SCAMPP+FastTree: Improving Scalability for Likelihood-based Phylogenetic Placement

Gillian Chu
gchu4@illinois.edu

Tandy Warnow
warnow@illinois.edu

August 8, 2022

Abstract

Phylogenetic placement is the problem of placing "query" sequences into an existing tree (called a "backbone tree"), and is useful in both microbiome analysis and to update large evolutionary trees. The most accurate phylogenetic placement method to date is the maximum likelihood-based method pplacer, which uses RAxML to estimate numeric parameters on the backbone tree and then finds the edge in the tree to add the query sequence that maximizes the probability that the resultant model tree generates the query sequence. Unfortunately, pplacer fails to return valid outputs on many moderately large datasets, and so is limited to backbone trees with at most 10,000 or so leaves. We present a technique to enable pplacer to scale to large backbone trees. We draw on two prior approaches: the divide-and-conquer strategy in SCAMPP (Wedell et al., TCBB 2022) and the use of FastTree2 (Price et al., PLOS One 2010) instead of RAxML to estimate numeric parameters. We find that pplacer-SCAMPP-FastTree matches or improves on the accuracy of other placement methods, can scale to large backbone trees with 200,000 sequences, and is fast and has only moderate memory usage. In addition, pplacer-SCAMPP-FastTree enables the user to vary the placement tree size, thus enabling an exploration of the runtime-accuracy trade-off. Our software for pplacer-SCAMPP-FastTree 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'). Phylogenetic placement methods can be used to add new sequences into an existing phylogenetic tree, to taxonomically characterize new sequences, and to perform abundance profiling for microbiome samples. The accuracy of these analyses tends to improve with large backbone trees [17], but only a few phylogenetic placement methods can run on large backbone trees. In particular, the most accurate placement methods, which are based on maximum likelihood (e.g., pplacer [11] and EPA-ng [3]) are generally limited to small backbone trees due to computational reasons. Alternative approaches for phylogenetic placement include distance-based techniques, such as APPLES-2 [1]. 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 pplacer; instead, their advantage is computational efficiency, rather than accuracy. Given our interest in accuracy, our focus is on improving the scalability of maximum likelihood placement methods.

The current most accurate maximum likelihood-based placement method is `pplacer`. The input to `pplacer` is a backbone tree with a multiple sequence alignment that includes the sequences at the leaves as well as the query sequences. The first step in `pplacer` is to estimate the numerical model parameters, such as branch lengths defining expected numbers of substitutions and the substitution rate matrix for the Generalized Time Reversible model [22]. By default, `pplacer` uses RAxML [20] to estimate these parameters. Then, for each query sequence, pplacer finds the edge into which to place the query sequence that maximizes the likelihood (in addition, `pplacer` also outputs the top placements, each with its probability for being the placement edge).

However, `pplacer` fails to run on many datasets (or may produce illegal outputs) when the backbone tree exceeds about 10,000 sequences. A recent study [23] suggests that the problem may be numerical, so that once the backbone tree is very large, the log likelihood scores may become too large in magnitude for `pplacer`. 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 [23]. 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 placement methods, such as `pplacer`. The major approach for improving placement scalability uses divide-and-conquer, with `SCAMPP` [23] the leading such technique. `SCAMPP` adds each query sequence into the backbone tree by first letting the query sequence select a small placement subtree within the backbone tree, and then runs the phylogenetic placement method of choice (e.g., `pplacer` or `EPA-ng`) to place the query sequence into that placement subtree. By using branch length information, `SCAMPP` then identifies the correct edge in the backbone tree for the query sequence. Using `SCAMPP` with `pplacer` (i.e., `pplacer-SCAMPP`) and limiting the placement subtree to at most 2000 leaves allows `pplacer` to run on backbone trees with up to 200,000 leaves, achieving high accuracy.

The second approach is anecdotal: some studies (e.g., [1] and [23]) have used `FastTree-2` (commonly referred to as `FastTree`) [16] instead of `RAxML` to estimate numeric parameters on large backbone trees, followed by the use of the software package `taxtastic` [6] to reformat the numeric parameters suitably for `pplacer`. For some unknown reason, this substitution allows `pplacer` to run on larger datasets without producing negative infinity log likelihood values, as shown in [1, 23].

Despite these advances, nothing is known about the relative accuracy of `pplacer` in default mode (which means using `RAxML` for numeric parameter estimation) compared to using `pplacer` with FastTree for numeric parameter estimation (i.e., comparing `pplacer-RAxML` to `pplacer-FastTree`). In addition, very little is known about the relative accuracy of `pplacer-FastTree` compared to `pplacer-SCAMPP-RAxML`. Most importantly, 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-RAxML` to place the query sequence into the placement subtree, use `pplacer-FastTree`.

We report on an extensive study using both simulated and biological datasets comparing all these phylogenetic placement methods. Our most important finding is that `pplacer-SCAMPP-FastTree` (i.e. using pplacer within the divide-and-conquer strategy of SCAMPP

and FastTree for numeric parameter estimation) provides excellent accuracy, has low computational requirements, and can scale to ultra-large backbone trees with 200,000 leaves. Furthermore, `pplacer-SCAMPP-FastTree` allows a user-input parameter B for the placement size; the fact that `pplacer-FastTree` 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-FastTree` for the placement step.

2 Study Design

2.1 Overview

We explored six 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-RAxML` (also referred to simply as pplacer), `pplacer-FastTree`, `pplacer-SCAMPP-RAxML`, and `pplacer-SCAMPP-FastTree`. We additionally compare these methods to `EPA-ng` and `APPLES-2`. We used two collections of simulated datasets and three biological datasets to evaluate accuracy.

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).

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

The pplacer-RAxML 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-FastTree 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-RAxML 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-RAxML` 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 [23]). To place the query sequence into the subtree, `pplacer-SCAMPP-RAxML` 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-FastTree pipeline This is identical to pplacer-SCAMPP-RAxML 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-FastTree` 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-FastTree can potentially run on larger trees than pplacer, the optimal value for B may be larger than its default setting in pplacer-SCAMPP-RAxML. To place the query sequence into the subtree, `pplacer-SCAMPP-FastTree` 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.

The EPA-ng and EPA-ng-SCAMPP pipeline The EPA-ng pipeline is also a maximum likelihood approach to phylogenetic placement, but takes a significantly different approach. While other papers [12] [23] have noted that there are not significant differences between using `EPA-ng` and `pplacer`, some [23] have found that the divide-and-conquer techniques used to scale `pplacer` could improve the accuracy of the phylogenetic placements, whereas such techniques did not produce accuracy improvements when used to scale `EPA-ng` (e.g. `pplacer-SCAMPP` was more accurate than `pplacer`, whereas `EPA-ng-SCAMPP` never produced better accuracy than `EPA-ng`). The EPA-ng software recommends the use of RAxML-ng to estimate branch lengths and model parameters.

The APPLES-2 pipeline As a distance-based method, APPLES-2 differs from all other methods presented so far, which are maximum likelihood based methods. Here we evaluate APPLES-2 on the use case where we provide a FastTree backbone tree where the branch lengths were estimated using a distance-based method (minimum evolution, abbreviated to "ME"). By default, the branch lengths are re-estimated within APPLES-2, prior to placement.

2.3 Datasets

We explore the phylogenetic placement methods on four publicly available simulated datasets (`ROSE 1000M1`, `ROSE 1000M5`, `RNASim`, `nt78`), and three biological datasets (`CRW 16S.B.ALL`, `PEWO green85`, and `PEWO LTP_s128_SSU`). We have summarized the empirical properties of each dataset in Table 1. For all simulated datasets, we use the true published alignment for methods which require an extended alignment.

ROSE 1000-sequence datasets The ROSE datasets (1000M1 and 1000M5) have 1,000 sequences each and were simulated with substitutions (under the GTR+GAMMA model) with medium length indels using ROSE [21]; these were originally simulated for the SATé study [10] but have also since been used in studies of alignment accuracy [19, 18, 15] and placement accuracy [12]. 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 [10]. 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 [13]. The RNASim simulation is under a non-homogeneous biophysical fitness model which reflects selective pressures to maintain the RNA secondary structure (as opposed to models such as GTR with no selective pressures) [13]. This RNASim dataset has been widely used in studies evaluating alignment accuracy [13, 18, 19, 14] and random subsets of the RNASim dataset (RNASim-VS) have also been used to study phylogenetic placement accuracy [23, 1]. This dataset was specifically used to study `APPLES` [2], `APPLES-2` [1], `pplacerDC` [7], `pplacer-SCAMPP-RAxML`, and in this paper will be used to evaluate `pplacer-SCAMPP-RAxML` [23]. The RNASim Variable-Size (`RNASim-VS`) datasets provide true phylogenetic trees, multiple sequence alignments and estimated FastTree-2 [16] 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 [16]. We chose the first replicate to use as it has been used in other studies as well [23]. The nt78 dataset was simulated with ROSE [21] 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 [16]	78,132	1,287	simulated	.404	.639	.006	1
RNASim-50k [13]	50,000	1,620	simulated	.410	.618	.051	5
RNASim-100k [13]	100,000	1,620	simulated	.410	.618	.051	1
RNASim-200k [13]	200,000	1,620	simulated	.410	.618	.051	1
ROSE 1000M1 [9]	1,000	3,895	simulated	.697	.770	.738	5
ROSE 1000M5 [9]	1,000	1,762	simulated	.501	.607	.426	5
green85 [8]	5,088	1,486	biological	.250	.479	.146	1
LTP_s128_SSU [8]	12,953	1,598	biological	.228	.468	.090	1
16S.B.ALL [4]	27,643	6,857	biological	.210	.769	.118	1

Table 1: **Dataset Statistics:** We provide details for all the publicly available nucleotide simulated datasets and biological datasets used in this study. 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.

