pplacerDC: a New Scalable Phylogenetic Placement Method

Elizabeth Koning
ekoning2@illinois.edu
University of Illinois
Urbana-Champaign
Urbana, Illinois, USA

Malachi Phillips
malachi2@illinois.edu
University of Illinois
Urbana-Champaign
Urbana, Illinois, USA

Tandy Warnow
warnow@illinois.edu
University of Illinois
Urbana-Champaign
Urbana, Illinois, USA

ABSTRACT

Motivation: Phylogenetic placement (i.e., the insertion of a sequence into a phylogenetic tree) is a basic step in several bioinformatics pipelines, including taxon identification in metagenomic analysis and large scale phylogeny estimation. The most accurate current method is pplacer, which attempts to optimize the placement using maximum likelihood, but it frequently fails on datasets where the phylogenetic tree has 5000 leaves. APPLES is the current most scalable method, and EPA-ng, although more scalable than pplacer and more accurate than APPLES, also fails on many 50,000-taxon trees. Here we describe pplacerDC, a divide-and-conquer approach that enables pplacer to be used when the phylogenetic tree is very large.

Results: Our study shows that pplacerDC has excellent accuracy and scalability, matching pplacer where pplacer can run, improving accuracy compared to APPLES and EPA-ng, and is able to run on datasets with up to 100,000 sequences.

Availability: The pplacerDC code is available on GitHub at https://github.com/kodingkoning/pplacerDC.

CCS CONCEPTS

• **Applied computing** → **Life and medical sciences**.

KEYWORDS

phylogenetic placement, pplacer

ACM Reference Format:
Elizabeth Koning, Malachi Phillips, and Tandy Warnow. 2021. pplacerDC: a New Scalable Phylogenetic Placement Method. In *12th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics (BCB '21), August 1–4, 2021, Gainesville, FL, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3459930.3469516

1 INTRODUCTION

Phylogenetic placement (the addition of a sequence into a "backbone" phylogenetic tree) is a basic step in many bioinformatics pipelines, including large-scale phylogenetic tree construction [1] and taxon identification of microbial reads (e.g., providing an estimate of the taxonomic characterization of a read from a metagenomic sample, as performed in TIPP [12]). In both of these cases, the backbone tree is generally large and the number of sequences that need to be added may also be very large; hence, phylogenetic placement methods that can operate efficiently and with high accuracy given large backbone trees and numbers of query sequences is important.

Several methods have been developed for phylogenetic placement, including pplacer [8], EPA-ng [2], APPLES [1], RAPPAS [6], and App-SpaM [3]. Of these, pplacer and EPA-ng are based on maximum likelihood (i.e., they find the placement in the tree where the extended alignment/tree pair has the largest maximum likelihood score among all possible placements), while the other methods are based on distances. As a result, pplacer and EPA-ng are more computationally intensive than the other methods. Several studies have shown that pplacer and EPA-ng tend to have better accuracy than the distance-based methods, with pplacer having an accuracy advantage over EPA-ng [1, 3, 6]. However, pplacer and EPA-ng are limited to smaller backbone trees (pplacer frequently fails on 5000-leaf backbone trees and EPA-ng fails on 50,000-leaf backbone trees), whereas APPLES can work with much larger backbone trees (e.g., 100,000 leaves). Given the planned use of phylogenetic placement methods (which is to add query sequences into large trees), this substantially limits the utility of these likelihood-based placement methods.

Here we present a new phylogenetic placement method that achieves the scalability of APPLES and has much better accuracy than APPLES. The method, which we call pplacerDC, operates by dividing the backbone tree into disjoint subsets that are small enough for pplacer, thus producing a set of disjoint (but smaller) backbone trees. It then uses pplacer to place each query sequence into each of the smaller backbone trees; naturally, this also indicates a position within the original backbone tree. The set of placements can then be compared for likelihood scores, and the best placement returned. By design, pplacerDC matches the accuracy of pplacer on all datasets on which pplacer can run, and is easily parallelized. Most importantly, pplacerDC is more accurate than both EPA-ng and APPLES, and can place into 100,000-taxon trees. Thus, although pplacerDC is slower than APPLES, it provides accuracy advantages and excellent scalability, and represents an advance in phylogenetic placement methodology.

2 APPROACH

The input to many phylogenetic placement methods is a backbone tree T with leafset $\mathcal{L}$ and an alignment A of the sequences in $\mathcal{L} \cup \{x\}$, where x is a query sequence. Most phylogenetic placement methods also require some numeric parameters (e.g., branch lengths), and

BCB '21, August 1–4, 2021, Gainesville, FL, USA

ACM ISBN 978-1-4503-8450-6/21/08…$15.00
https://doi.org/10.1145/3459930.3469516

some require them in particular formats. In particular, pplacer and pplacerDC require numeric parameters that are produced using RAxML7 [14], and EPA-ng required numeric parameters produced by RAxML8 [15]. RAxML is a program using maximum likelihood to analyse phylogenies, and one of its features is the estimation of model parameters from an alignment. The output of each phylogenetic placement method is the backbone tree with the query sequence placed.

We have designed a divide-and-conquer version of pplacer, which we call pplacerDC. A schematic of the pipeline for pplacerDC is shown in Figure 1, and pseudocode for pplacerDC is shown in Algorithm 1. Note that the input to pplacerDC is nearly the same as the input for pplacer, with the addition of an option to specify the maximum size of the subtrees in the decomposition step.

The first step of pplacerDC is to decompose the backbone tree into subsets. To achieve this, pplacerDC uses the same decomposition as used in PASTA [9], which operates as follows: pplacerDC finds a centroid edge in the backbone tree (i.e., an edge whose removal separates the leaf set into two subtrees of approximately equal size). The decomposition is recursively applied to each subtree until each subset is small enough, which is determined by a parameter value selected by the user. (This centroid decomposition is also the same decomposition used in SATé-2 [7], TIPP [12], and SEPP [10]). For the pplacerDC setting, we ensure that the subsets are small enough that pplacer can successfully place query sequences into subtrees of that size. Given the prior trends about pplacer failures established in [1], pplacer reliably places on trees with 1000 sequences, but frequently fails for 5000-sequence trees.

