# BSCAMPP: Batch-Scaled Phylogenetic Placement on Large Trees

Eleanor Wedell, Chengze Shen, and Tandy Warnow

***Abstract*—Phylogenetic placement is the problem of placing sequences into a given phylogenetic tree, called a "backbone tree". EPA-ng and pplacer are the two most accurate phylogenetic placement methods, but both can fail to complete when the backbone tree is very large. Our recently designed SCAMPP framework has been shown to scale both pplacer and EPA-ng to larger backbone trees of up to 180,000 sequences by building a small placement subtree for each query sequence and then using the phylogenetic placement method to place that query sequence into that subtree. However, the technique in SCAMPP produces many placement subtrees (potentially a different one for each query sequence), making it computationally expensive when placing many query sequences. Here we present BSCAMPP (Batch-SCAMPP), a new technique that overcomes this barrier by using the query sequences to select a much smaller number of placement subtrees. We show that BSCAMPP used with EPA-ng is much faster than SCAMPP used with EPA-ng, and scales to ultra-large backbone trees. We also show that BSCAMPP used with pplacer is much faster than SCAMPP used with pplacer, and somewhat more accurate but slower than BSCAMPP used with EPA-ng.**

***Index Terms*—Phylogenetic placement, EPA-ng, microbiome analysis, taxonomic identification, abundance profiling, pplacer.**

## I. 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, a taxonomy with leaves labeled by sequences for a single gene [1], [2], [3], or a species tree [4]. Phylogenetic placement has been used to taxonomically characterize shotgun sequencing reads generated for an environmental sample in a metagenomic analysis; the methods in the TIPP family [1], [2], [3] are based on pplacer and achieve high accuracy, but other approaches have also been used, see [5]. Phylogenetic placement into gene trees can also be used to update existing gene trees with one or more new sequences, an application that is relevant to evolutionary biologists studying specific gene families. Thus, phylogenetic placement is a general tool with applications in both incremental tree building and taxon identification and abundance profiling in microbiome analysis [1], [2], [3], [5], [6], [7], [8], [9].

Prior studies [10], [11], [12], [13], [14] have established the high accuracy of phylogenetic placement methods based on maximum likelihood (e.g., EPA-ng [15] and pplacer [16]). However, the runtime of these methods is impacted by the number of query sequences, the size of the backbone tree, and the length of the reference alignment, and each of these can be very large, depending on the application. In particular, for the metagenomics application, the number of sequences placed into the backbone tree can be very large (in the tens or hundreds of thousands) [1], and future analyses might involve millions of reads. Furthermore, many studies have shown improvement in accuracy for abundance profiling, phylogenetic tree estimation, etc., when placing into large backbone trees (e.g., [3]); hence, phylogenetic placement methods that can process large numbers of query sequences and run on large backbone trees are useful tools.

Unfortunately, prior studies [10], [11], [12], [13] have also shown that EPA-ng [15] can fail to complete due to high memory requirements and pplacer [16] can fail to complete due to numerical issues (reporting negative infinity log likelihood scores) when they are used on very large backbone trees. This observation has led to the development of methods, such as APPLES-2 [17], which use distances to place into large trees. There are also methods for phylogenetic placement that are alignment-free, such as RAPPAS [18] and App-SpaM [19]. These methods are potentially faster and more scalable than pplacer and EPA-ng.

While pplacer has shown some accuracy advantages compared to EPA-ng, EPA-ng is much faster [12]. In particular, EPA-ng is optimized for placing a large number of query sequences (see Fig. 2 from [17]) and is capable of placing millions of sequences into phylogenetic trees of up to a few thousand sequences [15]. However, other studies have shown that EPA-ng has a memory requirement that can be large for large backbone trees [11]; hence, the backbone trees used with EPA-ng will be limited to a few thousand sequences unless there is access to a large amount of memory.

Previously we introduced the SCAMPP framework [11] to enable both pplacer and EPA-ng to perform phylogenetic placement into ultra-large backbone trees, and we demonstrated its utility for placing into backbone trees with up to 200,000

Received 17 July 2024; revised 15 April 2025; accepted 15 April 2025. Date of publication 17 April 2025; date of current version 8 August 2025. The work of Eleanor Wedell was supported by a SURGE Fellowship and a Wing Kai Cheng Fellowship. The work of Chengze Shen was supported by NSF under Grant 2006069 and Grant 2316233 (to Tandy Warnow). This work was supported by the Siebel School of Computing and Data Science at the University of Illinois Urbana-Champaign. *(Corresponding author: Tandy Warnow.)*

The authors are with the Siebel School of Computing and Data Science, The University of Illlinois Urbana-Champaign, Urbana, Illinois 61801 USA (e-mail: warnow@illinois.edu).

BSCAMPP is freely available at https://github.com/ewedell/BSCAMPP.

This article has supplementary downloadable material available at https://doi.org/10.1109/TCBBIO.2025.3562281, provided by the authors.

Digital Object Identifier 10.1109/TCBBIO.2025.3562281

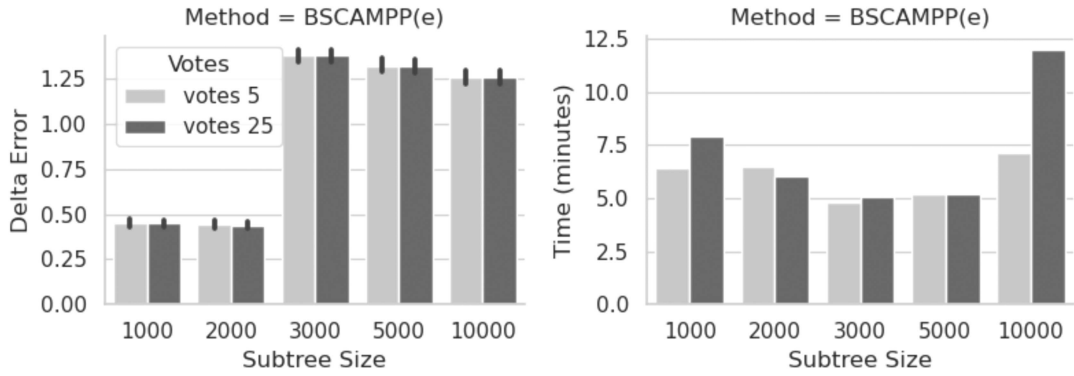



Fig. 1. Experiment 1: Impact of subtree size and number of votes for BSCAMPP(e) on algorithm design dataset using the true query alignments. Mean Delta Error plus standard error (left) and total runtime (right) for placement of all 10,000 fragmentary query sequences on an RNASim backbone tree with 50,000 leaves. 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 (i.e., ∼155 nt) with a standard deviation of 10 nucleotides.

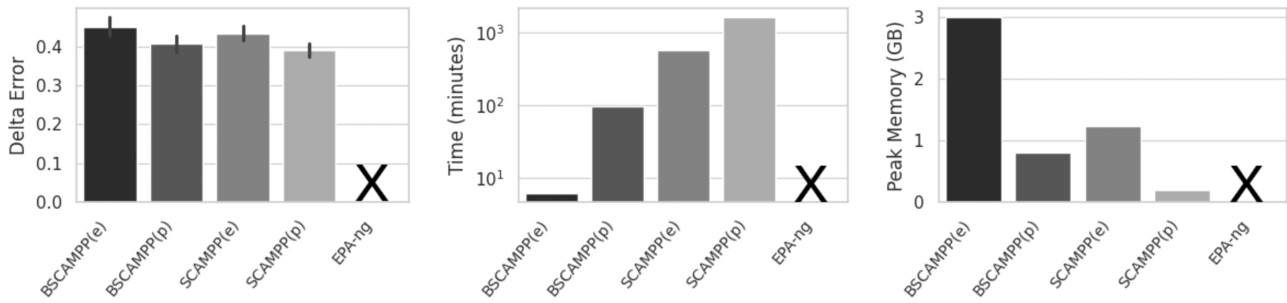



Fig. 2. Experiment 1: Performance on the RNASim 50 k algorithm design dataset, placing 10 k query sequences using true query sequence alignments. Mean delta error (left), runtime (center), and peak memory usage in GB (right) for placement of all 10,000 query sequences on the estimated RNASim backbone tree with 50,000 leaves. The query sequences are a mean of 10% of the original ungapped sequence length (i.e., ∼ 155 nt) with a standard deviation of 10 nucleotides. We show placement time for BSCAMPP(e) and BSCAMPP(p) 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 SCAMPP are also included for runtime, peak memory usage, and delta error. EPA-ng is not included because it was unable to run given 64 GB of memory and 16 cores on these datasets due to out-of-memory issues. SCAMPP produced 8778 subtrees on this dataset but BSCAMPP produced only 101 subtrees.

sequences. SCAMPP places each query sequence independently into the backbone tree. To place a given query sequence, it finds a "nearest leaf" in the tree, extracts a small subtree around that leaf, and then places the query sequence into that subtree using the selected phylogenetic placement method. In the final step, the location of that placement is used to find the corresponding location in the backbone tree. This divide-and-conquer technique enables SCAMPP to scale pplacer and EPA-ng to ultra-large backbone trees (up to 180,000 leaves so far) and achieves high accuracy. However, because each query sequence extracts its own subtree, this technique has the potential to pick as many subtrees as there are query sequences, with the consequence that SCAMPP has a high runtime and is memory-intensive.