We used three biological datasets (Table 1), ranging in size from 5,088 to 27,643 sequences. The green85 biological dataset from the PEWO collection [8] (originally from the Greengenes database [5]) contains 5,088 aligned sequences and a corresponding reference tree. We also used the 16S.B.ALL biological dataset from the Comparative Ribosomal Website [4], with 27,643 sequences. We use an additional biological dataset, the LTP_s128_SSU dataset from the PEWO collection [8], which contains 12,953 aligned sequences and a corresponding reference tree. Since the reference trees are all estimated, we do not report placement accuracy, but do show runtime and memory usage on these datasets.

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 a normal distribution with mean as 25% of l_o and standard deviation as 60 basepairs ($\mathcal{N}(\mu = .25l_o, \sigma = 60)$). 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, which is the increase in missing branch error incurred by placing the query sequence into the backbone tree [23, 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 delta error is:

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

Note that delta error is always non-negative and that if the query sequence is placed on the same edge as in the reference tree, the delta error can still be 0. For all result plots, we present delta error (with standard error) runtime and memory consumption (with standard deviation). Where no placement tree size was specified, the entire dataset (besides the query) was used.

2.5 Experiments

- Experiment 1: We compared the behavior of `pplacer-RAxML` and `pplacer-FastTree` when placing full-length query sequences using `ROSE-1000M1`, `ROSE-1000M5`, and `nt78` datasets.
- Experiment 2: We explored `pplacer-FastTree` and `pplacer-SCAMPP-RAxML` on `nt78` and `RNASim-50k`, using both full and fragmentary query sequences.
- Experiment 3: We evaluate the impact of changing the placement tree size in `pplacer-SCAMPP-FastTree` on all datasets, and compare to `APPLES-2` and `EPA-ng-SCAMPP`.

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

Experiment 1 is summarized in Figure 1, where we explored the behavior of `pplacer-RAxML` and `pplacer-FastTree` on the simulated `ROSE` and `nt78` datasets. On the 1000-sequence ROSE 1000M1 and 1000M5 datasets (Figure 1(a) and Figure 1(b), respectively), we observe that `pplacer-RAxML` and `pplacer-FastTree` have very close delta error, with perhaps a slight advantage to pplacer-RAxML. However, `pplacer-RAxML` fails to run on the nt78 dataset ($78,132$ sequences), while `pplacer-FastTree` runs fine. Additionally, `pplacer-FastTree` and `pplacer-RAxML` have similar runtimes (with a slight advantage to `pplacer-FastTree`) 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.

An important observation here motivates the second experiment. The only difference between `pplacer-RAxML` and `pplacer-FastTree` is the software used to estimate the numeric parameters, we can see that evidently when both pipelines can be run, there is very little accuracy difference between the two methods. This is particularly surprising, as previous comparisons of `RAxML` and `FastTree` established that `RAxML` produced more accurate numeric parameter estimates than `FastTree` produced [9, 15]. Figure 1(b) and Figure 1(c) suggest that although `RAxML` may produce more accurate maximum likelihood estimates, this does not in fact help phylogenetic placement accuracy significantly. However, on the large `nt78` dataset in Figure 1(a), only the pipeline using `FastTree` numeric parameters allows `pplacer` to successfully produce placements. Thus, there is a clear advantage to using `pplacer-FastTree` on large backbone trees where `pplacer-RAxML` fails to run.

3.2 Experiment 2

Figure 2 described Experiment 2, which explored the behavior of `pplacer-FastTree` and `pplacer-SCAMPP-RAxML` on the simulated `nt78` and `RNASim-50k` datasets, using both full and fragmentary query sequences. On both datasets (Figure 2(a-b) and Figure 2(c-d)), there

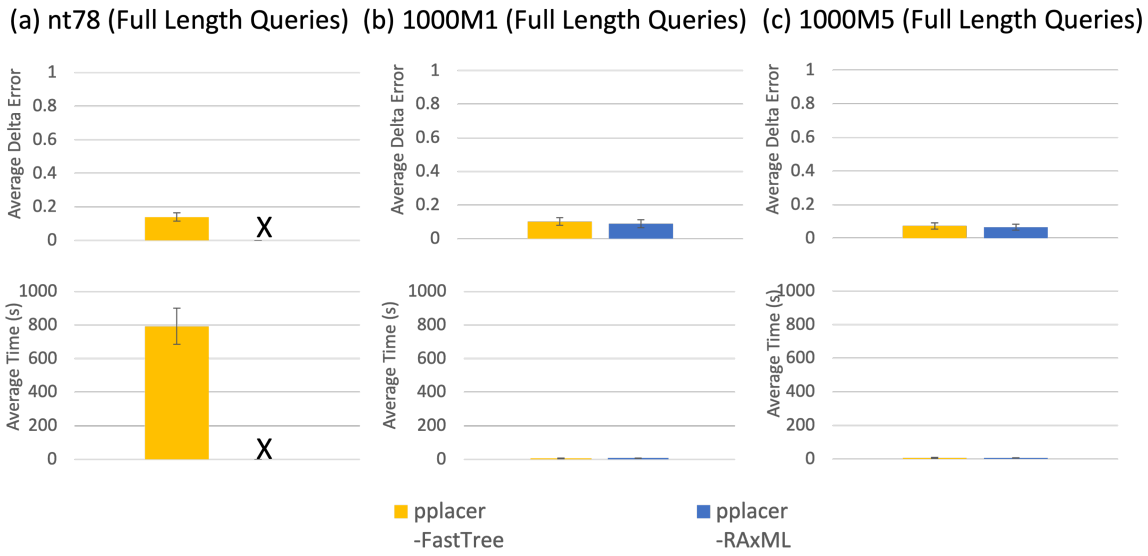

Figure 1: **(Experiment 1) Per-query comparison between pplacer-RAxML and pplacer-FastTree on the nt78 and ROSE datasets.** (a) Results on one replicate of the nt78 dataset containing $78,132$ sequences. (b) and (c) Results on five replicates each of the ROSE 1000M1 and 1000M5 datasets, containing $1,000$ sequences each. Note that the runtime and memory usage y-axes are different between the ROSE and nt78 datasets. The "X" indicates that pplacer-RAxML failed with an out of memory error on these experiments. Peak memory usage on the ROSE datasets is negligible ($<$ 1GB), so we include these results in the Supplement.