After dividing the tree into disjoint subset trees, pplacerDC operates as follows. First, pplacerDC uses pplacer independently on each of the subtrees to determine the best query placement for that subtree. Each tentative query placement is then grafted onto the original backbone tree, yielding a candidate resultant tree and placement. Note that the input also contains an alignment that includes the sequences at the leaves of the backbone tree and the query sequence; hence, this procedure produces an alignment/tree pair for each of the subsets produced by the decomposition.

RAxML-ng [5] is then used to score each alignment/tree pair with respect to maximum likelihood. The placement position with the largest log-likelihood score is returned by pplacerDC. Note that each tree given to RAxML-ng to score contains the placement position for the added sequence and has all the GTRGAMMA numeric parameters (i.e., branch lengths, 4x4 substitution matrix, gamma shape parameter, stationary distribution for the nucleotides) already defined. Hence, RAxML-ng can score the tree without trying to optimize the numeric parameters; this simplifies the problem and enables RAxML-ng to be reasonably efficient. We use the following command for the RAxML-ng scoring:

```
raxml-ng –msa referenceAln –model GTR+G –tree
treeFile –threads threads –opt-branches off
–opt-model off –evaluate –nofiles –log result
```

We prototyped this approach using Python, with substantial reliance on the Dendropy library [16]. As Dendropy is not designed for large datasets and is instead focused on easy prototyping, our goal was to evaluate whether this approach provided good accuracy more than running time or memory usage, especially on large datasets. In order to be applicable in either a shared or distributed memory context, two versions of pplacerDC are implemented in Python. The first is tailored to running on shared memory machines using Python's native thread pool paradigm. The second is written using the message passing interface (MPI) standard bindings implemented in mpi4py and is tailored for distributed memory machines [4].

Algorithm 1: Divide-and-conquer pplacer

Result: T', tree T with query sequence q added
Input: Tree T on N sequences, the MSA of $N+1$ sequences, and query sequence q
// centroidDecomposition decomposes a tree into roughly equal size, disjoint parts until the trees are no larger than the prescribed size.;
// modifyTree adds sequence to a tree based on the sequence's location in the a subtree with the sequence added;
$\{T_1, \ldots, T_n\} \leftarrow$ centroidDecomposition(T, `maxSize`);
$\{S_1, \ldots, S_n\} \leftarrow 0$ // Score for each tree;
parallel for $i = 1, \ldots, n$ **do**
  // Place query sequence q into the subtree;
  $T'_i \leftarrow$ pplacer(T_i, q);
  // Add the location of the query sequence q to a copy of T;
  $T_{q_i} \leftarrow$ modifyTree(T, T'_i, q);
  // RAxMLScorer runs RAxML in fixed tree mode.;
  // The output score is the maximum likelihood found on the tree.;
  $S_i \leftarrow$ RAxMLScorer(T_{q_i});
endfor
// Do a maxLoc reduction for the tree;
$bestTreeIndex \leftarrow \text{argmax}_i(S_1, \ldots, S_n)$;
return $T'_{q_{bestTreeIndex}}$;

3 EXPERIMENTAL EVALUATION

3.1 Methods

To run pplacerDC, we specify the maximum number of leaves in each of the subtrees it uses to be 2500; this means that it decomposes the backbone trees by deleting centroid edges until each subtree has at most 2500 leaves. We compare pplacerDC to APPLES and EPA-ng; however, we do not compare pplacer to pplacerDC, because the two methods are equal when pplacer can run. We selected APPLES for an example of a scalable and fast phylogenetic placement method and EPA-ng as the leading alternative to pplacer for accuracy, while being more scalable than pplacer. Our goal is to achieve the scalability of APPLES but improve on the accuracy of EPA-ng.

3.2 Experiments

We use the same experimental protocol as [1] to evaluate phylogenetic placement methods for scalability and accuracy on large datasets. We use the RNASim-VS (variable size subsets of the RNASim dataset) with 1K to 100K sequences; these datasets were sampled

