

© 2022 Gillian Chu

PERFORMING PHYLOGENETIC PLACEMENT AT SCALE

BY

GILLIAN CHU

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Ecology, Evolution and Conservation Biology
in the Graduate College of the
University of Illinois Urbana-Champaign, 2022

Master's Committee:

Professor Tandy Warnow, Chair and Director of Research
Associate Professor James O'Dwyer
Associate Professor Katy Heath
Dr. Kevin Johnson

ABSTRACT

Phylogenetic placement is the problem of placing “query” sequences into an existing tree (called a “backbone tree”) whose leaves are aligned sequences, and has applications in updating large trees as well as in microbiome analysis. This is a study of phylogenetic placement pipelines which use extended alignment methods to first align the query sequence to the backbone alignment, before leveraging this information to place the query sequence into the tree using a placement method. We show that using methods specifically targeted for fragmentary sequences lead to higher accuracy, and we present pplacer-SCAMPP-taxtastic, which leverages several scalable techniques to extend the scalability of likelihood-based placement methods to ultra-large backbone trees (e.g., 200,000 sequences) with high accuracy.

ACKNOWLEDGEMENTS

This research was supported by the US National Science Foundation grant 2006069 (to TW) and by support from the National Science Foundation Graduate Research Fellowship Program (to GC). This thesis contains content that has been published by Gillian Chu and Professor Tandy Warnow on bioRxiv [1].

I would like to thank my advisor, Dr. Tandy Warnow, for her invaluable guidance and feedback, and without whom this thesis would not exist. I am constantly inspired by her energetic dedication to her craft and to her students, her attention to detail and her passion for her work. Her genius for guiding her students through seemingly impossible tasks is something I will ponder for a long time. I have thoroughly enjoyed working with her and I look forward to finding new ways in which her advice is indispensable to me in the future.

I would also like to thank my lab mates: Eleanor Wedell, Minhyuk Park, Chengze Shen, Baqiao Liu, James Willson, Yasamin Tabatabaee, and Dr. Paul Zaharias. I have learned far more from our discussions than I could otherwise have learned on my own.

Finally, I would like to thank my friends and family, particularly Rosalind Chu, Vinay Koshy and Callie the Cat, for their unyielding support and fantastic senses of humor in good and bad times alike.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION.....	1
CHAPTER 2: EXPERIMENT DESIGN.....	5
CHAPTER 3: SMALL BACKBONE ALIGNMENTS	15
CHAPTER 4: LARGE BACKBONE ALIGNMENTS.....	35
CHAPTER 5: PLACEMENT METHODS ¹	44
CHAPTER 6: CONCLUSIONS.....	57
REFERENCES.....	60
APPENDIX A	64

CHAPTER 1: INTRODUCTION

Phylogenetic placement is the problem of placing sequences (called “queries”) into an existing phylogeny (called a “backbone tree”). Generally, we begin with a set of reference sequences, a corresponding backbone tree on those reference sequences, and a set of homologous but unaligned query sequences.

Phylogenetic placement addresses the core problem of quantitatively relating a query sequence to existing backbone sequences. The end goal of this pipeline is to output an assignment of each query sequence to an edge in the backbone phylogeny. In this thesis, we are interested in scalable phylogenetic placement, where scalable refers to placing queries into large trees ($> 5,000$ sequences) in a computationally feasible manner.

Although placing query sequences into existing alignments and trees is not as accurate as re-estimating the alignment and tree from scratch [2], re-estimating alignments and trees are each very computationally intensive processes [2], especially when working with large numbers of backbone sequences. In this use case, there is variance in the quality of the new gene sequences to be aligned; these can be incompletely assembled (and be fragmentary as a result), or be well-assembled (and full-length) sequences. In such cases of large datasets, re-estimating entire alignments and trees is often entirely computationally infeasible. Thus, in these cases, placing query sequences into existing alignments and trees is preferable.

Many downstream applications benefit from phylogenetic placement, including taxonomic identification [3], abundance profiling [4] and incrementally building ultra-large trees [5]. Phylogenetic placement has most often been studied within the context of placing gene sequences into large existing gene trees [6] [5] [7]. There has been extensive literature on the benefits of taxon sampling for improving phylogenetic tree error [8] [9] [10] (or see [11] for a more nuanced perspective).

Another promising application of scalable phylogenetic placement is that of studying microbial communities via abundance profiling (i.e., the profiling of a community at taxonomic levels) and taxonomic identification methods [3]. The phylogenetic-placement-based approach to abundance profiling is leveraged in best-in-class methods like TIPP [3] and its improvement TIPP2 [4], both of which use a reference database of marker genes (single copy and universal genes) in order to characterize new sequencing reads (which are

fragmentary). In this case, however, phylogenetic placement is used to place gene sequences into taxonomies rather than gene trees [3].

TIPP2's improvement on TIPP can be entirely attributed to the denser sampling in each marker gene sequence collection [4]. Where the original TIPP used 30 marker genes with an average of 1,300 sequences per gene [3], TIPP2 updated this set to contain 39 bacterial and archaeal marker genes, with an average of 4,300 sequences per gene [4]. Therefore, one promising hypothesis that might further improve TIPP2's abundance profiling accuracy consists of enabling even larger backbone trees to be used in these pipelines.

Phylogenetic placement pipelines can be broadly classified into two categories: alignment-based and alignment-free. In the alignment-based scenario, we begin with an input backbone alignment, a corresponding backbone tree, and a set of unaligned query sequences. Phylogenetic placement can then be understood as two steps. In Step 1, we add each unaligned query sequence to the backbone alignment (independently of all other query sequences) to produce an **extended alignment** (containing the backbone alignment and the query sequence). This is done using an **extended alignment method**. In Step 2, we use a **placement method** to insert the query sequence into the backbone tree using Step 1's extended alignment. Placement methods that do not require this extended alignment step have also been developed, and are referred to as alignment-free placement methods [12] [13] [5].

Alignment-free methods are applicable when a meaningful alignment is not possible on the reference and/or query sequences. One example is that of genome skims, where the available reads cannot be meaningfully aligned, and so only alignment-free methods can be used [14]. When both approaches are possible, the advantage of alignment-free methods is the ability to run on many backbone sequences in a fast and memoryefficient fashion [12] [5]. However, several studies (e.g., the studies exploring RAPPAS [13], APPLES-2 [15] and App-Spam [12]) have noted that alignment-based methods such as pplacer are consistently more accurate than alignment-free methods. In this thesis, we are particularly concerned with increasing the scalability and accuracy phylogenetic placement pipelines, so we limit our discussion to the alignment-based phylogenetic placement pipelines.

The accuracy of alignment-based placement is dependent on the quality of the Step 1 extended alignment. The SEPP study [6], which compared the two extended alignment methods HMMalign [16] and SEPP [6], found

that the choice of extended alignment method had a major impact on the phylogenetic placement accuracy. SEPP was improved in the form of UPP, which has an improved way of handling fragmentary sequences, and which we will discuss further in this thesis [17].

Some alignment methods that were developed for other problems (some MAFFT flags [18] and UPP [17]) can also be used as Step 1 extended alignment methods, but they have not yet been evaluated for their effect on phylogenetic placement accuracy. Consequently, a more thorough investigation of the effect that extended alignment methods have on phylogenetic placement accuracy is in order.

Understanding how Step 2 placement methods affect phylogenetic placement accuracy is also crucial. The current most accurate placement method is pplacer, which takes a maximum likelihood approach, and which requires the query sequences to already be aligned to the backbone sequences [7] [19]. Unfortunately, pplacer has been shown to fail to run (produces illegal outputs) on some datasets larger than 10,000 sequences [7] [15]. Here, we explore possible explanations and solutions to scale phylogenetic placement.

A recent paper also explored phylogenetic placement in the context of placing gene sequences into species trees using the method DEPP [20]. However, as this paper tackles a different (although related) problem, we do not discuss or compare to DEPP here [20].

In this thesis, we address the aforementioned problems and provide the user with several recommendations to improve the pipeline of phylogenetic placement. Chapter 3 compares several extended alignment methods on the task of aligning a fragmentary query sequence to “small” backbone alignments, both in the case of using the true reference backbone alignment as well as an estimated backbone alignment. This chapter is related to the biological application of placing gene sequences into large gene trees as well as metagenomic applications, as we study extended alignment on fragmentary sequences. However, we should note that this is not a perfect study of such an application, as we do not simulate any sequencing error.

Chapter 4 evaluates how these extended alignment methods scale to “larger” datasets. This chapter is similarly related to metagenomic applications (with the same caveats), as we study scalable extended alignment on fragmentary sequences. Finally, Chapter 5 compares existing techniques for scaling placement methods, and combines techniques to propose a new approach for scalable phylogenetic placement. This

chapter is relevant to both applications of metagenomic analyses and placing into large gene trees, as we work with both fragmentary and full-length sequences.

CHAPTER 2: EXPERIMENT DESIGN

Several of the datasets and methods evaluated in this thesis are shared across chapters, so we begin by summarizing those here. This chapter is not a comprehensive list of all datasets and methods.

2.1 Shared Datasets

Phylogenetic placement is often used to place sequencing reads (e.g., in the case of microbiome analysis with TIPP2, where they placed two kinds of sequencing reads Roche and Illumina) [3]. Between the two, Illumina sequencing reads are more fragmentary, ranging from 150 basepairs to 250 basepairs. This informed the fragmentary protocols we use within this thesis. However, we should note the limitation in modeling the Illumina sequencing reads here; we have not introduced any sequencing error.

Fragmentary Protocol A

For the chapters concerning extended alignment methods, we applied two levels of fragmentation to each dataset: “low fragmentation” (LF), indicated that 250 sequences were sampled and made fragmentary. On average each fragment has length 50% of the original median sequence length. The second level was “high fragmentation” (HF) which indicated that 500 sequences were sampled and made fragmentary. On average each fragment has length 25% of the original median sequence length.

This protocol was applied to the ROSE and RNASim-VS datasets in Chapters 3 and 4. We report details on fragment lengths in each respective chapter.

2.1.1 Dataset Characteristics

In this subsection, we discuss important characteristics of each shared dataset. We preface the description of the datasets with some general comments about dataset selection.

Within this thesis we are interested in studying the accuracy and scalability of extended alignment methods and placement methods on extremely large datasets. As a result, the choice of each dataset was twofold: first, we picked datasets that were used in other studies of phylogenetic placement so that we could benchmark methods appropriately, and secondly, there is a limited selection of extremely large, simulated datasets within the literature. Large biological datasets were not evaluated on this thesis. There are two main reasons for this. First, reference alignments on real data tend to be on sequences that are extremely conserved

so that they can all be aligned well, which mean that they do not challenge the different methods, nor do they accurately capture some of the use cases where we want to use phylogenetic placement pipelines (e.g., novel sequences). Secondly, real datasets mean that there is no “true” tree. As such, even the published “reference” trees on real datasets are estimated trees containing estimation errors. To evaluate phylogenetic placement accuracy with respect to an estimated tree containing estimation errors is ill-defined, since the placement of a query sequence can be very affected by small errors in the estimated tree. Evaluating different placement methods on cases where the estimated tree may not contain the true placement location for a given query sequence does not tackle the problem we are trying to answer. Thus, we leave this question open for future work.

For each dataset, we report the normalized average p-distance or Hamming distance. The p-distance between two aligned sequences is the number of positions in which two sequences are different and neither is gapped, so we normalize these values by dividing by the total number of positions where neither is gapped. We calculate the alignment statistics in this section using the AlignmentStatistics scripts written by Mike Nute, which can be found online at [https://wiki.illinois.edu/wiki/display/warnowlab/Get+alignment+ statistics](https://wiki.illinois.edu/wiki/display/warnowlab/Get+alignment+statistics) or https://www.dropbox.com/s/wqt8hpvqkdjge04/alignment_stats.zip?dl=0.

ROSE Datasets

We used the published ROSE sequence datasets [21] to explore different rates of evolution. These datasets consist of sequences simulated using the ROSE simulator [22] down each nonultrametric birth-death model tree (generated using r8s v1.7 [23]). The ROSE datasets are small datasets with 1,000 sequences each, simulated with indels and substitutions under the GTR+GAMMA+Gap model [24]. This means that substitutions were drawn from the GTR+GAMMA model, and gap probabilities scaled with the substitution probabilities.

We explored three gap lengths, where “S” indicates short gap lengths, “M” indicates medium gap lengths, and “L” indicates large gap lengths [21]. For each gap length, we explored three different model conditions, where 1 indicates the highest rate of evolution, and 5 indicates the lowest rate of evolution [21]. The rates of evolution are also reflected in the average proportion of sites at which two sequences are different, which is also known as p-distance (i.e., normalized Hamming distance).

It is well-known that alignment estimation is more challenging on datasets with greater evolutionary diameter (higher p-distance), and that the alignment error also affects the subsequent tree estimation error [25]. The ROSE datasets, while originally used in the SAT’e study [25], have been used extensively to explore

over the years to illustrate that alignment error given fragmentary sequences increases with p-distance and fragmentation [2], as well as to benchmark new alignment methods [26] [27].

However, the ROSE datasets have some limitations that make them less realistic. One concern is that these simulated sequences evolved under an independent and identically distributed (*i.i.d.*) process. As biological data is not only not *i.i.d.*, but also has conserved segments due to selection pressures, the ROSE datasets are not realistic in this respect (they were not simulated with GAMMA+I, where I stands for invariable).

Another concern about the ROSE datasets might be that the various model conditions have a p-distance that is much higher than the p-distance seen in other biological datasets. However, the site heterogeneity in biological datasets means that at some locations in a genome, p-distance may be extremely high despite the overall average p-distance being low. Therefore, it is still useful to evaluate the various methods using the model conditions in the ROSE dataset, since this captures the accuracy of each extended alignment method to align the different challenging model conditions found in biological data.

In this thesis, for each ROSE model condition, we used the first five published replicates, and the provided binary model trees and true alignment. We used the ROSE datasets in Chapters 3 and 5.

RNASim-VS Datasets

The million-sequence RNASim dataset was published in an evaluation of alignment methods on ultra-large datasets, and to study the effect of alignment error on tree error [28]. The RNASim datasets explore conditions under a more realistic model of evolution (where the sites are not independent and identically distributed, as in the ROSE dataset, and are instead subject to selection pressures) [29]. Subsequently, as one of a few ultra-large and realistic datasets, the RNASim dataset has been widely used to study alignment methods [2], such as PASTA [28], MAGUS [29] and UPP [17] and placement methods APPLES [5], APPLES-2 [15], pplacerDC [30], pplacer-SCAMPP [7].

For replicates that contain at least 5,000 sequences, sites containing gaps in 95% or more of the alignment sequences have been removed [2]. For each replicate, we use the published true phylogenetic trees, multiple sequence alignments and published estimated FastTree-2 maximum likelihood trees [5].

In this thesis, the RNASim-VS datasets were used in Chapters 4 and 5. We describe the datasets in further detail in each chapter respectively.

2.2 Existing Methods to Analyze

In the following subsections, we detail existing extended alignment and placement methods. However, we do not discuss the new methods proposed in Chapter 5.

2.2.1 (Step 1) Extended Alignment Methods

UPP [17] and some variants of MAFFT can be used as extended alignment methods [18]. We evaluated these methods for their effect on phylogenetic placement accuracy. Commands are provided in Appendix A.

UPP

One advance in the field of sequence comparison has been to use probabilistic inference methods based on profile HMMs [31]. Profile HMMs are probabilistic models which summarize position-specific evolutionary changes, capturing the heterogeneity among sequences in an alignment. This technique has been particularly adept at performing alignment by capturing the homologies between sequences [16] [32].

UPP is an alignment method which leverages ensembles of profile Hidden Markov models (HMMs) to model the backbone alignment [17]. However, UPP can also be used to add a query sequence to an existing alignment. We describe this version as UPP-add here, but will refer to this method as UPP for brevity in the rest of the thesis.

UPP-add takes as input an existing backbone sequence alignment, a tree estimated on that backbone alignment, and a set of unaligned query sequences. Instead of summarizing the backbone sequence alignment with a single profile HMM, UPP-add uses the backbone tree to decompose the set of backbone sequences into several subsets. UPP-add restricts the full backbone alignment to each set of sequences, and trains a profile HMM on each sub-alignment so that in sum we have an ensemble of profile HMMs [17]. For each query sequence, UPP-add picks the best profile HMM in the ensemble and uses it to align the query sequence to the backbone alignment. Although UPP-add’s extended alignments have not been evaluated for its impact on phylogenetic placement accuracy, we will do so within this thesis.

The UPP study made key observations which make it a promising candidate as an extended alignment method. One such observation was that UPP’s approach produced particularly high accuracy when generating full alignments on datasets containing fragmentary sequences [17] [2]. UPP’s approach is to first estimate an alignment on the backbone sequences, before aligning the fragmentary query sequences to an existing backbone alignment; this problem is identical to the extended alignment problem.

In terms of scalability, the UPP study also saw that UPP and PASTA were the only methods to perform a full alignment when given 200,000 sequences [17]. Therefore, by using UPP as an extended alignment method, which should not be quite as computationally intensive [2], we can hope to tackle this problem of scalability in phylogenetic placement.

MAFFT

MAFFT is a well-known multiple sequence alignment package which provides the user with several techniques to align a query sequence to an existing alignment. To add general query sequences, MAFFT recommends the flag `-add` [33]. To add specifically fragmentary query sequences to an existing alignment, MAFFT recommends the flag `-addfragmentary` [33].

A key difference between the two is that `-addfragmentary` does not consider the relationship among the query sequences whereas `-add` does. For details on the algorithmic differences between the two, we refer the reader to the papers introducing these methods [33] [18]. MAFFT also provides several additional flags governing how to compute a distance matrix for all provided backbone and query sequences. We summarize the published documentation [18] on these flags below:

- `-default -add` switches between the available algorithms (`-6mer`, `-localpair`, `-globalpair`) according to the size of the input dataset.
- `-localpair -add` uses local pairwise alignment information to produce slow but accurate alignments. This flag is not recommended for alignments containing > 200 sequences.
- `-multipair -addfragmentary` (also referred to in this thesis as `-addfragments`) this is the default for the `-addfragmentary` flag, where dynamic programming is used to compute the distances. This flag is expensive in terms of runtime and memory. It is applicable to approximately $< 30,000$ sites and $< 1,000$ sequences.
- `-6mer -addfragmentary` (also referred to in this thesis as `-addfragments`) computes distance based on the number of shared 6mers. This flag runs the most quickly and consumes less memory than `-multipair -addfragmentary` memory.
- `-lastmultipair -addfragmentary` (also referred to in this thesis as `-addfragments`) is also available for faster computation, but results in poor accuracy.

We explored the following MAFFT variants: -default-add, -localpair-add, -multipair-addfragmentary and -6mer-addfragmentary.

We should also note that MAFFT is generally not as accurate on large datasets as UPP [17].

2.2.2 (Step 2) Placement Methods

Alignment-based maximum likelihood placement methods like pplacer require the query sequences to already be aligned to the backbone sequences [7] [19], and numeric parameters to be estimated on the input topology [34]. Such numeric model parameters include the branch lengths and the substitution rate matrix (e.g., for the Generalized Time Reversible model [35]).

We will begin with a discussion of the current most accurate placement method on fragmentary sequences, pplacer [6] [7] [15]. The placement problem can be formulated as an optimization problem involving the required numeric parameters: for each placement edge considered, pplacer introduces a node on that branch to be the parent, and estimates the placement (pendant) branch length for that leaf query [34]. Run in the maximum likelihood mode, pplacer evaluates the likelihood weight ratio (normalized so that across all placement locations sum to 1) as a reasonable proxy for the more statistically rigorous posterior probability calculation [34]. It outputs the top placements with a corresponding probability for each candidate placement edge.

However although pplacer is the most accurate method, as mentioned in Chapter 1 pplacer produces illegal outputs on some datasets larger than 10,000 sequences [7] [15]. As several studies have characterized [13] [5] [7], pplacer experiences “numerical issues” on larger backbone trees. The SCAMPP paper explored this in greater detail [7], showing that as pplacer placed into larger subtrees, the negative log likelihood of a placement calculation increased until it produced negative infinity. When pplacer produced negative infinity, it failed to produce a legal placement for that query sequence.

EPA-ng [19] is another maximum likelihood placement method. EPA and pplacer take about the same runtime and produce very similar placement accuracy on full-length query sequences, but have slightly different memory usage and flexibility in the input parameters [6].

Other distance-based methods such as APPLES-2 are much faster and have lower memory requirements [15]. However, to date pplacer is still the most accurate placement method according to phylogenetic placement error. Additionally, pplacer-SCAMPP was more accurate than APPLES-2 throughout several simulated and biological dataset, especially on datasets with fragmentary sequences [7].

Scaling pplacer

Thus far, we have discussed several reasons to align sequences to larger trees. We should also note the benefits of placing query sequences into larger trees. Studying the effect of denser taxon sampling on phylogenetic placement is difficult, as taxon sampling may affect the backbone tree or alignment accuracy rather than the accuracy of the placement process. However, an existing observation within the literature is that when working with the true simulated alignment, increasing the size of the true simulated model tree which pplacer places into, increases the phylogenetic placement accuracy [6]. This observation has also been made in other placement method studies, using estimated trees [15] [7]. Thus, placing into larger trees is desirable; in the following paragraphs we discuss the approaches to scaling pplacer.

There are several published approaches to scale pplacer, namely those using divide-and-conquer techniques [7] [30], and an approach involving changing the software used to estimate the numeric parameters (using FastTree-2 instead of RAxML) [5]. Some of these approaches to scaling pplacer are well understood (e.g., divide-and-conquer [7]), but the numeric parameter approach is not well studied at all. In this section we discuss some of these approaches in more detail.

Although the divide-and-conquer approach has been tried in a few papers [7] [30] [36], we refer the reader to existing literature for a thorough discussion of these approaches [7]. We summarize here that pplacer-SCAMPP has two key advantages [7]: it can run on unrooted trees [36] and can scale to place into trees containing 200,000 sequences [30].