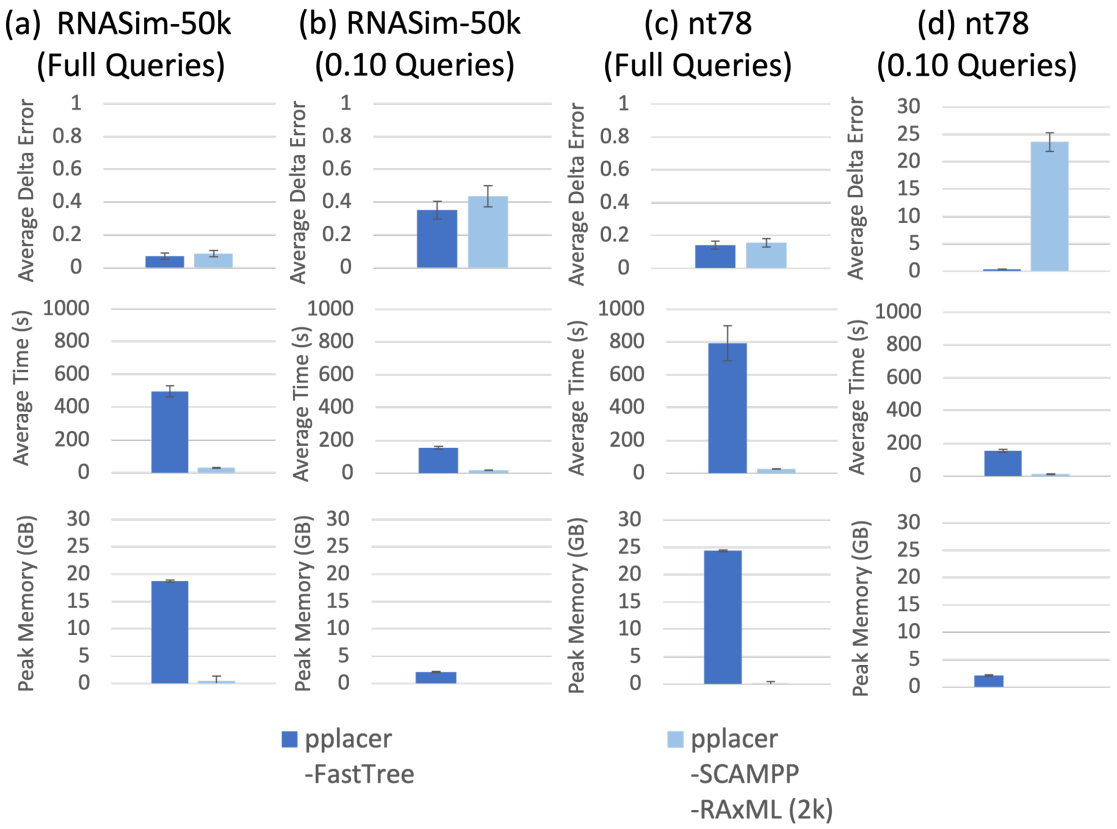

Figure 2: **(Experiment 2) Per-query comparison between pplacer-SCAMPP-RAxML and pplacer-FastTree.** Panels (a) and (b) show results on RNASim-50k (50,000 sequences) and panels (c) and (d) show results on nt78 (78,132 sequences).

were no noteworthy differences in delta error except when placing the shortest query sequences, where pplacer-FastTree had lower error than pplacer-SCAMPP-RAxML. However, both pplacer-FastTree and pplacer-SCAMPP-RAxML versions were consistently faster and used less memory than pplacer-FastTree. We do observe bigger differences between the methods on the nt78 dataset; pplacer-FastTree has much lower delta error than pplacer-SCAMPP-RAxML.

There are two key differences between pplacer-FastTree and pplacer-SCAMPP-RAxML. First, the software used for numeric parameter estimation is different; however, as we saw in Figure 1, the use of one software over the other does not significantly impact accuracy and FastTree only has an advantage on large datasets. The second difference is that pplacer-FastTree and pplacer-SCAMPP-RAxML place into differently sized trees. While pplacer-FastTree places each query sequence into the full provided backbone tree, pplacer-SCAMPP-RAxML finds a placement tree for each query sequence of size $2,000$ (by default), and places the query into this smaller placement tree. Thus, the observed difference in accuracy in Figure 2 can be attributed to the difference in the size of the tree pplacer places each query sequence into. As the placement step is the only source of phylogenetic placement error (the true alignment is used), this result suggests that placing queries into larger trees improves phylogenetic placement accuracy. This result motivates Experiment 3, where we study the effect of varying the size of the placement subtree on the accuracy and runtime trade-off.

3.3 Experiment 3

Figure 3 summarizes Experiment 3, which evaluates variants of pplacer-SCAMPP-FastTree with different placement tree sizes to pplacer-FastTree, APPLES-2, and EPA-ng-SCAMPP(2k). There are some instances where using pplacer-SCAMPP-FastTree with some large placement tree size may take additional runtime than using pplacer-FastTree directly. However, across the explored datasets, we consistently observe that pplacer-SCAMPP-FastTree consumed less memory than pplacer-FastTree. Additionally, pplacer-SCAMPP-FastTree(10k) is often as accurate as pplacer-FastTree, more accurate than pplacer-SCAMPP-RAxML and significantly faster than pplacer-FastTree. Thus, while the exact value for the placement tree size should be determined on a case-by-case basis, using pplacer-SCAMPP-FastTree is decidedly advantageous.

In Figure 3, we can see a surprising difference between pplacer-SCAMPP-RAxML(2k) and pplacer-SCAMPP-FastTree(10k) on the 0.25 and 0.10 fragment lengths. Most notably, the placement subtree size differs significantly ($2,000$ versus $10,000$ sequences in the placement tree). If there is an advantage to using the RAxML numeric parameters, it is only visible when placing the 0.10 fragmentary queries, despite the fact that pplacer-SCAMPP-FastTree(10k) can benefit from placing into a larger placement tree. However, the advantage of placing into a larger subtree size is much more important on such large datasets (nt78 contains $78,132$ sequences). That is, pplacer-SCAMPP-FastTree(10k) represents placing into approximately 13%th of the overall dataset, 25% for pplacer-SCAMPP-FastTree(20k) and 50% for pplacer-SCAMPP-FastTree(40k). Thus, pplacer-SCAMPP-FastTree has a clear advantage when flexibility between accuracy, runtime, and memory is necessary.

In Figure 3, EPA-ng failed to run on the nt78 datasets with queries of any fragment length. APPLES-2 had the highest delta error by a small margin when placing full length query sequences. However, when placing fragmentary query sequences, pplacer-FastTree and APPLES-2 had approximately the same delta error on 0.25 fragmentary queries, and APPLES-2 produced higher error than pplacer-FastTree on the 0.10 fragmentary queries.

In Figure 3, APPLES-2 achieved surprisingly low delta error when placing the fragmentary query sequences on the nt78 dataset. This contrasts with the behavior of APPLES-2 observed in previous studies [23] on the RNASim-VS dataset, where APPLES-2 always produced higher delta errors than pplacer-SCAMPP-RAxML(2k). However, the general trend is consistent; as the queries become more fragmentary, APPLES-2 delta error increases. Additionally, APPLES-2 consistently produces higher delta error than pplacer-FastTree. Therefore, this means that when placing fragmentary query sequences, either using pplacer-SCAMPP-FastTree or pplacer-FastTree on these larger placement trees may be a better choice.

Figure 3 (see also Table 1) also illustrates pplacer-SCAMPP-FastTree's behavior as the user-specified B parameter changes the placement subtree size. As the placement subtree size increases,

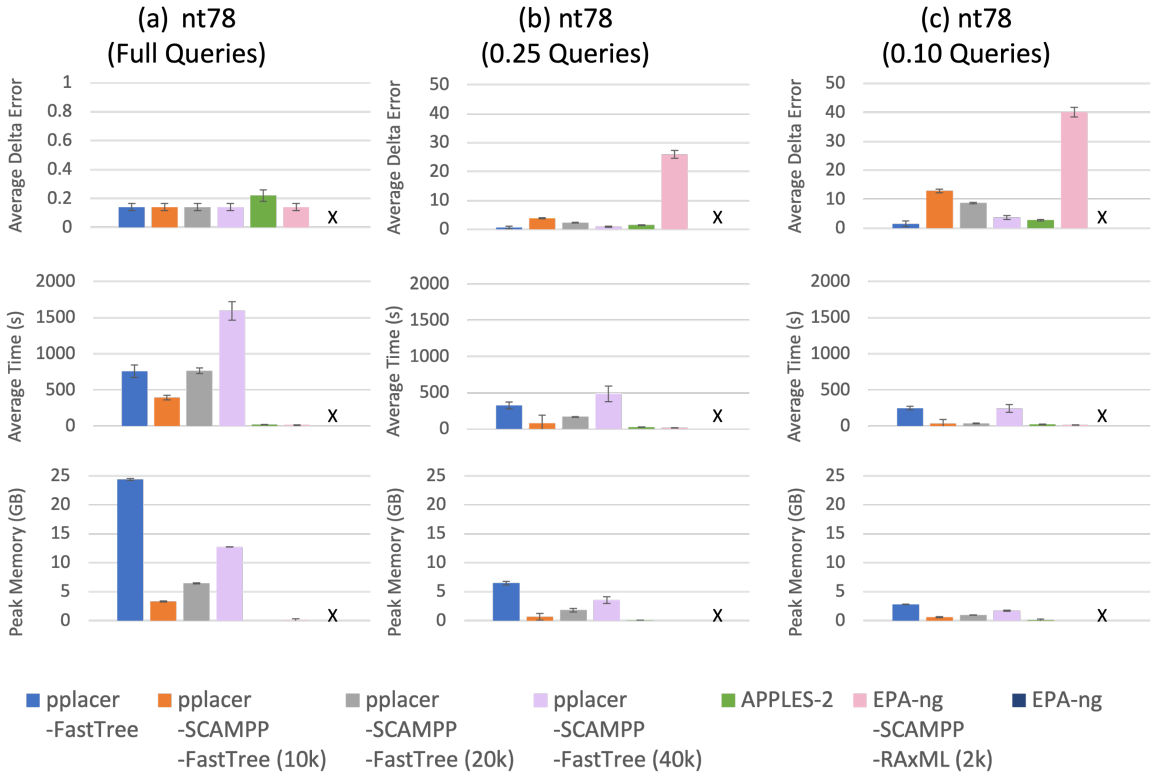

Figure 3: **(Experiment 3) Per-query results on the nt78 dataset** ($78,132$ **sequences) on full and fragmentary query sequences, comparing the pplacer variants to APPLES-2 and EPA-ng.** Panel (a) shows results on the full query sequence length. Panel (b) shows results on the 0.25 fragment length (322 basepairs) and panel (c) shows results on the 0.10 fragment length (127 basepairs). The "X" indicates that each EPA-ng query placement failed with an out-of-memory kill event and resulted in a core dump.

