BATCH-SCAMPP: Scaling phylogenetic placement methods to place many sequences

Eleanor Wedell, Chengze Shen, Tandy Warnow[1*]

[1]*Dept. of Computer Science, University of Illinois Urbana-Champaign, Urbana IL, 61801, USA*

**warnow@illinois.edu*

Abstract

Phylogenetic placement, the problem of placing sequences into phylogenetic trees, has been limited either by the number of sequences placed in a single run or by the size of the placement tree. The most accurate scalable phylogenetic placement method with respect to the number of query sequences placed, EPA-ng (Barbera et al., 2019), has a runtime that scales sub-linearly to the number of query sequences. However, larger phylogenetic trees cause an increase in EPA-ng's memory usage, limiting the method to placement trees of up to 10,000 sequences. Our recently designed SCAMPP (Wedell et al., 2021) framework has been shown to scale EPA-ng to larger placement trees of up to 200,000 sequences by building a subtree for the placement of each query sequence. The approach of SCAMPP does not take advantage of EPA-ng's parallel efficiency since it only places a single query for each run of EPA-ng. Here we present BATCH-SCAMPP, a new technique that overcomes this barrier and enables EPA-ng and other phylogenetic placement methods to scale to ultra-large backbone trees and many query sequences. BATCH-SCAMPP is freely available at `https://github.com/ewedell/BSCAMPP_code`.

Keywords: Phylogenetic Placement, EPA-ng, Phylogenetics.

1 Introduction

Phylogenetic placement is the problem of placing one or more query sequences into a phylogenetic "backbone" tree, which may be a maximum likelihood tree on a multiple sequence alignment for a single gene or a taxonomy with leaves labeled by sequences for a single gene [23]. The most accurate techniques for phylogenetic placement use likelihood-based analyses and can be computationally intensive when the backbone trees are large. This application occurs when updating existing gene trees with newly observed sequences. However, phylogenetic placement can also be applied in the "bulk" context, where many sequences are placed at the same time into the backbone tree. For example, it can be applied when seeking to taxonomically characterize shotgun sequencing reads generated for an environmental sample in metagenomic analysis. Thus, phylogenetic placement is a general tool with applications in both incremental tree building and taxon identification and abundance profiling in microbiome analysis [6, 9, 10, 18, 21, 23]. In these analyses, however, the number of sequences placed into the backbone tree can be very large, especially when attempting to place all the shotgun sequencing reads in the sample. One way to reduce the number of reads for placement is to restrict the input to only those reads that map to one of the specified marker genes (which are genes universally found but appear only once in a given genome). Despite the restriction to a subset of the input data, using marker genes leads to improved accuracy in abundance profiling [21]. However, these restrictions can still result in thousands to tens of thousands of reads being placed in a single run into backbone trees.

Another aspect impacting the runtime of phylogenetic placement is the size of the backbone tree. Since many studies have shown the improvement in accuracy for abundance profiling, phylogenetic tree estimation, etc., when placing into large backbone trees [23], the importance of using phylogenetic placement methods that can run on large datasets is well established. Moreover, not all phylogenetic placement methods can scale to large backbone trees (and many methods are computationally intensive on large trees) [4, 27, 28], which has led to the development of methods, such as APPLES-2 [1], which use distances to place into large trees. There are also methods for phylogenetic placement that are alignment-free, such as those methods that use k-mers [7, 15]. These methods are much faster and more scalable than likelihood-based analyses, such as pplacer [17] and EPA-ng [5]. That said, likelihood-based methods are more accurate than these alignment-free methods and the distance-based methods [28]. Therefore, both the number of query sequences (i.e., sequences that need to be placed into trees) and the size of the backbone tree represent challenges for phylogenetic placement methods, especially when seeking to use likelihood-based placement methods.

Previously we introduced the SCAMPP framework to enable both pplacer and EPA-ng to perform phylogenetic placement into ultra-large backbone trees and demonstrated its utility on backbone trees with up to 200,000 sequences. By using maximum likelihood methods pplacer or EPA-ng within the SCAMPP framework, the resulting placements are more accurate than with APPLES-2, with the most notable accuracy improvement for fragmentary sequences, and are computationally similar for single query sequence placement [28]. However, SCAMPP was designed to incrementally update a large tree, one sequence at a time, and was not optimized for the other uses of phylogenetic placement, where batch placement of many sequencing reads is required.

EPA-ng is optimized for scaling the number of query sequences and is capable of placing millions of sequences into phylogenetic trees of up to a few thousand sequences [5]. EPA-ng performs this batch placement very efficiently, achieving sublinear runtime in the number of query sequences (see Figure 2 from [1]). However, other studies have shown that EPA-ng has a large memory requirement that depends on the size of the backbone tree; hence, the backbone trees used with EPA-ng are typically limited to a few thousand sequences. Thus, APPLES-2 has been the most scalable method capable of fast placement of many query sequences into ultra-large backbone trees [1].

Here we introduce BATCH-SCAMPP, a technique that improves scalability in both dimensions: the number of query sequences being placed into the backbone tree and the size of the backbone tree. BATCH-SCAMPP is based on SCAMPP, our prior method for scaling phylogenetic placement methods to large backbone trees. However, by its design, SCAMPP is not suitable for batch placement, so a significant modification to its design was required. Specifically, SCAMPP operates one query sequence at a time. Given the query sequence, SCAMPP finds and extracts a single (small) placement subtree out of the large backbone tree and then uses the selected phylogenetic placement method to place the query sequence into that placement subtree. Then, by superimposing the placement tree into the backbone tree and considering branch lengths, SCAMPP can identify the correct location within the original backbone tree to add the query

sequence. While this approach enables phylogenetic placement methods to scale to ultra-large backbone trees, the fact that a different subtree is selected for each query sequence makes this approach ineffective for BATCH placement.

Our BATCH-SCAMPP method operates by allowing the input set of query sequences to suggest and then vote on placement subtrees, thus enabling many query sequences to select the same placement subtree. We pair BATCH-SCAMPP with EPA-ng to explore the capability of this approach for scaling to many query sequences. We show that this combination of techniques (which we call BSCAMPP) not only provides high accuracy and scalability to large backbone trees, matching that of SCAMPP, but also achieves the goal of scaling sublinearly in the number of query sequences. Moreover, it is much more scalable than EPA-ng and faster than SCAMPP-EPA-ng: when placing 10,000 sequences into a backbone tree of 50,000 leaves, **EPA-ng is unable to run due to memory issues, SCAMPP requires 1628 minutes, and BSCAMPP places all sequences in 14 minutes (all given the same computational resources, see Table 1)**.