Thus, we focus our discussion here on the SCAMPP framework’s divide-and-conquer technique instead. At a high level, the so-called EPA-ng-SCAMPP and pplacer-SCAMPP work as follows: given a user-provided parameter B determining the size of the tree pplacer will place into, first a “placement subtree” including the $B - 1$ closest leaves to the query is identified within the full backbone tree. This means that this placement subtree is specific to a given query sequence. Then, a placement method is used as a subroutine to place the query sequence into the identified subtree, before using this placement to finally resolve the query’s placement into the full backbone tree [7]. Thus, a method like pplacer-SCAMPP scales pplacer by using a divide-and-conquer technique to run it on small subtrees.

The SCAMPP paper also noted the benefits of dense taxon sampling [7]; placing queries into larger placement subtrees within the provided backbone tree decreased the delta error for both tested placement methods (e.g., pplacer and EPA-ng) [7]. However, this trend is limited when using pplacer as the placement method; pplacer’s accuracy breaks down on larger subtrees (increasing in error slightly on 4,000 sequences, and significantly on 8,000 sequences) which explains the default placement subtree size for pplacer-SCAMPP

as 2,000 sequences [7]. Notably, as the size of the placement subtree increases the negative log likelihoods (when using RAxML numeric parameters) increase dramatically, hitting negative infinity at which point it failed to produce a valid output for some query sequences on a placement subtree size of 11,000 sequences [7].

Overall, using SCAMPP greatly reduced runtime and memory consumption in both cases, with slight variations in accuracy [7]. Notably, on full-length query sequences, it was observed that EPA-ng-SCAMPP performed worse in terms of delta error than EPA-ng, whereas pplacer-SCAMPP sometimes performed better than pplacer [7]. The greatest improvement was on large backbone trees where neither pplacer nor EPA-ng could run, so that using SCAMPP allowed them to run with low memory consumption. As pplacer-SCAMPP improved on pplacer's accuracy, much of the exploration focused on this method [7].

Another approach to scaling pplacer is changing the numeric parameters. Although RAxML v7.2.6 is recommended for use with pplacer [34], using FastTree-2 numeric parameters bundled with taxtastic were observed to solve the numeric instability on the large RNASim-200k dataset [5]. In a related paper [15], RAxML v8 was shown to have the same numeric instability problems on the large RNASim-200k dataset, despite use with the taxtastic package. Finally, the numeric instability problems were also observed when placing some taxa from the biological LTP_s128_SSU dataset (using the published PhyML tree and using RAxML numerical parameters) containing approximately 12,000 sequences [7].

One hypothesis that might explain pplacer's illegal outputs on large datasets could be that the input RAxML-estimated numeric parameters have numeric problems when the backbone tree is very large. Although the pplacer documentation recommends using RAxML v7.2.6 to perform numeric parameter estimation [34], a proposed new way uses FastTree2 (run with single precision) to estimate numeric parameters on the topology instead [15]. We then use a package called taxtastic [37] to reformat the inputs into a reference package which pplacer can understand. We refer to this approach as pplacer-taxtastic [37] in the rest of the thesis, and detail how this approach differs from pplacer (default) below. As pplacer-taxtastic has not yet been studied in comparison to other highly scalable phylogenetic placement methods, in this thesis we shed light on these techniques used for scalability of placement methods.

pplacer Variants

In this section we detail the different ways of running pplacer.

The pplacer-default Pipeline

This is the use of pplacer in default mode, which operates as follows. Given a backbone alignment and tree, we use RAxML v. 7.2.6 to estimate numeric parameters on the topology. We then place the query sequence(s) into the tree using pplacer. The key difference between the following two methods, pplacer-SCAMPP and pplacer-taxtastic, is that the numeric parameters are estimated differently.

The pplacer-taxtastic Pipeline

Given a backbone alignment and tree, we use FastTree-2 to estimate numeric parameters on the topology. Then, we use the taxtastic package [37] to reformat the tree topology, numeric parameters and alignment into a reference package for use with pplacer. Then, pplacer uses this reference package to place queries into large datasets.

The pplacer-SCAMPP Pipeline

Given a backbone alignment and tree, we use RAxML 7.2.6 to estimate the numeric parameters on the topology. Then, given a query sequence, pplacer-SCAMPP uses the SCAMPP framework to find the nearest leaf (using Hamming distance) to the given query sequence, and uses this to extract a placement subtree of size B (with $B=2,000$ by default, as recommended in [7]). To place the query sequence into the subtree, pplacer-SCAMPP runs pplacer for query placement. The placement in the subtree provides the location as well as how the selected branch subdivides its branch length, which is then used to place the query sequence.

2.2.3 Criteria

We evaluated phylogenetic placement accuracy using delta error, a distance metric quantifying the increase in estimation error incurred by placing the query sequence into the backbone tree [7], [5]. Suppose T is the backbone tree used, T^* is the true tree, and P is the tree produced when our query q is added into backbone tree T using the phylogenetic placement pipeline. Let L represent the set of leaves in T , and let $B(t)$ refer to the set of bipartitions for a tree t . Then the equation for delta error is provided below:

$$\Delta e(P) = |B(T^*) \setminus B(P)| - |B(T^*|_L) \setminus B(T)|$$

Note that regardless of whether the estimated backbone tree contains estimation errors, if the query sequence is placed on the same edge as in the reference tree, the delta error can still be 0.

2.2.4 Runtime Constraints

In addition to delta error, we also evaluated each extended alignment method for runtime, memory consumption, and ability to produce an extended alignment on large sets of backbone sequences.

Experiments 1, 2 and 3 (Chapter 3) were run on the Campus Cluster with 64 GB and 4 wall clock hours available. For the experiments described in Experiments 4 and 5 (Chapter 4), all pplacer runs besides the 200,000-leaf dataset run were performed on the Campus Cluster, with 64 GB and 168 wall clock hours available. For the last 200,000-leaf run, we limited analyses to 256 GB and a 168 wall clock hour limit on the tallis queue. All other methods in Experiment 5 were run with 64 GB and 24 wall clock hours available. Experiments 6, 7 and 8 (Chapter 5) were performed on the UIUC Campus Cluster, using the secondary queue with a time-limit of 4 wall-time hours per query and 64 GB of available memory.

CHAPTER 3: SMALL BACKBONE ALIGNMENTS

3.1 Experiment Design

Given a backbone alignment, corresponding tree, and a single query sequence, Step 1 in the alignment-based phylogenetic placement pipeline is to first obtain an extended alignment. Step 2 is to place the query sequence into the backbone phylogeny. In this chapter, we varied only the extended alignment method, exploring the use of UPP and MAFFT on the task of aligning a fragmentary query sequence to the backbone alignment. We evaluated each method’s performance in terms of phylogenetic placement error, runtime, memory and ability to run on large datasets.

3.1.1 Experiment Pipeline Overview

Given a set of sequences as input:

- **Input Creation:** We used the Fragmentary Protocol A to split our input dataset into a set of backbone sequences and a set of fragmentary query sequences (see Section 2.1 for more details). This protocol determined the number of query sequences and the average length of each query sequence after fragmentation.

Given the set of unaligned backbone sequences, we obtained a backbone alignment either by inducing from the reference alignment (Experiments 1 and 2) or by re-estimating the backbone alignment (Experiment 3) using the scalable and highly accurate alignment method MAGUS.

We then obtained a corresponding backbone tree either from the reference tree (Experiment 1), or we used the phylogeny-inference method FastTree-2 to estimate a corresponding topology on this backbone alignment.

- **2-Step Phylogenetic Placement:** Given the backbone alignment, we aligned the query sequence to the backbone alignment, exploring UPP and MAFFT. This generates an extended alignment.

Using this extended alignment, we then placed the query sequence into the backbone phylogeny using the highly accurate placement method pplacer.

- **Evaluation:** We evaluated each of the extended alignment methods in terms of delta error, runtime, memory, and the size of the largest backbone sequence set they are able to run on. To calculate phylogenetic placement error we compared the tree after placement to the reference tree.

MAGUS is a state-of-the-art highly accurate and fast multiple sequence alignment method [29], which has been shown to produce more accurate alignments more quickly than existing methods like PASTA on large datasets. On smaller datasets, MAGUS matches PASTA’s accuracy and speed [29]. We use MAGUS to estimate backbone alignments.

In Experiment 1, we evaluated the different MAFFT variants on the ROSE dataset. From this exploration, we selected two “best” MAFFT variants to further evaluate (“best” in terms of the accuracy-runtime trade-off). In Experiment 2, we compared these two “best” MAFFT variants to UPP on reference backbone alignments and trees in order to understand these methods in the context of accuracy-runtime considerations. In Experiment 3, we evaluated the “best” MAFFT variants to UPP on estimated backbone alignments and trees.

3.1.2 Datasets

We ran all experiments on all ROSE datasets model conditions, excluding 1000M1. As mentioned in Chapter 2, we excluded the results for ROSE 1000M1 in because we have reason to believe these analyses produced especially high delta error due to a bug in the analysis code.

Table 3.1 provides further detail on the ROSE datasets before and after fragmentation according to Fragmentary Protocol A described in Section 2.1.

3.1.3 Biological Relevance

Recall that alignments when given fragmentary sequences (or any kind of sequence length heterogeneity) is challenging. This chapter explores the biological application of aligning fragmentary reads, such as sequencing reads, to a large existing alignment. Experiments 1 and 2 evaluate the extended alignment methods on the more ideal case of having the true backbone alignment and the true backbone tree. However, these experiments are valuable because in these simulated datasets where we have “truth”, the delta error can be entirely attributed to the extended alignment method.

Experiment 3 evaluates a more biologically realistic case, where the backbone sequence alignment is estimated. This allows us to extrapolate the conclusions from Experiments 1 and 2 to a more widely relevant

conclusion. Furthermore, it allows us to observe how errors in the Step 1 extended alignment pipeline, rather than errors from the Step 2 placement pipeline, affect the ending phylogenetic placement error.

3.2 Results and Discussion

3.2.1 Experiment 1

Figure 3.1 illustrates delta error and runtime results for the MAFFT variants on the high fragmentary ROSE datasets. Across all model conditions, in general as p-distance increased the delta error also increased for all methods. There was a wide margin between methods in terms of both delta error and runtime. MAFFT linsi add and MAFFT-multipair-addfragmentary were the two most accurate methods according to delta error. A wide margin in delta error separated these methods from MAFFT default -add and MAFFT 6mer -addfragmentary. However, MAFFT default -add and MAFFT 6mer -addfragmentary proved to be much faster than the two more accurate methods.

Although MAFFT linsi add was the most accurate method, it was the most expensive method in terms of runtime. The next most accurate method, MAFFT multipair -addfragmentary, was also much more expensive in terms of runtime, although it was at least four times faster than MAFFT linsi add. The remaining extended alignment methods described in Figure 3.1 were much faster, but unfortunately on model conditions with high p-distance, these methods suffered extremely high delta error. We discuss the differences within each group (the faster but less accurate and the slower but more accurate) in Figures 3.5 and 3.4.

Figure 3.2 illustrates delta error and runtime results for the MAFFT variants on the low fragmentary ROSE datasets. Similarly, across all model conditions, in general as p-distance increased, the delta error also increased for all methods. We also observe the same clear grouping of methods which were more accurate (MAFFT linsi add and MAFFT multipair addfragmentary) but took longer to run, and methods which were much faster to run but much less accurate (MAFFT default add and MAFFT 6mer addfragmentary). It should be noted that the delta error and runtime values for Figures 3.1 and 3.2 can be compared, but it should be noted that we placed into backbone trees with different numbers of sequences in each figure.

Figure 3.3 illustrates the memory consumed by each MAFFT variant in both the high and low fragmentary scenarios. MAFFT linsi add and MAFFT multipair -addfragmentary consumed the most memory by a wide margin. MAFFT linsi add consumed much more memory than MAFFT multipair addfragmentary. As the model conditions' p-distance increased, the amount of memory consumed also tended to increase, with 1000L3 being an exception to this trend. Across the two fragmentary cases, more memory was consumed in the low

fragmentary case, but this can be explained by the fact that there were more backbone sequences in the low fragmentary case.

We focused on the runtime-accuracy trade-off of the two slowest MAFFT variants in Figure 3.4. This figure illustrated the delta error and runtime of MAFFT linsi add and MAFFT multipair addfragmentary when aligning high fragmentary query sequences. As the model conditions' p-distance increased, delta error and runtime taken also generally increased. Both MAFFT variants produced approximately the same delta error when the p-distance was low (although there were runtime differences), but on model conditions with higher p-distance, MAFFT linsi add produced slightly higher delta error. Additionally, MAFFT linsi add took strictly more time than MAFFT multipair addfragmentary on all datasets.

Notably, MAFFT linsi add had higher variance across replicates, and was also less consistent in the runtime consumed across model conditions when compared to MAFFT6mer-addfragmentary. In particular MAFFT linsi add's average runtime generally increased as the model conditions' p-distance increased. MAFFT linsi add's variable runtime could be explained by the documentation, which notes that the -addfragmentary flag does not consider the relationship between the query sequences, whereas -add does. This might explain why as the p-distance increased, MAFFT linsi add took slightly more time on model conditions with high p-distance than on model conditions with low p-distance. Since MAFFT multipair addfragmentary enjoys better accuracy as well as less runtime than MAFFT linsi add, it would seem that the extra runtime effort to consider the relationship between query sequences is not justified on fragmentary sequences.

We explored the runtime-accuracy trade-off of the two fastest MAFFT variants further in Figure 3.5. This figure illustrates the delta error and runtime of the two fastest explored MAFFT variants when aligning high fragmentary query sequences. In general, as p-distance increased, delta error and runtime taken also increased, with a few exceptions to this trend. Both MAFFT variants produced approximately the same delta error when the p-distance is low (although there were runtime differences), but on higher p-distance datasets, MAFFT6mer-addfragmentary produced better accuracy. MAFFT default add took nearly twice the runtime compared to MAFFT6mer-addfragmentary on model conditions with low p-distance. However, on model conditions with high p-distance, the runtime was extremely similar between the two methods, with MAFFT default add taking slightly more time. This is surprising because MAFFT6mer-addfragmentary computes distance based off of 6mer's and is optimized for speed over accuracy. However, the delta error and runtime differences between the two methods were small.

We explored the runtime-accuracy trade-off of the two slowest MAFFT variants further in Figure 3.6. This figure illustrates the delta error and runtime of the two slowest explored MAFFT variants when aligning low

fragmentary query sequences. In general, as p-distance increased, delta error and runtime taken also increased, with a few exceptions to this trend.

Interestingly, both MAFFT variants produced approximately the same delta error across all model conditions. This trend is different from the trend in Figure 3.4, where we observed that MAFFT linsi add incurred slightly more error on model conditions with high p-distance. However, the observation that MAFFT linsi-add took much more time than MAFFT multipair addfragmentary on all model conditions is consistent with the results observed in the high fragmentary scenario. Additionally, in Figure 3.6 we also observe that the runtime is much more variable both across replicates and across model conditions in the results for MAFFT linsi add than in the results for MAFFT multipair addfragmentary. Although this observed difference between the -add flag and the -addfragmentary flag can be observed across the two fragmentation scenarios, it is most clearly observed in the comparison between the “slow but accurate” methods.

We explored the runtime-accuracy trade-off of the two fastest MAFFT variants further in Figure 3.7. This figure illustrates the delta error and runtime of the two fastest explored MAFFT variants when aligning low fragmentary query sequences on the ROSE dataset. In general, as p-distance increased, delta error and runtime taken also increased, with a few exceptions to this trend. Both MAFFT variants produced approximately the same delta error when p-distances are small, with a slight advantage to MAFFT6mer-addfragmentary when p-distances increased. This is consistent with the observations from Figure 3.5. However, one major difference is that MAFFT default add took less runtime than MAFFT6mer-addfragmentary on two model conditions with high p-distance in this case.

Table 3.2 summarizes the exploration of the selected MAFFT variants. MAFFT linsi -add and MAFFT multipair -addfragmentary produced nearly identical delta error, with a slight advantage to MAFFT multipair -addfragmentary. In comparison, MAFFT 6mer -addfragmentary and MAFFT default -add achieved very similar delta error values, but both achieved consistently much worse delta error than the other MAFFT variants.

In terms of runtime, MAFFT default add and MAFFT6mer-addfragmentary were also very similar, with a slight advantage to MAFFT6mer-addfragmentary. MAFFT linsi add was the slowest of all MAFFT variants, and also consumed much more memory than any other method.

We selected two MAFFT versions to continue exploring. MAFFT6mer-addfragmentary and MAFFT default add are very similar in terms of delta error, runtime and memory. However, the documentation for MAFFT 6mer-addfragmentary suggests that this method achieves its speed by using an approximation which quickly suffers poor accuracy on larger datasets. For these reasons, when runtime and memory considerations are

more important than delta accuracy, we recommend using MAFFT default add. When delta accuracy is more important, we recommend using MAFFT multipair addfragmentary.

3.2.2 Experiment 2

Figure 3.8 compares the MAFFT variants determined in Experiment 1 to UPP, when given high fragmentary query sequences and reference backbone alignments and trees on the ROSE datasets. As in Experiment 1, we observed that as the model conditions' p-distance increased, delta error also increased. On model conditions with low p-distance, all methods produced very similar delta error results. However, on model conditions with higher p-distances we observe that MAFFT default add produced much worse delta error values. UPP demonstrated a slight advantage over MAFFT multipair addfragmentary on a few cases, especially as the dataset p-distance increased. However, across all model conditions MAFFT multipair addfragmentary was consistently the most expensive method in terms of runtime. UPP was approximately twice as fast as MAFFT multipair-addfragmentary, but MAFFT default add was the fastest method by a wide margin.

Figure 3.9 compares the recommended MAFFT variants to UPP when given low fragmentary query sequences and reference backbones on the ROSE datasets. Similar to in Figure 3.8, we observed that MAFFT default add produced much worse delta errors on model conditions with higher p-distances. UPP and MAFFT multipair addfragmentary achieved nearly identical delta error values across all model conditions, although MAFFT multipair-addfragmentary was consistently the most expensive method in terms of runtime.

Figure 3.10 illustrates memory consumption for our selected MAFFT variants and UPP across all model conditions and in both high and low fragmentary scenarios. Although the figure shows that more memory is consumed in the low fragmentary scenario, this can be explained by the fact that the backbone sequence sets are larger in the low fragmentary scenario. However, within each fragmentary scenario, MAFFT multipair addfragmentary consumed the most memory, followed by UPP. MAFFT default add consumed negligible amounts of memory.

Table 3.3 summarizes our comparisons in Experiment 2. UPP and MAFFT multipair addfragmentary can both be used in cases which require low delta error. MAFFT default add should be used in cases which require fast runtime and low memory consumption.

3.2.3 Experiment 3

Figure 3.11 illustrates our “best” MAFFT variants compared to UPP on the task of aligning high fragmentary query sequences to a MAGUS estimated backbone alignment. On the low p-distance ROSE datasets, all methods achieved very similar delta error. On the high p-distance ROSE datasets, MAFFT default -add incurred more delta error any other compared method, and UPP achieved the lowest delta error. In terms of runtime and runtime, we observed that UPP was also much faster and consumed less memory than MAFFT multipair -addfragmentary, although MAFFT default -add was still the fastest method (and consumed the least memory). Thus, UPP represents a middle-ground, achieving high accuracy while taking a reasonable runtime and consuming little memory.

Figure 3.12 illustrates our “best” MAFFT variants compared to UPP on the task of aligning low fragmentary query sequences to an estimated backbone alignment. We observed the same trends as in Figure 3.11; in terms of delta error, on the high p-distance ROSE datasets, MAFFT default add became much worse than any other compared method, and UPP achieved the lowest delta error. In terms of runtime and memory, we similarly observed that UPP was much faster and consumed less memory than MAFFT multipair addfragmentary, although MAFFT default add was still the fastest method and consumed the least memory.

3.3 Conclusions

In this chapter we explored the effect of extended alignment methods on phylogenetic placement accuracy by evaluating method behavior when given reference alignments on the 1,000-leaf ROSE datasets. We found that on these small ROSE datasets, MAFFT default add was fastest and consumed negligible memory, but it achieved lower accuracy than MAFFT linsi add and MAFFT multipair addfragmentary. Although MAFFT linsi add and MAFFT multipair addfragmentary achieved similar accuracy, MAFFT multipair addfragmentary consistently runs faster and consumes less memory than MAFFT linsi add. We found that UPP and MAFFT multipair addfragmentary achieved comparable phylogenetic placement accuracy on the task of placing fragmentary sequences, but that MAFFT default add was the fastest method. However, UPP was the method that provided the best balance with the best accuracy, with acceptably fast runtime and low memory consumption. We found that the trends across methods were consistent across fragmentary proportions (high and low fragmentary scenarios), on both reference and estimated backbone alignments.

In this chapter, we addressed the choice of extended alignment methods when given reference alignments on simulated “small” backbone sets of sequences. In the following chapters, we evaluate whether these conclusions generalize to “larger” datasets containing more sequences in the backbone alignment and tree.

3.4 Tables and Figures

Datasets	Avg. Align Len	Avg. Unalign Seq Len	Avg. Query Len (HF)	Avg. Query Len (LF)	Avg. p-dist	Max. p-dist
1000S1	1901	779	249	500.2	0.692	0.774
1000S3	1715	791	250	500.4	0.689	0.760
1000S5	1151	782	248	498.9	0.506	0.617
1000M3	2361	830	251	502.3	0.658	0.747
1000M5	1690	822	251	499.1	0.495	0.609
1000L1	3517	890	251	506.0	0.695	0.768
1000L3	7142	943	255	512.6	0.692	0.766
1000L5	1859	938	250	500.9	0.498	0.599

Table 3.1: **Dataset Statistics on ROSE simulated datasets used for extended alignment method analysis.** All used datasets are publicly available. Datasets presented span three different model conditions, where 1 is the highest rate of evolution (most difficult model condition) and 5 is the lowest rate of evolution (easiest model condition). We also have introduced gaps, varying from short gaps (S) to long gaps (L). These fragments were generated using Fragmentary Protocol A described in Section 2.1. We report dataset statistics on the five used replicates.

