SCAMPP: SCALABLE ALIGNMENT-BASED PHYLOGENETIC PLACEMENT

BY

ELEANOR WEDELL

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois Urbana-Champaign, 2022

Urbana, Illinois

Adviser:

Professor Tandy Warnow

ABSTRACT

Phylogenetic placement, the problem of placing a "query" sequence into a precomputed phylogenetic "backbone" tree, is useful for constructing large trees, performing taxon identification of newly obtained sequences, and other applications. The most accurate current methods, such as pplacer and EPA-ng, are based on maximum likelihood and require that the query sequence be provided within a multiple sequence alignment that includes the leaf sequences in the backbone tree. This approach enables high accuracy but also makes these likelihood-based methods computationally intensive on large backbone trees, and can even lead to them failing when the backbone trees are very large (e.g., having 50,000 or more leaves). We present SCAMPP (SCAlable alignMent-based Phylogenetic Placement), a technique to extend the scalability of these likelihood-based placement methods to ultra-large backbone trees. We show that pplacer-SCAMPP and EPA-ng-SCAMPP both scale well to ultra-large backbone trees (even up to 200,000 leaves), with accuracy that improves on APPLES and APPLES-2, two recently developed fast phylogenetic placement methods that scale to ultra-large datasets. EPA-ng-SCAMPP and pplacer-SCAMPP are available at `https://github.com/chry04/PLUSplacer`.

ACKNOWLEDGMENTS

This research was supported by the US National Science Foundation grants 1458652 and 2006069 (to TW), by fellowship support from the Siebel Scholars Program (to EW), and by the Grainger Foundation (to TW).

This thesis contains content that has been submitted by Eleanor Wedell, Yirong Cai, and Professor Tandy Warnow to IEEE-TCBB, and is currently under review. The submitted journal paper and work herein was initially started as a course project with Yirong Cai under Professor Tandy Warnow, and the prototype of the SCAMPP method pplacer-XR was originally published in the conference proceedings for Algorithms in Computational Biology 2021 [1].

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

Phylogenetic placement is the process of taking a sequence (called a "query sequence") and adding it into a phylogenetic tree (called the "backbone tree"). These methods are used for taxonomic identification, obtaining microbiome profiles, and biodiversity assessment [2, 3, 4, 5, 6, 7]. Furthermore, phylogenetic placement can be used to update very large phylogenies [8], where they offer a computationally feasible approach in comparison to *de novo* phylogeny estimation (which is NP-hard in most formulations).

Phylogenetic placement based on optimizing the maximum likelihood score is a natural approach, and is employed in pplacer [9], EPA [10], and EPA-ng [11] (an improved version of EPA). These likelihood-based phylogenetic placement methods have generally been found to have excellent accuracy but can be computationally intensive, due to their use of likelihood calculations. Another limitation of likelihood-based placement methods is that they depend on multiple sequence alignments, which can reduce their applicability and also increase the computational effort in using the methods.

Other phylogenetic placement methods have been developed that enable potentially greater scalability and speed. RAPPAS [12] and App-SpaM [13] both focus on placement for unaligned query sequences. RAPPAS in particular shows promise in scalability to large backbones since it uses k-mers. However, both App-SpaM and RAPPAS were reported as being less accurate than pplacer [12, 13]. APPLES [8] is a distance-based approach to phylogenetic placement that has shown particularly good scalability, including to backbone trees with up to 200,000 sequences. APPLES-2 [14], a new version of APPLES, has subsequently been developed using a divide-and-conquer strategy to substantially improve upon the accuracy and speed of APPLES while maintaining its scalability. However, although APPLES and APPLES-2 can both scale to very large backbone trees, the maximum likelihood-based placement methods provide better accuracy on those datasets on which they can run [14].

Thus, maximum likelihood phylogenetic placement methods have accuracy advantages over alternative approaches, but many studies have restricted these methods, such as pplacer, to relatively small backbone trees due to a combination of reasons, including limitations in computational resources and potentially numeric issues (see further discussion in Chapter 5. Of the other methods, APPLES-2 may be the most scalable and perhaps most accurate method, but has reduced accuracy compared to pplacer and has not been extensively studied. In particular, APPLES-2 has not been examined for accuracy in placing fragmentary sequences.

To enable pplacer, EPA-ng, and other computationally intensive likelihood-based phy-

logenetic placement methods to be used on larger backbone trees, we have developed the SCAMPP framework, which we now describe. Rather than attempting to find the best location in the entire backbone tree into which we insert the query sequence, the SCAMPP framework uses an informed strategy to select a subtree of the backbone tree, places the query sequence into that subtree using the selected placement method, and then identifies the correct location in the backbone tree associated with that location. Using the SCAMPP framework with pplacer yields pplacer-SCAMPP and similarly, EPA-ng yields EPA-ng-SCAMPP, but any standard phylogenetic placement method that uses aligned sequences can be used within SCAMPP. The SCAMPP framework thus extends the provided phylogenetic placement method to enable it to scale to larger backbone trees and does not change the method when the backbone tree is small enough. This approach to phylogenetic placement focuses on placement locally within a subtree of the backbone tree, rather than searching the entire backbone tree for where to place the query sequence.

Our experimental study, using both biological and simulated datasets, shows that the SCAMPP framework enables pplacer and EPA-ng to be used with large backbone trees and maintains their accuracy on those datasets on which the placement methods can run, while reducing runtime and peak memory usage. A particular outcome of our study is that EPA-ng-SCAMPP and pplacer-SCAMPP can place into backbone trees with 200,000 leaves with accuracy that improves on APPLES-2, the prior leading method for phylogenetic placement on large backbone trees. Furthermore, although APPLES-2 remains generally the fastest of these methods, the difference in running time between pplacer-SCAMPP and APPLES-2 is relatively small on the largest datasets, and pplacer-SCAMPP is faster than APPLES-2 when placing fragmentary sequences into the largest backbone trees. Thus, the SCAMPP framework not only enables alignment-based phylogenetic placement methods to scale gracefully to large datasets, but its use with pplacer provides the best accuracy of all the existing phylogenetic placement methods we explore.

CHAPTER 2: THE SCAMPP FRAMEWORK

2.1 OVERVIEW

The SCAMPP framework is designed to work with a provided phylogenetic placement method Φ, under the following basic assumptions about Φ. The input to Φ is (a) T, the "backbone tree", which is an unrooted binary tree with numeric parameters (including branch lengths) for its specified model of sequence evolution, (b) a set Q of query sequences, and (c) a multiple sequence alignment of the sequences at the leaves of the tree and the query sequences.

For each query sequence, Φ returns an output jplace file [15], consisting of multiple possible placement edges within the tree, each with a corresponding distal length, likelihood weight ratio, likelihood, and pendant branch length. The output can be used to identify a single edge into which the query sequence should be placed, as well as to produce support statistics about edge placements. The statistical support values are useful for metagenomic taxon identification and abundance profiling (e.g., as used in TIPP [4] and TIPP2 [5]). However, the output of the single best placement is also relevant when using phylogenetic placement for the purpose of incrementally building a large tree, as discussed in [8].

In this study, we focus on the use of phylogenetic placement to identify a single best edge within the backbone tree for a single query sequence; this is an application that can be used both for adding sequences into very large trees (e.g., incrementally building a gene tree) as well as for taxon identification.

Here we describe the SCAMPP framework for use with any given phylogenetic placement method Φ, when placing a single query sequence from Q; we note that inserting all the sequences in Q can be performed independently, and so this description will suffice to define the framework. We also describe the SCAMPP framework for the Generalized Time Reversible (GTR) [16] model for nucleotide evolution with gamma-distributed rates across sites, noting that modifications to this approach for other models (e.g., protein models) is trivial. The input to the SCAMPP framework has two algorithmic parameters, which are ϕ (the phylogenetic placement method) and B, the maximum size for the placement subtree. The remaining parameters are the usual ones given to likelihood-based placement methods, and are:

- T, an unrooted tree with numeric substitution model (e.g., GTR) parameters (e.g., branch lengths, substitution rate matrix, stationary distribution, gamma distribution), with S the set of sequences labelling the leaves of T

- q: the query sequence to be inserted into T
- A: the multiple sequence alignment on $S \cup \{q\}$

When we use SCAMPP with Φ, we refer to the combination as Φ-SCAMPP; hence, EPA-ng-SCAMPP refers to using SCAMPP with the EPA-ng phylogenetic placement method, pplacer-SCAMPP refers to using SCAMPP with pplacer, etc. At a high level, our three-stage technique for Φ-SCAMPP operates as follows (see Figure 2.1):

- Stage 1: A subtree T' of T is identified (defined by its set S' of leaves), with the restriction that T' cannot contain more than B leaves. This is referred to as the "placement tree".
- Stage 2: We apply Φ to T'; this returns a jplace file with the set of the edges selected by Φ for having good likelihood scores for the query sequence.
- Stage 3: For each edge e' in the jplace file, we find the associated edge e in T.

The output is therefore a jplace file containing all the potential placement edges and their associated likelihood scores. We study the SCAMPP framework in the context of finding the single best placement, but the output can be used more generally.

In what follows, we will assume that the backbone tree has n leaves and that the sequence alignment has length k.

2.2 STAGE 1

The input to Stage 1 includes the value for B, which defines the size of the placement subtree, as well as the backbone tree T. Note that if the backbone tree is small enough (i.e., has at most B leaves), then the SCAMPP framework just defaults to the selected phylogenetic placement method; hence this algorithm only applies when the backbone tree has more than B leaves.

The first stage needs to select the subtree of B leaves into which to place the query sequence. The first step is to find a closest leaf l (defined by the Hamming distance, which is number of sites where the two sequences are different (i.e., both have different letters or one is gapped and the other is not). This is modified when the query sequence is identified as a fragment by the user, in which case the calculation is performed after removing the leading and trailing gaps. This calculation takes $O(nk)$ time. We call this closest leaf the "nearest taxon" to the query sequence.

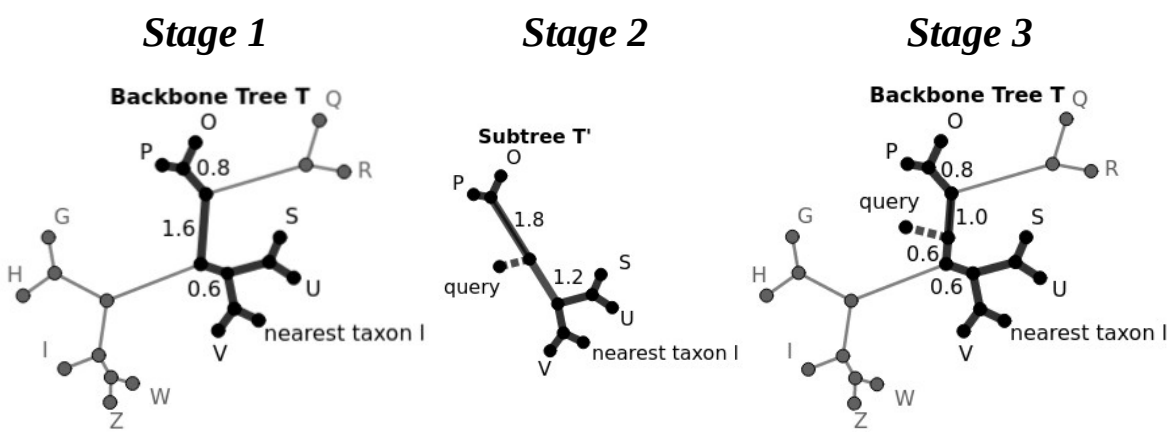


Figure 2.1: Description of the SCAMPP technique. In Stage 1, we select the placement subtree T' from the backbone tree T, for a specified query sequence. To find the placement subtree T' of T, we first find the leaf l with the smallest Hamming distance to the query sequence (called the "nearest taxon"). Then, we greedily pick the $B-1$ leaves (here $B=6$) with the smallest distance to l. In this case, we select five leaves O,P,S,U,V, and the placement subtree T' is induced by the set $\{P,O,S,U,V,l\}$ of six leaves. Here we show the given placement method selecting an edge in T' separating leaves $\{P,O\}$ from $\{S,U,V,l\}$, and this single edge in T' corresponds to a path of three edges in T. Note that a viable phylogenetic placement method for the SCAMPP framework returns not only which edge in the placement subtree to insert the query sequence into, but the branch lengths on either side; this is used to find the correct placement of the query sequence in Stage 3.

Once the leaf l is found, we select the $B-1$ leaves in order of their path distance to l, as we now define. The path distance in T from a given leaf l' to l is $\sum_{e_i \in P} L(e_i)$ where P is the path in T from l to l' and $L(e_i)$ is the length of the edge e_i in P. Starting from l, we use a breadth-first search to select those leaves in T that have the lowest path distance to l until we select the $B-1$ additional leaves (thus forming the set of B leaves, after we add l). Once the set of B leaves is identified, the induced subtree T' is returned, with the branch lengths in T' computed by using the associated branch lengths in T (note that this subtree T' may not be a clade in T).

Stage 1 takes $O(nk)$ time, and returns a set of B leaves and the induced subtree T' (with its associated numeric parameters, induced on it by the backbone tree), which is the placement tree passed to a phylogenetic placement method in Stage 2. Therefore Stage 1 is polynomial time. Moreover it is computationally efficient. However the runtime and the accuracy depends on the value of B, which we evaluate in this study. As we will see later there is a wide rage of values for B that work quite well.

2.3 STAGE 2

We then run the given phylogenetic placement method on the placement tree T' we obtain from Stage 1. This identifies a collection of edges, each of which has a good likelihood score.

2.4 STAGE 3

For each edge e' found in Stage 2, we find the single edge e in T corresponding to that edge. To do this, we first determine the set of edges in T that define the same bipartition as e'. This set will either be a single edge e or will define a path of two or more edges in T. Figure 2.1 shows such an example of how an edge e' in the placement subtree T' corresponds to a path with more than a single edge from the given backbone tree T. We let $Path(e')$ denote the edge or path in T corresponding to e', noting that a single edge is also a path (albeit of length 1). To determine $Path(e')$ given e', note that e' defines a bipartition $\pi(e')$ on T'. At least one, and possibly more than one, of the edges in T define bipartitions that correspond to $\pi(e)$ (meaning specifically that they induce the same bipartition when restricted to the leafset of T'). The set of edges in T that define bipartitions corresponding to $\pi(e')$ form either a single edge or a path of two or more edges, and so defines $Path(e')$. We then set $L(e')$ (i.e., the length of edge e') to be $L(Path(e'))$, where $L(Path(e'))$ is the sum of the branch lengths in the path (or edge) in T denoted by $Path(e')$.

If e' corresponds to a single edge e in T, then we place the query sequence into that edge. However, if e' corresponds to a path with two or more edges in T, then we use the distances we obtained to find the correct placement edge for the query sequence, as we now describe and also show in Figure 2.1.

