© 2024 Baqiao Liu

EFFICIENT AND SCALABLE ALGORITHMS FOR PHYLOGENOMICS AND OVERLAPPING COMMUNITIES

BY

BAQIAO LIU

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois Urbana-Champaign, 2024

Urbana, Illinois

Doctoral Committee:

Professor Tandy Warnow, Chair
Professor Lawrence Rauchwerger
Assistant Professor Mohammed El-Kebir
Associate Professor Hanghang Tong
Professor David A. Bader, New Jersey Institute of Technology

Abstract

In computational biology, the reconstruction of evolutionary histories is a fundamental problem. Phylogenomics, the reconstruction of evolutionary history using genomics data, like many other sub-areas of computational biology, has benefited tremendously from computational techniques and has a rich tradition of both computational theory and practice. This thesis contributes to this field by developing scalable and accurate computational algorithms in two sub-areas of phylogenomics – species tree estimation, and multiple sequence alignment. Moreover, we generalize our discrete algorithm research into overlapping community detection in large-scale networks, with a slightly different focus on high performance computing.

Our work in phylogenomics centers first on species tree estimation, a challenging task due to evolutionary complexities like incomplete lineage sorting and gene tree reconstruction error. We introduce Weighted ASTRID, an extension of the ASTRID method; we notably enhance species tree estimations by incorporating gene tree uncertainty inspired by recent works. This approach significantly boosts both accuracy and robustness in the face of gene tree reconstruction error, and we designed a more efficient implementation of ASTRID. Additionally, we present NJst-J and FASTRAL-J, methods leveraging prior species tree knowledge for refined inference. We venture briefly into multiple sequence alignment with WITCH-NG, an optimized version of the prior method WITCH both algorithmically and implementation-wise for adding new sequences into existing alignments. These contributions collectively offer not only theoretical advancements but also efficiently implemented practical tools for the phylogenomics community.

In parallel, the dissertation extends into overlapping community detection in large-scale networks, acting as a generalization of our study of discrete algorithms in trees to real world networks. Using the Chapel programming language, we propose novel methodologies within the Arachne graph framework, designing a distributed version of the HDBSCAN clustering method specialized for graph clustering.

This body of work emphasizes a balance between theoretical accuracy and practical efficiency. We not only contribute to the theoretical efficiency of the algorithms, but all our algorithms are also efficiently implemented, leveraging modern programming languages, hardware capabilities, and software engineering practices.

To my parents, whom I am indebted to beyond measure.

Acknowledgments

It is easy to feel existential during the PhD journey. The existential dread is even harder when working in this vast interdisciplinary field of computational biology. Research is a wonderful endeavor, where one gets paid to pursue something pure, yet it also comes with great uncertainty. There is also something inherently existential when researching as a foreign student. Ha Jin wrote that a migrant writer must always confront the betrayal of their homeland by writing in a foreign language. Well, maybe Jin was not thinking about *this* kind of a migrant writer, a graduate student anxious in finishing their dissertation, but to type LaTeX, alone in a college town, drinking coffee from a Nespresso machine has become so familiar that I have forgot how alienating I first felt stepping onto this land.

I owe my deepest gratitude to my advisor, Dr. Tandy Warnow, whose guidance has been invaluable. I would also like to thank my collaborators, who have been a constant source of inspiration and support. Notable among them are Chengze Shen, Minhyuk Park, Yasamin Tabatabaee, James Willson from UIUC; Dr. David Bader, Oliver Alvarado Rodriguez, and Dr. Zhihui Du from NJIT; and Dr. Kelly Williams from Sandia National Laboratories. Additionally, I am grateful to Dr. Darko Marinov and Dr. Talia Ringer for their advice in times of need.

I extend my thanks to the rest of my dissertation committee: Dr. Lawrence Rauchwerger, Dr. Mohammed El-Kebir, Dr. Hanghang Tong, and Dr. David A. Bader, with special appreciation to Dr. Bader for introducing me to high performance computing.

I cannot state enough of my gratefulness to my family, especially my parents Zhongzhu Chen and Lan Liu, for their constant support. Finally, special thanks to Cat, Chicken, and Pusheen.

Table of Contents

Chapter 1 Introduction 1
1.1 Motivation 1
1.2 Scope and Contributions 1
1.3 Organization of the Thesis 2
1.4 Acknowledgements 3

Chapter 2 Background 4
2.1 Multiple Sequence Alignment 4
2.2 Hidden Markov Models 5
2.3 Two-Step Methods for Multiple Sequence Alignment 6
2.4 Phylogenetic Trees 7
2.5 Species Trees & The Multispecies Coalescent Model 10
2.6 Simulated Data & Measuring GTEE 14
2.7 Additional Considerations: GDL and HGT 15
2.8 Community Detection 16
2.9 Practical Computational Efficiency 20

Chapter 3 Weighted ASTRID: Better Internode Distance-Based Species Tree Estimation 21
3.1 Introduction 21
3.2 Materials and Methods 21
3.3 Experimental Study 26
3.4 Results 30
3.5 Discussion 40
3.6 Conclusions 43

Chapter 4 Fast & Scalable Species Tree Inference with Partial Constraints 49
4.1 Introduction 49
4.2 Background 50
4.3 New Methods: NJst-J and FASTRAL-J 52
4.4 Theoretical Results 53
4.5 Experimental Study 55
4.6 Results 56
4.7 Discussion 59

Chapter 5 WITCH-NG: Efficient Aggregation of eHMM Predictions 65
5.1 Introduction 65
5.2 Background 66
5.3 WITCH-NG: Redesign of WITCH 68
5.4 Experimental Design 70
5.5 Results 73
5.6 Discussion 75
5.7 Conclusion 78

Chapter 6 Overlapping Community Detection in Large-Scale Networks 80
6.1 Introduction 80
6.2 Background 80
6.3 Materials and Methods 84
6.4 Experimental Setup 92
6.5 Conclusion & Future Work 98

Chapter 7 Conclusion 100

Appendix A Additional Results and Materials for Weighted ASTRID 104
A.1 Commands 104
A.2 Supplementary Text 106
A.3 Supplementary Figures & Tables 107

Appendix B Additional Results and Materials for Fast and Scalable Species Tree Inference with Partial Constraints 108
B.1 Commands and Software Availability 108
B.2 Complete Figures for the Simulated Experiments 110
B.3 Constraint Tree for the Biological Experiment 110
B.4 Dataset Availability 110

Appendix C Additional Results and Materials for WITCH-NG 116
C.1 Software Versions and Commands 116
C.2 Dataset Availability 117

References 124

Chapter 1: Introduction

1.1 MOTIVATION

Evolution is one of the central topics of biology [1]. The reconstruction of the evolutionary history of all extant species on earth, reconstructing the so-called "tree-of-life", is one of the grand scientific challenges in our age. Given the vast amount of genomics data, deep understanding of computational and algorithmic techniques is essential to tackle this grand challenge [2].

Phylogenetics represents the study of evolutionary histories. Our focus is largely on phylogenetic (evolutionary) trees. Phylogenetics has undergone a radical transformation with the advent of high-throughput sequencing technologies [2]. While these technologies have a wealth of genetic data, they also come with significant computational challenges. Adding to the complexity, estimating species phylogeny suffers from gene tree discordance, evolutionary events such as incomplete lineage sorting (ILS), gene duplication and loss (GDL), and horizontal gene transfer (HGT) [3]. These evolutionary processes can create differences in evolutionary history between the genes and the species. In addition, the reconstruction of gene trees often require accurate intermediate steps of aligning biological sequences, leading to complexities in accurate phylogenetic reconstructions [4].

In this thesis, we explore theoretically scalable algorithms in phylogenomics and community detection. The latter, community detection in networks, is a topic that acts as a generalization of our study on discrete algorithms and also has widespread application both in and beyond biology. While our work is largely algorithmic, designing scalable algorithms that are both accurate and scale well in terms of asymptotics, we are also inspired by efficient techniques useful for any real-world usage, e.g., using high performance languages, and with occasional high-performance techniques.

1.2 SCOPE AND CONTRIBUTIONS

This thesis revolves around two primary areas of focus:

- **Phylogenomics:** Our work focuses on enhancing species tree estimation methods, particularly under gene tree discordance like incomplete lineage sorting. We introduce improvements to a prior fast and accurate species tree estimation method ASTRID [5], both in terms of algorithm and implementation, incorporating the uncertainty of gene tree reconstruction into species tree estimation. This advancement not only improves

the accuracy of ASTRID under the realistic condition of gene tree reconstruction error but also results in the fastest species tree algorithm that remains competitive in accuracy with the state of the art [6]. Additionally, we have developed two constrained species tree estimation methods that leverage prior information about the species tree topology for more accurate species tree inference. Switching our gears slightly to multiple sequence alignment, we also present a fast method for multiple sequence alignment under fragmentary input sequences, improving upon a prior method WITCH [7], leading to an as-accurate and theoretically better motivated version WITCH-NG [8].

- **Overlapping Community Detection:** In exploring community detection within complex real-world networks, our approach extends beyond biological applications. Here, the emphasis is on developing algorithms that are not only efficient in processing large-scale data but also robust and accurate for inferring overlapping communities under large-scale networks. We generalize a prior density-based clustering algorithm HDBSCAN [9] to a parallel version to handle overlapping community detection in large-scale networks.

1.3 ORGANIZATION OF THE THESIS

The rest of this dissertation is organized as follows:

Chapter 2 provides the theoretical background for phylogenomics and community detection, providing the necessary background and context.

Chapter 3 focuses on the development and evaluation of Weighted ASTRID, delving into the algorithm's structure, accuracy characteristics, and practical implementation.

Chapter 4 explores the problem of constrained species tree inference, where prior knowledge about the species tree is incorporated into the estimation process. We introduce two new methods, NJst-J and FASTRAL-J, and evaluate their performance on simulated and biological datasets.

Chapter 5 presents WITCH-NG, an efficient method for sequence alignment under sequence length heterogeneity. This chapter discusses the algorithm design, implementation optimizations, and accuracy and runtime comparisons with existing methods, particularly with the prior methods WITCH and UPP [10].

Chapter 6 investigates overlapping community detection in large-scale networks using high performance computing (HPC) techniques. We discuss the parallelization of a version of HDBSCAN under a non-shared-memory architecture, a popular density-based clustering method. Our attack angle shards the graph into smaller subgraphs and processes them in parallel, leading to a scalable and efficient algorithm for overlapping community detection.

Conclusion summarizes the contributions of this thesis and discusses future research directions.

Appendices provide additional details and supplementary materials for each chapter.

1.4 ACKNOWLEDGEMENTS

This dissertation has benefited from proofreading using generative AI services. GPT4 Turbo [11], Gemini 1.5 Pro [12], Mistral-7B-Instruct-v0.2 [13] have all done spell-check, grammar-check, and acted as a replacement for a "native speaker at the writing center" to help me proofread my writing. However, no whole phrases, except spelling suggestions, are taken from generative AI.

Chapter 2: Background

2.1 MULTIPLE SEQUENCE ALIGNMENT

Multiple sequence alignment (MSA) is a crucial technique in bioinformatics with diverse applications such as protein structure prediction [14], [15] and phylogenetic reconstruction [16], [17]. The process of MSA involves aligning sequences of DNA, RNA, or proteins to identify regions of similarity that indicate functional, structural, or evolutionary relationships among the sequences [18]. Although various alphabets are used in sequence analysis, this thesis primarily focuses on the DNA alphabet consisting of the nucleotides `ACTG`.

MSA seeks to uncover regions of *homology*, where sequences share similarities due to descent from a common ancestral region. The detection of homology is challenging due to evolutionary events such as insertions and deletions, which may cause sequences to differ in length. To accommodate these differences, MSA introduces gaps (denoted by "-") into the sequences, aligning them such that gaps represent evolutionary insertions or deletions, and aligned nucleotides suggest homology.

No single computational problem captures the wide applications of multiple sequence alignment. More combinatorial variants that fit well under the study of discrete algorithms or even string algorithms include optimizing for the Sum-of-Pair score [19], Total-Column scores [20]. Statistical variants of the problem include maximum likelihood under a probabilistic model [21], and relatedly, Bayesian methods [22]. More graph theoretical optimization problems [23] also exist.

As the first computational problem we introduce under this realm of computational biology, we start by expanding on one of the combinatorial optimization problems in MSA, the sum-of-pairs optimization. At its core, the sum-of-pairs optimization seeks to minimize the sum of pairwise differences between aligned sequences, and can be considered as a natural generalization of the classical pairwise alignment problem (either global or local pairwise alignment [24], [25]) often serve as introductory algorithmic problems for sequence comparison in computational biology. This optimization problem is NP-hard [26], and the problem is typically solved using dynamic programming (DP) techniques [19] in exponential time to the number of input sequences, and the DP algorithm is also typically a generalization of the DP algorithms in the pairwise alignment case.

Problem 2.1. Consider a set of DNA sequences $\{s_i\}$, each a string over the alphabet $\Sigma = \{A, C, T, G\}$. The task is to find gapped versions of these sequences, a_i, where each a_i is derived from s_i by inserting gaps (if necessary), optimizing towards minimizing the sum of

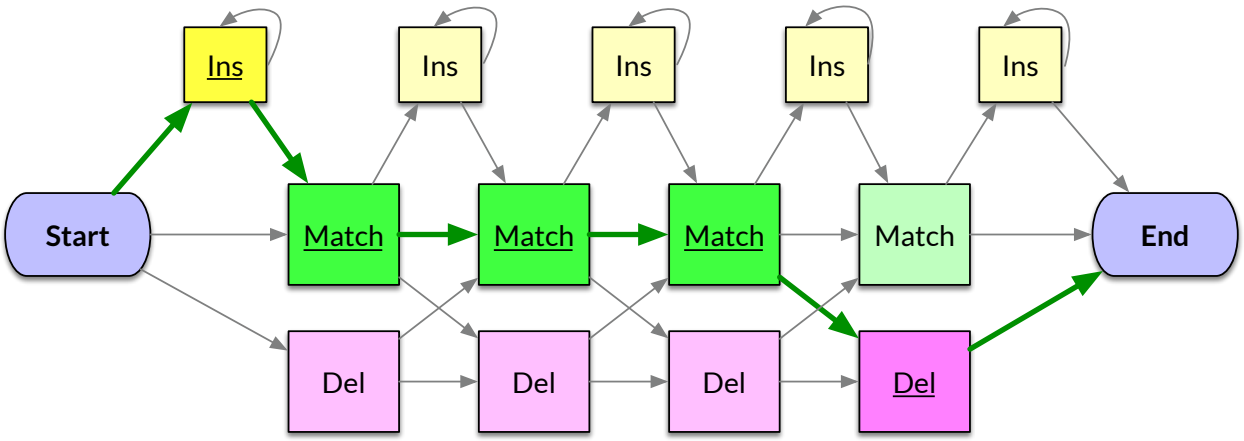


Figure 2.1: An illustration of a profile Hidden Markov Model (HMM) for a multiple sequence alignment, along-side the states and transitions (transition probabilities omitted for simplicity).

pairwise distances. For example, a version of the sum-of-pairs optimization (SP) dictates minimizing the sum of pairs of Hamming distance over non-gapped positions:

$$\text{argmax}_{\{a_i\}} \sum_{0 \le i,j < k} d(a_i, a_j) \tag{2.1}$$

where

$$d(a_i, a_j) = \#\{\text{positions } k \,|\, a_i[k] \neq a_j[k] \text{ and } a_i[k], a_j[k] \in \Sigma,\ 0 \le k < n\} \tag{2.2}$$

and $a_i[k]$ denotes the k-th character of the string a_i.

2.2 HIDDEN MARKOV MODELS

Profile Hidden Markov Models (HMMs) [27], [28] are generative models for a set of genetic sequences. They have wide applications including *de novo* sequence alignment, sequence search, and sequence annotation [29].

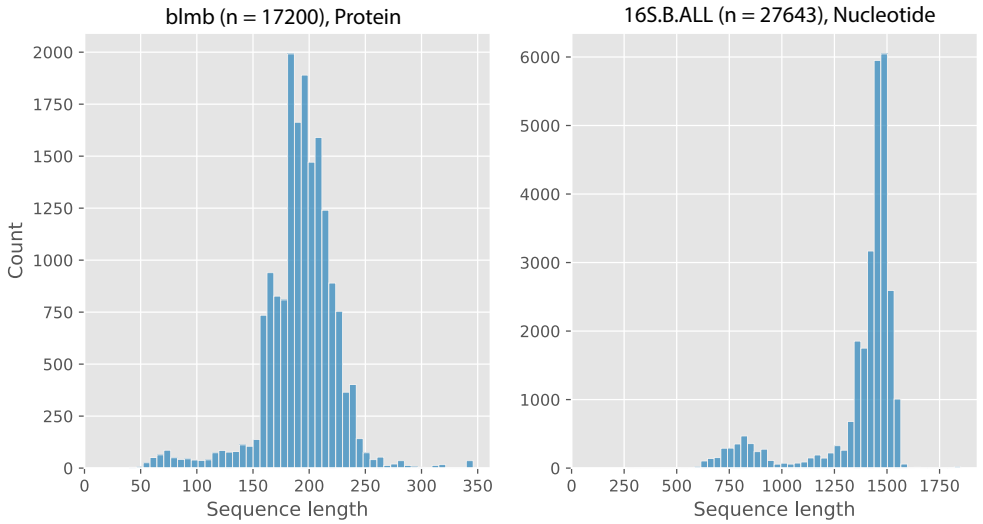


Figure 2.2: Sequence length histograms of two biological datasets that show sequence length heterogeneity. "blmb" comes from HomFam [34] and "16S.B.ALL" comes from the Comparative RNA Website (CRW) [35]. This figure is duplicated from chapter 5.

In our case, we largely focus on DNA sequences and the power of an HMM to align a sequence. In a HMM, a genetic sequence is defined by a path through the model (refer to Figure 2.1). This path begins with a start state, and proceeds through a series of transitions among "insertion", "deletion", and "match" states. Given the nature of HMMs as generative models, HMMs can be derived in many ways, arguably most commonly trained from an existing set of evolutionarily related sequences [29].

Given a multiple sequence alignment, an HMM built from the alignment can be used to generate new sequences that are similar to the original set, but more importantly also can be used to align new sequences to the original set. Often, one aims to "extend" an alignment to include previously unaligned, perhaps newly discovered sequences [30]–[32]. These types of use cases are handled by software pipelines, the most popular of which is using a two-step HMMER [33] pipeline consisting of `hmmbuild` on the MSA paired with `hmmalign` using the new sequences.

2.3 TWO-STEP METHODS FOR MULTIPLE SEQUENCE ALIGNMENT

Sequence length heterogeneity is a major challenge in multiple sequence alignment, where sequences with varying different lengths need to be aligned. Due to data collection, sequencing errors, or evolutionary events [36], it is common to have sequences that are much shorter

than the majority of the sequences in the dataset [37].

Traditional MSA methods such as MAFFT [38], Clustal-Omega [34], etc., experience substantial accuracy decline [10] under such cases where large parts of the sequences are "fragmentary", much shorter than a majority of the sequences.

Generalizing the two-step pipeline from `hmmbuild` paired with `hmmalign`, two-step methods are specifically designed to align sequences under sequence length heterogeneity. Two-step methods partition the input sequences into two buckets of sequences, one bucket containing "full-length" sequences and the other containing "fragmentary" sequences. The full-length sequences are aligned using a standard multiple sequence alignment method, assumed to have sufficient accuracy as the "full-length" sequences are assumed to be homogeneous in length, and the partial sequences are aligned to the full-length sequences. This merging step forms the bulk of the two-step MSA method. The resulting alignments are then merged to produce the final alignment.

We briefly survey two-step methods for sequence alignment since the introduction of `hmmbuild` and `hmmalign`. The first of these types of methods is UPP [39], generalizing the prior ensemble of HMMs approach from SEPP [40] designed for phylogenetic placement. UPP enjoys a later extension called UPP2 [41] that is algorithmically more efficient along with other improvements, while MAGUS+eHMMs [42] acts as an extension of UPP that demonstrates that using a more accurate "backbone" alignment on the full-length sequences from MAGUS improves accuracy further compared to baseline UPP. WITCH [43] is a more accurate version of UPP using ensemble-based methods. HMMerge [44], as opposed to perform a mixture of experts by combining the predictions *à la* WITCH, combines the "experts" in the mixture directly via the merging of HMMs then performing predictions directly using the merged HMM. EMMA [45], although in the same family as the other works, differs due to the use of MAFFT as opposed to HMMER for aligning fragments to the full-length sequences. Although not designed directly for *de novo* sequence alignment, PaPaRa [31], [46] showed more accurate under their evaluations as a replacement for `hmmalign` in the two-step pipeline for phylogenetic placement.

2.4 PHYLOGENETIC TREES

Phylogenetic trees [49], [50] serve as a fundamental structure in representing the evolutionary relationships among species or individuals. The branching pattern of evolution, in which a species can "speciate" into two or more species, naturally maps to the tree's branching structure.

Phylogenetic trees in evolutionary models are frequently rooted, indicating the most recent

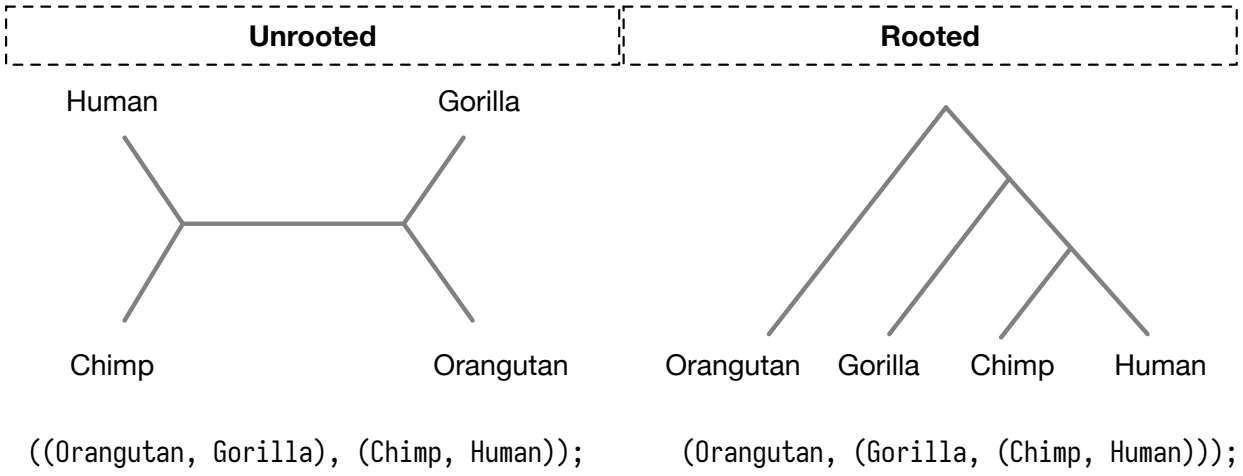


Figure 2.3: Phylogenetic tree of the currently accepted [47] evolutionary relationships among four of the great apes species: Human, Chimp, Gorilla, and Orangutan. We also list the Newick string representations. On the left, an unrooted tree illustrates the relative evolutionary distances between the species without specifying a root or direction of evolution. On the right, the tree is rooted, providing a perspective on the evolutionary pathway with the most recent common ancestor at the base. This figure is inspired by a similar figure in [48].

common ancestor of all taxa under consideration. This root node represents the point of origin from which various species have diverged. The leaves or tips of the tree correspond to extant species or sampled individuals, representing the current diversity observed. Internal nodes, on the other hand, denote common ancestors and signify speciation events.

Nodes and Edges

Phylogenetic trees consist of nodes (leaves and internal nodes) connected by edges. The leaves represent the sampled taxa, and the internal nodes signify speciation events or common ancestors. Edges in a phylogenetic tree represent evolutionary progress without speciation leading to multiple extant species in the dataset. The removal of an edge in a tree can induce a bipartition of the tree's leaf set. The set of bipartitions uniquely defines the tree's unrooted topology.

Rooted and Unrooted Trees

Rooted trees carry a direction in evolutionary pathways, whereas unrooted trees illustrate relationships without a defined origin. Inferring the rooted topology of a tree often presents practical challenges. For example, the tool ASTRAL [51] typically generates only unrooted

species trees, primarily because it relies on input gene trees inferred often under models such as GTR+Γ, where the root is unidentifiable. Consequently, rooting species trees becomes a distinct and complex issue that requires additional specialized methods.

Binary and Multifurcating Trees

An unrooted tree is said to be binary if all nodes have either degree 3 or 1, whereas a binary rooted tree is a tree where all nodes have either 0 or 2 children. Non-binary trees, also called multifurcating trees, often represent unresolved evolutionary relationships and are said to contain **polytomies**. We typically assume binary trees in this thesis to simplify the discussion.

2.4.1 Notations

- T**:** Represents a phylogenetic tree. It is a connected, acyclic graph that depicts the evolutionary relationships among a set of taxa. Often we assume that T is unrooted.
- $E(T)$: Denotes the edge set of a tree T. Each edge in $E(T)$ connects two nodes in T. Often edges have two attributes, length (representing evolutionary distance) and often practically inferred trees carry a "support" value, representing the confidence in the edge's existence.
- $L(T)$**:** Denotes the leafset (taxa) of a tree T. These are the terminal nodes of T representing extant species or sampled individuals in the phylogenetic study. Often for theoretical purposes, we assume that the leafset is labelled by $1, 2, \ldots, n$.
- $A \mid B$**:** A bipartition induced by deleting an edge e from the tree T. It divides the leafset $L(T)$ into two disjoint subsets A and B. We say that a bipartition is **trivial** if one of the subset is a singleton set, as this bipartition is present in all trees with the same leafset.
- $C(T)$**:** The set of all bipartitions of a tree T, defined by its edge set. Typically, we are interested in the set of non-trivial bipartitions of T, so for our purposes $C(T)$ is the set of non-trivial bipartitions unless defined otherwise. $C(T)$ uniquely defines the unrooted topology of T.
- $Q(T)$**:** The set of all (unrooted) quartet trees of a tree T. A quartet tree is a tree on four leaves, with $Q(T)$ of size $\binom{n}{4}$ for a tree with n leaves. Notably, three-leaf

unrooted subtrees contains no information about the tree's topology, and quartets are the smallest unrooted trees that contain information about the tree's topology.