The goal of this study is to improve SCAMPP with respect to computational performance. Furthermore, although several factors impact the runtime and memory usage of phylogenetic placement methods, in this study we focus on the impact of the backbone tree size and number of query sequences. To achieve this, we have modified the divide-and-conquer strategy in SCAMPP so that we make a small number of subtrees that suffice for the given set of query sequences. As we will show, this approach, which we call BSCAMPP (for Batch-SCAMPP), has the same benefit for scalability as SCAMPP but dramatically reduces the runtime and memory usage compared to SCAMPP, and incurs only a small reduction in accuracy. We also show that BSCAMPP used with EPA-ng is extremely fast, even on ultra-large backbone trees.

The rest of the paper is organized as follows. We begin in Section II with preliminary experiments evaluating EPA-ng and pplacer accuracy and scalability, motivating the design of BSCAMPP to improve the scalability these methods. We present BSCAMPP in Section III but the experimental study where we design BSCAMPP is described in Section IV. Experiments evaluating BSCAMPP with EPA-ng and pplacer in comparison to other phylogenetic placement methods are presented in Section V. We provide a discussion of results in Section VI, and we finish with conclusions in Section VII. Due to space constraints, some of the results are provided in the Supplementary Materials.

## II. Preliminary Experiment

In this preliminary experiment (described in full in Supplementary Materials Section S1, results in Figs. S1 and S2), we had two objectives: first, to compare EPA-ng and pplacer for accuracy and computational performance when placing a variable number of query sequences into a 1000-leaf tree, and second, to understand the impact of the query length and backbone tree size on EPA-ng. We explore phylogenetic placement error using

the "delta error" (see Section IV-G) and runtime. We found that pplacer is at least as accurate as EPA-ng and has a smaller peak memory usage, and that EPA-ng is much faster than pplacer. We also found that placement error increased for EPA-ng when the backbone tree size increased from 2000 to 5000 leaves, and that the increase in error was large for query sequences that were short (10% of full-length, so $\sim$ 155 nt).

These observations were used in the design and development of BSCAMPP, the subject of the next section.

## III. BSCAMPP

We designed BSCAMPP with the goal of developing a divide-and-conquer strategy that is much faster than SCAMPP but achieves the same scalability goal. The input to BSCAMPP is the same as for SCAMPP: a backbone tree $T$ with relevant numeric parameters (e.g., branch lengths and substitution rate matrix), a set $Q$ of query sequences, a multiple sequence alignment of the sequences at the leaves of the tree as well as $Q$, and a phylogenetic placement method $M$. In this study, we explore BSCAMPP only with pplacer and EPA-ng, but it could be implemented to work with other placement methods. In a sequence of experiments (see Supplementary Materials Section S2), we explored variants on the BSCAMPP design and developed one that we present here.

BSCAMPP has three stages (see Algorithm 1); here we describe it for use with EPA-ng. Stage 1: The similarity score $S(q, s)$ is computed between every query sequence $q$ and leaf sequence $s$, using the multiple sequence alignment (see Section S2.1in the Supplementary Materials). Each query sequence $q$ then votes for $v$ leaves with the largest similarity scores to $q$; the top-scoring leaf is called $closest(q)$. Stage 2a: Using the similarity scores and votes from Step 1, we iteratively grow a set $\mathcal{T}$ of placement subtrees along with initial assignments of each query sequence to one of the subtrees, until each query sequence is assigned to a subtree. Stage 2b: We allow reassignments of each query sequence to a different subtree, based on a sensitive criterion. Stage 3: 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 $T$. This three-stage approach is an elaboration on the SCAMPP technique, except that in SCAMPP, each query sequence picks a single placement subtree; therefore, in the SCAMPP design, it is possible that there will be as many placement trees as there are query sequences.

*Implementation details* BSCAMPP is written in Python with certain parts written using OpenMP in C++. Since computing the similarity score is a computationally intensive portion of the BSCAMPP 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.

**Algorithm 1:** BSCAMPP Algorithm.

Algorithmic parameters: $B$ (default 2000), $v$ (default 5), and phylogenetic placement method $M$.
Input: backbone tree $T$ and leafset $S$, query sequences $Q$, multiple sequence alignment $A$ on $S \cup Q$.
**Stage 1 (Initialization)**:
**for** every query sequence $q \in Q$ **do**
  **Compute** similarity score $S(q, s)$ between $q$ and $s, \forall s \in S$.
  **Select** $v$ top-scoring leaves as the votes of $q$.
  $closest(q) \leftarrow \text{argmax}_s S(q, s)$.
**end for**
**Initialize** $\mathcal{T} \leftarrow \emptyset, Seeds \leftarrow \emptyset$.

**Stage 2a (Constructing $\mathcal{T}$, the set of subtrees, and initial assignment of query sequences)**:
**while** there are query sequences not yet assigned to any subtree **do**
  **Choose** the most-voted leaf $x$ in the tree as $seed$.
  **Build** a subtree $t_x$ with $B$ leaves using breadth-first search based on branch length.
  $Seeds \leftarrow x, \mathcal{T} \leftarrow t_x$.
  **for** every unassigned query sequence $q$ **do**
    **if** $closest(q) \in t_x$ **then**
      assign $q$ to $t_x$ and remove the votes from $q$.
    **end if**
  **end for**
**end while**

**Stage 2b (Allow reassignments of query sequences)**:
**for** every query sequence $q$ and for every tree $t_x \in \mathcal{T}$ **do**
  **Let** $t$ be the tree that $q$ is assigned in Stage 2a.
  **Compute** weighted path distance from $closest(q)$ to $x$.
  **Let** $y \in Seeds$ be the seed sequence that has the minimum distance to $q$.
  **Reassign** $q$ to $t_y$ if $t_y \neq t$.
**end for**

**Stage 3 (Phylogenetic Placement)**:
**for** every subtree $t_x \in \mathcal{T}$ **do**
  **Run** method $M$ on $t_x$ and its assigned query sequences to produce $t'_x$.
  **Use** the technique from SCAMPP to add query sequences in $t'_x$ to $T$ (the backbone tree).
**end for**

**Output:** a file containing all placements, with their requisite confidence score, distal length, placement edge number, etc.

## IV. PERFORMANCE STUDY DESIGN

### A. Overview

We evaluate placement methods for use with short sequences, the application that would be encountered in placing short reads into phylogenetic trees or taxonomies. Since many reads are placed in each run, scalability to large numbers of reads is a relevant question. We use simulated and biological datasets in this study, dividing the datasets into algorithm design data and testing data.

We use simulated datasets for both the algorithm design and testing phases, and we also include a biological dataset in the

TABLE I
TESTING DATASET STATISTICS.

| Dataset | # backbone sequences | # queries | mean sequence length | alignment length | type (bio or sim) | p-distance mean | gaps proportion |
|---|---|---|---|---|---|---|---|
| 16S.B.ALL [28] | 20,000 | 5,093 | 1,366 | 6,857 | biological | .210 | .801 |
| RNASim 50k [25] | 50,000 | $\leq$ 10,000 | 1,545 | 1,590 | simulated | .373 | .028 |
| RNASim 180k [25] | 180,000 | $\leq$ 20,000 | 1,545 | 1,590 | simulated | .373 | .028 |
| nt78 [22] | 68,132 | 10,000 | 1,279 | 1,287 | simulated | .404 | .006 |
| 5000M2 [32] | 4,000 | 10,000 | 1,018 | 52,606 | simulated | .693 | .981 |
| 5000M3 [32] | 4,000 | 10,000 | 992 | 24,062 | simulated | .660 | .958 |
| 5000M4 [32] | 4,000 | 10,000 | 966 | 22,403 | simulated | .530 | .957 |

The first column gives the name of the dataset and the publication describing the dataset. For each dataset, we show the number of sequences, the number of queries, the mean (ungapped) sequence length, the length of the reference alignment, its type (biological or simulated), the mean p-distance (i.e., normalized Hamming distances) between pairs of sequences in the alignment, and the proportion of the alignment that is gapped. Results shown for the RNASim datasets are based on post-processed results (i.e., after sites with at least 95% gaps are masked).

testing data collection. The simulated datasets have known true trees and the biological dataset has an estimated maximum likelihood tree that serves as a reference tree. We report placement error using "delta error", a standard metric used in prior studies [10], [13] (see Section IV-G).

Experiments 1–5 provide an evaluation of BSCAMPP used in conjunction with pplacer or EPA-ng in comparison to SCAMPP used with these methods, and we refer to these combinations as BSCAMPP(e), BSCAMPP(p), SCAMPP(e), and SCAMPP(p). Some of these experiments also include other phylogenetic placement methods (App-SpaM, APPLES-2, UShER [20]).

