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

Eleanor Wedell, Chengze Shen, and Tandy Warnow

May 8, 2023

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`.

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 [9, 17, 21, 23, 7, 4]. 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) [28, 27, 2], 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 [13, 5]. These methods are much faster and more scalable than likelihood-based analyses, such as pplacer [15] and EPA-ng [3]. 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 [28] to enable both pplacer and EPA-ng to perform phylogenetic placement into ultra-large backbone trees, and we 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 query sequence at a time, and was not optimized for the other uses of phylogenetic placement, where batch placement of many sequencing reads is required. Furthermore, APPLES-2 has been improved since we designed SCAMPP!

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 [3]. 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+EPA-ng, or BSCAMPP(e)) 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 1421 minutes, and BSCAMPP(e) places all sequences in 7 minutes (all given the same computational resources, see Table 3)**.

The rest of the paper is organized as follows. We describe the overall divide-and-conquer approach of the BSCAMPP algorithm in Section 3. 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 4. The study results are shown in Section 5. We provide a discussion of results in Section 6, and we finish with conclusions in Section 7.

2 Understanding EPA-ng accuracy

The goal of this experiment was to understand the impact on EPA-ng placement accuracy of the backbone tree size, when placing short sequences (the metagenomics context, see Supplementary Materials for results when placing also full-length sequences). We extracted a subset of the RNASim training dataset (see Section 4.4) 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 ranging from 10%-length and 50%-length (results for 10%- and 25%-length are shown here, see Supplementary Materials for the full set of results). 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 delta error on various backbone tree sizes and placement strategies (batch vs. one-by-one) in Figure 1.

For both query sequence lengths, accuracy improves for one-by-one placement as the backbone size increases, indicating the beneficial impact of taxon sampling. Interestingly, this is not true for batch placement. For batch placement, phylogenetic placement accuracy is very much impacted by query length, but very fragmentary sequences are only placed well on small backbone trees (up to 2,000 leaves).

Since our interest is in placing short sequences (corresponding to read placement), we focus now on the results shown for queries of 10%-length. For these very short query sequences, we see a jump in delta error for "batch" from backbone tree size of 2000 to 5000 and up. Thus, high accuracy is only obtained when placing into smaller trees.

In sum, this experiment shows that EPA-ng performs very differently between the batch mode (the case where it is able to obtain a runtime advantage, according to prior studies) and the one-by-one mode. When the backbone tree is small enough then batch mode can be very accurate, but on larger backbone trees the one-by-one mode is more accurate for 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.

3 BATCH-SCAMPP

BATCH-SCAMPP (which we will call BSCAMPP for short) was designed with the goal of developing 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 stages: Stage 1: the sequences in Q vote for their top v placement subtrees, each of bounded size B (both parameters specified by the user); Stage 2: We construct a set $\mathcal{T}$ of placement subtrees, using the votes, and we assign each query in Q to one of the placement subtrees; Stage 3: We allow reassignments of query sequences to subtrees. Stage 4: we run EPA-ng on the placement subtrees to add the assigned query sequences into the subtrees, and then use branch lengths to find appropriate positions in the backbone tree T.

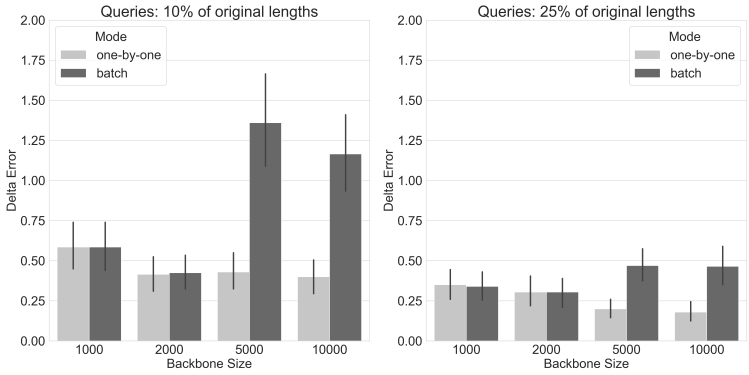


Figure 1: Mean delta error of EPA-ng when placing 200 queries into a backbone tree with 1000, 2000, 5000, and 10000 leaves of the training RNASim dataset. From left to right: query sequences (for placement) are made into fragments of 10% and 25%. We show two strategies for using EPA-ng for placement: "batch" means that all queries are placed at once, while "one-by-one" means that each query sequence is placed individually.

This four-stage approach is an elaboration on the SCAMPP technique, except that in SCAMPP, each query sequence picks a single placement subtree, and it is completely feasible that there will be as many placement trees as there are query sequences. Such an approach would fail to take advantage of the beneficial speed-up in EPA-ng when placing a large number of query sequences into the same 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. These variants differ mainly in Stage 2, which uses the votes to determine the subtree to assign each query sequence. The comparison between these variants is in the supplementary materials; here we show the variant that had the best accuracy and speed. We refer to this as 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.

Algorithm BSCAMPP(e)
The input to BSCAMPP(e) is a placement tree T, with edge lengths, and a set Q of query sequences. BSCAMPP(e) has four stages, as we now describe.

- **Stage 1 (Initialization)**:
  - We compute the Hamming distance between every query sequence q in Q and every leaf in the tree T. Every query sequence q votes for the v leaves (ordered by Hamming distance to the query sequence), thus identifying their closest leaf, $closest(q)$.
  - We initialize $\mathcal{T}$ to the emptyset.
- **Stage 2 (Constructing $\mathcal{T}$, the set of subtrees, and initial assignment of query sequences)**:
  - While there are some query sequences not yet assigned to any subtree, DO:
    - Choose the most voted leaf x in the tree to use as a seed sequence, build a subtree using a breadth-first search based on branch length, until B leaves are in the subtree; call this tree t_x. We refer to x as the seed for t_x. Add subtree t_x to the collection $\mathcal{T}$.
    - For every query sequence q, if the leaf $closest(q)$ is in t_x, then assign q to t_x and remove the votes from query sequence q
- **Stage 3 (Allow reassignments of query sequences)**:
  - For every query sequence q and for every tree $t_x \in \mathcal{T}$,
    - compute the path distance (taking branch lengths into account) from $closest(q)$ to x. Let y_q be the seed sequence that has the minimum distance to q. Reassign q to t_y if $y \neq closest(q)$.
- **Stage 4 (Perform Placement)**:
  - For every subtree $t_x \in \mathcal{T}$,
    - Run EPA-ng on t_x and its assigned query sequences, to produce t'_x
    - Use technique from SCAMPP to add the query sequences in t'_x to T (the backbone tree). This technique uses branch lengths in the placement and full backbone tree.
  - Return a file containing all placements, with their requisite confidence score, distal length, placement edge number, etc.

Implementation details On top of EPA-ng's parallel implementation, BSCAMPP(e) is written in python with certain parts written using openMP in C++. Since the

Hamming distance computation is a computationally intensive portion of the BSCAMPP(e) framework (requiring $\mathcal{O}(rql)$ for q queries of length l compared to r reference leaves), a parallel implementation using openMP allows for easier batch processing of queries.

4 Performance Study Design

4.1 Overview

We evaluate placement methods for use with short sequences, the application similar to what would be encountered in placing short reads into phylogenetic trees or taxonomies. Since many reads are characterized in each run, scalability to large numbers of reads is a relevant question. We use both simulated and biological datasets in this study, dividing the datasets into training data and testing 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 [20, 2] (see Section 4.7).

We ran the following experiments for this study.