2.4.2 Comparing Trees & Distances

We are often interested in the "topology" of the tree, the "shape" of the tree regardless of the edge annotations. Given two trees T and T' on the same leafset $L(T) = L(T')$, we can compare the (unrooted) topologies of T and T' by comparing their bipartitions. The Robinson-Foulds (RF) distance [52] between two trees T and T' is defined as the symmetric distance of their set of non-trivial bipartitions, i.e., $d_{RF}(T, T') = |C(T)\Delta C(T')|$. The RF distance is a metric, and thus it is zero if and only if T and T' are identical. The RF distance is also known as the bipartition distance. Typically, given a ground truth T and an estimated tree T', we want to quantify the difference between T and T' as an error metric in $[0, 1]$. In this case, we define the nRF error rate as $d_{RF}(T, T')/|C(T)|$.

While the RF distance is a popular metric for comparing trees, it is not the only one. One well-known limitation of RF distance is that RF distance is not robust under rogue taxa [53], [54]. Other metrics include the quartet distance, the subtree prune and re-graft (SPR) distance, etc..

2.5 SPECIES TREES & THE MULTISPECIES COALESCENT MODEL

Gene trees do not necessarily reflect the same evolutionary history as the species tree, due to incomplete lineage sorting (ILS), or due to gene duplication and loss and other evolutionary events. Species trees capture the evolutionary history of the species, and can be different compared to how genes evolve, leading to gene tree discordance.

Given this context, the Multispecies Coalescent (MSC) model [55], [56] is designed to statistically model ILS. This model extends the standard coalescent theory [57], which usually addresses the ancestry of alleles within a single population, to multiple species populations. Incomplete lineage sorting is particularly pronounced when species have large population sizes and short speciation times. In general, the evolutionary lineage of a particular gene might not align with the species tree and the most frequent gene tree topology might not represent the true species tree topology. This is known as the "anomaly zone" [58]. The MSC model provides a theoretical framework to address such discordance due to ILS for species tree estimation.

A popular [59]–[61] approach in species tree estimation under ILS is called concatenation. This method involves horizontally joining the alignments of individual genes and applying a

maximum likelihood heuristic on the combined dataset. However, studies have shown that concatenation can be statistically inconsistent under the MSC, converging in cases to an incorrect tree topology as the number of genes increases [62], [63]. Additionally, empirical evidence suggests that concatenation's accuracy deteriorates with higher levels of ILS and faces scalability challenges with extensive data [51], [60].

Given both theoretical and practical shortcomings of concatenation, we briefly survey existing methods for species tree inference that are designed with the MSC model explicitly in mind, specifically focusing on so-called **summary** methods.

Summary methods typically operate in two phases. The first phase involves estimating a gene tree for each gene's alignment using a maximum likelihood method (e.g., FastTree [64], RAxML [65], [66], or IQ-TREE [67]). This gene tree estimation step in practice can often be error-prone [68] due to the limited signal of gene alignments [4], and specialized methods have been attempted and developed [60], [69] to specifically mitigate the reconstruction error of gene trees. Subsequently, in the second step, the summary method uses these gene trees as input to estimate the output species tree. This second phase is often referred to as the summary method itself, and the bulk of the design of the summary method occurs here.

An important property of a species tree estimation method is **statistical consistency**. We say that a summary method is statistically consistent if the estimated species tree converges to the true species tree as the number of genes increases, typically assuming that the input only contains true gene trees. In other words, a summary method f that maps a set of true gene trees $\{\mathcal{G}\}$ to a species tree S is statistically consistent if $f(\mathcal{G}) \to S$ as $|\mathcal{G}| \to \infty$. In other words, f is a consistent estimator for the species tree.

MP-EST (Maximum Pseudo-likelihood for Estimating Species Trees)

One of the earlier widely used summary methods, MP-EST [70] searches for a species tree using a heuristic that maximizes a "pseudo-likelihood score" of a species tree derived from the (rooted) triplets from the input rooted gene trees, notably using both the topology and the branch lengths from the input gene trees. MP-EST is statistically consistent under the MSC model.

ASTRAL (Accurate Species TRee ALgorithm)

ASTRAL [51], [71], [72] optimizes for the species tree that minimizes the total quartet distance to the gene trees. The quartet distance between a set of input gene trees $\mathcal{G}$ and a

candidate species tree T is defined by the expression

$$\sum_{G \in \mathcal{G}} |Q(G) \cap Q(T)|, \tag{2.3}$$

where $Q(t)$ denotes the set of quartets of t. This metric measures the number of quartet trees shared between each gene tree and the species tree, thus quantifying their congruence. Minimizing this above-defined distance is called the maximum quartet-support supertree (MQSST) problem. MQSST is NP-hard [73], but an exact solution to MQSST is a statistically consistent method for species tree estimation under the MSC [51]. ASTRAL is recognized as a statistically consistent method under the multispecies coalescent (MSC) model, and as of early 2024, remains the preferred approach [74]–[76] for species tree inference under the MSC.

To go into more details, ASTRAL solves the **X-constrained MQSST problem**, which we now define. The input is a set $\mathcal{G}$ of gene trees and also a set X of "allowed bipartitions", and the output is a tree T optimizing the MQSST score subject to $C(T) \subseteq X$, and ASTRAL uses dynamic programming to solve the X-constrained MQSST problem in polynomial time.

NJst & ASTRID

We now expand on two important methods belonging to the class of "distance-matrix"-based species tree estimation methods.

In 2011, the NJst method was developed, which uses this approach [77]. In NJst, the distance between two taxa i and j within a tree is the number of nodes on the path between the leaves for those taxa. This "internode distance" is then averaged across all the gene trees, thus defining the "average internode-distance matrix". Given the distance matrix, neighbor joining [78] is then used to compute a tree. This method, referred to as NJst, is polynomial time and statistically consistent under the MSC [79].

The proof that NJst is statistically consistent under the MSC for this approach has two parts: first, that the average internode distance matrix converges, as the number of genes increases, to an additive matrix for the species tree, and second, that neighbor joining return the species tree when the average internode distance matrix is close enough to this additive distance matrix. The proof that the average internode distance matrix converges to an additive matrix for the species tree was provided in [79], and the guarantee for neighbor joining returning the species tree when the estimated distance matrix is close enough to an additive matrix was proven in [80]. This last condition is referred to by saying that neighbor joining has a "positive safety radius", which means that for the true tree T with additive matrix D, there is some value $\epsilon > 0$ so that when given a distance matrix d such

that $L_\infty(d, D) < \epsilon$, then neighbor joining is guaranteed to return the unrooted topology of T.

Note that the proof that NJst is statistically consistent under the MSC enables any distance-based method to replace neighbor joining, as long as the substitute method also has a positive safety radius. Once such method is balanced minimum evolution (BME), which also has a positive safety radius [81], [82]. Moreover, BME has several theoretical properties that suggest that it will have a better sample complexity (and hence better accuracy) than neighbor joining (see discussion about the edge safety radius in [82]). This observation led to the development of ASTRID, which we now describe.

ASTRID is very similar to NJst in design in that it computes a distance matrix from the gene trees and then computes a tree on the distance matrix. The major difference between ASTRID and NJst is that it uses a heuristic for BME available in FastME [83] to construct a species tree from the distance matrix it computes instead of neighbor joining. A minor difference between ASTRID and NJst is that ASTRID uses a slightly different distance matrix: instead of using the number of nodes in a gene tree between the leaves for each pair of taxa, it uses the number of edges. It is trivial to see that this way of defining the distance matrix also converges to an additive matrix for the species tree; hence, this change does not impact statistical consistency. Because the matrix it computes is not exactly the same, we will refer to it as the intertaxon distance matrix.

Like NJst, ASTRID is statistically consistent under the MSC. However, the proof for statistical consistency for ASTRID is somewhat more complicated than for NJst, since ASTRID uses a heuristic search within FastME for BME, which is NP-hard [84]. This heuristic search begins with a greedy BME tree, followed by either NNI (nearest-neighbor interchange) or SPR (subtree prune and regraft) moves, until a local optimum is found. Although this heuristic uses a local search strategy, this approach is guaranteed to converge to the true species tree topology when the input distance matrix is sufficiently close to the additive matrix for the species tree (i.e., like neighbor joining, it also has a positive safety radius) [81], [82]. Thus, we have the following theorem:

Theorem 2.1. ASTRID is statistically consistent under the MSC, provided that the default heuristic is used and run until convergence to a local optimum.

Several simulation studies have shown ASTRID has accuracy that can be comparable to that of ASTRAL and can scale to large datasets [5], [60]. Moreover, early implementations of ASTRID were shown to be much faster than early versions of ASTRAL on large datasets [5], and although ASTRAL has continued to improve in speed, there may still be a runtime advantage to ASTRID. Finally, ASTRID is a key technique used in FASTRAL for speeding up ASTRAL.

wASTRAL and ASTER

wASTRAL [85] is an extension of ASTRAL that incorporates gene tree uncertainty by weighting gene tree quartets based on their branch support and lengths. This effectively discounts the contribution of unreliable quartets, leading to improved accuracy under conditions where gene tree estimation error is high. ASTER [86] is a software suite that includes wASTRAL and other methods for species tree estimation and downstream analyses. It offers an expansive, more optimally implemented suite of ASTRAL-like methods, and includes the canonical implementation of wASTRAL.

FASTRAL

FASTRAL [87] addresses the computational challenges posed by ASTRAL, particularly when the size of the set X—which includes all bipartitions from input gene trees—becomes large due to high levels of incomplete lineage sorting (ILS) or gene tree estimation error (GTEE). FASTRAL optimizes ASTRAL's execution by initially generating a reduced set of allowed bipartitions using subsampled trees produced by ASTRID. It then employs ASTRAL to solve the constrained MQSST problem more efficiently with this reduced set, thereby maintaining statistical consistency while enhancing computational performance under a wide range of model conditions. This approach is especially beneficial for large datasets where traditionally ASTRAL suffer more in running time – such as large datasets with high levels of ILS or GTEE.

2.6 SIMULATED DATA & MEASURING GTEE

Real-world data tend to have a significant amount of gene tree estimation error [60], [68]. Given that in phylogenomics the "ground truth" is frequently lacking, it is common to generate simulated data. An example pipeline is to use SimPhy [88] to simulate the species tree and gene trees, INDELible [89] to simulate alignments on the "true" (model) gene trees, and then use an off-the-shelf alignment software such as FastTree to reconstruct the gene trees.

Given the impact of gene tree estimation error on the quality of species tree estimation and the importance of inducing a level of GTEE that is "real", two common approaches have been developed to quantify the extent of GTEE. The first approach, common for simulated studies, is simply compare the pairwise Robinson-Foulds (RF) distance between the true gene trees and the estimated gene trees. This average RF-distance approach allows a form

of objective measure of the GTEE, useful for simulated studies. More formally, the metric **GTEE** is defined as such, as the average (pairwise) RF distance between the true gene trees and the estimated gene trees. In general, the more signal a method has for inferring the gene trees, the lower the GTEE. As such, in simulated studies, it is common for methods to use sequence length ("seq len") of the simulated sequenecs used to infer the gene trees as a proxy for GTEE.

However in empirical studies where the true gene trees are unknown, this approach is not feasible. In such cases it is common to use the average support value of the gene trees as a proxy for GTEE. In general, the support value of a branch denotes the statistical confidence of the branch's existence in the gene tree, and a low support value indicates a high level of uncertainty in the branch's placement. The most well-known form of support is bootstrap support due to Felsenstein [90].

2.7 ADDITIONAL CONSIDERATIONS: GDL AND HGT

While this thesis primarily focuses on ILS as a source of gene tree discordance, other evolutionary events like gene duplication and loss (GDL) [91] and horizontal gene transfer (HGT) [56] can also complicate species tree estimation.

GDL events can lead to multi-copy gene families, where multiple copies of a gene exist within a species. This can create gene trees with multiple copies of species, known as MUL-trees [92]. Summary methods like ASTRAL and ASTRID have been extended to handle MUL-trees, with methods like ASTRAL-Pro [93] and DISCO [94] demonstrating promising results. HGT, the transfer of genetic material between distantly related species, can also lead to gene tree discordance. Quartet-based methods like ASTRAL and Weighted Quartet MaxCut [95] have been shown to have good accuracy even when both HGT and ILS are present [96]. Addressing GDL and HGT effectively in species tree estimation remains an active area of research. While this thesis focuses on ILS, the developed methods and insights can potentially be extended to handle more complex models of gene tree evolution, as ILS is ubiquitous and co-occurs with other evolutionary events in practice.

In addition to summary methods, other species tree estimation method types include site-based methods [97], [98], co-estimation methods [99], [100], and, as mentioned, concatenation. In this dissertation, our focus is on summary methods.

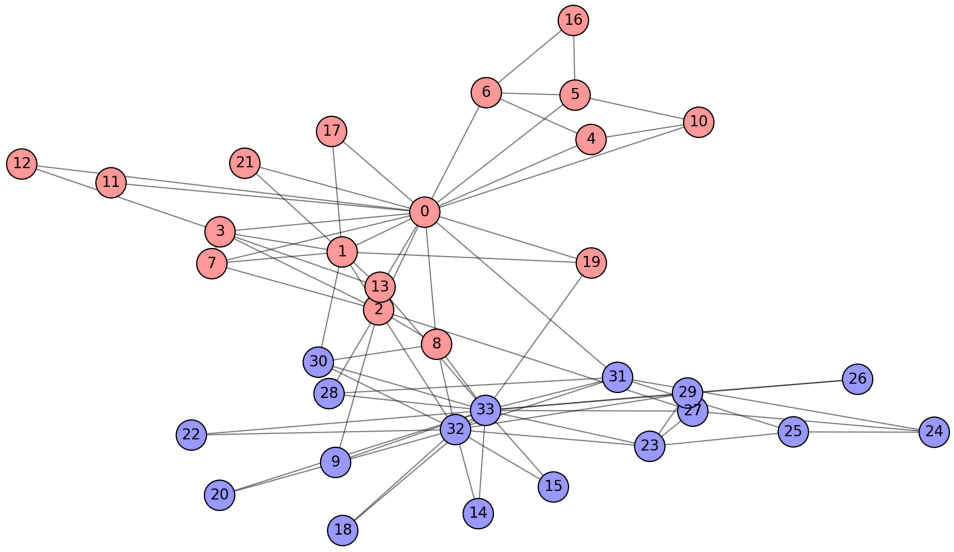


Figure 2.4: Zachery's Karate Club network [101], a popular example of community structure in networks. The network depicts the social interactions between members of a karate club, with the edges representing friendships between members. The colors depict the two ground truth communities. The network exhibits a clear community structure, with two distinct groups of members.

2.8 COMMUNITY DETECTION

Real-world graphs exhibit community structure [102]. The problem of community detection concerns recovering such communities from network structures. Analogously, graph clustering forms an important task in unsupervised learning on graphs, and we use community detection and graph clustering interchangeably, as the underlying algorithms and theory often overlap [103].

Community detection can be broadly divided into two classes: overlapping and disjoint community detection. Many types of real-world communities have overlapping structures [104], such as a user belonging to multiple social groups. In contrast, disjoint communities are non-overlapping, where each node belongs to exactly one community. In this thesis, we focus specifically on the topic of efficient overlapping community detection.

We briefly survey community detection methods. The most popular of them focus on disjoint community detection, with the Louvain method [105], an effective heuristic for the NP-hard [106] optimization problem of modularity optimization [107], among the most well-known and used. Unfortunately, modularity optimization is now well-known to suffer from a "resolution limit" [108]; communities that are small relative to the number of nodes

in the network may be undetectable given a large enough network. Consequently, resolution-limit-free methods [109], the most well-known perhaps Leiden [110], in its default setting optimizing for the Constant-Potts Model [109] problem, have been developed. Other popular disjoint community detection methods include label propagation [111], spectral clustering [112], [113], and InfoMap [114]. Given the generality of the graph clustering problem, many other methods exist [115], and the field is very active [116], [117].

Overlapping community detection sometimes first produce disjoint communities then generalize to overlapping ones [118], [119]. Other overlapping community detection methods are formulated modelling overlapping communities in mind, including BigClam [120] and SLPA [121].

2.8.1 Graph Theory

We provide a brief excerpt on relevant graph theory. Given a set of **nodes** V, we can equip these points with a set of **edges** $E \subseteq V^2$. This definition gives rise to a graph $G = (V, E)$ that is a **simple directed graph** or simple digraph. Undirected graphs, under this framework, can be considered as a special case of directed graphs where the edges are symmetric, i.e., $(u, v) \in E \iff (v, u) \in E$. The out-degree of a node u, is the number of edges in E of form $(u, \cdot)$, and the in-degree of a node u, is the number of edges in E of form $(\cdot, u)$. The degree of a node u, $deg(u)$, is the sum of the in-degree and out-degree of u.

Given the topology G, often times for clustering or data mining purposes G, it helps to equip an inferred form of metric on G, either on edges or nodes. The stronger structure imposed by a metric space allows easier clustering (e.g. HDBSCAN [122]), embedding (e.g., t-SNE [123]), etc.. In other words, metrics allow generic clustering algorithms to be used for community detection purposes.

Given the recent rise of representation learning [124], a natural way of imposing metrics is to infer vector representations, say, on the nodes, giving each $u \in V$ where we have a representation of u denoted as $f(u) \in \mathbb{R}^k$ in some embedding space, and this gives a form of metric where $d(u, v) = d(f(u), f(v))$ for $u, v \in V$. Similarly, edge representations can also be used to impose metrics on edges through the metric on the embeddings. Graph representation learning is an active area of research [125]–[128] and has been successfully applied for community detection [129], [130].

However, given attributed graphs, it can be easier to simply use the attributes of the nodes and edges to infer metrics. For example, given edges annotated with timestamps, one can infer a metric on the edges based on the temporal distance between the timestamps.

In general, given a graph without metrics, it is often useful to infer metrics based on the

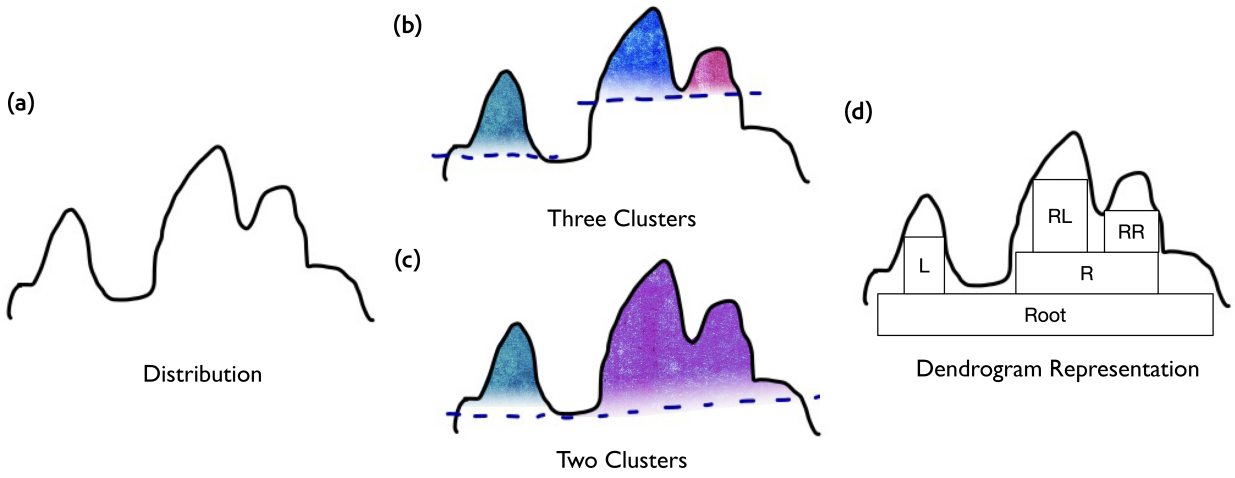


Figure 2.5: Conceptual drawing to help understand density-based clustering techniques, such as HDBSCAN [122]. (a) shows an arbitrary distribution highlighting "peaks," or denser areas with higher probability mass. (b) shows a clustering scenario with two separate "cuts," resulting in three distinct clusters. (c) shows an alternative clustering with a single "cut," yielding two clusters. (d) shows the implicit dendrogram structure underlying the distribution. The "L", and "R" labels indicate the left and right children of a node in the dendrogram, highlighting the hierarchical structure of the dendrogram.

attributes of the nodes and edges, and if such attributes are unavailable, representation learning can serve as a popular framework for inferring metrics.

2.8.2 Density-Based Clustering

We provide a brief motivation for some fragments associated with density-based clustering, setting the stage for discussing HDBSCAN [122] (refer to Figure 2.5). Intuitively, in a distribution of data points to be clustered, regions with higher probability mass, termed "dense" areas, are likely candidates for cluster centers. This principle motivates density-based clustering. However, in practical scenarios, the true distribution is unknown, and we rely on data samples to approximate it.

Despite having a model of the true distribution, determining when a "sub-cluster" should be considered as an independent cluster or considered part of a larger cluster remains unsolved. As depicted in Figure 2.5 (a), one can imagine two plausible cluster configurations, shown in Figure 2.5 (b) and Figure 2.5 (c). These configurations could be seen as different interpretations of the same hierarchical clustering. Indeed, a hierarchical clustering, or dendrogram, can be viewed as a discretized version of the original continuous distribution, as shown in Figure 2.5d. However, this connection to dendrograms, coincidentally a structure

also vital to phylogenomics, does not address the main challenge of determining optimal cluster boundaries in most graph clustering applications, which typically focus on non-hierarchical (flat) clusterings.

A straightforward strategy might involve setting a fixed probability density threshold to "cut" the distribution, as depicted in Figure 2.5 (c). More sophisticated approaches, such as the strategy used by the popular HDBSCAN algorithm, optimize metrics such as **stability**. To illustrate, consider the choice between maintaining a single cluster A or subdividing it into clusters B and C. The more substantial the "area" of A relative to the combined area of $B \cup C$, the greater the stability of A. Notably, the larger single cluster in Figure 2.5 (c) covers a greater area than the union of the two smaller clusters depicted in Figure 2.5 (b), raising the question of the magnitude of this area differential—the smaller the differential, the stronger the case for subdividing the cluster. Stability optimization is the general principle behind HDBSCAN's criterion for determining clusters, but can also be viewed as a general optimization criterion ("stability maximization") that HDBSCAN by default opts in to flatten the clusters.

HDBSCAN has many theoretical motivations and varieties in its implementation [122], [131]. For completeness, here we sketch a version of HDBSCAN that we focus on.

Given a set of finite points M and a metric d (in practice d can be a dissimilarity), HDBSCAN produces a set of disjoint clusters on M given algorithmic parameter n (a threshold for noise), using the following steps:

1. **(MST/Dendrogram Construction)** Construct a Minimum Spanning Tree T on M using d, which is analogous to creating a single-linkage dendrogram. Each hierarchical cluster C in T "births" when two smaller clusters A and B join, defining C's **birth time** as $d(A, B)$. C "dies" when it merges into a larger cluster. If C merges with D to form E, C's **death time** is $d(C, D)$. Birth and death times of clusters are recorded, where birth time for a single-node cluster is 0 and death time for the dataset-encompassing cluster is ∞. Let $\lambda_{\text{birth}} = \frac{1}{\text{birth time}}$ and $\lambda_{\text{death}} = \frac{1}{\text{death time}}$ for each cluster.

2. **(Stability Computation)** For each point p in cluster C, determine λ_p, the reciprocal distance when p leaves C. This occurs between λ_{birth} and λ_{death}, marking either the split of C or when p falls out during the cluster's lifetime. Compute the stability of C as:

$$\sum_{p \in C} (\lambda_p - \lambda_{\text{birth}}), \tag{2.4}$$

   capturing the persistent presence of points in C.

3. **(Cluster Selection/Cutting)** Starting with all leaf clusters as selected, process the dendrogram from the leaves upward in reverse topological order. For each cluster, if the combined stabilities of its child clusters exceed the stability of the cluster itself, replace the cluster's stability with the sum from its children. Otherwise, certify the cluster as selected and cut off all its child clusters from selection. The algorithm finalizes when the root is reached, yielding the selected clusters as the flat clustering. If a cluster has size less than n, it is marked as noise.

Note that usually given metric space (M, d), HDBSCAN transforms the d into a form of reachability distance [132], [133] to make the algorithm more accurate. However, this does not change the overall structure of the algorithm and for our purposes, we omit this transformation, as it might not be as well suited for metrics defined on graphs. Observe that HDBSCAN does not require the number of clusters as input, and the algorithm automatically determines the number of clusters based on the data.

2.9 PRACTICAL COMPUTATIONAL EFFICIENCY

In computer science, computational efficiency frequently revolves around asymptotic analysis, such as Big O notation. This notation, while providing insights into the theoretical limits of algorithmic efficiency, in many cases falls short in predicting actual runtime performance. For instance, algorithms classified as exponential-time may complete executions in under a second for $n \leq 12$ [134]. Conversely, some algorithms with favorable asymptotic behavior hide large constants that in practice, they are thought to be as never as useful as more practical algorithms. These high-constant algorithms are often referred to as "galactic" [135] algorithms.

In phylogenetics, practical execution speed is often an important consideration, particularly due to the extensive datasets involved. Well-known software in this field, like RAxML [17], [66], FastTree2 [136], MAFFT [38], and ASTRAL [137], are predominantly written in performance-oriented languages like C++ or Java. While the choice of programming language provides a foundational advantage in terms of speed, modern performance engineering encompasses a broader array of factors, influenced by the intricacies of contemporary computer architectures [138], [139]. These factors include cache efficiency, memory allocation optimization, and the impact of compilers and language choices on vectorization and parallelization.