Method	Accuracy	Runtime	Memory Consumption
MAFFT 6mer -addfragmentary	3 // 4	2	3
MAFFT default -add	3 // 4	1	1
MAFFT linsi -add	1 // 2	4	4
MAFFT multipair -addfragmentary	1 // 2	3	2

Table 3.2: **(Experiment 1) Summary of MAFFT variants.** In each column, the methods are ordered from “best” (1) to “worst” (4), where “best” refers to the lowest delta error, lowest runtime and lowest memory consumption. “//” is used to indicate a tie between two methods. The “best” method in each category is bold.

Method	Accuracy	Runtime	Memory Consumption
UPP	1 // 2	3	2
MAFFT multipair -addfragmentary	1 // 2	4	3
MAFFT default -add	3	1	1

Table 3.3: **(Experiment 2) Summary of compared extended alignment methods.** In each column, the methods are ordered from “best” (1) to “worst” (3), where “best” refers to the lowest delta error, lowest runtime and lowest memory consumption. “//” is used to indicate a tie between two methods. The “best” method in each category is bold.

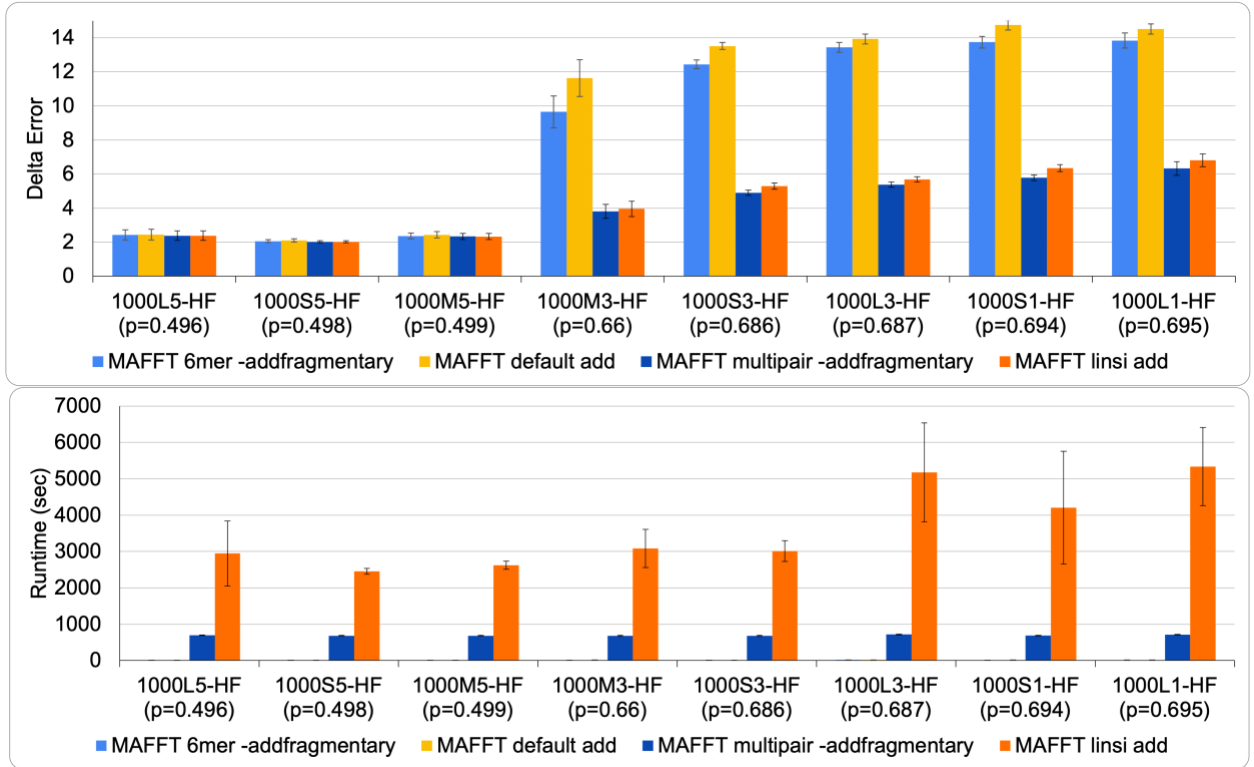


Figure 3.1: **(Experiment 1) Exploring the use of MAFFT variants to align 500 high fragmentary (HF) query sequences to the reference backbone alignment for placement into the true backbone tree.** On the x-axis, the ROSE datasets are sorted according to the p-distance calculated in the full-length dataset. In the top row, we report the average delta error and standard error. In the bottom row, we report the average runtime and standard deviation. Recall that in HF we present the per query delta error (over 500 queries) when placing into a backbone tree of 500 sequences.

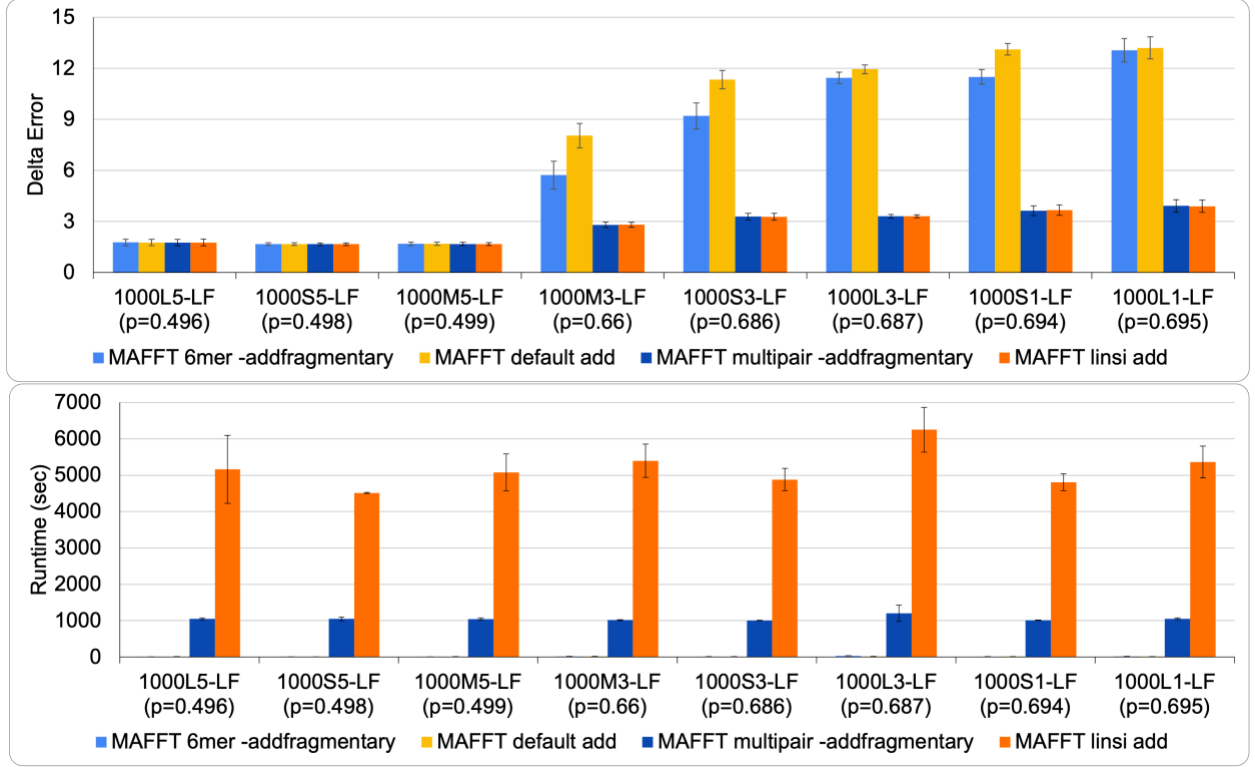


Figure 3.2: **(Experiment 1) Explores the use of MAFFT variants to align all 250 low fragmentary (LF) query sequences to the reference backbone alignment for placement into the true backbone tree.** On the x-axis, the ROSE datasets are sorted according to the p-distance calculated in the full-length dataset. In the top row, we reported the average delta error and standard error. In the bottom row, we reported the average runtime and standard deviation. Recall that in LF we presented the delta error (over 250 queries) when placing into a backbone tree of 750 sequences.

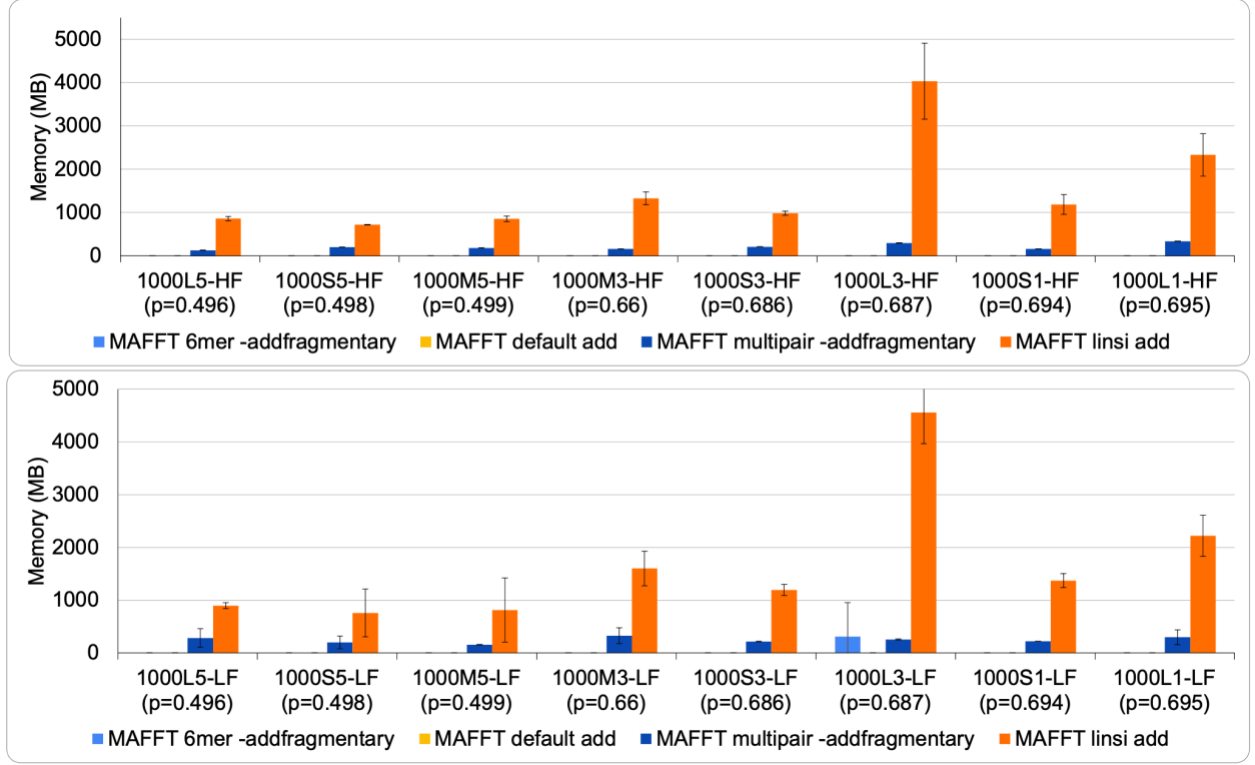


Figure 3.3: **(Experiment 1) Memory consumed by MAFFT variants when aligning all query sequences to the true backbone alignment, before placement of the queries into the true backbone tree.** The top row is the high fragmentary scenario (placing into backbone trees with 500 sequences) and the bottom row is the low fragmentary scenario (placing into backbone trees with 750 sequences). Average memory consumption (with standard deviation) was reported over 5 replicates for each dataset. Note that MAFFT 6mer -addfragmentary and MAFFT default -add may not be visible as both consumed negligible memory for several datasets.

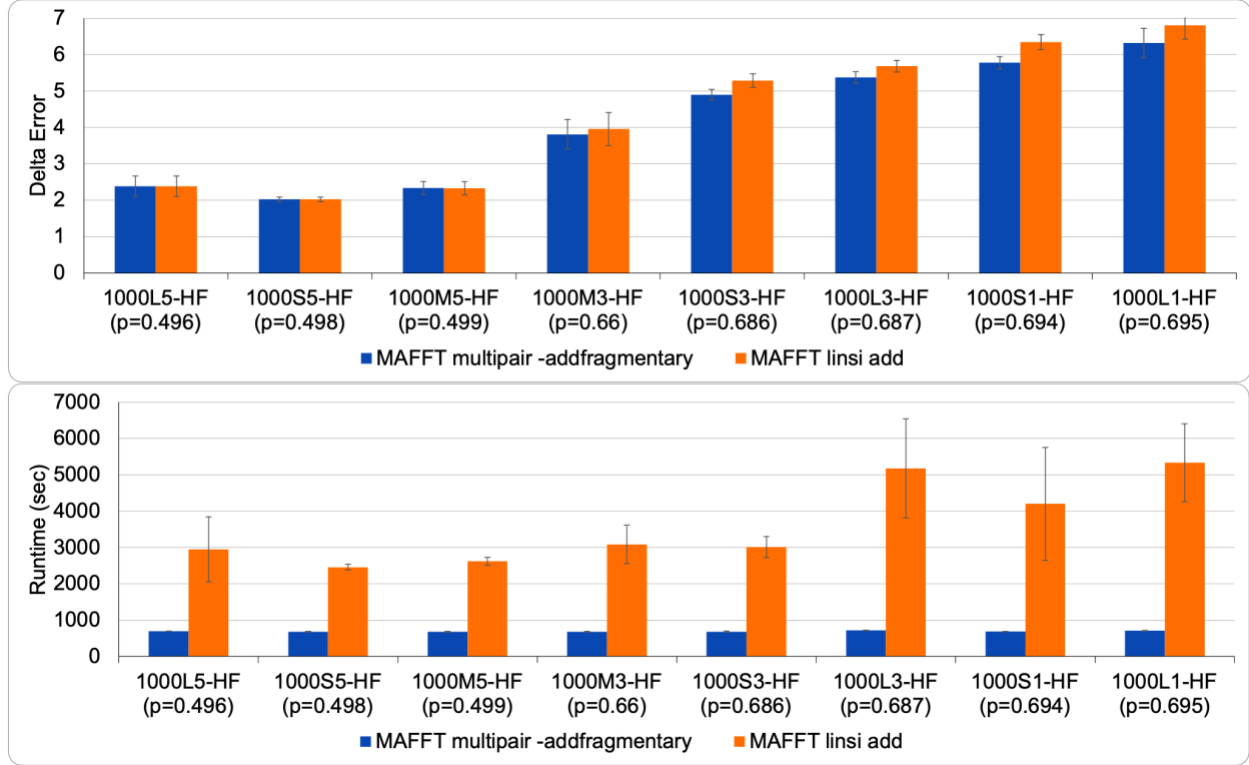


Figure 3.4: **(Experiment 1) Runtime of the two slowest explored MAFFT variants when aligning all 500 high fragmentary query sequences to the true backbone alignment, before placement into the true backbone tree.** Average runtime (with standard deviation) was reported over 5 replicates for each dataset.

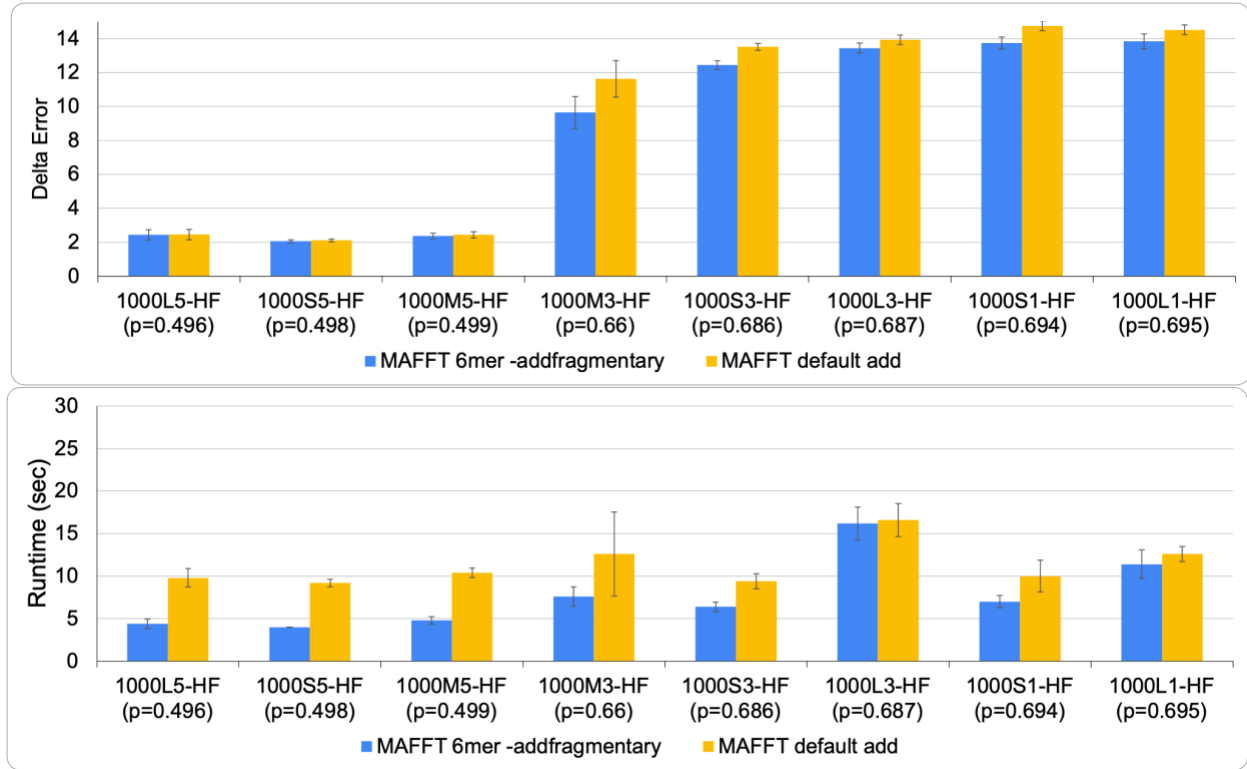


Figure 3.5: **(Experiment 1) Delta error and runtime of the two fastest explored MAFFT variants when aligning all 500 of the high fragmentary query sequences to the true backbone alignment, before placing into the true backbone tree.** Average delta error (with standard error) and runtime (with standard deviation) was reported over 5 replicates for each dataset.

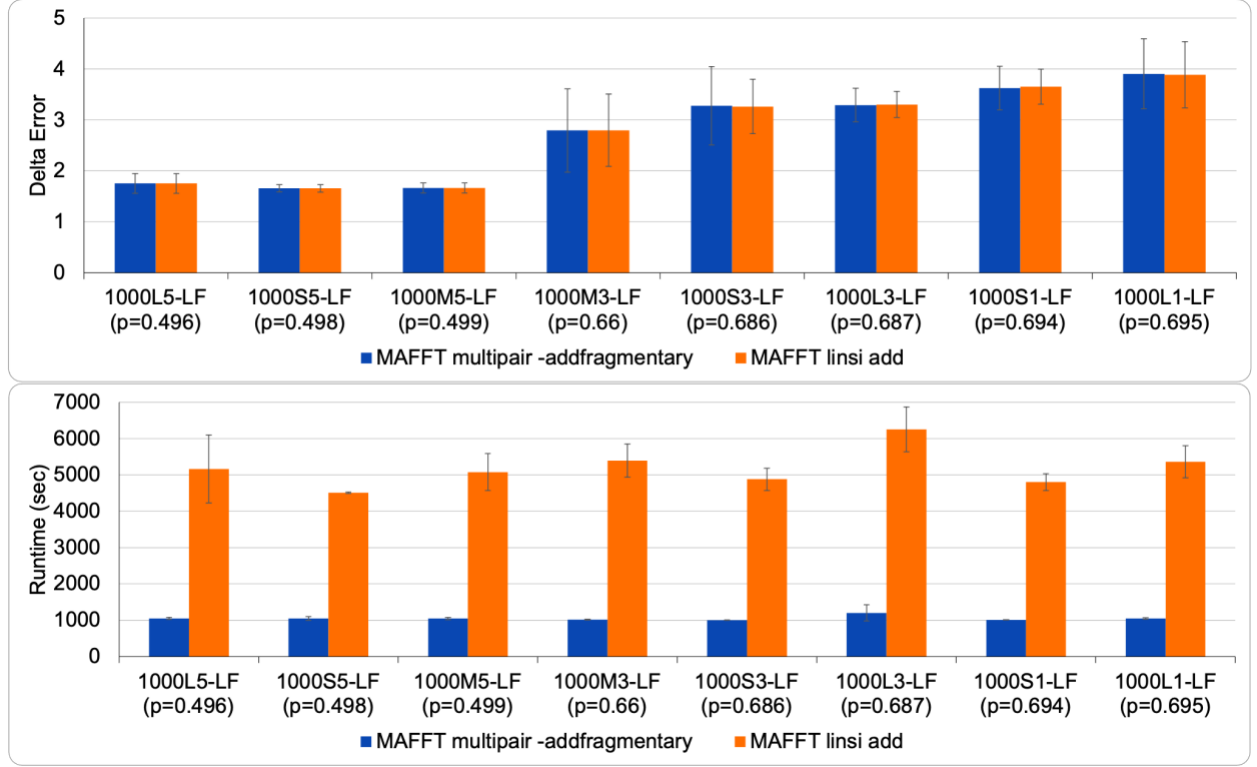


Figure 3.6: **(Experiment 1)** Delta error and runtime of two slowest explored MAFFT variants when aligning low fragmentary query sequences to the true backbone alignment and placing into the true backbone tree. Average delta error (with standard error) and runtime (with standard deviation) was reported over 5 replicates for each dataset.