- Experiment 0 provides a study of scalability for EPA-ng and pplacer as a function of the number of query sequences, and motivates the development of EPA-ng, and why SCAMPP is insufficient.
- Experiment 1 is the design of BSCAMPP(e) (i.e., BSCAMPP used with EPA-ng) on training data.
- Experiment 2 compares BSCAMPP(e) to EPA-ng, SCAMPP(p), SCAMPP(e), and APPLES-2 on testing data.
- Experiment 3 evaluates BSCAMPP(e) with a large backbone tree (180,000 leaves) and varying the number of query sequences.

See Supplementary Materials for additional details.

4.2 Methods

We refer to BSCAMPP used with EPA-ng by BSCAMPP(e), and when used with pplacer by BSCAMPP(p). Similarly, we refer to SCAMPP (v2.1.1) used with EPA-ng by SCAMPP(e) and when used with pplacer by SCAMPP(p). In our analyses, we use pplacer used with taxtastic [10] for placement, since this improves accuracy and scalability according to [8, 1]. We also examine APPLES-2 (v2.0.11) and EPA-ng (v0.3.8).

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 (v1.0.3) [12] for EPA-ng, SCAMPP(e), and BSCAMPP(e), and FastTree-2 (v2.1.11) [22] under Minimum Evolution for APPLES-2.

4.3 Computational resources

All methods are given four hours to run with 64 GB of memory and 16 cores. For SCAMPP(e) and SCAMPP(p), 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(e) and SCAMPP(p) were then run 40 times for each subset containing 250 query sequences.

4.4 Datasets

All datasets are from prior studies and are freely available in public repositories.

4.4.1 RNASim

We use samples from the RNASim dataset [19], which is a simulated dataset of 1,000,000 sequences that evolve down a model tree under a biophysical model to preserve RNA secondary structure. Subsets of the million-sequence simulated dataset were used in prior studies evaluating phylogenetic placement methods [2, 1, 28], and provide a substantial challenge due to the dataset size. For this study, we split this dataset into two subsets by taking the model tree and splitting it into two clades, with one having approximately 600,000 sequences and the other having approximately 400,000 sequences. This defines two sets of sequences, with the smaller one used for training (Experiments 0 and 1) and the larger one for testing (Experiments 2 and 3). We place into a maximum likelihood tree on the true alignment (estimated using FastTree2), using the true alignment.

4.4.2 nt78

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. We place into a maximum likelihood tree on the true alignment (estimated using FastTree2), using the true alignment. This dataset is used in the training phase (Experiment 1).

4.4.3 16S.B.ALL

For biological dataset analysis, we use 16S.B.ALL, a large biological dataset with 27,643 sequences and a reference alignment based on structure from CRW [6]. 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 and the remaining were made backbone sequences. A maximum likelihood tree for this dataset was computed on this reference alignment using RAxML [25] and is available at [18]; this serves as the "reference tree" for this dataset and was reported in [14]. When computing delta error, we used the 75% bootstrap tree (estimated using RAxML) from [14] (available at [18]) as the reference topology.

4.5 Fragmentary sequence generation

We generated fragments 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.

4.6 Additional details about experiments

In Experiment 1, 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.

In Experiment 0, we used a subset of the sequences in Experiment 1. Additionally, the 1,000-sequence backbone tree for Experiment 0 was built subsampling 1000 sequences from the 50,000-leaf tree. For the query sets, we randomly selected subsets of 10, 100, and 1000 queries from the 10,000 already chosen from Experiment 1.

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.

4.7 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 [28, 1], and is the same as the node distance when the backbone tree is the true tree. The methods are also evaluated with respect to runtime and peak memory usage.

5 Results

5.1 Experiment 0: Motivating BSCAMPP+EPA-ng

The goal of this experiment was to understand the impact of varying the number of query sequences placed with EPA-ng and pplacer.

We show results on a 1000-leaf backbone tree for various numbers of query sequences (Figure 2). The runtime results are of note: pplacer's runtime increases linearly for increasing numbers of query sequences, whereas there is a sub-linear trend for EPA-ng. In fact, EPA-ng takes less time to place 100 and 1000 sequences than it does to place 10 sequences. Furthermore, neither the peak memory usage nor the delta error notably increase when placing more queries.

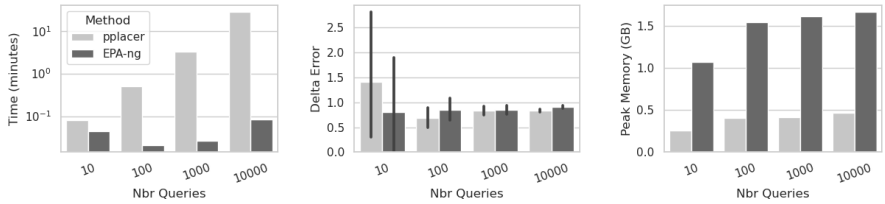


Figure 2: Experiment 0: Maximum-likelihood method comparison on training datasets on a 1000-leaf RNASim backbone tree. Runtime (left), Mean delta error (center), and peak memory usage in GB (right) are shown for placement of 10, 100, 1000, and 10,000 query sequences on an RNASim backbone tree with 1,000 leaves. We show results for pplacer and EPA-ng (both methods that can be used within the SCAMPP and BSCAMPP frameworks).

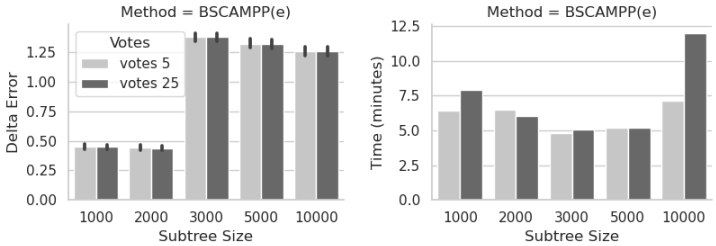
Furthermore, previous work has shown that EPA-ng can fail to run on larger backbone tree sizes once the subtree size reaches over 50,000 sequences. For example, [28] showed that EPA-ng is unable to place a single sequence in the tree on the nt78 dataset (with approximately 78,000 leaves) due to out-of-memory issues, whereas EPA-ng-SCAMPP (here referred to as SCAMPP(e)) is able to return a placement with low error. While the SCAMPP framework can place a sequence onto a large backbone tree, it was designed for placement of a single sequence, and will build a separate subtree for every query sequence. This will result in running EPA-ng for only one query sequence at a time, and thus the SCAMPP(e) cannot take advantage of EPA-ng's sub-linear runtime scaling with respect to the number of query sequences.

This motivates the potential time saving utility of using EPA-ng to place multiple sequences in a single run within BSCAMPP.

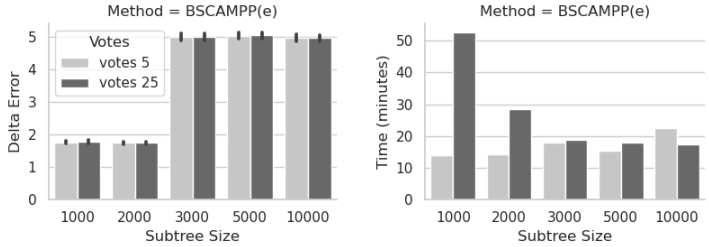
5.2 Experiment 1: BSCAMPP(e) design

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. We varied the subtree parameter setting from 1000, 2000, 3000, 5000 and 10,000 leaves. For each subtree size, we ran BSCAMPP(e) with 5 and 25 votes per query sequence.



(a) RNASim - 50,000 sequences in the backbone tree and 10,000 query sequences



(b) nt78 - 68,132 sequences in the backbone tree and 10,000 query sequences

Figure 3: Experiment 1 - Subtree Size and Vote settings for BSCAMPP(e) on training datasets. Mean Delta Error plus standard error (left) and total runtime (right) for placement of all 10,000 fragmentary query sequences. We show placement time and error for BSCAMPP(e) varying parameter v (the number of votes per query) and the parameter B (the size of the subtree). The fragmentary sequences are a mean of 10% of the original ungapped sequence length with a standard deviation of 10 nucleotides.