Chapter 3: Weighted ASTRID: Better Internode Distance-Based Species Tree Estimation

The contents of this chapter are largely based on the following publication:

- B. Liu and T. Warnow, "Weighted ASTRID: Fast and accurate species trees from weighted internode distances," *Algorithms for Molecular Biology*, vol. 18, no. 1, p. 6, 2023

3.1 INTRODUCTION

Gene tree heterogeneity, particularly due to incomplete lineage sorting (ILS), poses a significant challenge in species tree estimation. While statistically consistent summary methods like ASTRAL have become popular for species tree estimation under ILS, their accuracy can degrade under errors in the input gene trees. Recent developments, such as weighted ASTRAL, have shown that incorporating gene tree uncertainty can substantially improve accuracy under gene tree estimation error (GTEE). This type of approach explicitly addressing GTEE recognizes that not all gene trees are equally reliable and that weighting their contribution based on their confidence can lead to more robust species tree estimation.

We introduce weighted ASTRID, an extension of the computationally efficient ASTRID method that integrates gene tree uncertainty into its internode distance calculations. ASTRID, known for its speed and competitive accuracy compared to ASTRAL, uses internode distances to infer species trees. By incorporating branch support and potentially branch lengths as weights, we aim to enhance ASTRID's robustness to gene tree estimation error while maintaining its computational efficiency. This chapter delves into the design of weighted ASTRID, exploring different weighting schemes and evaluating its performance against existing methods through simulated and empirical datasets.

3.2 MATERIALS AND METHODS

3.2.1 Basic definitions

We recall some basic definitions. Assuming a set of gene trees, let n denote the number of taxa and let k denote the number of genes. Given an unrooted phylogenetic tree T, we denote its leafset by $\mathcal{L}(T)$ and its edge-set by $E(T)$. For each edge e in $E(T)$, deleting e from

T partitions the leaves into two sets defined by the two connected components separated by e; we denote this bipartition by π_e, with the entire set of bipartitions of T denoted by $C(T)$.

For taxa $u, v \in \mathcal{L}(T)$, let $P_T(u, v)$ denote the set of edges on the unique path connecting u and v in T. Given an estimated gene tree G, we assume that each internal edge e is associated with a branch support value $s(e)$ denoting the confidence that this edge is correctly estimated, where $s(e) \in [0, 1]$. We say an edge is correctly estimated (in topology) if the bipartition associated with that edge is present in the true gene tree, thus $s(e) = 1$ if e is incident to a leaf. We also let $l(e)$ denote the length of the edge e, normally given in substitution units.

We run ASTRID on many datasets in this study, and hence we begin with some notation that we will use throughout this chapter.

Notation

We let G denote a gene tree in the set $\mathcal{G}$ of gene trees. For u, v two taxa, we let $d_G(u, v)$ denote the number of edges in the path $P_G(u, v)$ between the leaves for u, v in G, and we let $\mathcal{G}_{u,v}$ denote the set of gene trees that have both u and v.

ASTRID given complete distance matrices

We now describe how ASTRID operates when the average internode distance matrix has no undefined entries; thus, we assume that for every pair of taxa u, v there is some gene tree that has both u and v, and so $\mathcal{G}_{u,v} \neq \emptyset$ for all u, v. We refer to this as saying that the distance matrix is complete.

ASTRID proceeds as follows:

1. For each pair of taxa u, v, we set $D[u, v]$ to be the empirical mean of $d_G(u, v)$ where G ranges over $\mathcal{G}_{u,v}$. Thus, we set $D[u, v] = \frac{\sum_{G \in \mathcal{G}_{u,v}} d_G(u,v)}{|\mathcal{G}_{u,v}|}$.
2. We run FastME's heuristic for balanced minimum evolution [83], under the accurate setting of using extra rounds of NNI and SPR moves (referred to as FastME from here on) on D, outputting an unrooted species tree.

ASTRID given incomplete distance matrices

To handle incomplete distance matrices, i.e., the case where some pair of taxa u, v do not appear together in any gene tree and so $|\mathcal{G}_{u,v}| = 0$, ASTRID instead first marks the

corresponding entry of $D[u, v]$ as "missing" in Step 2. FastME requires complete distance matrices, and as such these missing entries must be imputed.

In the original version of ASTRID [5], this issue was addressed by using methods from the PhyD* family [141] of tree estimation methods, which are specifically designed to handle distance matrices that are incomplete (i.e., entries that are undefined). However, in subsequent research (published in Pranjal Vachaspati's PhD dissertation [48]), a modification to this approach was found to produce better accuracy, which we now describe:

a Run a variant of UPGMA [142] on D (that contains missing entries) called UPGMA*. Recall that UPGMA is an agglomerative clustering algorithm defined on the "average" distance between two clusters. In UPGMA*, two clusters A and B are candidates for joining only if there exists taxa $u \in A$ and $v \in B$ such that the distance between u and v is defined (non-missing), and in such case the average distance between the clusters A and B is the average of all such distances between such pairs u, v. The resulting UPGMA* tree is denoted by U_1 and we let A_1 be the matrix of path lengths (counting the number of edges) between the leaves in U_1.

b We define matrix D_1, a completion of D, as follows. For each u, v such that $D[u, v]$ is undefined (because $\mathcal{G}_{u,v} = \emptyset$), we set $D[u, v] = A_1[u, v]$. We denote this filled-in matrix by D_1, and we note that the matrix D_1 is complete.

c We run FastME with only NNI moves on D_1, obtaining a tree U_2. As before, we let A_2 denote the matrix of path lengths for U_2, and we complete D using A_2, thus producing complete matrix D_2.

As described, the distance matrix D_2 is complete, and can be used as input to any tree estimation method that operates using distances. In [48], FastME trees estimated using this distance matrix were more accurate than trees estimated using methods from PhyD* applied to the incomplete distance matrices given as input. This way of running ASTRID is available in the GitHub site [143], where it is referred to as ASTRID-2.

3.2.2 Weighted ASTRID

In the definition of the internode distance, each edge contributes equally to the internode distance for each pair of taxa it separates. Intuitively, under the realistic assumption that gene trees are estimated with a non-trivial amount of error, some branches will be more reliably estimated than the others. As such it makes sense to assign weights to branches

as some confidence of them correctly contributing to the internode distance. The branch lengths could also be used as such a proxy, because short branches are empirically hard to estimate. Thus, our problem becomes to choose appropriate weighting schemes for the edges based on information already annotated in the gene trees, that is, the branch support and branch lengths. Because branch support is already designed as some statistical confidence of the correctness of some branch, it seems natural to naively assign the support directly as the weight for each branch. We alternatively explore simply assigning the branch length as the weight. The details are presented as follows.

Distance defined by branch support

We now formally introduce **wASTRID-s** (weighted ASTRID by support), analogous to the naming of weighted ASTRAL by support. Here we try one simple approach, defining each branch's contribution to the internode distance as its support instead of 1, which gives rise to the following definition of $d_G(u, v)$, the new support-weighted internode distance replacing the internode distance from step 2 of ASTRID:

$$d_G(u, v) = \sum_{e \in P_G(u,v)} s(e) \tag{3.1}$$

In reality, several different measures of support exist with different running time and accuracy trade-offs [144]. As reported in [85], the approximate Bayesian support [144] of IQ-TREE led to the most accurate species tree reconstruction, although other measures of support also led to accuracy improvements over unweighted ASTRAL. We leave this choice of support as a parameter to be decided later for wASTRID-s in an experiment for parameter exploration (Experiment 1).

Distance defined by branch lengths

Unlike branch support, which is designed to be a measure of statistical confidence on the correctness of a branch, branch lengths can only serve as proxies to such information, where shorter branches are empirically harder to estimate, likely due to containing less phylogenetic signal (fewer substitutions) [145]. We do not attempt a complex conversion here, and simply just assign the branch length as the confidence similar to how we use the support values:

$$d_G(u, v) = \sum_{e \in P_G(u,v)} l(e) \tag{3.2}$$

Notably, this definition of $d_G(u, v)$ coincides with STEAC [146] (motivated by a different perspective of the coalescence time between genes), and potentially under a more accurate setting when paired with the FastME step of ASTRID. In addition, we also explore whether and how to normalize the input branch lengths of the gene trees for the weighting. We name this final algorithm **wASTRID-pl** (weighted ASTRID by path-lengths).

3.2.3 Running Time

The runtime for wASTRID-s and wASTRID-pl consists of two parts: the calculation of the distance matrix and then running FastME. Here we derive the runtime for each part.

Theorem 3.1. The average internode distance matrix of wASTRID-s and wASTRID-pl can be obtained in $O(kn^2)$ time, where n is the number of species and k is the number of genes. The runtime using FastME on this input matrix, and using p NNI or SPR moves, is $O(pn^2)$. Hence, the total runtime is $O((k + p)n^2)$.

Proof. We begin by demonstrating that the asymptotic runtime for calculating the distance matrix is the same for wASTRID-s, wASTRID-pl, and ASTRID. For each gene tree, our metricization simply assigns already-computed values as lengths to each edge; thus, using dynamic programming, calculating the internode distance across all pairs of taxa per gene tree takes $O(n^2)$ time (for normalizing branch lengths in wASTRID-pl, we only explore ways to normalize that do not affect this asymptotic running time). Hence, the average internode distance matrix can be calculated in $O(kn^2)$ time.

The runtime for FastME, given the distance matrix, depends on the number p of NNI or SPR moves. According to [83], [147], FastME optimizing BME, using NNI or SPR moves, uses $O(n^2)$ for the starting tree and $O(n^2 + pn \times diam(T))$, where $diam(T)$ is the topological diameter of the output tree and p is the number of moves, for the heuristic search. Since $diam(T) \leq n - 1$, this is $O(pn^2)$. QED.

We also re-implemented the calculation of the distance matrix to improve empirical runtime compared to the available version of ASTRID in [143]. The asymptotically optimal algorithm is easy to devise because the naive algorithm, which, given a gene tree, starts a tree traversal at each leaf to obtain the all-pairs internode distance, is already quadratic time per gene and also asymptotically optimal due to each gene tree having $\binom{n}{2}$ distances. The original ASTRID implementation, in this vein, uses an algorithm which implicitly performs multiple traversals in the tree. We instead implemented an internode-distance algorithm from TreeSwift [148] based on post-order traversal, through which we hope to achieve better empirical performance

due to better cache locality in its simultaneous maintenance of multiple distances from the leaves in an array. This produces for us a speed advantage for wASTRID as well as an alternative way of running ASTRID, which we refer to as ASTRID-3.

3.3 EXPERIMENTAL STUDY

3.3.1 Overview

We conduct five experiments. The first experiment is parameter exploration and uses training data; all subsequent experiments use the testing data, which are disjoint from the training data.

- Experiment 1: we explore parameter choices (choice of branch support for wASTRID-s and normalization scheme for wASTRID-pl) for weighted ASTRID.
- Experiment 2: we compare the accuracy and running time of weighted ASTRID against other methods on a diverse set of simulated conditions, where all genes are complete, and so have all the species.
- Experiment 3: we compare ASTRID, wASTRAL-h, and the best variant of wASTRID (as determined by previous experiments) on the Jarvis et al. [149] avian biological dataset.
- Experiment 4: we explore weighted and unweighted ASTRAL and ASTRID on datasets where the average internode distance matrix might be incomplete (i.e., where there is at least one pair of taxa that do not appear together in any gene tree). Specifically, we explored two models of missing data, one uniformly deleting a fixed number of taxa across gene trees, and also a "clade-based" model of missing data, denoted by M_{clade} from [150]. Both types of missing data results in gene trees that each almost always has taxa missing.
- Experiment 5: we conducted a detailed running time comparison between ASTRID-2 and ASTRID-3, where ASTRID-3 differs from ASTRID-2 by a faster implementation for calculating the average internode distance matrix.

3.3.2 Datasets

We assembled a set of diverse data from prior studies (see Table 3.1), consisting of various simulated conditions with estimated gene trees and one biological dataset ("avian

Dataset	Taxa & Genes	# reps	AD % (ILS)	Branch Sup.	Exp.
ASTRAL-II MC2 [71]	201, 1000	10	33 (M)	SH/BS/aBayes	1
ASTRAL-III S100 [151]	101, 1000	50	46 (H)	aBayes	2,5
M_{clade} missing data	101, 200	50	46 (H)	aBayes	4
iid missing data	101, 200	50	46 (H)	aBayes	4
ASTRAL-II MC3 [71]	201, 1000	50	21 (L)	aBayes	2
ASTRAL-II MC5 [71]	201, 1000	50	34 (M)	aBayes	2,5
ASTRAL-II MC1 [71]	201, 1000	50	69 (VH)	aBayes	2
ASTRAL-II MC6H [152]	201, 1000	50	9 (L)	aBayes	2
ASTRAL-II MC11H [152]	1001, 1000	50	35 (M)	aBayes	2,5
Avian 2x [69]	48, 1000	20	29 (M)	aBayes	2
Avian 1x [69]	48, 1000	20	47 (H)	aBayes	2
Avian 0.5x [69]	48, 1000	20	60 (VH)	aBayes	2
Mammalian 2x [69]	37, 200	20	21 (L)	aBayes	2
Mammalian 1x [69]	37, 200	20	29 (M)	aBayes	2
Mammalian 0.5x [69]	37, 200	20	50 (H)	aBayes	2
Jarvis et al. avian [149]	48, 14446	1	not known	BS	3

Table 3.1: Dataset statistics. The datasets include both simulated datasets (with known true gene and species trees) and the avian dataset from [149]. ILS levels are categorized by AD percentages: low ILS (L) $< 25\%$, mid ILS (M) 26-39%, high ILS (H) 40-59%, very high ILS (VH) above 59%. Branch support methods: SH-like (FastTree default), BS (standard bootstrap with FastTree or RAxML), aBayes (IQ-TREE approximate Bayesian support).

biological") from the avian phylogenomics project [149]. We separate the datasets into training (Experiment 1) and testing (Experiments 2–5).

We use the nomenclature of the original ASTRID study and refer to the SimPhy-simulated datasets from the ASTRAL-II study [71] by an "MC" name (where "MC" refers to "model condition"). We additionally replaced some ASTRAL-II datasets by some "H" variants generated by a prior separate study [152] to induce a level of gene tree estimation error closer to the rest of the conditions, which span from 23% to 55%. Recall that GTEE in simulated datasets is commonly measured by the average RF distance between the estimated gene trees and the true gene trees. The ILS levels of the datasets are measured in average discordance (AD), defined as the average nRF rate between the true species tree and the true gene trees. Like the original ASTRID study [5], we classify the ILS levels of the datasets into four categories according to their AD values, where below 25% is classified as low ILS (L), between 26% and 39% medium ILS (M), between 40% and 59% high ILS (H), and higher AD considered very high ILS (VH). For the simulated conditions, we subsample k of the gene trees with $k = 50, 200, 1000$, except for the mammalian simulation where we sample $k = 50, 100, 200$ instead, as only 200 gene trees were provided.

In Experiment 4, we evaluate methods on datasets where some gene trees are missing one

or more species. For this experiment, we created missing data variants of the S100 (seqlen[1] = 400) dataset under two models of missing data: the *i.i.d.* model and the M_{clade} model. For the *i.i.d.* case, we deleted a fixed number of taxa ($20\%, 40\%, 60\%, 80\%$ of the total number of taxa) from each gene; thus, every gene tree is incomplete (i.e., misses at least one species). For the M_{clade} model, the process is more complicated. Briefly, each gene tree uniformly samples a clade from the species tree above a certain size (a parameter x relative to the number of taxa in the species tree), and only retains taxa from this clade. We varied x in $0.2, 0.4$ and 0.6, which resulted in an average percentage of taxa being deleted of $46\%, 23\%$ and 10%, respectively for the M_{clade} model. Thus, for the M_{clade} model, it is *possible* for a gene tree to be complete (i.e., have all the species), but not very likely for this to occur.

We now describe how we compute branch support in our experiments. The weighted ASTRAL study [85] provided gene trees reannotated with aBayes support [144] and edge lengths inferred using IQ-TREE [67] for the ASTRAL-II and ASTRAL-III datasets. We also reannotated the avian and mammalian simulation with aBayes support and IQ-TREE edge lengths, because aBayes was determined as the best measure of support also for wASTRID-s. We use the MC2 condition as the training data for both wASTRID-s and wASTRID-pl, where to explore the choice of edge support for wASTRID-s, we took the original gene trees estimated using FastTree [64] and estimated branch support using two techniques: the default FastTree SH-like support and also bootstrap support [90] using 100 bootstrap trees. For Experiment 4, we only used the IQ-TREE aBayes support, computed on the reduced gene trees and using the reduced alignments. See Appendix A for additional discussion about branch support.

3.3.3 Methods

We ran weighted and unweighted versions of ASTRAL and ASTRID.

- **ASTRID** (v2.2.1), available at [143]. With the exception of Experiment 4, we turn off missing data imputation.
- **wASTRID**, available at [153], missing data imputation will be automatically turned on by the software when necessary.
- **ASTRAL(-III)** (v5.7.8), available at [137]. Although ASTRAL-III enables analyses of gene trees with low-support branches collapsed, we use the fully resolved gene trees as input, as this allows us the fully explore the impact of weighting.

[1] "Sequence length", a proxy of how much signal the phylogeny estimation software has from the gene trees. Higher amounts of signal usually entails more accurate reconstruction.

- **wASTRAL-h** (hybrid weighted ASTRAL, v.1.4.2.3). This was the most accurate version of weighted ASTRAL from the original study [85], using both branch lengths and support to weight gene tree quartets. wASTRAL-h supports parallelization, so we run wASTRAL-h with 16 threads. wASTRAL-h is available as part of the ASTER [86] software suite.

For all analyses using wASTRID and ASTRID, when we ran FastME to search for a BME solution, we ran FastME with the most accurate setting that uses the same greedy BME starting tree and follows with both NNI and SPR moves to improve the score.

Many of the datasets in our study have FastTree-inferred gene trees that were reannotated with IQ-TREE approximate Bayesian support. FastTree-inferred trees have polytomies when the input has identical sequences, but these polytomies will be resolved when the trees are reannotated by IQ-TREE. Since polytomy resolution may add false positive edges that adversely affect the accuracy of the unweighted methods, in these cases, we run the unweighted methods on the original FastTree gene trees.

3.3.4 Evaluation Criterion

For simulated datasets, we report the topological error rate of the reconstructed species trees using the normalized Robinson-Foulds error (nRF error) [52] with respect to the true species trees, where the nRF error rate is the number of bipartitions in the estimated and true species trees that do not appear in both trees, normalized by the total number of non-trivial bipartitions in the two trees. Because all the inferred and true species trees are binary, the nRF error rate is the same as the missing branch rate (i.e., the fraction of the non-trivial bipartitions in the true species tree that are missing from the reconstructed tree). For these calculations, the non-trivial bipartitions of a tree are defined by the internal edges in the tree, i.e., the edges that are not incident to leaves.

On the avian biological dataset, as the true tree is not known, we compare the estimated species trees against prior topologies (wASTRAL tree and published trees). We also compute the local posterior-probability (localPP) branch support [154] for the reconstructed species trees obtained using wASTARL-h to assess the reliability of the branches.

On all datasets, we keep track of the wall-clock running time of the methods, the time taken from consuming the input gene trees (that may have been preprocessed with new branch support values) until outputting the species tree.

3.3.5 Experimental Environment

All experiments before Experiment 5 were conducted on the Illinois Campus Cluster, a heterogeneous cluster that has a four-hour running time limit, with at least 64 GB memory available. The heterogeneity of the hardware makes the wall-clock running times not directly comparable across runs, but can still be used to gather obvious running time trends.

All runs in Experiment 5 were performed serially on an Apple M1 Macbook Pro (model Z11B000E3LL/A) with 16GB memory available to ensure accurate comparison of running times.

3.4 RESULTS

3.4.1 Experiment 1: Parameter Selection

In Figure 3.1, we explore the choice of branch support among the default FastTree SH-like support, IQ-TREE approximate Bayesian (aBayes) support (normalized to the $[0, 1]$ range), and bootstrap support (100 FastTree replicates) on the training datasets. The best accuracy of wASTRID-s is obtained by using the normalized aBayes support on gene trees. All measures of support, however, improved the species tree estimation error in general. The superiority of (normalized) aBayes support is consistent with the support chosen for weighted ASTRAL, where it was also found superior to SH-like support and bootstrap support. This advantage is even more pronounced when considering that aBayes support can be obtained much faster than bootstrap support [144].

On this training dataset, wASTRID-pl attained the highest accuracy when normalizing the branch lengths in each gene tree by the maximum path length in that gene tree (better than no normalization). Interestingly, while worse than ASTRID with fewer genes $k \in \{50, 200\}$, wASTRID-pl attained higher accuracy than ASTRID when $k = 1000$. However, when comparing wASTRID-s and wASTRID-pl, wASTRID-pl was always less accurate.

3.4.2 Experiment 2: Results on Simulated Datasets

In this experiment, we show four-way comparisons among ASTRID, ASTRAL, weighted ASTRAL (wASTRAL-h), and weighted ASTRID (wASTRID-s). We put an emphasis on the accuracy (nRF error), while later revisiting the problem of running time.

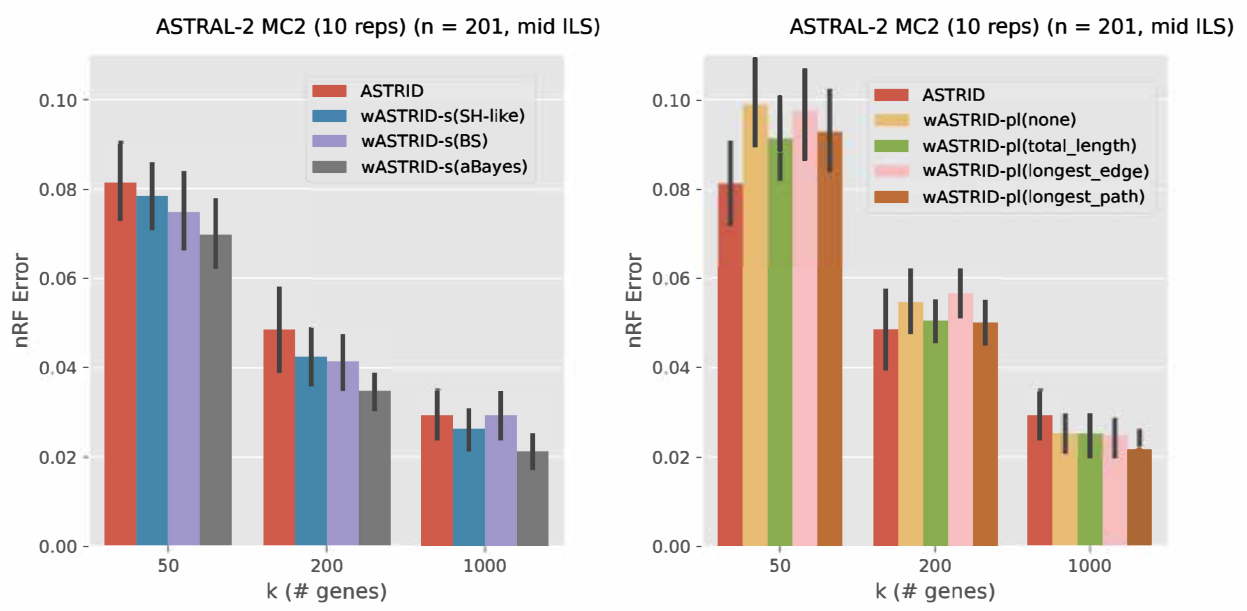


Figure 3.1: Comparison of the choice of branch support and the choice of branch length normalization strategy for wASTRID-s and wASTRID-pl respectively on the training data, showing the species tree topological error rates (nRF error). "SH-like" is FastTree SH-like support. "aBayes" is IQ-TREE approximate Bayesian support. "BS" is bootstrap support using 100 FastTree trees. The x-axis varies in the number of genes k in the input. Results are shown averaged across ten replicates. Error bars show standard error.

ASTRAL-III S100

This 101-species dataset contains four conditions that varied in the gene tree estimation error (GTEE, measured by the average nRF error between the estimated gene trees and their corresponding true gene trees) by varying the sequence lengths. We show the results in Figure 3.2. We show the unweighted methods (i.e., ASTRID and ASTRAL) in dotted lines.

Many trends are as expected. For example, summary methods become more accurate as k (the number of genes) increases, and all methods also improve in accuracy when given more accurate estimated gene trees. These two trends are unsurprising and well-documented across studies for summary methods in general (e.g., [60]).