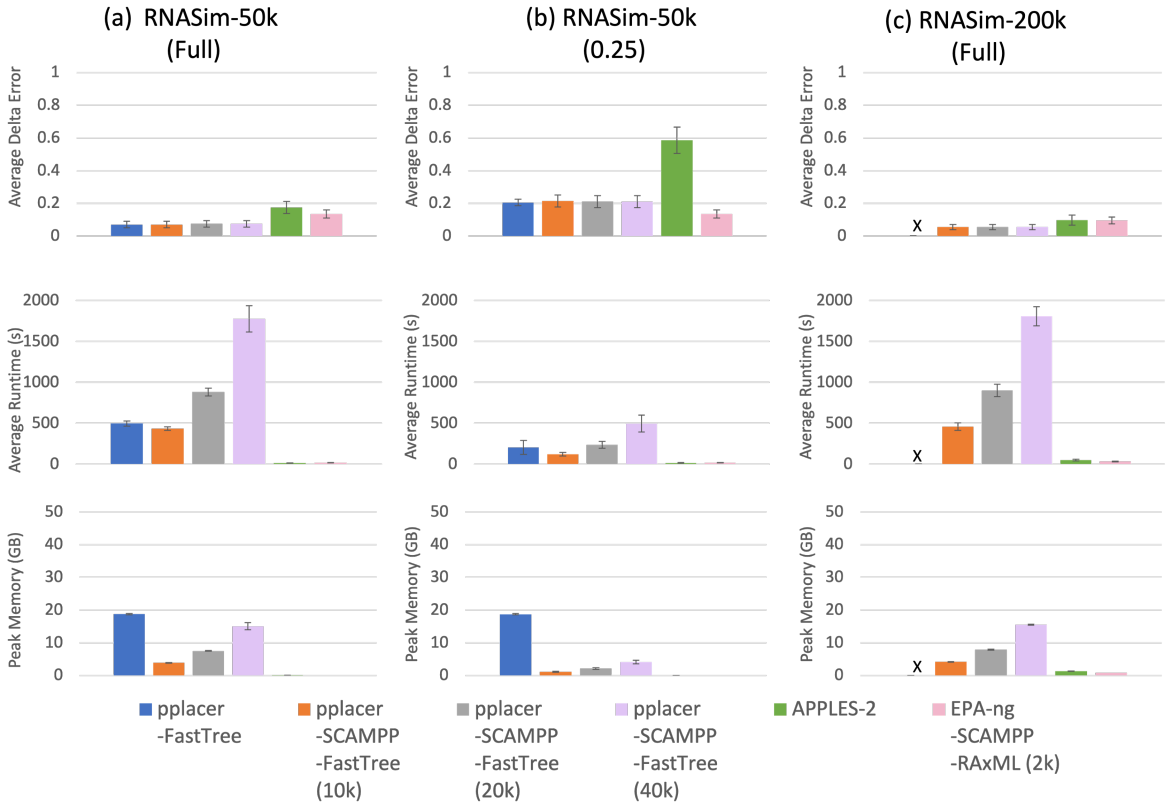

Figure 4: **(Experiment 3) Per-query comparison between pplacer-FastTree, pplacer-SCAMPP-FastTree(10k, 20k, 40k), APPLES-2, and EPA-ng-SCAMPP.** We place full length and 0.25 fragmentary query sequences into the RNASim-50k dataset and full length query sequences into the RNASim-200k datasets. We include a similar figure placing full query sequences into the RNASim-100k dataset in the Supplement.

`pplacer-SCAMPP-FastTree` incurs less delta error, but as expected, the runtime and memory consumption also increase. Furthermore, `pplacer-SCAMPP-FastTree` suffers greater delta error on the shortest fragment lengths, but the runtime and memory consumption in placing the 0.10 fragment query sequences is less than that consumed to place the longer 0.25 fragment query sequences.

In Figure 4, we explored the per-query results on `RNASim-50k` and `RNASim-200k` when placing full length and fragmentary query sequences. The key advantage of `pplacer-SCAMPP-FastTree` is the flexibility afforded by using the `FastTree` numeric parameters; the size of the placement tree can be increased significantly. As the size of the placement subtree increases from `50k` to `200k`, we observe that the average delta error for `pplacer-SCAMPP-FastTree` decreases and the runtime and memory consumption increases. Notably, on `200k`, `pplacer-FastTree` was entirely unable to run due to an out of memory error. However, even `pplacer-SCAMPP-FastTree(10k)` produces comparable delta error to `pplacer-FastTree`, and consumes significantly less runtime and memory. `APPLES-2` and `EPA-ng-SCAMPP` produce consistently worse delta error on the full length datasets. `APPLES-2`'s accuracy is most problematic on 0.25 fragmentary sequences as shown in Figure 4(b). Interestingly, `EPA-ng-SCAMPP` produces better delta error results when placing the fragmentary 0.25 query sequences.

In Figure 5 we can observe delta error results from placing full length query sequences on the biological datasets. Here we should also note that delta error here is defined as placement error

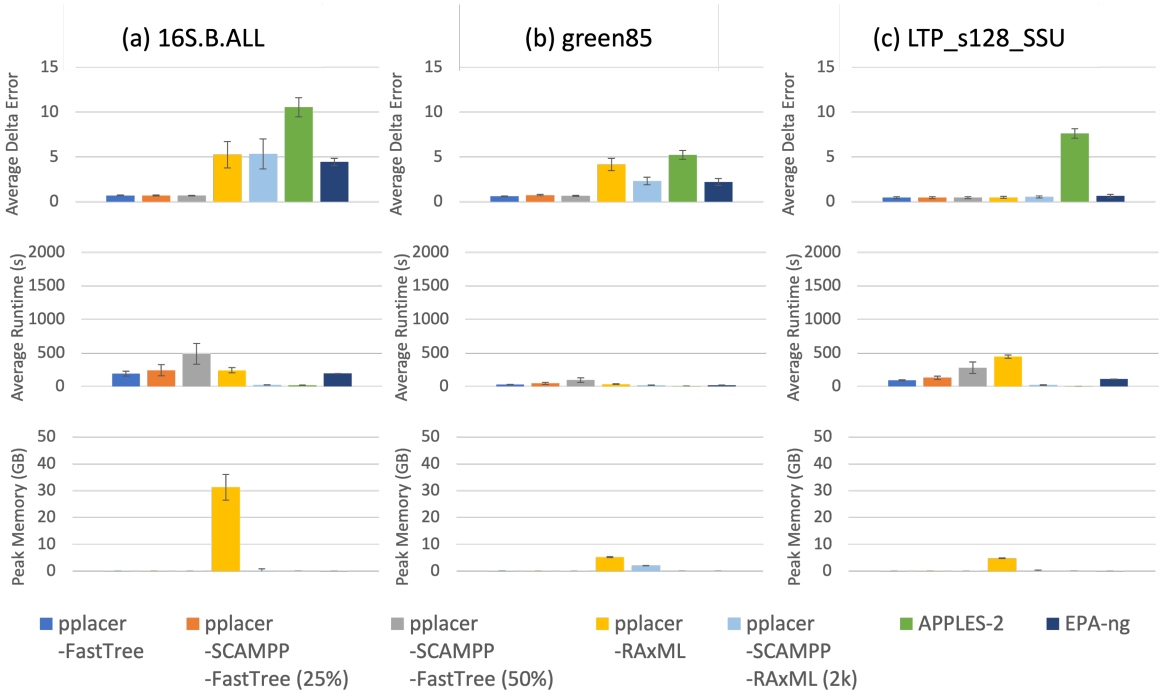

Figure 5: **(Experiment 3) Per-query results placing full length query sequences into three biological datasets.** We show delta error (with average standard error) as well as runtime and memory (with average standard deviation) on (a) 16S.B.ALL ($27,643$ sequences), (b) green85 ($5,088$ sequences), and (c) `LTP_s128_SSU` ($12,953$ sequences). The number of sequences used as the placement subtree size for `pplacer-SCAMPP-FastTree` is shown in parentheses as a percentage of the total number of sequences. We show a subset of the results for larger datasets where more placement tree sizes can be explored, and provide those results in the Supplement.

with respect to the estimated biological tree, as the true tree is not known as in simulations. Therefore, we cannot properly control for placement error attributed to the estimated tree error.