Recall that the tree T' is a subtree of T formed by specifying a set of leaves, and that the edges of T' have branch lengths that correspond to the branch lengths in T. Recall also that when a phylogenetic placement method inserts the query sequence into e' in T', it also subdivides the edge e' and specifies how the branch length is divided. For example, suppose $e' = (a, b)$ is an edge in T' with length $L(e')$ and the query sequence is attached to this edge. Then the given phylogenetic placement method subdivides the edge e', thus creating two new edges (a, v) and (v, b), whose lengths add up to $L(e')$. We then use those new lengths to determine exactly what edge in T we should insert the query sequence into and where in that edge we should create a new node (to which we attach the query sequence) so as to produce the lengths specified by the phylogenetic placement method. An example of this is provided in Figure 2.1, and another more complex example is provided in Appendix A.

CHAPTER 3: EXPERIMENTAL STUDY

3.1 OVERVIEW

Recall that SCAMPP has two algorithmic parameters: the phylogenetic placement method Φ and the value for B, which is the maximum size (i.e., number of leaves) of the placement subtree. In our first experiment we explore how to set B within SCAMPP for use with Φ being either pplacer or EPA-ng. After selecting B, we use that value in all subsequent experiments. The second experiment compares pplacer-SCAMPP and EPA-ng-SCAMPP to other phylogenetic placement methods on backbone trees with up to 78,000 leaves, also for placing full-length sequences. The third experiment explores larger backbone trees with up to 200,000 leaves, and explores placement of full-length as well as fragmentary sequences. All methods were evaluated with respect to delta error [8, 17] (a measure of how much tree error increases by adding a query sequence) as well as running time and peak memory usage within a leave-one-out experiment.

All our analyses were performed in the same computational infrastructure (the Campus Cluster at the University of Illinois), which provides 64GB of memory, 18 CPUs, and up to 4 hours of runtime. Additional details of the methods, including version numbers and commands, are provided in Section 3.6.

3.2 METHODS

We evaluate EPA-ng-SCAMPP (v1.0.0) and pplacer-SCAMPP (v1.1.0) in comparison to APPLES (v1.1.3), APPLES-2 (v2.2.0), EPA-ng (v0.3.8), and pplacer (v1.1.alpha19).

3.3 DATASETS

3.3.1 Overview

For our experiments we use five nucleotide datasets, with three biological and two simulated (Table 3.1). These datasets have alignments and reference trees (true alignments and true trees for the simulated datasets and estimated alignments and trees for the biological datasets) that range in size from roughly 5000 sequences to as large as 200,000 sequences. We use the two smallest datasets, both biological datasets from the PEWO collection [18], for Experiment 1, where we set the algorithmic parameter B (which determines the size of the subtree used in the SCAMPP framework); the other datasets are used in Experiments

2 and 3 for evaluating the impact of the SCAMPP framework. We also made versions of these datasets where the sequences are fragmentary. All datasets are available in public repositories, with locations provided at https://tandy.cs.illinois.edu/datasets.html. We now describe these datasets in greater detail.

Table 3.1: Dataset Statistics— The first column gives the name of the dataset and the publication describing the dataset. For each dataset we show the number of sequences, the length of the reference alignment, its type (biological or simulated), the mean and maximum p-distance (i.e., normalized Hamming distances) between pairs of sequences in the alignment, and the proportion of the alignment that is gapped.

Dataset	# of seqs.	alignment length	Type	p-distance mean	p-distance maximum	gaps prop.
green85 [18]	5088	1486	biological	.250	.479	.146
LTP_s128_SSU [18]	12,953	1598	biological	.228	.468	.090
16S.B.ALL [19]	27,643	6857	biological	.210	.769	.118
nt78 [20]	78,132	1287	simulated	.404	.639	.006
RNASim [21]	200,000	1620	simulated	.410	.618	.051

3.3.2 Biological Datasets

We have three biological datasets, two from the **PEWO** [18] collection and one from the Comparative Ribosomal Website [19]. The first PEWO dataset is the green85, originally from the Greengenes database [22], of 5088 aligned sequences and a reference tree that was computed on the alignment. The second PEWO dataset is LTP_s128_SSU, which contains 12,953 aligned sequences and a reference tree originally from [23, 24]. The final biological dataset is **16S.B.ALL**, which contains 27,643 sequences with an alignment based on secondary structure [19] and a RAxML [25] maximum likelihood tree [25].

3.3.3 Simulated Datasets

We have two collections of simulated datasets. The first is the **nt78** dataset, which contains 78,132 nucleotide sequences. This simulated dataset was created to evaluate the maximum likelihood method, FastTree 2 [26]. This dataset contains 20 simulated replicates, and we arbitrarily chose the first for this study. We generate an estimated tree for phylogenetic placement using FastTree 2 for this study.

The second collection comes from the **RNASim** dataset, which is a simulated dataset with ten replicates, each containing 1,000,000 sequences. The RNASim dataset is the result of a simulation where sequences evolve under a complex biophysical model that reflects selective pressures to maintain the RNA secondary structure. RNASim has been used in other studies to evaluate alignment accuracy [21, 27, 28, 29]. The RNASim Variable Size (RNASim-VS) datasets are subsets of varying sizes (up to 200,000 sequences), drawn at random from the million-sequence RNASim dataset. These RNASim-VS datasets were used in [14] to evaluate phylogenetic placement methods, and provide true phylogenetic tree, true multiple sequence alignment, and estimated maximum likelihood (ML) trees (obtained using FastTree 2 [26]) on each subset, which serve as the backbone trees. For each backbone tree size there are five replicates included in [8] (except for the largest which contains only one), and 200 query sequences per replicate.

3.3.4 Fragmentary Datasets

For Experiment 3, we created fragmentary versions of the RNASim datasets as follows. We created "low fragmentation" (LF) conditions where a quarter of the sequences are fragmentary (mean 25% of the original length, with a standard deviation of 60 nucleotides). We picked a random starting position within the randomly selected sequence, selected a random number L from a normal distribution with mean 25% the original length and standard deviation of 60, and extracted the next L nucleotides. "High fragmentation" (HF) conditions were also simulated in a similar manner, with a mean 10% of the original length with a standard deviation of 10 nucleotides. The resulting mean fragment length is 154 for the HF conditions and 385 for the LF conditions. The true alignments of resulting sequence fragments are used for placement.

3.3.5 Backbone Trees and Numeric Parameters

The phylogenetic placement methods need backbone trees with numeric parameters (branch lengths, substitution rate matrix, stationary distribution, and gamma distribution), with specific protocols for each method. It is recommended that APPLES-2 uses branch lengths estimated under minimum evolution [8, 14], and APPLES-2 will estimate these branch lengths using FastTree-2 prior to performing placement. In order to provide fair runtime analyses we provide APPLES with a tree estimated by FastTree-2 with the no ML option for all datasets. We used the minimum evolution branch lengths provided by [8] for use with APPLES and APPLES-2 on the RNASim-VS datasets, and for all other datasets we

estimated these using FastTree-2 [20]. For EPA-ng and EPA-ng-SCAMPP, we re-estimated branch lengths for each estimated ML tree using RAxML-ng [30]. For pplacer-SCAMPP, on the RNAsim dataset we used trees with branch lengths estimated by FastTree 2 [26]. For pplacer-SCAMPP on all other datasets we estimated branch lengths using RAxML. The remaining parameters (i.e., 4×4 substitution rate matrix) across all datasets were estimated using RAxML [25] version 7. For pplacer we used the branch lengths and numeric parameters directly from RAxML version 7. However, pplacer failed to provide valid results on some large backbone trees using the numeric parameters produced by RAxML. Therefore, on those backbone trees where pplacer produced negative infinite likelihood scores using the default technique for numeric parameter estimation, we produced numeric parameters using an alternative technique recommended in [14]: we computed numerical parameters using FastTree 2 and then provided these parameters to taxit within Taxtastic [31]; this produced numeric parameters that we then used with pplacer. See Section 3.6 for additional information.

3.4 LEAVE-ONE-OUT STUDY

Our leave-one-out evaluation operates as follows. Given a backbone tree on n leaves, a random leaf is selected and removed, thus producing a reduced tree on $n - 1$ leaves. The sequence for that leaf is then added back into the reduced tree using the given phylogenetic placement method.

3.5 CRITERIA

We report running time, peak memory usage, and placement error. For running time and peak memory usage, we report results on the University of Illinois Campus Cluster. In our analyses, each phylogenetic placement was given one node with 64 Gb of memory and allowed at most 4 hours to complete. Nevertheless, the machines vary in age and speed, and running times are not exactly comparable.

We report placement error by comparing trees, before and after a single query sequence is added to the backbone tree, to the true tree (when using simulated datasets) or a reliable estimated tree (when using biological data) on the corresponding set of leaves. We will refer to the true tree or reliable estimated tree as the "reference tree". This comparison is performed by representing each tree by its set of bipartitions, noting that each edge in a tree defines a bipartition on the leafset. We define the "false negative" error (also called

the number of "missing branches") of a given tree t with respect to the reference tree to be number of edges (or bipartitions) that are in the reference tree but not in t. The change in the number of false negatives produced by placing a query sequence into a backbone tree is the "delta error" produced by the placement method.

We illustrate this calculation with an example. Suppose the backbone tree has leafset S and is missing 5 edges found in the true tree on this leafset. Now suppose we use a phylogenetic placement method to add a new sequence s' into the tree, so that the extended backbone tree now has leaves $S \cup \{s'\}$, and suppose this extended tree is missing 7 edges from the true tree on $S \cup \{s'\}$. Then the delta error is 2, since the number of edges that were missing went up by 2. Note that the delta error can never go down, since an edge that is missing before s' is added is still missing after s' is added.

We now make this concept precise using mathematical notation. Given a tree Y we let $B(Y)$ denote the set of bipartitions of Y. We let T^* denote the reference tree (i.e., either the true tree or a reliable estimated tree), and we assume T^* has leafset S. In a leave-one-out study, we are given a tree t and we delete one leaf s' from t, thus producing a tree T on leafset $S' = S \setminus \{s'\}$. Note that the tree t may not be the reference tree. When we add query sequence s' into T, we obtain a tree P. Note that P has the entire leafset S. We let $T^*|_{S'}$ denote the subtree of T^* induced by leafset S'. Then the delta error for P, denoted by $\Delta_e(P)$, is given by the following formula:

$$\Delta e(P) = |B(T^*) \backslash B(P)| - |B(T^*|_{S'}) \backslash B(T)|, \tag{3.1}$$

where $|X|$ denotes the number of elements in the set X. Thus the first term is the number of false negatives for the tree P and second term is the number of false negatives for the tree T. Note that $\Delta_e(P) \geq 0$, since the number of missing branches produced by adding a query sequence to a tree cannot decrease.

We note that several earlier studies [11, 18, 32] have used the "node distance" to evaluate phylogenetic placement methods within leave-out studies, as follows. A starting tree is given and a leaf is deleted, and then reinserted using a placement method. The distance (i.e., number of nodes) from the final placement to the placement in the starting tree is the node distance. However, this is equal to the delta error when the starting tree is interpreted as the true tree. Therefore, the delta error is an extension of the node distance that allows error in the starting tree (which can be quantified in a simulation study) to be part of the evaluation. Therefore, throughout our experiments, we use the delta error for both simulated and biological datasets, with the trees provided for the biological datasets treated as reference trees. (In other words, when we report delta error on the biological datasets, it

is the same as reporting node distance for these datasets.)

3.6 ADDITIONAL DATASET INFORMATION AND COMMANDS

3.6.1 Additional Dataset Setup

The LTP_s128_SSU dataset was first converted from RNA to DNA.

We estimated an ML tree on the first replicate of the nt78 dataset using FastTree-2 with the following command:

- FastTreeMP -nosupport -gtr -gamma -nt -log true.fasttree.log < alignment.fasta > true.fasttree

3.6.2 Numeric Parameter Estimation for Backbone Trees

For all datasets the fasta formatted alignments were converted to phylip and numeric parameters were estimated according to the specific recommendations of each phylogenetic placement method. Thus, pplacer recommends RAxML v7.2.6, EPA-ng recommends RAxML-ng, and APPLES and APPLES-2 recommend the use of balanced minimum evolution branch lengths as computed by FastTree. On one dataset (LTP_s128_SSU), pplacer failed to return valid output using the recommended techniques for calculating numeric parameters and returned -inf values; therefore, for this dataset we re-estimated the numeric parameters using FastTree v2.1.11, as described below. However, when running pplacer-SCAMPP or EPA-ng-SCAMPP, we used the software recommended by its base method. Here we provide the commands we used.

In order to place with pplacer and pplacer-SCAMPP RAxML (v7.2.6) was used to estimate branch lengths and substitution rates with the following command:

- raxmlHPC-PTHREADS -f e -t reference.nwk -m GTRGAMMA -s alignment.phylip -n REF -p 1984 -T 16

For pplacer-SCAMPP on the RNAsim VS datasets we used the FastTree branch lengths available in [33]. For pplacer-SCAMPP on all other datasets we compute RAxML (v7.2.6) provided branch lengths. All other numerical parameters (e.g. 4×4 substitution rate matrix) were from with RAxML (v7.2.6) on all datasets, we used those available in [33] for RNAsim, and computed the rest using the command above.

For EPA-ng and EPA-ng-SCAMPP, RAxML-ng (v1.0.2) was used to estimate branch lengths and model parameters with the command:

- raxml-ng –evaluate –tree reference.nwk –model GTR+G –msa alignment.phylip -seed 1984

APPLES-2 will re-estimate branch lengths under minimum evolution, however to obtain a fair runtime and memory usage analysis we handled this ahead of the placement step. So APPLES-2 and APPLES were run using a tree with re-estimated branch lengths, as suggested in [14], using the following command for large trees under minimum evolution:

- FastTreeMP -nosupport -nt -nome -noml -log fasttree_me.log -intree reference.nwk < alignment.fasta > tree_me.nwk

The only dataset for which we deviated from the above procedure of obtaining backbone trees was on LTP_s128_SSU. On this dataset, pplacer failed using the RAxML (v7.2.6) estimated branch lengths and statistics file, producing -inf likelihood scores. To resolve this, in accordance with the reported procedure in APPLES-2, we re-estimated the branch lengths with FastTree-2 and used taxtastic to build a reference package to run pplacer with.

The FastTree-2 command was:

- FastTreeMP -nt -gtr -nome -mllen -log fasttree.log -intree reference.nwk < alignment.fasta > fasttree.nwk

In order to package this with taxtastic [31] for each leave-one-out experiment, we used the following command on the FastTree-2 newick file with the query pruned and internal parent node removed:

- taxit create -P my.refpkg -l locus_name –aln-fasta ref.fa –tree-file fasttree.nwk –tree-stats fasttree.log

On the nt78 dataset, we did not follow this procedure as the failure was due to memory problems (segmentation fault) and not the -inf likelihood scores obtained in LTP_s128_SSU dataset.

3.6.3 Phylogenetic Placement Commands and Delta Error Script

The following commands were used for running each phylogenetic placement method.