The rest of the paper is organized as follows. We describe the overall divide-and-conquer approach of the BSCAMPP algorithm in Section 2. Since this approach requires algorithmic parameters to be set, we select the algorithmic parameters using a collection of simulated datasets (our training datasets) and then evaluate additional testing datasets. The design of this experimental study is described in Section 3. The study results and comparison to other phylogenetic placement methods are shown in Section 4. We provide a discussion of results in Section 5, and we finish with conclusions in Section 6.

2 BSCAMPP

Our goal was to develop a divide-and-conquer strategy that allowed EPA-ng to scale datasets with large backbone trees (however, this approach could be used with any phylogenetic placement method). We assume that we have an input backbone tree T and its associated multiple sequence alignment and a set Q of query sequences that we will place into the tree T. At a high level, our approach has four steps: (a) the sequences in Q vote for their top v placement subtrees, each of bounded size B (both parameters specified by the user); (b) we assign each query in Q to one of the placement subtrees; (c) we run EPA-ng on the placement subtrees to add the assigned query sequences into the subtrees; (d) for each extended placement tree (so that it contains all of its query sequences), we superimpose the extended placement tree onto the backbone tree and use branch lengths to determine where to add the query sequences. This approach is very similar to SCAMPP, except that in SCAMPP, each query sequence picks a single placement subtree, there is no effort to group many query sequences into the same placement subtree, and each query sequence is placed independently into its placement subtree. Thus, SCAMPP fits into this framework by setting $v = 1$ (i.e., each query sequence votes for just one subtree), and each query is assigned to its preferred subtree.

In a collection of initial experiments (all on the same training data), we explored different ways of implementing the same general divide-and-conquer pipeline (see Section S3 in Supplementary Materials). These variants differ mainly in step (b), which uses the votes to determine the subtree to assign each query sequence. What makes these new variants different is the elaboration on step (b), which we explore in the Supplementary Materials. We selected one such way of running step (b) for further analysis, which we call BATCH-SCAMPP or BSCAMPP. This approach has two algorithmic parameters (subtree size B and the number v of votes), which we then explore here to determine the impact on accuracy and runtime.

BSCAMPP uses a leaf-voting strategy on the backbone tree to inform Stage 2, i.e., the subtree selection process (see Figure 1). We begin by defining the tree of the B nearest leaves to a given leaf l; this is obtained using a breadth-first search starting at the leaf l and referencing the branch lengths. To allow query sequences to vote for subtrees, we just let them vote for leaves, understanding this association between leaves and subtrees. Therefore, BSCAMPP has every query sequence vote for the v nearest leaves (using Hamming distances), thus also defining an associated set of subtrees. The BSCAMPP procedure is described in detail below.

- **Stage 1 - Hamming Distance Voting**: Each query sequence votes for v leaves based on the v closest leaves by Hamming distance in the backbone tree.
- **Stage 2 - Query Subtree Selection**: While there are query sequences not assigned to any subtree, choose the most voted leaf in the tree, build a subtree using a breadth-first search, collecting nearby

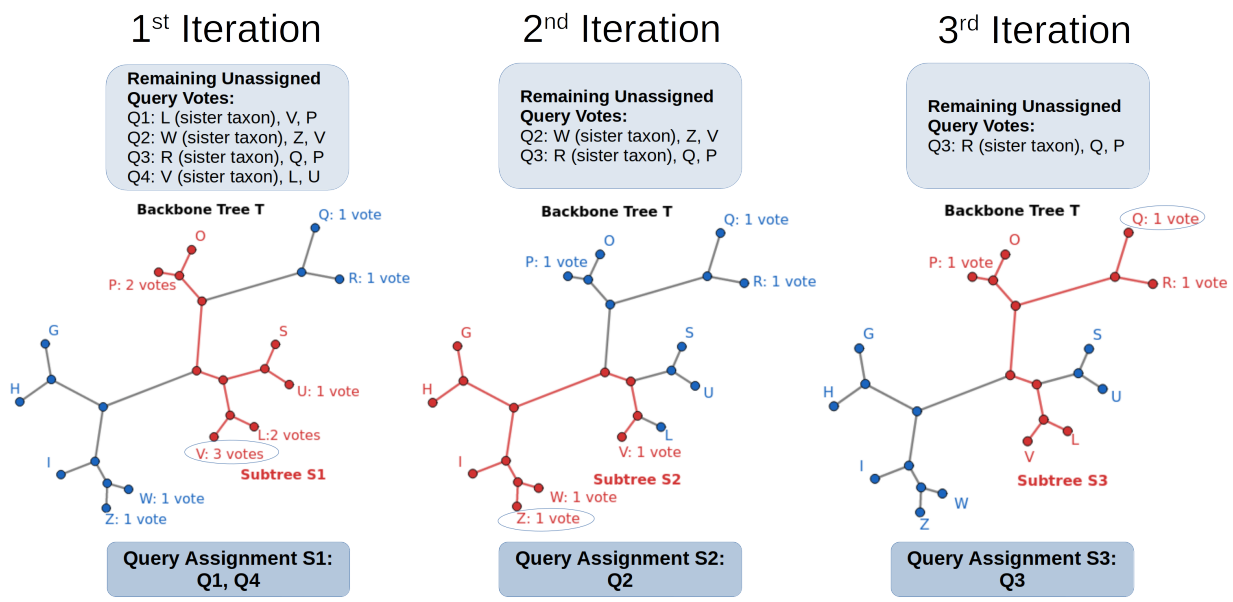


Figure 1: Toy example of the BSCAMPP procedure for building subtrees (Stage 2) with four query sequences. In the initial iteration (left), backbone tree T's leaf V has the most votes (3 votes), and is used as the seed sequence for subtree $S1$, containing leaves P, O, V, L, U, and S. Queries $Q1$ and $Q4$ are assigned to $S1$ because all of their voted leaves are in the $S1$. Votes from $Q1$ and $Q4$ are then removed from the backbone tree (center). Now the most number of votes for any leaf is 1, and leaf Z (1 vote) is arbitrarily chosen as subtree $S2$'s seed sequence, containing leaves G, H, I, Z, W, and V. $Q2$'s votes are all contained in $S2$, so $Q2$ is assigned to subtree $S2$ and $Q2$'s votes are removed. Finally, in the backbone tree (right), again the votes are tied at 1, and leaf Q (1 vote) is arbitrarily chosen as a seed sequence for subtree $S3$, containing leaves Q, R, P, O, V, and L. $Q3$ is assigned to subtree $S3$.

leaves from the full tree, until B leaves are in the subtree. Assign queries to the subtree if all of their votes are contained within that subtree or if the leaf with the closest Hamming distance to the query is used to start the breadth-first search. Remove the votes from query sequences after their assignments.