We note a few trends with respect to the optimal subtree size. Most notably, Figure 3 shows that when the subtree size increases over 2,000 leaves, fragmentary sequence placement with BSCAMPP(e) has over twice the error than it does for 1,000- and 2,000-leaf subtrees. The number of votes does not show a large change in delta error for either dataset.

The runtime trends are dataset dependent: on the RNASim dataset, the placements with all of the subtree sizes and votes were able to complete in under 15 minutes, whereas the nt78 dataset required over 50 minutes for some settings. On the nt78 dataset, where the runtime was more affected by parameter settings, BSCAMPP(e) is the fastest at 14.1 minutes for 5 votes with a subtree size of 1000. For both subtree sizes where the delta error is low, using 5 votes

Table 1: Training Data Results for BSCAMPP(e) Parameter Settings on RNASim (50,000 sequences in the backbone tree with 10,000 fragmentary query sequences).

	RNASim			
	Delta Error		Runtime (minutes)	
Subtree Size	5 Votes	25 Votes	5 Votes	25 Votes
1000	0.450	0.447	6.4	7.9
2000	0.442	0.437	6.5	6.0
3000	1.376	1.377	4.8	5.0
5000	1.321	1.317	5.2	5.2
10000	1.257	1.258	7.2	12.0

Table 2: Training Data Results for BSCAMPP(e) Parameter Settings on nt78 (68,132 sequences in the backbone tree with 10,000 fragmentary query sequences).

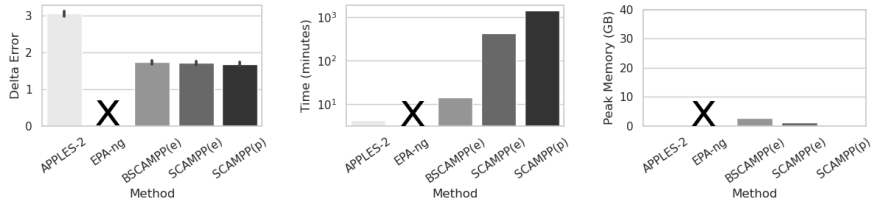
	nt78			
	Delta Error		Runtime (minutes)	
Subtree Size	5 Votes	25 Votes	5 Votes	25 Votes
1000	1.764	1.772	14.1	52.7
2000	1.744	1.734	14.4	28.4
3000	4.998	4.991	17.9	18.7
5000	5.028	5.038	15.5	18.0
10000	4.957	4.950	22.6	17.4

for parameter v results in faster or similar runtimes.Based on these trends a subtree size of 2,000 and 5 votes are chosen as defaults for parameters B and v, respectively.

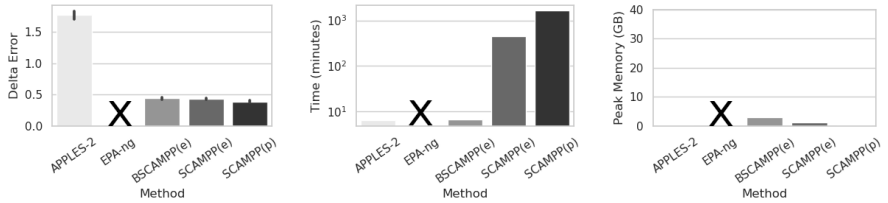
Using the chosen parameters, we show BSCAMPP(e) compared to other methods on the training datasets in Figure 4. Both training datasets show that the same trends. BSCAMPP(e) is nearly as accurate as SCAMPP(e) and SCAMPP(p) while taking a fraction of the runtime. SCAMPP(p) is the most accurate method, but takes 23 hours and 27 hours to complete on RNASim and nt78, respectively. APPLES-2 is the fastest method across both datasets, but shows the highest error. EPA-ng is unable to run, with the failure due to out-of-memory errors. However, BSCAMPP(e) is able to place all 10,000 sequences accurately and fast, while still using only 3 GB of memory.

5.3 Experiment 2: Comparing BSCAMPP(e) to other methods

In this experiment, we compare our BSCAMPP(e) framework with the selected parameters from Experiment 1 to SCAMPP [28], APPLES-2 [1], and EPA-ng [3] when placing fragmentary sequences into a backbone tree.



(a) nt78 (68,132 sequences in the backbone tree, placing 10,000 query sequences)

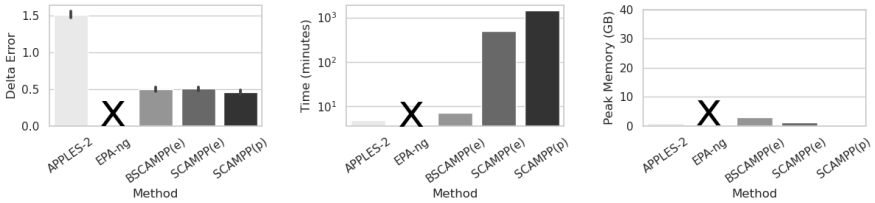


(b) RNASim (50,000 sequences in the backbone tree, placing 10,000 query sequences)

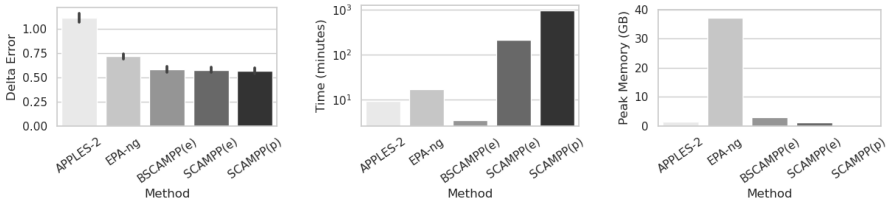
Figure 4: Experiment 1: Method Comparison on Training Datasets. Top: Mean delta error (left), runtime (center), and peak memory usage in GB (right) for placement of all 10,000 query sequences on the nt78 backbone trees with 68,132 leaves. Bottom: Mean delta error (left), runtime (center), and peak memory usage in GB (right) for placement of all 10,000 query sequences on the RNASim backbone trees with 50,000 leaves. We show placement time for BSCAMPP(e) for 5 votes using a subtree of 2000 leaves. SCAMPP(e) and SCAMPP(p) similarly use a subtree size of 2000 leaves. The results from APPLES-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.

We show results for RNASim with 50,000 sequences in the backbone tree and 16S.B.ALL with 20,000 sequences in the backbone tree in Figure 5. EPA-ng failed to run on the RNASim dataset given 64 GB of memory, reporting an out-of-memory issue; however, EPA-ng was able to place all sequences for the 16S.B.ALL dataset, within the memory limitations. An examination of the memory usage shows that EPA-ng had a much higher memory requirement (37 GB) on the 16S.B.ALL datasets, compared to the other methods (none of which exceeded 3 GB). Thus, EPA-ng demonstrates challenges on these two datasets due to memory requirements. All other methods succeeded in completing on these datasets within the available time.