- APPLES (v1.1.3)
  python run_apples.py -t backbone.tree -s ref.fa -q query.fa -T 16 -o apples.jplace

- APPLES-2 (v2.2.0)
  python run_apples.py -t backbone.tree -s ref.fa -q query.fa -T 16 -o apples-2.jplace -D -X -f 0.2 -b 25

- EPA-ng (v0.3.8)
  epa-ng –ref-msa ref.fa –tree backbone.tree –query query.fa –outdir output_dir –model RAxML.bestModel –redo -T 16

- pplacer (v1.1.alpha19)
  ./pplacer -m GTR -s RAxML_info.REF -t backbone.tree -o pplacer.jplace alignment.fasta -j 1

- pplacer (v1.1.alpha19) on LTP_s128_SSU
  ./pplacer -m GTR -c my.refpkg -o pplacer.jplace alignment.fasta -j 1

- pplacer-SCAMPP (v2.0.0)
  python pplacer-SCAMPP.py -i RAxML_info.REF -t backbone.tree -d output_dir -a msa.fa -b subtree_size

- EPA-ng-SCAMPP (Note: EPA-ng-SCAMPP is not enabled in Version 1.0.0)
  python EPA-ng-SCAMPP.py -i RAxML.bestModel -t backbone.tree -d output_dir -a msa.fa -b subtree_size

Commands used for placement of fragmentary sequences with pplacer-SCAMPP and EPA-ng-SCAMPP used the fragmentary sequences option ("-f True"), which ignores (masks) leading and trailing gaps in the alignment.

The script used for the delta error computation was originally published with the data in [34].

CHAPTER 4: RESULTS

Results for Experiments 1–3 are shown here for APPLES-2, EPA-ng, pplacer, EPA-ng-SCAMPP, and pplacer-SCAMPP. APPLES was clearly inferior to APPLES-2 with respect to runtime, memory usage, and accuracy, and the results for APPLES are thus described separately in Section 4.4.

4.1 EXPERIMENT 1: DESIGNING THE SCAMPP FRAMEWORK

An important algorithmic parameter is the size B of the subtree into which the query sequence is placed, and exploring this question is the main focus of this section. Previous studies have suggested that better placement accuracy is obtained by placing into a larger subtree [4, 17], indicating that the best placements may be obtained by increasing the value of B, which limits the placement subtree size given to the phylogenetic placement method. However, increasing the placement subtree size too much also increases the computational effort, and may also lead to failures in some cases.

In order to understand the impact of B, which determines the placement subtree size, we used two relatively small PEWO datasets (green85 with 5088 sequences and LTP_s128_SSU with 12,953 sequences) and tested a range of subtree sizes. We see (Figure 4.1) that small values for B (which limit the placement to small subtrees) produced high error, but values for B generally between 1000 and 4000 had good accuracy (with very small differences between $B = 1000$ and $B = 4000$). However, as subtree sizes increased, runtime and memory usage also increased. Based on these trends, we performed a more focused evaluation of settings for B in the range between 1000 and 4000.

In Figure 4.2, we compare pplacer-SCAMPP and EPA-ng-SCAMPP to the other phylogenetic placement methods using these three values for B to get a sense for how important it was to set B optimally. We examine the impact on placement error first, and then the impact on runtime and memory usage.

On the smaller of these two datasets (i.e., green85, with only 5088 sequences), we see that EPA-ng is slightly more accurate than EPA-ng-SCAMPP when $B = 1000$ or $B = 2000$ and then matches accuracy when $B = 4000$. This suggests that EPA-ng is able to provide a good analysis of the full dataset and that $B = 4000$ is slightly better than $B = 2000$. In contrast, for all settings of B, pplacer is clearly less accurate than pplacer-SCAMPP, and there is little difference between pplacer-SCAMPP for $B = 2000$ and $B = 4000$. This suggests that pplacer is unable to provide good accuracy on the full dataset and benefits from

restriction to a subtree, and that $B = 2000$ is as good as $B = 4000$. Results on the larger of the two datasets (i.e,, LTP_s128_SSU) are somewhat different than on the smaller dataset. First, all methods except for APPLES-2 have very low placement error, with average delta error below 1. In addition, there are very small differences the remaining methods, and changes to B on this dataset does not have much impact. Across the two datasets, setting $B = 2000$ or $B = 4000$ are both reasonable settings, with $B = 2000$ somewhat better for pplacer-SCAMPP and $B = 4000$ somewhat better for EPA-ng-SCAMPP.

A comparison of running time provides additional insights about how to set B. Specifically, increasing B increases the running time for both pplacer-SCAMPP and EPA-ng-SCAMPP, and has a larger impact on pplacer-SCAMPP than on EPA-ng-SCAMPP. In addition, pplacer has by far the highest running time (see for example runtime on the LTP_s128_SSU dataset) and EPA-ng is in second page, but setting $B = 2000$ in pplacer-SCAMPP or EPA-ng-SCAMPP greatly reduces the runtime. Furthermore, changing B from 2000 to 4000 approximately doubles the runtime for both pplacer-SCAMPP and EPA-ng-SCAMPP. Thus, B has a large impact on runtime, as expected.

The peak memory usage by pplacer-SCAMPP and EPA-ng-SCAMPP is also very impacted by the setting for B. On the green85 dataset, the lowest peak memory usage is achieved by both methods when $B = 1000$, and then increases substantially with increases in B. The highest peak memory usage is for pplacer, followed by EPA-ng in second place, and every explored setting for B reduces their peak memory usage. On the LTP_s128_SSU dataset, the same trends appear, but with the following difference: here, EPA-ng has by far the highest peak memory usage (more than three times that of every other method).

Overall, what these trends show is that the different settings for B between 1000 and 4000 result in at worst small changes to the placement error but very large changes to runtime and memory usage. If placement accuracy must be optimized, then these results suggest that the optimal setting for B when using pplacer-SCAMPP is probably 2000, but the optimal setting when using EPA-ng-SCAMPP is possibly $B = 4000$ (but $B = 2000$ produces very close results). However, the computational hit (both running time and memory usage) in changing from $B = 2000$ to $B = 4000$ is substantial for both methods. Based on these experimental results, we set $B = 2000$ for default usage with both pplacer-SCAMPP and EPA-ng-SCAMPP, and used this setting in the subsequent experiments. In addition we looked at the impact of the size of the parameter B on two additional datasets. These datasets are used for the comparison of methods, and were not considered and included when selecting a default for the parameter B. The results are shown in Section 4.5. In short those results indicate that the impact of the selection of B on those datasets does follow similar trends, and also indicate that a substantial gain in accuracy occurs once the size of

the parameter B is at least 2000.

4.2 EXPERIMENT 2: EVALUATING SCAMPP ON MODERATELY LARGE TREES

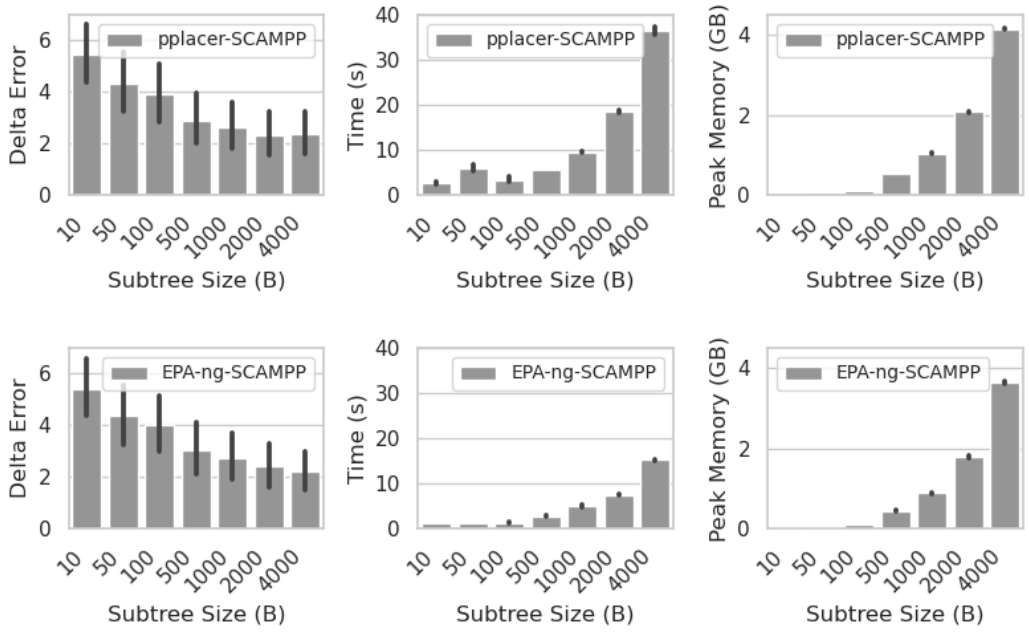
This experiment examines phylogenetic placement on two moderately large backbone trees, using the setting for B established in Experiment 1. We analyze the 16S.B.ALL biological dataset (with 27,643 sequences) and the nt78 simulated dataset (with 78,132 sequences). We see different trends for each of these datasets, and so we discuss them separately, starting with the smaller of the two datasets.

On the 16S.B.ALL dataset (Figure 4.3(a)), we see that APPLES-2 has double the delta error of the other methods. The most accurate method is EPA-ng, with delta error of 4.4, but the delta errors of the remaining methods are all between 5.3 and 5.4, and have overlapping error bars with EPA-ng. There are substantial differences in terms of running time, with APPLES-2 by far the fastest and pplacer by far the slowest. EPA-ng is the second slowest. In contrast, pplacer-SCAMPP and EPA-ng-SCAMPP (which are nearly identical in running time) are nearly as fast as APPLES-2. The methods also differ substantially with respect to memory usage, with APPLES-2 the best, followed closely by pplacer-SCAMPP and EPA-ng-SCAMPP, and then by EPA-ng and pplacer, which have about the same (large) memory usage. Specifically, SCAMPP enables a large reduction in peak memory usage for both pplacer and EPA-ng, from over 30Gb to under 3Gb on this dataset.

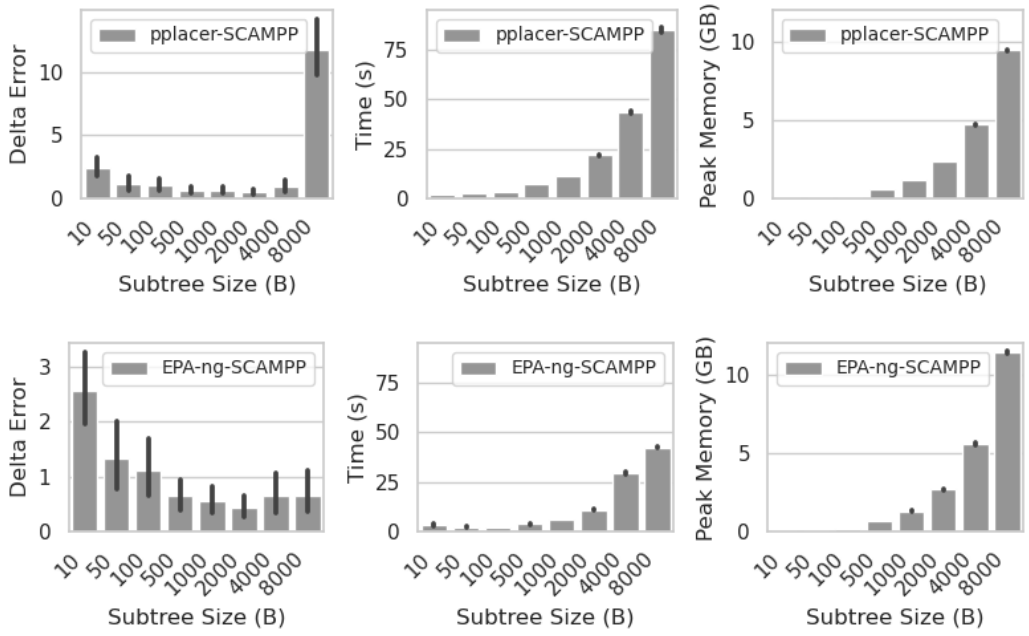
Results on the 78nt dataset (Figure 4.3(b)) show somewhat different trends. The first and most significant difference is that neither pplacer nor EPA-ng were able to perform the placements. On this dataset both pplacer and EPA-ng failed to return a jplace file due to segmentation faults (see Chapter 5). The comparison between the remaining methods shows APPLES-2 less accurate than EPA-ng-SCAMPP and pplacer-SCAMPP, and with a small advantage to EPA-ng-SCAMPP. The three methods are again distinguishable in terms of runtime and memory usage, with APPLES-2 the fastest and using the least memory. A comparison between EPA-ng-SCAMPP and pplacer-SCAMPP shows pplacer-SCAMPP slower than EPA-ng-SCAMPP but using less memory.

4.3 EXPERIMENT 3: EVALUATING SCAMPP ON VERY LARGE TREES

Here we explore performance of phylogenetic placement methods when the backbone trees are very large, using the RNASim-VS datasets with backbone trees ranging from 50K to 200K leaves. There are five replicates each for trees with 50K and 100K leaves and only one

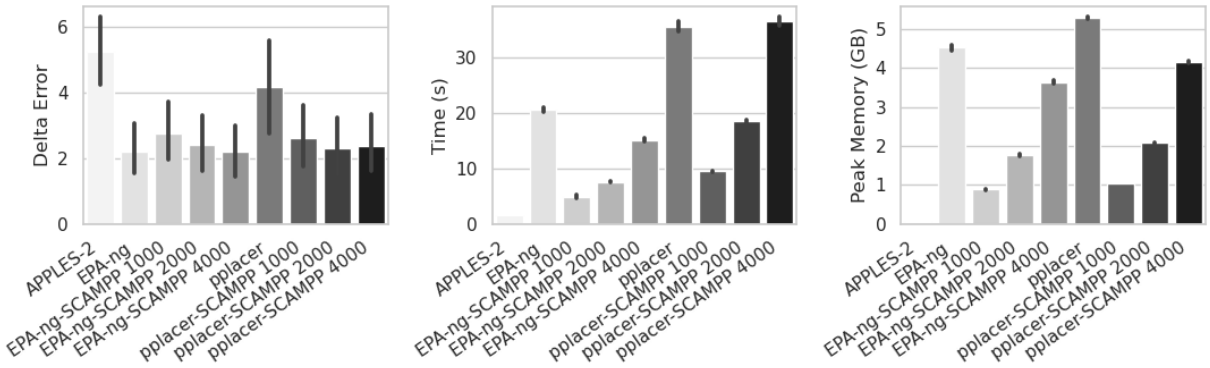


(a) green85 (5088 sequences)

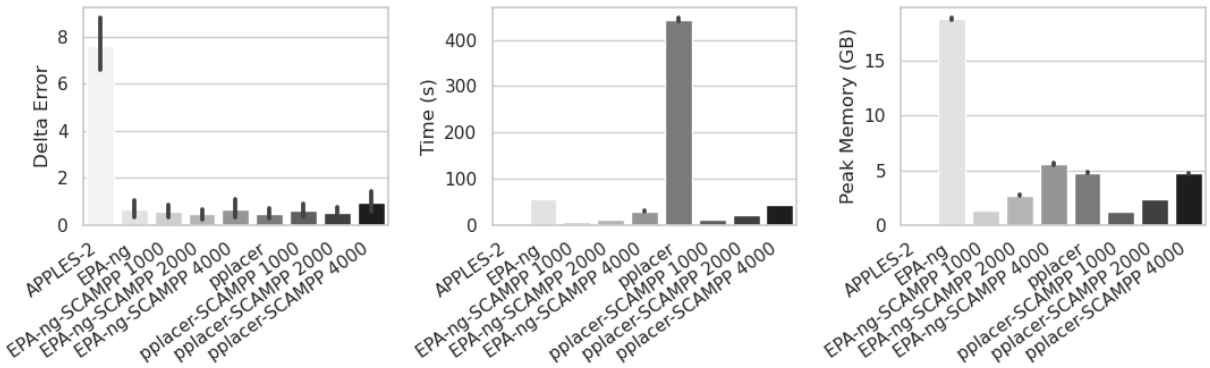


(b) LTP_s128_SSU (12,953 sequences)

Figure 4.1: Experiment 1: Exploring the impact of how B is set, which specifies the placement subtree size, on two PEWO biological datasets (LTP_s128_SSU and green85). Within each row the subfigures for pplacer-SCAMPP and EPA-ng-SCAMPP show: Mean delta error (left), Mean time in seconds (center), and Mean peak memory usage in Gb (right). Rows (from top to bottom) show results on green85 and LTP_s128_SSU.



