backbone trees where likelihood scores become very large in magnitude. But how it achieves this is unclear. As `taxtastic` can be used with both `RAxML` and `FastTree-2` numeric parameters, one particularly interesting question is whether the improvement in delta error should be attributed to the `FastTree-2` numeric parameters or whether it should be attributed to the `taxtastic` package.

Finally, this paper also suggests that increasing B, which specifies the placement subtree size within the `pplacer-SCAMPP -taxtastic` framework, does not hurt accuracy and can

improve accuracy, albeit at the cost of slower runtime and higher memory usage. However, the impact of this choice depends on the dataset, with significant accuracy benefits obtained in some cases but none in others (e.g., compare nt78 to RNASim-VS). At this time, it is unclear why there is such a difference between model conditions, and in particular no obvious empirical property (such as average p-distance) that would suggest the use of a larger value for B. Future work is needed to understand this issue, so as to inform practice.

Finally, we did not use biological datasets to evaluate placement accuracy. This is in contrast to some studies (e.g., [6]) that used biological datasets, and calculated placement error with respect a "reference tree" for the species set, based on an estimated tree on the dataset. Because estimated trees have error, we do not consider topological distance to an estimated tree to be a reliable indication of placement accuracy. Moreover, with the exception of species trees for certain subsets of species, the true evolutionary tree is rarely known with confidence. Importantly, gene trees are less reliably known than species trees, given the limited phylogenetic signal in any gene alignment. Finally, given discordance between gene trees and species trees, using a species tree as the "reference" is also problematic for evaluating phylogenetic placement accuracy. For this reason, we used a variety of simulated model conditions to explore phylogenetic placement methods. Instead, rather than using binary trees that are estimated as the reference tree, it would be better to use only the highly supported edges in the estimated maximum likelihood tree to define a reference (e.g., as done in [8]). One challenge for this kind of analysis, however, is that computing very large maximum likelihood trees with branch support is challenging for the dataset sizes we examine here (e.g., 50,000 sequences and more). Thus, future work is necessary to develop rigorous and feasible ways of using biological datasets to evaluate phylogenetic placement methods.

The authors would like to thank Eleanor Wedell for her helpful input as well as for providing datasets and scripts.

5 References

[1] Metin Balaban, Yueyu Jiang, Daniel Roush, Qiyun Zhu, and Siavash Mirarab. Fast and Accurate Distance-based Phylogenetic Placement using Divide and Conquer. *Molecular Ecology Resources*, 22(3):1213–1227, 2021.

[2] Metin Balaban, Shahab Sarmashghi, and Siavash Mirarab. APPLES: Scalable Distance-Based Phylogenetic Placement with or without Alignments. *Systematic Biology*, 69(3):566–578, 2020.

[3] Pierre Barbera, Alexey M Kozlov, Lucas Czech, Benoit Morel, Diego Darriba, Tomáš Flouri, and Alexandros Stamatakis. EPA-ng: massively parallel evolutionary placement of genetic sequences. *Systematic biology*, 68(2):365–369, 2019.

[4] Fred Hutchinson Cancer Research Center. taxtastic. URL: http://fhcrc.github.io/taxtastic/.

[5] Elizabeth Koning, Malachi Phillips, and Tandy Warnow. pplacerDC: a new scalable phylogenetic placement method. In *Proceedings of the 12th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics*, pages 1–9, 2021.

[6] Benjamin Linard, Nikolai Romashchenko, Fabio Pardi, and Eric Rivals. PEWO: a collection of workflows to benchmark phylogenetic placement. *Bioinformatics*, 36(21):5264–5266, 2021.

[7] Kevin Liu, C Randal Linder, and Tandy Warnow. RAxML and FastTree: comparing two methods for large-scale maximum likelihood phylogeny estimation. *PloS one*, 6(11):e27731, 2011.

[8] Kevin Liu, Sindhu Raghavan, Serita Nelesen, C Randal Linder, and Tandy Warnow. Rapid and accurate large-scale coestimation of sequence alignments and phylogenetic trees. *Science*, 324(5934):1561–1564, 2009.

[9] Frederick A Matsen, Robin B Kodner, and E Virginia Armbrust. pplacer: Linear time Maximum-Likelihood and Bayesian phylogenetic placement of sequences onto a fixed reference tree. *BMC Bioinformatics*, 11(1):1–16, 2010.

[10] Siavash Mirarab, Nam Nguyen, and Tandy Warnow. SEPP: SATé- enabled phylogenetic placement. In *Biocomputing 2012*, pages 247–258. World Scientific, 2012.

[11] Siavash Mirarab, Nam-phuong Nguyen, Sheng Guo, Li-San Wang, Junhyong Kim, and Tandy Warnow. PASTA: ultra-large multiple sequence alignment for nucleotide and amino-acid sequences. *Journal of Computational Biology*, 22(5):377–386, 2015.

[12] Nam-phuong Nguyen, Siavash Mirarab, Keerthana Kumar, and Tandy Warnow. Ultra-large alignments using phylogeny-aware profiles. *Genome biology*, 16(1):1–15, 2015.