- **Stage 3 - Run EPA-ng on Subtrees**: Run EPA-ng on the selected subtrees and their assigned query sequences.
- **Stage 4 - Return Full Backbone Tree Placement**: Determine each query sequence's placement in the full backbone tree (using branch lengths in the placement and full backbone tree), and return a file containing all placements, with their requisite confidence score, distal length, placement edge number, etc.

3 Performance Study Design

Overview We ran three experiments for this study; the first two are essentially parameter selection experiments based on the training data, and the third is where we evaluate the method (BSCAMPP) compared to other methods. For the third experiment, we use testing data, but we also demonstrate accuracy in comparison to existing methods on the training data. Here we describe the experimental protocol (see Supplementary Materials for full details).

In Experiment 0 (our pilot study), we tested EPA-ng on simulated data to evaluate its scalability with respect to placement tree size. This experiment revealed that EPA-ng accuracy reduces on large placement trees when placing fragmentary sequences (but not as much for full-length sequences), which led us to constrain the size of the placement subtrees differently for full-length and fragmentary sequences. The next experiment (Experiment 1) was for selecting the algorithmic parameters for BSCAMPP. These two

experiments were performed on the training data. In Experiment 2, we compared BSCAMPP to other placement methods, including EPA-ng, APPLES-2, and SCAMPP (also used with EPA-ng); we show these results on testing data (disjoint from the training data) as well as on the training data. Our training data are simulated datasets with known true trees, and the testing data are both biological datasets (with estimated maximum likelihood trees) as well as simulated datasets. We report placement error using "delta error", a standard metric used in prior studies [3, 20].

Datasets We used both simulated and biological datasets. For simulated datasets, we use samples from the RNASim dataset [12] (also used in [1, 4, 28]), which is a collection of 1 million sequences with a known true multiple sequence alignment and true tree, where the sequences have evolved using a complex model of evolution. The RNASim one million sequence tree is partitioned into two by extracting a clade of approximately 600,000 sequences for testing (discussed below). This defines two sets of sequences, one of which will be used for training and the other for testing. We also use the nt78 datasets, which were simulated for FastTree-2 [22]; these contain 10 replicates, simulated with Rose [26], each with 78,132 sequences in a multiple sequence alignment and the simulated backbone tree.

For biological dataset analysis, we use 16S.B.ALL, a large biological dataset with >27,000 sequences and a reference alignment based on structure from CRW [8]. A maximum likelihood tree has been published for this dataset that was computed on this reference alignment using RAxML [25]; this serves as the "reference tree" for this dataset and was reported in [16]. The reference tree and alignment are available at [19]. For the fragmentary sequence experiments, fragments are generated from the full-length sequences starting at a randomly selected location. The fragmentary sequences are a mean of 10% of the original ungapped sequence length with a standard deviation of 10 nucleotides. For the pilot study with EPA-ng, we make fragments $[10, 25, 50, 100]$% of the original lengths without variance in length. These datasets were used both for training experiments (Experiments 0 and 1) and for testing experiments (Experiment 2).

Training Datasets We use $\sim 400,000$ sequences from the RNASim dataset, from which we randomly select 50,000 sequences to define the backbone sequences, 10,000 sequences for query sequences, and the remaining 340,000 sequences are not used. Sites of the true alignment on the 60,000 sequences containing more than 95% gaps are removed. To define the backbone tree on the set of sequences, we run FastTree-2 under the GTRGAMMA model on the true alignment.

For the nt78 datasets, we use the third replicate for the experiment. As the true alignment is not very gappy, we do not perform any masking. We pick 10,000 sequences at random for the query sequences and use the remaining 68,132 sequences as backbone sequences. We also define the backbone tree by running FastTree-2 on the true alignment of the backbone sequences.

Testing Datasets The testing datasets include the larger clade from the RNASim collection, processed exactly the same way as the training datasets. We also include 16S.B.ALL [8], a large biological dataset with 27,643 sequences with an established structural alignment. 16S.B.ALL contains some duplicate sequences; these were removed before analysis, producing a dataset with 25,093 sequences. Of these, 5,093 sequences were randomly selected as query sequences. We use the published "reference tree" (a maximum likelihood tree estimated on the reference alignment) as the backbone tree.

Evaluation Criteria We report placement error using average delta-error, which is the increase in the number of false negatives (FN) produced by placing query sequences into the backbone tree. This requires the definition of the "true tree", which is the model tree for the simulated data and the published reference tree for the biological data. The Delta Error has been used in many previous studies [1,28]; see the supplementary materials for the formal definition of delta error.

The methods are also evaluated with respect to runtime and peak memory usage. All methods are given four hours to run with 64 GB of memory and 16 cores. For SCAMPP, when placement time was over four hours (which occurred in all experiments with 10,000 query sequences), the query sequences were split into 40 subsets of 250 sequences each. SCAMPP was then run 40 times for each 250 query sequence set.

Numeric parameter estimation for placement methods Each phylogenetic placement method requires numeric parameters to be estimated for the backbone trees. All backbone tree numeric parameters (branch lengths, 4×4 substitution rate matrix, etc.) are re-estimated according to the specifications of

the phylogenetic placement method: RAxML-ng [14] for SCAMPP and BSCAMPP, and FastTree-2 under Minimum Evolution for APPLES-2.