(a) green85 (5088 sequences)



(b) LTP_s128_SSU dataset (12,953 sequences)

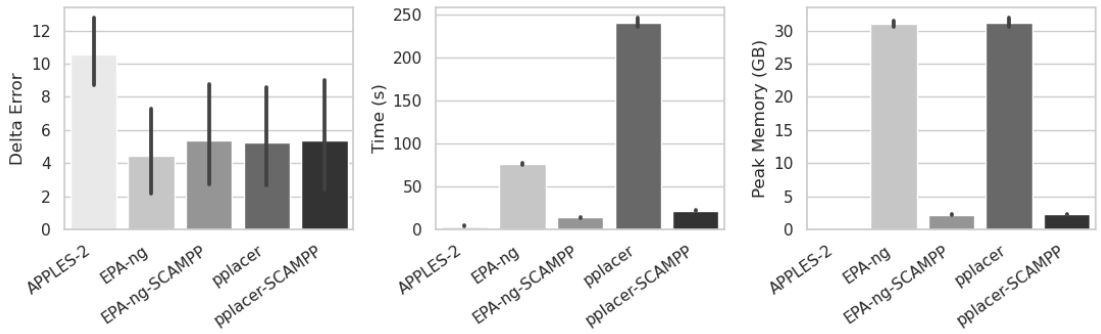
Figure 4.2: Experiment 1: Results of phylogenetic placement methods on the two PEWO datasets (green85 and LTP_s128_SSU), using $B = 1000$, $B = 2000$, and $B = 4000$. Within each row the subfigures show: Mean delta error (left), Mean time in seconds (center), and Mean peak memory usage in Gb (right).

replicate with a tree of 200K leaves. For this study, we do not use either pplacer or EPA-ng, as they fail to complete on the nt78 dataset, as shown in Experiment 2.

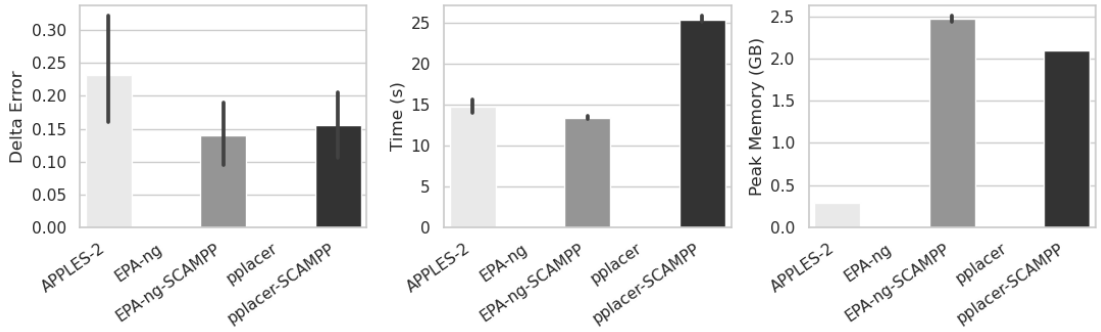
4.3.1 Experiment 3a: Scalability in Placing Full-Length Sequences

Placement error results on these data present interesting trends (Figure 4.5(c)). On the backbone trees with 50,000 leaves, pplacer-SCAMPP has the lowest placement error, followed by EPA-ng-SCAMPP, and then by APPLES-2. On the 100,000-leaf backbone trees, pplacer-SCAMPP again has the lowest error, and APPLES-2 and EPA-ng-SCAMPP have the same higher error. Results on the 200,000-leaf backbone tree show the same relative trends as on the 100,000-leaf backbone trees, but error rates have dropped somewhat for all methods.

A comparison of methods with respect to running time and memory usage is also in-



(a) 16S.B.ALL (27,643 sequences)



(b) nt78 (78,132 sequences)

Figure 4.3: Experiment 2: Results of phylogenetic placement methods on moderately large backbone trees. Within each row the subfigures show: Mean delta error (left), Mean time in seconds (center), and Mean peak memory usage in Gb (right). Rows (from top to bottom) show results on 16S.B.ALL and nt78. On the nt78 dataset, we do not show pplacer and EPA-ng because both fail to return valid results.

teresting (Figure 4.5(c)). APPLES-2 is clearly the fastest of the three methods, followed by EPA-ng-SCAMPP and then by pplacer-SCAMPP. Furthermore, EPA-ng-SCAMPP and APPLES-2 are not very far apart in terms of runtime on the 100,000-leaf tree and then identical in running time on the largest tree with 200,000 leaves. Memory usage also clearly favors APPLES-2, and the differences between EPA-ng-SCAMPP and pplacer-SCAMPP are very small.

4.3.2 Experiment 3b: Scalability of Fragmentary Sequence Placement

We examined two lengths for the fragmentary sequences: short fragments, averaging 154 nucleotides, and slightly longer fragments, averaging 385 nucleotides. We refer to the shorter sequence condition as HF (high fragmentary) and the slightly longer fragments as LF (low fragmentary).

Results on the short fragments show very clear trends (Figure 4.5(a)). First, APPLES-2 has substantially higher delta error than pplacer-SCAMPP and EPA-ng-SCAMPP for all backbone tree sizes, and the difference between pplacer-SCAMPP and EPA-ng-SCAMPP is very small. All three methods have essentially the same running time for backbone tree size 50,000 but differences appear as the backbone tree size increases so that APPLES-2 becomes the slowest of the three methods. EPA-ng-SCAMPP and pplacer-SCAMPP have close runtimes, with pplacer-SCAMPP slightly slower than EPA-ng-SCAMPP on backbone tree size 100,000 and then faster on backbone tree size 200,000. APPLES-2 and pplacer-SCAMPP both have relatively low peak memory usage at all sizes (though APPLES-2 uses more peak memory than pplacer-SCAMPP on the larger backbone trees), and EPA-ng-SCAMPP has by far the highest peak memory usage.

Results on the longer fragments show very similar trends, but with a few differences (Figure 4.5(b)). As with the short fragments, APPLES-2 is the least accurate, and differences between pplacer-SCAMPP and EPA-ng-SCAMPP are minor (though there is a small advantage to pplacer-SCAMPP over EPA-ng-SCAMPP on the largest backbone size). The same basic trends hold for running time, except that pplacer-SCAMPP is the slowest of the three methods until the largest backbone size, where it is faster than APPLES-2 but slightly slower than EPA-ng-SCAMPP. Peak memory usage is also slightly different, but EPA-ng-SCAMPP is still by far the most memory-intensive.

Some other trends are also worth noting. First, delta error rates drop with increases in the backbone tree size, while runtime increases. The increase in runtime is expected, but the decrease in delta error is surprising, and worth further investigation. Interestingly, peak memory usage is fairly constant as the backbone tree size increase for EPA-ng-SCAMPP, but grows for pplacer-SCAMPP and for APPLES. The impact of backbone tree size on peak memory usage for APPLES-2 follows from its algorithm design, but the differential impact on pplacer-SCAMPP and EPA-ng-SCAMPP is somewhat surprising and worth further investigation. We also see that error rates are higher on the short fragments than on the long fragments, which is also expected (since there is less information available for placement).

4.3.3 Impact of Query Sequence Length

We evaluated the impact of query sequence length using the RNASim-VS datasets with 50K to 200K sequences. As seen in Figure 4.4, query sequence length has a different impact on different phylogenetic placement methods. Specifically, delta error increases as sequence length decreases for all methods, but the increase is higher for APPLES-2 than for EPA-ng-SCAMPP or pplacer-SCAMPP. In addition both maximum likelihood methods within the

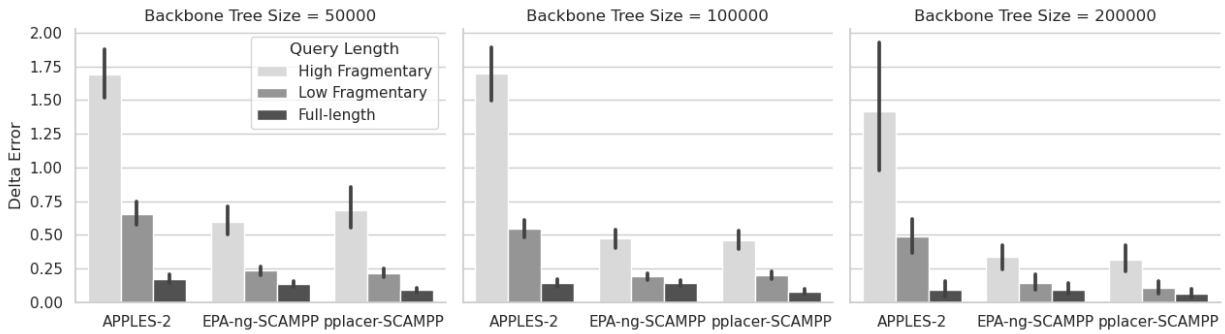


Figure 4.4: Experiment 3: Impact of query sequence length on very large backbone trees.

SCAMPP framework, EPA-ng-SCAMPP and pplacer-SCAMPP, had similar losses in delta error as the query sequence length decreased.

4.3.4 Computational Scalability of pplacer-SCAMPP

We finish this section with a direct evaluation of how well pplacer-SCAMPP scales in terms of runtime and memory usage, by comparing runtime and memory usage on the RNASim-VS datasets for both full-length and fragmentary sequences. The runtime of pplacer-SCAMPP is close to linear in the size of the backbone tree (Figure 6.2), and a detailed evaluation of the runtime of each step (Table 6.1) shows that the time used by pplacer itself is constant across all backbone tree sizes. We similarly see that the peak memory usage does not increase with backbone tree size (Figure 4.5), suggesting that the maximum likelihood phylogenetic placement method is likely the process where the memory usage peaks, because the placement tree within the backbone tree remains a fixed size within our experiments. Overall, the computational scalability of pplacer-SCAMPP is very good on these datasets, and suggests the potential for being scalable to even larger backbone trees.

4.4 RESULTS FOR APPLES

We show results for APPLES on the RNASim-VS datasets, to enable a comparison to APPLES-2 and the other methods. Our results show across all datasets that APPLES-2 substantially improved on APPLES with respect to runtime, peak memory usage, and delta error for both fragmentary and full-length sequences.

The improvement in accuracy between APPLES and APPLES-2 on full-length sequences on the RNASim-VS datasets was shown in [14] and matches what we show in Figure 4.6(c). We summarize these trends, noting that on full length sequences, APPLES-2 has an accuracy

advantage over APPLES but it is not very large (and both methods are extremely accurate), but APPLES-2 is clearly more efficient than APPLES for both runtime and peak memory usage. We also note that APPLES is not that fast compared to the other methods. On the 50,000-leaf tree it is faster than pplacer-SCAMPP but slightly slower than EPA-ng-SCAMPP, and on the larger trees it is slower than both EPA-ng-SCAMPP and pplacer-SCAMPP. However, APPLES has generally good peak memory usage, which is lower than EPA-ng-SCAMPP and pplacer-SCAMPP on the 50,000- and 100,000-leaf backbone trees, but then higher than both on the 200,000-leaf backbone trees. Thus, APPLES is clearly inferior to APPLES-2 on full-length sequences, and has close but lower accuracy than pplacer-SCAMPP and EPA-ng-SCAMPP while not offering a computational advantage over EPA-ng-SCAMPP.

We now examine results for the fragmentary sequences (Figure 4.6(a,b)). Here we see a dramatic degradation in accuracy for APPLES, so that it is by far the least accurate of all methods, whether analyzing long fragments or short fragments. Furthermore, the other methods seem to reduce in runtime and peak memory usage on fragmentary sequences compared to full-length sequences, but this is not true for APPLES. Thus, APPLES-2 represents a clear and dramatic improvement over APPLES, under all RNASim-VS conditions, but especially so for fragmentary query sequences.

In trying to explain these trends, we note that the main difference between APPLES and APPLES-2 is that APPLES-2 does not rely upon distances greater than a certain threshold [14], whereas APPLES uses all distances in the distance matrix when it decides where to place the query sequence. This difference may explain why APPLES-2 in general is more accurate, and especially when there are fragmentary sequences. Specifically, recall that the design of APPLES-2, and its restriction to "short distances", is based on research regarding distance-based tree estimation and the design of "absolute fast converging" distance-based tree estimation methods (see [35]). In essence, research over the last two decades has shown that distance-based tree estimation methods are impacted by large evolutionary distances, and that estimations of large distances tend to have higher error than estimations of smaller distances. Therefore, distance-based tree estimation benefits from techniques that (effectively) restrict attention to the smaller entries in the distance matrix (or reduce the impact of the larger distances) [35, 36, 37, 38]. Furthermore, there is less information in the fragmentary sequences than in full-length sequences, which also introduces additional error in the estimated distances. Therefore, the estimated distances used by APPLES could have higher error than those used by APPLES-2 (since APPLES-2 only uses the smaller distances), and this difference could be amplified on the fragmentary sequences. This would explain why APPLES-2 would have higher accuracy than APPLES in general, and especially on fragmentary data.