The base experimental condition uses query sequences that are 10% of the length of the average full-length sequence (i.e., $1279-1545$ nt, depending on the dataset, see Table I) and estimated backbone trees. In our experiments, we explore modifications to this default model condition by changing the query sequence length, adding sequencing error into the query sequences, and placing into true rather than estimated backbone trees.

- Experiment 1 is the design of BSCAMPP, and uses the algorithm design data.
- Experiment 2 compares our divide-and-conquer pipelines (SCAMPP and BSCAMPP used with EPA-ng or pplacer) to all other selected phylogenetic placement methods on the base experimental condition.
- Experiment 3 compares BSCAMPP(e) to UShER [20], APPLES-2, and App-SpaM, using reads with sequencing error.
- Experiment 4 compares BSCAMPP(e) to UShER, APPLES-2, and App-SpaM, on datasets with changing rates of evolution.
- Experiment 5 evaluates scalability of phylogenetic placement methods, and includes a comparison between placing into true and estimated backbone trees.

See Supplementary Materials Section S7 for additional details about datasets and commands used in our experimental study.

### B. Methods

We explore BSCAMPP used with either pplacer (v1.1.alpha19) or EPA-ng (v0.3.8). We compare these to SCAMPP (v2.1.1) used with pplacer and EPA-ng, and also to UShER [20], App-SpaM, RAPPAS, and APPLES-2 (v2.0.11).

Some phylogenetic placement methods require numeric parameters to be estimated for the backbone trees. All backbone tree numeric parameters (branch lengths, $4 \times 4$ substitution rate matrix, etc.) are re-estimated according to the specifications of the phylogenetic placement method: RAxML-ng (v1.0.3) [21] parameters were used for EPA-ng, UShER, App-SpaM, SCAMPP(e), and BSCAMPP(e); FastTree-2 (v2.1.11) [22] under Minimum Evolution parameters were used for APPLES-2. When we run pplacer (whether on its own or inside BSCAMPP(p) and SCAMPP(p)), we use taxtastic [23] with FastTree-2 [22] numeric parameters, since this improves accuracy and scalability according to [12], [17].

### C. Computational Resources

For Experiments 1–4, all methods are given four hours to run with 64 GB of memory and 16 cores. These analyses were run on the UIUC Campus Cluster, which is heterogeneous (i.e., some machines are older and hence slower than others). While all methods are given 16 cores and 64 GB, different analyses may have access to very different computational resources. For these analyses on the Campus Cluster, when placement time for SCAMPP(e), SCAMPP(p), and UShER, was over four hours (which occurred in all experiments with 10,000 or more query sequences), the query sequences were split into subsets of 250 sequences each. SCAMPP(e), SCAMPP(p), and UShER were then run for each subset containing 250 query sequences. Experiment 5 is performed on a dedicated machine with 1 TB of memory and where analyses of up to 4 weeks are permitted.

### D. Datasets

All datasets in this study include a tree and a set of reference sequences for a single gene that are in a multiple sequence alignment. The average ungapped length ranges from 966 to 1545 nt (Table I). For the simulated datasets, we can place into either the model tree or an estimated tree, while for the biological dataset, we can only use an estimated tree. To construct an estimated tree for the simulated datasets, we used FastTree-2 [22], a fast maximum likelihood method, under the GTRGAMMA model. For the 16S.B.ALL dataset, we use a published estimated tree for this dataset, which is a maximum likelihood tree computed using RAxML [24] on the reference sequence alignment. All datasets are from prior studies and are freely available in public repositories (see Data Availability statement).

*1) RNASim:* We use samples from the RNASim dataset [25], 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 [11], [13], [17], 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 algorithm design (Experiment 1) and the larger one for testing (Experiments 2, 3, and 5). We place into a maximum likelihood tree on the true alignment (estimated using FastTree-2), using the true alignment for our Experiments 1 and 5 and using alignments estimated with UPP [26] for Experiments 2 and 3. We have also included an experiment on true trees in the supplementary materials.

*2) nt78:* We also use the nt78 datasets, which were simulated for FastTree-2 [22]; these contain 10 replicates, simulated with Rose [27], each with 78,132 sequences in a multiple sequence alignment and the simulated backbone tree. We picked one replicate randomly, using 68,132 sequences for the backbone and 10,000 sequences for the query sequences. We placed the query sequences into a maximum likelihood tree, estimated using FastTree-2 [22] on the true alignment. This dataset is used in the testing experiments.

*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 The Comparative Ribosomal Website (CRW) [28]. 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 for the SATé-II [29] study on this reference alignment using RAxML [30] and serves as the backbone tree into which we place the query sequences. When computing delta error, we used the 75% bootstrap tree (i.e., the result of collapsing all edges with bootstrap support below 75% ) as the reference topology. The maximum likelihood tree and the 75% bootstrap tree are available at [31].

*4) 5000M(2-4):* This dataset, originally from [32] was generated using INDELible [33] with a heterogeneous indel model. Each set contains 5000 simulated sequences. The 5000M2 condition reflects the highest rate of evolution in the dataset, and the 5000M4 condition reflects the lowest rate of evolution. For our experiments, 1000 sequences are randomly selected as queries and used to generate 10,000 query sequences under the Illumina or PacBio models. We place these queries into a maximum likelihood tree, estimated using FastTree-2 [22] on the true alignment of the remaining 4000 sequences.

### E. Query Sequence Generation

For Experiments 1, 2, and 5 we generated fragments from the full-length sequences for the nt78, 16S.B.ALL, and RNASim datasets, starting at a randomly selected location. The fragmentary sequence lengths are a mean of 10% of the original ungapped sequence length with a standard deviation of 10 nucleotides. Since the average full-length sequences for these datasets are in the 1279–1545 range (Table I), these fragmentary sequences have average lengths in the range 128–155.

For Experiments 3 and 4 we simulated reads with sequencing error. Illumina reads (length 150) were generated using the ART sequence simulator [34], and PacBio reads (length 450) with higher sequencing error were simulated using PBSIM [35].

### F. Additional Details About Experiments

For the RNASim datasets with at least 50,000 sequences, we performed alignment site masking as follows. Those sites containing more than 95% gaps were masked, i.e., removed; this reduced the alignment length from 21,947 to 1,590. Masking was not performed for any other dataset. For the nt78 datasets, we used the third replicate for the experiment. We picked 10,000 sequences at random for the query sequences and used the remaining 68,132 sequences as backbone sequences.

### G. Evaluation Criteria

We report placement error using average delta error [10], [11], [17], where the delta error for a single query sequence is the increase in the number of missing branches (FN) produced by adding the query sequence into the backbone tree, and hence is always non-negative; this is the same as the node distance when the backbone tree is the true 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. See Section S8 in the Supplementary Materials for additional details. The methods are also evaluated with respect to runtime and peak memory usage.

## V. Results

### A. Experiment 1: BSCAMPP Design

To design BSCAMPP, we used EPA-ng as the base method and the algorithm design data. We considered four different strategies, described in Supplementary Materials Section S2.1. We found that variant 4 (see Section III) provided accuracy that was comparable with the next most accurate method, but had better computational performance (Fig. S3 in the Supplementary Materials). Based on this, we selected variant 4.

Having selected variant 4, we then performed additional experiments on the two algorithm design datasets to set the values for two parameters: the size of the subtrees and the number of votes per query sequence. 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. Results for this experiment are shown in Fig. 1 (see also the Supplementary Materials Table S1).

When the subtree size exceeds 2,000 leaves, BSCAMPP(e) had over twice the delta error than it did for 1,000- and 2,000-leaf subtrees; therefore, we set the default subtree size to 2,000. We also see a very small reduction in delta error as the number of votes increases, but the reduction is extremely small and

increases the runtime; therefore, we set the default number of votes to 5.

Using these parameter settings for BSCAMPP(e) and BSCAMPP(p), we show a comparison to SCAMPP(e), SCAMPP(p), and EPA-ng in Fig. 2. EPA-ng was unable to place into the 50,000-leaf backbone tree given the memory limitations. SCAMPP(p) was the most accurate method, with a delta error of 0.39, followed by BSCAMPP(p) at 0.41, SCAMPP(e) at 0.43, and BSCAMPP(e) at 0.45. These four methods differed substantially in runtime, with SCAMPP(p) the slowest, using over 27 hours to place the query sequences, and BSCAMPP(e) the fastest, using just 6 minutes. We also note that SCAMPP produced 8778 subtrees for this dataset while BSCAMPP only produced 101 subtrees; this is the driving factor for why BSCAMPP is so much faster than SCAMPP.

In Experiments 2–4, we will not examine BSCAMPP(p), SCAMPP(p), or SCAMPP(e), and will focus our attention just on BSCAMPP(e).

### *B. Experiment 2: Comparison of BSCAMPP(e) to Other Phylogenetic Placement Methods on the Base Experimental Condition*

In this experiment, we compare BSCAMPP(e) to App-SpaM, RAPPAS, UShER, APPLES-2, and EPA-ng on the testing datasets: nt78, RNASim 50 K, and 16S.B.ALL. All query sequences are fragmentary with lengths 10% of their full lengths. As with all our experiments, the methods were given 64 GB of memory and were run on the UIUC Campus Cluster, a heterogeneous computational infrastructure, which limits all analyses to four hours.