More interestingly, the weighted methods (wASTRID-s, wASTRAL-h) are clearly more accurate than their unweighted counterparts, especially at higher levels of GTEE (GTEE $= 0.55, 0.42$). The improvement in accuracy from the weighted methods does not seem to depend on the number of genes, suggesting that the noise brought by low-quality gene trees is not resolved by having ample data. This advantage of the weighted methods, however, is smaller as more accurate gene trees are used (GTEE $= 0.31$), as expected.

wASTRID-s clearly improves upon ASTRID on this dataset, across all conditions and all

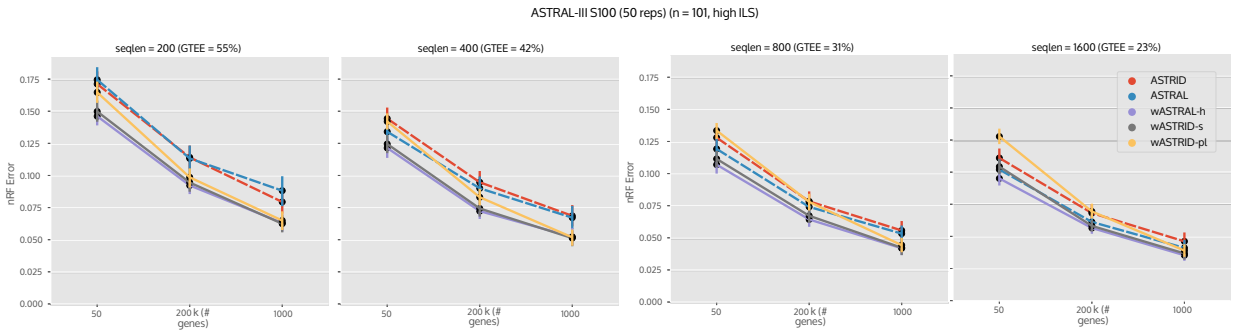


Figure 3.2: Topological error of species tree across methods on the ASTRAL-III S100 dataset ($n = 101$, AD $= 46\%$). Subfigures vary the sequence lengths, affecting the gene tree estimation error (measured in GTEE, the average distance between estimated gene trees and true gene trees). The x-axis varies in the number of genes. Results are shown averaged across 50 replicates with standard error bars. All weighted methods (wASTRID, wASTRAL) ran on gene trees reannotated with IQ-TREE aBayes support branch support and lengths. All methods achieve better accuracy when given more gene trees (larger k) or more accurate gene trees (lower GTEE). Weighted methods are more accurate than unweighted ones. wASTRID-s and wASTRAL-h have almost the same accuracy.

numbers of genes. wASTRID-s notably almost matches the accuracy of wASTRAL-h. With $k \in \{200, 1000\}$, no clear benefit exists for using wASTRAL-h on the shown conditions.

On this dataset, wASTRID-pl, similar to trends seen in the training dataset, attains better accuracy than ASTRID when $k = 1000$, but is almost always worse than wASTRID-s.

ASTRAL-II SimPhy

We show the results of the ASTRAL-II SimPhy datasets in Figure 3.3. Across all conditions in this dataset, the weighted methods are more accurate than the unweighted methods. This advantage does not seem to depend on the level of ILS or the number of species. Even under the easiest condition (MC3), wASTRAL-h and wASTRID-s still consistently achieved better accuracy. All methods also performed worse in accuracy as ILS increased, as expected.

While wASTRID-s still consistently improved upon ASTRID in accuracy on this dataset, we also see datasets where wASTRID-s is worse than wASTRAL-h. The relative performance of wASTRID-s and wASTRAL-h seems related to the relative performance of the base methods: MC1 and MC6H are the two conditions that ASTRAL is in general more accurate than ASTRID, but the relative performance of the base methods does not explain the whole picture – for MC1 going to $k = 1000$, ASTRID became more accurate than ASTRAL yet wASTRID-s is still worse than wASTRAL-h. More positively, on the other conditions of this dataset, wASTRAL-h has nearly the same accuracy as wASTRID-s, although wASTRAL-h

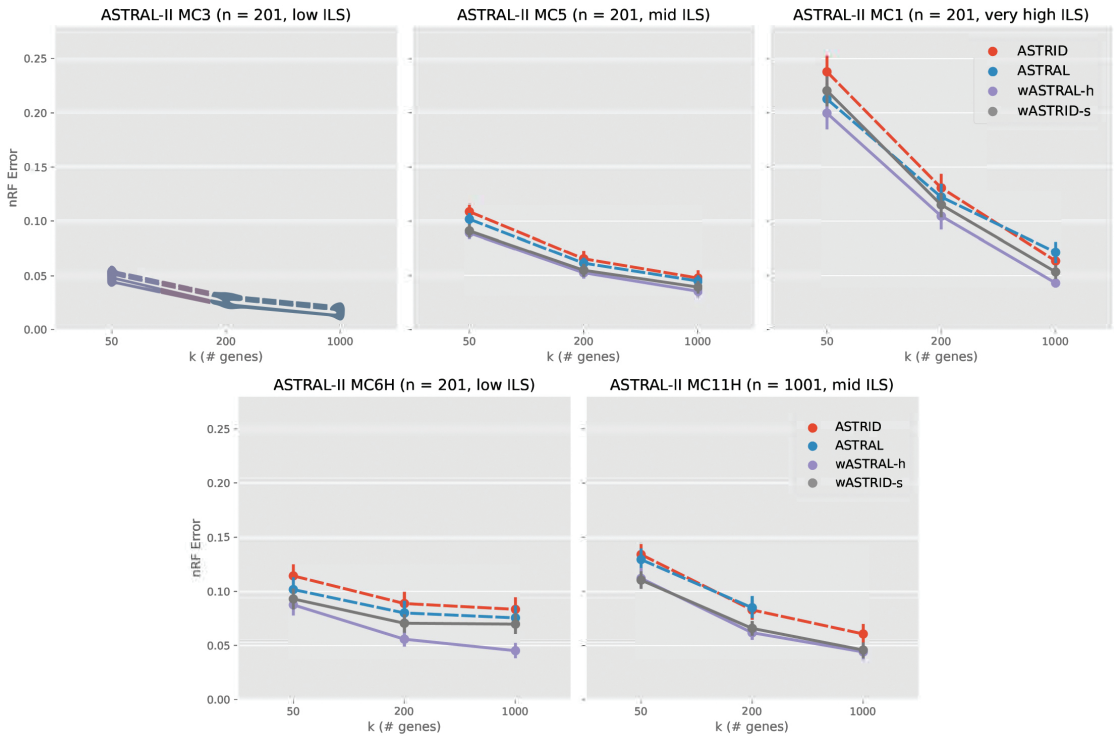


Figure 3.3: Topological error (nRF error rate) of species tree across methods on selected conditions on the ASTRAL-II SimPhy conditions ($n = 201, 1001, 201$, AD $= 21, 35, 69\%$ respectively). Each subfigure depicts a different model condition. The x-axis varies in the number of genes. Results are shown averaged across 50 replicates with standard error bars. ASTRAL did not finish 24 out of the 50 replicates within four hours for $k = 1000$ on MC11H and thus the data point was omitted. All weighted methods (wASTRID, wASTRAL) were run on gene trees reannotated with IQ-TREE aBayes support branch support and lengths. Weighted methods are more accurate than unweighted ones. wASTRID-s on MC1 was less accurate than wASTRAL-h and otherwise has the same accuracy.

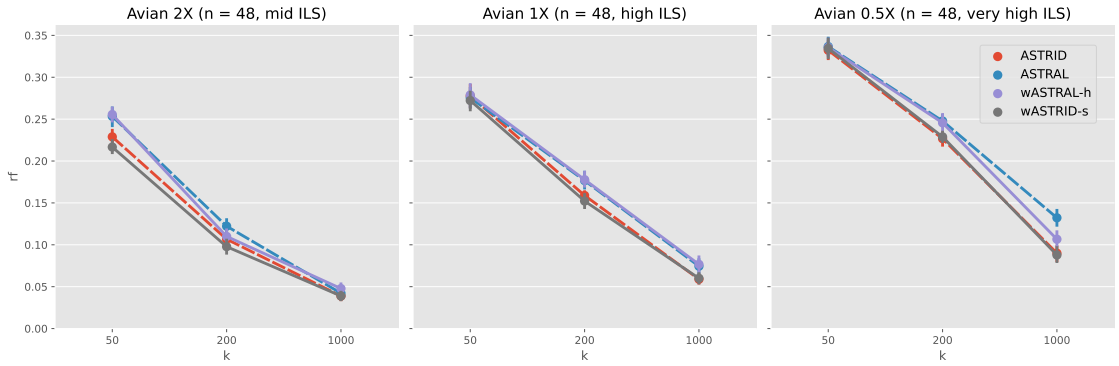


Figure 3.4: Topological error (nRF error rate) of species tree across methods on the avian simulation ($n = 48$). Each subfigure depicts a different model condition. The x-axis varies in the number of genes. Results are shown averaged across 20 replicates with standard error bars. All weighted methods (wASTRID, wASTRAL) ran on gene trees reannotated with IQ-TREE aBayes support branch support and lengths. ASTRID and wASTRID-s are more accurate than ASTRAL and wASTRAL-h, with a slight accuracy advantage to the weighted methods over the unweighted ones.

is marginally more accurate, which might be due to the hybrid weighting of wASTRAL-h, which also incorporates the branch lengths for better accuracy.

The model condition with the largest number of species, i.e., MC11H with 1001 species, presented computational challenges for ASTRAL. Specifically, on roughly half of the replicates of model condition MC11H with $k = 1000$ genes, ASTRAL did not finish under our four-hour time limit (see Appendix A Section A.2.1 for more details), but wASTRAL-h did.

Avian and mammalian simulations

The avian and mammalian simulation have model trees inferred on biological datasets. Both datasets have three conditions with varying ILS by scaling the model tree branch lengths by 2X, 1X, or 0.5X, with shorter branch lengths leading to higher degrees of ILS. Notably, prior results [5] showed that ASTRID outperformed ASTRAL on the avian simulation in accuracy, while on the mammalian simulation ASTRAL was more accurate. Also, the mammalian simulation only has 200 genes available, so we vary k among $50, 100, 200$ unlike the other datasets.

On the avian simulation (Figure 3.4), aside from obvious trends (ILS increases difficulty; more genes leads to more accurate reconstruction), same as the original study [5], ASTRID is consistently more accurate than ASTRAL. Strangely, although the weighted methods inherit the relative performance of their base methods, in a few cases the weighted methods do not help in accuracy, but they do not erode accuracy either. Even on conditions where the

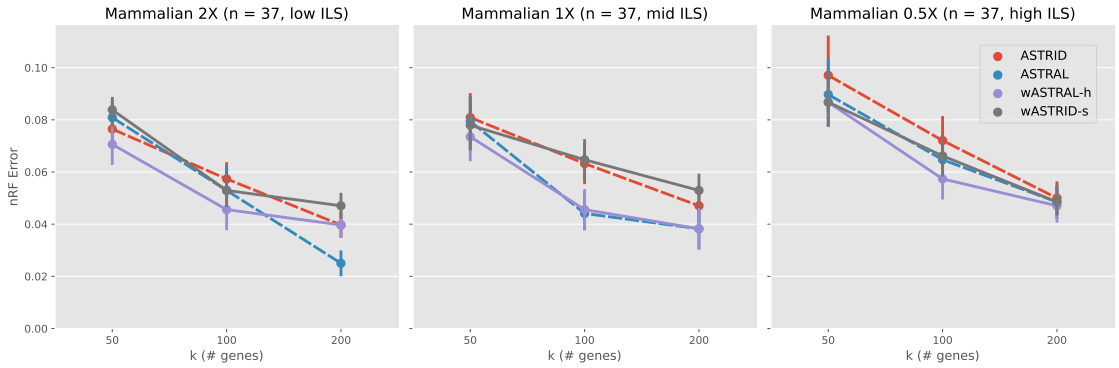


Figure 3.5: Topological error (nRF error rate) of species tree across methods on the mammalian simulation ($n = 37$). Each subfigure depicts a different model condition. The x-axis varies in the number of genes. Results are shown averaged across 20 replicates with standard error bars. All weighted methods (wASTRID, wASTRAL) ran on gene trees annotated with IQ-TREE aBayes support branch support and lengths. ASTRAL and wASTRAL-h are more accurate than ASTRID and wASTRID-s. The weighted methods have mixed accuracy compared to the unweighted ones.

weighted methods improved accuracy, the improvement was small. For example, wASTRAL-h, even though improving upon ASTRAL, is even less accurate than ASTRID, whereas on previously shown data wASTRAL-h was consistently the best in accuracy. This avian simulation does carry substantial GTEE ($> 50\%$), so it is not clear what led to the weighted methods underperforming.

The results for the mammalian simulation (Figure 3.5) paint a more perplexing picture. On the 2X condition, surprisingly, the weighted methods are less accurate than their unweighted counterparts in general. This trend continues with the 1X condition, where wASTRAL-h only mostly matches ASTRAL in accuracy, and wASTRID-s is worse than ASTRID in accuracy. Only on the 0.5X condition do both weighted methods clearly help in accuracy. wASTRAL-h is clearly better than wASTRID-s on this dataset, but this difference can be explained by the accuracy advantage of ASTRAL on ASTRID. While it is again unclear why the weighted methods underperformed, this dataset is relatively easy, compared to the previously shown datasets, with all methods achieving at most 0.05 nRF error rate at $k = 200$; thus, despite the puzzling relative performance of weighted and unweighted methods, the difference in accuracy among methods is very small.

Running time

We show the wall-clock running time of the four methods under three representative conditions ($n = 101, 201, 1001$) in Table 3.2, with a direct comparison of the two most

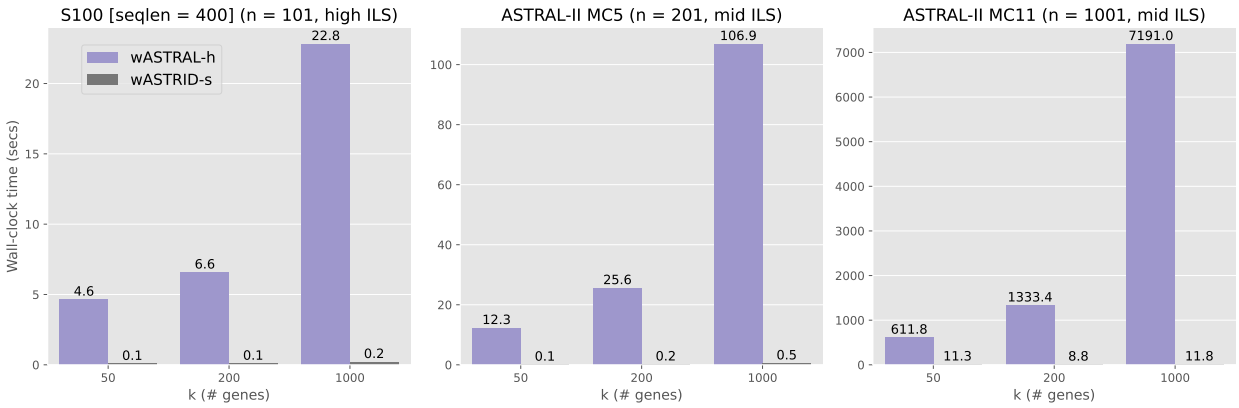


Figure 3.6: Wall-clock running time (sec) comparison of wASTRID-s and wASTRAL-h on selected representative simulated conditions on $n = 101, 201, 1001$ for k ranging in $50, 200, 1000$. Bars and labels show averages across 50 replicates. wASTRID-s is dramatically faster than wASTRAL-h.

accurate methods visualized in Figure 3.6. While the heterogeneity of the hardware dilutes the comparability of the running times, clearly wASTRID-s and ASTRID are much faster than wASTRAL-h and ASTRAL, with wASTRID-s on average taking less than 12 seconds even on the largest input, whereas on the same input wASTRAL-h on average takes roughly two hours. In general, wASTRID-s is around two orders of magnitude faster than wASTRAL-h. Although we note that the default flags of both ASTRAL and wASTRAL-h (that we used in the experiments) also calculate support and lengths for reconstructed species tree, in practice, this is a fast step relative to the species tree reconstruction, and does not affect our running time analysis in any substantial way. On MC11H, wASTRID-s and ASTRID took less time going from $k = 50$ to 200, likely due to the species tree inferred under the smallest input size experimented ($k = 50$) having larger diameters, negatively impacting the FastME step running time which has a linear dependency on the diameter of the output tree.

The weighted methods are faster than their unweighted counterparts. For example on S100 (seqlen = 400) with 1000 genes, wASTRAL-h is more than ten times faster than ASTRAL. In addition ASTRAL did not finish for approximately half of the datasets for the largest input (MC11H, $k = 1000$). Aside from the benefit of parallelization (we ran wASTRAL-h using 16 threads, but off-the-shelf ASTRAL does not support parallelization), this speed advantage under a large number of genes of wASTRAL-h over ASTRAL can also be attributed to the algorithmic change implemented in wASTRAL-h. The new weighted ASTRAL algorithm removes the in-practice quadratic dependency of ASTRAL's search algorithm on the number of genes [4] and instead has a linear dependency on k for the running time. wASTRID-s is consistently faster (at least two times faster on most conditions) than ASTRID, showing that

Running time (s)	k (# genes)	ASTRAL	ASTRID	wASTRAL-h	wASTRID-s
S100 ($n = 101$)	50	12.1	0.1	4.6	0.1
	200	29.6	0.2	6.6	0.1
	1000	300.7	1.4	22.8	0.2
MC5 ($n = 201$)	50	20.3	0.2	12.3	0.1
	200	57.5	0.6	25.6	0.2
	1000	600.7	1.8	106.9	0.5
MC11H ($n = 1001$)	50	552.9	18.0	611.8	11.3
	200	2059.6	14.9	1333.4	8.8
	1000	–	27.5	7191.0	11.8

Table 3.2: Wall-clock running time (sec) across methods on selected representative simulated conditions on $n = 101, 201, 1001$ for k ranging in $50, 200, 1000$. Data points show averages across 50 replicates. ASTRAL did not finish on 24 out of the 50 replicates within four hours for $k = 1000$ on MC11H, and thus the data point was omitted. The methods, sorted by the fastest to the slowest, are almost always wASTRID-s, ASTRID, wASTRAL-h, and ASTRAL across all shown conditions. ASTRID and wASTRID-s are much faster than ASTRAL and wASTRAL-h.

the new distance calculation algorithm implemented is more efficient than the original one.

3.4.3 Experiment 3: Results on the Avian Biological Dataset

Jarvis et al. studied the phylogeny of birds using a dataset on 48 taxa using 14446 genes [149]. The original gene trees were annotated with RAxML [17] bootstrap support, which we directly use in our wASTRID-s analysis. This dataset is known to have very low gene tree resolution, with the average branch support only 32% [69].

ASTRAL on the original set of gene trees reconstructed a species tree that failed to reproduce a few well-supported clades from prior studies, such as the Columbea clade. Nonetheless, contracting low support branches in the gene trees enabled ASTRAL to construct a very plausible topology in agreement with prior studies [151]. In a subsequent study, Zhang and Mirarab reanalyzed the original gene trees using wASTRAL-h and reconstructed the same topology as ASTRAL running on contracted gene trees [85].

We computed trees using both ASTRID and wASTRID on the Jarvis et al. [149] dataset, and compare them to these other estimated trees. We also compute the number of differing branches between the inferred trees and the two published trees of the Jarvis et al. study [149]: the ExaML-based concatenation tree, called the TENT ("total evidence nucleotide tree"), and the coalescent-based published tree based on the result of running MP-EST with

statistical binning [69] (i.e., the MP-EST* tree). We also examine whether these trees include the six clades that are proposed by Braun & Kimball to be strongly corroborated [155] for the avian phylogeny.

The wASTRID-s and wASTRAL trees are shown in Figure 3.7 and the ASTRID tree is shown in Figure 3.8, with branch support given by the localPP criterion. Note that all trees display the six reliable avian clades of Braun & Kimball. Nevertheless, some interesting differences do appear.

The ASTRID tree (shown in Figure 3.8) differed in eight or nine edges with each of the wASTRAL-h, TENT, and MP-EST* trees. The ASTRID tree did not recover the Columbea clade, which has also seen strong support in various analyses of this data [149], [151], [156].

Using wASTRID-s, we recovered a topology that is much more in agreement with the other trees, differing in two branches (4.4% of the branches) with the wASTRAL-h tree. It displays the six well-corroborated clades, and is also in much higher agreement with the published trees (differing in three branches with the TENT, two branches with the MP-EST* tree). Looking closer, the two branches where it differs from the wASTRAL-h tree coincide with the only two extremely low support branches (no more than 5% in bootstrap support), and thus contracting these branches in the wASTRID-s tree arrives at a topology that is a contraction of the wASTRAL-h tree. Both wASTRID-s and wASTRAL-h recover the Columbea-Passerea split, a major conclusion in the original analysis of this data, and even agree on the placement of the hard-to-place Hoatzin.

For running time, both ASTRID and wASTRID-s finished quickly. ASTRID completed in 6.4 seconds, and wASTRID-s completed in 1.1 seconds. Our rerun of wASTRAL-h on this data finished in 294.2 seconds, showcasing the much better scalability of the new weighted ASTRAL optimization algorithm in the number of genes, whereas ASTRAL on the same input took 32 hours in [151].

In summary, on this dataset, wASTRID-s inferred a more accurate tree compared to ASTRID, is much faster than wASTRAL-h, and is compatible with wASTRAL-h after contraction of two very low support branches.

3.4.4 Experiment 4: Accuracy Given Incomplete Gene Trees

In real-world datasets, genes that are incomplete, so that they miss some species, are commonplace. This "missing data" condition can reduce accuracy, as shown in [60], [150], and the impact of this is studied in this experiment.

Random missing data

We show the impact of missing data using a model of uniformly deleting a fixed number of taxa across gene trees in Figure 3.9. As the percentage of missing data increases, all methods increase in error, an expected trend as the amount of information in the input decreases. Comparing the weighted and unweighted versions, weighted methods were more accurate than their unweighted counterparts under low to moderate (20% to 60% taxa randomly missing) missing data. However, under the highest level of missing data (80% taxa missing), ASTRAL achieved the best accuracy across all methods, showing a robustness against extreme missing data while wASTRAL-h achieved the worst accuracy at this extreme. Comparing horizontally between the ASTRID and ASTRAL variants, we see a slight advantage of ASTRAL in accuracy across levels of randomly deleted taxa, suggesting ASTRAL might be more robust to this *i.i.d.* model of missing data. Although we only show results on one condition, our results suggest that the weighted methods still offer accuracy advantages except on the more extreme conditions of missing data, and both ASTRID variants and ASTRAL variants demonstrate reasonable robustness against missing data.

Clade-based missing data

We show the results under the M_{clade} based missing data in Figure 3.10. Recall that each gene tree uniformly samples a clade from the species tree above a certain size (a parameter $x = 0.2, 0.4, 0.6$ relative to the number of taxa in the species tree), and only retains taxa from this clade. This process resulted in average percentage of taxa being deleted of $46\%, 23\%$ and 10% across the genes.

Under these clade-based missing data conditions, the weighted methods are consistently more accurate than the unweighted variants. The most accurate method is wASTRID-s (tied with wASTRAL-h under the lowest level of missing data, the most accurate under higher levels). The ASTRID variants do not observably degrade in accuracy as the percentage of deleted taxa increases while the ASTRAL variants do. This relative robustness of weighted ASTRID to the clade-based missing data is largely expected given ASTRID's robustness shown in the prior study of [150].

3.4.5 Experiment 5: Detailed Comparison of the Distance Matrix Algorithm

Recall that ASTRID variants first calculate an average distance matrix D from the gene trees, and then run FastME on D. From previously shown results in Table 3.2, wASTRID-s is

clearly faster than ASTRID, only possible due to our change in the distance matrix calculation algorithm. To better benchmark this running time difference brought by this new algorithm for producing D, we broke down the running time of ASTRID-2 (the most efficient known implementation) and ASTRID-3, which is the ASTRID algorithm implemented inside our wASTRID software. We show the results in Figure 3.11, benchmarking on 1000 gene trees.

The runtimes for both ASTRID-2 and ASTRID-3 on the two smaller datasets are both small, but with an advantage to ASTRID-3. However, on the largest dataset with 1001 species and 1000 genes, we see large differences in runtime. Unsurprisingly perhaps, the running time for FastME is essentially the same for both ASTRID and wASTRID, so that the runtime difference between the two methods is due to the time used to calculate the distance matrix, D. A detailed examination of that runtime (Table A.1) shows that ASTRID-2 takes at least 3 times as long as ASTRID-3 to compute the distance matrix, across all conditions. The constant-time speed up is consistent with the fact that both ASTRID and wASTRID implement $O(n^2k)$ algorithms for calculating D.

3.5 DISCUSSION

Here we discuss the trends seen in this study, and how they compare to trends from previous studies. We begin with a discussion about accuracy before turning to computational performance.

3.5.1 Accuracy

Overall trends

For all model conditions, all tested methods achieved better accuracy when given more genes or as ILS or GTEE decreased. These trends are well known and quite expected for all summary methods [60]. The accuracy advantage of weighted methods over unweighted methods is present in nearly all conditions, but is larger for the conditions with shorter sequences (equivalently, the higher GTEE conditions) and fewer genes. This trend was already established for wASTRAL-h compared to ASTRAL in [85], but shows that wASTRID-s, just like ASTRAL, is able to better interpret the signal in the input through taking branch support into consideration.

In most cases, weighting improved accuracy for both ASTRAL and ASTRID, but not for the simulated mammalian dataset, where both ASTRAL and ASTRID had conditions where they were more accurate than their weighted versions, and not for the extreme missing data

condition (80% taxa missing), where ASTRAL (but not ASTRID) had a condition where it was more accurate than its weighted version. While we do not understand why this occurs, it is noteworthy that both ASTRAL and ASTRID had conditions where they were more accurate than their weighted versions.

wASTRAL-h vs. wASTRID-s

ASTRAL and ASTRID are known to be among the most accurate summary methods under ILS, and their relative accuracy is dataset dependent, as also shown in our results. The relative accuracy of their weighted counterparts is clearly influenced by the accuracy of their base methods, as seen in the performance differences in the avian and mammalian simulation, where either wASTRID-s and wASTRAL-h can be the more accurate method. On the ASTRAL simulated datasets (ASTRAL-III S100, ASTRAL-II SimPhy), wASTRID-s and wASTRAL-h have comparable accuracy, with a small advantage to wASTRAL-h.