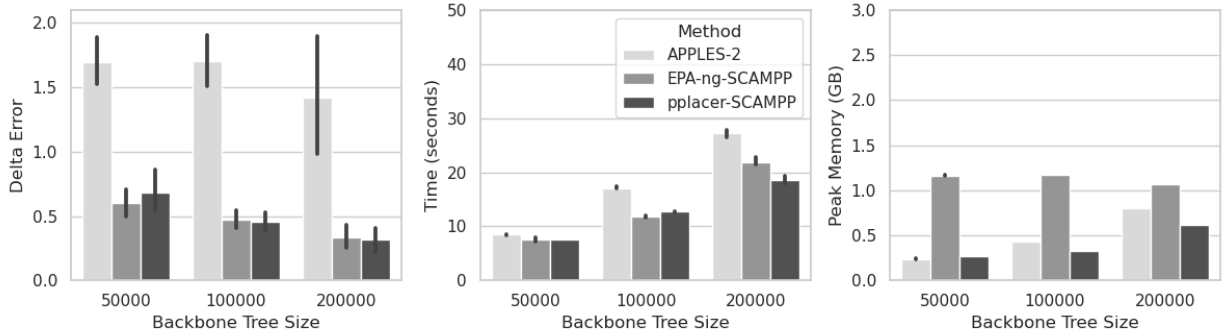
Therefore our results indicate that APPLES-2 outperforms APPLES in all tested conditions with respect to accuracy, runtime and peak memory usage.

4.5 ADDITIONAL SUBTREE SIZE EXPERIMENTS

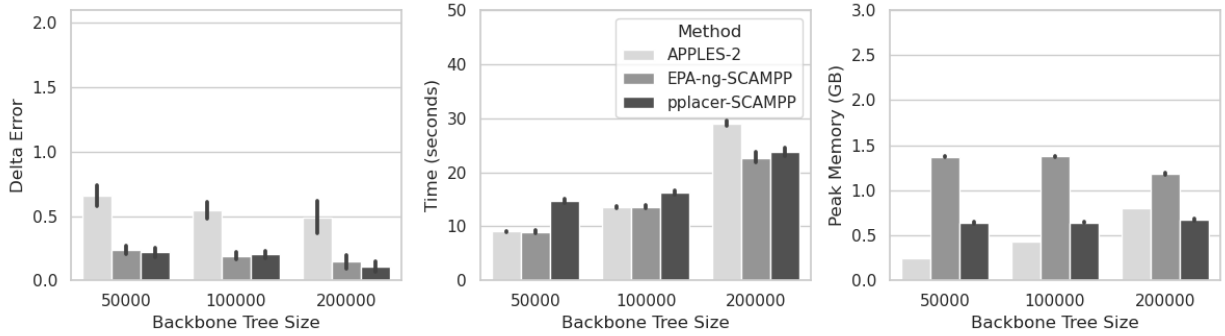
Figure 4.7 shows the impact of varying the parameter B for use in pplacer-SCAMPP and EPA-ng-SCAMPP on two datasets, 16S.B.ALL and nt78. Note that on these two datasets, small values of B produce higher delta error for both methods, but that there are no important differences in accuracy once B is large enough. There is remarkably little variation in accuracy (and overlapping error bars) once B exceeds 100 for both datasets and both methods, but even smaller values for B suffice for comparable accuracy. For example, setting $B \geq 500$ suffices for optimal accuracy for both placement methods on the nt78 dataset. These results are somewhat surprising, since we had hypothesized that further increases to B would show further reduction in placement error. However, this does not seem to be the case on the two datasets shown in Figure 4.7, nor on the two datasets shown in Experiment 1.

We also see that choices for B impact runtime and peak memory usage, with increasing B resulting in increased computational effort for both methods. The increase in runtime is greater for pplacer-SCAMPP than for EPA-ng-SCAMPP, but the increases in memory usage for both methods are comparable in magnitude.

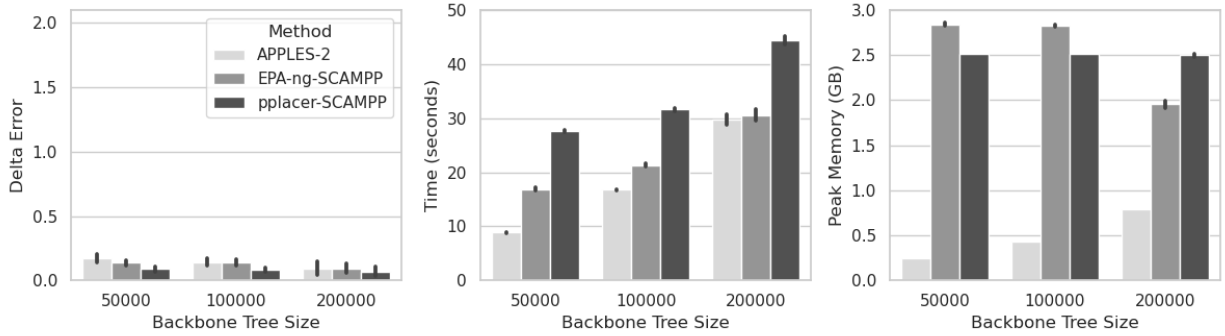
Overall these trends support the finding from Experiment 1 that setting $B = 2000$ provides good accuracy without too large an impact in terms of computational effort.



(a) Short fragments (average length 154)

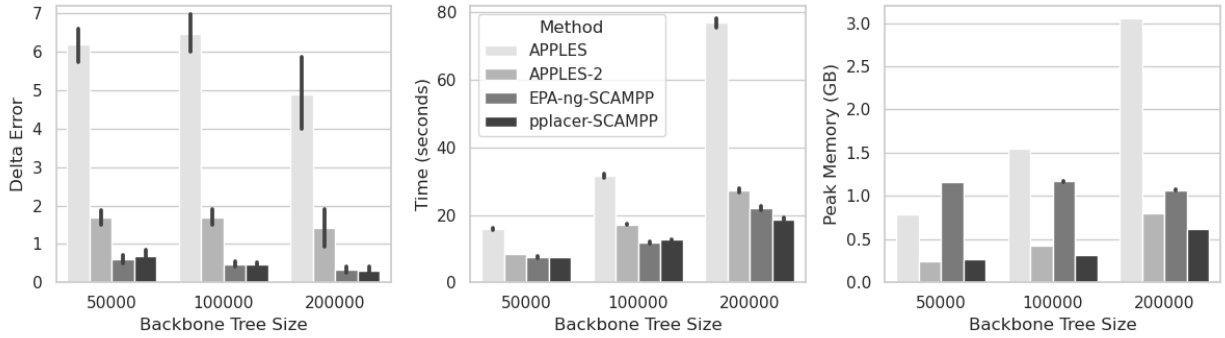


(b) Long fragments (average length 385)

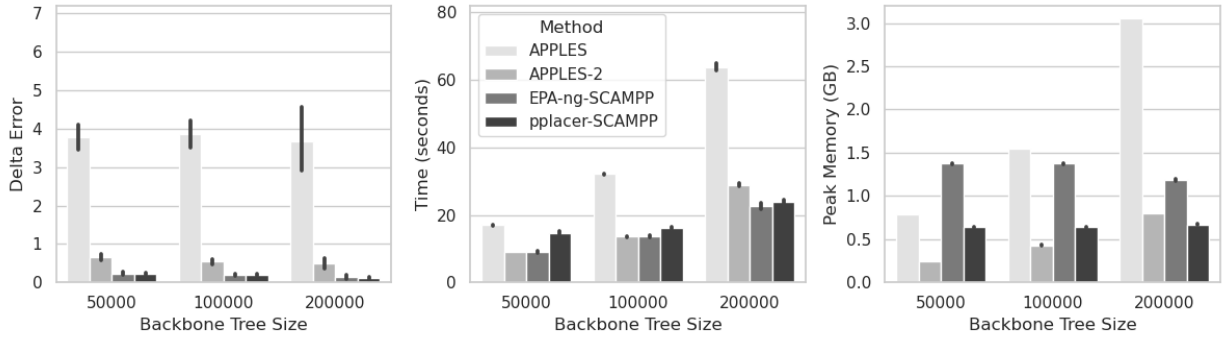


(c) Full-length sequences (average length 1539)

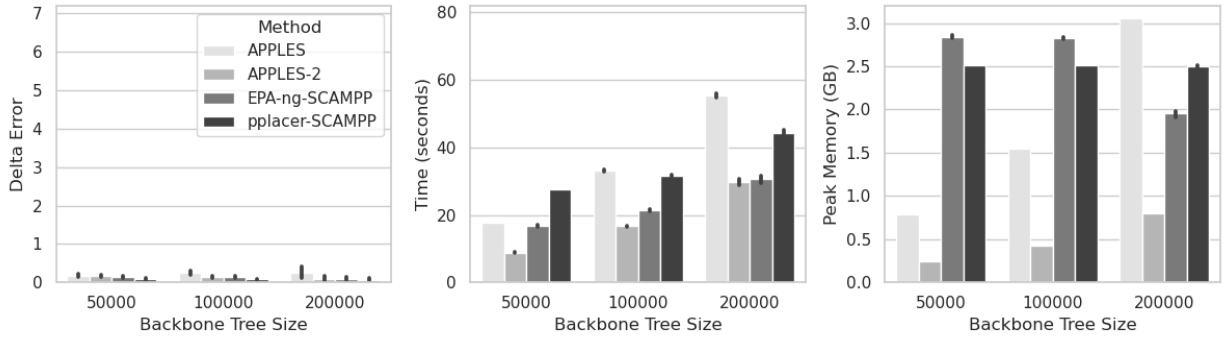
Figure 4.5: Experiment 3: Comparison of placement of full length and fragmentary sequences on the RNASim-VS backbone trees with 50,000 to 200,000 leaves. Short fragments (top row), long fragments (middle row), and full-length sequences (bottom row). Within each row, we show placement delta error (left), runtime (center), and peak memory usage (right). We report results for 1000 queries on the 50,000- and 100,000-taxon backbone trees and 200 queries on the 200,000-taxon backbone trees.



(a) Short fragments (average length 154)

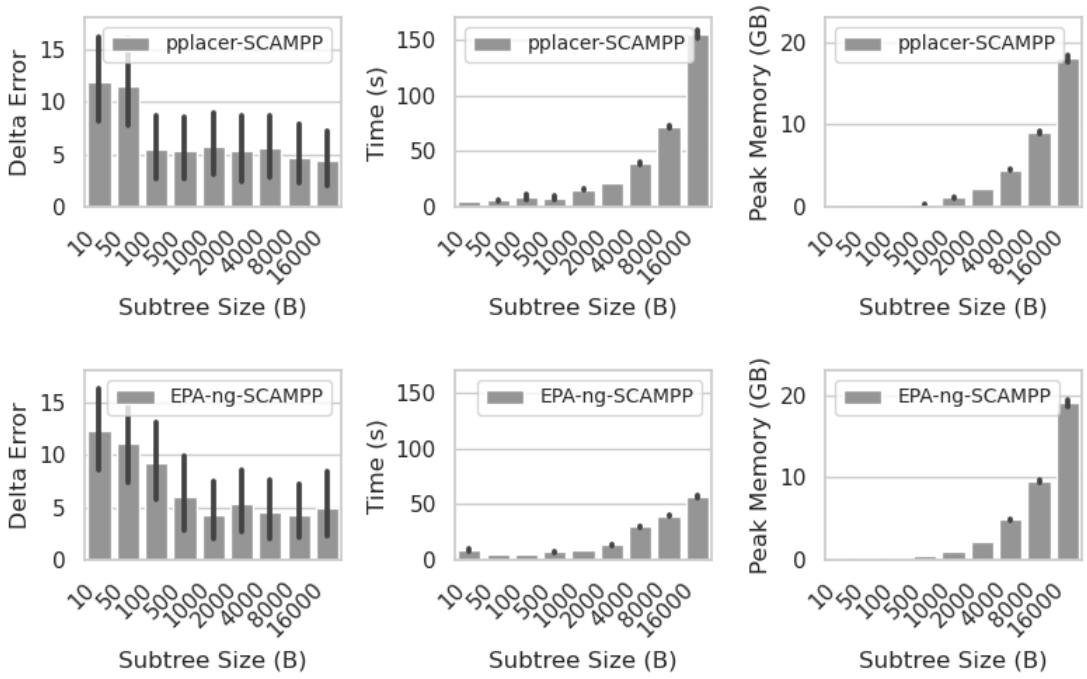


(b) Long fragments (average length 405)

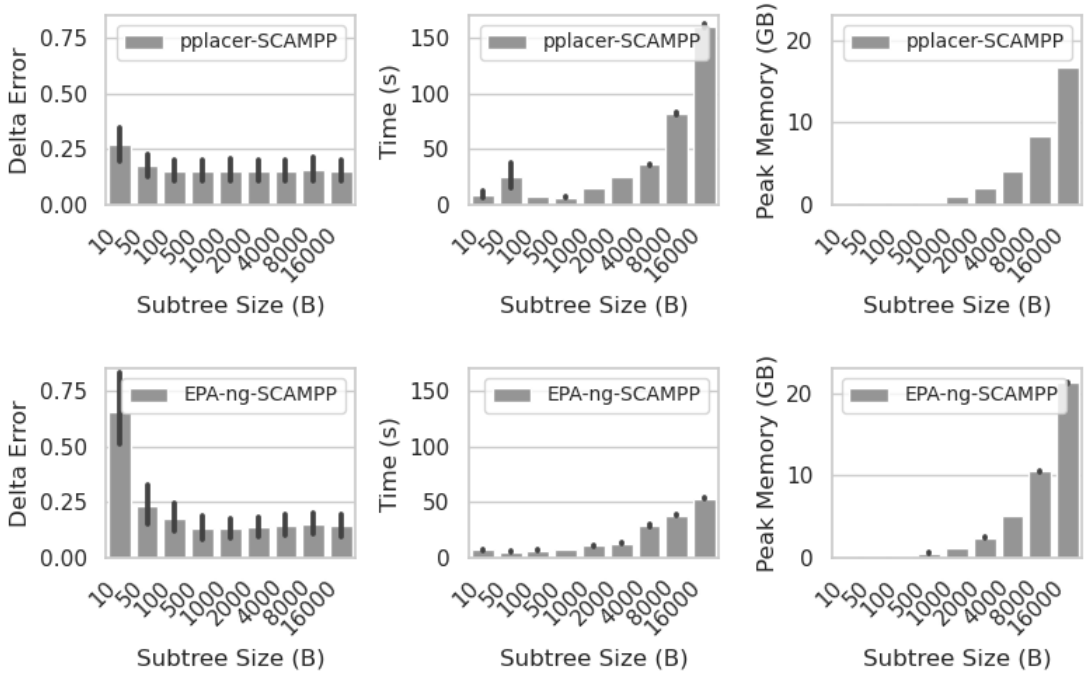


(c) Full-length sequences (average length 1539)

Figure 4.6: Experiment 3: Comparison of placement of full length and fragmentary sequences, with APPLES, included on the RNASim-VS backbone trees. Short fragments (top row), long fragments sequences (middle row), and full-length sequences (bottom row). Within each row, we show placement delta error (left), runtime (center), and peak memory usage (right). We report results for 1000 queries on the 50,000- and 100,000-taxon backbone trees and 200 queries on the 200,000-taxon backbone trees.