The methods differed substantially in terms of runtime to place all the query sequences (10K for RNASim and 5093 for 16S.B.ALL). APPLES-2 never used more than 9 minutes on either dataset, BSCAMPP(e) never used more than 8 minutes on either dataset, EPA-ng used 17 minutes on 16S.B.ALL. In contrast, SCAMPP(e) used 214 and 466 minutes on 16S.B.ALL and RNASim, respec-



(a) RNASim (50,000 sequences in the backbone tree, placing 10,000 query sequences)



(b) 16S.B.ALL (20,000 sequences in the backbone tree, placing 5,093 query sequences)

Figure 5: Experiment 2: Method Comparison on Testing Datasets - Top: Mean delta error (left), runtime (center), and peak memory usage in GB (right) for placement of all 10,000 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 query sequences on the 16S.B.ALL backbone trees with 20,000 leaves. We show placement time for BSCAMPP(e) for 5 votes with a subtree size of 2000. The results from APPLES-2 and EPA-ng are also included (an X indicates that EPA-ng was unable to run due to out-of-memory issues). SCAMPP(e) and SCAMPP(p) (using a taxtastic reference package) are both run for a subtree size of 2000.

tively, while SCAMPP(p) used 952 and 1421 minutes on these two datasets. Thus, APPLES-2, BSCAMPP(e), and EPA-ng are very fast, completing in just a few minutes on these datasets, and SCAMPP(e) and SCAMPP(p) are much slower.

The methods also differed in terms of placement accuracy. Specifically, APPLES-2 had higher error than the likelihood-based methods for both datasets (delta error above 1.0 in both cases), and EPA-ng had higher delta error than the other likelihood-based methods on the one dataset on which it ran (16S.B.ALL). The comparison between BSCAMPP(e), BSCAMPP(p), and SCAMPP(p) shows very close accuracy on both datasets (all close to 0.5), with perhaps a very slight edge to SCAMPP(p) on the RNASim dataset.

5.4 Experiment 3: Method comparison on ultra-large trees

Our final experiment is designed to verify that BSCAMPP(e) scales to both large query sets and ultra-large backbone tree sizes. Additionally, we check

that BSCAMPP(e) is able to achieve sub-linear runtime scaling with respect to the number of query sequences. We use RNASim for this study, with a backbone tree having 180,000 leaves, and placing from 20 to 20,000 query sequences into the tree. We limited this study to BSCAMPP(e) and those methods that are able to place into the 50,000-leaf tree within 1 hour; this limited the study to only APPLES-2 and BSCAMPP(e). These analyses were run on the UIUC Campus Cluster, which is heterogeneous (i.e., some machines are older and hence slower than others). While both methods are given 16 cores and 64 GB, any given analysis may have access to very different computational resources.

Results shown in Figure 6 show that BSCAMPP(e) is able to place 200,000 sequences in just 25 minutes, whereas 20 sequences requires 1 minute. The runtime for BSCAMPP(e) increases by a factor of 6.7, 2.5, and 1.5 each time the number of queries increases tenfold. Similarly, APPLES-2 shows a sub-linear curve; the runtime increases by a factor of 3.6, 3.8, and then 6.9 each time the number of queries increases tenfold. APPLES-2 is faster for 20, 200, and 2,000 query sequences, but BSCAMPP(e) is faster at 20,000 query sequences. Moreover, both methods are able to complete in under 1 hour, even when placing 20,000 query sequences into this large tree with 180,000 sequences. Finally, both methods use under 3 GB, well under the computational limit of 64 GB.

Results in Figure 6 also allow us to compare methods for accuracy, and see how changing the number of query sequences affects this. We see that BSCAMPP(e) delta error is not impacted by the number of query sequences, and when at least 200 sequences are considered the delta error for APPLES-2 is also not impacted; this makes sense. However, BSCAMPP(e) shows much lower delta error than APPLES-2 (consistent with trends in previous experiments).

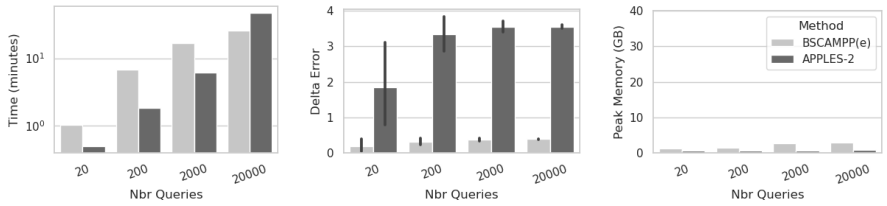


Figure 6: Experiment 3: Comparison of BSCAMPP(e) and APPLES2 given ultra-large backbone tree (180,000 sequences). We show total runtime (left) for placement of all query sequences, mean delta error (center), and peak memory usage (right). We run each method for 20, 200, 2,000, and 20,000 query sequences. Only APPLES-2 and BSCAMPP(e) are shown, since they were the only methods able to complete in under four hours for 20,000 queries.

6 Discussion

We begin with an overall assessment of the methods in terms of their computational performance (memory usage and runtime) as well as ability to place

Table 3: Runtime results across all datasets and backbone tree sizes for all methods. An "X" indicates that the analysis failed due to out-of-memory issues, and a "-" indicates the analysis was not attempted due to the expectation of excessive runtime (exceeding 500 minutes) or excessive memory requirements (exceeding 64 GB).

Method	Total Runtime (for all queries in minutes)				
	Training		Testing		
Dataset:	RNASim	nt78	RNASim		16S.B.ALL
Backbone Size:	50,000	68,132	50,000	180,000	20,000
Nbr Queries:	10,000	10,000	10,000	20,000	5,093
BSCAMPP(e)	6.5	14.4	7.2	25.5	3.5
SCAMPP(e)	457.4	429.2	466.0	-	214.6
SCAMPP(p)	1635.7	1426.4	1421.3	-	952.0
EPA-ng	X	X	X	-	17.4
APPLES-2	6.2	4.1	4.8	47.1	9.3

many query sequences into a large tree. As seen in Table 3, EPA-ng is the least capable, as it had out-of-memory issues on all datasets we attempted except for 16S.B.ALL. APPLES-2 and BSCAMPP(e) were the two fastest, with APPLES-2 never using more than 47.1 minutes and BSCAMPP(e) never using more than 25.5 minutes. We also see that although SCAMPP(e) and SCAMPP(p) were able to run on all the datasets, they were by far the most computationally intensive of the tested methods, using at least 60 times as much time as BSCAMPP(e), and up to 272 times as much time as BSCAMPP(p).

It is worth noting that our prior study evaluating SCAMPP [28] showed SCAMPP(p) being faster than APPLES-2 when placing a smaller number of query sequences. Thus, this study shows that the new version of APPLES-2 that we used in this study is significantly faster than the previous version we used in [28], and even so that we are able to be competitive in terms of runtime when the number of query sequences is large enough.

We also note that BSCAMPP(e) is able to run on datasets where EPA-ng fails due to out-of-memory issues. Indeed, the only dataset in our study where EPA-ng was able to complete was the 16S.B.ALL dataset, and on this dataset, EPA-ng has higher error than BSCAMPP(e). The memory usage was never that high for any method that succeeded in running (the most was EPA-ng, at 37 GB), but as no method had access to more than 64 GB, it is possible that EPA-ng might have succeeded in running if it had access to a larger amount of memory.

Given these observations about scalability and computational performance, we now discuss accuracy. The first observation of this study is that BSCAMPP(e) (i.e., BATCH-SCAMPP used with EPA-ng) essentially matches the accuracy of the other likelihood-based placement methods (pplacer and EPA-ng, and their SCAMPP-boosted versions); furthermore, BSCAMPP(e) is much faster than the other likelihood-based placement methods when run on datasets with many

query sequences. Our experiments consistently showed that all likelihood-based placement methods had very good accuracy with relatively minor differences between them (although EPA-ng was less accurate than the other likelihood-based methods) making computational performance the main distinguishing feature. Hence, the result that BSCAMPP(e) is much faster than the other likelihood-based methods is noteworthy. The second observation is that BSCAMPP(e) improves on the accuracy of APPLES-2, and even improves on speed when the number of query sequences is large enough (at smaller numbers of query sequences, APPLES-2 is faster).