We observe that on all three explored biological datasets in Figure 5, APPLES-2 consistently produced the greatest delta error, although the margin of this error with the next best method varied across the biological datasets. In contrast, in Figure 5 pplacer-FastTree and all versions of pplacer-SCAMPP-FastTree consistently produced the lowest delta error, although with the greatest margin on the 16S.B.ALL dataset. Increasing the size of the placement tree size for pplacer-SCAMPP-FastTree had a smaller, less obvious impact on delta error. We provide further details exploring additional subtree sizes for pplacer-SCAMPP-FastTree in the Supplement.

In Figure 5, pplacer-RAxML produces the second highest delta error after APPLES-2 on two out of the three biological datasets evaluated, with EPA-ng at third. We also observe that pplacer-FastTree achieves lower delta error than pplacer-RAxML on two out of the three datasets. As previous literature evaluating RAxML and FastTree has suggested that RAxML provides better maximum likelihood estimates when both are able to run, this result is especially surprising.

We can also observe from Figure 5 that the method consuming the most runtime was pplacer-SCAMPP-FastTree(25k), whereas pplacer-FastTree and pplacer-SCAMPP-RAxML(2k) consumed much less - around the same as running pplacer-FastTree directly. This suggests that on biological datasets, especially the ones tested here which contain less than $30,000$ sequences, pplacer-FastTree is an appropriate option, achieving the least runtime and memory consumption while also achieving the best delta error. However, pplacer-SCAMPP-FastTree consistently consumes less memory than pplacer-FastTree, suggesting that especially on larger datasets where memory consumption may be a concern, pplacer-SCAMPP-FastTree may be an excellent alternative.

In Table 2 we see a summary of delta error, runtime, and memory usage of the key methods on a subset of the datasets we have run on. These results show many trends discussed already, but here we focus on the comparison between the methods that have the highest accuracy – pplacer-FastTree and pplacer-SCAMPP-FastTree (especially when using a large placement subtree size). The first observation is pplacer-FastTree fails to complete on the largest backbone tree we examine with 200K sequences, indicating a basic limitation of this method. However, for those datasets on which it is able to complete, pplacer-FastTree has the lowest delta error, with pplacer-SCAMPP-FastTree in most cases very close in delta error (especially for the large placement trees). We also see that pplacer-FastTree is at least as fast as pplacer-SCAMPP-FastTree, but uses more memory.

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 to provide pplacer with FastTree estimated numeric parameters, and the second approach is to leverage the SCAMPP framework's divide-and-conquer approach. We provided conclusive evidence that both techniques are separately 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 pplacer-FastTree). The two techniques have different impacts on placement accuracy, with FastTree numeric parameters typically not changing accuracy but SCAMPP sometimes reducing accuracy compared to pplacer-FastTree (when used with its default setting for $B = 2000$, which only allows it to place into subtrees of size 2000). This shows the benefit of being able to place into the full backbone tree (achieved by pplacer-FastTree) compared to only placing into a (relatively small) placement tree when using pplacer-SCAMPP-RAxML. However, we note that the restriction to placement subtrees with only 2000 leaves in pplacer-SCAMPP-RAxML 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-FastTree 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-RAxML on larger placement subtrees) led to the design of a combined method, pplacer-SCAMPP-FastTree, which follows the SCAMPP pipeline but replaces default pplacer by pplacer-FastTree. Further-

Delta Error	nt78			RNASim		
	nt78 (full)	nt78 (0.25)	nt78 (0.10)	50k (full)	50k (0.25)	200k (full)
pplacer-FastTree	**0.140**	**0.685**	**1.535**	**0.074**	**0.205**	X
pplacer-RAxML	X	X	X	X*	X*	X*
pplacer-SCAMPP -FastTree (10k)	**0.140**	3.885	12.885	**0.070**	**0.215**	**0.055**
pplacer-SCAMPP -FastTree (40k)	**0.140**	0.900	3.735	**0.075**	**0.210**	**0.055**
APPLES-2	0.221	1.500	2.838	0.175	0.585	0.097
EPA-ng	X	X	X	X*	X*	X*

Delta Error	biological		
	16S.B.ALL	LTP_s128_SSU	green85
pplacer-FastTree	**0.700**	**0.457**	**0.595**
pplacer-RAxML	5.250	**0.475**	4.170
pplacer-SCAMPP -FastTree (25%)	**0.680**	**0.462**	0.730
pplacer-SCAMPP -FastTree (50%)	**0.670**	**0.452**	**0.645**
APPLES-2	10.565	7.620	5.225
EPA-ng	4.430	0.640	2.205

Runtime (s)	nt78			RNASim		
	nt78 (full)	nt78 (0.25)	nt78 (0.10)	50k (full)	50k (0.25)	200k (full)
pplacer-FastTree	760.300	323.860	248.495	495.824	202.46	X
pplacer-RAxML	X	X	X	X*	X*	X*
pplacer-SCAMPP -FastTree (10k)	393.820	79.435	36.830	434.765	120.065	**457.315**
pplacer-SCAMPP -FastTree (40k)	1592.135	482.880	244.735	1775.300	493.935	1808.900
APPLES-2	**21.030**	**23.760**	**25.328**	**13.985**	**14.090**	749.540
EPA-ng	X	X	X	X*	X*	X*

Runtime (s)	biological		
	16S.B.ALL	LTP_s128_SSU	green85
pplacer-FastTree	**190.495**	91.970	29.150
pplacer-RAxML	240.339	444.066	35.566
pplacer-SCAMPP -FastTree (25%)	241.905	129.935	45.825
pplacer-SCAMPP -FastTree (50%)	484.110	279.200	97.275
APPLES-2	195.725	**0.515**	**3.480**
EPA-ng	484.110	279.200	97.275

Memory (GB)	nt78			RNASim		
	nt78 (full)	nt78 (0.25)	nt78 (0.10)	50k (full)	50k (0.25)	200k (full)
pplacer-FastTree	24.362	6.455	2.799	18.703	18.699	X
pplacer-RAxML	X	X	X	X*	X*	X*
pplacer-SCAMPP -FastTree (10k)	3.295	0.993	0.583	3.867	1.087	4.176
pplacer-SCAMPP -FastTree (40k)	12.695	3.531	1.668	15.040	4.116	15.475
APPLES-2	**0.000**	**0.018**	**0.004**	**0.011**	**0.000**	**1.271**
EPA-ng	X	X	X	X*	X*	X*

Memory (GB)	biological		
	16S.B.ALL	LTP_s128_SSU	green85
pplacer-FastTree	0.009	0.004	0.396
pplacer-RAxML	31.246	4.796	5.282
pplacer-SCAMPP -FastTree (25%)	0.003	0.001	0.341
pplacer-SCAMPP -FastTree (50%)	0.005	0.002	0.700
APPLES-2	**0.000**	**0.000**	**0.000**
EPA-ng	0.027	0.017	0.000

Table 2: **(Summary for Delta Error, Runtime, and Peak Memory Consumption):** We provide a summary of the per-query delta error (top) and runtime (bottom) results shown across the methods shown in the six figures within this paper. We show the query lengths in the row below the dataset. 'X' indicates the method failed to run on this dataset. 'X*' indicates these methods were not run but were expected to fail from previous studies. For instance, RNASim-50k and pplacer-RAxML were not run because they have already been shown to fail [23]. In the case of the ROSE datasets, these methods were not run because we are interested in evaluating the SCAMPP, APPLES, and EPA-ng variants on large datasets. The lowest delta error for each dataset column (including all ties within 0.1) is in boldface.

more, because `pplacer-FastTree` can be used with large backbone trees, this replacement allows `pplacer-SCAMPP-FastTree` to be used with larger settings for the placement subtree size B.

Our study shows that `pplacer-SCAMPP-FastTree` provides the best features of both techniques for improving scalability of pplacer, achieving the same scalability as `pplacer-SCAMPP-RAxML` (which has been shown to scale to backbone trees with 200,000 sequences, matching the scalability of APPLES-2), and matching (in some cases improving) on the placement accuracy of `pplacer-SCAMPP-RAxML` 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.

Finally, this paper also suggests that increasing B, which specifies the placement subtree size within the `pplacer-SCAMPP-FastTree` 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.

5 Competing interests

No competing interest is declared.

6 Author contributions statement

T.W. and G.C. conceived the experiments, G.C. conducted the experiments, T.W. and G.C. analysed the results, wrote and reviewed the manuscript.