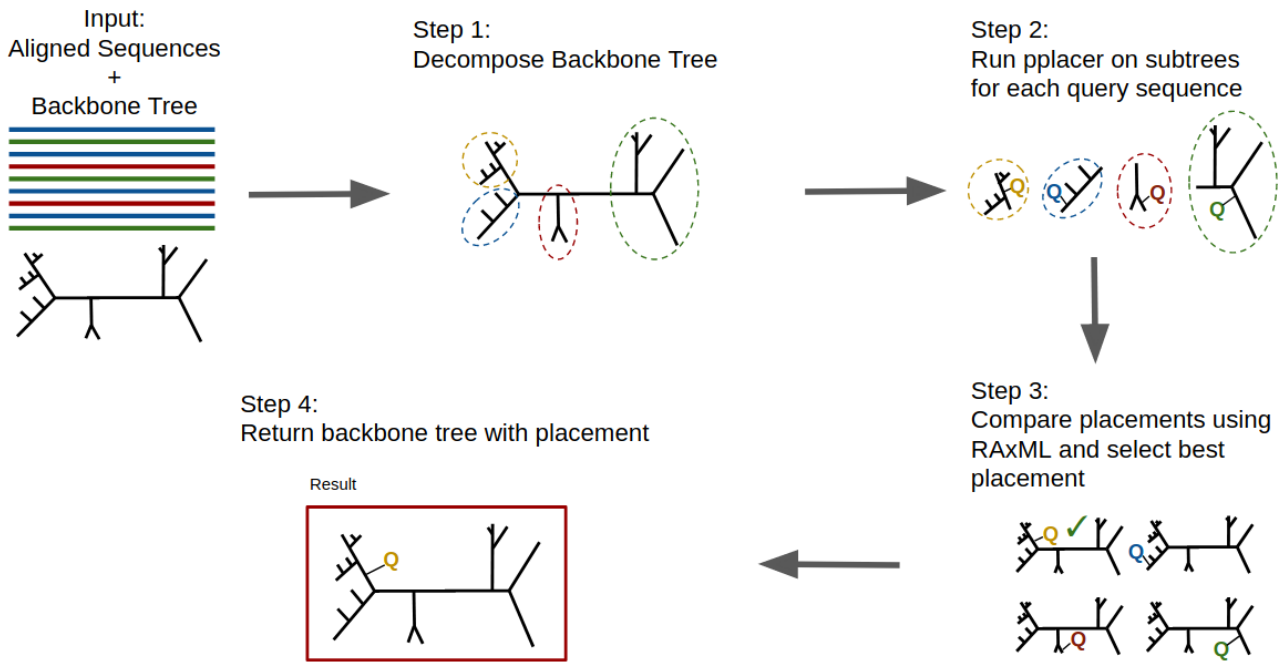

Figure 1: The pplacerDC pipeline. The input to pplacerDC is a set of aligned sequences, a backbone tree (with branch lengths), and a query sequence. The set of aligned sequences includes the sequences at the leaves of the backbone tree and the query sequence being placed into the tree. In Step 1, pplacerDC splits the tree into subtrees; in Step 2, it uses pplacer to place the query sequence into each of the subtrees; and in Step 3, it uses RAxML to compute the maximum likelihood score of these placements within the full backbone trees. The best of these placements (according to ML score) is then returned.

randomly from the RNASim data set described in [9]. Briefly, the RNASim simulation (described in detail in the supplementary materials for [9]) reflects the selective pressures that occur in order to maintain the secondary structure of the molecule. The fixation probability of a mutation (whether of indels or substitutions) is computed by a function of the folded free energy of the resulting RNA sequence. The RNASim simulation has varying rates across sites, which also reflects the properties of RNA sequence evolution.

Because the RNASim datasets are simulated, they have true trees and true sequence alignments, which allow us to exactly quantify error in estimated trees before and after placement. We have five replicates for each number of sequences, ranging among 1K, 5K, 10K, and 100K.

- Experiment 1: We compared different subtree sizes for pplacerDC to optimize the accuracy.
- Experiment 2: We compared pplacerDC to the leading alternative methods, APPLES and EPA-ng, with respect to accuracy of the placement.
- Experiment 3: We compared pplacerDC to APPLES and EPA-ng with respect to running time and scalability.

As in [1], we used a leave-one-out strategy for the tests. Out of the set of sequences in the tree, 200 sequences are selected as query sequences. For each of the query sequences, the leave-one-out strategy starts with the true tree, T, and removes the sequence from T, creating T'. Then, the placement software adds that query to T' to obtain the estimated placement.

3.3 Backbone trees

For the backbone trees, we use trees estimated on the true alignment. We use FastTree2 [13], a maximum likelihood heuristic, for these analyses. This selection is also the same as the experimental design used in [1].

Each of the phylogenetic placement methods requires the backbone tree to be provided with branch lengths, but are optimized for different branch length calculations. We follow the protocol described in [1] for the branch length estimation for APPLES and EPA-ng on the RNASim-VS datasets. For APPLES, we used the branch lengths estimated by FastTreeMP. For EPA-ng, we used branch lengths from RAxML version 8 and required the RAxML8 info file [15]. For pplacerDC, we used used branch lengths estimated by RAxML version 7.2.6, and provide these when we pass the subtrees to pplacer, which requires an info file generated by RAxML7. All estimated tree topologies and branch lengths we used are from [1], with the exception of the RAxML7 branch lengths for two of the 100,000 replicates. Due to multiple identical sequences, RAxML failed. RAxML was rerun after removing the duplicate sequences.

3.4 Error Assessment

We compared pplacerDC to pplacer, EPA-ng, and APPLES for accuracy, runtime, and memory usage. To enable a direct comparison to the results reported in [1], we use the same technique for quantifying placement error, called the "delta error", and which we now define. Each edge in a tree T with leafset $\mathcal{L}$ defines a bipartition on $\mathcal{L}$, and so each tree T can be defined by its set $B(T)$ of bipartitions. Given the true tree T^*, the False Negatives in an estimated tree are those bipartitions in the true tree that are missing from the estimated tree. Based on this definition of false negatives, the delta error is the increase in the number of false negatives produced by adding a query sequence into a backbone tree (and note that the delta error, denoted Δe, is always non-negative). We formalize this as follows:

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

where $\mathcal{L}$ denotes the leafset, P denotes the tree after placement, T^* denotes the true tree on $\mathcal{L} \cup \{q\}$, and $T^* \restriction_{\mathcal{L}}$ denotes the true tree restricted to $\mathcal{L}$. (Note that when the computed tree and the true tree are both fully resolved (i.e., when every internal node in the unrooted tree has degree 3) then the increase in the number of false negatives is the same as the increase in the number of false positives.)

Computational infrastructure. We performed the analysis on NCSA's Campus Cluster on Dell C8220 compute sleds. Each has two Intel E5-2670V2 (Ivy Bridge) processors. The Campus Cluster is a heterogeneous system, so that each sled has a minimum of 64GB RAM, and some have 128GB or 256GB RAM.

4 RESULTS

4.1 Experiment 1: Subtree Size

	1000	2000	2500	3000
pplacerDC	0.159	0.156	**0.146**	0.148

Table 1: Delta Error (Δe) for RNASim-VS with trees of 10,000 taxa where the maximum size of the subtrees varies. We show the average Δe, across all query sequences and replicates; the best result for each criteria is boldfaced. There are five replicate trees for the backbone size 10,000 placing 200 queries each.

In order to select a suitable maximum size for the subtrees used by pplacerDC, we ran pplacerDC with varying subtree sizes on trees of 10,000 taxa. The maximum subtree sizes tested were 1000, 2000, 2500, and 3000, and the results are shown in Table 1 and Figure 2.

The lowest average delta error occurs at maximum subset size 2500, but the differences between the highest and lowest average delta error among these tested values is not large. In addition, for each of the tested settings for the maximum subtree size, the majority of placements had 0% delta error, and even more are only one branch away from the optimal placement. In the case of maximum subtree sizes 1000 and 2000, there were one and two placements with much higher error, but these outlier values only changed the average delta error by a small amount.