To understand why we see these trends, we consider what we learned about EPA-ng scalability on large backbone trees, both in terms of computational performance but also accuracy. Prior studies have suggested limits for EPA-ng to relatively small backbone trees due to computational reasons [28, 1, 2], but our Experiment 1 showed that EPA-ng had a jump in placement delta error as we increased the subtree size for the nt78 and RNASim datasets. Thus, our study suggests potentially that EPA-ng may have some numeric issues when placing into very large trees that result in increased placement error, a trend that has been previously observed for pplacer [28]. Thus, we would hypothesize that when placing into very large trees, even if EPA-ng manages to finish it may have reduced accuracy due to numeric issues, and that BSCAMPP(e) can have higher accuracy than EPA-ng since it restricts the placement subtrees for EPA-ng to be relatively small (at most 2000 sequences).

From a computational viewpoint, the main 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. Thus, by limiting the placement tree size to at most 2000 sequences, not only does BSCAMPP(e) succeed in maintaining good accuracy, but it also ensures good scalability to very large backbone trees.

We posit that our study suggests that BSCAMPP(e) may be able to scale to much larger numbers of query sequences while maintaining the sublinear runtime. The basis for this optimism is that EPA-ng has been shown to place millions of queries [3] into smaller trees of a few hundred sequences, yet we have limited this study to larger backbone trees with at most 20,000 query sequences. The other point is that Experiment 3 suggests that the full efficiency of BSCAMPP(e) has not yet been reached at 20,000 query sequences, and that it may continue to be able to place even larger numbers of query sequences with runtimes that grow sublinearly with that number even for backbone trees of this size.

BSCAMPP(e) allows for multiple queries to be placed into a subtree, using a new process for selecting subtrees and processing query sequences. We show in Experiment 3 that BSCAMPP(e) is able to achieve a sub-linear runtime scaling when increasing the number of queries; a similar sub-linear scaling to that of EPA-ng as shown in Experiment 0.

7 Conclusions

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, by design SCAMPP did not ensure that the subproblems resulted in many query sequences being placed into the same subtree, which meant that it could not take advantage of EPA-ng's ability to place many query sequences with a runtime that is sublinear in the number of query sequences. By redesigning the divide-and-conquer strategy, we were able to ensure that many query sequences went to the same subproblems, and thus were able to take advantage of this feature of EPA-ng. Our redesign, which we call Batch-SCAMPP (or BSCAMPP for short) matches the accuracy of the SCAMPP and scalability to large backbone trees, and when used with EPA-ng (i.e., BSCAMPP(e)) achieves sublinear runtime in the number of query sequences. Moreover, BSCAMPP is extremely fast, nearly as fast as APPLES-2 for placing a moderate number of query sequences into a large backbone alignment, and slightly faster when placing large numbers of query sequences into very large backbone trees. Thus, BSCAMPP is the first likelihood-based phylogenetic placement method that is both highly accurate (due to its use of likelihood) and scalable in both dimensions – the number of query sequences and the size of the backbone tree (due to its use of divide-and-conquer and the reliance on EPA-ng for placement of many query sequences into small subtrees).

This study leaves several directions for future research. 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 [11]. For example, the divide-and-conquer tree estimation pipeline GTM (Guide Tree Merger) [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 and abundance profiling that use phylogenetic placement, such as TIPP [21] and its improved version TIPP2 [23], which have used pplacer [16]. As shown in [23], accuracy improves as the backbone trees increase in size, suggesting that the use of BSCAMPP(e) on large trees might lead to increased accuracy and reduced runtime compared to the current use of pplacer and smaller backbone trees.

8 Data availability

All datasets used in this study are from prior publications.

9 Acknowledgments

The authors acknowledge the financial support of the Department of Computer Science. EW was supported by a Siebel Scholars scholarship, a SURGE Fellowship, and a Wing Kai Cheng Fellowship.

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

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

[4] 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.

[5] Matthias Blanke and Burkhard Morgenstern. App-SpaM: phylogenetic placement of short reads without sequence alignment. *Bioinformatics Advances*, 1(1), 10 2021. vbab027. http://arxiv.org/abs/https://academic.oup.com/bioinformaticsadvances/article-pdf/1/1/vbab027/41191909/vbab027.pdf `arXiv:https://academic.oup.com/bioinformaticsadvances/article-pdf/1/1/vbab027/41191909/vbab027.pdf`, https://doi.org/10.1093/bioadv/vbab027 `doi:10.1093/bioadv/vbab027`.

[6] 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.

[7] 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.

[8] Gillian Chu and Tandy Warnow. SCAMPP+FastTree: improving scalability for likelihood-based phylogenetic placement. *Bioinformatics Advances*, 3(1), 01 2023. vbad008. http://arxiv.org/abs/https://academic.oup.com/bioinformaticsadvances/article-pdf/3/1/vbad008/49228138/vbad008_supplementary_data.pdf `arXiv:https://academic.oup.com/bioinformaticsadvances/article-pdf/3/1/vbad008/49228138/vbad008\_supplementary\_data.pdf`, https://doi.org/10.1093/bioadv/vbad008 `doi:10.1093/bioadv/vbad008`.

[9] 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.

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

[11] 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.

[12] 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.

[13] Benjamin Linard, Krister Swenson, and Fabio Pardi. Rapid alignment-free phylogenetic identification of metagenomic sequences. *Bioinformatics*, 35(18):3303–3312, 01 2019. http://arxiv.org/abs/https://academic.oup.com/bioinformatics/article-pdf/35/18/3303/30024623/btz068.pdf `arXiv:https://academic.oup.com/bioinformatics/article-pdf/35/18/3303/30024623/btz068.pdf`, https://doi.org/10.1093/bioinformatics/btz068 `doi:10.1093/bioinformatics/btz068`.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

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

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

[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. http://arxiv.org/abs/https://academic.oup.com/bioinformatics/article-pdf/14/2/157/9731868/140157.pdf `arXiv:https://academic.oup.com/bioinformatics/article-pdf/14/2/157/9731868/140157.pdf`, https://doi.org/10.1093/bioinformatics/14.2.157 `doi:10.1093/bioinformatics/14.2.157`.

[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.

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

Eleanor Wedell, Chengze Shen, Tandy Warnow

Contents

S1 Delta Error Definition **2**

S2 Study with EPA-ng **3**

S3 BSCAMPP Design **5**

S4 Datasets and Commands **9**

S4.1 Dataset Preparation 9

S4.2 Method Commands 10

List of Figures

S1 EPA-ng Study 4

S2 BSCAMPP Design: Comparing BSCAMPP implementations on RNASim backbone trees . . 7

S3 BSCAMPP Design: Comparing BSCAMPP implementations on nt78 backbone trees 8

S1 Delta Error Definition

The delta error computed for this study is always measured for a single query, and the mean for all queries in a query set is reported. We begin with some necessary notation.

Notation

- T**:** the backbone tree used as input for the phylogenetic placement method with leafset S'
- q**:** the query sequence to be added to T
- S**:** the set $S' \cup \{q\}$
- P**:** the tree on the leaf set S that results from adding q to T using a phylogenetic placement method.
- T^***:** the true tree on S
- $T^*|_{S'}$**:** the true tree restricted to the leafset S', i.e., the result of removing q from T^*

Note that P and T^* have the same set S of leaves, and that $T^*|_{S'}$ and T have the same set S' of leaves. Thus we can define the *missing branches* of P with respect to T^* and the *missing branches* of $T^*|_{S'}$ with respect to T, as we now define.

The missing branches and bipartitions Each edge in a tree defines a bipartition on the leaf-set, produced by deleting the edge but not its endpoints. We denote the bipartitions of a tree t are denoted by $B(t)$. The missing branches in an estimated tree $\hat{t}$ with respect to a true tree t on the same leaf set is just $B(t) \setminus B(\hat{t})$.