The improvement of wASTRAL-h over wASTRID-s may be due to its weighting also incorporating branch lengths. A branch-length weighted version of ASTRID (wASTRID-pl) has mixed accuracy compared to ASTRID, and does not compete against the support-weighted wASTRID-s in accuracy. In general, wASTRID-s and wASTRAL-h serve as accurate species tree inference methods under ILS and are more robust to GTEE than ASTRID and ASTRAL, two of the most accurate summary methods. Both can scale very well, with wASTRID-s much faster. However, the differences in accuracy are dataset dependent, just as the comparison between ASTRAL and ASTRID for accuracy seems dataset dependent.

Impact of missing data

In this study, we examined two models of missing data: the more commonly studied *i.i.d.* model and the M_{clade} model. As expected, error rates increase with deletions of taxa, a trend that arises from reduction in information content (and has been noted in other studies [60], [150]). Of greater interest, however, is the relative impact on ASTRAL and ASTRID, and on their weighted versions. We see that ASTRAL (weighted and unweighted) is more accurate than ASTRID (weighted or unweighted) under the *i.i.d.* model of missing data, but the reverse occurs for the clade-based model of missing data; this trend was already observed in [150] for ASTRID and ASTRAL, and it is noteworthy that it appears for their weighted versions as well.

We also see that weighted versions are generally more accurate than unweighted versions under missing data, except that ASTRAL is more accurate than wASTRAL-h for the

condition of a very high degree of *i.i.d.* missing data. This trend was not reported in [85], but we note that they only examined very low levels of missing data (5% of the taxa deleted per gene). Thus, overall our findings are consistent with prior studies, and show both that the weighted versions of ASTRID and ASTRAL have comparable levels of robustness to missing data (with advantage to wASTRID for clade-based missing data and advantage to wASTRAL for *i.i.d.* missing data).

Given these trends, it is interesting to consider the statistical consistency guarantees for these methods under these models of missing data. Under the *i.i.d.* model, ASTRAL is statistically consistent while ASTRID is not [150], [157], but both methods are statistically consistent under the clade-based model. That said, the condition under which ASTRAL is consistent for the clade-based model requires that it be run in exact mode, rather than the heuristic mode where a set of allowed bipartitions is computed and the returned tree must draw its bipartitions from that set [150]. In this experiment, we explored ASTRAL only in heuristic mode (and hence not in such a way as to guarantee statistical consistency), and yet wASTRAL-h and ASTRAL both showed relatively good robustness, for both types of missing data. Similarly, it is worth noting that ASTRID is not statistically consistent under the *i.i.d.* model, and yet both ASTRID and wASTRID-s had accuracy that was fairly close to that of both ASTRAL and wASTRAL-h. Thus, all four methods (weighted and unweighted versions of ASTRAL and ASTRID) exhibited comparable robustness to missing data.

3.5.2 Computational Performance

All tested methods, with the exception of ASTRAL, exhibit good scalability for large datasets, as wASTRAL-h, wASTRID-s, and ASTRID can efficiently handle genome-scale data. For instance, wASTRAL-h, wASTRID-s, and ASTRID completed the analysis of the avian biological dataset, a genome-scale dataset, in under five minutes, while ASTRAL required more than a day [151]. On simulated conditions, ASTRAL was the only method that ever exceeded our computational constraints during experimentation. This may be due to ASTRAL's quadratic dependency [4] on the number of allowed bipartitions for the constrained search, which includes all the bipartitions from the input gene trees. Given many gene trees and high ILS or high GTEE, this set of allowed bipartitions can be large, which can increase the runtime for ASTRAL.

While wASTRID-s and wASTRAL-h are both very scalable, wASTRID-s is clearly faster than wASTRAL-h. Despite wASTRAL-h's improved algorithm and parallelism, wASTRID-s is still two orders of magnitude faster than wASTRAL-h, and hence might provide better scalability on large-scale data.

3.6 CONCLUSIONS

While the estimation of species trees using summary methods, such as ASTRAL, ASTRID, MP-EST, and others, is now commonplace, it is known that gene tree estimation error reduces the accuracy of the estimated species tree. We presented support-weighted ASTRID (wASTRID-s), an improvement over ASTRID that incorporates uncertainty in gene tree branches into its estimation of the average internode distance matrix. The development of wASTRID-s is inspired by the recent work of weighted ASTRAL (wASTRAL-h), which improved upon ASTRAL. We showed that wASTRID-s achieved similar accuracy improvements over ASTRID. The advantage provided by wASTRID-s over ASTRID is most noteworthy under higher degrees of gene tree estimation error. While wASTRID-s has very close accuracy to wASTRAL-h and is sometimes more accurate, overall, wASTRAL-h holds a small advantage in accuracy, whereas wASTRID-s has an advantage in speed.

Although weighting usually helped accuracy for both ASTRID and ASTRAL, there were cases where it did not, and even some where it reduced accuracy. All tested methods show a degree of robustness to missing data, but ASTRAL variants and ASTRID variants differ in their relative performance depending on the model of missing data.

This study was limited to datasets where the only cause for gene tree discordance with the species tree was ILS and gene tree estimation error. When considering real-world datasets that may have other sources of gene tree heterogeneity, such as GDL or HGT, it seems likely that ASTRAL and other quartet-based methods may have an advantage over ASTRID and wASTRID-s due to the theoretical proofs of statistical consistency for quartet-based methods under conditions involving gene duplication and loss or HGT (and the lack of such proofs for distance-based species tree estimation methods under the same conditions) [158]–[161].

Although wASTRID-s is highly accurate and very fast, we recommend using wASTRID-s in conjunction with wASTRAL-h and other species tree estimation methods. Due to its speed, the inclusion of wASTRID-s adds little computational burden, and having multiple approaches for estimating the species tree, each based on different techniques, can provide insights into which parts of the species tree are most reliably recovered and which parts may need further data for full resolution.

This study suggests several directions for future work. Most importantly, finding a way to incorporate branch lengths into the branch certainty scoring for wASTRID-s, i.e., a hybrid weighting, could improve accuracy and might close the accuracy gap between wASTRID-s and wASTRAL-h under some conditions. Another direction for future research is species tree estimation in the presence of gene duplication and loss, where gene family trees have multiple copies of species and are called MUL-trees [92]. The combination of DISCO [94],

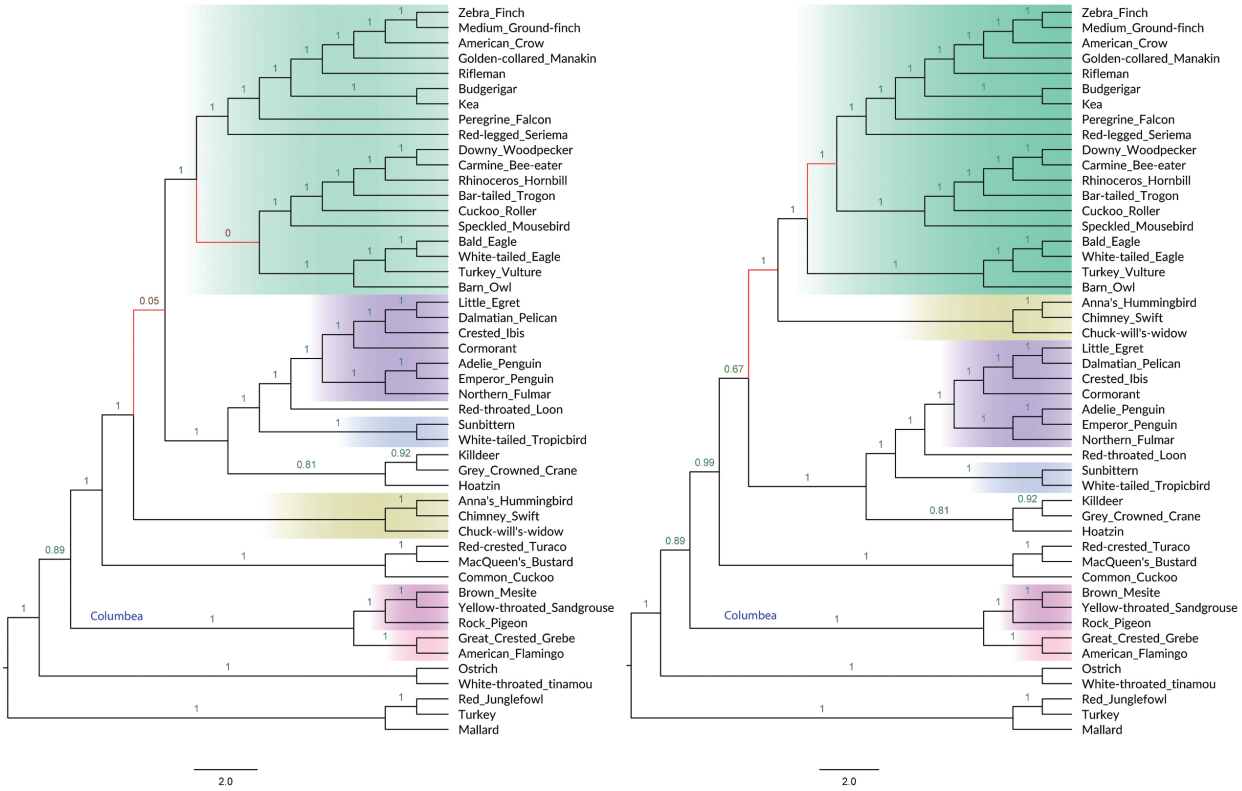
a method for decomposing MUL-trees into single-copy gene trees, with ASTRID produced very good accuracy and scalability [94], suggesting that combining DISCO with wASTRID might be even more accurate.

We also note that wASTRID might be useful for supertree estimation, a major challenge that currently has no fast and accurate methods that can scale to large datasets [50], [162]. Although ASTRID performed poorly as a supertree method for some model conditions [162], this may have been due to its reliance on PhyD* methods for handling missing data. Given the more accurate handling of missing data introduced in this updated version of ASTRID and our accuracy improvement using branch support, wASTRID might be a more accurate method for supertree estimation.

It may also be useful to develop a weighted version of FASTRAL [87], which speeds up ASTRAL. Recall that FASTRAL uses ASTRID as a subroutine to limit its search-space. Replacing ASTRID with wASTRID to produce a weighted version of FASTRAL might provide substantial improvements over FASTRAL.

Although this study examined relatively large datasets (up to 1001 species and 1000 genes), larger datasets may create additional challenges. For example, branch support in gene trees is known to be very low on large datasets, as discussed in [163], potentially making it difficult to reliably use weighted summary methods like wASTRAL and wASTRID. Very large datasets also present computational challenges, and improving speed for distance matrix calculation through parallelization is a natural direction. Future work should examine these questions on larger phylogenomic datasets.

Since the publication of wASTRID, subsequent research on a larger avian dataset, as highlighted in [6], further demonstrates wASTRID's effectiveness and scalability. This reaffirms its comparable performance to wASTRAL (if not better in that study), adding data to support the use of wASTRID in complex phylogenomic analyses.



(a) wASTRID-s avian tree

(b) wASTRAL-h avian tree

Figure 3.7: Results on the avian biological dataset ($n = 48, k = 14446$). In (a) and (b), we show the reconstructed species tree topology of wASTRID-s and wASTRAL-h, annotated with local posterior-probability branch support (localPP) computed using wASTRAL-h. The red branches show where the two trees differ. The six well corroborated clades according to Braun and Kimball [155] are displayed by both trees and highlighted in gradients. For wASTRID-s, the red branches also coincide with the only two low support branches. Contracting the very low support branches for wASTRID-s arrives at a topology compatible with wASTRAL-h. wASTRAL-h took around 294 seconds to infer its tree. wASTRID-s was very fast and took 1 second to infer its topology.

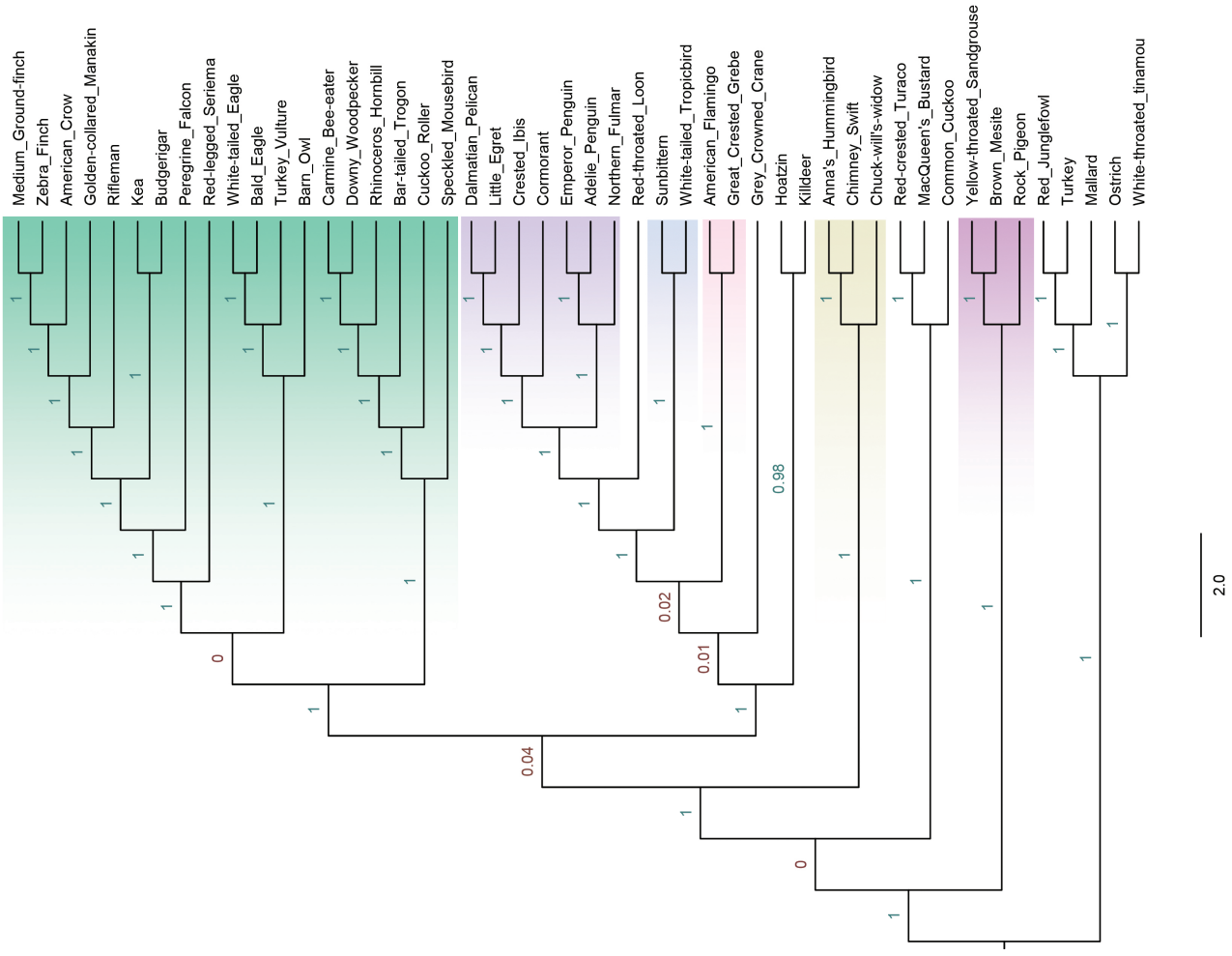


Figure 3.8: Reconstructed species tree on the Jarvis et al. avian biological data using ASTRID. Branch support values are in localPP values calculated by wASTRAL-h.

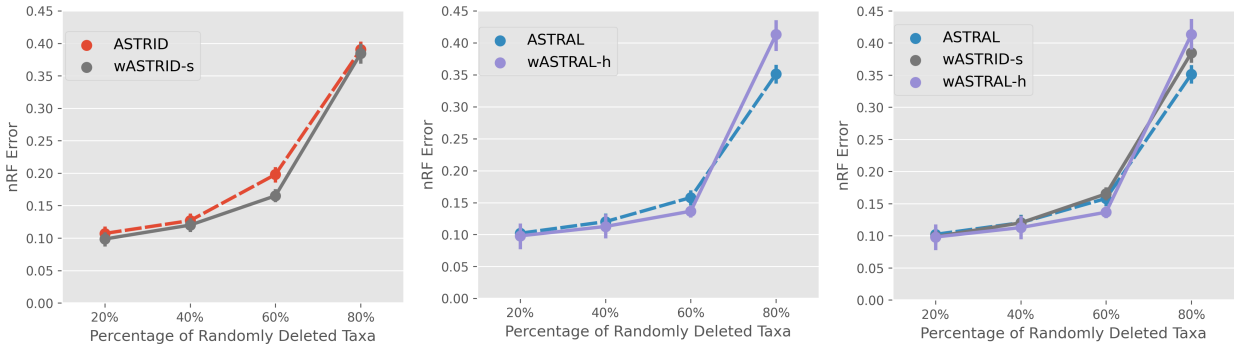


Figure 3.9: Topological error of species trees estimated for sets of incomplete gene trees due to deletion of a fixed number of taxa on the ASTRAL-III S100 dataset ($n = 101$, AD = 46%) given $k = 200$ genes with sequences of length 400 (GTEE = 42%). Results are shown averaged across 50 replicates with standard error bars. x-axis varies in the percentage of taxa deleted across gene trees, under the model where for each gene, a random subset of taxa of a fixed size is deleted from each gene tree.

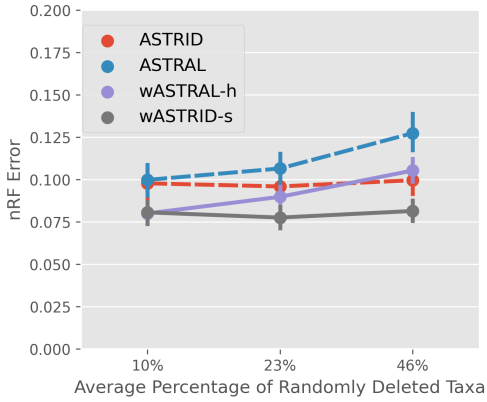


Figure 3.10: Topological error of species trees estimated for sets of incomplete gene trees due to clade-based missing data, on the ASTRAL-III S100 dataset ($n = 101$, AD = 46%) given $k = 200$ genes with sequences of length 400 (GTEE = 42%), varying the percentage of randomly deleted taxa. Results are averaged across 50 replicates with standard error bars. x-axis varies in the average percentage of deleted taxa in the gene trees.

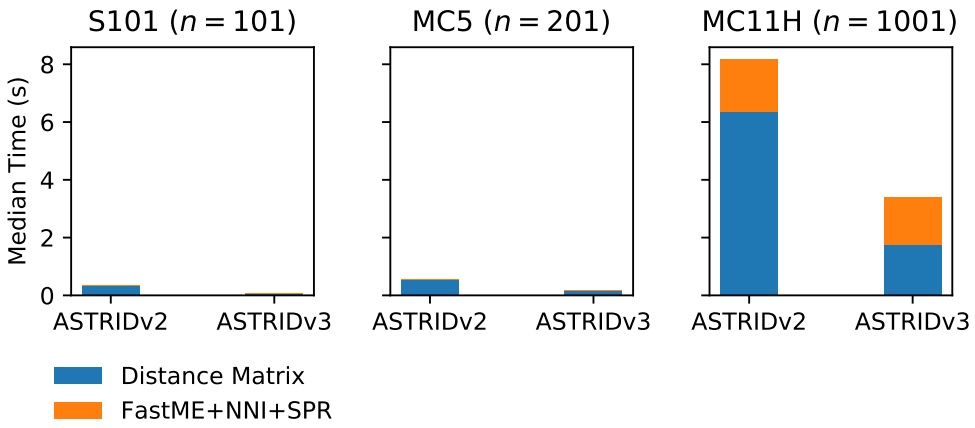


Figure 3.11: Wall-clock running times (in seconds) for ASTRID-2 and ASTRID-3, which differ only in the calculation of the distance matrix, on three different model conditions. Left: S101 with 101 species; middle: MC5 with 201 species; right: MC1H with 10001 species. Results shown are medians across 50 replicates. The portion in blue is for calculating the distance matrix, and the portion in orange is for running FastME.

Chapter 4: Fast & Scalable Species Tree Inference with Partial Constraints

The contents of this chapter are largely based on the following publication:

- B. Liu and T. Warnow, "Scalable Species Tree Inference with External Constraints," *Journal of Computational Biology*, vol. 29, no. 7, pp. 664–678, Jul. 2022. DOI: 10.1089/cmb.2021.0543

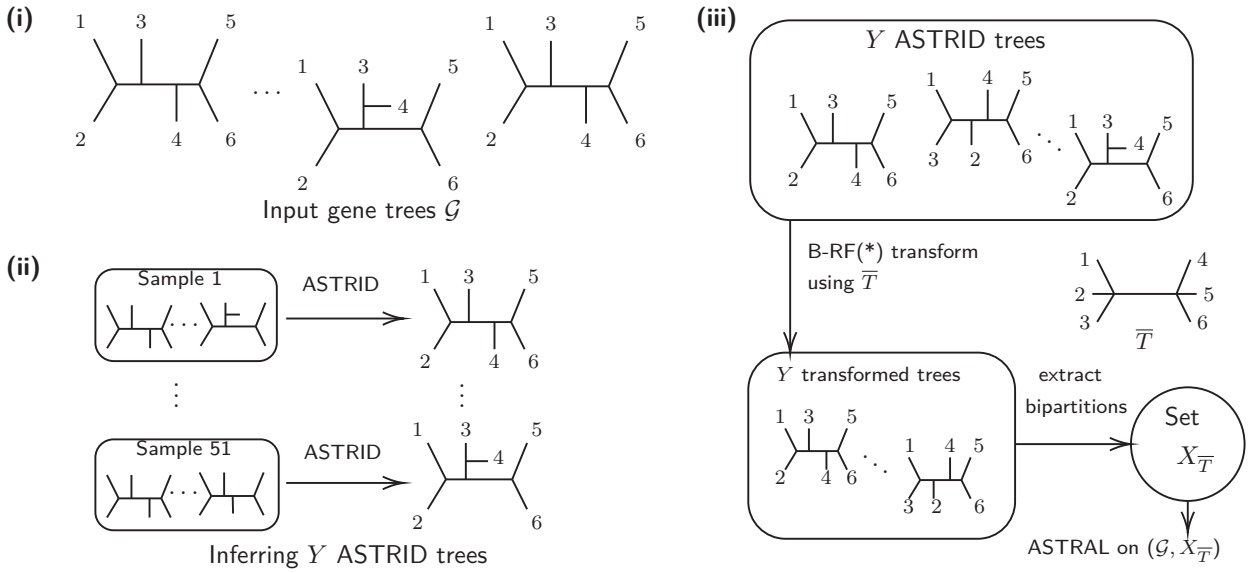


Figure 4.1: The FASTRAL-J pipeline. The input to the pipeline is a set $\mathcal{G}$ of unrooted gene trees, shown in (i), as well as a partially resolved constraint tree $\bar{T}$, shown in (iii). *(ii)* FASTRAL-J creates in total Y samples from $\mathcal{G}$, and infers ASTRID species trees from these subsamples. *(iii)* The Y ASTRID trees are transformed using the B-RF(*) algorithm (developed in ASTRAL-J) to force them to become refinements of the user-defined constraint tree (the backbone $\bar{T}$, which here is 123|456). The polytomies of this transformed set of ASTRID trees are then resolved, and the bipartitions of these trees induce the search space set $X_{\bar{T}}$. Finally ASTRAL is run on $(\mathcal{G}, X_{\bar{T}})$.

4.1 INTRODUCTION

Gene tree discordance, primarily driven by incomplete lineage sorting (ILS), poses a significant challenge in species tree estimation. Summary methods like ASTRAL are popular and useful for addressing ILS under the multispecies coalescent (MSC) model. However, their computational requirements can become substantial for large datasets or when gene tree heterogeneity is high. Additionally, well-studied phylogenetic groups often have established

relationships alongside unresolved controversies, presenting an opportunity to leverage prior knowledge in the form of incompletely resolved constraint trees.

This chapter explores the problem of species tree estimation with constraint trees, building upon existing methods like constrained ASTRAL ("ASTRAL-J"). We introduce NJst-J and FASTRAL-J, designed to efficiently incorporate constraint information while maintaining statistical consistency under the MSC model. These methods offer a means to refine incompletely resolved relationships while leveraging the computational advantages of their respective base methods. Through simulations and empirical analyses, we demonstrate the effectiveness of NJst-J and FASTRAL-J, particularly highlighting the speed and accuracy of FASTRAL-J in resolving species trees under the presence of partial constraints.

4.2 BACKGROUND

Basic definitions

All trees are assumed to be unrooted and on leafset S, with $|S| = n$. We also assume that gene trees evolve within a species tree under the multi-species coalescent (MSC) model. To recall some definitions, each unrooted tree T can be represented by its set $C(T)$ of bipartitions on its leafset produced by deleting edges in T. Tree A is said to be a **refinement** of tree B if $C(B) \subseteq C(A)$.

We will explore methods on estimated gene trees, and will report the average gene tree estimation error, denoting this quantity by GTEE.

Recall that a **summary method** takes as input a set $\mathcal{G}$ of unrooted gene trees on leafset S and returns an estimated species tree. We will say that summary method f is "tree-constrained" if the input includes a (potentially multifurcating) tree $\bar{T}$ on S, and we constrain the output tree T to satisfy $C(\bar{T}) \subseteq C(T)$.

We reuse the nomenclature of [165] and refer to $\bar{T}$ as the "backbone". We will say the backbone is "valid" if it has no false positive edges (i.e., every bipartition in the backbone appears in the true species tree, and so the backbone is a contraction of the true species tree).

Definition 4.1. A tree-constrained summary method f is said to be **statistically consistent under the MSC given a valid backbone** if the probability that f returns the true species tree converges to 1 as the number of gene trees increases.