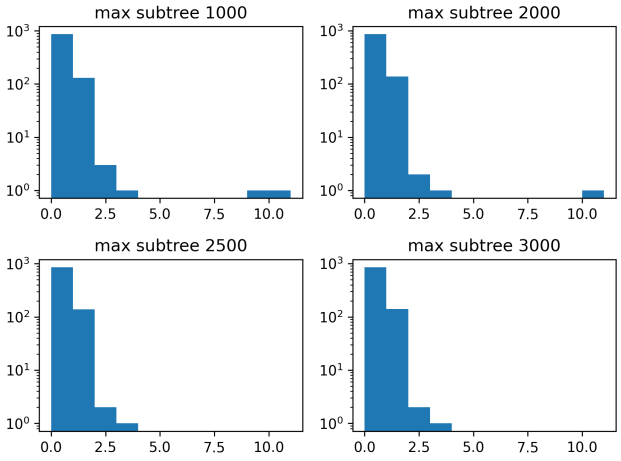

Figure 2: Delta error as a function of the maximum subtree size using pplacerDC on the RNASim-VS datasets. Error is averaged across single query placements, using an estimated backbone tree for 200 leave-one-out queries on each of five replicates on trees of 10,000 taxa.

For the experiments comparing pplacerDC to the other methods, we used subtrees of size 2500, as it minimized the average delta error.

4.2 Experiment 2: Evaluating accuracy

	n = 1000		n = 5000		n = 10000		n = 100000	
	%	Δe	%	Δe	%	Δe	%	Δe
pplacerDC	**81**	**0.20**	**86**	**0.15**	**86**	**0.15**	**91**	**0.15**
EPA-ng	73	0.30	78	0.24	79	0.22	-	-
APPLES	71	0.43	77	0.37	79	0.33	84	0.25

Table 2: Delta Error (Δe) for RNASim-VS, where the number of taxa (n) in the tree varies. We show the percentage of placements that produce $\Delta e = 0$ and the average Δe, across all query sequences and replicates; the best result for each model condition is boldfaced. There are five replicate trees for every backbone size n, and 200 queries for each replicate. Results are not shown for pplacer, since it is identical to pplacerDC on 1000-taxon backbone trees and fails on many large backbone trees. Dashes for EPA-ng for 100,000-taxon backbone trees are shown, since it fails on many placements on backbone trees of 100,000 leaves according to [1].

Results on the data sets with 1000 or more sequences are shown below in Figure 3. Although pplacer results are not shown, pplacer is identical to pplacerDC on 1000-taxon backbone trees, and so its performance on 1000-taxon backbones can be inferred from

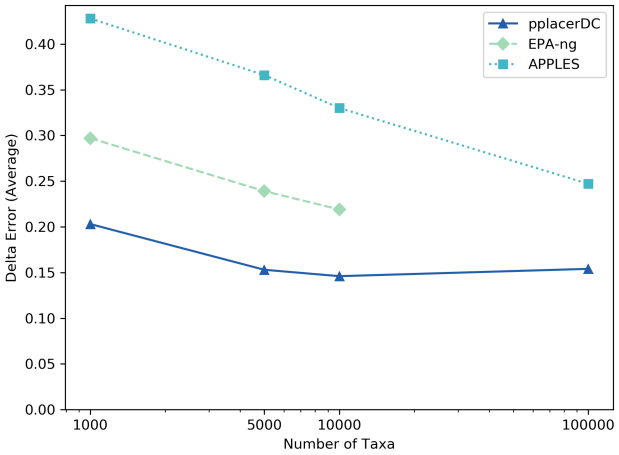

Figure 3: Delta error as a function of the size of the backbone tree on the RNASim-VS datasets. Error is averaged across single query placements, using an estimated backbone tree for 200 leave-one-out queries on each of five replicates (a total of 1000 queries for each backbone tree size). Balaban et al. [1] found that EPA-ng fails on some placements on trees of 50,000 or more taxa, so EPA-ng is omitted for the 100,000-taxon backbone trees.

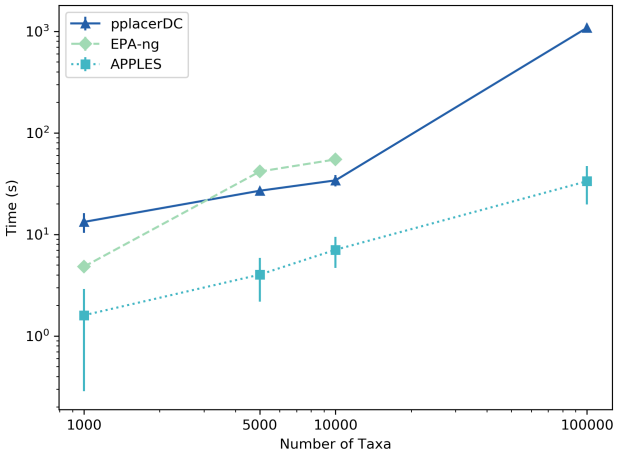

Figure 4: Runtime as a function of the backbone tree size, using 16 threads for trees of 1000-10,000 taxa and 3 threads for the trees of 100,000 taxa due to memory requirements. We report the average time to place a single query sequence, using an estimated backbone tree, for 200 leave-one-out queries on each of five replicates for a total of 1000 queries for each number of taxa. pplacer failed on many trees of 5000 taxa and so is not included for any trees larger than 1000 taxa. Balaban et al. [1] found that EPA-ng fails on some placements on trees of 50,000 or more taxa, so EPA-ng is omitted for the 100,000-taxon backbone trees.

the figure. However, we do not show pplacer results on the larger datasets, because it fails frequently on 5000-taxon backbone trees and does not run on larger datasets, as reported in [1] and also confirmed in this study.

For all backbone tree sizes, pplacerDC has the best accuracy, followed by EPA-ng, and then by APPLES. Furthermore, APPLES has more than twice the delta error of pplacerDC at the smaller backbone trees (up to 10,000 leaves) and still 1.6 times the error on the larger trees. We also observe that EPA-ng cannot complete some analyses on the 50,000-taxon backbone trees (hence results on 50,000-taxon backbone trees are not shown), but that both APPLES and pplacerDC complete on all the backbone trees. Interestingly, error rates drop as the size of the backbone trees increase from 1000 to 100,000, and this trend holds for both APPLES and pplacerDC.