Formula for delta error Then the delta error is defined as follows:

$$\Delta e(P) = |B(T^*) \backslash B(P)| - |B(T^*|_{S'}) \backslash B(T)|, \tag{1}$$

In other words, the delta error is the *increase in the number of missing branches produced by adding the new taxon* q.

Comments By restricting the edges that count toward delta error to those that are caused only by the phylogenetic placement and not by those that were estimated incorrectly in the backbone tree T, performance of the phylogenetic placement can be measured directly without the bias of the estimated tree error.

The delta error is an extension of node distance, the number of nodes separating each query sequence's true and target placement, originally used in [3, 6, 7] in that it removes bias from the estimated tree error, when the true tree topology is know. For all biological datasets where the true tree is unknown, by using the estimated tree as the true tree and placing into the estimated tree, the delta error is the same as the simple node distance [9].

S2 Study with 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 ranging from 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 delta error on various backbone tree sizes, placement strategies (batch vs. one-by-one), and for either full-length sequences or fragmentary sequences (of lengths that are 10%, 25%, 50% of the full-length sequences) in Figure S1.

Not that placement accuracy is highest when placing full-length sequences, and that placing very short sequences (the standard situation in metagenomics) has the highest error rates. For all query sequence lengths, accuracy improves for one-by-one placement as the backbone size increases, indicating the beneficial impact of taxon sampling. Interestingly, this is not true for batch placement.

For batch placement, 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.

Since our interest is in placing short sequences (corresponding to read placement), we focus now on the results shown for queries of 10%-length. For these very short query sequences, 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.

In sum, this experiment shows that EPA-ng performs very differently between the batch mode (the case where it is able to obtain a runtime advantage) and the 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.

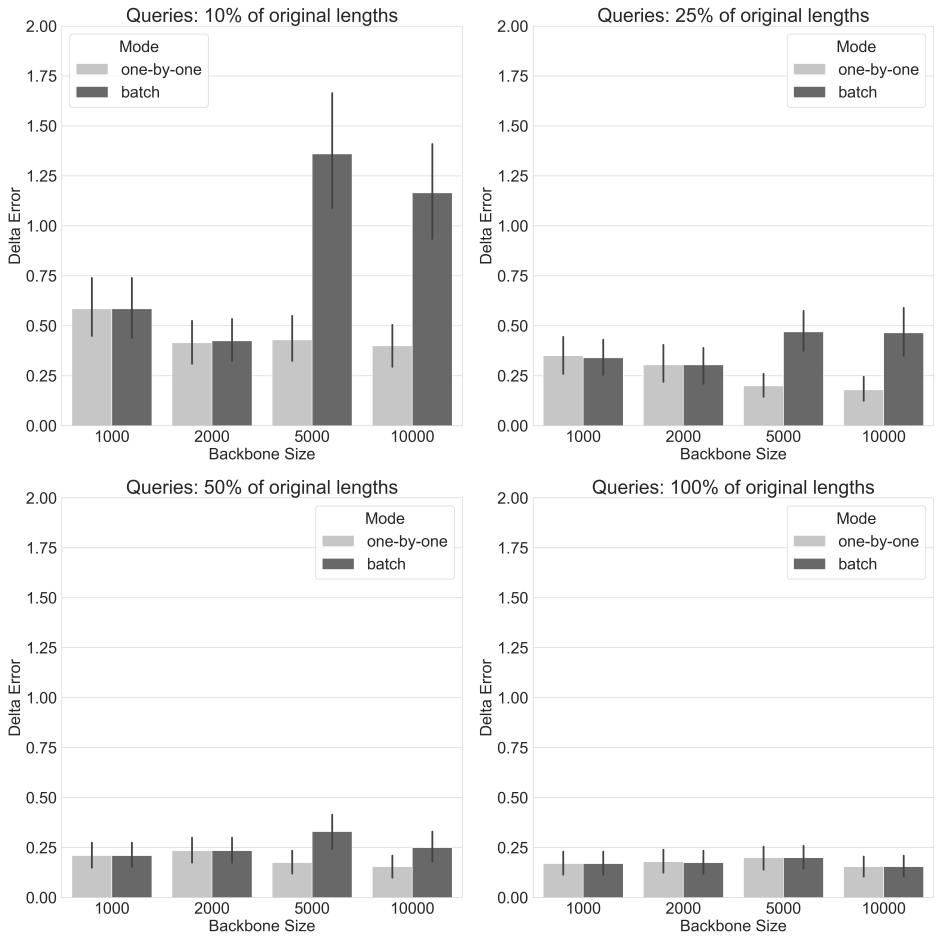


Figure S1: Mean Delta Error of EPA-ng when placing 200 queries into a backbone tree with 1000, 2000, 5000, and 10000 leaves of the training RNASim dataset. The queries and backbone sequences are selected from the same clade. From top to bottom, left to right: query sequences (for placement) are made into fragments of 10%, 25%, 50%, and 100% of their original lengths (100% means full-length). We show two strategies for using EPA-ng for placement: "batch" means that all queries are placed at once, while "one-by-one" means that each query sequence is placed individually.

S3 BSCAMPP Design

The SCAMPP framework scales EPA-ng to larger backbone trees, but has a linearly increasing runtime when increasing the number of query sequences. The current approach of running EPA-ng for a single query sequence on a backbone tree does not take advantage of EPA-ng's sub-linear runtime scalability with respect to the number of query sequences.

In order to enable SCAMPP for batch query sequence processing, four new approaches are explored to enable the phylogenetic placement method to place multiple query sequences into a subtree. These four new approaches are implemented and tested specifically with EPA-ng, since this is optimized already for fast parallel placement of many query sequences.

There are four general strategies taken and tested in this study described here as EPA-ng-BSCAMPP-1, EPA-ng-BSCAMPP-2 and EPA-ng-BSCAMPP-3, and EPA-ng-BSCAMPP-4. All four approaches use a backbone tree, leaf voting strategy to inform the subtree selection process. The subtree size, B, is parameterized and the number of votes per query sequence, v, is parameterized and determined by the v leaves in the tree with the smallest Hamming distance to the query sequence. Once all leaves in the backbone tree have been voted on, the leaf, l, with the most votes is selected, and a subtree is built by greedily adding leaves with the shortest distance to l, until B leaves are in the subtree. It is at this point that the four methods differ, but in all approaches subtrees are selected, query sequences are assigned, and finally, once all queries are included in a subtree, EPA-ng is run for each subtree, query set pairing.

The four approaches are described below in more depth with their differences in bold.

EPA-ng-BSCAMPP-1.

- The input is a set of query sequences, backbone tree for placement, and a joint backbone and query multiple sequence alignment.
- The query sequences vote on v leaves based on the v closest leaves by Hamming distance in the backbone tree.
- Then while there are query sequences not assigned to a subtree, choose the most voted leaf in the tree, build a subtree using a breadth first search collecting nearby leaves from the full tree until B leaves are in the subtree. Assign queries to the subtree if **they have any votes within that subtree.** And remove the votes for assigned query sequences.
- Run EPA-ng for the subtrees and their assigned query sequences.
- Determine the each query sequence's placement in the full backbone tree, and return a file containing all placements, with their requisite confidence score, distal length, placement edge number etc.

EPA-ng-BSCAMPP-2.

- The input is a set of query sequences, backbone tree for placement, and a joint backbone and query multiple sequence alignment.
- Each query sequence votes for v leaves based on the v closest leaves by Hamming distance in the backbone tree.
- Then while there are query sequences not assigned to a subtree, choose the most voted leaf in the tree, build a subtree using a breadth first search collecting nearby 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.** And remove the votes for assigned query sequences.
- Run EPA-ng for the subtrees and their assigned query sequences.
- Determine each query sequence's placement in the full backbone tree, and return a file containing all placements, with their requisite confidence score, distal length, placement edge number etc.