ASTRAL-J

ASTRAL-J [165] (named after the command line flag of this variant) allows the user to specify a constraint tree $\bar{T}$ and then guarantees that the output tree T satisfy $C(T) \subseteq C(\bar{T})$. Given input set $\mathcal{G}$ of gene trees and constraint tree $\bar{T}$, ASTRAL-J associates to every $G \in \mathcal{G}$ a binary tree t_G that refines $\bar{T}$. To construct t_G given binary gene tree G and $\bar{T}$, ASTRAL-J adapted a linear-time tree algorithm [166] referred to as B-RF(+) (itself an improvement upon OCTAL [167] in running time) into the B-RF(*) algorithm. Concretely, the B-RF(*) algorithm returns a tree, referred to as $Comp(\bar{T}, G)$, that has the minimum RF distance to G among all refinements of $\bar{T}$. We informally call this step the "B-RF(*) transformation of trees in $\mathcal{G}$". As the B-RF(*) algorithm produces trees that may have polytomies, ASTRAL-J then employs heuristics from ASTRAL to produce binary refinements of these trees. The binary tree thus associated to G is referred to as t_G, and by construction is a binary tree that refines $\bar{T}$.

ASTRAL-J also computes additional binary trees that are refinements of $\bar{T}$; the goal of this step is to expand the search space (and hence potentially improve topological accuracy) while still enforcing the constraint tree $\bar{T}$. Finally, ASTRAL-J computes the set $X_{\bar{T}}$ to be the bipartitions appearing in any of the produced trees (i.e., the trees t_G or the additional trees computed by ASTRAL-J). ASTRAL-J then runs ASTRAL with input ($\mathcal{G}$, $X_{\bar{T}}$) (i.e., the same set of gene trees but a set of allowed bipartitions that enforces compatibility of the output tree with $\bar{T}$). By design, therefore, ASTRAL-J guarantees that the output tree will be a refinement of $\bar{T}$.

FASTRAL

FASTRAL addresses the computational challenges of ASTRAL, particularly when high levels of incomplete lineage sorting (ILS) or gene tree estimation errors (GTEE) lead to a substantial increase in the size of the bipartition set X. By generating a reduced set of allowed bipartitions, FASTRAL maintains statistical consistency while enhancing computational efficiency [87].

Operationally, FASTRAL proceeds as follows:

1. Computing Y subsamples of the gene tree set $\mathcal{G}$, with at least one subsample proportional to k.

2. Running ASTRID on each subsample to produce trees that contribute their bipartitions to the set X'.

3. Delegating to ASTRAL to solve the constrained MQSST problem with $\mathcal{G}$ and X', ultimately outputting the species tree.

This method reduces ASTRAL's runtime, allowing more efficient species tree inference in scenarios with significant gene tree heterogeneity.

As proven in [87], FASTRAL is statistically consistent under the MSC. The specific settings for the subsamples explored in [87] used $Y = 51$, with one subsample containing all the genes, 10 subsamples containing half the genes, 20 subsamples containing 25% of the genes, and 20 subsamples containing 10% of the genes. This approach provided very good accuracy and improved the runtime substantially compared to ASTRAL.

4.3 NEW METHODS: NJST-J AND FASTRAL-J

NJst-J

Recall that NJst has two steps: first it computes the internode distance matrix and then it runs neighbor joining (NJ). NJst-J modifies the second step so that the returned tree is a refinement of the backbone $\bar{T}$. Recall that NJ uses an agglomerative clustering algorithm that begins with a star tree and then progressively makes pairs of nodes into siblings until the tree is fully resolved. Our modification to NJst uses the backbone $\bar{T}$ to determine whether to accept a siblinghood proposal. Effectively, in NJst-J we reject a siblinghood proposal if accepting the proposal would produce a tree that is incompatible with $\bar{T}$. By construction, NJst-J will always return a tree, and the tree it returns will never violate the constraint tree.

FASTRAL-J

As FASTRAL invokes ASTRAL as a subroutine to solve the dynamic programming problem on $(\mathcal{G}, X')$, we can reuse the construction from ASTRAL-J, and transform X to $X_{\bar{T}}$ with respect to the backbone $\bar{T}$ analogously. FASTRAL-J directly builds on FASTRAL and takes an additional input, the backbone tree $\bar{T}$; see Figure 4.1. FASTRAL-J executes steps 1 to 3 the same way as FASTRAL for constructing set X'. Then, FASTRAL-J constructs $A_{\bar{T}} = \{Comp(\bar{T}, g) \,|\, g \in A\}$, and resolves polytomies in $A_{\bar{T}}$ using FASTRAL's heuristic for refining polytomies. Let $X_{\bar{T}}$ be the induced set of bipartitions of $A_{\bar{T}}$. FASTRAL-J invokes ASTRAL to solve the constrained-MQSST problem on $(\mathcal{G}, X_{\bar{T}})$.

4.4 THEORETICAL RESULTS

We now address theoretical properties of the methods we have developed. Specifically, we ask whether the methods are statistically consistent under the MSC model, which assumes that the method is given an increasingly large sample of the true gene trees.

Theorem 4.1. FASTRAL-J is statistically consistent under the MSC given valid backbones. Furthermore, if a polynomial number of ASTRID trees are computed, then FASTRAL-J runs in polynomial time.

Proof. Let T be the unrooted topology of the true species tree and let $\bar{T}$ denote a valid backbone; hence, T refines $\bar{T}$. The proof that FASTRAL is statistically consistent under the MSC given in [87] operates by establishing that at least one of the ASTRID trees computed on the different samples will produce T with probability converging to 1 as the number of genes increases; the rest follows since the optimal solution to MQSST will converge uniquely to T as the number of genes increases [51]. The argument we provide here is very similar.

It suffices to show that the true species tree topology T is feasible in the limit. By construction, at least one subsampled gene set increases in size as the number of genes increases, and these samples are drawn at random from the set of possible genes. Hence, at least one produced ASTRID tree will converge to the true species tree T as the number of genes increases. Since T refines $\bar{T}$, in the limit that ASTRID tree will refine $\bar{T}$ with probability converging to 1. Note that among all trees that refine $\bar{T}$, T has the minimum RF distance to itself. Therefore T's bipartitions will appear in the search space of ASTRAL $X_{\bar{T}}$ that is given by FASTRAL-J. As a result, T is a feasible solution to the MQSST problem that ASTRAL will run. Once the number of genes is large enough that the most frequently observed quartet tree on every four leaves is the most probable quartet tree for that set, the true species tree is the unique tree topology that optimizes the MQSST criterion. Hence, as the number of genes increases, FASTRAL-J will return the true species tree with probability converging to 1.

For the running time analysis, computing each ASTRID tree uses $O(kn^2)$ time to compute the internode distance matrix and then $O(n^3)$ to compute the FastME tree from the matrix, where k is the number of gene trees and n is the number of species. We only compute a polynomial number of ASTRID trees, so that the size of the constraint set given to ASTRAL is polynomial in n. Hence, the entire time is polynomial. QED.

Theorem 4.2. NJst-J is polynomial time and statistically consistent under the MSC given valid backbones.

Proof. We begin by reviewing the proof in [168] that NJst is statistically consistent under the MSC. The first part of this proof is that as the number of genes increases, the internode distance matrix D will converge to an additive matrix A for the species tree T, so that from a large enough number of genes the matrix D will satisfy $L_\infty(D, A) < f/2$ where f is the shortest length of any internal edge in the species tree T. The second part of the proof relies on the theorem in [80] that NJ will return the true species tree topology given such an estimated distance matrix D (i.e., one that is within $f/2$ of a matrix that is additive for the species tree).

Now consider NJst-J, and assume that $L_\infty(D, A) < f/2$; in this case, the agglomerative clustering step within NJ will, in each iteration, only pick true sibling pairs of clusters. Therefore given an estimated distance matrix D satisfying $L_\infty(D, A) < f/2$, the modified NJ in NJst-J will behave exactly the same as the original NJ algorithm and so return the true species tree; in other words, NJst-J is statistically consistent under the MSC given a constraint tree without any false edges.

For the running time, we note that NJst-J uses $O(kn^2)$ to compute the internode distance matrix and then runs NJ (which is $O(n^3)$ time). Hence, NJst-J runs in polynomial time. QED.

We end with a proof of consistency for ASTRAL-J.

Theorem 4.3. ASTRAL-J is statistically consistent under the MSC given valid backbones.

Proof. Let T denote the unrooted topology of the true species tree and $\bar{T}$ denote the valid backbone (thus, T refines $\bar{T}$). By Theorem 2 of [51], under the MSC model, in the limit, T is the unique solution to the (unconstrained) MQSST problem. Therefore, it suffices to prove that in the limit (as the number of genes increases), T is in the feasible set of ASTRAL-J for this valid backbone. We first note that T has non-zero probability under the MSC, and so as the number of genes increases T will appear in $\mathcal{G}$ with probability converging to 1. Recall that ASTRAL-J replaces each of its input gene trees t by another tree that is a refinement of the backbone $\bar{T}$ and that is closest to t with respect to RF distance. However, when t is a refinement of the backbone, then there is no replacement needed. Because $\bar{T}$ is a valid backbone, automatically T refines $\bar{T}$. Hence, if T appears among the gene tree input, then the set of allowed bipartitions that ASTRAL-J produces will contain all the bipartitions of T. Hence, T will be a feasible solution to ASTRAL-J given the input set of gene trees and backbone $\bar{T}$. QED.

We note that although the above argument is straightforward, this result is not presented in [165]. We also note that when estimated gene trees are used instead of true gene trees, then

summary methods, including ASTRID, NJst, and ASTRAL, are not statistically consistent, as shown in [63]. Statistical consistency when some gene trees are missing species is a complex issue; under some models of missing data ASTRID, NJst, and ASTRAL remain consistent, but under others ASTRID and NJst become inconsistent because the internode distance matrix can converge to a matrix that is additive for a tree other than the species tree [157].

4.5 EXPERIMENTAL STUDY

We evaluated our new methods, NJst-J and FASTRAL-J (using the same settings for FASTRAL as used in [87]) in comparison to ASTRAL-J (available at `https://github.com/maryamrabiee/Constrained-search`) on simulated datasets with 48-1001 species and 1000 genes, where ILS causes discordance between true gene trees and the species tree.

For these datasets we had the true species tree (known to us since we performed a simulation study) and so could precisely quantify species tree estimation error. We also explored results on the avian phylogenomics dataset from [149] with 48 species and 14,446 genes. In both cases, we used gene trees estimated using FastTree [64], a maximum likelihood software. We report the empirical statistics of these datasets in Table 4.1, showing number of taxa, number of genes, number of replicates per model condition, average discordance (AD) between true gene trees and true species tree, and GTEE. For the AD and GTEE statistics, we report RF error rates.

For the simulated datasets, we used the true species tree, and collapsed the branches that were short, as these are the branches that are statistically the hardest to reconstruct. We varied the percentage B of the branches that were retained from 25% to 75%, thus producing constraint trees T_B; hence, T_{25} contains the 25% longest branches and T_{75} contains the 75% longest branches (and is therefore more resolved than T_{25}).

For the biological dataset we used a constraint tree containing the "Magnificent Seven" (clades that were considered highly likely to be true, as these were the clades defining the strict consensus of the [169] tree and the [149] tree). On this dataset we also computed a tree using default ASTRAL (version 5.7.7).

Our evaluation criteria for the simulated datasets is the RF error rate for the estimated species tree. For the avian biological dataset since the true tree is not known, we evaluated results using MQSST scores and branch support (as calculated using ASTRAL's local posterior probability values [154]). We also report wall clock running time for all analyses.

Analyses of the simulated datasets were performed on the University of Illinois campus cluster, a heterogeneous computing cluster with a four-hour time limit. The analyses of the Jarvis et al. avian dataset with 48 species and 14,446 genes required a more substantial

Dataset	# taxa	# genes	# replicates	AD	GTEE
MC1	201	1000	50	69	49
MC6H	201	1000	50	9	38
MC11H	1001	1000	50	35	45
avian-sim 0.5x	48	1000	20	59	58
avian-sim 1x	48	1000	20	47	60
avian-sim 2x	48	1000	20	35	56
Jarvis et al. avian	48	14,446	1	N/A	N/A

Table 4.1: Empirical statistics for the datasets in this study (top datasets are simulated, bottom is biological). We report the number of taxa, number of gene trees, number of replicates, the degree of ILS as measured by AD (average discordance), and the amount of GTEE (gene tree estimation error). Dataset MC1 is from [71], the avian simulated datasets (avian-sim 0.5x, avian-sim 1x, avian-sim 2x) are from [69], and the biological avian dataset is from [149]. The MC6H and MC11H datasets are versions of MC6 and MC11 from [71] with shorter alignments to increase GTEE.

computational environment; these were was run on the Tallis cluster on the Illinois campus cluster, which allows four weeks of analysis. Each task had at least 64GB of memory available. We run all methods single-threaded and use the recommended default settings when possible.

4.6 RESULTS

Results on simulated datasets.

We show tree error rates across all model conditions in Figure 4.2. Error rates are lower when the constraint tree is better resolved (e.g., lower for T_{75} than for T_{25}) and also lower when the number of genes increases; both of these trends are expected. Error rates are also highest under the highest ILS condition (MC1, where AD=69%). For the avian simulated datasets, the error rates are higher with shorter species tree branches (i.e., avian 0.5X) than for the longer species tree branches (i.e., avian 2X), which is also the same as saying the error rates are higher under the higher ILS conditions. This trend is also expected.

Other trends, and in particular the relative accuracy of the different methods, are less predictable. For example, the gap between methods in some cases decreases as the number of genes increases, but not in all cases. Interestingly, NJst-J is best of all methods on the avian simulated datasets but least accurate on the other model conditions. The relative performance between ASTRAL-J and FASTRAL-J is also somewhat variable. There is a large differences in accuracy between the two methods on the avian simulated datasets that

favors FASTRAL-J, but little difference on the other model conditions.

The methods are, however, clearly differentiated in terms of running time (Figure 4.3). In this comparison, estimated gene trees are given as input and so do not count towards the running time; however, all other steps for each method are counted. In particular, FASTRAL-J includes the time to compute the ASTRID trees on the sub-samples.

Across all conditions, NJst-J is the fastest of the three methods, and its running time is not impacted by the number of genes; this is a trend easily explained by the two-step process it uses where it first computes the internode distance matrix and then computes a tree from that distance matrix (only the first step is impacted by the number of genes, but it is a very fast step). The running time advantage of NJst-J over the other two methods depends on the model condition (amount of ILS, GTEE, and number of genes) as well as on the resolution of the constraint tree, so that the gap can be small given a highly resolved constraint tree (i.e., T_{75}) and otherwise can be large. The largest gap on these simulated data was for the model condition MC11H, with 1001 species and 1000 genes; given the constraint tree T_{25} those datasets, ASTRAL-J used 9440 seconds (i.e., more than 2.6 hours), FASTRAL-J used about 1110 seconds (i.e., approximately 18.5 minutes), and NJst-J used about 139 seconds (i.e., just over 2 minutes).

The comparison for running time between FASTRAL-J and ASTRAL-J shows that in some cases they have very close running times, especially when the constraint tree is highly resolved. However, when there was a noteworthy difference between FASTRAL-J and ASTRAL-J, FASTRAL-J was faster. Furthermore, there are some cases where ASTRAL-J failed to complete analyses within the allowed four hour time limit on the Campus Cluster, but no other method failed in that time period. Thus, ASTRAL-J has a more substantial computational profile compared to the other methods.

Results on the Avian Phylogenomic dataset of [149]

[149] published an avian phylogeny using 14,446 genes on 48 taxa. Their published tree, called the "TENT" (total evidence nucleotide tree), was based on a maximum likelihood concatenation analysis; among its major conclusions is that Neoaves splits into two clades, Columbea and Passerea. Several findings of the study, including the split of Neoaves into these two clades, are contested in [169]. [170] established a set of seven clades they considered highly reliable, because they were found in both the Prum and Jarvis trees, and referred to these as the "Magnificent Seven". Subsequently, [171] reconsidered the Magnificent Seven and found that one of these clades (Clade IV, Otidimorphae) was not quite as clearly reliable as the others.

Method	Time (hh:mm:ss)	Norm. MQSST Score
ASTRAL-J	03:49:05	0.50019
FASTRAL-J	00:01:31	0.50028
NJst-J	00:00:06	0.49793
ASTRAL (unconstrained)	20:15:28	0.50038

Table 4.2: Running time (hours, minutes, and seconds) and normalized MQSST score of ASTRAL-J, FASTRAL-J, NJst-J, and default ASTRAL on the avian biological dataset of [149] with 14,446 genes and 48 species. The constraint tree contains the "Magnificent Seven" clades [170].

We used the Magnificent Seven to define a constraint tree containing these seven clades. We then analyzed the 14,446 gene trees from [149] using ASTRAL-J, FASTRAL-J, and NJst-J with this constraint tree; we also computed the ASTRAL tree (run in default mode) on this dataset (Figure 4.4). Since the true tree is not known (beyond what is in the constraint tree), we evaluate the results using MQSST scores and branch support values (using local posterior probability); further, we report the running time of these methods on this dataset (Table 4.2).

ASTRAL run without any constraint tree satisfies six of the seven clades in the constraint tree, but violates Clade IV (Otidimorphae); interestingly, this is the only clade from the Magnificent Seven that was identified recently in [171] as not being sufficiently reliable to be considered a valid constraint. This analysis thus could be considered potential confirmation of the reliability of the remaining six clades, which were returned by ASTRAL when it was run without any constraints. This ASTRAL tree also has the highest normalized MQSST score (0.50038) and good branch support: most branches are above 0.97, the lowest branch support in the tree is 0.56, and there are only two branches with support less than 0.75.

A comparison between the three tree-constrained analyses is also interesting. Their overall MQSST scores range from 0.49793 (for NJst-J) to 0.50028 (for FASTRAL-J), with ASTRAL-J in the middle at 0.50019. For the most part, all these trees are in topological agreement with each other (i.e., never differing in more than 7 out of 45 internal edges). However, although most branches are highly supported in all three trees, the trees do differ somewhat in terms of branch support. In particular, the NJst-J tree has three branches with support less than 70% (one with 56% support, one with 8%, and one with 3%). The ASTRAL-J tree has three branches with support less than 70%: one with 61%, one with 26%, and one with 3%, making it somewhat better supported than the NJst-J tree. The FASTRAL-J tree is somewhat better supported than the ASTRAL-J tree, with only three branches with support less than 70% (one with 67%, one with 56%, and one with 40%). Thus, of these three trees,

FASTRAL-J has the highest branch support and the highest MQSST scores, ASTRAL-J is in second place, and NJst-J is in third place.

An interesting result of this study is how none of the trees contain the Columbea or Passerea clades; the monophyly of these two clades are two of the key differences between the Jarvis et al. tree (which found these to be monophyletic) and the Prum et al. tree (which did not find them to be monophyletic). Whether these clades are "true" or not is debated, but it is also possible that the ability to recover these two clades may depend on the choice of data (i.e., coding versus non-coding data), as suggested by [172].

4.7 DISCUSSION

We have presented FASTRAL-J and NJst-J, two species tree methods that allow users to provide unrooted non-binary trees as constraints on the final tree. Previous to this study, ASTRAL-J was the only summary method that allowed such constraints to be provided. Our simulation study shows that NJst-J is very fast, much faster than the other methods we explored, and that FASTRAL-J is also faster than ASTRAL-J except when the number of genes is very small. We also saw that FASTRAL-J either matched or improved on the accuracy of ASTRAL-J, and that NJst-J varied in its relative accuracy compared to ASTRAL-J (sometimes more accurate, sometimes less accurate). Overall, FASTRAL-J is generally the most accurate or close to the most accurate and is much faster than ASTRAL-J, while NJst-J (although the fastest) is not as reliable.

Results on the avian phylogenomics dataset [149] are consistent with these trends: NJst-J is clearly the fastest (6 seconds), followed by FASTRAL-J (1.5 minutes), and then by ASTRAL-J (3 hours and 49 minutes). In addition, FASTRAL-J improved on both ASTRAL-J and NJst-J with respect to MQSST scores and branch support, showing its value on this biological dataset.

A major value in these tree-constrained methods is their greatly reduced speed compared to default ASTRAL. For example, default ASTRAL is dramatically slower on the avian phylogenomic dataset than all the other methods: it uses more than 20 hours and the slowest of the alternative methods uses less than 4 hours. Overall, while FASTRAL-J is particularly fast, all three tree-constrained species tree estimation methods provide benefits to biological systematics. In addition to speeding up species tree estimation, they enable biologists to explore the impact of assumptions about reliable clades; thus, multiple different hypotheses could be explored without the need for extensive computational resources.

Taking all these trends into consideration, when the dataset is sufficiently small that a full ASTRAL-J analysis is not prohibitive due to computational reasons, then the best protocol

may be to perform all three analyses–ASTRAL-J, FASTRAL-J, and NJst-J–and then select the tree that has the best MQSST score or use some other criterion to select among the trees. When the dataset size prohibits performing the ASTRAL-J analysis but allows FASTRAL-J and NJst-J, then the best protocol may be to use FASTRAL-J and NJst-J, and again select between the two trees based on a preferred criterion. When the dataset is so large that even FASTRAL-J cannot be run, then NJst-J may be used to estimate the species tree. Finally, if the user prefers to just use a single method and not perform more than one analysis, then FASTRAL-J may be the best method to use: faster than ASTRAL-J and generally as accurate and sometimes more accurate, and generally more reliable in terms of accuracy than NJst-J.

Although statistical consistency under the MSC has been established for ASTRAL, FASTRAL-J, and ASTRAL-J (as well as other methods), this assumption assumes that the input gene trees are topologically correct and that the only cause for deviation from the species tree topology is ILS. However, ASTRAL, NJst, ASTRID, and other summary methods can be used to estimate species trees from any input set of single-copy gene trees, even when the gene trees are estimated rather than true gene trees, and when they evolve under more complex models than the MSC. Furthermore, studies have shown that species trees estimated using summary methods can be highly accurate on estimated gene trees (even when there is substantial missing data) and competitive with concatenation or even more accurate than concatenation [60], [150]. Another previous study has shown that ASTRAL and Weighted Quartet MaxCut [95], both of which estimate species trees based on quartet-tree distributions, can have good accuracy on estimated gene trees even when HGT and ILS are both present [96]. Summary methods have also been extended to handle multi-copy gene family trees, with ASTRAL-Pro [93] (a new variant of ASTRAL) having particularly good accuracy and DISCO (a technique that transforms a set of multi-copy gene trees into single-copy gene trees that can then be used as input to a summary method) also enabling highly accurate reconstructions when followed by ASTRID [94].

This study suggests many future research directions. In terms of empirical performance, for example, future work should examine a broader set of model conditions to see how methods perform with much larger numbers of genes and higher or lower levels of GTEE and ILS. Since many phylogenomic datasets involve very large numbers of genes, these studies are needed to understand performance under realistic conditions.

In addition, this study did not examine concatenation analyses, such as the use of maximum likelihood software. Interestingly, although maximum likelihood concatenation analysis is provably inconsistent under the MSC, its performance on simulated datasets varies from poor to very accurate. In an early study, [173] showed that concatenation analyses can have

low accuracy when there is high gene tree discordance resulting from ILS, leading them to conjecture that maximum likelihood concatenation is inconsistent under the MSC (a result later proved by [62]). However, the relative performance of concatenation analyses and summary methods has also been examined and shown variable differences that depend on the model condition: given a large enough number of reasonably accurate gene trees, summary methods tend to be more accurate than concatenation (especially if there is high ILS), but when there is high GTEE then concatenation can be more accurate ([60]). Some maximum likelihood methods allow constraints on the output tree to be specified, and therefore evaluating concatenation analyses in this context is of definite interest.

Another interesting question we did not address in this study is the impact of missing data on methods. In terms of theory, even given "valid backbones" NJst-J and ASTRID will not be statistically consistent under an *i.i.d* model of missing data (see [157]), but will be consistent under some clade-based models of missing data [150]. Empirical performance, however, may still be good, as [150] showed that ASTRID exhibited a fair degree of robustness to missing data, making it comparable to ASTRAL under many conditions.

It will also be interesting to extend NJst-J, FASTRAL-J, and ASTRAL-J to the case where the input gene trees differ from the species tree due to gene duplication and loss (GDL), and study the theoretical and empirical performance of these extensions. Since ASTRAL-Pro and ASTRID-DISCO (i.e., the combination of ASTRID and DISCO [94]) have excellent empirical performance when given multi-copy gene family trees, the potential for improved empirical performance (or at least improved speed) using such extensions is certainly a promising direction.

Finally, the main advantage of FASTRAL-J over ASTRAL-J is running time, but NJst-J is even faster than FASTRAL-J. Given the improved speed of ASTRID compared to NJst, it is worthwhile considering whether it is feasible to develop a version of ASTRID that can incorporate user-defined constraints in a computationally efficient manner. However, ASTRID uses the BME (balanced minimum evolution) commands within FastME [174] and the BME technique uses local search strategies (e.g., NNI moves) to search through tree space for a (hopefully) optimal tree; modifying this approach in a computationally efficient manner to only consider trees that obey the input constraint seems non-trivial. Thus, at least using currently available methods, NJst-J might be the only method able to complete analyses quickly on very large phylogenomic datasets (e.g., with thousands of genes and using standard computational resources). Furthermore, given the very fast running time for NJst-J, it would be feasible to perform bootstrapping to assess branch support using NJst-J.