4 Results

4.1 Experiment 0 (Pilot Study): Impact of backbone tree size for EPA-ng

The goal of this experiment was to understand the impact of the backbone tree size for EPA-ng using both fragmentary and full-length query sequences. We extracted a subset of the RNASim training dataset for this experiment. The subset is a clade with 10,200 leaves of the training RNASim dataset, from which we randomly selected 200 leaves as queries. For the remaining 10,000 leaves, we randomly selected from 1000 to 10,000 leaves to form the backbone tree. For the queries, we made them into fragments of 10%-length to full-length. All queries were placed into the backbone trees using EPA-ng in one of two modes: "batch", where we place all 200 queries all at once (a feature allowed by EPA-ng), and "one-by-one", where we place each query independently by running EPA-ng on each query.

We show results on various backbone tree sizes, query fragmentation levels, and placement strategies (batch vs. one-by-one) in Figure S1 in the Supplementary Materials. For batch placement, note that phylogenetic placement accuracy is very much impacted by query length, with full-length sequences placed well at all explored backbone tree sizes (up to 10,000-leaf trees). However, fragmentary sequences are only placed well on small backbone trees (up to 2,000 leaves). Interestingly, this is not the case when using one-by-one placement, where all phylogenetic placement error rates remained low, even for fragmentary sequences. More specifically, we see a jump in delta error for "batch" from backbone tree size of 2000 to 5000 and up. This trend is also observed with less fragmentary queries (e.g., fragments between 25% and 50% of the original length), except that the gap in accuracy between the two strategies becomes smaller. There is no difference in accuracy between the two strategies for all backbone tree sizes on full-length queries.

In sum, this experiment shows that EPA-ng performs very differently between the batch mode (which is where it gets the speed-up) and one-by-one mode, and that it can be much more accurate in one-by-one mode when placing fragmentary sequences. The key lesson is that if we wish to obtain good accuracy in placing fragmentary sequences, we need to either use EPA-ng in one-by-one mode or keep the backbone tree at most 2000 sequences.

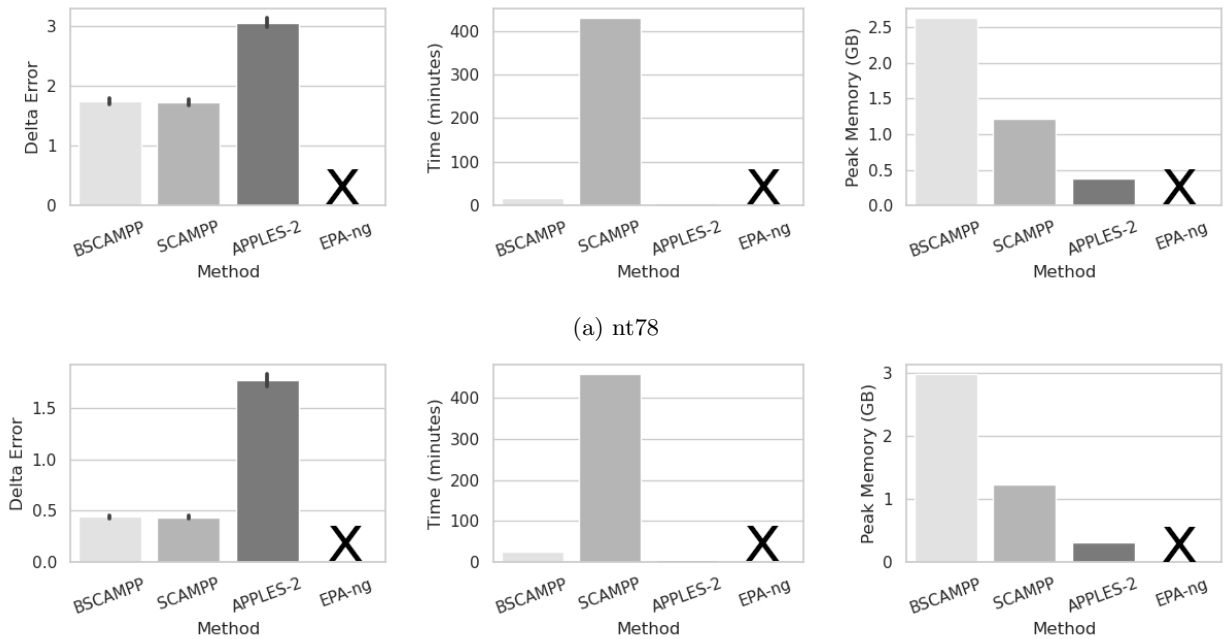
4.2 Experiment 1: BSCAMPP design with training datasets

In order to understand how to set the size of the subtree and the number of votes for BSCAMPP, we performed a number of experiments on the two training data sets for both full-length and fragmentary sequences. We varied the subtree parameter setting from 1000, 2000, and 10,000 leaves. For each subtree size, we ran BSCAMPP with 5, 10, 15, 20, and 25 votes per query sequence. The results of these datasets (Supplementary Figures S4-S7 and Tables S1 and S2) reveal the following trends:

- For full-length query sequences, increasing the subtree size from 1000 to 2000 and then to 10,000 leaves leads to a (minute) improvement in placement accuracy. In the case of RNASim, the runtime on placement trees with 10,000 leaves was substantially higher than on 2000 leaves, leaving us to select subset size 2000. Increasing the number of votes led to a slight improvement in accuracy but a larger increase in runtime. However, the runtimes were low enough to allow 10 votes.
- For fragmentary query sequences, using placement subtrees of size 10,000 led to a huge increase in placement error, consistent with Experiment 0, and the best accuracy (but not by a large amount) was obtained on subtree size 2000. Increasing the number of votes improved accuracy but led to a huge increase in runtime for the RNASim dataset. Hence we selected subtree size 2000 and 5 votes for the fragmentary sequences.

4.3 Experiment 2: Comparing BSCAMPP to other methods

In this experiment, we compare our BSCAMPP framework with the selected parameters from Experiment 1 to SCAMPP [28], APPLES-2 [2], and EPA-ng [5] when placing full-length or fragmentary sequences into



(a) nt78

(b) RNASim

Figure 2: Experiment 2: Method Comparison on Training Datasets for Fragmentary Sequences. Top: Mean Delta Error (left), runtime (center), and peak memory usage in GB (right) for placement of all 10,000 fragmentary query sequences on the nt78 backbone trees with 68,132 leaves. Bottom: Mean Delta Error (left), runtime (center), and peak memory usage in GB (right) for placement of all 10,000 fragmentary query sequences on the RNASim backbone trees with 50,000 leaves. We show placement time for BSCAMPP for 5 votes. The results from APPLE-2 and SCAMPP are also included for runtime, peak memory usage, and delta error. EPA-ng was not shown because it was unable to run given 64 GB of memory and 16 cores on these datasets due to out-of-memory issues.