EPA-ng-BSCAMPP-3.

- The input is a set of query sequences, backbone tree for placement, and a joint backbone and query multiple sequence alignment.
- Each query sequence votes for v leaves based on the v closest leaves by Hamming distance in the backbone tree.
- Let set $A = \emptyset$.
- Then while there are query sequences not **in set A**, choose the most voted leaf in the tree, build a subtree using a breadth first search collecting nearby leaves from the full tree until B leaves are in the subtree. **Add** queries to **set A** 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.** And remove the votes for **query sequences in set A**.
- **Assign each query sequence to the subtree with minimum total Hamming distance from the query sequence to all of the leaves in the subtree**.
- Run EPA-ng for the subtrees and their assigned query sequences.
- Determine the each query sequence's placement in the full backbone tree, and return a file containing all placements, with their requisite confidence score, distal length, placement edge number etc.

EPA-ng-BSCAMPP-4.

- The input is a set of query sequences, backbone tree for placement, and a joint backbone and query multiple sequence alignment.
- Each query sequence votes for v leaves based on the v closest leaves by Hamming distance in the backbone tree. **Note the top vote leaf for each query as the single leaf with the smallest hamming distance in the backbone tree.**
- Let set $A = \emptyset$.
- Then while there are query sequences not **in set A**, choose the most voted leaf in the tree. Let this leaf be a seed sequence, and build a subtree using a breadth first search collecting nearby leaves from the full tree until B leaves are in the subtree. **Add** queries to **set A** if **the top vote leaf for that query is in the subtree** And remove the votes for **query sequences in set A**.
- **Assign each query sequence to the subtree with smallest topological distance from the top voted leaf to the subtree's seed sequence**.
- Run EPA-ng for the subtrees and their assigned query sequences.
- Determine the each query sequence's placement in the full backbone tree, and return a file containing all placements, with their requisite confidence score, distal length, placement edge number etc.

The difference between methods occurs in the third and fourth steps where the subtrees are selected and queries are assigned to subtrees. Versions 1 and 2 assign queries while building subtrees, but they do so differently. Version 1 allows a query to be assigned to the first subtree built that contains any of the v voted leaves with low Hamming distance. Version 2 is more selective in the query assignment, in that it will only allow a query to be used in a tree if all of its votes are contained in the tree, or if the best vote with the lowest Hamming distance is used as the leaf to start the greedy search to build the tree.

Version 3 uses the same subtrees as version 2, but it reassigns queries rather than assigning the query to the first subtree for which it may be used. The reassignment metric is as follows: each query is assigned to the subtree which minimizes the sum of the Hamming distance of the query to each leaf in the subtree.

Finally, version 4 deviates from version 3 with two adjustments. It requires the nearest leaf to be in the subtree; it does not look at other votes. It also reassigns queries to subtrees by choosing the subtree minimizing edge length distance from the nearest leaf to the subtree's seed leaf.

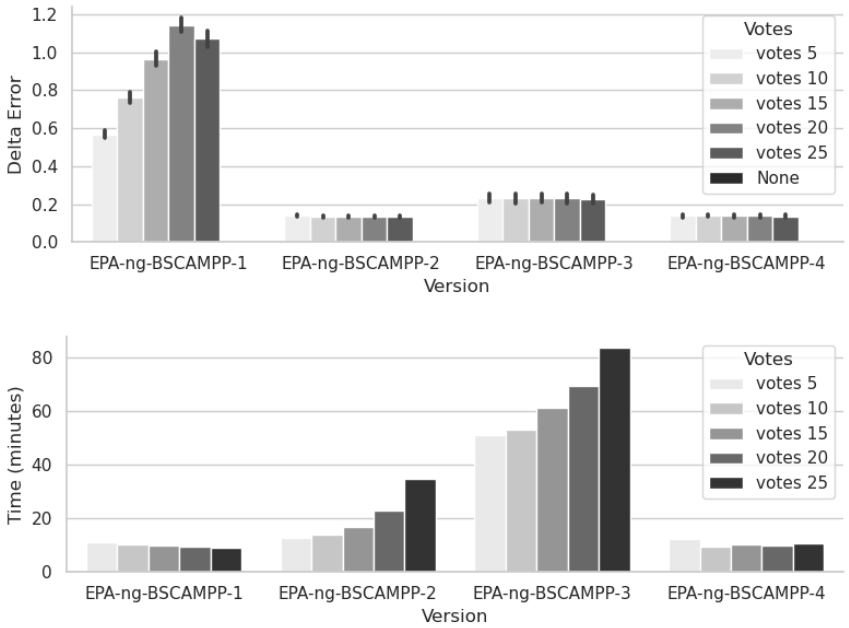


Figure S2: Top: Mean Delta Error plus standard error for placement of full length sequences on the RNASim backbone trees with 50,000 leaves and 10,000 query sequences. We show placement delta error for four versions of EPA-ng-BSCAMPP varying parameter v (the number of votes per query). Bottom: Total runtime for placement of all 10,000 full length query sequences on the RNASim backbone trees with 50,000 leaves. We show placement time for four versions of EPA-ng-BSCAMPP varying parameter v (the number of votes per query).

We test all four versions of EPA-ng-BSCAMPP on two datasets, nt78 and RNAsim. The delta error (Figure S2, top) shows that versions 2 and 4 are the most accurate. In addition, the runtime for version 4 (Figure S2, bottom) shows that EPA-ng-BSCAMPP-4 is able to place all 10,000 query sequences in just under 10 minutes when there are 10 votes, and just over 12 minutes in the worst case when there are 5 votes. EPA-ng-BSCAMPP-4 is faster than EPA-ng-BSCAMPP-2 in all cases.

EPA-ng-BSCAMPP-1 was able to place all 10,000 query sequences in around 10 minutes in all cases, but was the worst performing with respect to delta error, showing 3 to 5 times more error than EPA-ng-BSCAMPP-2 depending on the number of votes. EPA-ng-BSCAMPP-3 showed lower error than EPA-ng-BSCAMPP-1; however, it is not as accurate as EPA-ng-BSCAMPP-2 and EPA-ng-BSCAMPP-4, and takes between 50 and 80 minutes to run, giving EPA-ng-BSCAMPP-4 the advantage.

The accuracy with respect to the number of votes shows different trends based on the version. While the number of votes increases error for EPA-ng-BSCAMPP-1, it only has a small impact on the accuracy of EPA-ng-BSCAMPP-2, EPA-ng-BSCAMPP-3, and EPA-ng-BSCAMPP-4.

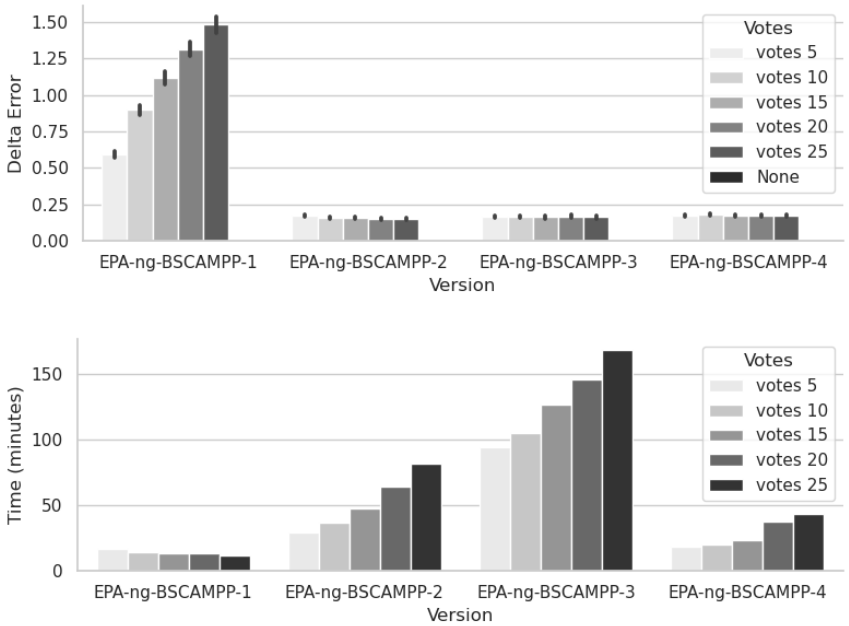


Figure S3: Top row: Mean Delta Error plus standard error for placement of full length sequences on the nt78 backbone trees with 68,132 leaves and 10,000 query sequences. We show placement delta error for four versions of EPA-ng-BSCAMPP varying parameter v (the number of votes per query). Bottom Row: Total runtime for placement of all 10,000 full length query sequences on the nt78 backbone trees with 68,132 leaves. We show placement time for four versions of EPA-ng-BSCAMPP varying parameter v (the number of votes per query).