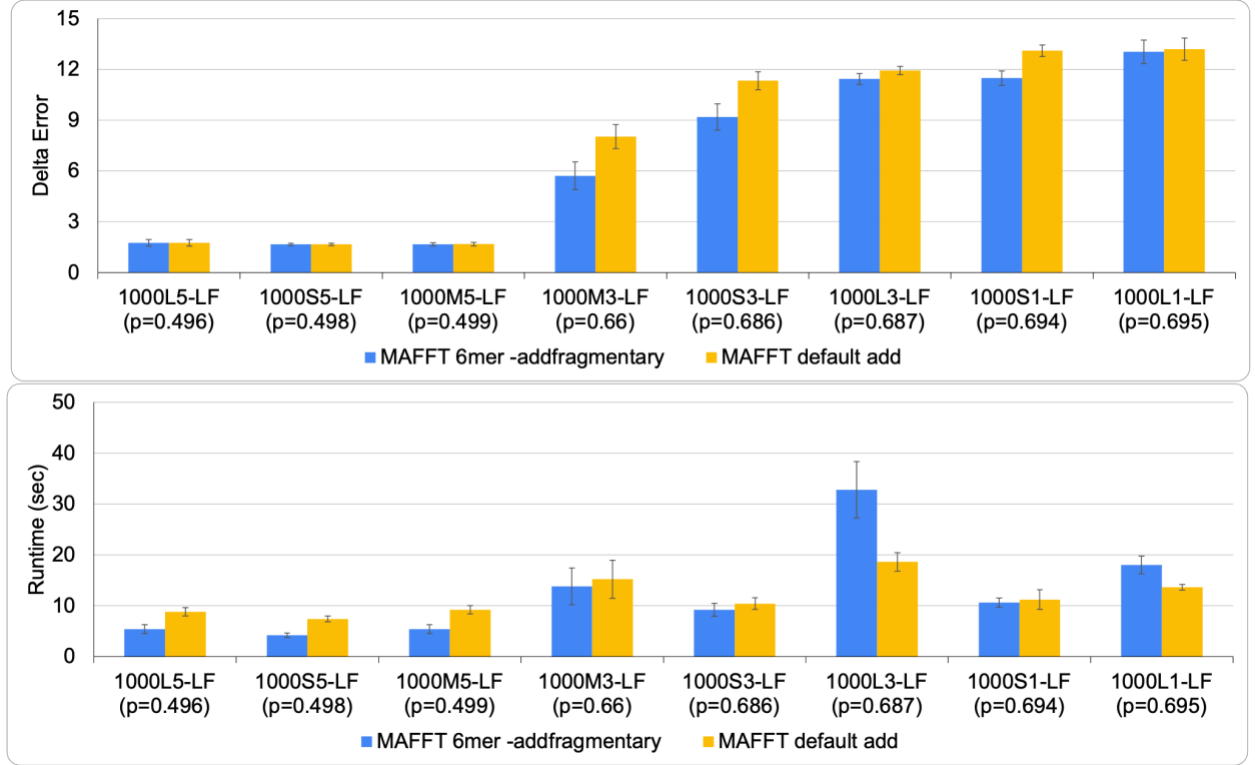


Figure 3.7: **(Experiment 1) Delta error and runtime of two fastest explored MAFFT variants when aligning low fragmentary query sequences to the true backbone alignment and placing into the true backbone tree.** Average delta error (with standard error) and runtime (with standard deviation) was reported over 5 replicates for each dataset.

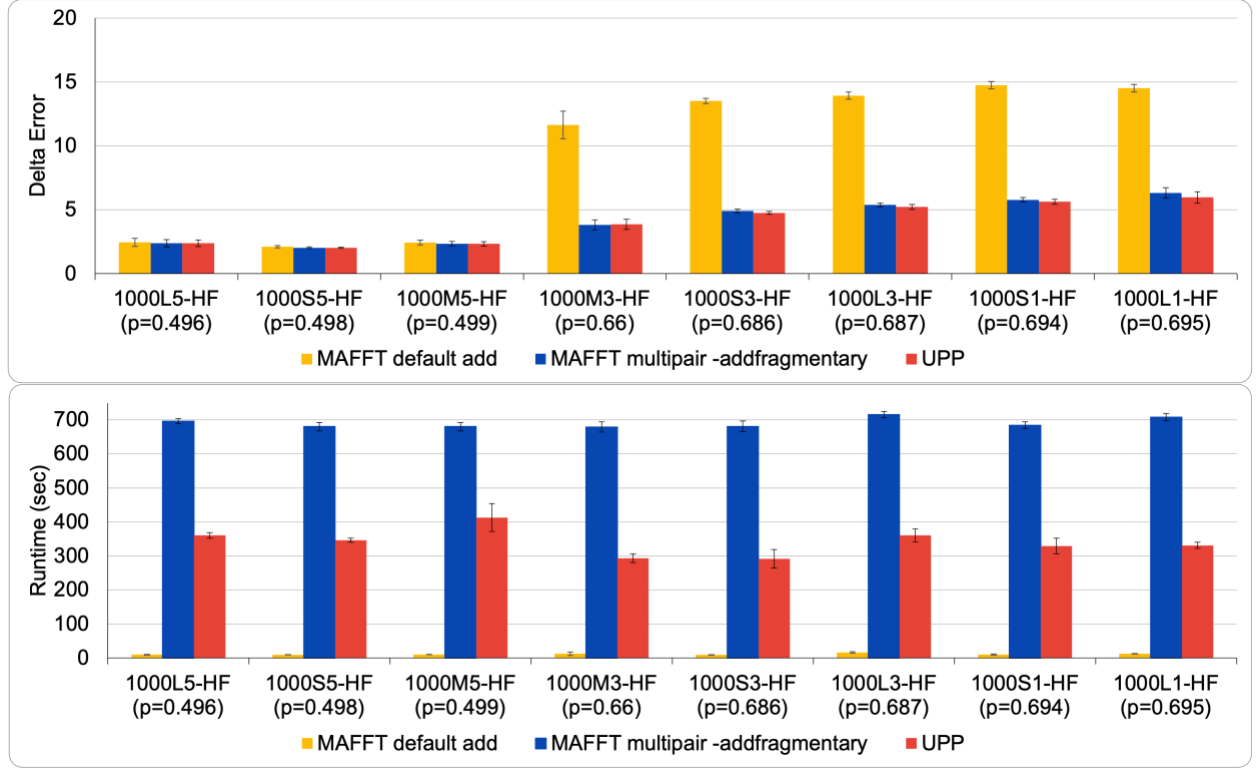


Figure 3.8: **(Experiment 2) Delta error and runtime reported when aligning all 500 high fragmentary query sequences to the true backbone alignment, before placing them into the true backbone tree. We present results comparing the recommended MAFFT variants with UPP, applied to the ROSE dataset.** On the x-axis, the ROSE datasets are sorted according to the p-distance calculated in the full-length dataset. Average delta error (with standard error) and runtime (with standard deviation) is reported over 5 replicates for each dataset.

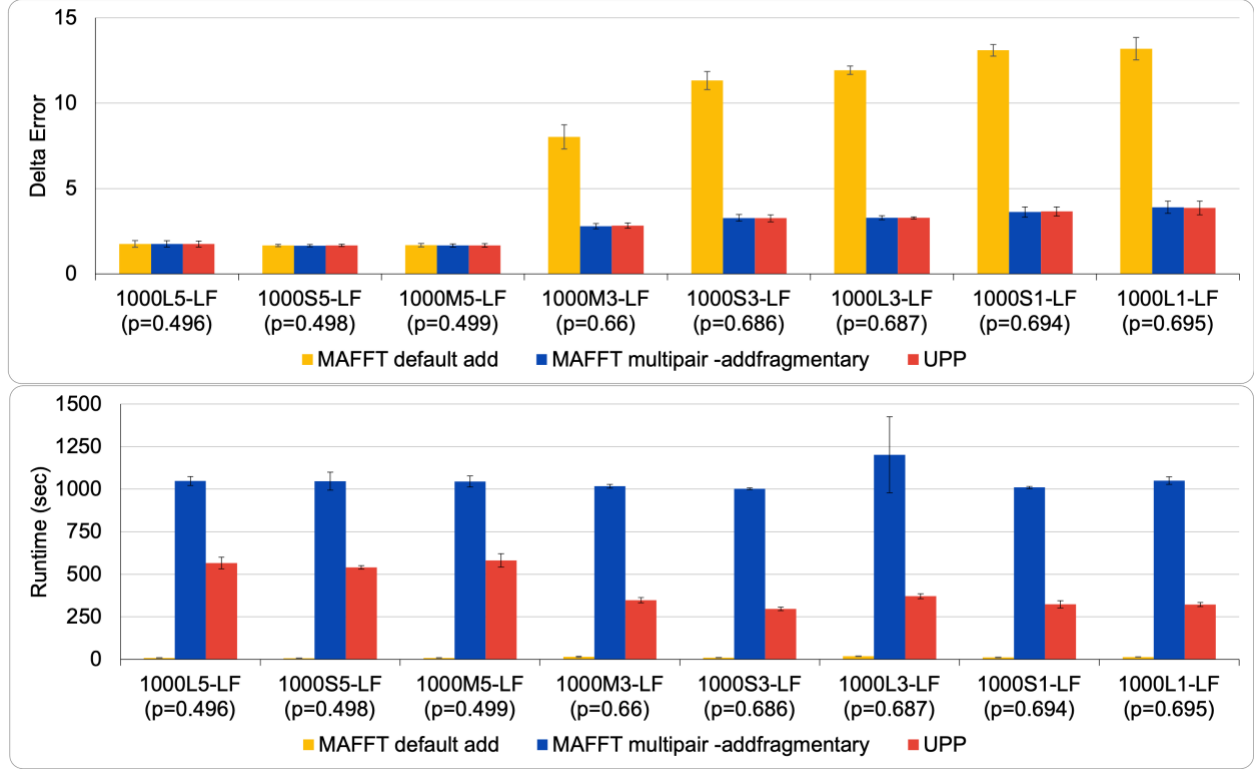


Figure 3.9: **(Experiment 2) Delta error and runtime reported when aligning all 250 low fragmentary query sequences to the true backbone alignment, before placing them into the true backbone tree. We present results comparing the recommended MAFFT variants with UPP, applied to the ROSE dataset.** On the x-axis, the ROSE datasets are sorted according to the p-distance calculated in the full-length dataset. Average delta error (with standard error) and runtime (with standard deviation) is reported over 5 replicates for each dataset. The top row illustrates the results when datasets have undergone high fragmentation (HF), and the lower row illustrates results when datasets have low fragmentation (LF).

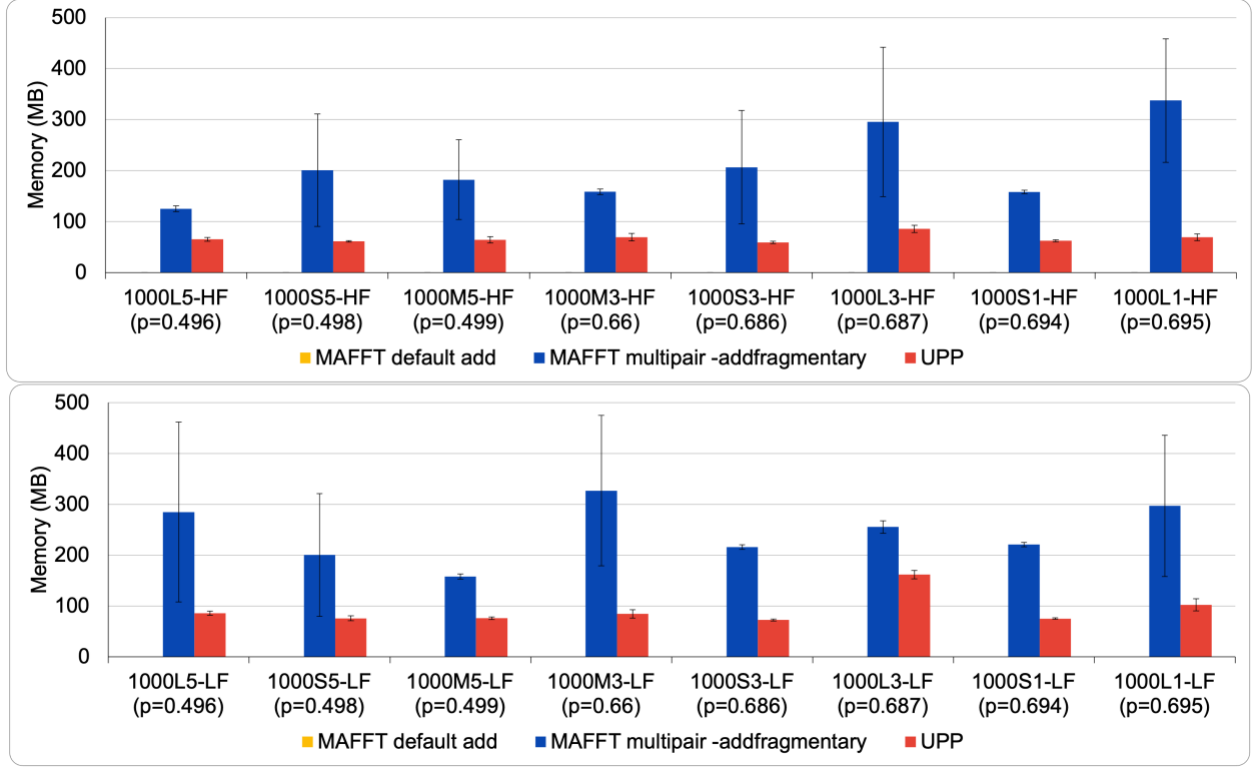


Figure 3.10: **(Experiment 2) Memory usage reported when aligning all fragmentary query sequences to the true backbone alignment, before placing them into the true backbone tree. We present results comparing the recommended MAFFT variants with UPP, applied to the ROSE dataset.** On the x-axis, the ROSE datasets were sorted according to the p-distance calculated in the full-length dataset. We reported the average memory and standard deviation. The top row illustrates the results when datasets have undergone high fragmentation (HF), and the lower row illustrates results when datasets have low fragmentation (LF).

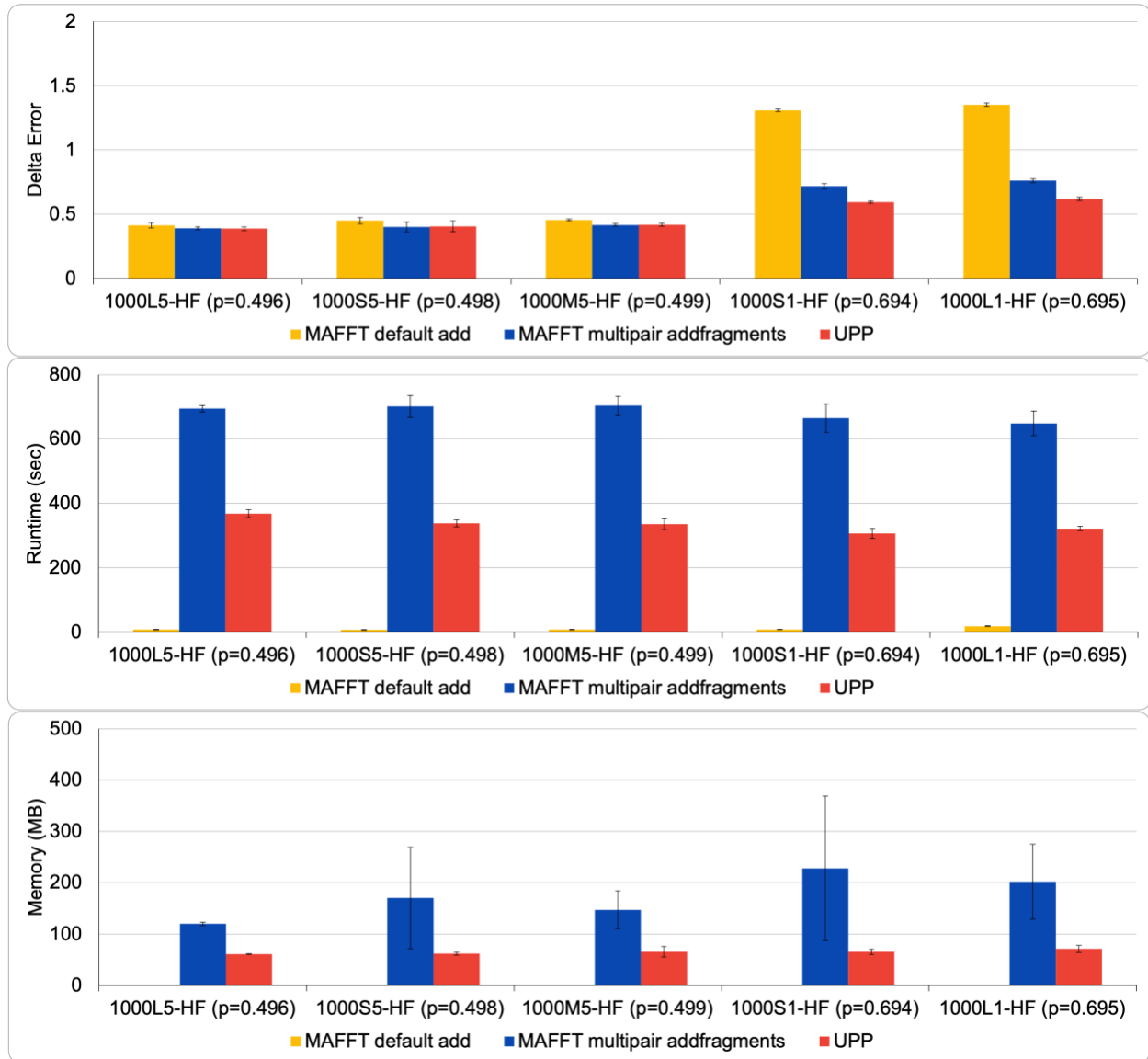


Figure 3.11: **(Experiment 3) Delta error, runtime and memory results when aligning all 500 high fragmentary query sequences to a MAGUS-estimated backbone alignment (containing 500 sequences), before placement of query sequences into a FastTree-2 estimated backbone tree.** On the x-axis, the ROSE datasets are sorted according to average p-distance. Average delta error (with standard error) and runtime and memory (with standard deviation) was reported over 5 replicates for each ROSE dataset.

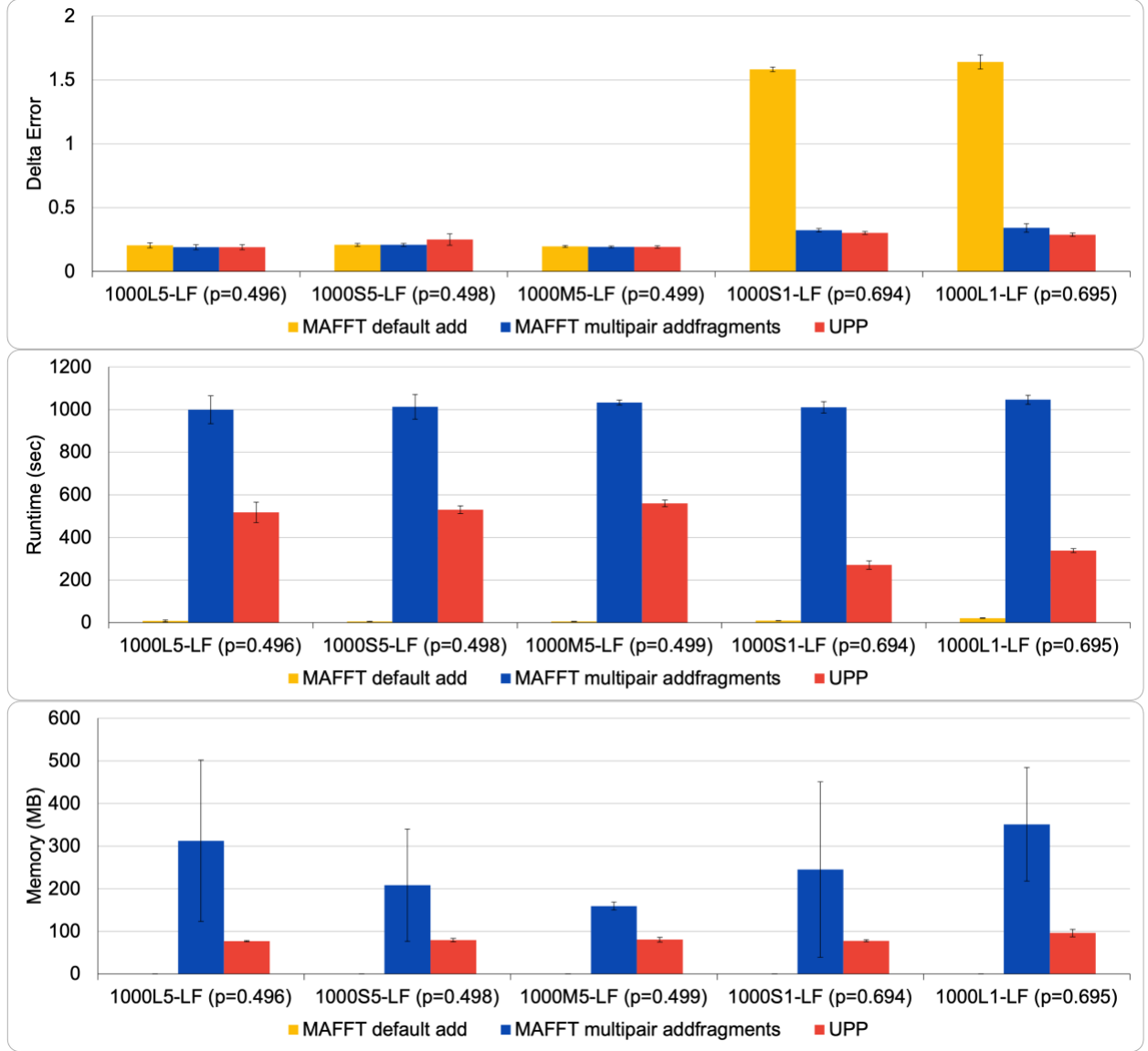


Figure 3.12: **(Experiment 3) Delta error, runtime and memory results when aligning all 250 low fragmentary query sequences to a MAGUS-estimated backbone alignment (containing 750 sequences), before placement of query sequences into a FastTree-2 estimated backbone tree.** On the x-axis, the ROSE datasets were sorted according to average p-distance. Average delta error (with standard error) and runtime and memory (with standard deviation) was reported over 5 replicates for each ROSE dataset.