We also show the percentage of times each phylogenetic placement method has $\Delta e = 0$ in Table 2. Here too we see pplacerDC improving relative to both EPA-ng and APPLES throughout this range of backbone tree sizes. Interestingly, APPLES and EPA-ng are close with respect to this criterion, and identical for the 10,000-taxon backbone trees. (Note that our Table 2 matches results for APPLES and EPA-ng reported in Table 3 of [1]).

4.3 Experiment 3: Evaluating computational scalability

Here we report the average (across query sequences) for the runtime and peak memory usage for each method. We report results for pplacer only on the 1000-taxon backbone trees, since it fails on many of the 5000-taxon backbone trees, and similarly we do not report results for EPA-ng for backbone trees larger than 10,000 leaves.

As seen in Figure 4, APPLES is by far the fastest method, with a substantial improvement over EPA-ng and pplacerDC. The comparison between EPA-ng reveals that EPA-ng is faster than both pplacer and pplacerDC at 1000-taxon backbone trees, and then slower than pplacerDC for 5000 and 10,000-taxon backbone trees; however, EPA-ng does not run on the largest backbone trees. Peak memory usage for the different methods is shown in Figure 5. EPA-ng has the highest peak memory usage, followed by pplacerDC, and then by APPLES.

A focused evaluation of the running time of each step is provided in Figure 6. The left subfigure shows the breakdown by step for 10,000-taxon backbone trees, and the right subfigure shows the breakdown for 100,000-taxon backbone trees. For the smaller backbone tree size (10,000-taxon backbone trees), the vast majority of the time is spent in Step 2 (pplacer analyses on subset trees), and the other steps all use about the same amount of time. On the largest backbone trees we examined (with 100,000 leaves), the total amount of time goes up, but the proportions also change. Specifically, now Step 1 becomes the smallest part, Step 2 uses significant time, and Step 3 uses about 2/3 of the entire time. Furthermore, both Steps 3a and Step 3b are computationally expensive, and more expensive than Step 2, and Step 3b is more expensive than Step 3a. Thus, the most expensive part of the pipeline on large datasets is Step 3b, where RAxML is used to compute the maximum likelihood score

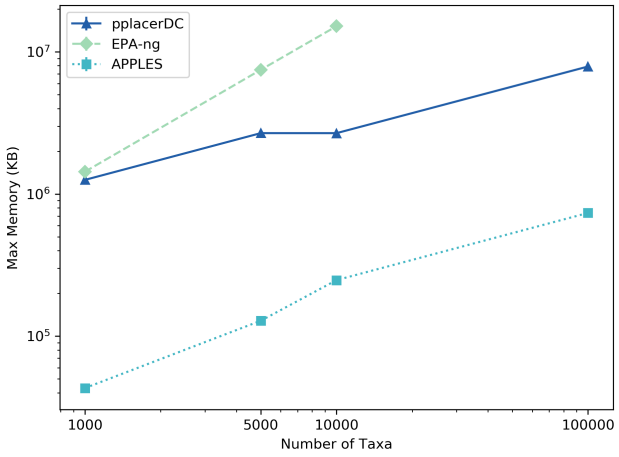

Figure 5: Maximum memory usage on 16 threads. We report average peak memory usage to place a single query sequence using an estimated backbone tree for 200 leave-one-out queries on each of five replicates, for a total of 1000 queries for each number of taxa. pplacer failed on all trees of 5000 taxa and so is not included for any trees larger than 1000 taxa. [1] found that EPA-ng fails on some placements on trees of 50,000 or more taxa, so EPA-ng is omitted for the 100,000-taxon backbone trees.

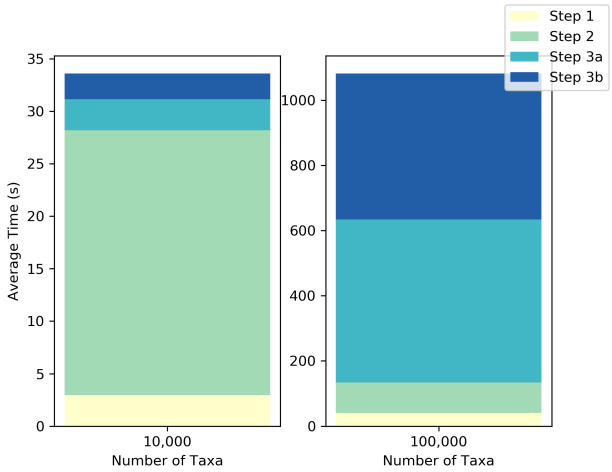

Figure 6: Runtime for pplacerDC by step (Left subfigure for 10,000-taxon backbone trees using 16 threads, Right subfigure for 100,000-taxon backbone trees using 3 threads). Step 1 is decomposing the backbone tree and preparing input FASTA files for pplacer. Step 2 is running pplacer on each of the subtrees and placing the query in each subtree. Step 3a modifies the main tree to include the query sequence of the predicted placement, and Step 3b uses RAxML-ng to compute the likelihood score (based on the given branch lengths and substitution rate matrix) of the resultant trees. We report the average time to place a single query sequence, using an estimated backbone tree, for 200 leave-one-out queries on each of five replicates for a total of 1000 queries for each number of taxa. On 10,000-taxon backbone trees, Step 2 (pplacer analyses) use the majority of the time. However, on 100,000-taxon backbone trees, Step 3 uses the majority of the runtime, with Step 3b using more time than Step 3a. Times shown here are also provided in Table 3 in the appendix.

of the competing trees, but Step 3a, which relies on Dendropy, is also computationally intensive.

5 DISCUSSION

We followed the protocol used in [1], using RNASim-VS (simulated datasets with backbone trees having 1000 to 100,000 sequences) in order to evaluate pplacer DC in comparison to EPA-ng, APPLES, and pplacer with respect to placement accuracy and computational resource (time and peak memory) usage. Our study confirms results obtained in [1] for the other methods, and shows that pplacerDC improves on both APPLES and EPA-ng for accuracy, and is able to analyze datasets with backbone trees having up to 100,000 leaves.