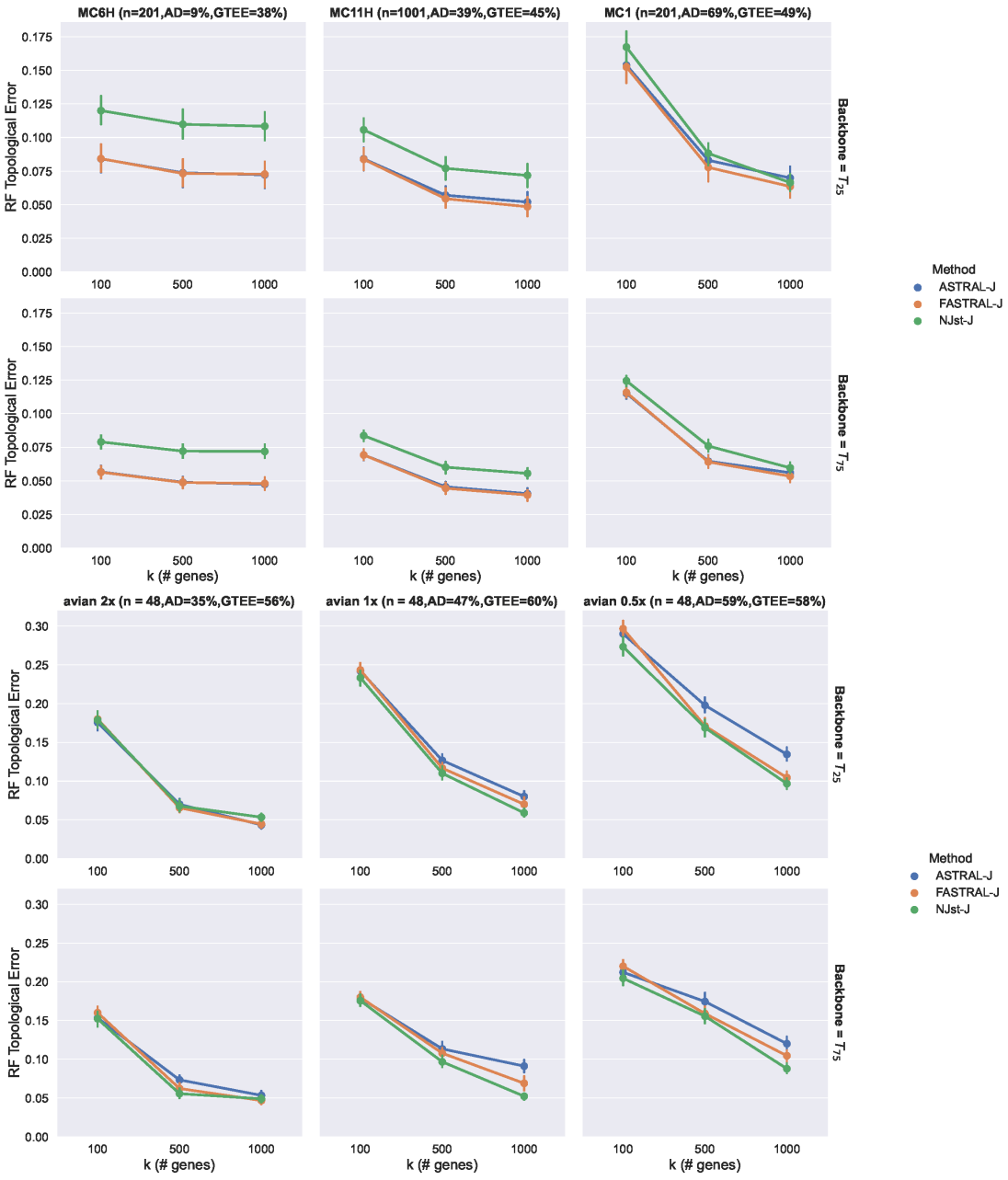


Figure 4.2: Tree error rates comparison across various model conditions and constraint trees (T_{25} and T_{75}), including simulated avian datasets. Error rates are averaged over replicates, with standard error indicated, and time-out instances noted for ASTRAL-J.

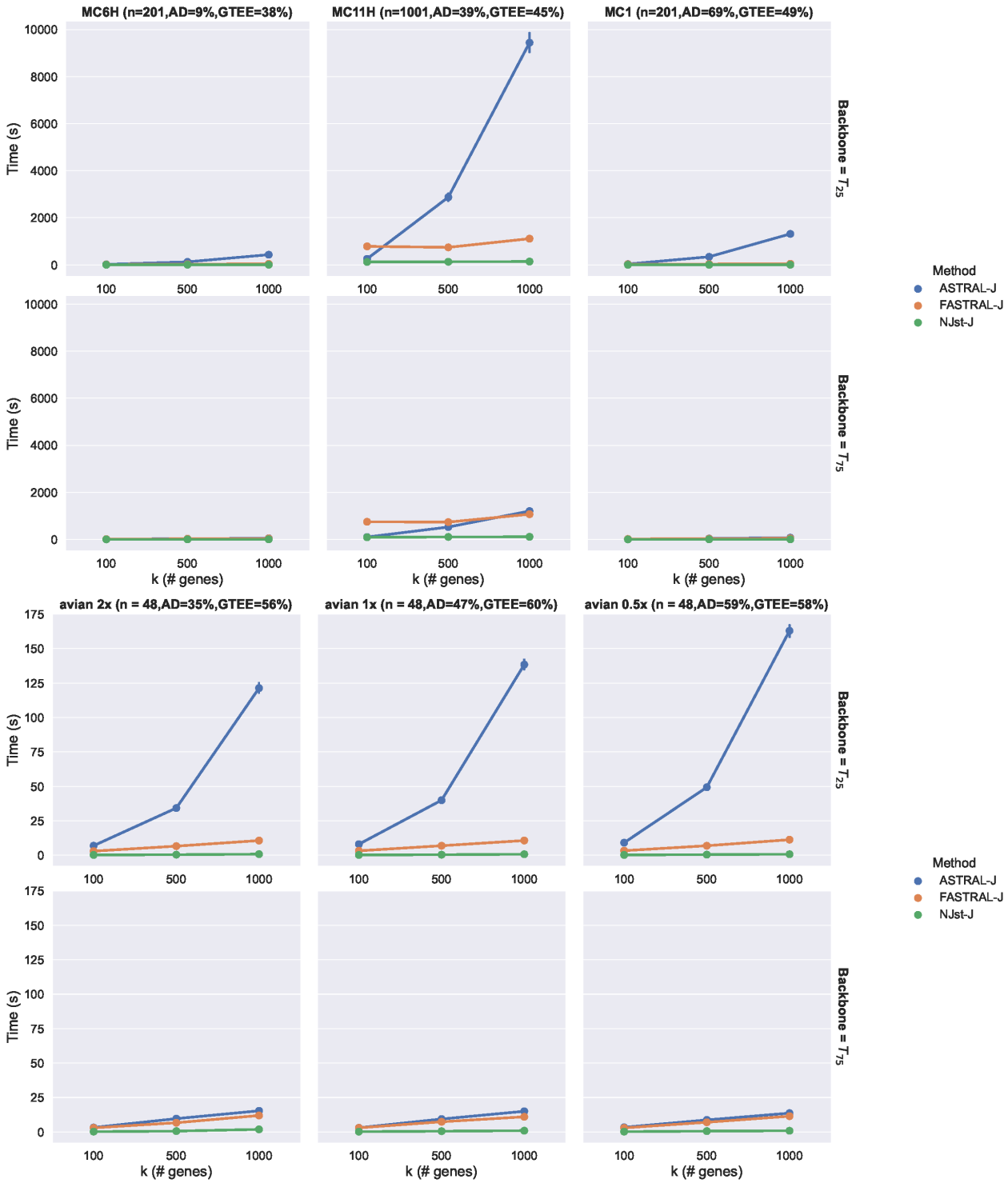


Figure 4.3: Wall-clock running times: top two rows are MC6H, MC11H, and MC1 datasets, and bottom two rows are avian simulated datasets. The running time shown is the mean across 50 replicates under MC6H and MC1. Under MC11H, when backbone is T_{25}, ASTRAL-J timed out on 16 replicates, and thus for MC11H the average on the completed 34 replicates is shown for all the methods. The error bars indicate standard error.

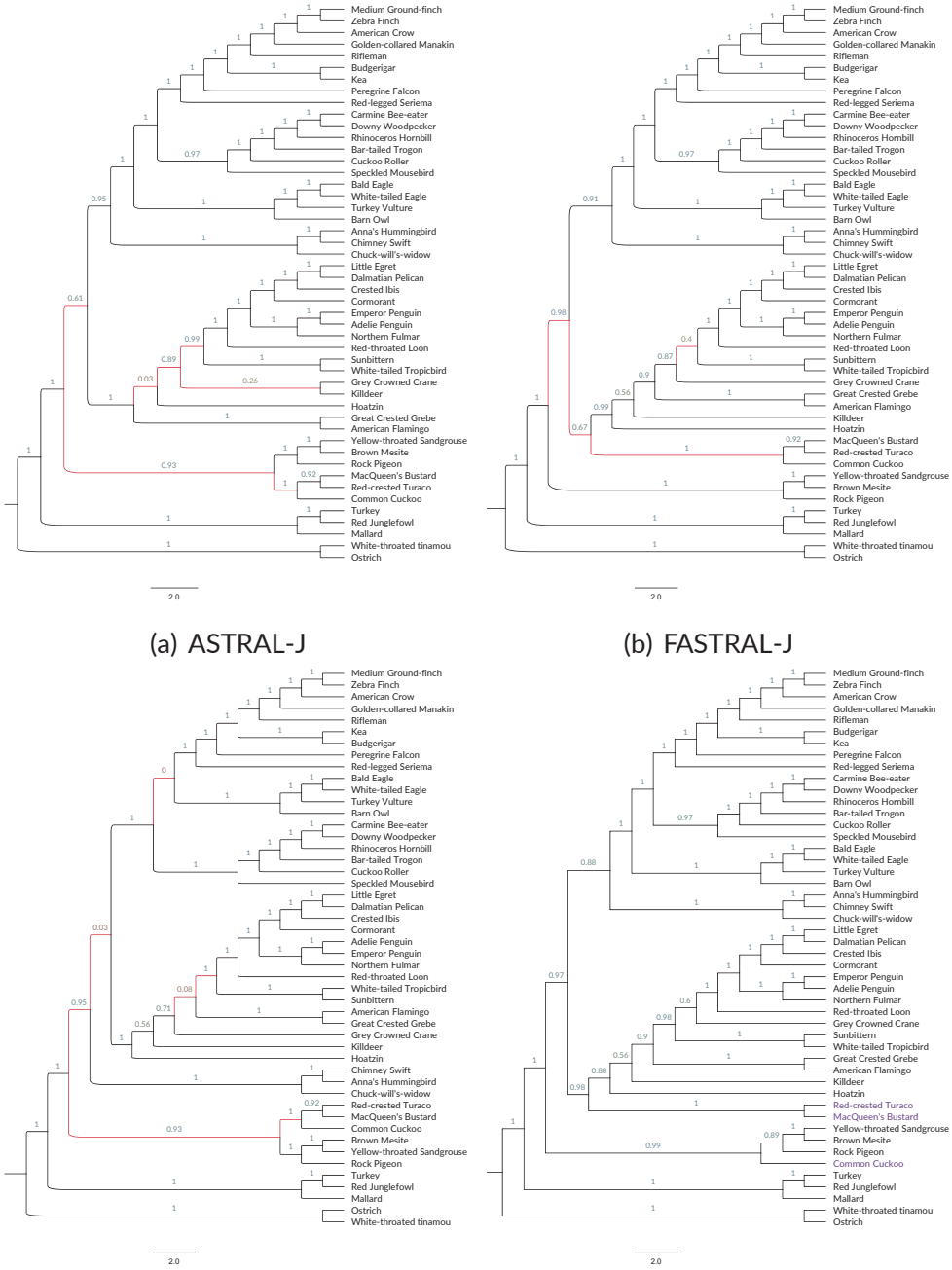


Figure 4.4: Comparative results on avian phylogenomic dataset using ASTRAL-J, FASTRAL-J, NJst-J, and default ASTRAL. Focus on "Magnificent Seven" clades, highlighting the one clade (Ortidimorphae) split by default ASTRAL, with localPP branch support and deviations marked in red.

Chapter 5: WITCH-NG: Efficient Aggregation of eHMM Predictions

The contents of this chapter are largely based on the following publication:

- B. Liu and T. Warnow, "WITCH-NG: Efficient and Accurate Alignment of Datasets with Sequence Length Heterogeneity," *Bioinformatics Advances*, vol. 3, no. 1, vbad024, Mar. 2023, ISSN: 2635-0041. DOI: 10.1093/bioadv/vbad024

5.1 INTRODUCTION

As the size and complexity of biological datasets increase, the role of Multiple Sequence Alignment (MSA) in computational biology has become more crucial. MSA underpins various applications such as phylogeny estimation, metagenomics, and protein structure prediction. However, aligning large and diverse datasets, especially those exhibiting high rates of evolution or sequence length heterogeneity, presents significant challenges. These challenges necessitate the development of robust MSA methods capable of maintaining accuracy across varied conditions.

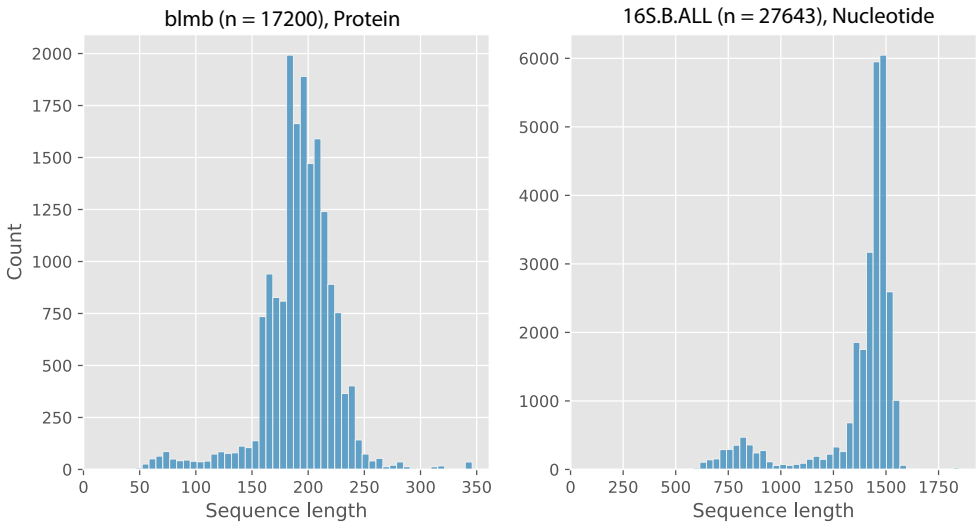


Figure 5.1: Sequence length histograms of two biological datasets showing sequence length heterogeneity: "blmb" and "16S.B.ALL". n denotes the number of sequences in each dataset.

Sequence length heterogeneity is a prevalent issue, arising from diverse sources such as short reads, partially assembled sequences, or evolutionary events like domain-level deletions (Figure 5.1). This variability can severely degrade the performance of traditional MSA methods, affecting the accuracy of downstream analyses.

To address datasets with sequence length heterogeneity, an effective strategy involves using a "backbone" alignment of "full-length" sequences as a reference, to which "query" sequences are subsequently added. This is, as introduced, the common "two-step" approach.

We recall several methods facilitate the integration of query sequences into a backbone alignment, including, HMMER (through `hmmalign`) the MAFFT (using the `--add` option), UPP, and WITCH.

WITCH, the latest among these, utilizes a two-stage technique involving an ensemble of Hidden Markov Models (eHMMs) to represent the backbone alignment. Unlike UPP, which aligns query sequences using a single HMM from the ensemble, WITCH employs a weighted consensus approach. It generates multiple potential alignments for each query sequence using the top ten HMMs and integrates these using the "Graph Clustering Merger" (GCM) method from MAGUS, enhancing the accuracy of the alignment process.

The WITCH-NG, an optimized version of WITCH, replaces GCM with a simpler and computationally less intensive approach using the Smith-Waterman algorithm. This adjustment preserves the alignment accuracy of WITCH but significantly improves runtime efficiency, making it nearly as fast as UPP. WITCH-NG's performance demonstrates the potential for algorithmic efficiency without sacrificing the quality of results in bioinformatics tasks.

5.2 BACKGROUND

5.2.1 WITCH

WITCH has five basic phases. In Phase 0, the input sequences are divided into a set of backbone sequences and the remaining sequences (called "query sequences"), with the backbone sequences sampled from those sequences deemed to be full-length. An alignment (default: MAGUS) and its maximum likelihood tree (default: FastTree2 [64]) are then constructed for the backbone sequences. In Phase 1, an ensemble of Hidden Markov Models (HMMs) is built on the backbone alignment. Phases 2 and 3 are then used to define an "extended alignment" for each query sequence, which is an alignment that contains the backbone alignment and the given query sequence. Phase 4 then combines the information in different extended alignments, one per query sequence, using transitivity, so that the output is an alignment of the entire set of sequences that induces the constructed extended alignments.

Since Phases 2 and 3 together are the most complex and together define how to add a given query sequence into the backbone alignment, we describe them in detail here. For a given query sequence, Phase 2 operates by computing a weighted score (larger is better)

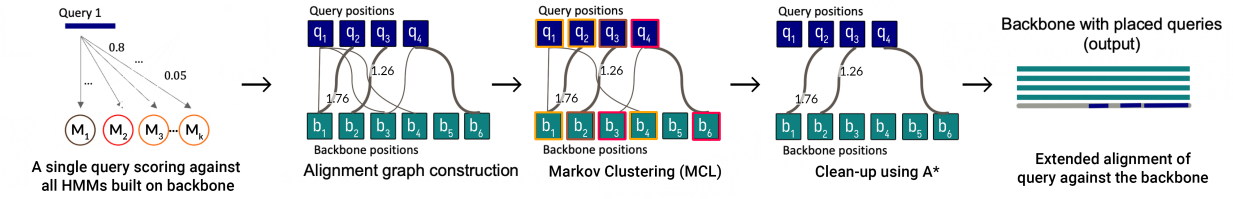


Figure 5.2: Overview of Phases 2 and 3 of the original WITCH algorithm (applied to a single query sequence). Given a single query sequence and the ensemble of HMMs, in Phase 2 a fitness score of that query is derived for each HMM in the ensemble. In Phase 3(a), for each of the top $k = 10$ HMMs, an alignment of the query to the backbone alignment (i.e., an "extended alignment") is constructed, and the pairwise homologies define a bipartite "alignment graph" with weights on the edges indicating the support for that homology (see text for more details). Phase 3(b): WITCH then runs Markov clustering (MCL) on the weighted alignment graph, producing a clustering of the nodes. If the modified graph may contain "violations" – i.e., sets of homologies (edges) that cannot coexist in an alignment – then a post-clustering Phase 3(c) occurs. In this post-clustering analysis, WITCH runs an A* heuristic search to modify the clustering to remove violations, so that it defines a "valid trace" (i.e., a valid alignment). The MCL and A* methods are the computationally intensive parts of Phase 3 in the WITCH pipeline. This figure is partially based on Figure 2 from [39] used under the CC BY 4.0 license.

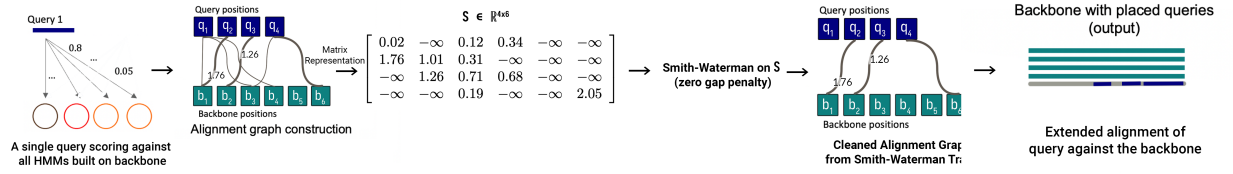


Figure 5.3: WITCH-NG replaces Phases 3(b) and 3(c) in the WITCH pipeline. The information in the alignment graph (Step 3(a) in the WITCH pipeline) is used to define the scoring matrix S, where $S[x, y]$ is the reward in matching position x in the query sequence to position y in the backbone alignment (and with $S[x, y] = -\infty$ if the two positions are not aligned by one of the extended alignments used to construct the alignment graph). This matrix is then used by Smith-Waterman, with zero cost gap penalty, to produce the alignment of the query sequence to the backbone alignment. The final output from WITCH-NG may be the same as for WITCH since they both effectively try to solve the same optimization problem (one explicitly and the other implicitly), but WITCH-NG solves the problem exactly in polynomial time while WITCH uses computationally intensive heuristics. This figure is partially based on Figure 2 from [39] used under the CC BY 4.0 license.

for each HMM in the ensemble, and then picking the top k such HMMs (default: $k = 10$). Phase 3 has several steps that we now describe, which jointly constitute a weighted version of the Graph Clustering Merger (GCM) from MAGUS. In Phase 3(a), it builds an extended alignment for the query sequence for each of these k selected HMMs, and uses these to construct an"alignment graph" where the nodes of the graph correspond to the positions in the query sequence or the backbone alignment and the edges are present between two nodes if and only if the corresponding positions are aligned in one of these extended alignments. These edges are weighted by the support defined by these extended alignments. In Phase 3(b), this weighted alignment graph is clustered using Markov Clustering (MCL), described initially in [175], which partitions the nodes into disjoint clusters. Since this set of clusters may not be a "valid trace" [23] (i.e., it may not define a valid pairwise extended alignment of the query sequence to the backbone alignment), Phase 3(c) then runs an A* algorithm to post-process the clustering to make it consistent with a valid alignment. Thus, the output of Phases 2 and 3 is an extended alignment for each query sequence (i.e., an alignment containing the query sequence and the backbone alignment).

5.3 WITCH-NG: REDESIGN OF WITCH

The key difference between WITCH and WITCH-NG is how each query sequence q is added to the backbone alignment. In WITCH, this is accomplished during Phases 2 and 3, where for each query sequence a weighted alignment graph is computed, then passed to MCL, then passed to an A* method (to remove violations) so that the final output is a valid extended alignment for each query sequence (see Figure 5.2 for how WITCH handles Phase 3). MCL can be computationally intensive and employs randomness to perform the graph clustering, and the A* algorithm is also often computationally intensive. Thus, while Phase 2 and Phase 3(a) are both fast and deterministic, Phases 3(b) (running MCL) and 3(c) (running the A* method) are not fast and due to reliance on randomness, are not even predictable in accuracy or runtime.

In WITCH-NG, the calculation of the extended alignment for each query sequence is handled differently. Phase 2 (scoring the query sequence against each HMM in the ensemble and computing the weight) and Phase 3a (keeping the top $k = 10$ HMMs and using them to construct the weighted alignment graph) are the same. However, WITCH-NG replaces Phases 3b and 3c (compare Figure 5.2 and Figure 5.3), as we now describe. Phase 3(b) and Phase 3(c), which run MCL and then the A* method to produce a valid trace, are replaced by a much simpler approach: the weighted alignment graph is used to define a scoring matrix S, where $S[x, y]$ denotes the reward of matching position x in the query sequence to position

y in the backbone sequence, and where $S[x, y] = -\infty$ if x and y are not aligned by any of the top k HMMs (to forbid aligning two positions that are not supported by any of the selected HMMs). The local alignment method, Smith-Waterman [25], is then run using S as a scoring matrix with zero cost gap penalty, to align the query sequence to the backbone alignment. As a result of this simplification, WITCH-NG achieves the same alignment accuracy as WITCH, but in a fraction of the time.

We now provide some intuition as to why the use of local alignment (computed using Smith-Waterman) instead of the Graph Clustering Method (GCM) produces alignments of very similar accuracy. [176] have argued that there is an implicit optimization problem in GCM that is identical to the solution to the local alignment problem using this cost function. Specifically, Zaharias *et al.* showed that GCM is a good heuristic for the optimization problem "Maximum Weight Trace for Alignment Merging (MWT-AM)" (a generalization of the classical Maximum Weight Trace (MWT) problem [23] in bioinformatics). In short, MWT-AM defines an optimization criterion to the scenario when merging multiple disjoint "constraint" alignments (i.e., the homologies in these alignments cannot be altered in the merging process) and when similarity scores between the constraint alignment columns have been obtained. Moreover, [176] also showed that maximizing the MWT-AM score for merging many alignments is beneficial for alignment accuracy.

Given that the original WITCH always uses GCM to merge two alignments, we consider the problem of optimizing for the MWT-AM criterion (restricted to the case of two alignments). We give the definition of MWT-AM on two alignments below:

Definition 5.1. (MWT-AM, trivial case) Given a weighted undirected bipartite graph with nodes $q_1, \ldots, q_m$ and $b_1, \ldots, b_n$, edges of form (q_i, b_j) with weight function $w((q_i, b_j)) > 0$, select a subset of edges T (the "trace") maximizing

$$\sum_{(q_i, b_j) \in T} w((q_i, b_j)) \tag{5.1}$$

subject to an additional "non-crossing" constraint, where for any two different (q_i, b_j) and (q_x, b_y) in T, either both $i < x$ and $j < y$, or both $x < i$ and $y < j$.

This trivial case is solvable in $O(mn)$ time and $O(mn)$ space by a simple DP algorithm (i.e., a simplified version of either Smith-Waterman or [24] without gap penalty), which coincides with our presented algorithm on S. This observation is far from new and is mentioned in the paper introducing the Maximum Weight Trace (MWT) problem [23].

Therefore, *WITCH-NG can be seen as replacing a computationally intensive heuristic (i.e., GCM) for the MWT-AM problem by an exact polynomial time algorithm for the same*

problem.

5.3.1 Implementation

We note two major differences in the implementation of WITCH-NG aiming for better running time compared to WITCH. Our first change is a strict improvement to the very last step of WITCH before the output, when the extended alignments (alignments only containing a query sequence and the backbone) are transitively merged. The original implementation in WITCH not only conceptually builds these extended alignments, but also writes these extended alignments to the disk verbatim before consuming them using a generic transitivity merging subroutine. This strategy is heavy in IO usage (writing extended alignments to disk then parsing them back into memory) and is extraneous. In WITCH-NG, we directly memorize each query letter's matched position in the backbone, which is sufficient to output the final alignment without incurring extra runtime cost in IO or parsing.