CHAPTER 4: LARGE BACKBONE ALIGNMENTS

4.1 Experiment Design

4.1.1 Overview

In this chapter we describe several experiments to explore the role of extended alignment methods when aligning queries to larger backbone sets.

Experiment Pipeline on Large Backbones

Note that we used exactly the same pipeline as in Chapter 3, but in this chapter we used FastTree to estimate the backbone topology and numeric parameters and the taxtastic package to build the reference packages for the placement method pplacer. The pipeline is described below.

Given a set of aligned sequences as input:

- **Input Creation:** We split the given set of sequences into a set of aligned backbone sequences and a group of 200 unaligned query sequences. Each of the query sequences is fragmented (at 50% for high fragmentation and 25% for low fragmentation levels, see Section 2.1 for more details).

Given the reference backbone alignment, we obtain a corresponding backbone tree by using the phylogeny-inference method FastTree to estimate a topology and numeric parameters on this backbone alignment. Given the backbone alignment, topology and numeric parameters, we use the taxtastic package to compile this into a reference package for use with pplacer.

- **2-Step Phylogenetic Placement:** Given the backbone alignment, we align the query sequence to the backbone alignment using one of our extended alignment methods. Using this extended alignment method, we then place this query sequence into the backbone phylogeny using pplacer.

Experiment 4 explored the largest datasets pplacer is able to run on. Using the reference backbone alignment, reference extended alignment containing all sequences, and an estimated backbone tree, we evaluated what the largest dataset pplacer was able to successfully complete on.

Exact commands for Experiment 4 can be found in the Appendix. We ran pplacer in Experiment 4 using the taxtastic library to compile this into a reference package containing the backbone alignment, topology and numeric parameters. We ran default pplacer on replicate *R0* for each dataset.

Experiment 5 explored the behavior of extended alignment methods on larger datasets. More specifically, we evaluated each extended alignment method in terms of runtime, memory and phylogenetic placement error incurred. The goal of this experiment was to understand whether the trends in behavior of extended alignment methods on the smaller ROSE datasets from Chapter 3 generalized to these larger datasets.

We tested two MAFFT variants (default-add and multipair-addfragmentary) and UPP. As Experiment 4 succeeded, we evaluated these extended alignment methods for delta accuracy as well as runtime and memory consumption.

4.1.2 Datasets

We sampled subsets of the million-taxon RNASim simulated dataset [28] in order to have more variable sizes with which to evaluate our extended alignment methods. These newly sampled subsets can be found published in the Illinois public repository [38]. We used the same sampling protocol as in previous studies of RNASim [2]. Each of the first six datasets contains five replicates, but the last 200,000-sequence RNASim dataset only contains one replicate. For each dataset, 200 sequences were used as queries and were put through Fragmentary Protocol A (see Section 2.1 for more details). The other $n - 200$ sequences are used as the backbone sequences.

Since all the sequences were sampled from the same million-sequence dataset, we note that the median backbone sequence length was 1539.0 basepairs and that the backbone alignment's p-distance was 0.410 for all our RNASim-VS datasets. The average query length in the case of the low fragmentation protocol was 769.5 basepairs and in the case of the high fragmentation protocol was 384.8 basepairs.

4.1.3 Biological Relevance

Recall that phylogenetic placement is especially challenging on large datasets ($> 5,000$ sequences) because of challenges in both the alignment and placement method steps. Furthermore, as we have already established, working with sequence length heterogeneity is challenging. This chapter explores scalability of both the extended alignment methods and the highly accurate maximum likelihood placement method pplacer.

Experiment 4 is directly relevant to both applications of phylogenetic placement (e.g., microbiome analysis and placement into large gene trees). Experiment 5's evaluation of the scalability of the extended alignment

methods is likewise also relevant to both applications of phylogenetic placement, as it is run on fragmentary query sequences.

4.2 Results and Discussion

4.2.1 Experiment 4

Figure 4.1 illustrates the runtime taken for pplacer to place 200 query sequences into the backbone alignment on various subsets of the RNASim dataset, when using the reference alignment as the extended alignment. Notably, pplacer was able to place 200 query sequences into extremely large backbone phylogenies - containing up to 200,000 sequences in the backbone alignment and tree. As the size of the backbone tree increased, the number of seconds required to place each query also increased. The memory also increased proportionally, approximately doubling as the backbone dataset size doubled. However, the overall pplacer runtime and seconds to place per query did not increase linearly as the dataset size increased, while the memory did increase proportionally. However, we should also note that pplacer's ability to place query sequences into large RNASim datasets is specific to the RNASim dataset. Other studies have shown pplacer's inability to place into much smaller backbone sequence sets in different datasets [7]. We address this topic in greater detail in Chapter 5.

As Experiment 4 illustrated the ability to use pplacer to place query sequences into large backbone phylogenies, we then proceeded with Experiment 5's exploration of the extended alignment methods' delta placement accuracy, runtime and memory consumption.

4.2.2 Experiment 5

Figure 4.2 illustrates the accuracy-runtime trade-off among the extended alignment methods when aligning low fragmentary query sequences with increasingly large backbone alignments (and placing into the corresponding trees). Understanding this trend is complicated, but similar trends in terms of placement accuracy have been observed in other papers. For example, in Figure 1A of the APPLES-2 paper [15], we see that placement accuracy increases as the size of the backbone tree increased for pplacer-taxtastic, APPLES, APPLES-2 and EPA-ng. Although placement accuracy is a different metric, our results confirm the same trend [15]. Similarly, the same trend of increasing phylogenetic placement accuracy with backbone tree size appears for the same datasets with other placement methods in Figure 5 of the paper introducing SCAMPP [7]. We do not yet understand why this is.

MAFFT default add consistently produced the worst delta error. When it could run, MAFFT multipair addfragmentary produced the best delta error. However, although MAFFT multipair addfragmentary performed slightly better in terms of accuracy than UPP on the RNASim dataset containing 10,000 sequences, this MAFFT variant failed to run on each of the three largest RNASim datasets (the 50,000- and 100,000- and 200,000-leaf datasets respectively). Out of all tested extended alignment methods, only UPP was able to produce a result on the 200,000-leaf RNASim dataset. In terms of runtime, we observed trends similar to our observations in Experiments 2 and 3; the runtime for MAFFT multipair addfragmentary was consistently twice that of the UPP runtime. Although MAFFT default add was the fastest extended alignment method on small RNASim datasets, on the larger 100,000-sequence RNASim dataset UPP and MAFFT default add took the same amount of time to run. We observe that when compared to the other MAFFT variants, UPP was the best-in-class method in terms of both accuracy, runtime and scalability.

Figure 4.3 illustrates the accuracy-runtime trade-off among the extended alignment methods when aligning low fragmentary query sequences with increasingly large backbone alignments (and placing into the corresponding trees). We observed that the delta error decreased as the size of the backbone set of sequences increased. On the low fragmentary query sequence case, all extended alignment methods performed quite similarly in terms of delta accuracy. However, out of all tested extended alignment methods, only UPP was able to produce a result on the 200,000-leaf RNASim dataset. MAFFT multipair addfragmentary failed to run on the larger RNASim datasets containing 50,000-, 100,000- and 200,000-leaves.

From the bottom panel of Figure 4.3 the runtime of MAFFT multipair addfragmentary was consistently twice that of the UPP runtime. Although MAFFT default add was the fastest extended alignment method on all RNASim datasets in this case, it failed to run on the 200,000-leaf RNASim dataset.

In both the high fragmentary and low fragmentary cases (Figures 4.2 and 4.3), MAFFT multipair addfragmentary ran into memory issues. All datasets were able to run on sequence sets up to 10,000 sequences. However, we were not able to calculate delta error on the RNASim dataset with 10,000 sequences. On the 50,000 dataset, MAFFT multipair addfragmentary ran for the reported amount of time on

every replicate but produced an extended alignment only on replicate 2. The other replicates failed to produce a result, ending with an out of memory error. Thus, we do not show any results for MAFFT multipair addfragmentary on the 50,000-leaf RNASim dataset. On the 100,000- and 200,000-sequence datasets, MAFFT multipair addfragmentary ran on all replicates briefly before ending with an out of memory error. Thus, we also do not report any results for MAFFT multipair addfragmentary on these datasets.

As noted before, delta error decreases in Figure 4.3 as the backbone tree size increases. We leave an explanation of this trend to future work.

Figure 4.4 illustrates the memory consumption of all methods on both fragmentary scenarios. We observed that the memory of MAFFT default add far exceeded that of UPP on large datasets. This spike in memory usage began on the 50,000-leaf RNASim dataset.

4.3 Conclusions

We evaluated pplacer on its ability to place queries into large backbone trees, and subsequently explored the behavior of extended alignment methods on larger datasets. We found that when enough memory is provided, pplacer can be used to place query sequences into extremely large backbone phylogenies. Among the compared extended alignment methods, UPP was the fastest method achieving the lowest delta error when placing highly fragmentary query sequences into large backbone phylogenies. UPP had the clearest advantage on the largest datasets, as it was the only extended alignment method able to align query sequences to the 50,000-leaf and 100,000-leaf RNASim datasets. Improved accuracy with denser taxon sampling is a trend observed in Experiment 5, but an explanation is left to future work.

In this chapter, we addressed the choice of extended alignment methods when given “large” reference alignments. In the next chapter, we evaluate Step 2 of the phylogenetic placement pipeline, and explore several existing placement methods before proposing a combination of these techniques.

4.4 Figures

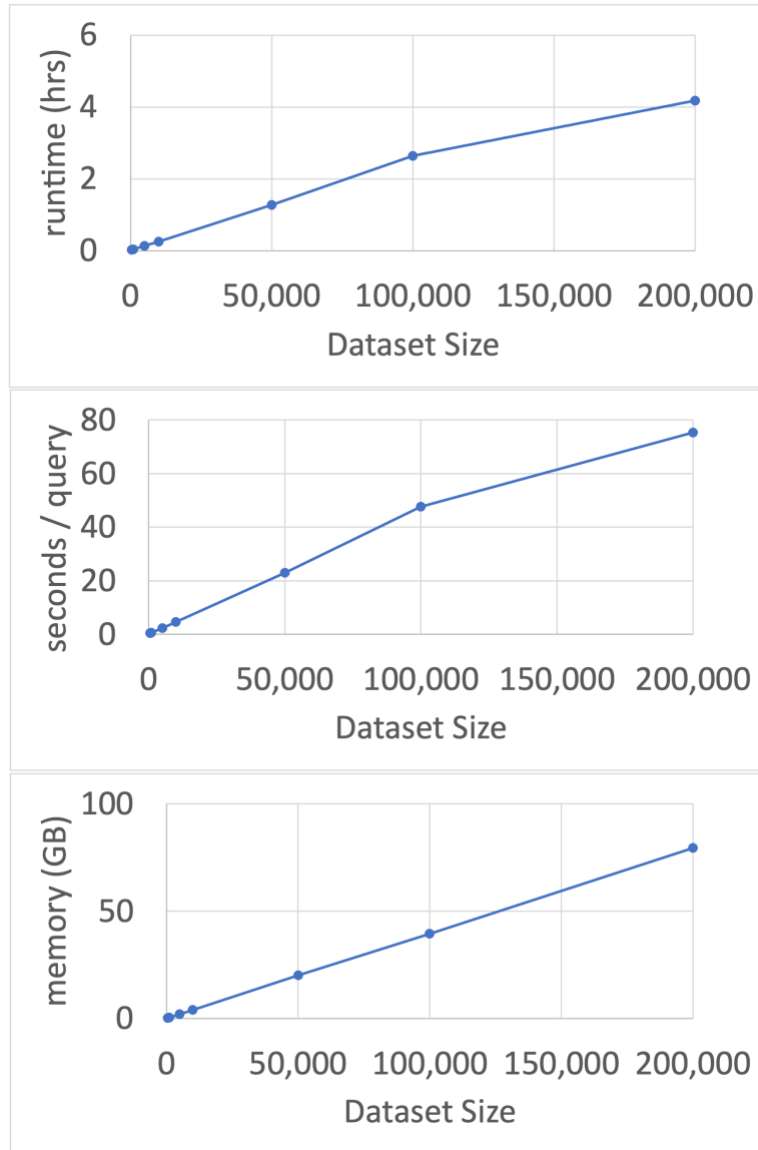


Figure 4.1: **(Experiment 4) Runtime and memory of pplacer.** We varied the number of backbone sequences which pplacer placed 200 query sequences into. The top row illustrates the total runtime in hours for placement of all 200 query sequences as the dataset size increases on the x-axis. The middle row illustrates the seconds to place each query as the dataset size increases on the x-axis. The last row shows the memory consumed to place all 200 query sequences as the dataset size increases on the x-axis. yet understand this trend.

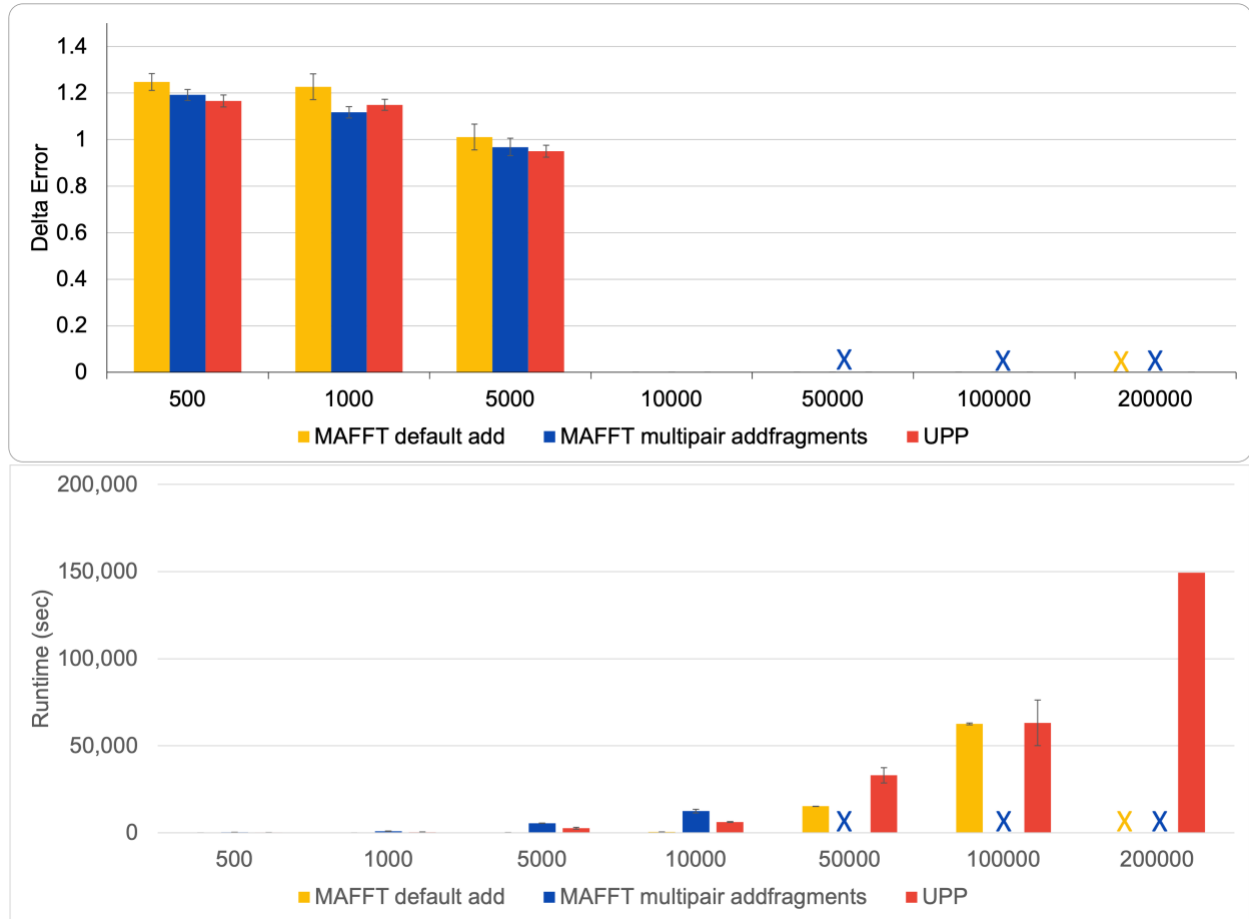


Figure 4.2: **(Experiment 5) Aligning high fragmentary query sequences with the reference backbone alignment and placing into the estimated backbone tree.** On the x-axis, the RNASim datasets were sorted according to the size of the backbone sequence set. The top row illustrates delta error accuracy results for each dataset size, and the bottom row illustrates the runtime considerations for each extended alignment method. Note that MAFFT multipair addfragmentary met an “out of memory” error on the 50,000-leaf and 100,000-leaf datasets. X’s indicate where out of memory errors were encountered and results could not be generated on the five replicates. Each dataset has five replicates except for the 200,000-leaf dataset, which has only one replicate. We reported average delta error (with standard error) and average runtime (with standard deviation). Where the delta error was not reported, although those extended alignment methods completed, delta error was not calculated because the time and memory to calculate delta error on each query sequence was too great. The focus of this analysis was on the runtime and memory consumption of these methods on large backbone alignment RNASim datasets.

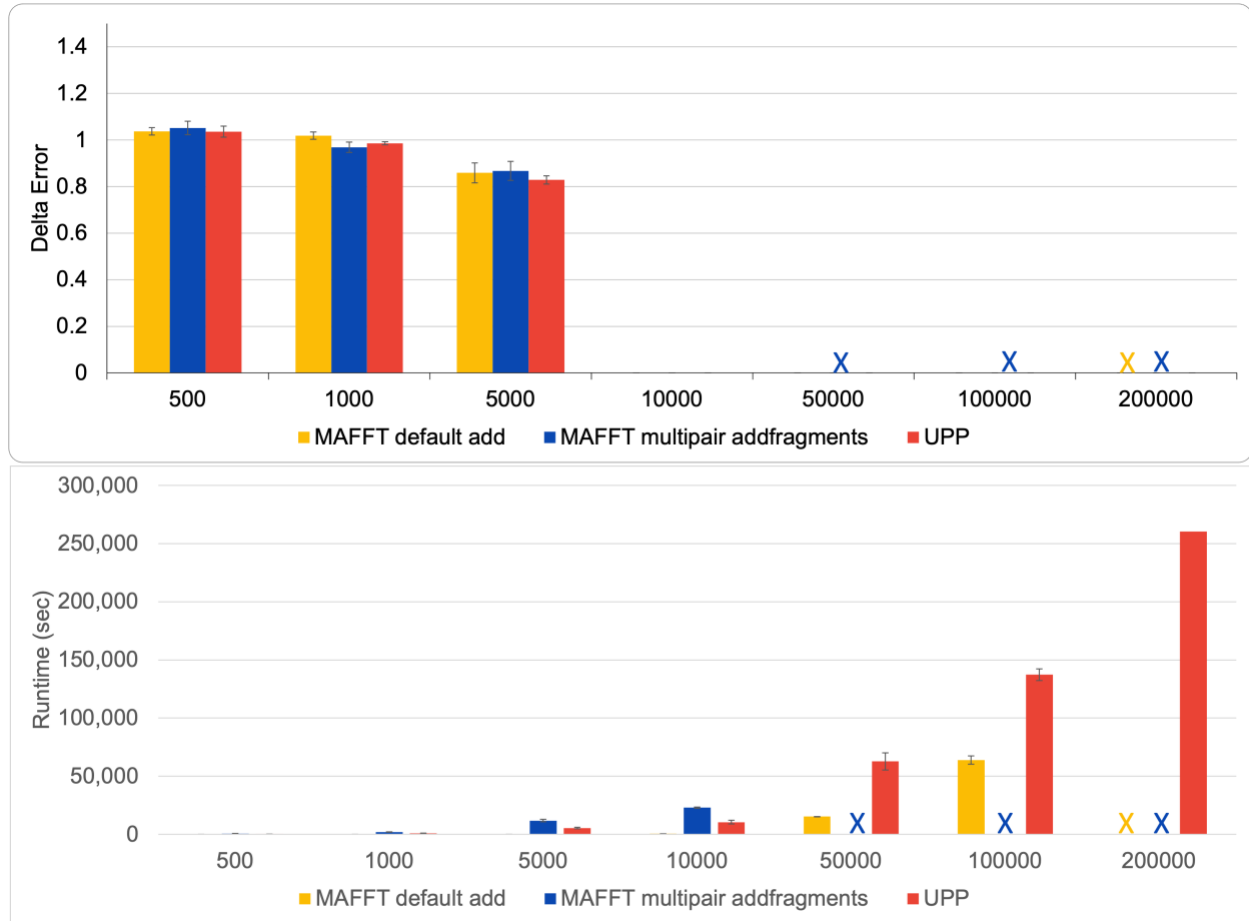


Figure 4.3: **(Experiment 5) Aligning low fragmentary query sequences with the reference backbone alignment and placing into the estimated backbone tree.** On the x-axis, the RNASim datasets were sorted according to the size of the backbone sequence set. The top row illustrates delta error accuracy results for each dataset size, and the bottom row illustrates the runtime considerations for each extended alignment method. Note that MAFFT multipair addfragmentary met an “out of memory” error in the 50,000-leaf and 100,000-leaf cases. X’s indicate where out of memory errors were encountered and results could not be generated on the five replicates. Each dataset had five replicates except for the 200,000-leaf dataset, which has only one replicate. We reported average delta error (with standard error) and average runtime (with standard deviation). Where the delta error was not reported, although those extended alignment methods completed, delta error was not calculated because the time and memory to calculate delta error on each query sequence was too great. The focus of this analysis was on the runtime and memory consumption of these methods on large backbone alignment datasets.