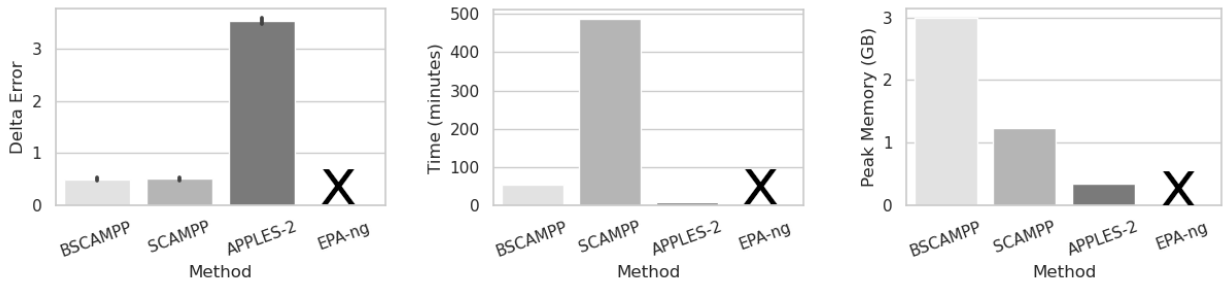
a backbone tree. We first compare delta error, runtime, and memory on the training datasets (RNASim for training and nt78) and then on the testing datasets (RNASim for testing and 16S.B.ALL).

Method Comparison on Training Datasets For all datasets, SCAMPP was unable to finish within four hours. Thus, the query sequences were broken up into 40 subsets, and SCAMPP was run 40 times in order to allow for a comparison. For this reason, the process of loading the alignment and the backbone tree is repeated 40 times. However, this amounts to a difference of 2 seconds per run. Therefore, the runtime results may be between one and two minutes longer than a normal sequential run.

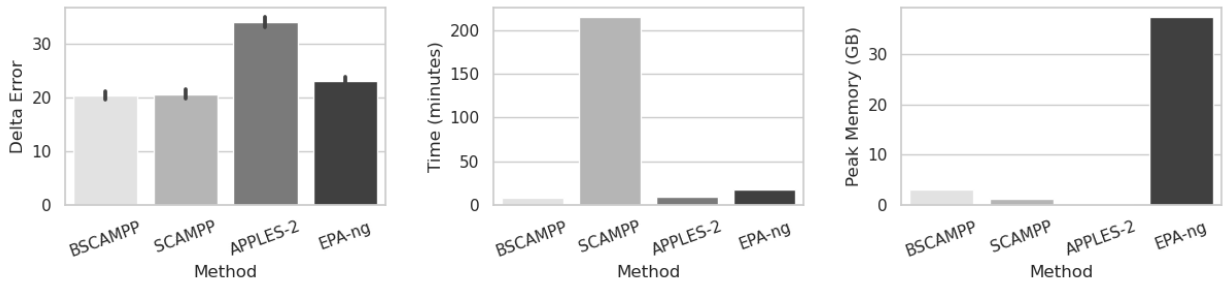
Fragmentary sequence results on the training datasets (Figure 2) show a lower delta error for both SCAMPP and BSCAMPP than for APPLES-2. BSCAMPP is able to maintain a similar accuracy to SCAMPP while taking a fraction of the runtime in all cases. BSCAMPP places all 10,000 query sequences in under 30 minutes for both datasets, which is a substantial improvement from SCAMPP (which takes between 4 and 8 hours depending on the dataset and subtree size used) although not as fast as APPLES-2 at around five minutes. Finally, all three methods have peak memory usage much less than the 64 GB memory limit (BSCAMPP at 3 GB, SCAMPP at 1 GB, and APPLES-2 at .5 GB).

The results for full-length sequences can be found in Supplementary Materials Figure S8 but show similar improvements to runtime using BSCAMPP compared to SCAMPP while maintaining the accuracy obtained by SCAMPP. We note that EPA-ng failed to run on all training datasets for both full-length and fragmentary sequences due to out-of-memory issues.

Method Comparison on Testing Datasets All methods are now run using their default settings, meaning both BSCAMPP and SCAMPP use a subtree size of 2000 sequences. On the RNASim dataset



(a) RNASim (50,000 sequences in the backbone tree)



(b) 16S.B.ALL (20,000 sequences in the backbone tree)

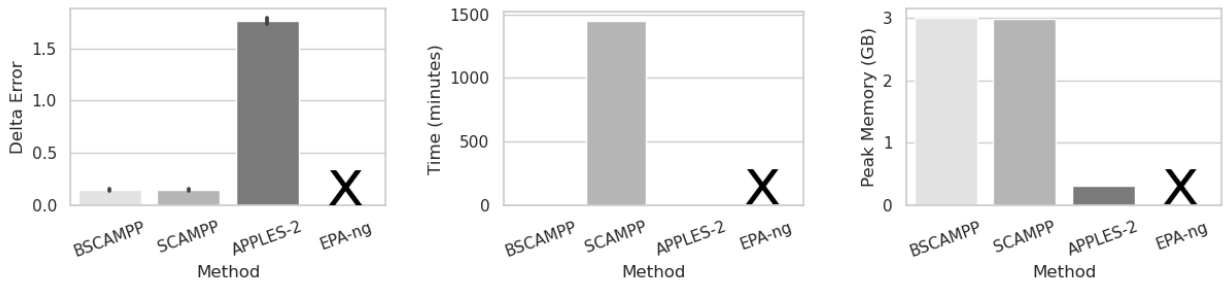
Figure 3: Experiment 2: Method Comparison on Testing Datasets for Fragmentary Sequences - Top: Mean Delta Error (left), runtime (center), and peak memory usage in GB (right) for placement of all 10,000 fragmentary query sequences on the RNASim backbone trees with 50,000 leaves. Bottom: Mean Delta Error (left), runtime (center), and peak memory usage in GB (right) for placement of all 5,093 fragmentary query sequences on the 16S.B.ALL backbone trees with 20,000 leaves. We show placement time for BSCAMPP for 5 votes with a subtree size of 2000. The results from APPLE-2 and EPA-ng are also included (an X indicates that EPA-ng was unable to run). SCAMPP was run for a subtree size of 2000.

with a backbone tree size of 50,000 sequences, EPA-ng failed to run given 64 GB of memory; however, on the 20,000 leaf backbone tree in the 16S.B.ALL dataset EPA-ng was able to place all sequences within the memory limitations.