7 Acknowledgments

This work was supported by funds from the National Science Foundation (# 2006069) (to TW) and by fellowship support from the National Science Foundation Graduate Research Fellowship Program (to GC). The authors would like to thank Eleanor Wedell for her helpful input as well as for providing datasets and scripts.

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] Jamie J Cannone, Sankar Subramanian, Murray N Schnare, James R Collett, Lisa M D'Souza, Yushi Du, Brian Feng, Nan Lin, Lakshmi V Madabusi, Kirsten M Müller, et al. The comparative RNA web (CRW) site: an online database of comparative sequence and structure information for ribosomal, intron, and other RNAs. *BMC bioinformatics*, 3(1):1–31, 2002.

[5] Todd Z DeSantis, Philip Hugenholtz, Neils Larsen, Mark Rojas, Eoin L Brodie, Keith Keller, Thomas Huber, Daniel Dalevi, Ping Hu, and Gary L Andersen. Greengenes, a chimera-checked 16S rRNA gene database and workbench compatible with ARB. *Applied and environmental microbiology*, 72(7):5069–5072, 2006.

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

[7] 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.

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

[9] 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.

[10] 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.

[11] 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.

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

[13] 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.

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

[15] 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.

[16] 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.

[17] 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.

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

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

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

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

[22] 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.

[23] 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.

Supplementary Materials for "SCAMPP+FastTree: Improving Scalability for Likelihood-based Phylogenetic Placement"

Gillian Chu and Tandy Warnow

Department of Computer Science, University of Illinois at Urbana-Champaign

August 8, 2022

Contents

1 Commands **2**
- 1.1 Numeric Parameters . . . 2
- 1.2 Placement Methods . . . 2
  - 1.2.1 pplacer-SCAMPP-RAxML . . . 2
  - 1.2.2 pplacer-FastTree . . . 2
  - 1.2.3 pplacer-SCAMPP-FastTree . . . 3
  - 1.2.4 EPA-ng . . . 3
  - 1.2.5 EPA-ng-SCAMPP . . . 3
  - 1.2.6 APPLES-2 . . . 3

2 Datasets **3**
- 2.1 The nt78 simulated dataset . . . 3
- 2.2 Biological Data . . . 4

3 Additional Results **4**

1 Commands

The scripts to calculate delta error were adapted from published scripts [1], and borrow utilities from `newick_utils`. All other commands have been organized into those used to estimate numeric parameters on a phylogeny and placement methods.

1.1 Numeric Parameters

Numeric parameters are estimated on a tree topology; here we define numeric parameters to include three kinds: branch lengths, substitution rate matrix (e.g. `4x4` Generalized Time Reversible), and gamma distribution for rates-across-sites. Each placement method specifies a preferred software for this step. In this paper we used existing numeric parameters where possible and estimated using `RAxML` and `FastTree` if necessary. We provide relevant commands below.

For `pplacer-SCAMPP-FastTree`, we use published `FastTree-2`-estimated numeric parameters (e.g. branch lengths, substitution rate matrix and gamma distribution) when possible, and where there are no published numeric parameters compute all three numeric parameters using `FastTree-2` with the following command:

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

For `pplacer-SCAMPP-RAxML`, we used published numeric parameters (e.g. branch lengths, substitution rate matrix and gamma distribution) [3] for all datasets besides `ROSE`, which has not been previously evaluated in this context. We provide the `RAxML` command used to estimate all three numeric parameters on the `ROSE` dataset below:

- `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 other datasets, `RAxML 7` is used to estimate all three numeric parameters for `pplacer-SCAMPP-RAxML`, with one exception. On the RNASim dataset, we used published `FastTree-2`-estimated branch lengths [3], and `RAxML 7`-estimated substitution rate matrix and gamma distribution. We use the published substitution rate matrix and gamma distribution on the `RNASim-200k` dataset [1] for all smaller subsets of the RNASim dataset, as all subsets of the RNASim dataset come from the same million-sequence tree so we expect the substitution rate matrix and gamma distribution estimated on the RNASim-100k dataset to be quite similar to the numeric parameters for RNASim-200k.

1.2 Placement Methods

1.2.1 pplacer-SCAMPP-RAxML

We used pplacer-SCAMPP-RAxML (v2.0.0, Accessed: May 10, 2022), which can be found on github at https://github.com/chry04/PLUSplacer. We used estimated branch lengths and all other numeric parameters from `RAxML 7` where possible (i.e. exception for RNASim, detailed in previous section). "jobID" is only necessary if running several jobs at once.

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

1.2.2 pplacer-FastTree

We used estimated branch lengths and all other numeric parameters from `FastTree-2`.

To run `pplacer` with `FastTree` numeric parameters, we can use the `taxtastic` software to reformat the `FastTree` info files. This builds a "reference package" which consists of a backbone alignment, corresponding phylogeny and estimated numeric parameters on that phylogeny.

- `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`

The new version of `pplacer-FastTree` (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-FastTree` not implementing this speedup on fragmentary query sequences was used.

1.2.3 pplacer-SCAMPP-FastTree

We used estimated branch lengths and all other numeric parameters from `FastTree-2`. This method can be found on github at https://github.com/gillichu/PLUSplacer-taxtastic.

- `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`

1.2.4 EPA-ng

We estimated branch lengths and all other numeric parameters using `RAxML 8`.

- `epa-ng --ref-msa refaln --tree bbtree --query qaln --outdir outdir --model infofile --redo -T 16`

This method failed on several datasets, due to the following out of memory error:
`slurmstepd: error: Detected 1 oom-kill event(s) in StepId=6152434.batch. Some of your processes may have been killed by the cgroup out-of-memory handler.`

1.2.5 EPA-ng-SCAMPP

We estimated branch lengths and all other numeric parameters using `RAxML 8`.

- `python EPA-ng-SCAMPP.py -i tinfo -t bbtree -d outdir -a aln -b subtree_size -o outjson -n n`

1.2.6 APPLES-2

We estimated branch lengths and all other numeric parameters using `FastTree-2` under minimum evolution.

- `run_apples.py -t bbtree -s refaln -q ${qaln} -T 16 -o outjson -D -X -f 0.2 -b 25`

2 Datasets

2.1 The nt78 simulated dataset

The nt78 simulation protocol also 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 $1,000$x (not 100x) and set MeanSubstitution = 0.00134" [2].

We use the published FastTree-2 results (tree and numeric parameters estimated under Minimum Evolution) on the `nt78` datasets [3]. Published datasets from [3] can be found at https://databank.illinois.edu/datasets/IDB-9257957.

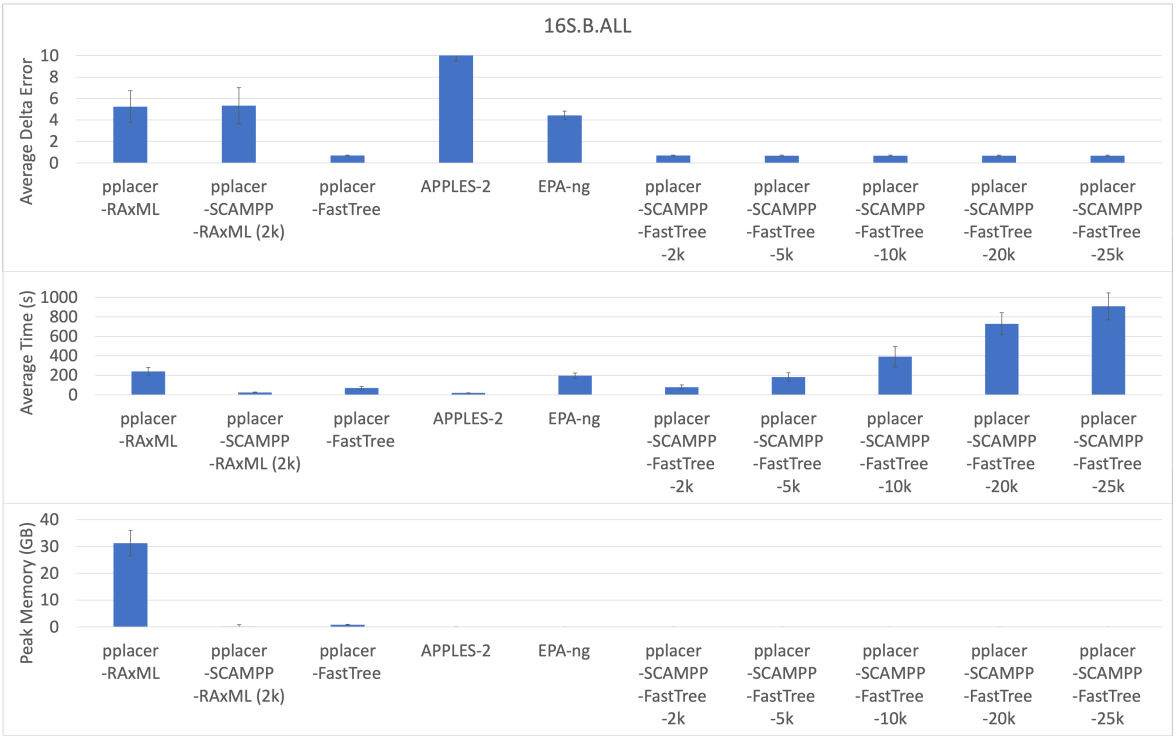

Figure 1: **(Experiment 4) Per-query results on the 16S.B.ALL (27,643 sequences) biological dataset, given full length query sequences.** The first row shows delta error results, the second row shows average time (in seconds) taken, and the third row takes peak memory (GB).