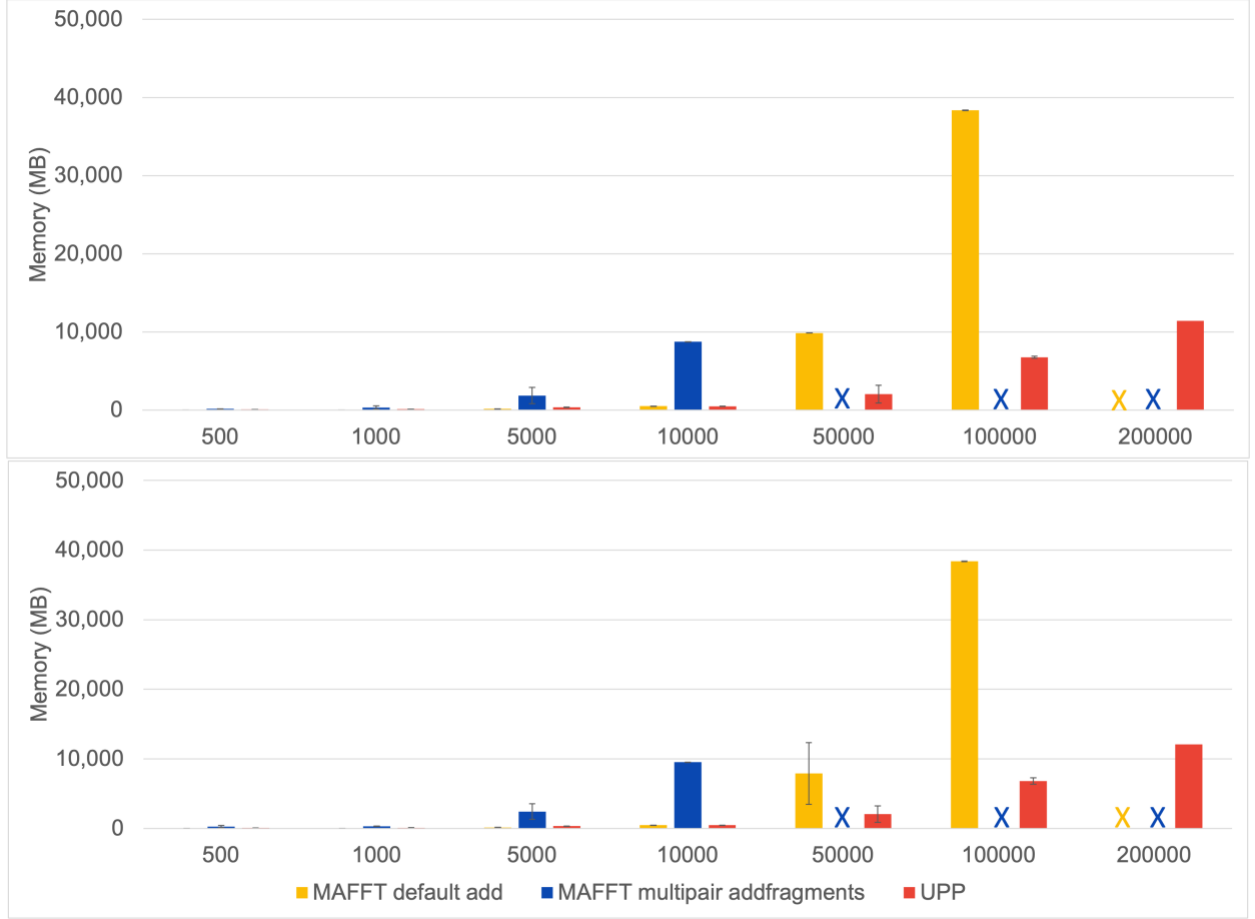


Figure 4.4: **(Experiment 5) Aligning query sequences with the reference backbone alignment and placing into the estimated backbone tree. We present a memory consumption comparison of the recommended MAFFT variants with UPP, applied to the RNASim dataset.** On the x-axis, the RNASim datasets were sorted according to the size of the backbone sequence set. The top row illustrates the results when datasets have undergone high fragmentation (HF), and the lower row illustrates results when datasets have low fragmentation (LF). X's indicate where out of memory errors were encountered and results could not be generated on the five replicates. Each dataset had five replicates except for the 200,000-leaf dataset, which had only one replicate. We reported averages and standard deviation for memory.

CHAPTER 5: PLACEMENT METHODS¹

5.1 Experiment Design

5.1.1 Overview

We explored four phylogenetic placement pipelines, which depend on both the extended alignment method, as well as the method used to estimate numeric parameters as well as the phylogenetic placement method.

Our experiments focused on scalability to large datasets and accuracy on large datasets, both when placing full-length sequences (the application relevant to adding sequences into large phylogenies) and when placing fragmentary sequences (the application most relevant to analyses of reads produced in microbiome analyses and metagenomics).

5.1.2 Exploring Placement Pipelines

The input to a phylogenetic placement pipeline is a backbone tree (in which the leaves are already aligned) and a single query sequence. Since the query sequences are inserted independently, the extension to multiple query sequences is trivial. The pipelines we explored first estimate branch lengths and other numeric parameters for each backbone tree, then use the specified phylogenetic placement method to insert the query sequence into the tree.

The phylogenetic placement methods we studied are: pplacer-default (also referred to simply as pplacer), pplacer-taxtastic, pplacer-SCAMPP (Chapter 2), and pplacer-taxtastic-SCAMPP. The developers of the phylogenetic placement methods each specify a particular method for numeric parameter estimation; these are indicated in Figure 5.1. In particular, the *taxtastic* package [37] recommends use with FastTree-2 numeric parameters, whereas pplacer (default) recommends use with RAxML v7.2.6 (and by extension, pplacer-SCAMPP recommends use with RAxML v7.2.6).

¹This work was previously published on bioRxiv.

Full citation: G. Chu and T. Warnow, “Exploring Trade-offs in Scalable Phylogenetic Placement Methods,” *bioRxiv*, [Online]. Available: <https://www.biorxiv.org/content/10.1101/2022.05.23.493012v1>.

The pplacer-SCAMPP-taxtastic Pipeline

This is identical to pplacer-SCAMPP (described in Chapter 2) except for the numeric parameter estimation, as we now describe. Given a backbone alignment and tree, we use FastTree 2 to estimate the numeric parameters on the topology. Then, given a query sequence, pplacer-SCAMPP-taxtastic uses the SCAMPP framework to find the nearest leaf (using Hamming distance) to the given query sequence, and uses this to extract a placement subtree of size B . Since pplacer-taxtastic can potentially run on larger trees than pplacer, the optimal value for B may be larger than its default setting in pplacer-SCAMPP. To place the query sequence into the subtree, pplacer-SCAMPP-taxtastic builds a taxtastic reference package on the subtree using the backbone sequences and numeric parameters, and provides pplacer with this taxtastic reference package for query placement.

Leave-one-out Experiments

All experiments in this chapter were run as leave-one-out experiments, with results reported on a per-query basis. For all datasets, we randomly choose 200 sequences to be used as the query sequences. Thus, for a given dataset with n sequences, with one selected as a query sequence, we use each phylogenetic placement method to place the selected query sequence into the backbone tree containing the other $n - 1$ sequences.

Datasets

We used three collections of simulated datasets (ROSE, RNASim-VS and nt78) to evaluate accuracy.

Fragmentary Protocol B

As opposed to Fragmentary Protocol A described in Section 2.1, to study the placement methods we applied more fragmentation to match other studies of placement methods [7]. In each dataset used, we selected 200 query sequences and made these fragmentary. This fragmentary protocol was applied to both the RNASim-VS and nt78 dataset, but not to the ROSE dataset.

Given a query sequence with original length l_o , a random starting position was selected by sampling from a normal distribution. Two sequences of different length were generated for each query sequence: in the low fragmentary case, sequence length was sampled using $N(\mu = .25l_o, \sigma = 60)$. In the high fragmentary case, sequence length was sampled using $N(\mu = .10l_o, \sigma = 10)$. Note that these distributions were chosen because 154 basepairs is close to the Illumina read sequence length, making a study of placing read-length query sequences more relevant to downstream applications of phylogenetic placement.

RNASim-VS Pipeline

We used three subsets from the published RNASim-VS dataset [5], containing 50,000, 100,000 and 200,000 sequences respectively. We refer to each subset as RNASim-50k, RNASim-100k and RNASim-200k. For RNASim-50k we arbitrarily choose to use the first 5 replicates. For the two larger datasets, we run on one replicate (arbitrarily picking R0).

nt78 Pipeline

This dataset was used to study the placement methods. nt78 contains 78,132 sequences and 20 simulated replicates, and was created to study FastTree-2 [39]. We chose the first replicate to use as it has been used in other studies as well [7]. The nt78 dataset was simulated with ROSE [22] under the HKY model [40] to resemble 16S sequences, and with different evolutionary rates for each site selected from 16 different rate categories. See the Appendix for additional information about this simulated dataset. Table 5.1 describes the dataset characteristics used in the placement method analysis.

5.1.3 Experiments

For all experiments, we used the true alignment as the extended alignment.

- Experiment 6: We compared the behavior of pplacer-default and pplacer-taxtastic when placing full-length query sequences using ROSE-1000M1, ROSE-1000M5, and nt78 datasets.
- Experiment 7: We explored pplacer-taxtastic and pplacer-SCAMPP on nt78 and RNASim-50k, using both full and fragmentary query sequences.
- Experiment 8: We evaluated the impact of changing the subtree size in pplacer-SCAMPP-taxtastic, on both the nt78 and RNASim-VS and nt78 datasets, in comparison to pplacer-SCAMPP and pplacer-taxtastic.

5.2 Results and Discussion

5.2.1 Experiment 6

In Experiment 6 we explored the behavior of pplacer(default) and pplacer-taxtastic on the simulated ROSE and nt78 datasets. On the 1,000-sequence ROSE 1000M1 and 1000M5 datasets (Figures 5.2 (a) and (b), respectively), we observed that pplacer(default) and pplacer-taxtastic had very close delta error, with perhaps a slight advantage to pplacer-default. We also note that pplacer (default) took slightly more time to run than pplacer-taxtastic. Recalling that the difference between pplacer(default) and pplacer-taxtastic is really a

difference in the software used to estimate numeric parameters (RAxML and FastTree-2 respectively), from this experiment we see that using RAxML numeric parameters lead to slightly more accurate placements than using FastTree-2 numeric parameters. As discussed earlier, this is not surprising since RAxML has better maximum likelihood estimates than FastTree-2, whereas FastTree-2 is more computationally efficient [21].

However, pplacer-default did not run on the nt78 dataset with 78,132 sequences (Figure 5.2(c)) while pplacer-taxtastic did succeed in running, showing a clear advantage when using pplacer-taxtastic.

Thus, the key difference is that changing the numeric parameter estimation software from RAxML to FastTree-2 while holding all else constant is indeed what allows pplacer to scale to larger datasets. This is consistent with previous results on the nt78 dataset [7], which also illustrated that given a FastTree-2 tree estimated under GTR+GAMMA, and RAxML v7.2.6 numeric parameters, pplacer(default) was unable to complete on this dataset due to a memory error.

Finally, both methods incurred lower delta error on the lower rate of evolution (1000M5) than on the higher rate of evolution (1000M1). Both methods consumed negligible memory. Thus, there was a clear advantage to using pplacer-taxtastic on large backbone trees where pplacer (default) fails to run.

5.2.2 Experiment 7

Experiment 7 explored the behavior of pplacer-taxtastic and pplacer-SCAMPP on the simulated nt78 and RNASim-50k datasets, using both full and fragmentary query sequences. On the RNASim-50K datasets (Figure 5.3), there were no noteworthy differences in delta error except for the shortest query sequences, where pplacer-taxtastic had lower delta error than pplacer-SCAMPP. However, pplacer-SCAMPP was consistently faster and used less memory than pplacer-taxtastic.

On the nt78 dataset (Figure 5.4), we saw bigger differences between the methods. Specifically, except for the full-length sequences (where both did very well), pplacer-taxtastic had much lower delta error than pplacer-SCAMPP (an improvement of several orders of magnitude). On the other hand, for all query lengths, pplacer-SCAMPP was much faster and consumed less memory than pplacer-taxtastic.

As mentioned earlier, there are two key differences between pplacer-taxtastic and pplacer-SCAMPP: firstly, pplacer-taxtastic uses FastTree-2 numeric parameters, whereas pplacer-SCAMPP uses RAxML numeric parameters. Secondly, pplacer-taxtastic places into the full backbone tree, whereas pplacer-SCAMPP places into a user-specified subtree of size B , where B is significantly smaller than the size of the full backbone

tree.

With this context, the results in Figure 5.4 and 5.3 are somewhat surprising. In Experiment 6, we observed that using RAxML numeric parameters led to slightly more accurate placements; this suggests that pplacer-SCAMPP would be more accurate than pplacer-taxtastic. Instead, as pplacer-taxtastic is more accurate than pplacer-SCAMPP, this suggests accuracy of the numeric parameters matters less than the ability to place into the full tree.

Although the relative accuracy of each method in Figure 5.4 and 5.3 are the same, the reason for the margin of improvement on the two different datasets is unclear. It is possible that the lower proportion of gaps in the nt78 dataset is responsible, but this requires further exploration.

5.2.3 Experiment 8

Experiment 8 evaluated pplacer-SCAMPP-tax using several different subtree sizes on the nt78 and RNASim datasets. Figure 5.5 (see also Table 5.1) illustrates pplacer-SCAMPP-tax's behavior as the setting for B changes (which changes the placement subtree sizes in the SCAMPP framework). We observed the accuracy-runtime trade-off as we increased the placement subtree size on both fragment query lengths. As pplacer-SCAMPP-tax incurred less delta error as the placement subtree size increased, but the runtime and memory consumption also increased. As expected, pplacer-SCAMPP-tax incurred greater delta error on the shorter fragment lengths, but the runtime and memory consumption on 0.10-fragment query sequences was less than on the longer 0.25-fragment query sequences. As discussed in a previous chapter, decreasing delta error as we increase the placement subtree size is a consistent trend, although we do not know why.

We saw that pplacer-SCAMPP-taxtastic(20k) uses a fraction of the runtime used by pplacer-taxtastic. All pplacer-SCAMPP-taxtastic variants tested definitively improved on the peak memory usage when compared to pplacer-taxtastic. Additionally, we saw that as the subtrees grew larger, the runtime and memory increased but the delta error decreased. We observed that pplacer-taxtastic still performed best in terms of the delta error from Table 5.2. However, we also see that among all variants of subtree size, pplacer-SCAMPP-taxtastic(40k) achieved similar delta error accuracy to pplacer-taxtastic, while consuming slightly less time and much less memory.

In Figure 5.6, we explored the same per-query results on large RNASim datasets. In particular, we looked at RNASim-100k and RNASim-200k with 100,000 and 200,000 leaves respectively. On each dataset, we compared variants of pplacer-SCAMPP-tax using different sizes of subtrees, pplacer-SCAMPP and pplacer-taxtastic. However, as we can observe from Panel (a), pplacer-SCAMPP suffered slightly higher delta error while pplacer-

taxtastic achieved slightly lower delta error on RNASim-100k. We observed that as the size of the placement subtree increases, the average delta error for pplacer-SCAMPP-tax decreased and the runtime increased. Likewise in Figure 5.6, decreasing delta error as we increase the placement subtree size is a consistent trend, although we do not know why.

In Panel (b) of Figure 5.6, we saw that all methods which could run performed quite similarly in terms of delta error. Notably, unfortunately pplacer-taxtastic was completely unable to run due to an out of memory error. In terms of runtime and memory consumption, we saw that pplacer-SCAMPP actually took more time (perhaps due to the memory constraints). Consistently, however, we saw that pplacer-SCAMPP-tax increased in runtime and memory consumption as the size of the placement subtree increased.

5.3 Conclusion

This chapter set out to understand two different techniques for scaling pplacer, a highly accurate phylogenetic placement method, to large backbone trees. The first of these techniques is taxtastic, which provides the numeric model parameters needed by pplacer, and the second of these techniques is SCAMPP, which uses divide-and-conquer. We provided conclusive evidence that both techniques are able to improve scalability (though SCAMPP is superior in this regard) and that SCAMPP also reduces runtime and memory usage (but this is not true for taxtastic). The two techniques have different impacts on placement accuracy, with taxtastic typically not changing accuracy but SCAMPP sometimes reducing accuracy compared to pplacer-taxtastic (when used with its default setting for $B = 2,000$, which allows it only to place into subtrees of size 2,000). This shows the benefit of being able to place into the full backbone tree (achieved by pplacer-taxtastic) compared to only placing into a (relatively small) placement tree when using pplacer-SCAMPP. However, we note that the restriction to placement subtrees with only 2,000 leaves in pplacer-SCAMPP is due to pplacer's tendency to fail or have poor accuracy on larger backbone trees, due to numerical issues. We also note that pplacer-taxtastic does not have this limitation, as our study shows it can be used with very large backbone trees (seemingly up to 100,000 sequences).

These observations (and most importantly on the ability to use pplacer-SCAMPP on larger placement subtrees) led to the design of a combined method, pplacer-SCAMPP-taxtastic, which follows the SCAMPP pipeline but replaces default pplacer by pplacer-taxtastic. Furthermore, because pplacer-taxtastic can be used with large backbone trees, this replacement allows pplacer-SCAMPP-taxtastic to be used with larger settings for the placement subtree size B .

Our exploration shows that pplacer-SCAMPP-taxtastic provides the best features of both techniques for improving scalability of pplacer, achieving the same scalability as pplacer-SCAMPP (which has been shown to scale to backbone trees with 200,000 sequences, matching the scalability of APPLES-2), and matching (and in some cases improving) on the placement accuracy of pplacer-SCAMPP when the parameter B is increased. Moreover, this approach allows the user to select the value for B based on preferences: when accuracy is more important, increasing B (perhaps as a function of the backbone tree size) can be helpful, but when speed and memory usage is more important, keeping B small (e.g., $B = 2,000$) can be most desirable.

5.4 Tables and Figures

Dataset	num seqs	avg. align length	fragment basepairs	p-distance average	p-distance maximum	prop gaps	used reps
nt78 [39]	78,132	1,287	385/154	.404	.639	.006	1
RNASim-50k [28]	50,000	1,620	322/127	.410	.618	.051	5
RNASim-100k [28]	100,000	1,620	322/127	.410	.618	.051	1
RNASim-200k [28]	200,000	1,620	322/127	.410	.618	.051	1
ROSE 1000M1 [21]	1,000	3,880	N.A.	.690	.765	.738	5
ROSE 1000M5 [21]	1,000	1,690	N.A.	.496	.609	.426	5

Table 5.1: **Nucleotide Dataset Statistics:** We provide details for all the datasets used to study placement methods. All used datasets are publicly available. The first column provides the dataset name. For each dataset, we provide the number of sequences, average alignment length, type of alignment, p-distance (normalized Hamming distance) average and maximum, proportion of gaps and number of replicates. These are all calculated on the replicates used. For the column illustrating fragment basepairs, we list 0.25- then 0.10-fragment lengths. These are listed as N.A. or “not applicable” for the ROSE dataset, since we only evaluated the full-length sequences when studying placement methods. These fragments were generated using Fragmentary Protocol B described in Section 5.1.2.

Method	delta error	runtime (s)	peak memory (GB)
pplacer-SCAMPP-taxtastic (2k)	6.83	27.50	0.09
pplacer-SCAMPP-taxtastic (10k)	3.86	79.43	0.63
pplacer-SCAMPP-taxtastic (20k)	2.30	164.88	1.79
pplacer-SCAMPP-taxtastic (30k)	1.25	358.25	2.68
pplacer-SCAMPP-taxtastic (40k)	0.90	482.88	3.53
pplacer-SCAMPP	6.91	14.65	0.02
pplacer-taxtastic	0.69	323.86	6.46

Table 5.2: Summary of results from Experiment 8, when placing 0.25-fragment query sequences into the nt78 dataset (see Figure 5.5).