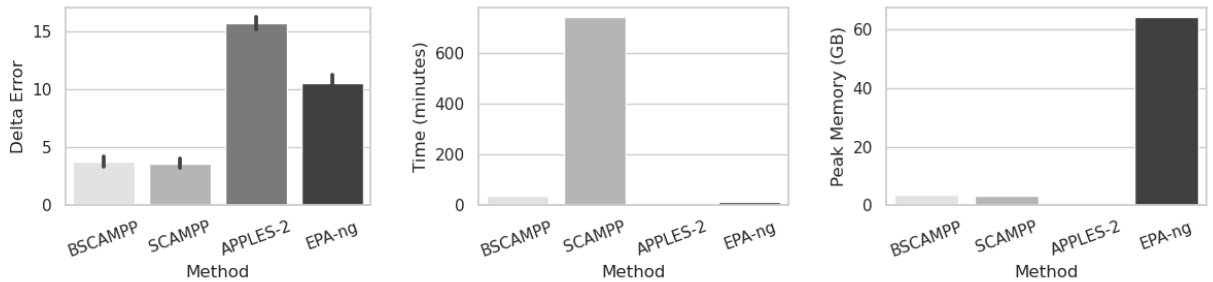
We show results for fragmentary sequences in Figure 3. BSCAMPP obtains the most accurate placements with the lowest mean error for both datasets. SCAMPP shows similar accuracy to BSCAMPP. Interestingly, EPA-ng shows higher error than SCAMPP and BSCAMPP on the 16S.B.ALL dataset, showing that using EPA-ng within BSCAMPP or SCAMPP can improve placement accuracy. Finally, all maximum likelihood based methods (EPA-ng, SCAMPP, and BSCAMPP) show lower delta error than APPLES-2.

The runtime results show different trends depending on the dataset. For 16S.B.ALL, BSCAMPP is the fastest method at 8 minutes, followed by APPLES-2 at 9, EPA-ng at 17, and SCAMPP at 214 minutes. APPLES-2 is the fastest method on RNASim at 9 minutes, followed by BSCAMPP at 53 minutes and SCAMPP at 486 minutes. While the order of the faster methods varies, SCAMPP is consistently the slowest method for batch query sequence placement. Results for the peak memory usage show that all methods except EPA-ng (at 37 GB) use at most 3 GB of memory.

Full-length sequence results (Figure 4) show similar trends to fragmentary sequences. Notably, full-length sequences show a larger decrease in error for BSCAMPP and SCAMPP vs. EPA-ng than fragmentary sequences. The runtime for SCAMPP is over three times longer when placing full-length sequences than for fragmentary sequences, and all other methods take a small fraction of the runtime. Finally, we note that EPA-ng uses all 64 GB of memory for full-length sequence placement on 16S.B.ALL, while all other methods continue to use under 3 GB for all datasets.



(a) RNASim (50,000 sequences in the backbone tree)



(b) 16S.B.ALL (20,000 sequences in the backbone tree)

Figure 4: Experiment 2: Method Comparison on Testing Datasets for Full-length Sequences - Top: Mean Delta Error (left), runtime (center), and peak memory usage in GB (right) for placement of all 10,000 full-length query sequences on the RNASim backbone trees with 50,000 leaves. Bottom: Mean Delta Error (left), runtime (center), and peak memory usage in GB (right) for placement of all 5,093 fragmentary query sequences on the 16S.B.ALL backbone trees with 20,000 leaves. We show placement time for BSCAMPP for 10 votes with a subtree size of 2000. The results from APPLE-2 and EPA-ng are also included (an X indicates that EPA-ng was unable to run). SCAMPP was run for a subtree size of 2000.

Table 1: Runtime comparison of BSCAMPP and SCAMPP on all datasets (backbone size is listed below each dataset). Both methods are run with their default settings (BSCAMPP: $B = 2000$ and $v = 5$ for fragments and $v = 10$ for full-length sequences, SCAMPP: $B = 2000$). Runtimes are rounded to the nearest minutes.

Method	Runtime (minutes)							
	Full-length Queries				Fragmentary Queries			
	Testing		Training		Testing		Training	
	RNASim 50,000	nt78 68,132	RNASim 50,000	16S.B.ALL 20,000	RNASim 50,000	nt78 68,132	RNASim 50,000	16S.B.ALL 20,000
BSCAMPP	14	37	17	36	27	17	53	8
SCAMPP	1628	1413	1445	743	457	429	486	215

5 Discussion

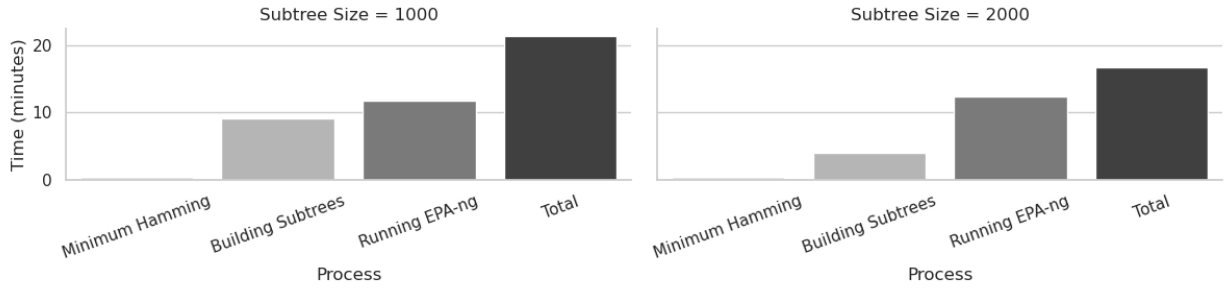
The main observation of this study is that BSCAMPP (used with EPA-ng) achieves the same accuracy as SCAMPP (also used with EPA-ng) but at a dramatically reduced runtime. BSCAMPP is also able to run on datasets where EPA-ng fails. Furthermore, on large backbone trees where EPA-ng can run when placing many query sequences, EPA-ng has higher error than BSCAMPP.