(a) 16S.B.ALL (27,643 sequences)



(b) nt78 (78,132 sequences)

Figure 4.7: Subtree Size Placement Results on Two Additional Datasets (one biological and one simulated). Within each row the subfigures show: Mean delta Error (left), Mean time in seconds (center), and Mean peak memory usage in Gb (right). Rows (from top to bottom) show results on 16S.B.ALL for pplacer-SCAMPP and EPA-ng-SCAMPP, and nt78 for pplacer-SCAMPP and EPA-ng-SCAMPP.

CHAPTER 5: FAILURES FOR PLACEMENT METHODS

We observed two types of failures for placement methods in our experiments: memory problems and returning invalid outputs, indicated by -inf likelihood calculations. These failures occurred only for pplacer and EPA-ng and not for their usage within SCAMPP (when applied to $B = 2000$). Furthermore, pplacer was the only method to return -inf likelihood scores, but both pplacer and EPA-ng encountered memory issues.

5.1 NEGATIVE INFINITY LIKELIHOOD SCORES FOR PPLACER

Our analyses using pplacer initially produced -inf likelihood scores for all returned placements of query sequences in the LTP_s128_SSU dataset, when used with numeric parameters estimated using RAxML (v7.2.6), which is the technique recommended by pplacer.

In order to diagnose the reason for these invalid likelihood scores, we performed an experiment restricting the size of the backbone tree to be used with pplacer to smaller subtrees on this dataset. These subtrees are selected by reading in the full tree and then randomly selecting N leaves, where N is the desired subtree size. This experiment was performed for one query sequence, and tested on subtrees from size 5000, incrementing by 1000 until the returned confidence score is -inf. The randomly selected query is Nonomuraea_monospora__FJ347524__Streptosporangiaceae.

Given a backbone tree size of 9000 sequences, pplacer gave the runtime warning

> Warning: GSL problem with location 16724 for query
> Nonomuraea_monospora__FJ347524__Streptosporangiaceae;
> Skipped with warning "computed function value is infinite or NaN".

Table 5.1: pplacer's returned likelihood score when run on subtrees of different sizes for the query Nonomuraea_monospora__FJ347524__Streptosporangiaceae from the LTP_s128_SSU dataset. The likelihood scores are reported for the best (most likely) placement.

Subtree Size	log-likelihood score of placement
5000	-995,214.972501
6000	-1,119,560.99797
7000	-1,257,558.03492
8000	-1,393,610.63579
9000	-1,514,897.48593
10,000	-1,628,585.12695
11,000	-inf

```
"placements":
  [
    {"p":
      [[0.0603660571521, 0, -nan, -inf, 1.99999275208],
        [0.0400326190167, 1, -nan, -inf, 1.99999275208],
        [0.0289007951157, 2, -nan, -inf, 1.99999275208],
        [0.0395654648659, 3, -nan, -inf, 1.99999275208],
        [0.0197602138302, 4, -nan, -inf, 1.99999275208],
        [0.00312862106398, 5, -nan, -inf, 1.99999275208],
        [0.00100212520434, 6, -nan, -inf, 1.99999275208]
      ], "nm": [["Nonomuraea_monospora__FJ347524__Streptosporangiaceae", 1]]
    }
  ]
```


Figure 5.1: Sample pplacer Output with Negative Infinity Likelihood Scores

We also note the output of a failed placement (Figure 5.1), as it appears in the jplace file: The likelihood scores of different size subtrees is reported in the Table 5.1. Note that the likelihood scores are decreasing quickly (equivalently, negative values that are increasing in magnitude) with the size of the backbone tree. The decreasing likelihood scores suggest the numerical issues might be the source of the -inf likelihoods score.

5.2 MEMORY PROBLEMS FOR PPLACER

A placement 'failure' for pplacer in the nt78 dataset occurred when pplacer encountered a segmentation fault and failed to produce any jplace file.

5.3 FAILURES FOR EPA-NG, EPA-NG-SCAMPP, AND PPLACER-SCAMPP

All failures encountered by EPA-ng were segmentation faults, and these also occurred on the nt78 datasets. We did not attempt to run EPA-ng on the RNASim-VS datasets with 50K or more sequences. The EPA-ng-SCAMPP and pplacer-SCAMPP analyses never failed with any setting of B we tried.

5.4 IMPLICATIONS FOR EPA-NG AND PPLACER

A comparison between pplacer and EPA-ng is helpful. As noted, pplacer failed to place sequences on many large backbone trees, returning $-\infty$ log likelihood values, when using RAxML to estimate numerical parameters on the backbone tree (as recommended in [9]).

Instead, estimating numerical parameters with FastTree 2 (and then using taxit to format the output suitably for pplacer) enables it to produce valid outputs.

Why FastTree 2 produces numerical parameters that allows pplacer to run properly is not known, but clearly there is some sensitivity in pplacer to these choices. In general, FastTree 2 is known to not be as accurate as RAxML at estimating numeric parameters (e.g., the likelihood scores returned by FastTree 2 are not as high as the likelihood scores returned by RAxML, even when the trees are of comparable topological accuracy [39]), and this trend suggests that less accurate parameter estimation may somehow be useful for scalability of pplacer. Certainly, these trends further support the hypothesis that pplacer has numerical issues.

In contrast, when EPA-ng failed to complete on these datasets, the issue was always inadequate available memory. Therefore, EPA-ng and pplacer have different vulnerabilities on large backbone trees, explaining why they respond differently within the SCAMPP framework.

CHAPTER 6: DISCUSSION

6.1 IMPACT OF USING SCAMPP

In general, we find that SCAMPP improves computational performance (both memory usage and running time) for both phylogenetic placement methods, but the impact on accuracy depends on the model condition and even depends on whether pplacer or EPA-ng is used. We also see that the relative accuracy and computational efficiency (both speed and memory) compared to a leading fast phylogenetic placement method, APPLES-2, depends on the model condition. We therefore begin with a discussion of these trends.

Starting with a comparison between pplacer and pplacer-SCAMPP, we note that on those datasets on which pplacer could run, pplacer-SCAMPP was always at least as accurate as pplacer, often substantially more accurate, and was also faster and had a smaller peak memory usage. We also see that the subtree size, as defined by B, has a large impact on pplacer: when B is very small (e.g., below 500 on the datasets in Experiment 1), delta error rates are high, then error rates drop as B increases to (approximately) 2000, but as B increases beyond that the error rates can become very high. This is most noteworthy on the LTP_s128_SSU dataset, where there is a very large increase in error at $B = 8000$. This trend shows that pplacer accuracy degrades on very large placement trees, which is an intriguing finding. Since we also observed that pplacer can return $-\infty$ values on large backbone trees (Chapter 5, and Table 5.1), this suggests that the issue is numerical. Taking these observations together, we infer that pplacer has numerical issues that make it not work that well (in terms of accuracy) on large placement trees, which is why the use of SCAMPP improves accuracy.

The comparison between EPA-ng and EPA-ng-SCAMPP presents somewhat different trends. On those datasets on which both methods run, we sometimes see EPA-ng more accurate and sometimes less accurate than EPA-ng-SCAMPP; while these differences are small, the fact that EPA-ng-SCAMPP can be less accurate than EPA-ng indicates a noteworthy difference between EPA-ng and pplacer. We also see that increases in B beyond 2000 has a small but occasionally negative imapct on EPA-ng-SCAMPP. On the other hand, EPA-ng can fail to run on some datasets due to memory requirements. Overall, these trends suggest that EPA-ng may not have the same numeric vulnerability as we saw in pplacer (see Chapter 5 for additional discussion about this issue). In sum, our study shows that EPA-ng can provide good accuracy on those large backbone trees on which it can run, and the benefit to using EPA-ng within the SCAMPP framework may only be to enable it to run within the available computational resources.

SCAMPP has a larger beneficial impact, especially for runtime and memory usage, for placing fragmentary sequences than it does for full-length sequences, which is also interesting.

Specifically, pplacer-SCAMPP and EPA-ng-SCAMPP become much more computationally efficient on fragmentary sequences compared to full-length sequences, while APPLES-2 does not become more efficient on fragmentary sequences. These reductions in runtime and memory use are likely due to the masking techniques for leading and trailing gaps used in the SCAMPP framework as well as in EPA-ng and pplacer [11].

Overall, therefore, using SCAMPP greatly reduced runtime and memory usage for both pplacer and EPA-ng, and either improved accuracy or at worst slightly decreased accuracy; however, the decreases in accuracy were limited to EPA-ng-SCAMPP. Moreover, there were many large backbone trees on which neither pplacer nor EPA-ng could run, and using SCAMPP enabled them to run and with low memory usage. Thus, there was always a benefit obtained in using SCAMPP in our experiments, but the type of benefit and its magnitude depends on the placement method, backbone tree, and query sequence length.

6.2 CHOOSING BETWEEN PHYLOGENETIC PLACEMENT METHODS

This study establishes that pplacer-SCAMPP, EPA-ng-SCAMPP, and APPLES-2 can be used on large backbone trees, and that pplacer and EPA-ng are not as scalable as these three methods. Here we discuss the question of which of the three scalable methods should be used, and under which conditions. A comparison between pplacer-SCAMPP and EPA-ng-SCAMPP shows little difference in terms of accuracy, and differences for runtime and memory usage that depend on the dataset. Here we examine the difference between pplacer-SCAMPP and APPLES-2, as an example of when this framework can be useful.

The datasets we explored can roughly be divided into two sets: RNASim-VS with full-length query sequences, and everything else. We will begin with a discussion of the RNASim-VS with full-length sequences, since these present what may be a special case.

On the RNASim-VS with full-length sequences, all the methods achieved very low average delta error (below 0.3), and although pplacer-SCAMPP had the lowest error, the differences in delta error between any two methods were very low. Given this, the driving factor in choosing between methods may be computational performance. Although pplacer-SCAMPP does provide an accuracy advantage over APPLES-2 on these datasets, it is also more computationally intensive (somewhat slower and using much more peak memory). Hence, the accuracy advantage provided by pplacer-SCAMPP over APPLES-2 probably does not make up for the large computational hit.

In contrast, when placing fragmentary sequences into the same RNASim-VS backbone

trees, the relative performance changes, and dramatically. First, the accuracy advantage provided by pplacer-SCAMPP over APPLES-2 is much larger, and increases as the query sequence length decreases. We also note that the delta error increases for all methods (but more so for APPLES-2) as the query sequence length decreases. In addition, on the shortest query sequences, where the length approximates that of many sequencing reads, APPLES-2 and pplacer-SCAMPP have very similar memory usage and runtime (but with an advantage to pplacer-SCAMPP). Thus, query sequence length impacts the relative accuracy and computational performance of APPLES-2 and pplacer-SCAMPP, with shorter query sequences favoring pplacer-SCAMPP.

It is therefore worthwhile to consider the four other datasets we studied, where the backbone trees varied in size from about 5000 leaves to at most $78,000$ leaves and the query sequences were full-length. On these four datasets, pplacer-SCAMPP–although fast and reasonably memory efficient–was always slower than APPLES-2 and had higher memory usage. Therefore, the choice of method on these four datasets would most likely be driven by the degree to which pplacer-SCAMPP provided an accuracy advantage. On each of these four datasets, pplacer-SCAMPP produced improved accuracy over APPLES-2, and in many of these cases the improvement was large. Although these cases involved full-length query sequences, many of them produced high placement error for all methods.

This study suggests that the typical condition may be one where APPLES-2 has insufficient accuracy and a likelihood-based method has sufficiently improved accuracy to merit its use. In particular, the running time and peak memory usage advantage of APPLES-2 seems to disappear when placing fragmentary sequences into large backbone trees, and this application may be the most common one (especially for analyses of metagenomic data).

6.3 RELATED STUDIES

This work builds off of an earlier prototype [1], which was limited to the SCAMPP framework's use with pplacer, and only explored a single model condition (RNAsim VS) with only full-length sequences. We now compare that implementation of SCAMPP to the current implementation, demonstrating that the new implementation is much faster and has lower memory usage that the initial implementation. We then discuss other related approaches for scaling phylogenetic placement methods to large datasets.

6.3.1 Running Time Comparisons of pplacer-SCAMPP Implementations

Our SCAMPP framework is based on an early prototype that was specifically designed

for use with pplacer, and was referred to as "pplacer-XR". This first implementation of pplacer-SCAMPP was based on a Python script that relied upon Dendropy [40] and had other aspects that affected the runtime. The study [1] evaluating pplacer-SCAMPP v1 showed good accuracy and scalability to large backbone trees, but was only compared to APPLES and not to APPLES-2. The second implementation of the SCAMPP framework has been implemented for improved speed, largely through using better data structures, and studied with both pplacer and EPA-ng. The codebase for the SCAMPP framework contains both versions of the SCAMPP framework, and is available at `https://github.com/chry04/PLUSplacer`.

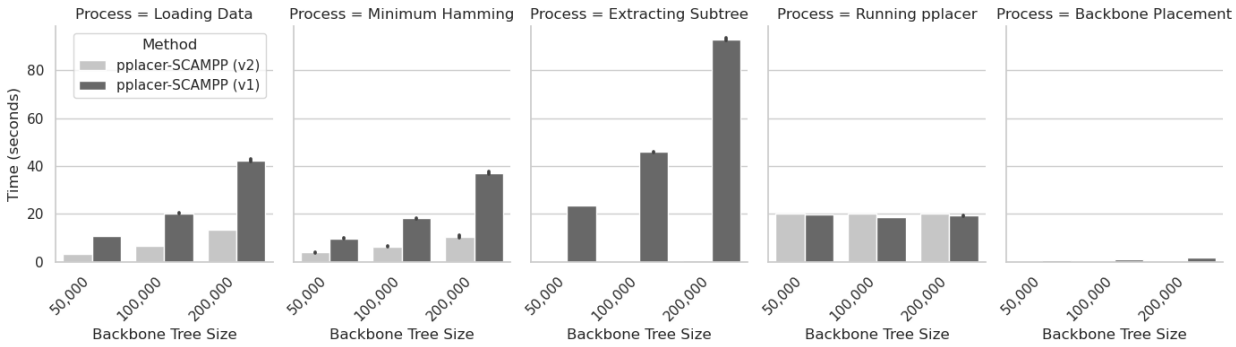


Figure 6.1: A breakdown of the mean runtime of the original (version 1) and revised (version 2) implementations of pplacer-SCAMPP. Both versions of pplacer-SCAMPP were run under the default settings.

Here we compare these two implementations of pplacer-SCAMPP with respect to runtime, breaking this down into the different phases of the SCAMPP framework:

- Phase 1: loading the input data,
- Phase 2: finding the nearest taxon (using Hamming distances) to the query sequence,
- Phase 3: extracting the placement subtree for the query sequence,
- Phase 4: placing the query sequence into the placement subtree using pplacer, and
- Phase 5: finding the corresponding edge in the backbone tree.

Thus, Phases 1–3 together comprise Stage 1 within SCAMPP, Phase 4 is the same as Stage 2, and Phase 5 is the same as Stage 3. We report running times for pplacer-SCAMPP for each phase when placing full-length sequences into the RNASim-VS datasets with backbone trees of size 50K up to 200K.