The results from the nt78 datasets show similar trends with a few exceptions (see Figure S3). One primary difference is that on this dataset EPA-ng-BSCAMPP-3 is now showing similar accuracy to EPA-ng-BSCAMPP-2 and EPA-ng-BSCAMPP-4.

EPA-ng-BSCAMPP-1 is able to complete all placements in under 20 minutes. Of the three more accurate methods, EPA-ng-BSCAMPP-4 is the fastest, placing all 10,000 query sequences in between 18 and 45 minutes, whereas EPA-ng-BSCAMPP-2 completes all placements in between 20 and 80 minutes, with the time increasing with the number of votes. In addition, EPA-ng-BSCAMPP-3 is the slowest version tested, with runtimes between 80 and 160 minutes, again increasing with the number of votes.

EPA-ng-BSCAMPP version 4 is henceforth referred to as BSCAMPP(e), since the results indicate that this is the most efficient implementation with low delta error.

S4 Datasets and Commands

All code is available on GitHub at `https://github.com/ewedell/BSCAMPP_code`.

S4.1 Dataset Preparation

The datasets were built using the following commands, with scripts included on the GitHub site.

For the RNASim dataset, all steps are followed. For the nt78 dataset 10,000 queries were randomly selected and bullets 1 and 2 below were skipped.

In addition the 16S.B.ALL dataset contained duplicate sequences. We removed these from the reference and "true topological" tree and alignments before running any method.

1. To generate replicates, choose query sequences, and remove sites with over 95% gaps:

   ```
   python3 ./build_dataset.py -m ./true.fasta -t ./true.tt -o ./ -x a
   ```

2. To convert from RNA to DNA:

   ```
   sed '/^[^>]/ y/uU/tT/' aln.fa > aln_dna.fa
   ```

3. To separate query and reference alignments:

   ```
   python3 ./build_dataset.py -m aln_dna.fa -o ./ -x b -q query.txt
   ```

4. To estimate the backbone tree before queries are pruned using FastTree v2.1.11 single precision [8]:

   ```
   FastTreeMP -nosupport -gtr -gamma -nt -log ./true.fasttree.log
   < ./aln_dna.fa > ./true.fasttree
   ```

5. For 16S.B.ALL where the published reference topology was used, we re-estimated the branch lengths for pplacer with FastTree v2.1.11 using the following command:

   ```
   FastTreeMP -nosupport -gtr -gamma -nt -mllen -nome -intree reference.tree
   -log true.fasttree.log < aln_dna.fa > true.fasttree
   ```

6. To re-estimate branch lengths for use with EPA-ng using RAxML-ng v1.0.3 [5]:

   ```
   /raxml-ng/bin/raxml-ng --evaluate --msa ref.fa --model GTR+G --tree tree
   --brlen scaled
   ```

7. To build fragmentary sequences:

   ```
   python3 get_frag_aln.py -i query.fa -p 1 -a .10 -s 10 -o a_higher_frag.fa
   -of ua_higher_frag.fa
   ```

8. To re-estimate branch lengths under Minimum Evolution for use with APPLES-2 using FastTree v2.1.11 single precision [8]:

   ```
   FastTreeMP -nosupport -nt -nome -noml -log fasttree.log -intree reference.nwk
   < alignment.fasta > fasttree.nwk
   ```

9. To build a reference package with taxtastic taxit [4] for use with SCAMPP(p) (backbone_pplacer.tree is the tree with query sequences pruned preserving branch lengths from the fasttree-2 estimated tree and true.fasttree.log is the log file created from running bullet 4 or 5 above):

   ```
   taxtastic-env/bin/taxit create -P my.refpkg -l 16S --aln-fasta ref.fa
   --tree-file backbone_pplacer.tree --tree-stats true.fasttree.log
   ```

S4.2 Method Commands

Methods are called using the following commands:

1. BSCAMPP(e) (v1.0.0)

```
python3 EPA-ng-BSCAMPP.py -t backbone_epa.tree -a ref.fa -q a_higher_frag.fa
-b {subtreesize} -i aln_dna.fa.raxml.bestModel -d {dir} -V {votes} -f True
```

2. SCAMPP(e) (v2.1.1)

```
python3 EPA-ng-SCAMPP.py -t backbone_epa.tree -a ref.fa -q a_higher_frag.fa
-b {subtreesize} -i aln_dna.fa.raxml.bestModel -d {dir} -f True
```

3. SCAMPP(p) (v2.1.1)

```
python3 pplacer-SCAMPP-tax.py -t backbone_pplacer.tree -a ref.fa -q
a_higher_frag.fa -b {subtreesize} -i true.fasttree.log -d {dir} -f True
```

4. EPA-ng (v0.3.8)

```
epa-ng --ref-msa ref.fa --tree backbone_epa.tree --query a_higher_frag.fa
--outdir {dir} --model aln_dna.fa.raxml.bestModel --redo -T 16
```

5. APPLES (v2.0.11)

```
python3 run_apples.py -t backbone_app.tree -s ref.fa -q a_higher_frag.fa
-T 16 -o APPLES-2-f.jplace -X -D
```

Scripts with commands used to run each method are also available on GitHub. SCAMPP(p and e) [9], EPA-ng v0.3.8 [2] and APPLES-2 [1] v2.0.11 are used for phylogenetic placement. The data including estimated backbone trees, query lists, ungapped alignments and fragmentary sequences will be uploaded to the Illinois Databank once the paper is accepted for publication.

References

[1] Metin Balaban, Yueyu Jiang, Daniel Roush, Qiyun Zhu, and Siavash Mirarab. APPLES-2. `https://github.com/balabanmetin/apples`, 2021.

[2] Pierre Barbera, Alexey M Kozlov, Lucas Czech, Benoit Morel, Diego Darriba, Tomáš Flouri, and Alexandros Stamatakis. EPA-ng. `https://github.com/Pbdas/epa-ng`, 2020.

[3] Simon A Berger, Denis Krompass, and Alexandros Stamatakis. Performance, accuracy, and web server for evolutionary placement of short sequence reads under maximum likelihood. *Systematic Biology*, 60(3):291–302, 2011.

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

[5] Alexey M Kozlov, Diego Darriba, Tomáš Flouri, Benoit Morel, and Alexandros Stamatakis. RAxML-ng. `https://github.com/amkozlov/raxml-ng`, 2021.

[6] Benjamin Linard, Nikolai Romashchenko, Fabio Pardi, and Eric Rivals. PEWO: a collection of workflows to benchmark phylogenetic placement. *Bioinformatics*, 36(21):5264–5266, 07 2020.

[7] 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.

[8] Morgan N Price, Paramvir S Dehal, and Adam P Arkin. FastTree-2. `http://www.microbesonline.org/fasttree/#Install`, 2010.

[9] 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.