To understand why we see these trends, we consider what we learned about EPA-ng scalability on large backbone trees. Many studies have suggested limits for EPA-ng to relatively small backbone trees [1, 4, 28], but here we did observe its ability to run on the 16S.B.ALL backbone tree (with 20,000 sequences). The

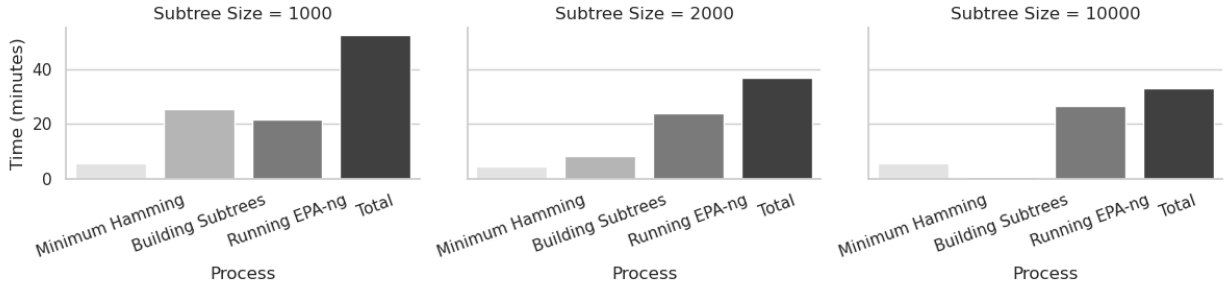
issue for EPA-ng is its large memory usage, which can explain why EPA-ng is limited to backbone trees of at most moderate size. In contrast, by limiting the placement tree size to at most 2000 sequences, BSCAMPP can place into very large backbone trees. While this study only examined backbone trees with up to 68,132 sequences, prior work on SCAMPP used with EPA-ng in [28] showed it could scale to 200,000-leaf backbone trees. Thus, this study suggests that this approach should also scale to ultra-large backbone trees and a large number of query sequences.

While we did not see any increase in error for EPA-ng for full-length sequences on smaller query set sizes in the pilot study, Figure 4 does show that EPA-ng on the 16S.B.ALL dataset places 10,000 query sequences with more than twice the error than BSCAMPP and SCAMPP for full-length sequences. Future work should explore whether the EPA-ng heuristics for larger query sets can impact accuracy for full-length sequences beyond this initial observation.

Additionally, it is not clear why EPA-ng has poor accuracy run in batch mode in placing fragmentary sequences on large backbone trees, and future work is needed to understand this issue. Thus, we advise cautious usage of EPA-ng batch mode when placing many fragmentary sequences if the placement tree has more than 2000 leaves.



(a) Fragmentary Sequences on nt78 (68,132 sequences in the backbone tree and 10,000 queries)



(b) Full-length Sequences on nt78 (68,132 sequences in the backbone tree and 10,000 queries)

Figure 5: Runtime for three processes within BSCAMPP and the total runtime for placement of all 10,000 query sequences into the nt78 training backbone tree with 68,132 leaves. Top: Fragmentary query sequence placements. The subtree size is 1000 (left) and 2000 (right), and the number of votes per query is 5. Bottom: Full-length query sequence placements. The subtree size is 1000 (left), 2000 (center), and 10,000 (right), and the number of votes per query is 10.

We now discuss runtime and memory usage for BSCAMPP, and the ramifications for future development. Figure 5 shows the runtime breakdown of different processes within BSCAMPP for full-length and fragmentary sequences while varying the subtree size. In almost all cases, the process of running EPA-ng is the slowest, suggesting the importance of limiting the number of subtrees used by BSCAMPP. The process of building subtrees dominates more of the runtime for full-length sequences on the smallest subtree size. We note that as the subtree size increases, fewer subtrees are needed to capture all queries. However, EPA-ng will take longer to run on larger subtrees (and may also have higher error). This may explain why the 2000 leaf subtree size was the fastest (see Supplementary Table S1).

6 Conclusion

Phylogenetic placement into large backbone trees is fundamental to several bioinformatics problems, including microbiome analysis (e.g., taxonomic characterization of shotgun sequencing reads) and updating large phylogenetic trees. Yet, only a very small number of methods provide acceptable accuracy and scalability. Prior to this study, SCAMPP (used with maximum likelihood phylogenetic placement methods, such as pplacer or EPA-ng) had the best accuracy of all methods that could place into large backbone trees. However, SCAMPP could not be used to address the challenge of placing large numbers of query sequences. In this study, we showed that BSCAMPP matches the accuracy of the SCAMPP but is much faster for placing large numbers of query sequences. Thus, BSCAMPP is the first phylogenetic placement method that is scalable in both dimensions – the number of query sequences and the size of the backbone tree.

A possible improvement for the runtime can be explored by implementing parallel processing of subtrees (i.e., running instances of EPA-ng in parallel for different query-subtree sets). This might be particularly helpful in cases with few queries per subtree. Future work should include an analysis of the runtime and memory usage impacts of running EPA-ng concurrently on multiple compute nodes, in addition to running multiple instances of EPA-ng using fewer threads on a single compute node.

There are applications of phylogenetic placement methods that could be improved through the use of larger backbone tree sizes and many query sequences, now enabled with better accuracy through BSCAMPP. One such application is that of de novo phylogenetic tree building. Studies have shown that scalable phylogeny estimation methods have suffered in the presence of sequence length heterogeneity, and phylogenetic placement may provide a more accurate alternative [13]. For example, the divide-and-conquer tree estimation pipeline GTM [24] could benefit by using BSCAMPP. BSCAMPP can facilitate the initial tree decomposition of GTM for better placement of shorter, fragmentary sequences into an initial tree containing the longer full-length sequences, potentially leading to better final tree estimation.

Another application of using more scalable backbone trees with many query sequences would be for methods of taxon identification, such as TIPP2 [23]. TIPP2 typically needs to place thousands of fragmentary reads, and it is shown in its study to benefit from the use of larger backbone tree sizes. In addition, by implementing SCAMPP for the phylogenetic placement step within TIPP2, a comparison might be made between the accuracy of SCAMPP and the "Russian Doll" multi-level method for scaling phylogenetic placement based taxon identification to much larger reference databases [11]. Such applications of BSCAMPP are left to future studies.