2.2 Biological Data

We provide results exploring additional placement tree sizes on the the three biological datasets below: `16S.B.ALL` is shown in Figure 1, `green85` is shown in Figure 2, and `LTP_s128_SSU` is shown in Figure 3.

3 Additional Results

We provide three tables (Table 1, Table 2, and Table 3) summarizing all methods run on the simulated datasets, with fragmentary versions provided for nt78 and RNASim. Additionally, we provide results for additional methods on the biological datasets.

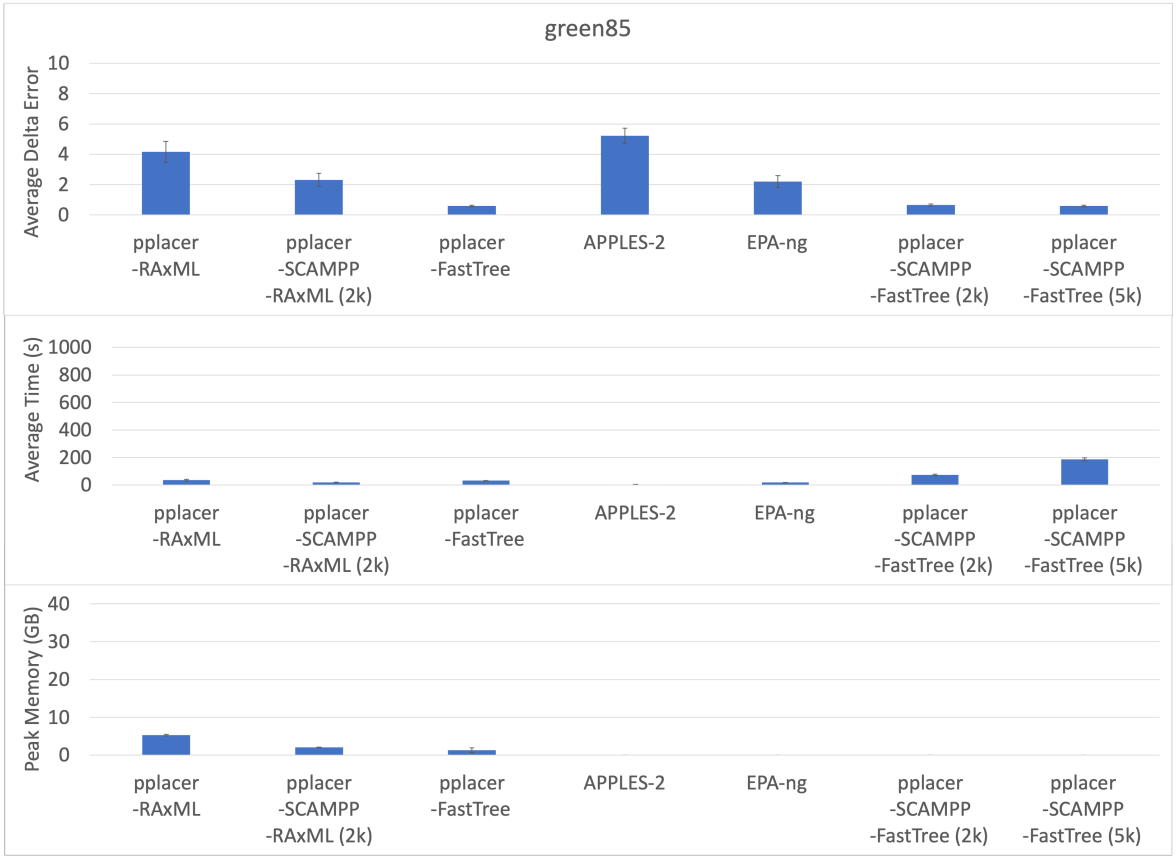

Figure 2: **(Experiment 4) Per-query results on the green85 (5,088 sequences) biological dataset, given full length query sequences.** The first row shows delta error results, the second row shows average time (in seconds) taken, and the third row takes peak memory (GB).

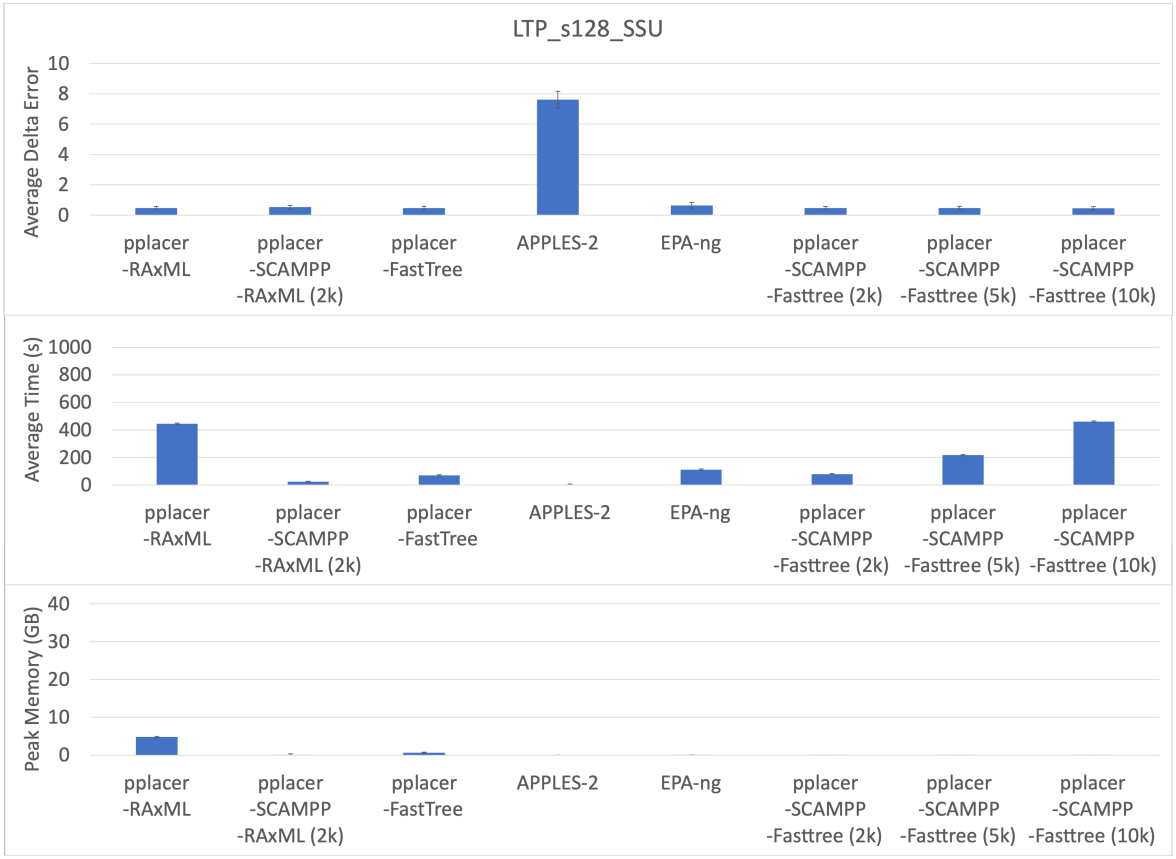

Figure 3: **(Experiment 4) Per-query results on the LTP_s128_SSU (12,953 sequences) biological dataset, given full length query sequences.** The first row shows delta error results, the second row shows average time (in seconds) taken, and the third row takes peak memory (GB).