To avoid biasing in favor of the alignment-based methods, we used estimated rather than true query alignments for this experiment. The results when query sequences are placed with an alignment using UPP [26] are shown in Fig. 3.

RAPPAS and EPA-ng failed to complete on these datasets (except for EPA-ng on 16S.B.ALL) due to needing more than the available memory (64 GB); the other methods succeeded in running on all the datasets and placed all the query sequences into the backbone trees.

APPLES-2 was consistently much less accurate than BSCAMPP(e) and UShER. App-SpaM was slightly more accurate than BSCAMPP(e) and UShER on the 16S.B.ALL dataset and then much less accurate on the other two datasets. UShER was slightly more accurate than BSCAMPP(e) on the 16S.B.ALL dataset but less accurate (by a slightly larger amount) on the other two datasets. Overall, therefore, BSCAMPP(e) and UShER are the two most accurate methods on these three datasets, with perhaps a small advantage to BSCAMPP(e).

App-SpaM was by far the fastest method and UShER was the slowest method. BSCAMPP(e) and APPLES-2 were very close in runtime, each with an advantage over the other on one dataset. EPA-ng only completed on one of the three datasets, and on this dataset its runtime was similar to BSCAMPP(e) and APPLES-2. As seen in Table S2 in the Supplementary materials, the speed advantage of App-SpaM over all other methods is due to the time used to perform the query alignment (approx. 29 minutes for 16S.B.ALL, 115 minutes for RNASim, and 55 minutes for nt78). Thus, App-SpaM, which is alignment-free, is much faster than the other methods, which all require query alignments. The methods also differed with respect to peak memory usage, with EPA-ng having the highest memory requirement and App-SpaM the second highest; the other methods have very low memory usage on these datasets.

### *C. Experiment 3: Performance Using Reads With Sequencing Error*

In Experiment 3, we compare BSCAMPP(e) to APPLES-2, UShER, and App-SpaM on simulated reads under Illumina and PacBio error models for the testing datasets 16S.B.ALL, nt78, and RNASim 50 k. All simulated reads were aligned to the reference sequences with UPP, and the alignment runtime is included along with the placement time for alignment-based methods (BSCAMPP(e), UShER, and APPLES-2) in this experiment.

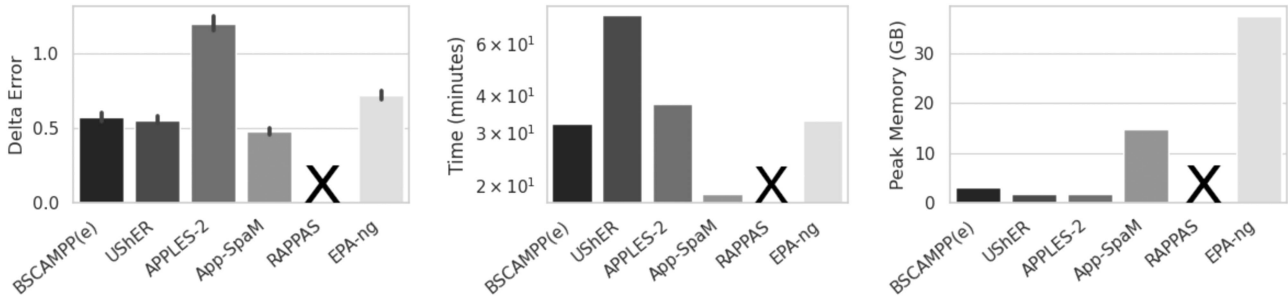
On the Illumina read condition (Fig. 4), we see the following trends. On the simulated datasets (RNASim and nt78), BSCAMPP(e) was the most accurate method, followed by UShER, APPLES-2, and then App-SpaM. On the biological dataset (16S.B.ALL), App-SpaM was the most accurate method, and BSCAMPP(e) and UShER closely follow. APPLES-2 shows over twice the placement error of BSCAMPP(e) and UShER.

Runtime for all alignment-based methods includes the read alignment process. As in Experiment 2, this accounts for the majority of the runtime for both BSCAMPP(e) and APPLES-2 (see Table S3 in the Supplementary Materials). UShER was the slowest method and App-SpaM was the fastest, with near instantaneous placement. App-SpaM used more memory than the alignment-based methods, requiring at least 30 GB for the simulated datasets and 15 GB for the biological dataset, while the alignment-based methods used at most 3 GB on each dataset.

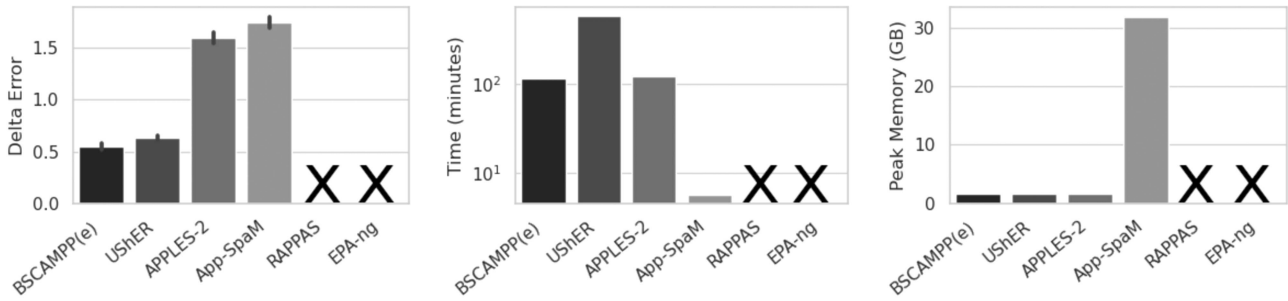
Results on the PacBio-style reads show similar trends (Fig. 5). In all cases, BSCAMPP(e) was the most accurate method, closely followed by UShER. APPLES-2 had more than twice the error of BSCAMPP(e) and UShER, and App-SpaM had the highest error. Runtime trends are similar to the Illumina-style reads. For each condition, App-SpaM placed all query sequences in at most 7 minutes, while the other methods used at least two hours (with UShER using the most time). Furthermore, the query alignment step dominates the runtime for BSCAMPP(e) and APPLES-2 (see Table S4 in the Supplementary Materials). BSCAMPP(e) had slightly higher memory usage than UShER and APPLES-2, using up to 4 GB for the biological dataset, and App-SpaM used the most memory of all methods shown.

### *D. Experiment 4: Performance on Datasets With Variable Rates of Evolution*

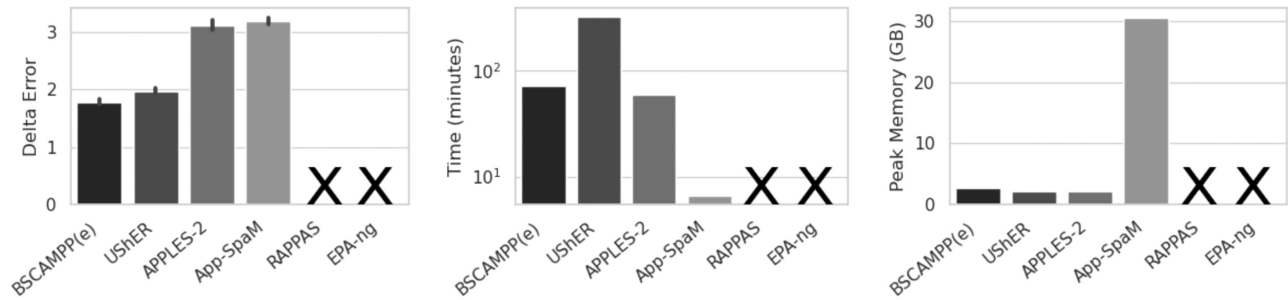
The purpose of Experiment 4 is to evaluate placement methods as the rate of evolution changes. We used the 5000M2–5000M4 datasets (see Table I). These datasets have 5000 sequences and different rates of evolution (moderate for 5000M4 to high for 5000M2). We selected 4000 sequences for the backbone tree and used the remaining 1000 sequences to generate query




(a) 16S.B.ALL (20,000-leaf backbone tree, placing 5,093 query sequences)




(b) RNASim (50,000-leaf backbone tree, placing 10,000 query sequences)




(c) nt78 (68,132-leaf backbone tree, placing 10,000 query sequences)

Fig. 3. Experiment 2: Performance using estimated query sequence alignments on testing data. We show from left to right - Mean delta error, total runtime, and peak memory usage in GB for three datasets placing large sets of queries into large estimated reference trees. The query sequences are a mean of 10% of the original ungapped sequence length (i.e., $\sim$ 137 nt for 16S.B.ALL, 155 nt for RNASim, and 128 nt for nt78) with a standard deviation of 10 nucleotides. Query sequence alignments are estimated using UPP, and the alignment time is included in the runtime for the alignment-based methods (all except RAPPAS and App-SpaM). We show placement time for BSCAMPP(e) for 5 votes with a subtree size of 2000 (default settings). The results from UShER, APPLES-2, App-SpaM, RAPPAS, and EPA-ng are also included (an **X** indicates that RAPPAS and EPA-ng were unable to run due to out-of-memory issues).

sequences. These query sequences were generated under two models of sequencing error: Illumina and PacBio.