Acknowledgement

This work was supported by the US National Science Foundation grant 2006069 (to TW), and by fellowship support from the Siebel Scholars Program, and the Computer Science Department at UIUC (to EW).

References

[1] Metin Balaban, Yueyu Jiang, Daniel Roush, Qiyun Zhu, and Siavash Mirarab. Fast and accurate distance-based phylogenetic placement using divide and conquer. *Molecular Ecology Resources*, 22(3):1213–1227, 2022.

[2] Metin Balaban, Daniel Roush, Qiyun Zhu, and Siavash Mirarab. APPLES-2: Faster and more accurate distance-based phylogenetic placement using divide and conquer. *bioRxiv*, 2021. DOI: 10.1101/2021.02.14.431150.

[3] Metin Balaban, Shahab Sarmashghi, and Siavash Mirarab. APPLES: distance-based phylogenetic placement for scalable and assembly-free sample identification. *bioRxiv*, page 475566, 2019.

[4] Metin Balaban, Shahab Sarmashghi, and Siavash Mirarab. APPLES: scalable distance-based phylogenetic placement with or without alignments. *Systematic Biology*, 69(3):566–578, 2020.

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

[6] Holly M Bik, Dorota L Porazinska, Simon Creer, J Gregory Caporaso, Rob Knight, and W Kelley Thomas. Sequencing our way towards understanding global eukaryotic biodiversity. *Trends in Ecology & Evolution*, 27(4):233–243, 2012.

[7] Matthias Blanke and Burkhard Morgenstern. App-SpaM: phylogenetic placement of short reads without sequence alignment. *Bioinformatics Advances*, 1(1), 10 2021. vbab027.

[8] Jamie J Cannone, Sankar Subramanian, Murray N Schnare, James R Collett, Lisa M D'Souza, Yushi Du, Brian Feng, Nan Lin, Lakshmi V Madabusi, Kirsten M Müller, Nupur Pande, Zhidi Shang, Nan Yu, and Robin 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*, 3(1):2, 2002.

[9] Pierre-Alain Chaumeil, Aaron J Mussig, Philip Hugenholtz, and Donovan H Parks. GTDB-Tk: a toolkit to classify genomes with the Genome Taxonomy Database. *Bioinformatics*, 36(6):1925–1927, 2020.

[10] Sean Conlan, Heidi H Kong, and Julia A Segre. Species-level analysis of DNA sequence data from the NIH Human Microbiome Project. *PloS one*, 7(10):e47075, 2012.

[11] Lucas Czech, Pierre Barbera, and Alexandros Stamatakis. Methods for automatic reference trees and multilevel phylogenetic placement. *Bioinformatics*, 35(7):1151–1158, 08 2018.

[12] Sheng Guo, Li-San Wang, and Junhyong Kim. Large-scale simulation of RNA macroevolution by an energy-dependent fitness model. *arXiv preprint arXiv:0912.2326*, 2009.

[13] Stefan Janssen, Daniel McDonald, Antonio Gonzalez, Jose A Navas-Molina, Lingjing Jiang, Zhenjiang Zech Xu, Kevin Winker, Deborah M Kado, Eric Orwoll, Mark Manary, et al. Phylogenetic placement of exact amplicon sequences improves associations with clinical information. *Msystems*, 3(3):e00021–18, 2018.

[14] Alexey M Kozlov, Diego Darriba, Tomáš Flouri, Benoit Morel, and Alexandros Stamatakis. RAxML-NG: a fast, scalable and user-friendly tool for maximum likelihood phylogenetic inference. *Bioinformatics*, 35(21):4453–4455, 2019.

[15] Benjamin Linard, Krister Swenson, and Fabio Pardi. Rapid alignment-free phylogenetic identification of metagenomic sequences. *Bioinformatics*, 35(18):3303–3312, 01 2019.

[16] Kevin Liu, Tandy J Warnow, Mark T Holder, Serita M Nelesen, Jiaye Yu, Alexandros P Stamatakis, and C Randal Linder. SATe-II: very fast and accurate simultaneous estimation of multiple sequence alignments and phylogenetic trees. *Systematic biology*, 61(1):90, 2012.

[17] Frederick A Matsen, Robin B Kodner, and E Virginia Armbrust. pplacer: linear time maximum-likelihood and Bayesian phylogenetic placement of sequences onto a fixed reference tree. *BMC bioinformatics*, 11(1):538, 2010.

[18] Connor O McCoy and Frederick A Matsen. Abundance-weighted phylogenetic diversity measures distinguish microbial community states and are robust to sampling depth. *PeerJ*, 1:e157, 2013.

[19] Siavash Mirarab. 16S.B.ALL reference tree and alignment. `https://sites.google.com/eng.ucsd.edu/datasets/alignment/16s23s`.

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

[21] Nam-phuong Nguyen, Siavash Mirarab, Bo Liu, Mihai Pop, and Tandy Warnow. TIPP: taxonomic identification and phylogenetic profiling. *Bioinformatics*, 30(24):3548–3555, 2014.

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

[23] Nidhi Shah, Erin K. Molloy, Mihai Pop, and Tandy Warnow. TIPP2: metagenomic taxonomic profiling using phylogenetic markers. *Bioinformatics*, 2021. DOI: 10.1093/bioinformatics/btab023.

[24] Vladimir Smirnov and Tandy Warnow. Unblended Disjoint Tree Merging using GTM improves species tree estimation. *BMC genomics*, 21:1–17, 2020.

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

[26] J Stoye, D Evers, and F Meyer. Rose: generating sequence families. *Bioinformatics*, 14(2):157–163, 03 1998.

[27] Eleanor Wedell, Yirong Cai, and Tandy Warnow. Scalable and accurate phylogenetic placement using pplacer-XR. In *International Conference on Algorithms for Computational Biology*, pages 94–105. Springer, 2021. `https://link.springer.com/chapter/10.1007/978-3-030-74432-8_7`.

[28] Eleanor Wedell, Yirong Cai, and Tandy Warnow. SCAMPP: Scaling alignment-based phylogenetic placement to large trees. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, pages 1–1, 2022. DOI:10.1109/TCBB.2022.3170386.