The running time and peak memory usage also shows that pplacerDC is more efficient than EPA-ng, but far less efficient than APPLES. The pplacerDC implementation we used, however, is based on Python and relies on Dendropy, and is thus research grade code. Clearly a more careful implementation will be needed for pplacerDC to be able to scale to larger datasets.

A consideration of the computational effort reveals different issues for the different steps. In the first step, we divide the backbone tree into subsets; this is fast and uses very little memory. In the second step, we use pplacer to place each query sequence into each subset tree; each of these placements will use (roughly) the same amount of time, since each subset tree is of bounded size. Also, this step is embarrassingly parallelizable (since each subtree can be analyzed independently). Hence, given sufficient infrastructure, this is feasible even for very large backbone trees. The last step of the pplacerDC pipeline, however, requires that we compare different placements in the backbone tree with respect to the maximum likelihood criterion, using RAxML. Note that this is a potentially large number of different trees that need to be compared (i.e., one for every subset tree computed in the decomposition). Therefore, Step 1 should be very fast and scale linearly with the number of leaves in the backbone; Step 2 will require significant time, but parallelism should allow this to scale linearly with the number of leaves in the backbone; and Step 3 will be fast on small to moderate backbone trees, but then possibly become infeasible on very large backbone trees because of running time and memory issues involved in using RAxML to score the different trees. Furthermore, there is even a potential for reduced accuracy on the very largest trees, if numerical issues in scoring large trees arise.

This prediction is consistent with results shown in Figure 6, which shows the running time breakdown by step, for 10,000-taxon backbone trees (left subfigure) and then for 100,000-taxon backbone trees (right subfigure). For the smaller backbone tree size, the majority of the effort is used by Step 2 (pplacer analyses on subset trees), but for the larger backbone trees, the majority of the effort

is spent in Step 3, with Step 3b (which is where RAxML is used to score the different trees) the largest part of the effort.

The differences between the running times on 10,000- and 100,000-taxon backbone trees are due to a combination of changes in the number of threads used, the number of placements proposed and assessed, and the cost of a single placement in each step. Between 10,000 taxa and 100,000 taxa, Step 2's time increases by a combination of using 3 threads instead of 16 threads (to avoid out-of-memory errors in Step 3) and the number of subtrees increasing by a factor of 10. As Step 2 uses pplacer to place into trees of constant size regardless of the overall number of taxa, the time required for Step 2 is a linear function of the number of leaves. However, both Step 3a and 3b (and Step 3b in particular) increase by a larger factor when the number of leaves increases from 10,000 to 100,000, and there is a straightforward explanation. Recall that Step 3b requires the use of RAxML to score each of the trees on the full set of leaves; the number of these trees is the total number of placements, which is the total number of subtrees; hence, the number of these evaluations scales linearly with the number of leaves. However, the cost of using RAxML to score a tree is linear in the input size (the product of the number of leaves and number of sites), so that Step 3b scales quadratically with the number of leaves rather than linearly. This is why Step 3b contributes a larger proportion of the running time as the number of leaves increases.

6 CONCLUSION

Phylogenetic placement is a basic algorithmic step in several bioinformatics analyses, including the construction of very large trees (i.e., by adding newly discovered sequences into existing phylogenies) and metagenomic taxon identification and abundance profiling. Because these analyses are based on large backbone trees, potentially with hundreds of thousands of leaves, phylogenetic placement methods that can scale to large datasets are needed.

To date, the most accurate phylogenetic placement methods have been based on likelihood calculations, with pplacer the most successful for accuracy but the least successful for scalability. The next most accurate method, EPA-ng, is somewhat more scalable than pplacer, but it too fails on backbone trees with 50,000 sequences (and may have a lower threshold than this). Alternative approaches based on distances, such as APPLES, have been developed, which can scale to large datasets, but these have displayed lower accuracy than the likelihood-based methods. Thus, prior to this study, the only scalable methods were based on distances, with APPLES being perhaps the current most accurate such method.

Here we presented pplacerDC, which uses a divide-and-conquer approach to extend pplacer to large backbone trees. Our study establishes that this approach provides excellent accuracy, matching pplacer where pplacer can run, and superior to that of EPA-ng. In consequence, pplacerDC is also more accurate than APPLES. Our study showed that pplacerDC can run on backbone trees with up to 100,000 leaves, thus substantially exceeding the limit of a few thousand for pplacer (which fails on many backbone trees with 5000 or more leaves) and EPA-ng (which fails on backbone trees with 50,000 or more leaves), thus establishing pplacerDC as the first likelihood-based phylogenetic placement method that can run on datasets of this size.

Thus, the divide-and-conquer approach we used in pplacerDC addresses a limitation of existing maximum likelihood based placement methods: the limitation to small backbone trees. However, pplacerDC is much slower than APPLES, and we did not evaluate pplacerDC on the 200,000-taxon backbone trees. Thus, while pplacerDC provides improved accuracy in the range in which we examined it, it does not match the speed of APPLES and may not match it for scalability to ultra-large backbone trees.

This study leaves much room for future work. First, our implementation is in Python, and so improving the implementation to reduce memory usage and running time is needed. In particular, reducing the reliance on Dendropy (which is designed for fast prototyping but not for efficiency on large datasets) would be a good start towards improving the speed, and would directly address the running time usage for Step 3a. Equally importantly, as discussed earlier, Step 3b in pplacerDC has a reliance on maximum likelihood calculations on the entire backbone tree, which may make it infeasible for ultra-large backbone trees. As we have not evaluated pplacerDC on the largest backbone trees with 200,000 leaves, we do not yet know if this will be an issue, and so this determination is needed. However, in general, we hypothesize that for sufficiently large backbone trees, numerics issues will need to be addressed for pplacerDC to maintain its high accuracy. (We add, however, that this is a general problem for likelihood-based estimation of phylogenies, and not restricted to phylogenetic placement methods.)

Secondly, in using the same evaluation protocol as [1], we only explored phylogenetic placement methods when given full-length query sequences, rather than short query sequences. This is appropriate for the setting where the placement method is used to characterize a newly discovered full length sequence, or to add new sequences into an existing large phylogeny. However, phylogenetic placement is also used in characterizing metagenomic reads, which will typically be short and have sequencing error, which is a different setting. Here we note that [6] found RAPPAS to be less accurate than maximum likelihood placement methods (EPA-ng and pplacer) in placing query sequences that have sequencing error. Thus, we would predict that likelihood-based placement methods should generally be more accurate in placing reads (short query sequences with sequencing error) into backbone trees than alternative approaches that are based on distances (especially if not provided with accurately estimated multiple sequence alignments), but future work is needed to confirm this.