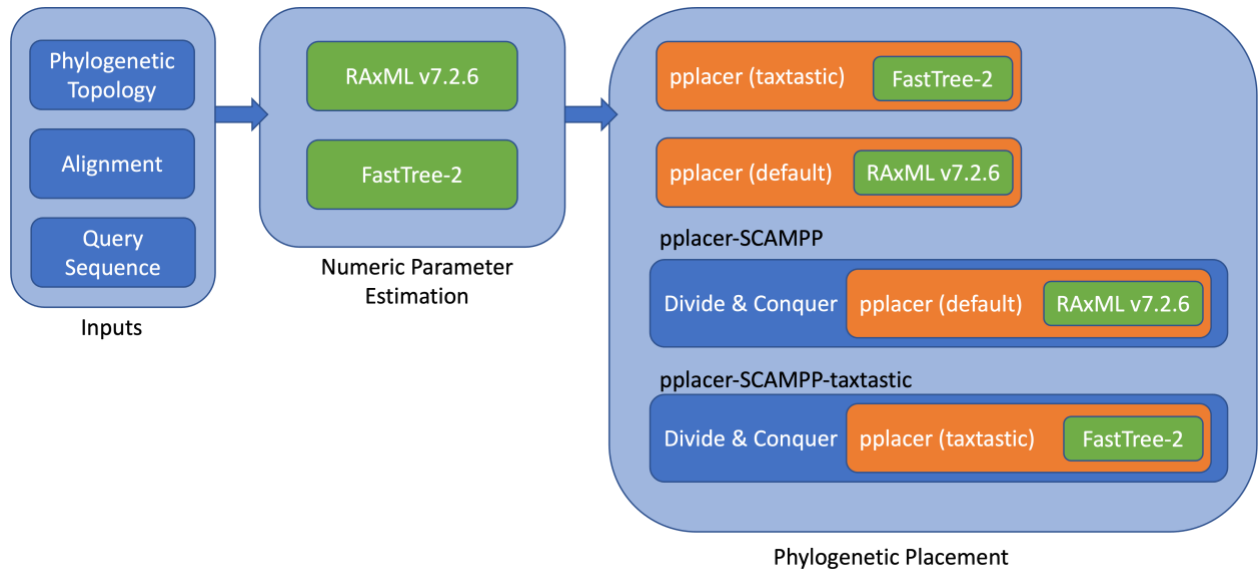


Figure 5.1: **(Placement Pipelines)** The input is a backbone tree (“phylogenetic topology”), an alignment on the backbone sequences, and an aligned query sequence. In the first step, we estimate the numeric parameters on the backbone tree using a selected maximum likelihood code (i.e., RAxML 7.2.6 or FastTree 2.). We then use the specified phylogenetic placement method (i.e., pplacer, pplacer-SCAMPP, or pplacer-SCAMPP-taxtastic). Each of these has a specific technique that is recommended for the previous step (i.e., numeric parameter estimation).

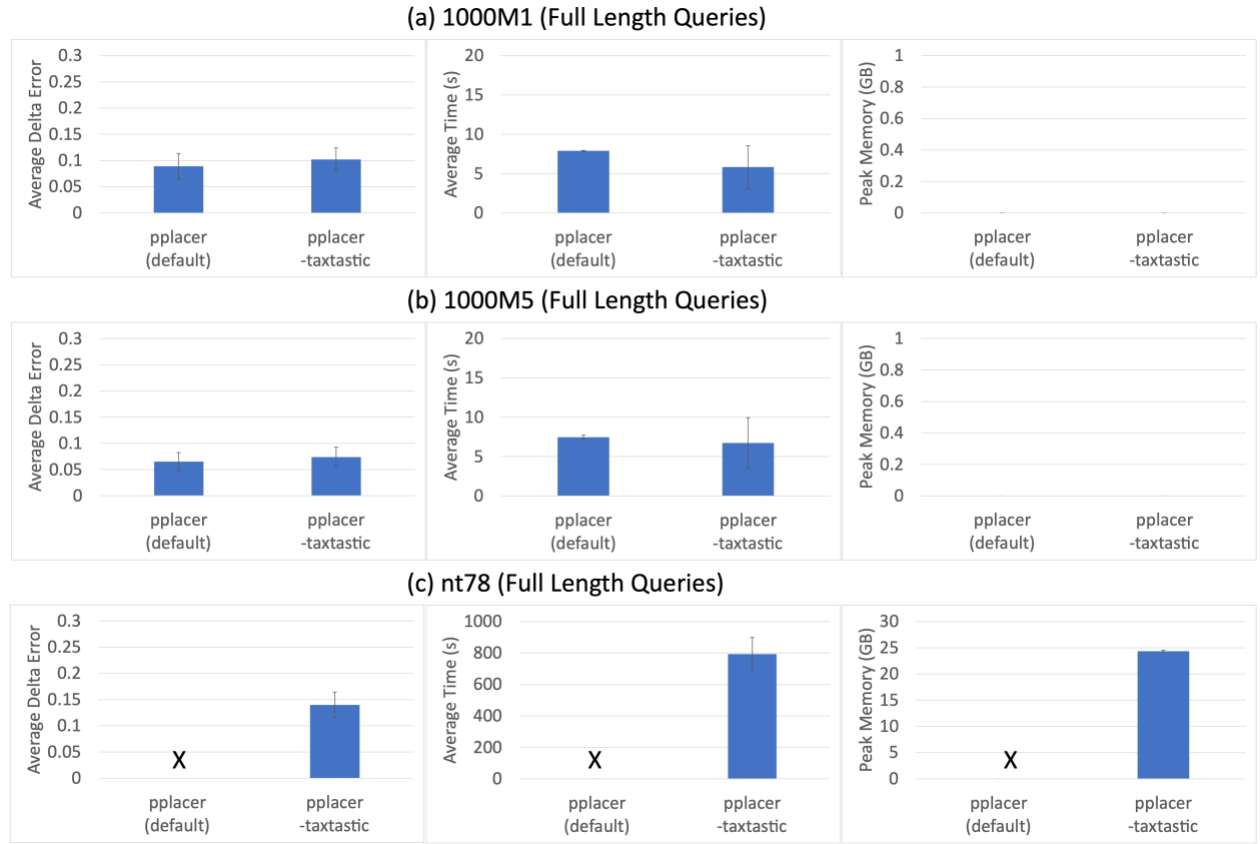


Figure 5.2: (Experiment 6) Comparison between pplacer(default) and pplacer-taxtastic on the ROSE and nt78 datasets. (a) Results on the ROSE 1000M1 datasets, (b) Results on the ROSE 1000M5 datasets, and (c) Results on one replicate of the nt78 dataset containing 78,132 sequences. The left two columns show per-query delta error results, the middle two columns show average per-query runtime to place each sequence, and the two rightmost columns illustrate per-query memory consumption. For delta error we present standard error, and for runtime and memory consumption we present standard deviation. Note that the runtime and memory usage y-axes are different between the ROSE and nt78 datasets. The ROSE 1000M1 and 1000M5 datasets contain 1,000 sequences and 5 replicates each. 1000M1 has a high rate of evolution and 1000M5 has a low rate of evolution. The “X” indicates that pplacer (default) failed with an out of memory error on these experiments. Peak memory usage on the ROSE datasets is negligible, so that panels (a) and (b) do not show visible memory usage for either method.

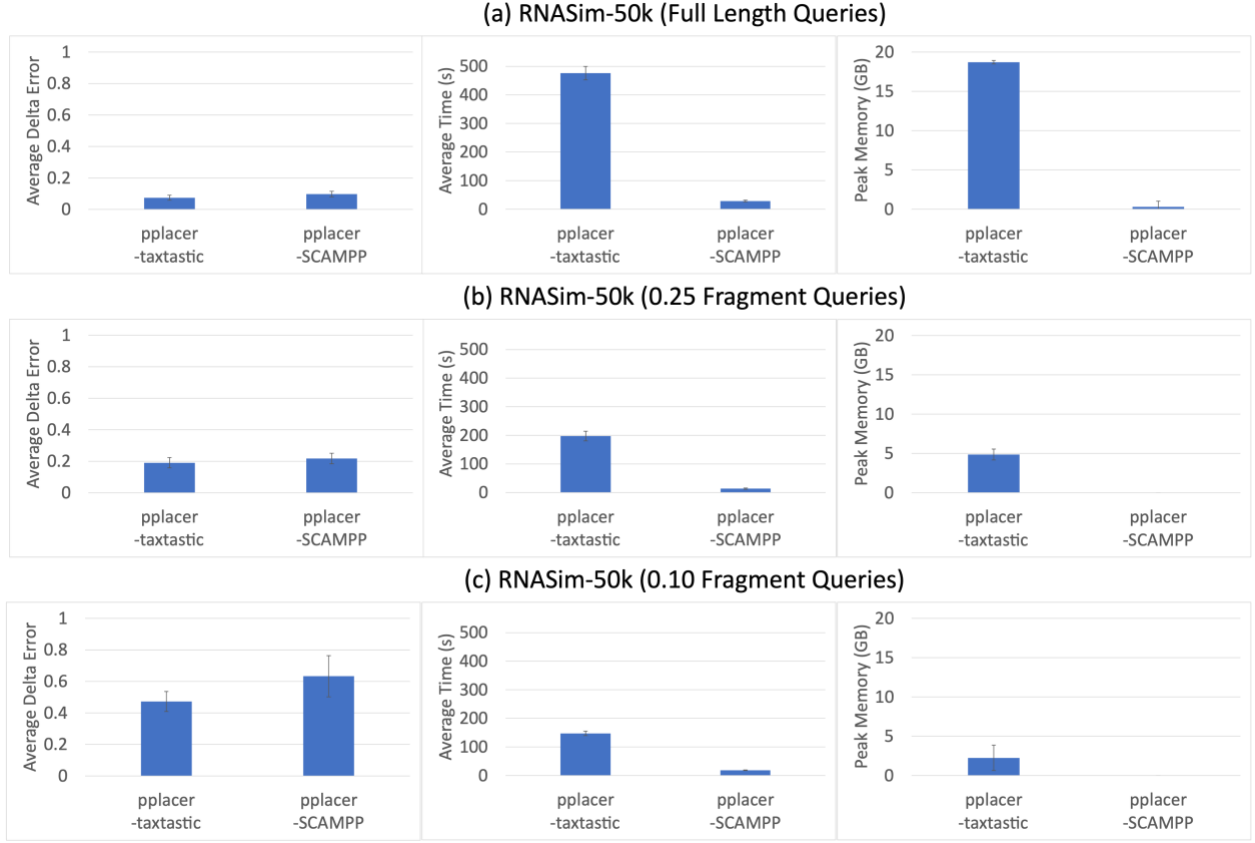


Figure 5.3: (Experiment 7) Comparison between pplacer-SCAMPP and pplacer-taxtastic on RNASim-VS datasets using both full and fragmentary query sequences. Per-query results on the fragmentary RNASim (simulated) dataset containing 50,000 sequences. Panel (a) shows results on full-length sequences, panel (b) shows results on 0.25-fragment length (385 basepairs) and panel (c) shows results on 0.10-fragment length (154 basepairs). For all panels, the first column shows per-query delta error results, the second column shows average per-query runtime to place each sequence and the third column illustrates per-query memory consumption. For delta error we presented average standard error, and for runtime and memory consumption we presented standard deviation. Please note that the value for B , which determines the placement subtree size for pplacer-SCAMPP, was set to the default ($B = 2,000$).

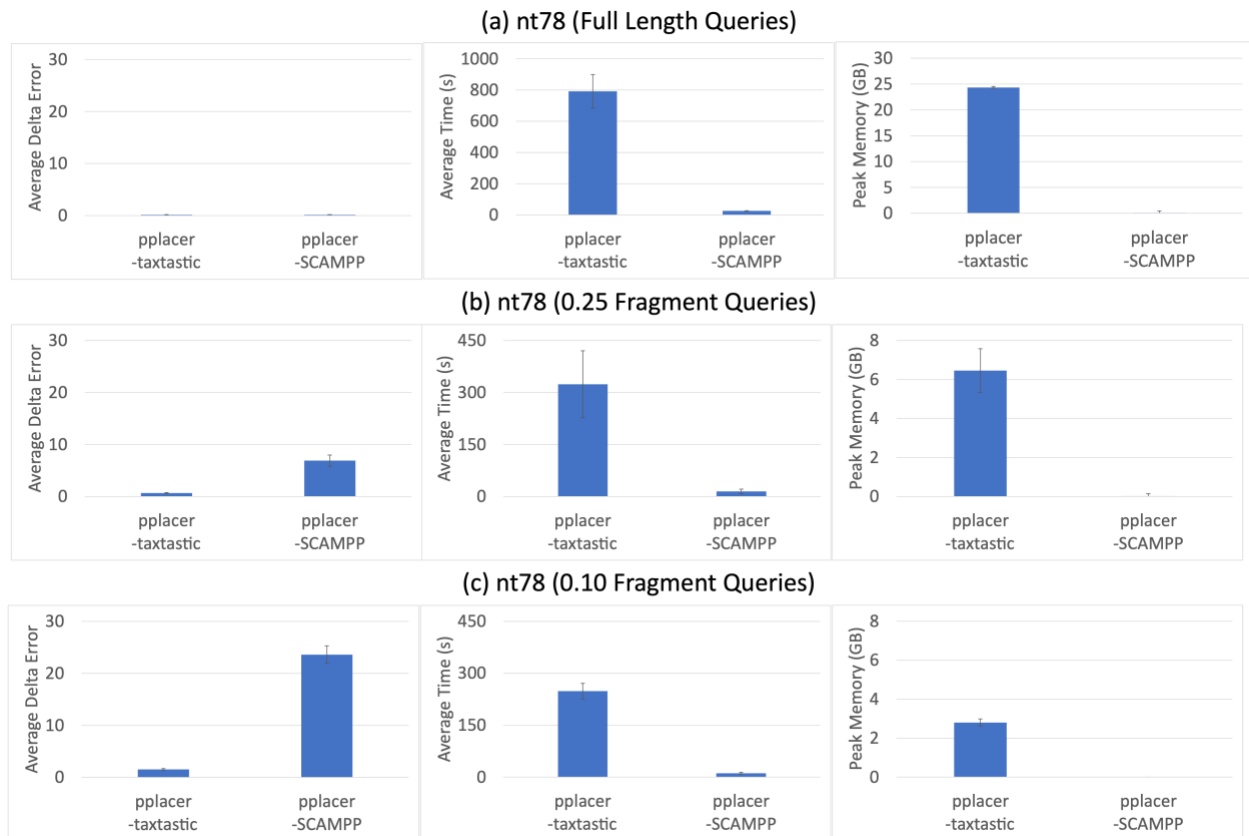


Figure 5.4: (Experiment 7) Comparison between pplacer-SCAMPP and pplacer-taxtastic on the nt78 datasets when placing full and fragmentary query sequences. Per-query results on the nt78 (simulated) dataset containing 78,132 sequences. Panel (a) shows results on full-length sequences, panel (b) shows results on 0.25-fragment length (322 basepairs), and panel (c) shows results on 0.10-fragment length (127 basepairs). For all panels, the first column shows per-query delta error results, the second column shows average per-query runtime to place each sequence and the third column illustrates per-query memory consumption. For delta error we presented average standard error, and for runtime and memory consumption we presented standard deviation. **Note that the y-axes are different for runtime and memory usage between the full-length and fragmentary datasets.**

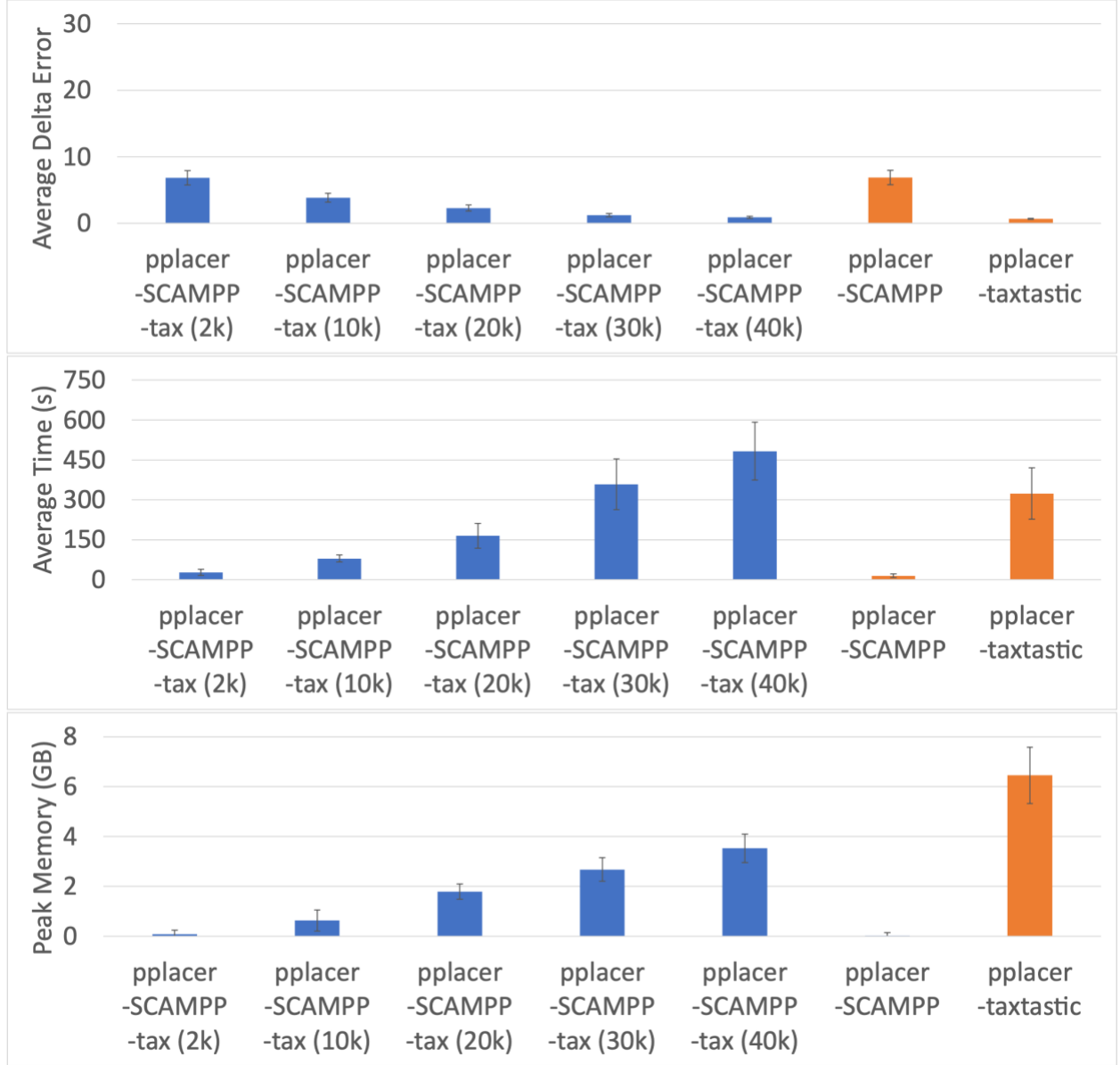
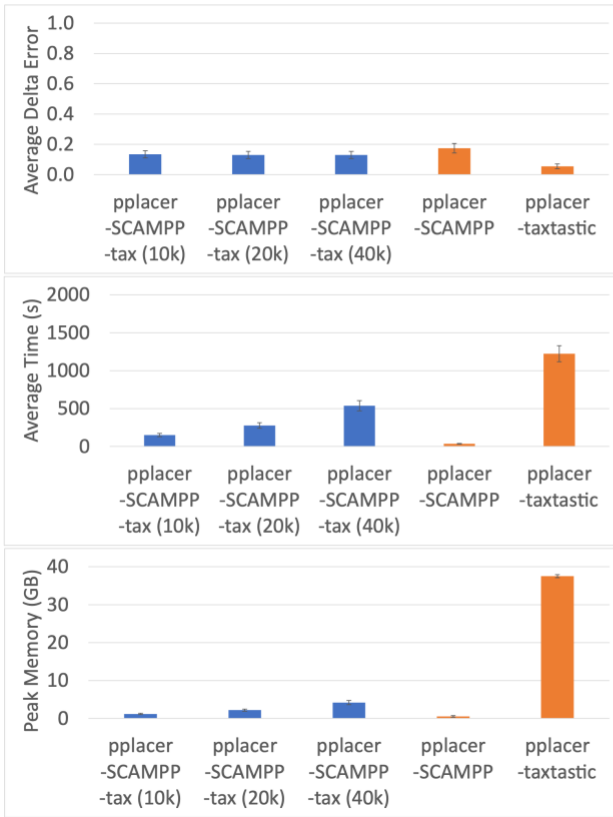


Figure 5.5: **(Experiment 8) Comparison between pplacer-SCAMPP-taxtastic, pplacer-SCAMPP and pplacer-taxtastic when varying subtree placement size and placing 0.25-fragment length (322 basepairs) query sequences into the nt78 tree.** Per-query results on the nt78 (simulated) dataset containing 78,132 sequences. The first row shows per-query delta error results, the second row shows average per-query runtime to place each sequence and the third row illustrates per-query memory consumption. For delta error we present average standard error, and for runtime and memory consumption we present standard deviation. Please note that the y-axes are different between the full-length and fragmentary datasets. Along the x-axis the methods are sorted by the placement subtree size B , given parenthetically. Specific numbers are shown for the 0.10-fragmentary case in Table 5.2.

(a) RNASim-100k (0.25 Fragment Queries)



(b) RNASim-200k (0.25 Fragment Queries)

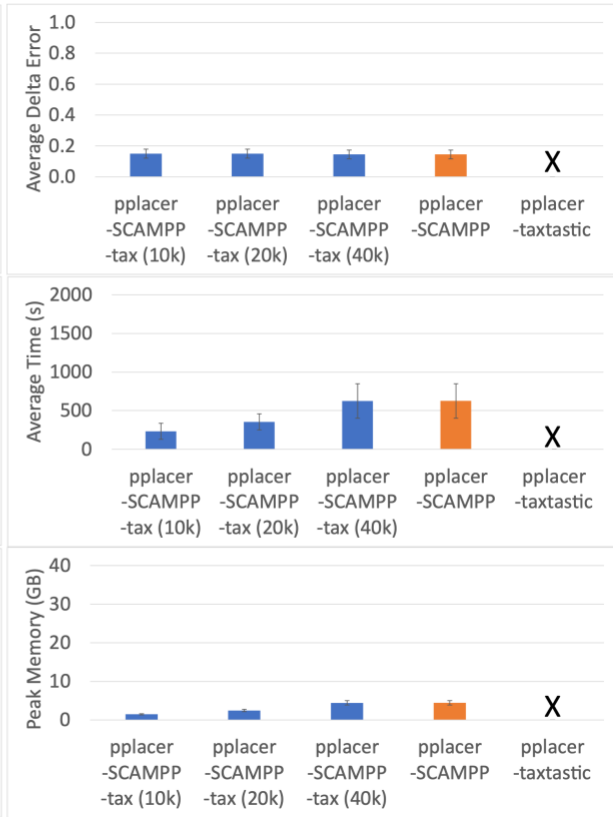


Figure 5.6: **(Experiment 8) Comparison between pplacer-SCAMPP-taxtastic, pplacer-SCAMPP and pplacer-taxtastic when varying subtree placement size and placing fragmentary query sequences in the RNASim-100K and RNASim-200K datasets.** Per-query results placing 0.25 fragmentary query sequences into the RNASim-100k and RNASim-200k (simulated) datasets. Panel (a) shows results on the 100,000-leaf backbone tree and panel (b) shows results on the 200,000-leaf backbone tree. For both panels, the first row shows per-query delta error results, the second row shows average-per-query runtime to place each sequence and the third row illustrates per-query memory consumption. For delta error we presented average standard error, and for runtime and memory consumption we presented standard deviation. pplacer-taxtastic failed with an out of memory error (64 GB provided to each placement job) for each of the 200 query sequences for the RNASim-200k dataset.