[13] Minhyuk Park, Stefan Ivanovic, Gillian Chu, Chengze Shen, and Tandy Warnow. UPP2: Fast and Accurate Alignment Estimation of Datasets with Fragmentary Sequences. *bioRxiv*, 2022. doi:10.1101/2022.02.26.482099.

[14] Morgan N Price, Paramvir S Dehal, and Adam P Arkin. FastTree 2–approximately maximum-likelihood trees for large alignments. *PloS one*, 5(3):e9490, 2010.

[15] Nidhi Shah, Erin K Molloy, Mihai Pop, and Tandy Warnow. TIPP2: metagenomic taxonomic profiling using phylogenetic markers. *Bioinformatics*, 37(13):1839–1845, 2021. DOI: 10.1093/bioinformatics/btab023.

[16] Vladimir Smirnov and Tandy Warnow. MAGUS: multiple sequence alignment using graph clustering. *Bioinformatics*, 37(12):1666–1672, 2021.

[17] Vladimir Smirnov and Tandy Warnow. Phylogeny estimation given sequence length heterogeneity. *Systematic Biology*, 70(2):268–282, 2021.

[18] Alexandros Stamatakis. RAxML-VI-HPC: maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models. *Bioinformatics*, 22(21):2688–2690, 2006.

[19] Jens Stoye, Dirk Evers, and Folker Meyer. Rose: generating sequence families. *Bioinformatics (Oxford, England)*, 14(2):157–163, 1998.

[20] Simon Tavaré. Some probabilistic and statistical problems in the analysis of DNA sequences. *Lectures on mathematics in the life sciences*, 17(2):57–86, 1986.

[21] Eleanor Wedell, Yirong Cai, and Tandy Warnow. SCAMPP: Scaling Alignment-based Phylogenetic Placement to Large Trees. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, pages 1–1, 2022. doi:10.1109/TCBB.2022.3170386.

A Summary

We provide commands for running `pplacer -taxtastic`, `pplacer -SCAMPP`, and `pplacer -SCAMPP -taxtastic`. We use the published FastTree-2 results (tree and numeric parameters estimated under Minimum Evolution) on the `nt78` datasets [21]. The scripts to calculate delta error was adapted from a published script [1], and borrow utilities from `newick_utils`. Further details are provided in the published scripts here. Datasets used from [21] can be found here.

B Reference Package

B.1 Numeric Parameters

For `pplacer-SCAMPP` on the ROSE datasets, we compute branch lengths using RAxML (v7.2.8).

- `RAxML-7.2.8-ALPHA/raxmlHPC-PTHREADS -f e -g reference.nwk -m GTRGAMMA -s alignment.phylip -n outname -p 1984 -T 16 -w outdir/`

For all datasets, we compute branch lengths using FastTree-2 in order to run `pplacer -taxtastic`.

- `FastTreeMP -nosupport -gtr -gamma -nt -log reestimated.fasttree.log -intree intree.fasta < alignment.fasta > reestimated.fasttree.tree`

C Placement Methods

pplacer-SCAMPP (v2.0.0, Accessed: May 10, 2022) (github): "jobID" is only necessary if running several jobs at once. Please note that there is now a newer version of pplacer-SCAMPP, which handles fragmentary query sequences slightly more quickly. Experiments 3 (RNASim) and 4 were run with this fix.
On the RNASim-200k dataset, we use the published FastTree estimated branch lengths [21]. As all subsets of the `RNASim-VS` dataset come from the same million-sequence tree, we expect the numeric parameters estimated on the `RNASim-100k` dataset containing $100,000$ sequences to be quite similar to the numeric parameters for `RNASim-200k`. As no `RAxML v. 7.2.6` info file or estimated tree was published [1], we use the numeric parameter file for the `RNASim-100k R0` dataset.

- `python pplacer-SCAMPP.py -i tree_info.log -t backbone_pp.tree -d out_dir/ -a full_msa.aln -b 2000 -n jobID`

pplacer-taxtastic: The new version of `pplacer-taxtastic` (using the speedup in placing fragmentary query sequences implemented on May 16, 2022) was run for results on `RNASim-100k` and `RNASim-200k`. For all other datasets and results on fragmentary query sequences, the version of `pplacer-taxtastic` not implementing this speedup on fragmentary query sequences was used.

- `taxtastic/taxtastic-env/bin/taxit create -P ng-my.refpkg -l name --aln-fast ref.fa --tree-file backbone_pp.tree --tree-stats reestimated.fasttree.log`

pplacer-SCAMPP-taxtastic (github):

- `python pplacer-tax-SCAMPP.py -i reestimated.fasttree.log -t backbone_pp.tree -d outdir -q query.fa -b 2000 -o jplace_name -a ref.fa -r ref.fa -n 1563`

D The nt78 simulated dataset

The nt78 simulation protocol has some interesting features worth considering. According to the `FastTree-2` documentation for these datasets, the internal branch lengths were "truncated to be no less than 0.001 (corresponding to roughly 1 substitution along the internal branch. To make such short branch lengths with Rose... branch lengths [were scaled] by 1,000x (not 100x) and set MeanSubstitution = 0.00134" [14].