As seen in Figure 6.1, the two implementations of pplacer-SCAMPP are essentially identical in runtime for the last two phases but have very different running times on the first

three phases (at least on the larger backbone trees), where the re-implementation of pplacer-SCAMPP is much faster than the initial implementation pplacer-SCAMPP v1. The runtime differences between the two implementations increase with the size of the backbone tree.

6.3.2 Other Studies

Table 6.1: Runtime Breakdown for pplacer-SCAMPP on full-length sequences

RNASim Tree Size	Time Per Process in Seconds					
	Loading Data	Finding Taxon	Extracting Subtree	Running pplacer	Backbone Placement	Total Runtime
50,000	3.24	3.88	0.37	20.21	0.06	27.76
100,000	6.64	6.43	0.48	20.29	0.07	33.91
200,000	13.37	10.60	0.80	20.33	0.09	45.19

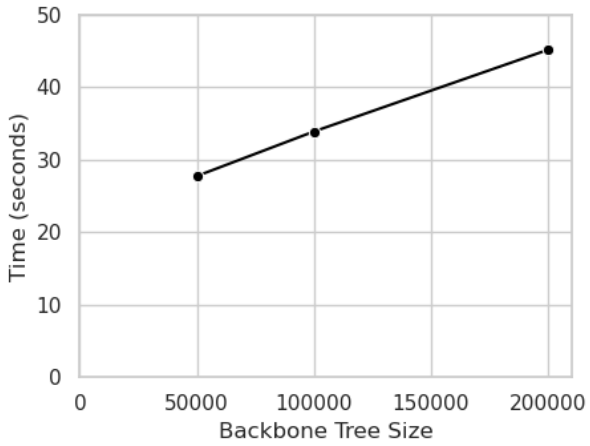


Figure 6.2: Mean running time for pplacer-SCAMPP on three different RNAsim backbone tree sizes, 50,000, 100,000, and 200,000, in placing full-length sequences.

There are two relatively closed related approaches to the SCAMPP framework that require discussion: pplacerDC [41] and the multilevel "Russian Doll" phylogenetic placement technique [42]. We discuss each in turn.

The pplacerDC [41] technique employs a more exhaustive approach than pplacer-SCAMPP, but also enables pplacer to scale to larger backbone trees. In pplacerDC, the backbone tree is divided (through edge deletions) into placement subtrees with a bounded number of leaves, and then the query sequence is placed into each of the placement subtrees using pplacer.

Each such placement is then embedded in the full backbone tree, thus producing an extended backbone tree, and the maximum likelihood score is estimated (using the provided numeric parameters, which are not re-optimized) using RAxML. The extended backbone trees thus have all the taxa (including the query sequence) and so can be compared to each other with respect to their maximum likelihood scores. The tree with the best likelihood score is then returned.

The study evaluating pplacerDC was limited to the RNASim VS datasets, where it was shown to be able to scale to backbone trees with 100,000 leaves. However, pplacerDC failed on backbone trees 200,000 leaves and was extremely computationally intensive on those datasets on which it completed – with higher running time and peak memory usage than pplacer-SCAMPP. Finally, pplacerDC was not evaluated on datasets with fragmentary sequences. Since both methods have been run on the same datsets (RNASim-VS with full-length sequences), it is possible to compare them. This comparison shows that pplacerDC is slower, has higher delta error (e.g., about twice as high on the 100,000-leaf backbone tree), and higher peak memory usage.

To understand the differences in runtime and peak memory usage, recall that pplacerDC requires that the query sequence be placed into all the placement subtrees created by the decomposition, which makes the runtime increase linearly with the backbone tree size (unless run with unbounded parallelism. There is also a memory and runtime hit produced by the use of RAxML to compute the likelihood score of each extended tree (and the number of these trees grows linearly with the size of the backbone tree). Thus, it is easy to see why pplacerDC is more computationally intensive than pplacer-SCAMPP. However, it is not clear why there should be an accuracy disadvantage. One possible explanation is that the placement subtrees produced by pplacerDC and pplacer-SCAMPP are very different: pplacerDC forms the placement subtrees by deleting a set of edges and then fixes this set for use by all the query sequences, whereas pplacer-SCAMPP allows each query sequence to optimize the selection of its placement subtree to be induced by the leaves that are closest to its selected nearest taxon. It may well be that these individually-selected placement subtrees produced by pplacer-SCAMPP are better suited for use with pplacer than the pre-computed placement subtrees produced by pplacerDC, which are not optimized for individual query sequences. However, it is also possible that using RAxML to compare different placements (through calculation of the maximum likelihood score) does not provide good accuracy on large backbone trees. Future work is needed to understand why pplacer-SCAMPP is more accurate than pplacerDC.

Another useful strategy for addressing the limited scalability of phylogenetic placement methods with respect to the backbone tree size is the multilevel placement method [42]

that is also available within the GAPPA suite of tools [43]. The multilevel "Russian Doll" placement approach is described for use with a taxonomy (on a carefully selected set of sequences), but the general technique can be extended for use with any rooted tree. A sparse but representative subset of leaves from the rooted tree is selected, and is then used as the backbone tree (where it is referred to as a "broad backbone tree" (BT)). A phylogenetic placement method is then used to place the query sequence into the BT, which allows it to identify the best clades (rooted subtrees) for more careful exploration. The query sequence can then be placed in each of the clades, and the best placement(s) for each query sequence can be identified. By design, this multi-stage process reduces the need to place into the full backbone tree, and so reduces the computational effort for phylogenetic placement. The approach is very different from ours in a few ways, but the main difference is that it requires rooted trees. Nevertheless, it is a very interesting approach, and extending this technique to work with unrooted trees merits investigation.

6.4 OTHER FUTURE WORK

In previous sections we have identified some directions for future work. Here we discuss additional opportunities for developing the approach we have described here, as well as alternative approaches.

6.4.1 Improving the SCAMPP Design

The current SCAMPP strategy places a query sequence by finding a nearest taxon (i.e., a leaf that has minimum Hamming distance to the query sequence) and then extracts a subtree with B leaves using that leaf. Thus, the current strategy has only one algorithmic parameter (B) beyond the choice of the placement method. Our default is $B = 2000$, but Experiment 1 and our additional evaluations reported in Section 4.5 suggested the possibility that the optimal setting for B might depend both on the dataset properties and on the phylogenetic placement method. In particular, if accuracy is the most important objective, then it seems possible that larger values of B might improve accuracy for EPA-ng-SCAMPP, and that very small values might suffice for sufficiently "easy" datasets. Hence a better understanding of the impact of dataset properties on this parameter selection is needed. Moreover, our placement subtree construction approach is very simple, and it is possible that other techniques for extracting a placement subtree might provide improved accuracy compared to this technique, even if the runtime and memory usage does not change.

We also note that EPA-ng has been optimized for placing large numbers of query sequences into backbone trees. This is an advantage for EPA-ng that is not enabled in the SCAMPP framework, which gives the placement method a different subtree for each query sequence. In order to take advantage of the batch processing offered by methods such as EPA-ng, a different divide-and-conquer framework would need to be explored.

6.4.2 Additional Evaluation

More generally, a full evaluation of the SCAMPP phylogenetic placement approach requires additional study. We performed a leave-one-out study, but a more extensive analysis including leave-clade-out study should be explored. We also did not explore the impact of alignment error in the phylogenetic placement pipeline, and so this should also be examined. Finally, we explored pplacer-SCAMPP and EPA-ng-SCAMPP in the context of growing a large tree, but they should also be evaluated for use in microbiome abundance profiling and taxon identification, as some of the most accurate such methods use phylogenetic placement. Thus, there are several directions for future work that have the potential to lead to improved understanding of how to design phylogenetic placement methods for use in different downstream applications.

CHAPTER 7: CONCLUSIONS

Phylogenetic placement is a basic computational step in several bioinformatics pipelines, including incremental construction of very large phylogenies and taxonomic identification of reads obtained in metagenomic analyses. Of the many phylogenetic placement methods that have been developed, methods that use maximum likelihood, such as pplacer and EPA-ng, has been found to be the most accurate. Unfortunately, these likelihood-based methods are difficult to use with moderately large backbone trees (i.e., they can fail to return valid outputs or may have excessive memory requirements), which has meant that other phylogenetic placement methods are necessary. Methods such as APPLES-2 use distance-based techniques to perform phylogenetic placement, and are particularly fast and scalable; however, this and other studies have shown distance-based placement to not provide the same level of accuracy as likelihood-based methods.

We have presented the SCAMPP framework, a three-stage procedure for scaling alignment-based phylogenetic placement methods. We evaluated the SCAMPP framework for use with two such methods, pplacer and EPA-ng. Our study showed that using SCAMPP allowed both pplacer and EPA-ng to scale to backbone trees with 200,000 leaves without high peak memory requirements, thus greatly surpassing the limitations of these methods when used outside the framework. For those datasets on which pplacer could run, we also saw that pplacer-SCAMPP had better accuracy than pplacer, was faster, and used less peak memory. While EPA-ng-SCAMPP was sometimes less accurate than EPA-ng, those reductions in accuracy tended to be small and the improvement in running time and peak memory usage was very high. Thus, in general SCAMPP provides computational benefits to both methods and either improves accuracy (for pplacer) or has a variable impact (for EPA-ng) that tends to be minor.

One of the interesting trends we saw in this study is that although pplacer-SCAMPP improved on APPLES-2 for accuracy in all the cases we evaluated, the differences in some cases were extremely small; furthermore, in nearly all cases, APPLES-2 was the fastest and least memory-intensive method. Thus, it is not at all obvious that any one method dominates the others. This is particularly important, given that computational performance may be a limiting factor, making it by necessity a requirement to use the fastest method, or the least memory-intensive method, on a given dataset. In considering the different factors that impact accuracy and runtime/memory usage, we suggest that APPLES-2 be used when highly accurate phylogenetic placement seems likely, as it tends to be the most computationally efficient, but that pplacer-SCAMPP or EPA-ng-SCAMPP be used under

other conditions. In particular, there may be a benefit to using pplacer-SCAMPP or EPA-ng-SCAMPP instead of APPLES-2 when the query sequences are short, as is typical in metagenomic datasets.

REFERENCES

[1] E. Wedell, Y. Cai, and T. Warnow, "Scalable and accurate phylogenetic placement using pplacer-XR," in *International Conference on Algorithms for Computational Biology*. Springer, 2021, https://link.springer.com/chapter/10.1007/978-3-030-74432-8_7. pp. 94–105.

[2] S. Conlan, H. H. Kong, and J. A. Segre, "Species-level analysis of DNA sequence data from the NIH Human Microbiome Project," *PloS one*, vol. 7, no. 10, p. e47075, 2012.

[3] C. O. McCoy and F. A. Matsen, "Abundance-weighted phylogenetic diversity measures distinguish microbial community states and are robust to sampling depth," *PeerJ*, vol. 1, p. e157, 2013.

[4] N.-p. Nguyen, S. Mirarab, B. Liu, M. Pop, and T. Warnow, "TIPP: taxonomic identification and phylogenetic profiling," *Bioinformatics*, vol. 30, no. 24, pp. 3548–3555, 2014.

[5] N. Shah, E. K. Molloy, M. Pop, and T. Warnow, "TIPP2: metagenomic taxonomic profiling using phylogenetic markers," *Bioinformatics*, 2021, dOI: 10.1093/bioinformatics/btab023.

[6] P.-A. Chaumeil, A. J. Mussig, P. Hugenholtz, and D. H. Parks, "GTDB-Tk: a toolkit to classify genomes with the Genome Taxonomy Database," *Bioinformatics*, vol. 36, no. 6, pp. 1925–1927, 2020.

[7] H. M. Bik, D. L. Porazinska, S. Creer, J. G. Caporaso, R. Knight, and W. K. Thomas, "Sequencing our way towards understanding global eukaryotic biodiversity," *Trends in Ecology & Evolution*, vol. 27, no. 4, pp. 233–243, 2012.

[8] M. Balaban, S. Sarmashghi, and S. Mirarab, "APPLES: scalable distance-based phylogenetic placement with or without alignments," *Systematic Biology*, vol. 69, no. 3, pp. 566–578, 2020.

[9] F. A. Matsen, R. B. Kodner, and E. V. Armbrust, "pplacer: linear time maximum-likelihood and Bayesian phylogenetic placement of sequences onto a fixed reference tree," *BMC bioinformatics*, vol. 11, no. 1, p. 538, 2010.

[10] S. A. Berger, D. Krompass, and A. Stamatakis, "Performance, accuracy, and web server for evolutionary placement of short sequence reads under maximum likelihood," *Systematic Biology*, vol. 60, no. 3, pp. 291–302, 2011.

[11] P. Barbera, A. M. Kozlov, L. Czech, B. Morel, D. Darriba, T. Flouri, and A. Stamatakis, "EPA-ng: massively parallel evolutionary placement of genetic sequences," *Systematic Biology*, vol. 68, no. 2, pp. 365–369, 2019.

[12] B. Linard, K. Swenson, and F. Pardi, "Rapid alignment-free phylogenetic identification of metagenomic sequences," *Bioinformatics*, vol. 35, no. 18, pp. 3303–3312, 2019.

[13] M. Blanke and B. Morgenstern, "App-spam: phylogenetic placement of short reads without sequence alignment," *Bioinformatics Advances*, vol. 1, no. 1, p. vbab027, 2021.

[14] M. Balaban, Y. Jiang, D. Roush, Q. Zhu, and S. Mirarab, "Fast and accurate distance-based phylogenetic placement using divide and conquer," *Molecular Ecology Resources*, 2021.

[15] F. A. Matsen, N. G. Hoffman, A. Gallagher, and A. Stamatakis, "A format for phylogenetic placements," *PLoS One*, vol. 7, no. 2, p. e31009, 2012.

[16] S. Tavaré, "Some probabilistic and statistical problems in the analysis of DNA sequences," *Lectures on mathematics in the life sciences*, vol. 17, no. 2, pp. 57–86, 1986.

[17] S. Mirarab, N. Nguyen, and T. Warnow, "SEPP: SATé-enabled phylogenetic placement," in *Biocomputing 2012*. World Scientific, 2012, pp. 247–258.

[18] B. Linard, N. Romashchenko, F. Pardi, and E. Rivals, "PEWO: a collection of workflows to benchmark phylogenetic placement," *Bioinformatics*, vol. 36, no. 21, pp. 5264–5266, 07 2020. [Online]. Available: https://doi.org/10.1093/bioinformatics/btaa657

[19] J. J. Cannone, S. Subramanian, M. N. Schnare, J. R. Collett, L. M. D'Souza, Y. Du, B. Feng, N. Lin, L. V. Madabusi, K. M. Müller, N. Pande, Z. Shang, N. Yu, and R. R. Gutell, "The comparative RNA web (CRW) site: an online database of comparative sequence and structure information for ribosomal, intron, and other RNAs," *BMC Bioinf*, vol. 3, no. 1, p. 2, 2002.