Delta Error	nt78			RNASim								
	nt78 (full)	**nt78 (0.25)**	**nt78 (0.10)**	**50k (full)**	**50k (0.25)**	**50k (0.10)**	**100k (full)**	**100k (0.25)**	**100k (0.10)**	**200k (full)**	**200k (0.25)**	**200k (0.10)**
pplacer-FastTree	0.140	0.685	1.535	0.074	0.205	0.055	0.055	0.055	0.055	X	X	X
pplacer-RAxML	X	X	X	X*	X*	X*	X*	X*	X*	X*	X*	X*
pplacer-SCAMPP -FastTree (2k)	0.135	6.825	23.355	0.075	0.185	21.895	didn't run	didn't run	didn't run	didn't run	didn't run	didn't run
pplacer-SCAMPP -FastTree (10k)	0.140	3.885	12.885	0.070	0.215	16.875	0.065	0.135	0.084	0.055	0.150	23.235
pplacer-SCAMPP -FastTree (20k)	0.140	9.165	21.305	0.075	0.210	10.450	0.065	0.130	0.052	0.055	0.150	17.485
pplacer-SCAMPP -FastTree (40k)	0.140	0.900	3.735	0.075	0.210	4.899	0.065	0.130	0.024	0.055	0.055	10.680
pplacer-SCAMPP -RAxML (2k)	0.155	6.905	23.610	didn't run	didn't run	26.120	15.085	0.175	0.131	X	0.145	X
APPLES-2	0.221	1.500	2.838	0.175	0.585	/	0.135	didn't run	didn't run	0.097	0.485	didn't run
EPA-ng	X	X	X	X*	X*	X*	X*	X*	X*	X*	X*	X*
EPA-ng-SCAMPP	0.14	25.775	39.285	0.135	0.135	didn't run	0.16	0.16	didn't run	X	0.095	didn't run

Delta Error	biological		
	16S.B.ALL	**LTP_s128_SSU**	**green85**
pplacer-FastTree	0.700	0.457	0.595
pplacer-RAxML	5.250	0.475	4.170
pplacer-SCAMPP -FastTree (2k)	0.685	0.472	0.655
pplacer-SCAMPP -FastTree (25%)	0.680	0.462	0.730
pplacer-SCAMPP -FastTree (50%)	0.670	0.452	0.645
pplacer-SCAMPP -FastTree (5k)	0.675	0.472	0.590
pplacer-SCAMPP -FastTree (10k)	0.680	0.457	/
pplacer-SCAMPP -FastTree (20k)	0.675	/	/
pplacer-SCAMPP -FastTree (25k)	0.675	/	/
APPLES-2	10.565	7.620	5.225
EPA-ng	4.430	0.640	2.205
EPA-ng-SCAMPP	didn't run	didn't run	didn't run

Table 1: **(Summary for Delta Error):** We provide a summary of the per-query delta error within this paper. 'X' indicates the method failed to run on this dataset. In the case of the ROSE datasets, these methods were not run because we are interested in evaluating the SCAMPP, APPLES and EPA-ng variants on large datasets. In the case of the RNASim and biological datasets, some variants were not run because these results have already been discussed in a previous paper [3].

Runtime	nt78			RNASim								
	nt78 (full)	**nt78 (0.25)**	**nt78 (0.10)**	**50k (full)**	**50k (0.25)**	**50k (0.10)**	**100k (full)**	**100k (0.25)**	**100k (0.10)**	**200k (full)**	**200k (0.25)**	**200k (0.10)**
pplacer-FastTree	760.300	323.860	248.495	495.824	202.460	1260.300	1178.190	1223.115	1260.3	X	X	X
pplacer-RAxML	X	X	X	X*	X*	X*	X*	X*	X*	X*	X*	X*
pplacer-SCAMPP -FastTree (2k)	82.545	27.500	16.325	73.935	29.900	22.725	n.r. - more interested in studying the different placement sizes				didn't run	didn't run
pplacer-SCAMPP -FastTree (10k)	393.820	79.435	36.830	434.765	120.065	85.495	442.165	152.525	85.495	457.315	231.690	121.020
pplacer-SCAMPP -FastTree (20k)	765.605	239.565	120.770	881.040	235.240	146.145	880.835	278.100	146.145	900.480	354.305	186.875
pplacer-SCAMPP -FastTree (40k)	1592.135	482.880	244.735	1775.300	493.935	289.880	1786.570	538.330	289.880	1808.900	1808.900	330.955
pplacer-SCAMPP -RAxML (2k)	26.025	14.650	11.310	didn't run	didn't run	34.595	36.205	37.210	34.595	X	626.155	X
APPLES-2	21.030	23.760	25.328	13.985	14.090	/	26.145	didn't run	didn't run	749.540	53.920	didn't run
EPA-ng	X	X	X	X*	X*	X*	X*	X*	X*	X*	X*	X*
EPA-ng-SCAMPP	14.425	17.3	17.005	17.180	16.92	didn't run	22.925	23.49	didn't run	X	31.302	didn't run

Delta Error	biological		
	16S.B.ALL	**LTP_s128_SSU**	**green85**
pplacer-FastTree	190.495	91.970	29.150
pplacer-RAxML	240.339	444.066	35.566
pplacer-SCAMPP -FastTree (2k)	77.675	77.910	72.755
pplacer-SCAMPP -FastTree (25%)	241.905	129.935	45.825
pplacer-SCAMPP -FastTree (50%)	484.110	279.200	97.275
pplacer-SCAMPP -FastTree (5k)	182.800	216.765	185.775
pplacer-SCAMPP -FastTree (10k)	391.240	460.180	/
pplacer-SCAMPP -FastTree (20k)	728.655	/	/
pplacer-SCAMPP -FastTree (25k)	908.005	/	/
APPLES-2	195.725	0.515	3.480
EPA-ng	484.110	279.200	97.275
EPA-ng-SCAMPP	didn't run	didn't run	didn't run

Table 2: **(Summary for Runtime):** We provide a summary of the runtime results shown across the methods. 'X' indicates the method failed to run on this dataset. In the case of the ROSE datasets, these methods were not run because we are interested in evaluating the SCAMPP, APPLES and EPA-ng variants on large datasets. In the case of the RNASim and biological datasets, some variants were not run because these results have already been discussed in a previous paper [3].

Runtime	nt78			RNASim								
	nt78 (full)	nt78 (0.25)	nt78 (0.10)	50k (full)	50k (0.25)	50k (0.10)	100k (full)	100k (0.25)	100k (0.10)	200k (full)	200k (0.25)	200k (0.10)
pplacer-FastTree	24.362	6.455	2.799	18.703	18.699	37.531	37.52335	37.529	37.531	X	X	X
pplacer-RAxML	X	X	X	X*	X*		X*	X*	X*	X*	X*	X*
pplacer-SCAMPP -FastTree (2k)	0.696	0.090	0.000	0.792	0.138	0.002	didn't run					
pplacer-SCAMPP -FastTree (10k)	3.295	0.993	0.583	3.867	1.087	0.745	3.92065	1.211	0.745	4.176	1.482	1.006
pplacer-SCAMPP -FastTree (20k)	6.428	1.837	0.959	7.494	2.086	1.153	7.64295	2.205	1.153	7.892	2.475	1.407
pplacer-SCAMPP -FastTree (40k)	12.695	3.531	1.668	15.040	4.116	2.026	15.24215	4.187	2.026	15.475	15.475	2.271
pplacer-SCAMPP -RAxML (2k)	0.066	0.021	0.000	didn't run	didn't run	0.372	2.633395	0.556	0.372	X	4.435	X
APPLES-2	0.000	0.018	0.004	0.011	0.000	/	0.0456	didn't run	didn't run	1.271	1.170	didn't run
EPA-ng	X	X	X	X*	X*	X*	X*	X*	X*	X*	X*	X*
EPA-ng-SCAMPP	0.03225	0	0	0	0	didn't run	0.0612	0.074705	didn't run	X	0.801	didn't run

Delta Error	biological		
	16S.B.ALL	LTP_s128_SSU	green85
pplacer-FastTree	0.009	0.004	0.396
pplacer-RAxML	31.246	4.796	5.282
pplacer-SCAMPP -FastTree (2k)	0.785	0.001	0.001
pplacer-SCAMPP -FastTree (25%)	0.003	0.001	0.341
pplacer-SCAMPP -FastTree (50%)	0.005	0.002	0.700
pplacer-SCAMPP -FastTree (5k)	1.870	0.002	0.001
pplacer-SCAMPP -FastTree (10k)	3.602	0.003	/
pplacer-SCAMPP -FastTree (20k)	6.963	/	/
pplacer-SCAMPP -FastTree (25k)	8.648	/	/
APPLES-2	0.000	0.000	0.000
EPA-ng	0.027	0.017	0.000
EPA-ng-SCAMPP	didn't run	didn't run	didn't run

Table 3: **(Summary for Memory):** We provide a summary of the memory results shown across the methods. 'X' indicates the method failed to run on this dataset. In the case of the ROSE datasets, these methods were not run because we are interested in evaluating the SCAMPP, APPLES and EPA-ng variants on large datasets. In the case of the RNASim and biological datasets, some variants were not run because these results have already been discussed in a previous paper [3].

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] 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.

[3] 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.