Thirdly, our study focused on enabling highly accurate likelihood-based analyses on large backbone trees, rather than on running time (which APPLES excels in) or scalability with the number of query sequences (which EPA-ng is designed for). Future work is needed to explore these aspects, and pplacerDC will need to be re-implemented for scalability to become competitive in these respects. In particular, pplacerDC was implemented using Dendropy [16], a library which is not well suited for large trees. The next implementation will take advantage of TreeSwift [11], which is a library for tree estimation algorithms designed specifically for large trees.

We close with some basic problems that are not specific to pplacerDC. A very basic question is why pplacer is more accurate than EPA-ng on the datasets on which it can run, since they both use the same optimization criterion and presumably similar heuristics. It is

also important to understand why pplacer has reduced accuracy towards the upper end of its range of applicability, and why it is is limited to small backbone trees. We hypothesize that pplacer may have numerical issues when placing into large backbone trees, and that an improved implementation of pplacer may produce large benefits.

We note also that the divide-and-conquer pipeline we presented could be used with any phylogenetic placement method, and is not restricted to pplacer. Hence, if new phylogenetic placement methods are developed that improve on the accuracy of pplacer, they could be used instead of pplacer in the pplacerDC divide-and-conquer pipeline. Finally, pplacerDC is just one attempt at divide-and-conquer, and other approaches may also lead to improvements. For example, a related divide-and-conquer technique that is more restricted in that it limits placement to a selected small subset of the backbone tree has just been developed and also shows promise [17]. The importance of phylogenetic placement methods suggests that future work should investigate additional divide-and-conquer strategies to enable highly accurate placement and advance large-scale phylogeny estimation.

APPENDIX

This appendix provides details about materials and methods used in this study.

APPLES

The version of **APPLES** used is v1.2.0. The source code can be accessed at
https://github.com/balabanmetin/apples/releases.

The command used to run `APPLES` is

```
run_apples.py -T 16 -t backbone_app.tre
  -q query.fa -s aln_dna.fa -o apples.jplace
```

RAxML-NG

The version of **RAxML-NG** used by pplacerDC is v1.0.1. A pre-built binary can be accessed at
https://github.com/amkozlov/raxml-ng/releases/tag/1.0.1.

The source code for RAxML-NG can be accessed at
https://github.com/amkozlov/raxml-ng.git

pplacer

The version of **pplacer** used is the pre-built binary for v1.1.alpha19, which can be accessed at
https://github.com/matsen/pplacer/releases/tag/v1.1.alpha19.

The command used to run `pplacer` is

```
pplacer -m GTR -s RAxML_info.REF
  -t backbone_pp.tre -o pplacer.jplace aln_dna.fa
```

EPA-ng

The version of **EPA-ng** used is v0.3.8. The source code can be accessed at
https://github.com/Pbdas/epa-ng.git

The command used to run `epa-ng` is

```
epa-ng –ref-msa aln_dna.fa –tree backbone_epa.tre –query
query.fa –model RAxML_info.REF8 –redo
```

pplacer-DC

pplacerDC uses **dendropy** for tree modification [16]. Dendropy is known to not work well on large trees, so while it was useful for this version of pplacerDC, future versions will use methods better suited to large trees. This should particularly have an impact on the time required for step 3a. (See figure 6.)

The source code for **pplacerDC** can be accessed at
https://github.com/kodingkoning/pplacerDC.git

The command to run `pplacerDC` is `pplacerDC.py -m 2500 -s RAxML_info.REF -t input.tre -q $query -r aln_dna.fa -o pplacerDC.tre`

Datasets

The **RNASim-VS** datasets may be accessed through the supplementary materials of [1].

Results from the **RNASim-VS** dataset and the scripts used to run the analyses may be accessed with the pplacerDC code on GitHub.

Other Software Commands

nw_prune:

The commands to use `nw_prune` to create backbone trees are:

```
nw_prune RAxML_result.REF $query
  &> backbone_pp.tre
nw_prune true_me.fasttree $query
  &> backbone_app.tre
nw_prune RAxML_result.REF8 $query
  &> backbone_epa.tre
```

Each of the full trees were included with the RNASim-VS data.

Processing jplace outputs:

The command to extract the tree from the jplace output of `pplacer` or `APPLES` is:

```
guppy tog -o output.tre output.jplace
```

Evaluating placement error:

To evaluate the topological error of a tree, either `measure_all.sh` or `treecompare.py` from the `common` directory of the `pplacerDC` repository can be used.

`measure_all.sh` also requires compareTrees.missingBranch from the repository found at https://github.com/smirarab/global
`measure_all.sh` evaluates pplacer, pplacerDC, APPLES, and EPA-ng in the given directory with the following command
`./measure_all.sh $dir $size`
where size exempts evaluating EPA-ng and pplacer in cases where the tree size is too large for them to run.

`treecompare.py` can be run using the command

```
python3 treecompare.py true_topo.tree output.tre
backbone.tre backbone_true.tre
```

where

- `true_topo.tre` is the full true tree including the query,
- `output.tre` is the resulting tree from any query placement,
- `backbone.tre` is the tree that was given as input to the placement method, and
- `backbone_true.tre` is `true_topo.tre` with the query pruned from it.

The output of pplacerDC is a tree, not a jplace file, so its output may be used directly in place of output.tre without using guppy.

For an example using pplacer, these can be generated with nw_prune and guppy using:

```
guppy tog -o pplacer.tre pplacer.jplace
nw_prune RAxML_result.REF $query
  &> backbone_pp.tre
nw_prune true_topo.tree $query
  &> backbone_true.tre
```

and for the other three approaches, the appropriate trees should be selected in place of `pplacer.jplace` and `RAxML_result.REF`.