APPLES-2 did not return placements for some queries for many of these datasets (see Supplementary Materials Table S5). We show delta error on placements for those query sequences that returned placements for all methods in Fig. 6; for results on all 10,000 queries (without APPLES-2), see Supplementary Fig. S4.

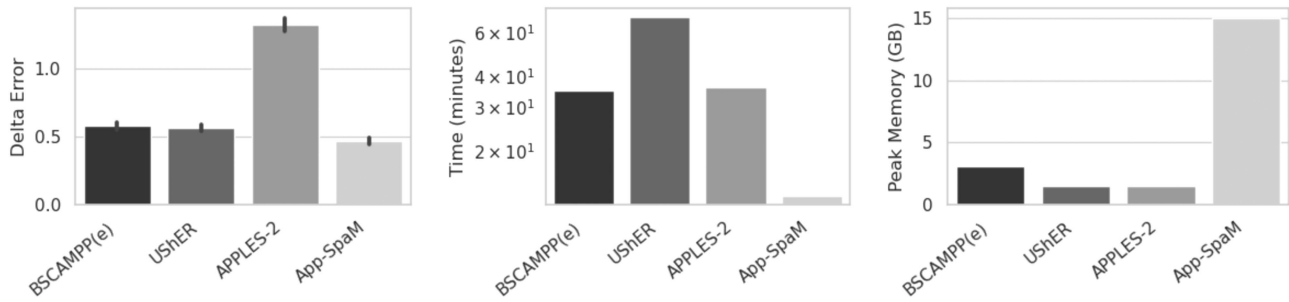
On Illumina reads (Fig. 6(a)), BSCAMPP(e) was the most accurate method for all conditions. On the 5000M2 data, the condition with the highest rate of evolution, BSCAMPP(e) had less than half the error of UShER, the second most accurate method. The other two methods, APPLES-2 and App-SpaM, had much higher error. As the rate of evolution lowers, the relative accuracy remains the same, but all methods improved in accuracy and the gap between methods decreased.

Results on PacBio reads (Fig. 6(b)) show a less clear delineation between the accuracy of BSCAMPP(e) and UShER, but BSCAMPP(e) was still more accurate, closely followed by UShER and APPLES-2. App-SpaM had a higher error than all other methods.

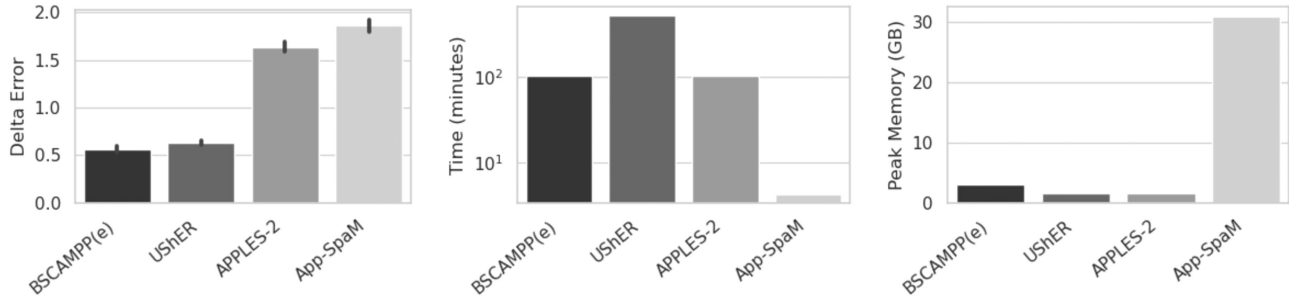
The runtime for the alignment-based methods have the alignment phase included. Of these methods, BSCAMPP(e) was the fastest, followed by APPLES-2 and then UShER. App-SpaM, which is alignment-free, was so fast that its runtime is not even visible in Fig. 6, and required the least memory usage. BSCAMPP(e) used more memory than all other methods, up to 15 GB.

### *E. Experiment 5: Scalability Experiment*

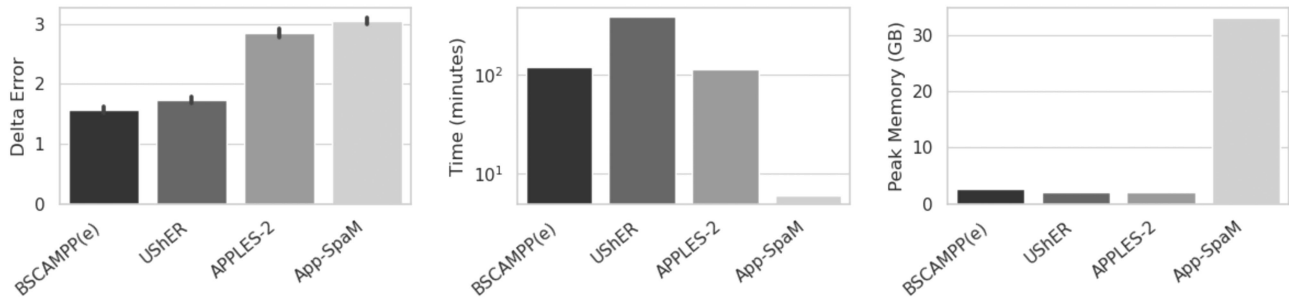
In this experiment, we explore BSCAMPP(e) scalability. The previous experiments were all run on the Campus Cluster, a




(a) 16S.B.ALL (20,000-leaf backbone tree, placing 5,093 Illumina reads)




(b) RNASim (50,000-leaf backbone tree, placing 10,000 Illumina reads)




(c) nt78 (68,132-leaf backbone tree, placing 10,000 Illumina reads)

Fig. 4. Experiment 3: Results on Illumina reads (query length 150 nt). For each of the three testing datasets (given in subfigures (a)– (c)), we show mean delta error (left), total runtime (middle), and peak memory usage (right) for phylogenetic placement methods (BSCAMPP(e), APPLES-2, UShER, and App-SpaM). The runtime shown includes the time to add sequences into the reference alignment using UPP for BSCAMPP(e), APPLES-2, and UShER.

heterogeneous infrastructure. Although all analyses were guaranteed 64 GB of memory, the heterogeneity of the Campus Cluster means that the runtime comparisons are noisy. Hence, in this section, we provide a limited evaluation using a dedicated machine with 1 TB of memory. This machine also allows runtime of up to 4 weeks and so allows us to explore all the best methods, even when placing a large number of sequences into very large trees.

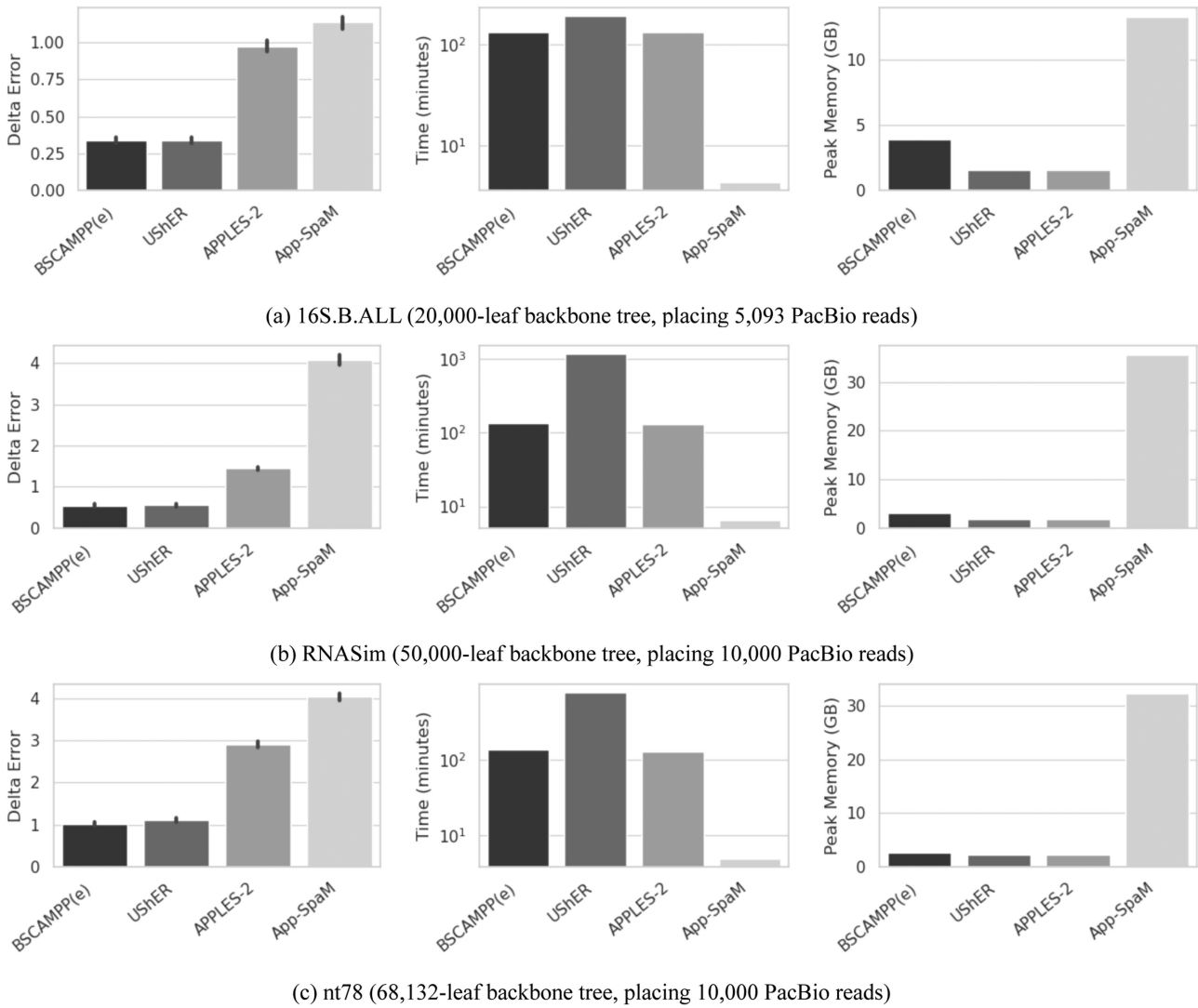
We examine computational performance for BSCAMPP(e) under three conditions:

- as a function of the number of query sequences
- as a function of the query sequence length
- as a function of the backbone tree size

For the first three evaluations (see Supplementary Materials Figs. S5 to S7), BSCAMPP(e) runtime increased as the query sequence length, number of query sequences, or backbone tree size increased, with at most a linear impact. However, the tree size had the largest impact on the runtime, in that doubling the tree size almost doubled the runtime, and the impact was less for the others. In addition, changes to these numbers did not impact the peak memory usage.

We also evaluated phylogenetic placement scalability to very large query sets and ultra-large backbone tree sizes. We limited this study to the alignment-based methods, which provided the highest accuracy in previous experiments, and so use the true alignment for this experiment. We used RNASim 180K for this study, with an estimated backbone tree (computed using FastTree) of 180,000 leaves and placing 20,000 query sequences of 10% of the full-length. We included all four of our pipelines, i.e., BSCAMPP(e), BSCAMPP(p), SCAMPP(e), and SCAMPP(p). We also included EPA-ng and APPLES-2. We did not attempt to run pplacer, as our prior studies [11] shown it fails on smaller subtrees of the RNASim simulation due to numerics issues.

The methods ranged substantially in placement accuracy. All four of our pipelines (BSCAMPP or SCAMPP used with EPA-ng or pplacer) had lower error than EPA-ng (which had more than




(a) 16S.B.ALL (20,000-leaf backbone tree, placing 5,093 PacBio reads)

(b) RNASim (50,000-leaf backbone tree, placing 10,000 PacBio reads)

(c) nt78 (68,132-leaf backbone tree, placing 10,000 PacBio reads)

Fig. 5. Experiment 3: Results on PacBio reads (average query length 450 nt). For each of the three testing datasets (given in subfigures (a)– (c)), we show mean delta error (left), total runtime (middle), and peak memory usage (right) for phylogenetic placement methods (BSCAMPP(e), APPLES-2, UShER, and App-SpaM). The runtime shown includes the time to add sequences into the reference alignment using UPP for BSCAMPP(e), APPLES-2, and UShER.

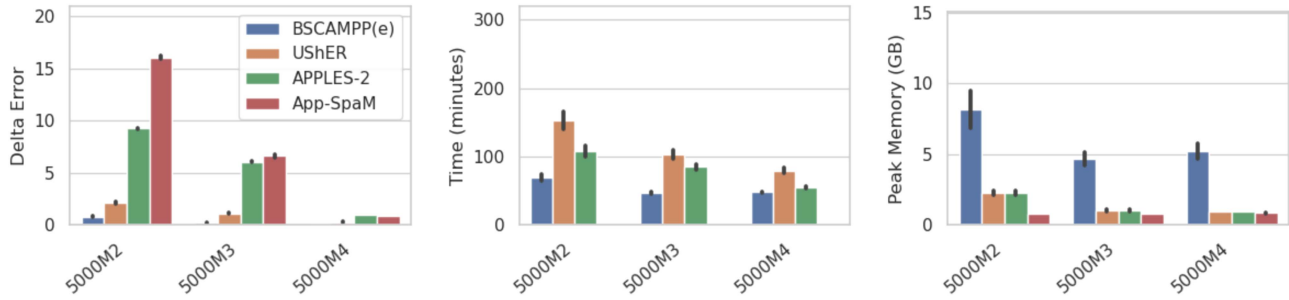
twice the error of the least accurate of these pipelines) and APPLES-2 had much higher error. Between the four pipelines, from most accurate to least accurate, we have SCAMPP(p), BSCAMPP(p), SCAMPP(e), and BSCAMPP(e), but the difference in error was extremely small (and all four pipelines had average delta error between 0.35 and 0.40).

The pipelines also differed for running time and memory usage. The three fastest methods were APPLES-2, BSCAMPP(e), and EPA-ng (and in that order), and all finished in under 20 minutes. BSCAMPP(p) was also reasonably fast, finishing in under an hour. The other two methods were much slower, with SCAMPP(e) finishing in almost 8 hours and SCAMPP(p) needing more than 13 hours to complete. Peak memory usage also varied between methods. From least to most peak memory usage we have APPLES-2 at 0.9 GB, BSCAMPP(p) at 0.9 GB, BSCAMPP(e) at 2.8 GB, SCAMPP(e) at 18.8 GB, SCAMPP(p) at 18.8 GB, and EPA-ng at 270.5 GB, Thus, BSCAMPP(e) and BSCAMPP(p) used a small amount of memory, SCAMPP(e) and SCAMPP(p) used a moderate amount of memory, and only EPA-ng used a large amount of memory.

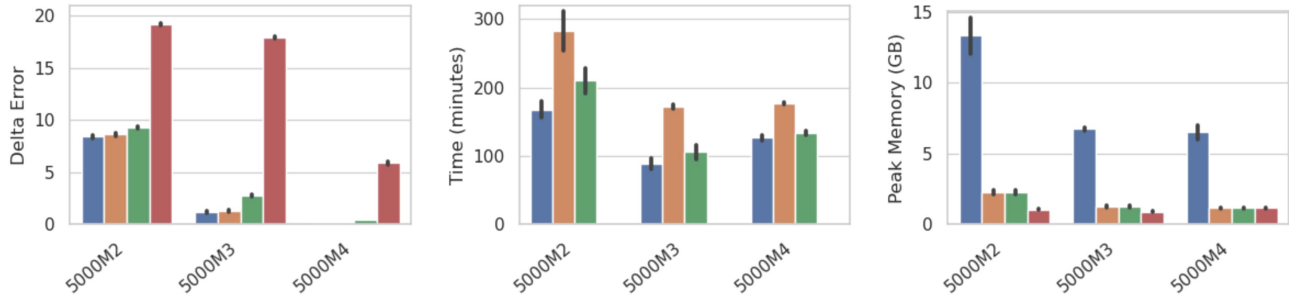
We also compared results for placing 2000 sequences into the RNASim 180 k tree using BSCAMPP(e), BSCAMPP(p), SCAMPP(e), SCAMPP(p), and APPLES-2 on true and estimated backbone trees; see Supplementary Materials Fig. S8. Using the true tree rather than estimated backbone tree had no impact on runtime or peak memory usage, as expected. Using true rather than estimated backbone trees had no impact on delta error for BSCAMPP(p) and SCAMPP(p), very slightly increased error for BSCAPP(e) and SCAMPP(e), and then reduced error for APPLES-2. However, even on true trees as well as estimated backbone trees, the four SCAMPP/BSCAMPP pipelines were still much more accurate than APPLES-2.

## VI. Discussion

The methods we explored differed in runtime, memory usage, and placement accuracy. RAPPAS, App-SpaM, and EPA-ng had




(a) Placing 10,000 Illumina reads into 4,000-leaf trees




(b) Placing 10,000 PacBio reads into 4,000-leaf trees

Fig. 6. Experiment 4: Impact of the rate of evolution and sequencing error model. Results are shown for three rates of evolution: 5000M2 (highest) to 5000M4 (lowest). (a) Illumina reads (query length 150 nt) and (b) PacBio reads (average query length 450 nt). Each row shows delta error (left), runtime (middle), and memory usage (right) for BSCAMPP(e), UShER, APPLES-2, and App-SpaM. Each method places 10,000 queries into 4,000-leaf estimated trees on two replicates for each dataset. Delta error is shown only for sequences that all methods successfully placed. APPLES-2 failed to place some query sequences (up to 208 out of 10,000) in some model condition; see Supplementary Material Table S5. The runtime shown includes the time to align the query sequences to the reference alignment (needed for all methods other than App-SpaM).

high memory requirements, making all of these methods unable to scale to the largest dataset we examined (RNASim 180K, with 180K sequences in the backbone tree and 20,000 query sequences) on the UIUC Campus Cluster, with the limit to 64 GB of memory. Indeed, EPA-ng and RAPPAS were unable to run on the RNASim 50 K dataset under these conditions, due to their memory requirements. Thus, these three methods had higher memory requirements than the remaining methods, which included APPLES-2, UShER, and the four SCAMPP and BSCAMPP pipelines. This finding is perhaps as expected, as APPLES-2 and the four SCAMPP/BSCAMPP pipelines use divide-and-conquer, limiting the phylogenetic placement effort to small placement subtrees, and UShER's mutation-annotated tree reduces the memory requirements. However, we also confirmed that EPA-ng *can* complete on the RNASim 180K tree, placing 20,000 sequences into the tree, when given adequate memory (Experiment 5).