[20] M. N. Price, P. S. Dehal, and A. P. Arkin, "FastTree 2–approximately maximum-likelihood trees for large alignments," *PloS one*, vol. 5, no. 3, p. e9490, 2010.

[21] S. Mirarab, N. Nguyen, S. Guo, L.-S. Wang, J. Kim, and T. Warnow, "PASTA: ultra-large multiple sequence alignment for nucleotide and amino-acid sequences," *Journal of Computational Biology*, vol. 22, no. 5, pp. 377–386, 2015.

[22] T. Z. DeSantis, P. Hugenholtz, N. Larsen, M. Rojas, E. L. Brodie, K. Keller, T. Huber, D. Dalevi, P. Hu, and G. L. Andersen, "Greengenes, a chimera-checked 16S rRNA gene database and workbench compatible with ARB," *Applied and Environmental Microbiology*, vol. 72, no. 7, pp. 5069–5072, 2006.

[23] P. Yarza, M. Richter, J. Peplies, J. Euzeby, R. Amann, K.-H. Schleifer, W. Ludwig, F. O. Glöckner, and R. Rosselló-Móra, "The All-Species Living Tree Project: A 16S rRNA-based phylogenetic tree of all sequenced type strains," *Systematic and Applied Microbiology*, vol. 31, no. 4, pp. 241–250, 2008.

[24] P. Yarza, W. Ludwig, J. Euzéby, R. Amann, K.-H. Schleifer, F. O. Glöckner, and R. Rosselló-Móra, "Update of the All-Species Living Tree Project based on 16S and 23S rRNA sequence analyses," *Systematic and Applied Microbiology*, vol. 33, no. 6, pp. 291–299, 2010.

[25] A. Stamatakis, "RAxML-VI-HPC: maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models," *Bioinformatics*, vol. 22, no. 21, pp. 2688–2690, 2006.

[26] M. N. Price, P. S. Dehal, and A. P. Arkin, "FastTree 2–approximately maximum-likelihood trees for large alignments," *PloS one*, vol. 5, no. 3, p. e9490, 2010.

[27] N.-p. D. Nguyen, S. Mirarab, K. Kumar, and T. Warnow, "Ultra-large alignments using phylogeny-aware profiles," *Genome Biology*, vol. 16, no. 1, pp. 1–15, 2015.

[28] V. Smirnov and T. Warnow, "Phylogeny estimation given sequence length heterogeneity," *Systematic Biology*, vol. 70, no. 2, pp. 268–282, 2021.

[29] V. Smirnov and T. Warnow, "MAGUS: Multiple sequence alignment using graph clustering," *Bioinformatics*, 2020, dOI: https://doi.org/10.1093/bioinformatics/btaa992.

[30] A. M. Kozlov, D. Darriba, T. Flouri, B. Morel, and A. Stamatakis, "RAxML-NG: a fast, scalable and user-friendly tool for maximum likelihood phylogenetic inference," *Bioinformatics*, vol. 35, no. 21, pp. 4453–4455, 2019.

[31] FHCRC, "Taxtastic software, version 9.9.2," 2022, https://github.com/fhcrc/taxtastic, last accessed Feb 20, 2022, distributed by Fred Hutchison Computational Biology.

[32] F. A. Matsen, R. B. Kodner, and E. V. Armbrust, "pplacer: linear time maximum-likelihood and Bayesian phylogenetic placement of sequences onto a fixed reference tree," *BMC bioinformatics*, vol. 11, no. 1, p. 538, 2010.

[33] M. Balaban, S. Sarmashghi, and S. Mirarab, "APPLES: distance-based phylogenetic placement for scalable and assembly-free sample identification," *bioRxiv*, p. 475566, 2019.

[34] M. Balaban, S. Sarmashghi, and S. Mirarab, "Data from: APPLES: scalable distance-based phylogenetic placement with or without alignments," 2019, dOI: https://doi.org/10.5061/dryad.78nf7dq.

[35] T. Warnow, B. M. Moret, and K. St. John, "Absolute convergence: true trees from short sequences," in *Proc. 12th Ann. ACM/SIAM Symp. Discrete Algs.(SODA01)*. SIAM Press, 2001, pp. 186–195.

[36] Q. Zhang, S. Rao, and T. Warnow, "Constrained incremental tree building: new absolute fast converging phylogeny estimation methods with improved scalability and accuracy," *Algorithms for Molecular Biology*, vol. 14, no. 1, pp. 1–12, 2019.

[37] W. J. Bruno, N. D. Socci, and A. L. Halpern, "Weighted neighbor joining: a likelihood-based approach to distance-based phylogeny reconstruction," *Molecular biology and evolution*, vol. 17, no. 1, pp. 189–197, 2000.

[38] O. Gascuel, "BIONJ: an improved version of the nj algorithm based on a simple model of sequence data." *Molecular biology and evolution*, vol. 14, no. 7, pp. 685–695, 1997.

[39] K. Liu, C. R. Linder, and T. Warnow, "RAxML and FastTree: comparing two methods for large-scale maximum likelihood phylogeny estimation," *PloS one*, vol. 6, no. 11, p. e27731, 2011.

[40] J. Sukumaran and M. T. Holder, "Dendropy: a python library for phylogenetic computing," *Bioinformatics*, vol. 26, no. 12, pp. 1569–1571, 2010.

[41] E. Koning, M. Phillips, and T. Warnow, "pplacerDC: a new scalable phylogenetic placement method." in *Proceedings of the 12th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics*, 2021, pp. 1–9.

[42] L. Czech, P. Barbera, and A. Stamatakis, "Methods for automatic reference trees and multilevel phylogenetic placement," *Bioinformatics*, vol. 35, no. 7, pp. 1151–1158, 2019.

[43] L. Czech, P. Barbera, and A. Stamatakis, "Genesis and Gappa: processing, analyzing and visualizing phylogenetic (placement) data," *Bioinformatics*, vol. 36, no. 10, pp. 3263–3265, 2020.

APPENDIX A: ADDITIONAL EXAMPLE

A.1 ILLUSTRATION OF STAGE 3

In Stage 3, SCAMPP takes the placement of the query sequence into the subtree (as defined by Stage 2) and determines where to add the query sequence into the backbone tree. We show two examples of how SCAMPP would perform the phylogenetic placement of two different query sequences, **a** and **b**, that place into the same edge of the same placement tree but with different branch lengths (Figure A.1), and how SCAMPP uses the branch lengths in this determination.

The backbone tree (shown at the top) is the same, but the two query sequences are different. In the top row we see the outcome of Stage 1, so that both query sequences **a** and **b** identify the same nearest taxon l and also extract the same placement subtree. Note that the edges in the placement subtree correspond to paths in the backbone tree, and the lengths of the edges in the placement subtree are computed by summing up the lengths of the edges they correspond to in the backbone tree. That is, the placement subtree and its edge lengths are obtained by inducing the subtree using the leafset and then summing up the relevant edge lengths. This correspondence and the fact that edge lengths are defined in this manner is used in the placement into the backbone tree, as the rest of this discussion illustrates.

In Stage 2, the two query sequences **a** and **b** are placed into the same edge (which we will call e') of the placement subtree, and e' has length 3.0 in the placement subtree. Note that the query sequence on the left is placed at a different location on e' (in terms of length along the edge) than the query sequence on the right: the query sequence on the left divides the edge length into 1.8 and 1.2 while the query sequence on the right divides the edge length into 2.7 and 0.3.

To describe Stage 3, we need to show how SCAMPP finds the associated path $Path(e')$ in the backbone tree corresponding to e', and then how, for each query sequence, it uses the position along the edge in e' in the placement tree to find exactly which edge in the backbone tree should have the query sequence, and exactly where along that edge in the backbone tree the query sequence should be placed.

Note that the edge e' in the placement subtree defines the bipartition $\pi_{e'} = \{P, O\}|\{S, U, V, l\}$ on the placement subtree. Furthermore, edge e' extends between two (unlabelled) nodes which we will refer to in this description as J and K, with J above K in the figure (so J is adjacent to both P and O in the placement subtree, and K separates nodes V, S, U, l from

the rest of the leaves in the placement subtree). Note that the backbone tree contains the placement tree, and so has nodes J and K as well. In examining the path in the backbone tree between nodes J and K, we see that it has three edges. In other words, the path in the placement tree between J and K is a single edge, but the path in the backbone tree between J and K has three edges. For the sake of exposition, we will refer to these three edges according to their position along the path (as depicted in the Figure) as "top", "middle", and "bottom". Thus the top edge includes endpoint J, the bottom edge includes endpoint K, and the middle edge has neither. The top edge has length 0.8, the middle edge has length 1.6, and the bottom edge has length 0.6. By construction, these edge lengths sum up to 3.0, the length of edge e'.

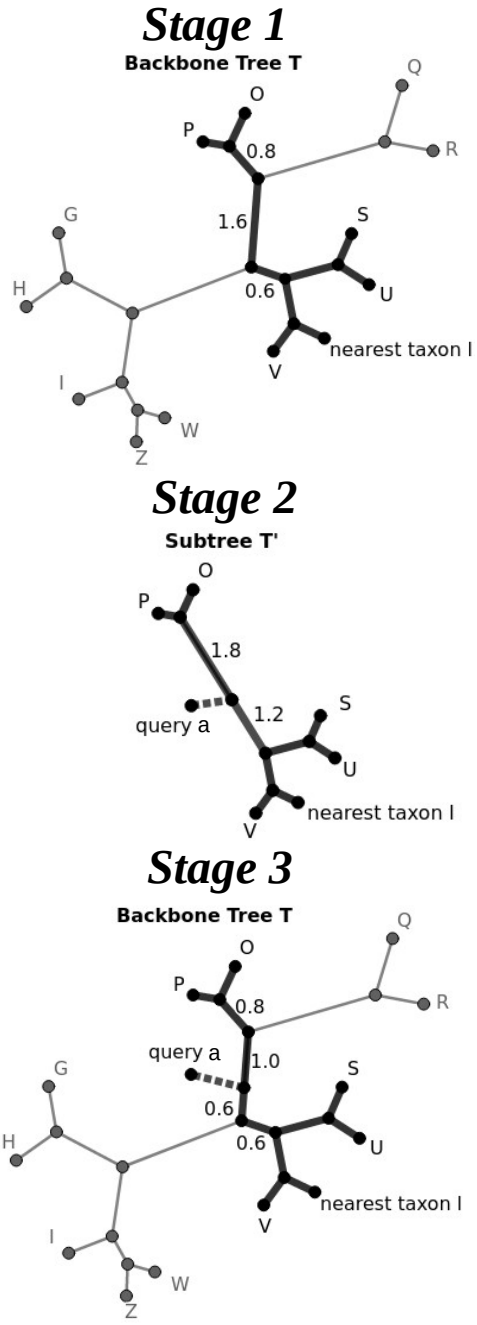
Recall that SCAMPP finds the path $Path(e')$ corresponding to edge e' by examining the bipartitions in the backbone tree that induce the same bipartition $\pi_{e'}$. The backbone tree has additional leaves beyond those in the placement subtree, but exactly three of the edges in the backbone tree produce bipartitions that induce $\pi_{e'}$. Those edges are exactly the edges on the path between J and K in the backbone tree. Thus, another way of describing how SCAMPP finds the path $Path(e')$ in the backbone tree corresponding to edge e' in the placement tree is by defining the two endpoint nodes for e', and then looking at the corresponding nodes in the backbone tree and the path between them.

Now we show how SCAMPP uses the length along e' where the query sequence is placed to determine where to insert the query sequence into the backbone tree. The two query sequences are both placed into edge e' but at different locations (in terms of branch lengths): the query sequence on the left splits 3.0 into 1.8 and 1.2, and the query sequence on the right splits 3.0 into 2.7 and 0.3.

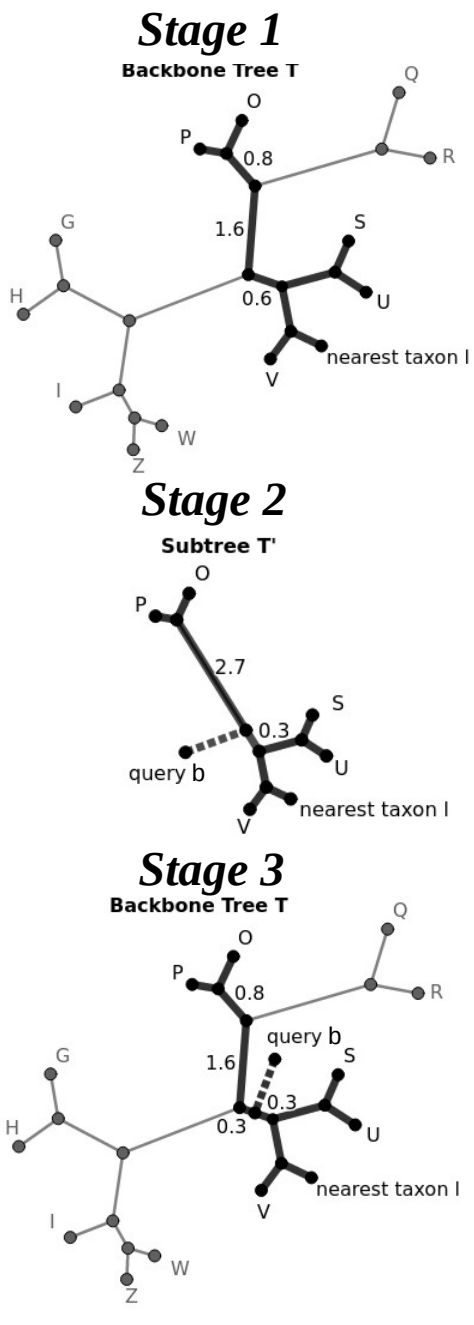
Consider first the query sequence **a**, for Figure A.1(a). It is inserted into the edge e' at a length 1.8 from the node J. Therefore, to find where to place it into the backbone tree, we traverse the path $Path(e')$, and we find that the middle edge (which has length 1.6) would be selected, and the query sequence **a** would be placed at length 1.0 from the top of that edge (i.e., the endpoint closer to J).

To place the other query sequence, **b**, we do the same analysis, but this time we need to find the edge in $Path(e')$ having distance 2.7 from node J (see Figure A.1(b)). This returns the bottom edge in $Path(e')$, and the location along that edge for inserting the query sequence from the right hand side would be 0.3 from each of its endpoints.

Thus, by using the branch lengths, the exact location in the backbone tree can be found. The key to this working is that the placement tree, along with its branch lengths, are defined by the backbone tree; otherwise this would not work as simply.



(a) Inserting query sequence **a**



(b) Inserting query sequence **b**

Figure A.1: Using SCAMPP to insert two different query sequences, **a** and **b**, into the same backbone tree. Stage 1: Each query sequence picks the same nearest taxon and extracts the same placement subtree. Stage 2: each query sequence is placed into the same edge, but at a different location along that edge as indicated by the way the length (3.0) of the edge is distributed. Stage 3: the two query sequences end up in different edges in the backbone tree. See text for additional details.