Taxa	10,000	100,000
Step 1	2.94	40.69
Step 2	25.25	93.09
Step 3a	2.97	499.91
Step 3b	2.46	448.67

Table 3: Times in seconds for each step of pplacerDC for two different tree sizes (results here are also shown in Figure 6).

ACKNOWLEDGEMENTS

We thank Siavash Mirarab and Metin Balaban for help in using APPLES. We thank Eleanor Wedell for help running RAxML for the trees of 100,000 taxa which had duplicate sequences. This work began as a course project by EK and MP for the Fall 2020 course CS 581: Algorithmic Genomic Biology, at the University of Illinois, taught by TW.

FUNDING

This work was supported in part by the US National Science Foundation through grant ABI-1458652 to TW. This study was performed on the Illinois Campus Cluster, a resource operated and financially supported by UIUC in conjunction with the National Center for Supercomputing Applications.

REFERENCES

[1] Metin Balaban, Shahab Sarmashghi, and Siavash Mirarab. 2020. APPLES: Scalable distance-based phylogenetic placement with or without alignments. *Systematic Biology* 69, 3 (May 2020), 566–578. DOI: https://doi.org/10.1093/sysbio/syz063.

[2] Pierre Barbera, Alexey M. Kozlov, Lucas Czech, Benoit Morel, Diego Darriba, Tomáš Flouri, and Alexandros Stamatakis. 2018. EPA-ng: Massively Parallel Evolutionary Placement of Genetic Sequences. *Systematic Biology* 68, 2 (09 2018), 365–369. DOI: https://doi.org/10.1093/sysbio/syy054.

[3] Matthias Blanke and Burkhard Morgenstern. 2020. Phylogenetic placement of short reads without sequence alignment. *bioRxiv* (2020). DOI: https://doi.org/10.1101/2020.10.19.344986.

[4] Lisandro D. Dalcin, Rodrigo R. Paz, Pablo A. Kler, and Alejandro Cosimo. 2011. Parallel distributed computing using Python. *Advances in Water Resources* 34, 9 (2011), 1124 – 1139. http://www.sciencedirect.com/science/article/pii/S0309170811000777 DOI: https://doi.org/10.1016/j.advwatres.2011.04.013.

[5] Alexey M. Kozlov, Diego Darriba, Tomáš Flouri, Benoit Morel, and Alexandros Stamatakis. 2019. RAxML-NG: a fast, scalable and user-friendly tool for maximum likelihood phylogenetic inference. *Bioinformatics* 35, 21 (05 2019), 4453–4455. DOI: https://doi.org/10.1093/bioinformatics/btz305.

[6] Benjamin Linard, Krister Swenson, and Fabio Pardi. 2019. Rapid alignment-free phylogenetic identification of metagenomic sequences. *Bioinformatics* 35, 18 (2019), 3303–3312. https://doi.org/10.1093/bioinformatics/btz068

[7] Kevin Liu, Tandy J. Warnow, Mark T. Holder, Serita M. Nelesen, Jiaye Yu, Alexandros P. Stamatakis, and C. Randal Linder. 2012. SATé-II: very fast and accurate simultaneous estimation of multiple sequence alignments and phylogenetic trees. *Systematic Biology* 61, 1 (2012), 90. DOI: https://doi.org/10.1093/sysbio/syr095.

[8] Frederick A. Matsen, Robin B. Kodner, and E. Virginia Armbrust. 2010. pplacer: linear time maximum-likelihood and Bayesian phylogenetic placement of sequences onto a fixed reference tree. *BMC Bioinformatics* 11, 1 (Oct. 2010), 538. DOI: https://doi.org/10.1186/1471-2105-11-538.

[9] Siavash Mirarab, Nam-phuong Nguyen, Sheng Guo, Li-San Wang, Junhyong Kim, and Tandy Warnow. 2015. PASTA: Ultra-Large Multiple Sequence Alignment for Nucleotide and Amino-Acid Sequences. *Journal of Computational Biology* 22 (2015), 377–386. DOI: https://doi.org/10.1089/cmb.2014.0156.

[10] Siavash Mirarab, Nam-phuong Nguyen, and Tandy Warnow. 2012. SEPP: SATé-enabled phylogenetic placement. In *Biocomputing 2012*. World Scientific, 247–258. https://doi.org/10.1142/9789814366496_0024 DOI: https://doi.org/https://doi.org/10.1142/9789814366496_0024.

[11] Niema Moshiri. 2020. TreeSwift: A massively scalable Python tree package. *SoftwareX* 11 (2020), 100436. https://doi.org/10.1016/j.softx.2020.100436 DOI: https://doi.org/10.1016/j.softx.2020.100436.

[12] Nam-phuong Nguyen, Siavash Mirarab, Bo Liu, Mihai Pop, and Tandy Warnow. 2014. TIPP: taxonomic identification and phylogenetic profiling. *Bioinformatics* 30, 24 (Dec. 2014), 3548–3555. DOI: https://doi.org/10.1093/bioinformatics/btu721.

[13] Morgan N. Price, Paramvir S. Dehal, and Adam P. Arkin. 2010. FastTree 2 – Approximately maximum-likelihood trees for large alignments. *PLOS ONE* 5, 3 (03 2010), 1–10. DOI: https://doi.org/10.1371/journal.pone.0009490.

[14] Alexandros Stamatakis. 2006. RAxML-VI-HPC: maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models. *Bioinformatics* 22, 21 (08 2006), 2688–2690. DOI: https://doi.org/10.1093/bioinformatics/btl446.

[15] Alexandros Stamatakis. 2014. RAxML version 8: a tool for phylogenetic analysis and post-analysis of large phylogenies. *Bioinformatics* 30, 9 (01 2014), 1312–1313. DOI: https://doi.org/10.1093/bioinformatics/btu033.

[16] Jeet Sukumaran and Mark T. Holder. 2010. DendroPy: a Python library for phylogenetic computing. *Bioinformatics* 26, 12 (04 2010), 1569–1571. DOI: https://doi.org/10.1093/bioinformatics/btq228.

[17] Eleanor Wedell, Yirong Cai, and Tandy Warnow. 2021. Scalable and Accurate Phylogenetic Placement Using pplacer-XR. In *International Conference on Algorithms for Computational Biology*. Springer, 94–105.