APPLES-2, BSCAMPP(e), and App-SpaM were the fastest methods, with a definite advantage to App-SpaM that is due to its not needing to perform a query alignment. However, as noted above, App-SpaM did not complete on the largest backbone trees, due to the limitation to 64 GB, while SCAMPP(p), SCAMPP(e), and UShER were the slowest of the methods we tested.

The large speed advantage of BSCAMPP over SCAMPP is due to its producing a much smaller number of subtrees (at most 300 for any condition, whereas SCAMPP produced 39-86 times as many subtrees as BSCAMPP (Table S6 in the Supplementary Materials); for example, SCAMPP produced 18,590 subtrees for placing 20,000 query sequences into the RNASim 50 K tree, while BSCAMPP produced only 300. This is the driving reason that BSCAMPP is so much faster than SCAMPP.

Our experiments showed that the four pipelines we present usually had the lowest delta error of the tested methods, with relatively minor differences between them, making computational performance the main distinguishing feature. Therefore, the finding that BSCAMPP(e) is much faster than the other likelihood-based methods on these datasets makes it perhaps the method of choice for most applications where speed is important.

The accuracy achieved by BSCAMPP(e) on the conditions explored in this study was very high, surpassed only by BSCAMPP(p), SCAMPP(e), and SCAMPP(p). Moreover, BSCAMPP(e) had high accuracy even when placing into very large trees (e.g., the RNASim 180K tree studied in Experiment 5, see Fig. 7). It also had high accuracy, better than the methods it was compared to, when placing Illumina reads into trees with high evolutionary diameters and where the reference alignment was very gappy (the 5000M2 condition studied in Experiment 4, see Fig. 6).

Indeed, in this study, the only substantially challenging condition for our pipelines was placing PacBio reads into trees with very high rates of evolution, i.e., the 5000M2 condition. Even here, BSCAMPP(e) was more accurate than the other tested methods, although the accuracy gap between BSCAMPP(e),

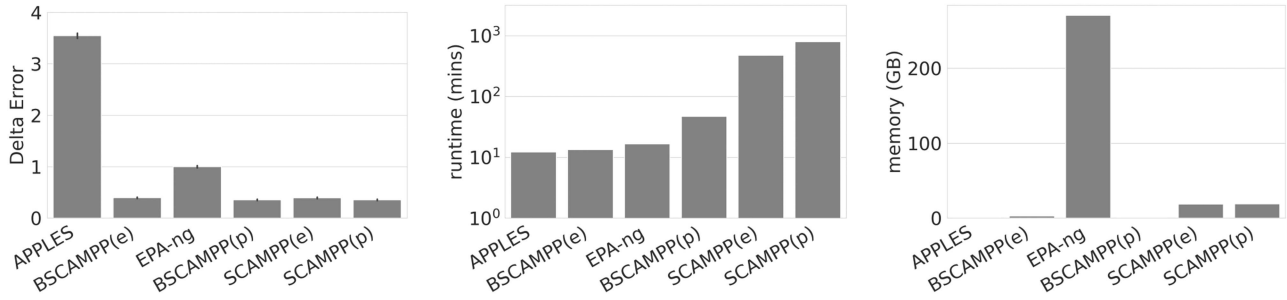



Fig. 7. Experiment 5: Evaluation of methods for placing 20,000 query sequences into the estimated RNA 180 K tree, using a dedicated machine. We show results for APPLES-2, BSCAMPP(e), BSCAMPP(p), SCAMPP(e), SCAMPP(p), and EPA-ng. Each method was given 16 cores and 512 GB of memory, and allowed to run to completion. We show delta error (left), runtime in minutes (middle, logarithmic scale) and peak memory usage in GBs (right) when placing 20,000 query sequences (average length 155 nt) into an RNASim tree with 180,000 leaves. All methods use the true alignment of query sequences to the reference alignment. SCAMPP produced 18,590 subtrees for this dataset but BSCAMPP produced only 300 subtrees.

UShER, and APPLES-2 was small. However, we also see that when the rate of evolution dropped (the 5000M3 and 5000M4 conditions), placement accuracy improved substantially. Thus, placement of reads with high sequencing error is in itself not a major challenge, nor is placing into trees with high evolutionary diameters; rather, it is the combination of the two that presents a large challenge to phylogenetic placement accuracy.

Also of interest is the comparison between EPA-ng and our four pipelines (BSCAMPP and SCAMPP used with either EPA-ng or pplacer) when placing into the largest tree we examine, RNASim 180K (see Fig. 7). In this experiment, each of our pipelines was much more accurate than EPA-ng. We also see that the pipelines based on pplacer were slightly more accurate than the pipelines based on EPA-ng, and pipelines based on SCAMPP were slightly more accurate than pipelines based on BSCAMPP. Even so, on this dataset, these differences were small compared to the gap between EPA-ng and BSCAMPP(e) (the least accurate of the four pipelines).

To understand why EPA-ng has high error in this experiment, 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 [11], [13], [17], but our preliminary study showed that EPA-ng had a jump in placement delta error as we increased the subtree size for the RNASim dataset. Thus, our study potentially suggests 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 [11]. Further research is needed to understand whether this explanation is correct.

## VII. Conclusion

Phylogenetic placement of sequences 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. Our prior work has shown that EPA-ng and pplacer, the two leading maximum likelihood based methods for phylogenetic placement, failed to run on large backbone trees (EPA-ng due to memory requirements and pplacer due to numerical issues). SCAMPP [11] was designed to improve scalability of likelihood-based phylogenetic placement methods to large backbone trees: each query sequence extract a small subtree, into which it is then placed using the likelihood-based method. This approach provides high accuracy and scalability to large trees, but is nevertheless computationally intensive for placing large numbers of sequences, because of the number of subtrees that are extracted.

We designed BSCAMPP in order to achieve comparable scalability but much reduced speed compared to SCAMPP. To reduce the number of subtrees that are extracted, BSCAMPP uses a voting technique to select a small number of subtrees that suffices for all the query sequences. Our study shows that BSCAMPP, used with either pplacer or EPA-ng, is very close to the accuracy of SCAMPP and provides the same scalability improvement for both EPA-ng and pplacer, while dramatically reducing the runtime. Moreover, BSCAMPP used with EPA-ng, i.e., BSCAMPP(e), is extremely fast, and in many cases as fast as APPLES-2. Furthermore, BSCAMPP(e) scales well with the number of query sequences and query sequence length, making it suitable for phylogenetic placement whenever the backbone tree or number of sequences is large. Our study also shows that BSCAMPP used with pplacer, i.e., BSCAMPP(p), provides slightly better accuracy than BSCAMPP(e) but is somewhat slower. Whether this improvement in accuracy is worth the extra time depends on the application and dataset.

This study leaves several directions for future research. A more extensive study should explore phylogenetic placement of full-length sequences, and possibly also consider the problem of placing genome-length sequences. This particular direction raises issues of heterogeneity across the genome [36], a problem that is addressed by the DEPP [4] method for phylogenetic placement. In addition, while BSCAMPP(e) is fast, a possible improvement for the runtime could 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. Furthermore, given the cost of computing the query alignments, research into speeding up these alignments while maintaining high accuracy is also merited.

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 maximum likelihood tree estimation when there is substantial sequence length heterogeneity. Our prior work has shown that FastTree2, a very fast maximum likelihood method, has reduced accuracy on datasets with many short sequences [14], [37]. Alternative approaches that first construct a tree on the full-length sequences and then add the short sequences into the tree using phylogenetic placement methods have the potential to provide improved accuracy and scalability on large trees. Exploring BSCAMPP in this context is therefore a good direction for future research.

This study also suggests a potential benefit for BSCAMPP(p), compared to BSCAMPP(e), for applications where high accuracy is important. TIPP3 [3] is a method for taxonomic abundance profiling of metagenomic reads that uses pplacer to locate reads within gene-based taxonomies, and provides improved accuracy over TIPP [1] and TIPP2 [2], mainly because it performs phylogenetic placement into larger trees. A fast variant of TIPP3, called TIPP3-fast, uses BSCAMPP(e) instead of pplacer [3], and provides a great improvement in speed over TIPP3, but increases the abundance profiling error slightly. The results in this paper suggest the possibility that using BSCAMPP(p) within TIPP3 might be a fruitful compromise between pplacer (which had the highest accuracy) and BSCAMPP(e) (which was the fastest), suggesting another direction for future work.