CHAPTER 6: CONCLUSIONS

In this thesis, we aimed to generate several recommendations to improve the pipeline of phylogenetic placement — both in terms of scalability and accuracy. Chapters 3 and 4 evaluated several extended alignment methods on the task of aligning a fragmentary query sequence to a simulated backbone alignment containing small and large numbers of sequences. We found that on small datasets, the extended alignment method UPP performed best-in-class in terms of delta error, runtime and memory consumption.

In Chapter 4 we also began to explore the scalability of the placement method pplacer on large backbone phylogenies. Importantly, we found that UPP was the only method able to place into extremely large backbone alignments (> 50,000 sequences). Furthermore, although within the literature pplacer has been inconsistent in its ability to place into large phylogenies containing > 5,000 sequences [7] [5], here we found that when provided with enough memory pplacer was able to run on RNASim trees containing up to 50,000 sequences.

In Chapter 5 we explored existing techniques for scaling placement methods and proposed a new approach which combines techniques to perform accurate and scalable phylogenetic placement. We evaluated several placement methods to better understand how different techniques achieve scalability, and how each technique affects final phylogenetic placement error. Our study showed that our newly proposed approach of pplacer-SCAMPP-taxtastic achieved both the scalability of the best-in-class placement methods (scaling to 200,000 sequences), and improved on the placement accuracy of pplacer-SCAMPP. This new approach leverages pplacer-SCAMPP's divide-and-conquer framework in addition to using FastTree-2 numeric parameters (reformatted using the taxtastic package). Within pplacer-SCAMPP-taxtastic, the user can select the value B for a placement subtree size based on user preferences; increasing B generates more accurate placements, but when speed and memory usage are the limiting factors, keeping B small (e.g., $B = 2,000$) is best.

Therefore, our final recommendation for phylogenetic placement pipelines is to use UPP to first align a query sequence to the backbone reference alignment, then to use pplacer-SCAMPP-taxtastic to place that query into the tree.

Chapters 3 and 4 leave some work to be done. Exploring the delta error accuracy when using extended alignment methods to align the shortest fragments (0.10, as done in Chapter 5) would lead to more complete results. Additionally, the insights that mafft-linsi is highly accurate but not scalable suggest an avenue of research which scales mafft-linsi. Some work exploring this approach has already been done to great

success [41].

Although it is not relevant to our biological application of aligning short fragmentary reads to an existing reference database, it would be algorithmically interesting to evaluate whether the same trends govern these extended alignment methods when given long query sequences (longer than the reference sequences). Within Chapter 3, we observed that using the `-add` flag seemed to change the runtime across model conditions and replicates, whereas using the `-addfragmentary` flag had low variability. One hypothesis to explain this higher variance could be that the `-add` flag considers the relationship between the query sequences, whereas the `-addfragmentary` flag does not. In the context of long query sequences, this suggests that the relationship between the input long query sequences may improve accuracy.

Chapter 5 suggests several avenues for further research. Perhaps the most intriguing question is why `taxtastic` benefits `pplacer`. It improves scalability to larger backbone trees (though it still fails on the largest backbone trees we examined, with 200,000 sequences). In some cases, using `taxtastic` also improves accuracy. Understanding the conditions where it improves accuracy is certainly of interest, but understanding why it consistently improves scalability is also important. We hypothesize that using `FastTree` to estimate numeric parameters overcomes the numerical instability in `RAxML` (noted in [7]), which would allow `pplacer` to scale to larger backbone trees. This may also help it to improve accuracy on backbone trees where likelihood scores become very large in magnitude. However, how `taxtastic` achieves this is unclear. As `taxtastic` can be used with both `RAxML` and `FastTree-2` numeric parameters, one particularly interesting question is whether the improvement in delta error should be attributed to the `FastTree-2` numeric parameters or whether it should be attributed to the `taxtastic` package.

This exploration also suggests that increasing B , which specifies the placement subtree size within the `pplacer-SCAMPP-taxtastic` framework, can actually improve accuracy, albeit at the cost of slower runtime and higher memory usage. However, the impact of this choice depends on the dataset, with notable accuracy benefits obtained in some cases but none in others (e.g., compare `nt78` to `RNASim-VS`). At this time, it is unclear why there is such a difference between model conditions, and in particular there is no obvious empirical property (such as average p -distance) that would suggest the use of a larger value for B . Future work is needed to understand this issue, so as to inform practice.

One interesting direction for future work is to explore the impact of different tree shapes on the various placement methods. A method like `SCAMPP`, which selects a placement subtree out of the provided backbone tree [34], will likely be affected by the shape of the backbone tree (e.g., pectinate, complete balanced binary tree) and different random models (e.g., birth-death, Yule).

We did not use biological datasets to evaluate placement accuracy. This contrasts with some studies (e.g., [42]) that used biological datasets, and calculated placement error with respect to a “reference tree” for the species set, which is actually based on an estimated tree. Because estimated trees have errors, we do not consider topological distance to an estimated tree to be a reliable indication of placement accuracy. Moreover, with the exception of species trees for certain subsets of species, the true evolutionary tree is rarely known with confidence. Importantly, gene trees are less reliably known than species trees, given the limited phylogenetic signal in any gene alignment. Given discordance between gene trees and species trees, using a species tree as the “reference” is also problematic for evaluating phylogenetic placement accuracy. For this reason, we used a variety of simulated model conditions to explore phylogenetic placement methods.

In future work, rather than using estimated binary trees as a pseudo-reference tree, it would be better to use only the highly supported edges in the estimated maximum likelihood tree as the reference tree (e.g., as done in [24]). However, one challenge here is that computing very large maximum likelihood trees with branch support is challenging for the dataset sizes we examine here (e.g., 50,000 sequences and more). Thus, future work is necessary to develop rigorous and feasible ways of using biological datasets to evaluate phylogenetic placement methods.

A clear next step is to adopt the work in all chapters in order to improve existing pipelines for the downstream applications of phylogenetic placement: abundance profiling and taxonomic identification (e.g., TIPP2 [4]). Future work should evaluate how these recommended methodological changes affect scalability and accuracy in these downstream applications.

REFERENCES

- [1] G. Chu and T. Warnow, "Exploring Trade-offs in Scalable Phylogenetic Placement Methods," *bioRxiv*, 2022, [Online]. Available: : <https://www.biorxiv.org/content/10.1101/2022.05.23.493012v1>.
- [2] V. Smirnov and T. Warnow, "Phylogeny Estimation Given Sequence Length Heterogeneity," *Systematic Biology*, vol. 70, no. 2, pp. 268-282, 2021.
- [3] N.-p. Nguyen, S. Mirarab, B. Liu, M. Pop, and T. Warnow, "TIPP: Taxonomic identification and phylogenetic profiling," *Bioinformatics*, vol. 30, no. 24, pp. 3548-3555, 2014.
- [4] N. Shah, E. K. Molloy, M. Pop, and T. Warnow, "TIPP2: Metagenomic taxonomic profiling using phylogenetic markers," *Bioinformatics*, vol. 37, no. 13, pp. 1839-1845, 2021, DOI: 10.1093/bioinformatics/btab023.
- [5] M. Balaban, S. Sarmashghi, and S. Mirarab, "APPLES: Scalable Distance-Based Phylogenetic Placement with or without Alignments," *Systematic Biology*, vol. 69, no. 3, pp. 566-578, 2020.
- [6] S. Mirarab, N. Nguyen, and T. Warnow, "SEPP: SAT'e-enabled Phylogenetic Placement," in *Biocomputing*, World Scientific, 2012, pp. 247-258.
- [7] E. Wedell, Y. Cai, and T. Warnow, "SCAMPP: Scaling Alignment-based Phylogenetic Placement to Large Trees," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2022, DOI:10.1109/TCBB.2022.3170386.
- [8] D. J. Zwickl and D. M. Hillis, "Increased taxon sampling greatly reduces phylogenetic error," *Systematic Biology*, vol. 51, no. 4, pp. 588-598, 2002.
- [9] D. D. Pollock, D. J. Zwickl, J. A. McGuire, and D. M. Hillis, "Increased taxon sampling is advantageous for phylogenetic inference," *Systematic Biology*, vol. 51, no. 4, p. 664, 2002.
- [10] H. Philippe and M. J. Telford, "Large-scale sequencing and the new animal phylogeny," *Trends in Ecology & Evolution*, vol. 21, no. 11, pp. 614-620, 2006.
- [11] M. S. Rosenberg and S. Kumar, "Incomplete taxon sampling is not a problem for phylogenetic inference," *Proceedings of the National Academy of Sciences*, vol. 98, no. 19, pp. 10751-10756, 2001.
- [12] M. Blanke and B. Morgenstern, "App-SpaM: Phylogenetic placement of short reads without sequence alignment," *Bioinformatics Advances*, vol. 1, no. 1, vbab027, 2021.

- [13] B. Linard, K. Swenson, and F. Pardi, "Rapid alignment-free phylogenetic identification of metagenomic sequences," *Bioinformatics*, vol. 35, no. 18, pp. 3303–3312, 2019.
- [14] S. Dodsworth, "Genome skimming for next-generation biodiversity analysis," *Trends in Plant Science*, vol. 20, no. 9, pp. 525–527, 2015.
- [15] M. Balaban, Y. Jiang, D. Roush, Q. Zhu, and S. Mirarab, "Fast and Accurate Distance-based Phylogenetic Placement using Divide and Conquer," *Molecular Ecology Resources*, vol. 22, no. 3, pp. 1213–1227, 2021.
- [16] S. C. Potter, A. Luciani, S. R. Eddy, Y. Park, R. Lopez, and R. D. Finn, "HMMER web server: 2018 update," *Nucleic Acids Research*, vol. 46, no. W1, W200–W204, 2018.
- [17] N.-p. D. Nguyen, S. Mirarab, K. Kumar, and T. Warnow, "Ultra-large alignments using phylogeny-aware profiles," *Genome Biology*, vol. 16, no. 1, pp. 1–15, 2015.
- [18] K. Katoh and D. M. Standley, "MAFFT multiple sequence alignment software version 7: Improvements in performance and usability," *Molecular Biology and Evolution*, vol. 30, no. 4, pp. 772–780, 2013.
- [19] P. Barbera, A. M. Kozlov, L. Czech, *et al.*, "EPA-ng: Massively parallel evolutionary placement of genetic sequences," *Systematic Biology*, vol. 68, no. 2, pp. 365–369, 2019.
- [20] Y. Jiang, M. Balaban, Q. Zhu, and S. Mirarab, "DEPP: Deep learning enables extending species trees using single genes," *Systematic Biology*, 2022, DOI:10.1093/sysbio/syac031.
- [21] K. Liu, C. R. Linder, and T. Warnow, "RAxML and FastTree: Comparing Two Methods for Large-Scale Maximum Likelihood Phylogeny Estimation," *PloS One*, vol. 6, no. 11, e27731, 2011.
- [22] J. Stoye, D. Evers, and F. Meyer, "ROSE: Generating sequence families.," *Bioinformatics (Oxford, England)*, vol. 14, no. 2, pp. 157–163, 1998.
- [23] M. J. Sanderson, "R8s: Inferring absolute rates of molecular evolution and divergence times in the absence of a molecular clock," *Bioinformatics*, vol. 19, no. 2, pp. 301–302, 2003.
- [24] K. Liu, S. Raghavan, S. Nelesen, C. R. Linder, and T. Warnow, "Rapid and Accurate Large-Scale Coestimation of Sequence Alignments and Phylogenetic Trees," *Science*, vol. 324, no. 5934, pp. 1561–1564, 2009.
- [25] K. Liu, T. J. Warnow, M. T. Holder, *et al.*, "SATE-II: Very fast and accurate simultaneous estimation of multiple sequence alignments and phylogenetic trees," *Systematic Biology*, vol. 61, no. 1, p. 90, 2012.
- [26] C. Shen, P. Zaharias, and T. Warnow, "MAGUS+eHMMs: Improved multiple sequence alignment accuracy for fragmentary sequences," *Bioinformatics*, 2021, DOI: 10.1093/bioinformatics/btab788.

- [27] M. Park, S. Ivanovic, G. Chu, C. Shen, and T. Warnow, "UPP2: Fast and Accurate Alignment Estimation of Datasets with Fragmentary Sequences," *bioRxiv*, 2022. [Online]. Available: <https://www.biorxiv.org/content/10.1101/2022.02.26.482099>.
- [28] S. Mirarab, N. Nguyen, S. Guo, L.-S. Wang, J. Kim, and T. Warnow, "PASTA: Ultra-Large Multiple Sequence Alignment for Nucleotide and Amino-Acid Sequences," *Journal of Computational Biology*, vol. 22, no. 5, pp. 377–386, 2015.
- [29] V. Smirnov and T. Warnow, "MAGUS: Multiple sequence Alignment using Graph clUstering," *Bioinformatics*, vol. 37, no. 12, pp. 1666–1672, 2021.
- [30] E. Koning, M. Phillips, and T. Warnow, "pplacerDC: A new scalable phylogenetic placement method," in *Proceedings of the 12th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics*, 2021, pp. 1–9.
- [31] S. R. Eddy, "Profile hidden Markov models,," *Bioinformatics (Oxford, England)*, vol. 14, no. 9, pp. 755–763, 1998.
- [32] S. R. Eddy, "Accelerated profile HMM searches," *PLoS Computational Biology*, vol. 7, no. 10, e1002195, 2011.
- [33] K. Katoh and M. C. Frith, "Adding unaligned sequences into an existing alignment using MAFFT and LAST," *Bioinformatics*, vol. 28, no. 23, pp. 3144–3146, 2012.
- [34] F. A. Matsen, R. B. Kodner, and E. V. Armbrust, "pplacer: Linear time maximum-likelihood and Bayesian phylogenetic placement of sequences onto a fixed reference tree," *BMC Bioinformatics*, vol. 11, no. 1, pp. 1–16, 2010.
- [35] S. Tavar'e, "Some probabilistic and statistical problems in the analysis of DNA sequences," *Lectures on Mathematics in the Life Sciences*, vol. 17, no. 2, pp. 57–86, 1986.
- [36] L. Czech, P. Barbera, and A. Stamatakis, "Methods for automatic reference trees and multilevel phylogenetic placement," *Bioinformatics*, vol. 35, no. 7, pp. 1151–1158, 2019.
- [37] Fred Hutchinson Cancer Research Center, *Taxtastic*, May 20, 2022. [Online]. Available: <http://fhcrc.github.io/taxtastic/>.
- [38] G. Chu and T. Warnow, *RNASim-VS2*, 2022. [Online]. Available: https://doi.org/10.13012/B2IDB-8812049_V1.
- [39] 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, e9490, 2010.

- [40] M. Hasegawa, H. Kishino, and T.-a. Yano, "Dating of the human-ape splitting by a molecular clock of mitochondrial DNA," *Journal of Molecular Evolution*, vol. 22, no. 2, pp. 160–174, 1985.
- [41] C. Shen, B. Liu, and T. Warnow, "eMAFFTadd: Scaling MAFFT-linsi-add to large datasets," *bioRxiv*, 2022. [Online]. Available: <https://doi.org/10.1101/2022.05.23.493139>.
- [42] B. Linard, N. Romashchenko, F. Pardi, and E. Rivals, "PEWO: A collection of workflows to benchmark phylogenetic placement," *Bioinformatics*, vol. 36, no. 21, pp. 5264–5266, 2021.

APPENDIX A

A.1 Shared

The scripts to calculate delta error were adapted from scripts published in the APPLES-2 paper and SCAMPP paper, which borrow utilities from the newick_utils package. We used Mike Nute’s AlignmentClass scripts to calculate p-distance statistics, which can be found at <https://wiki.illinois.edu/wiki/display/warnowlab/Get+alignment+statistics> or https://www.dropbox.com/s/wqt8hvpqkdjge04/alignment_stats.zip?dl=0.

A.2 Extended Alignments

Given a query sequence and a backbone alignment and tree, the pipeline steps are:

1. Pre-Processing: Construct backbone alignments and subsequently estimate trees.

- True Trees: We used the “true” alignment and “true” tree. We estimated the numerical parameters using RAxML.

- Estimated Backbone Alignment:

We used MAGUS to estimate a backbone alignment with the following command:

```
python3 magus.py -np 16
-i <unaligned file> -d <output dir>
-o <output alignment>
--recurse false
```

- Estimated Backbone Tree: We used the best tree found by RAxML-NG v. 1.0.3 in four hours.

```
${raxml} --model GTR+G --msa $1 --prefix $2
```

- Numeric Parameters on Estimated Tree: We used the model parameters found by RAxML version 7.2.4.

```
${raxml} -m GTRGAMMA -F -n $3 -s $1 -T 2 -g $2 > $4
```

2. Alignment: Add query sequence into the backbone alignment. We tested with UPP and the MAFFT variants (default, linsi and 6mer), with (add, addfragmentary). We only used add and addfrag to place fragmentary sequences.

Each method was given as input (1) the unaligned fragmentary sequence fasta file, (2) true PASTA backbone alignment and (3) the backbone tree (for UPP, but not for MAFFT). It output an alignment with backbone and query sequences.

(a) MAFFT v7.487 (2021/Jul/25)

```
mafft / -linsi / -6mer
--thread 16
--quiet
--addfragments / --add
${input_unaln}
${bb_aln}
> ${output_aln}
```

(b) UPP 4.5.1: We ran UPP in default mode with the following command:

```
python3 ${upp} -s ${input_unaln} -x 1 -p
${bb_aln}
-o ${output_aln}
-t ${bb_tree}
```

3. Phylogenetic Placement: To place the query sequence into the backbone tree, we used pplacer v1.1.alpha13.

```
${pplacer}
-t ${ref_tree}
-r ${ref_aln}
-s ${tinfo}
-o ${out_json}
```

```
        --timing
    ${queryfas}
```

A.3 Running pplacer On Large Backbones

To create a taxtastic package, we run the following command:

```
taxit create -P ${my_refpkg}
-l ${locus_name}
--aln-fasta ${ref_fa}
--tree-file ${fasttree_nwk}
--tree-stats ${fasttree_log}
```

To run pplacer, we run with the following flags:

```
pplacer-64 -m GTR -c <refpkg>
-o <output.jplace>
<true_ext_aln.fa>
-j 1 --timing
--no-pre-mask (optional)
<logfile>
```

In Experiment 6 we estimated trees and numeric parameters using FastTree-2. FastTreeMP-nt-gtr

```
-nome-mllen-log fasttree.log-intree
reference.nwk< alignment.fasta
> fasttree.nwk—
```

A.4 Running Placement Methods

We provide commands for running pplacer-taxtastic, pplacer-SCAMPP, and pplacer-SCAMPP-taxtastic. We use the FastTree-2 results (tree and numeric parameters estimated under Minimum Evolution) on the nt78 datasets published in the SCAMPP paper.

A.4.1 Numeric Parameters

For pplacer-SCAMPP on the ROSE datasets, we compute branch lengths using RAxML (v7.2.8).

- RAxML-7.2.8-ALPHA/raxmlHPC-PTHREADS-f e -greference.nwk-m GTRGAMMA
-s alignment.phylip-n outname-p 1984-T 16-w outdir/

For all datasets, we compute branch lengths using FastTree-2 in order to run pplacer-taxtastic.

- FastTreeMP-nosupport-gtr -gamma-nt-log reestimated.fasttree.log
-intree intree.fasta< alignment.fasta> reestimated.fasttree.tree

A.4.2 Placement Methods

pplacer-SCAMPP (v2.0.0, Accessed: May 10, 2022) <https://github.com/ewedell/SCAMPP>

There may also now be a newer version of pplacer-SCAMPP, which handles fragmentary query sequences slightly more quickly. Experiments 3 (RNASim) and 4 were run with this newer version containing a potential fix. On the RNASim-200k dataset, we use the FastTree estimated branch lengths published in the SCAMPP paper. As all subsets of the RNASim-VS dataset come from the same million-sequence tree, we expect the numeric parameters estimated on the RNASim-100k dataset containing 100,000 sequences to be quite similar to the numeric parameters for RNASim-200k. As no RAxML v. 7.2.6 info file or estimated tree was published with the APPLES-2 paper, we use the numeric parameter file for the RNASim-100k R0 dataset. Please note that “jobID” is only necessary if running several jobs at once.

- pplacer-SCAMPP.py-i tree_info.log-t backbone_pp.tree-d out_dir/ -a full_msa.aln-b 2000-n jobID

pplacer-taxtastic

- taxtastic/taxtastic-env/bin/taxit create -P ng-my.refpkg
-l name --aln-fast ref.fa --tree-file backbone_pp.tree
--tree-stats reestimated.fasttree.log

pplacer-SCAMPP-taxtastic <https://github.com/gillichu/PLUSplacer-taxtastic>

- python pplacer-tax-SCAMPP.py -i reestimated.fasttree.log

```
-t backbone_pp.tree -d outdir -q query.fa  
-b 2000 -o jplace_name -a ref.fa  
-r ref.fa -n 1563
```