Overall, this study shows the potential for phylogenetic placement methods based on maximum likelihood to provide very high accuracy, even under very challenging conditions, such as placing into very large trees, placing into trees with high evolutionary diameters, or placing reads with high sequencing error. While the relative accuracy between methods depends on the conditions (properties of the query sequences and backbone tree), very often maximum likelihood-based methods provide better accuracy than other approaches, and especially better than methods that are alignment-free. These observations support the continued development of methods that methods like EPA-ng and pplacer, as well as methods like BSCAMPP that aim to improve the scalability of these methods to large trees.

## VII. Data Availability

All datasets used in this study are from prior publications. The RNASim dataset is available at https://databank.illinois.edu/datasets/IDB-1048258. The 16S.B.ALL and nt78 datasets are available at https://databank.illinois.edu/datasets/IDB-9257957. The 5000M2–5000M4 datasets are available at https://databank.illinois.edu/datasets/IDB-2567453.

## References


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

[2] N. Shah, E. K. Molloy, M. Pop, and T. Warnow, "TIPP2: Metagenomic taxonomic profiling using phylogenetic markers," *Bioinformatics*, vol. 37, pp. 1839–1845, 2021, doi: 10.1093/bioinformatics/btab023.

[3] C. Shen, E. Wedell, M. Pop, and T. Warnow, "TIPP3 and TIPP3-fast: Improved abundance profiling in metagenomics," *PLoS Comput. Biol.*, vol. 21, no. 4, 2025, Art. no. e1012593.

[4] Y. Jiang, M. Balaban, Q. Zhu, and S. Mirarab, "DEPP: Deep learning enables extending species trees using single genes," *Systematic Biol.*, vol. 72, no. 1, pp. 17–34, 2023.

[5] L. Czech, A. Stamatakis, M. Dunthorn, and P. Barbera, "Metagenomic analysis using phylogenetic placement—a review of the first decade," *Front. Bioinf.*, vol. 2, 2022, Art. no. 871393. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fbinf.2022.871393

[6] S. Conlan, H. H. Kong, and J. A. Segre, "Species-level analysis of DNA sequence data from the NIH human microbiome project," *PLoS One*, vol. 7, no. 10, 2012, Art. no. e47075.

[7] C. O. McCoy and F. A. Matsen, "Abundance-weighted phylogenetic diversity measures distinguish microbial community states and are robust to sampling depth," *PeerJ*, vol. 1, 2013, Art. no. e157.

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

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

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

[11] E. Wedell, Y. Cai, and T. Warnow, "SCAMPP: Scaling alignment-based phylogenetic placement to large trees," *IEEE/ACM Trans. Comput. Biol. Bioinf.*, vol. 20, no. 2, pp. 1417–1430, Mar./Apr. 2022, doi: 10.1109/TCBB.2022.3170386.

[12] G. Chu and T. Warnow, "SCAMPP+FastTree: Improving scalability for likelihood-based phylogenetic placement," *Bioinf. Adv.*, vol. 3, no. 1, 2023, Art. no. vbad008, doi: 10.1093/bioadv/vbad008.

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

[14] P. Zaharias and T. Warnow, "Recent progress on methods for estimating and updating large phylogenies," *Philos. Trans. Roy. Soc. B*, vol. 377, no. 1861, 2022, Art. no. 20210244.

[15] P. Barbera et al., "EPA-ng: Massively parallel evolutionary placement of genetic sequences," *Systematic Biol.*, vol. 68, no. 2, pp. 365–369, 2019.

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

[17] M. Balaban, Y. Jiang, D. Roush, Q. Zhu, and S. Mirarab, "Fast and accurate distance-based phylogenetic placement using divide and conquer," *Mol. Ecol. Resour.*, vol. 22, no. 3, pp. 1213–1227, 2022.

[18] B. Linard, K. Swenson, and F. Pardi, "Rapid alignment-free phylogenetic identification of metagenomic sequences," *Bioinformatics*, vol. 35, no. 18, pp. 3303–3312, 2019, doi: 10.1093/bioinformatics/btz068.

[19] M. Blanke and B. Morgenstern, "App-SpaM: Phylogenetic placement of short reads without sequence alignment," *Bioinf. Adv.*, vol. 1, no. 1, 2021, Art. no. vbab027, doi: 10.1093/bioadv/vbab027.

[20] Y. Turakhia et al., "Ultrafast Sample placement on Existing tRees (UShER) enables real-time phylogenetics for the SARS-CoV-2 pandemic," *Nature Genet.*, vol. 53, no. 6, pp. 809–816, 2021.

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

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

[23] FHCRC, "Taxtastic software, version 9.9. 2," 2022. Accessed: Feb. 20, 2022. [Online]. Available: https://github.com/fhcrc/taxtastic

[24] A. Stamatakis, "RAxML version 8: A tool for phylogenetic analysis and post-analysis of large phylogenies," *Bioinformatics*, vol. 30, no. 9, pp. 1312–1313, 2014.

[25] S. Mirarab, N. Nguyen, S. Guo, L.-S. Wang, J. Kim, and T. Warnow, "PASTA: Ultra-large multiple sequence alignment for nucleotide and amino-acid sequences," *J. Comput. Biol.*, vol. 22, no. 5, pp. 377–386, 2015.
[26] N.-P. D. Nguyen, S. Mirarab, K. Kumar, and T. Warnow, "Ultra-large alignments using phylogeny-aware profiles," *Genome Biol.*, vol. 16, no. 1, 2015, Art. no. 124.
[27] J. Stoye, D. Evers, and F. Meyer, "Rose: Generating sequence families," *Bioinformatics*, vol. 14, no. 2, pp. 157–163, 1998, doi: 10.1093/bioinformatics/14.2.157.
[28] J. J. Cannone et al., "The comparative RNA web (CRW) site: An online database of comparative sequence and structure information for ribosomal, intron, and other RNAs," *BMC Bioinf.*, vol. 3, no. 1, 2002, Art. no. 2.
[29] K. Liu et al., "SATé-II: Very fast and accurate simultaneous estimation of multiple sequence alignments and phylogenetic trees," *Systematic Biol.*, vol. 61, no. 1, 2012, Art. no. 90.
[30] A. Stamatakis, "RAxML-VI-HPC: Maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models," *Bioinformatics*, vol. 22, no. 21, pp. 2688–2690, 2006.
[31] S. Mirarab and T. Warnow, "Data for 16s and 23s rRNA alignments," 2017. [Online]. Available: https://doi.org/10.13012/B2IDB-1614388_V1
[32] C. Shen, B. Liu, K. P. Williams, and T. Warnow, "EMMA: A new method for computing multiple sequence alignments given a constraint subset alignment," *Algorithms Mol. Biol.*, vol. 18, no. 1, Dec. 2023, Art. no. 21, doi: 10.1186/s13015-023-00247-x.
[33] W. Fletcher and Z. Yang, "INDELible: A flexible simulator of biological sequence evolution," *Mol. Biol. Evol.*, vol. 26, no. 8, pp. 1879–1888, 2009.
[34] W. Huang, L. Li, J. R. Myers, and G. T. Marth, "ART: A next-generation sequencing read simulator," *Bioinformatics*, vol. 28, no. 4, pp. 593–594, Feb. 2012, doi: 10.1093/bioinformatics/btr708.
[35] Y. Ono, K. Asai, and M. Hamada, "PBSIM: PacBio reads simulator—toward accurate genome assembly," *Bioinformatics*, vol. 29, no. 1, pp. 119–121, Jan. 2013, doi: 10.1093/bioinformatics/bts649.
[36] W. P. Maddison, "Gene trees in species trees," *Systematic Biol.*, vol. 46, no. 3, pp. 523–536, 1997.
[37] V. Smirnov and T. Warnow, "Phylogeny estimation given sequence length heterogeneity," *Systematic Biol.*, vol. 70, no. 2, pp. 268–282, 2021.

**Eleanor Wedell** received the undergraduate degree in applied mathematics from Drexel University, in 2015 and the MS degree in computer science from the University of Illinois Urbana-Champaign in 2022. She is currently working toward the PhD degree with Tandy Warnow in the Siebel School of Computing and Data Science, University of Illinois Urbana-Champaign. Her current work focuses on bioinformatics and network science.

**Chengze Shen** received the bachelors' degree with the University of California San Diego, and the MS degree from Carnegie Mellon University. He is currently working toward the PhD degree in the Siebel School of Computing and Data Science with the University of Illinois Urbana-Champaign. His current research is focused on bioinformatics.

**Tandy Warnow** received the PhD degree in mathematics from the University of California at Berkeley in 1991 under the direction of Gene Lawler, and her research focuses on reconstructing complex and large-scale evolutionary histories. She is the Grainger Distinguished Chair in Engineering in the Siebel School of Computing and Data Science with the University of Illinois Urbana-Champaign.