The second difference is that WITCH-NG implements a different strategy in invoking `hmmsearch` (the bottleneck step in obtaining the bitscores for UPP and in turn the weights for WITCH). WITCH-NG aggressively avoids invoking extra IO, and especially disk writes. Both WITCH and UPP use temporary files to divide up the query sequences into chunks for `hmmsearch` to consume and also to save outputs of `hmmsearch` across often hundreds of HMMs. Combined with the IO usage of `hmmsearch` reading the HMMs (saved on disk), these steps might hinder efficient parallelization. In other words, operations can become IO-bound instead of CPU-bound, in which case adding more cores will not speed up the computation of this bottleneck. To allow efficient parallelization under many cores, WITCH-NG instead relies on piping small chunks of sequences to the `hmmsearch` command while also parsing the `hmmsearch` output in-memory to minimize IO operations. This simple design allows more cores to be efficiently used until the IO bottleneck is hit.

5.4 EXPERIMENTAL DESIGN

We assembled a diverse set of publicly available datasets (both simulated and biological, locations provided in Appendix C Section 3), to evaluate the difference of WITCH-NG and WITCH, along with WITCH's predecessor UPP. The list of datasets and their respective statistics are shown in Table 5.1, where simulated conditions have been intentionally fragmented (1000M-HF series have half of the sequences fragmented to roughly 250bp in length [37]. RNASim-500bp has half of its sequences fragmented to an average length of 500bp

Dataset	# seqs.	Seq. length	Align length	p-dist.[4] (avg)
Simulated Nucleotide				
1000M1-HF [1]	1000	631.3	3960	0.694
1000M2-HF	1000	634.3	3972	0.683
1000M3-HF	1000	629.6	2723	0.660
1000M4-HF	1000	629.6	2571	0.495
RNASim-500bp [2]	1500-7000	1025.4	21946	0.408
Biological Nucleotide				
CRW	5507–27643	105.6–1557.2	414–11856	0.210–0.425
Biological Protein				
10AA	303-807	432.7	1745.3	0.671
Homfam[2]	14950-93681	149.8	273.5	0.690

[1] The 1000M series data have 20 replicates. 1000M1 has an outlier replicate as indicated in [177], hence removed.

[2] RNASim-500bp has five replicates (on 5000 full-length sequences and 5000 fragmented sequences) and we sampled different configurations of backbone-query size to benchmark the algorithm.

[3] HomFam datasets only have reference alignments on a small subset of the sequences. The last two of this row are derived from these small references.

[4] Average p-dist (p-distance) is the proportion of homologous pairs of letters that are different, where a "homologous pair of letters" is two letters in the same column in the alignment.

Table 5.1: Dataset statistics (average or range). See Tables S4–S7 in Appendix C for additional details.

[39], and when we later vary the backbone and query set size, we directly assign full-length sequences to the backbone and the fragments to the query.)

We performed two experiments. In Experiment 1, we compare WITCH, WITCH-NG, and UPP on both simulated and biological nucleotide datasets (from the Comparative RNA Website (CRW) [35]) where we have reference alignments on the entire set of sequences. In Experiment 2, we compare WITCH, WITCH-NG, and UPP on the 10AA dataset (a selected set of ten protein alignment datasets that have curated alignments) [39], [178], [179] and also on the ten largest HomFam datasets [34] (up to 93681 sequences), which only have reference alignments on very small subsets of the input sequences.

5.4.1 Evaluation Criteria

To evaluate estimated alignments, we use the error metrics computed by FastSP [180]. Briefly, each alignment encodes a set of homology pairs (defined by the columns of the alignment). SPFN (Sum-of-Pairs False Negative rate) is the proportion of true homology pairs (those from the reference alignment) that are not present in the estimated alignment. SPFP (Sum-of-Pairs False Positive rate) is the proportion of estimated homology pairs not found in the reference. Both rates are defined from 0 to 1. For convenience, we often report the average of SPFN and SPFP, sometimes referred as the alignment error or average error. We evaluate the rates across the entire alignment (not just restricted on the query sequences) both to show the final accuracy of the methods as *de novo* alignment methods aligning sequences from scratch and to also take into account the homologies between query sequences and the backbone sequences (which will be ignored if only restricted to the query sequences).

For speed, we record the wall-clock running time of the methods assuming the backbone tree and backbone alignment are given (i.e., ignoring the time used in Phase 0). Aside from the benefit of excluding a shared identical stage for all the methods, this wall-clock running time is exactly the running time for downstream analyses, such as phylogenetic placement or taxon identification, where the backbone alignment and tree are assumed to be already computed.

5.4.2 Computing Environment

All experiments were conducted on the Illinois Computing Cluster, a heterogeneous computing cluster where most runs are constrained to four hours with at least 64GB of available memory. For the most time-consuming dataset (16S.B.ALL), we allowed methods to run for more than four hours. We ran all methods across 16 cores.

5.4.3 Other MSA Methods

We compare WITCH-NG to WITCH and UPP, each run in default mode, but all three methods used the same backbone alignment and eHMM. See Appendix C Section 2 for commands and version numbers.

Methods other than UPP and WITCH were not selected for comparison as prior literature and preliminary results suggest that UPP is more accurate than other published methods (that we know of). For example, prior studies suggest that UPP is more accurate than `hmmbuild+hmmalign` [39] and MAFFT `--add` [181] (among those `--add` options that can

scale to large datasets).

5.5 RESULTS

5.5.1 Experiment 1: Simulated and Biological Nucleotide Datasets

Alignment Error

WITCH and WITCH-NG are essentially tied for average alignment accuracy (i.e., average of SPFN and SPFP) and both are more accurate than UPP (Table 5.2), with the advantage over UPP most noticeable under higher levels of evolution (1000M2-HF and 1000M1-HF). Results on RNASim are shown in Table C.1, as the methods have low SPFN (0.092 to 0.102) and SPFP (0.086 to 0.097) error, never differing in error for any model condition by more than 0.004, but when there are differences then UPP is less accurate. When separating into SPFN and SPFP, we see that WITCH and WITCH-NG have higher SPFP than UPP and lower SPFN rates, but the decrease in SPFN is larger than the increase in SPFP, so that the average alignment error is lower for WITCH and WITCH-NG than for UPP.

Running time

Figure 5.4 compares runtimes for WITCH, WITCH-NG, and UPP on datasets from Experiment 1. WITCH-NG is faster than WITCH across all settings, in some cases much faster (e.g., many of the HomFam datasets). While both methods finished very quickly on some datasets (1000M-HF, 5S.3, and 5S.T), even on those datasets, WITCH-NG was faster than WITCH (Appendix C Table C8). WITCH failed on one replicate of the RNASim datasets (with a backbone of 2000 sequences and 5000 query sequences) due to exceeding the four-hour tine limit, but UPP and WITCH-NG completed on all replicates.

The comparison between WITCH-NG and UPP depends on the dataset, with small differences on the RNASim datasets (often favoring WITCH-NG), bigger differences on the ROSE datasets (favoring WITCH-NG) and three largest CRW datasets (favoring UPP), and variable differences on the HomFam datasets (sometimes favoring UPP and sometimes favoring WITCH-NG). The two methods were essentially tied for speed on the 1000M-HF, 5S.3, and 5S.T datasets (Appendix C Table C8). .

WITCH-NG differs from WITCH in two ways: algorithmic design and algorithmic engineering (implementation). . To isolate the impact of the two types of changes, we created a version

of WITCH where we replaced WITCH's GCM subroutine by our described variant of Smith-Waterman. This version of WITCH thus only includes the changes in algorithmic design. We then compare the runtimes of WITCH (the original implementation), "WITCH(Smith-Waterman)" (the modified WITCH just described), and WITCH-NG on two very different datasets used in this study, with the results shown in Figure 5.5. The simplification in algorithm design is accountable for most (in the case of 1000M1-HF) and roughly half (for zf-CCHH) of the speed-up achieved by WITCH-NG. This shows that both the algorithmic design and engineering contribute to the speed-up.

WITCH-NG is faster than WITCH in all the conditions we explored, sometimes by a large amount (e.g., the simulated datasets and the 5S datasets from the CRW collection. Thus, WITCH-NG speeds up WITCH on all datasets, sometimes by a great margin. In addition, WITCH-NG is reasonably close to UPP in runtime, while this is not true for WITCH.

5.5.2 Experiment 2: Protein Datasets

Alignment accuracy

This experiment focused on two benchmark collections (10AA and Homfam) of protein datasets with reference alignments; here we discuss both runtime and average alignment error for these benchmark collections, with results on each dataset shown in the Appendix C Tables S2 and S3.

On the 10AA datasets (first row in Table 5.3), WITCH and WITCH-NG have identical average alignment SPFN and SPFP error. UPP has slightly higher SPFN (by 0.003) and slightly lower SPFP (by 0.004) compared to WITCH and WITCH-NG, so that UPP is 0.001 lower in average alignment error than WITCH and WITCH-NG. On the 10AA dataset, WITCH and WITCH-NG achieved essentially identical accuracy, differing more than 0.001 in SPFN or SPFP on only one dataset (Appendix C Table C3).

On the ten largest HomFam datasets (second row, Table 5.3), WITCH and WITCH-NG are identical in SPFN but with WITCH better (by 0.001) than WITCH-NG for SPFP; thus, the differences are very small. Compared to WITCH, UPP shows a large increase in SPFN (by 0.009) and a small improvement in SPFP (by 0.02), so that on average UPP is less accurate than both WITCH and WITCH-NG.

Running time

For running times on HomFam (Figure 5.4, middle subfigure), we see a dramatic reduction of the running time from WITCH to WITCH-NG in many cases. For example, on 7/10 HomFam datasets tested, WITCH-NG achieved at least a five-fold speedup. Furthermore, WITCH-NG uses nearly the same time as UPP for the HomFam datasets, in most cases only paying a small penalty in running time compared to UPP. On the 10AA datasets, all three methods finished under two minutes for Phase 1 – 4, with WITCH-NG always faster than WITCH, and faster than UPP on 8 of the 10 datasets (Table C3).

5.6 DISCUSSION

WITCH-NG is designed to be a fast version of WITCH. Specifically, WITCH-NG replaces the critical step in WITCH where a collection of extended alignments for the same query sequence is merged into a single extended alignment for that query sequence. This consensus alignment step is performed in WITCH through a computationally intensive heuristic, the Graph Clustering Merger from [177], which uses Markov Clustering (MCL) followed by an A* heuristic. In contrast, WITCH-NG obtains its consensus clustering using a local alignment step computed using Smith-Waterman. Since the GCM method and the local alignment method essentially aim to produce the same consensus (i.e., optimizing the Maximum Weight Trace for Alignment Merging), WITCH-NG is designed to produce an alignment similar to that of WITCH, but to do it faster.

Our study shows that WITCH-NG and WITCH produced nearly identical alignments, never differing in more than 0.001 SPFN or SPFP on any biological dataset (Table C9 in Appendix C). This very high similarity is not surprising, given that both WITCH and WITCH-NG optimize the same criterion (MWT-AM) when computing the consensus alignment.

We also saw a substantial speed-up in WITCH-NG over WITCH in many datasets. This is also to be expected, since GCM is heuristic and both its steps are computationally intensive while the Smith-Waterman method is deterministic and polynomial time. We also saw that while the improvement in runtime varied, the improvement in runtime could be very large. Finally, we note that the runtime improvement resulted from both the algorithmic innovation (using Smith-Waterman for the consensus alignment instead of using the Graph Clustering Merger) as well as from the improved algorithm engineering in WITCH-NG.

Dataset	Method	SPFN	SPFP	Avg. Error
1000M4-HF	WITCH	**0.014**	**0.010**	**0.012**
	WITCH-NG	**0.014**	**0.010**	**0.012**
	UPP	0.016	**0.010**	0.013
1000M3-HF	WITCH	**0.048**	0.039	**0.043**
	WITCH-NG	**0.048**	0.039	**0.043**
	UPP	0.054	**0.038**	0.046
1000M2-HF	WITCH	**0.128**	**0.106**	**0.117**
	WITCH-NG	**0.128**	**0.106**	**0.117**
	UPP	0.137	**0.106**	0.121
1000M1-HF	WITCH	0.172	0.143	**0.157**
	WITCH-NG	**0.171**	0.143	**0.157**
	UPP	0.182	**0.142**	0.162
5S.3	WITCH	**0.089**	0.086	**0.088**
	WITCH-NG	**0.089**	0.086	**0.088**
	UPP	0.093	**0.085**	0.089
5S.T	WITCH	**0.117**	0.104	**0.110**
	WITCH-NG	**0.117**	0.104	0.111
	UPP	0.120	**0.103**	0.112
16S.3	WITCH	**0.089**	**0.166**	**0.128**
	WITCH-NG	**0.089**	**0.166**	**0.128**
	UPP	0.090	**0.166**	**0.128**
16S.T	WITCH	**0.172**	0.177	**0.175**
	WITCH-NG	**0.172**	0.178	**0.175**
	UPP	0.189	**0.171**	0.180
16S.B.ALL	WITCH	**0.044**	**0.045**	**0.044**
	WITCH-NG	**0.044**	**0.045**	**0.044**
	UPP	0.045	**0.045**	0.045

Table 5.2: Alignment error rates on nucleotide datasets (four simulated and four biological datasets) placing into the same backbone alignment and tree. SPFN and SPFP are alignment error rates (lower is better, see text) and "Avg. Error" is the average of the two values, best values boldfaced.

Dataset	Method	SPFN	SPFP	Avg. Error
10AA	WITCH	**0.233**	0.189	0.211
	WITCH-NG	**0.233**	0.189	0.211
	UPP	0.236	**0.185**	**0.210**
Homfam	WITCH	**0.327**	0.124	**0.225**
	WITCH-NG	**0.327**	0.125	0.226
	UPP	0.336	**0.122**	0.229

Table 5.3: Alignment error rates on 10AA and Homfam (averaged across the ten datasets in each collection).

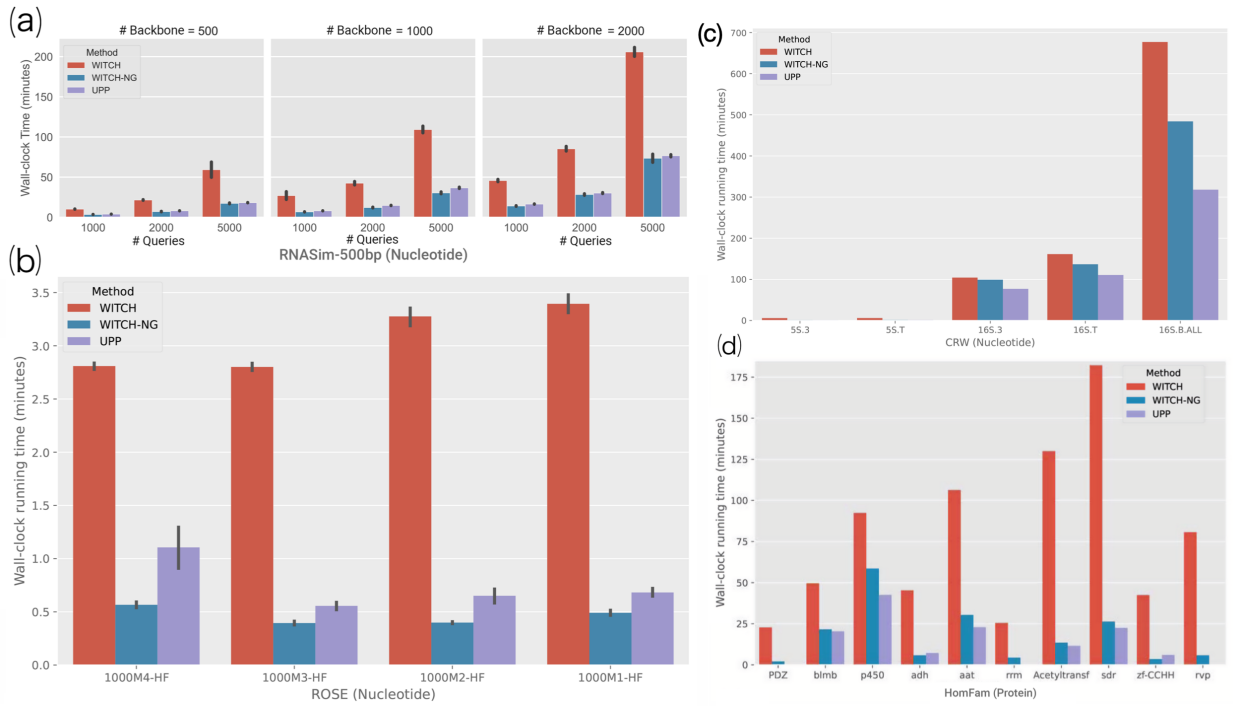


Figure 5.4: Wall-clock running time (minutes) for Experiments 1 and 2, time measured for Phases 1–4 (i.e., not counting Phase 0). (a) Runtimes on RNASim-500bp. WITCH fails on one replicate for the condition with 5000 query sequences and 2000 backbone sequences; results shown here are on only the four replicates where all methods complete. (b) Runtimes on ROSE (simulated). (c) Runtimes for the CRW datasets. Left-to-right, the number of sequences in each dataset is 5507, 5751, 6323, 7350, and 27643. (d) Runtimes on Homfam datasets. Left-to-right, the number of sequences in each dataset is 14950, 17200, 21013, 21331, 25100, 27610, 46285, 50157, 88345, and 93681. Notes: RNASim and ROSE are HF conditions, so the backbone alignments are on all the full-length sequences. CRW and HomFam analyses have 1000 backbone sequences and the rest are included in the query set. See text for discussion about 10AA datasets (where all methods took less than 2 minutes outside Phase 0).

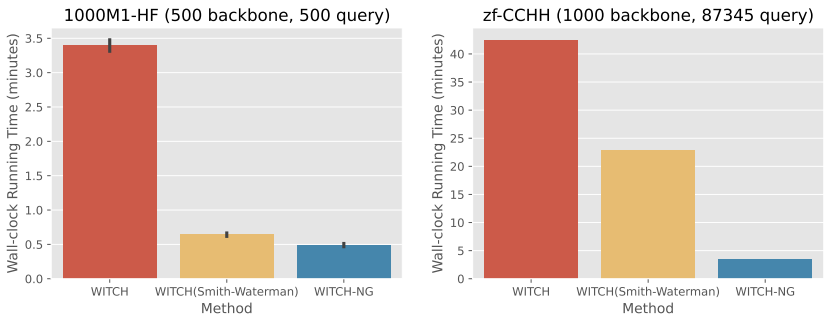


Figure 5.5: Exploring the effect of the algorithmic change (Smith-Waterman in lieu of default GCM) vs. the implementational changes, showing wall-clock running time (minutes, only measured on Phases 1–4) between methods on two very different data. "WITCH(Smith-Waterman)" denotes WITCH but with the GCM step replaced by the described Smith-Waterman step. On 1000M1-HF, the algorithmic change dominates the reduction in running time. On zf-CCHH, the implementation change is as important as the algorithmic change for the running time. On 1000M1-HF we show averages across replicates with standard error bars.

5.7 CONCLUSION

Multiple sequence alignment is a step in many bioinformatics pipelines, and yet datasets that are large, highly divergent, or contain fragmentary sequences are difficult to align with high accuracy. WITCH is a multiple sequence alignment method that aims to address the challenge of aligning datasets with many fragmentary sequences, and WITCH-NG is modification of the WITCH alignment method that aims to improve the running time without hurting accuracy. Our study shows that WITCH and WITCH-NG methods produce nearly identical alignments, but that WITCH-NG is generally faster, and sometimes greatly faster. The high similarity between WITCH and WITCH-NG alignments explains why WITCH-NG provides high accuracy in estimated alignments, and suggests that WITCH-NG should be useful for those applications where WITCH alignments have already been shown to be beneficial, such as phylogeny estimation [43].

This study suggests several directions for future work. For example, in WITCH-NG, we used a naive implementation of the naive Smith-Waterman algorithm to add a query sequence to the backbone alignment. The speed-up we obtain is therefore likely to be improved, given the many optimized variants of DP sequence alignment algorithms that are much faster on modern architectures (e.g., "striped Smith-Waterman" [182]).

It is also worth noting that WITCH-NG addresses only one of the aspects of the WITCH algorithm design that leads to a long runtime: how it combines extended alignments for each of the query sequence into a consensus extended alignment, so that the query sequence can be

added to the backbone alignment. However, WITCH has another computationally intensive step not addressed in WITCH-NG, where it compares each query sequence to each HMM in the ensemble of HMMs computed by WITCH for the backbone alignment. Modifying this design so that WITCH-NG only does a few of the query-HMM comparisons could potentially further speed-up WITCH-NG, and should be evaluated. One potential such modification could use the approach in UPP2 [41] to reduce the number of query-HMM comparisons used in UPP, and which enabled UPP2 to match the accuracy of UPP but be faster for large datasets.

Other future research could evaluate the impact of WITCH-NG for use in bioinformatics pipelines that use multiple sequence alignments (e.g., phylogeny estimation, as mentioned above). WITCH-NG can also be used directly to add sequences into alignments, a problem that arises in updating existing alignments and trees as new sequences are assembled and in microbiome analysis. WITCH-NG should be evaluated for use in these applications, especially for those analyses where runtime is extremely important (e.g., taxon identification in metagenomics.

Chapter 6: Overlapping Community Detection in Large-Scale Networks

This work is done with collaboration with Dr. David Bader's lab, under the advice of Dr. Bader, and with ideas from Dr. Zhihui Du and Oliver Alvarado Rodriguez.

6.1 INTRODUCTION

As introduced in chapter 2, many algorithms have been developed to tackle the challenge of overlapping community detection, where one seeks to cluster a graph into overlapping clusters, typically used for discovering hidden community structure in this graph. In this chapter, we focus on improving a density-based clustering algorithm for overlapping community detection, and our efforts to optimize it for high-performance computing (HPC) environments using the Chapel programming language under the Arachne [183] graph framework.

6.2 BACKGROUND

6.2.1 Edge-Based Clustering

Recall some notations. Let $G = (V, E)$ be a directed graph. In overlapping community detection, the identified clusters can be viewed as a set of node "covers," denoted as $C = \{c_1, c_2, \ldots, c_k\}$. Here, a node may belong to multiple clusters, each associated with a specific community.

Instead of directly clustering nodes, we focus on clustering edges to infer the community structure. In this framework, a node is considered to be labeled by multiple communities if the edges connected to it belong to multiple clusters.

A classical way of clustering edges is to cluster in the "line graph" space. Given a graph $G = (V, E)$, the line graph $L(G)$ is a graph whose nodes correspond to the edges of G, and two nodes in $L(G)$ are connected if the corresponding edges in G share a common node. In this context G can be thought of as a primal graph and $L(G)$ the corresponding dual graph (the "node-to-edge" dual). Line graphs give the benefit of allowing us to apply traditional clustering algorithms on graph nodes, such as spectral clustering or modularity maximization, to infer the community structure of the original graph. However, because the line graph can be quadratic in size compared to the original graph, it is impractical to use it for clustering large-scale networks, where a quadratic increase in size might be unaffordable.

Directly clustering in the edge space provides a computationally efficient strategy for

large-scale networks. Recall that a line graph node (an edge in the primal graph) is incident to another line graph node if the primal graph edges are incident to the same primal node. Thus, if we impose a (partial form of) metric on the edges of the (primal) graph that is only well-defined on adjacent edges, we have a way to cluster edges in a computationally efficient manner while retaining the spirit of clustering in the line graph space.

6.2.2 Distributed Computing with Chapel

To efficiently handle large-scale networks, we leverage distributed computing with the Chapel programming language. Chapel introduces the concept of locales, which represent distinct computing domains with their own memory and processing resources, typically corresponding to nodes in a distributed system. In Chapel, a locale is a first-class concept. Data lives in some locale, but programs can access data from other locales. However, in contrast to more explicit forms of distributed computing, Chapel abstracts away the details of data distribution and communication.

An example of this abstraction is Chapel's *distributed* arrays. Traditionally an array is a contiguous block of memory, but in Chapel, a *distributed* array is a collection of arrays, each potentially residing in a different locale. Accessing a region of the distributed array from a locale that does not own it triggers a remote memory access. The benefit here is the abstraction – accessing $A[i]$ in Chapel is the same semantics and syntax whether A is local or remote. However, Chapel cannot help us violate the laws of physics – remote memory access is slower than local memory access.

However, a benefit from the above abstraction is the seamless transition to shared memory architecture. For example, many structures can be backed by arrays (e.g., graphs, heaps, phylogenetic trees), and thus making these arrays as distributed arrays can allow for easy transition to distributed computing, but is likely less efficient than purposefully designed distributed algorithms or data structures.

6.2.3 Hierarchical Clustering and HDBSCAN

Hierarchical clustering algorithms build a hierarchy of clusters by successively merging or splitting groups of data points. We now formally introduce the key data structure of **dendrogram**, useful behind hierarchical clustering algorithms like HDBSCAN, as a forest of rooted trees that represents the hierarchical relationships between clusters. The dendrogram supports operations like Init (initializing the forest with singleton trees) and Join (merging two trees into a single tree). It is worth mentioning under the context of this

dissertation that phylogenetic trees are often thought of as dendrograms, and sometimes used interchangeably. For example, UPGMA [184] is often introduced under both clustering contexts and phylogenetic estimation contexts. However, here we explicitly use dendrograms under the hierarchical clustering sense. Each dendrogram defines a set of clusters, e.g., the dendrogram $(A, (B, C))$ defines $\{A, B, C, (B, C), (A, (B, C))\}$. We define two dendrograms as (topologically) **equivalent** if the set of clusters they represent is the same.

We define a **join sequence** for a dendrogram of size m as a list of tuples (u, v, t) sorted by t, where $t \in \mathbb{Z}$ represents a dissimilarity measure between elements u and v, and u, v represent "points" in the dendrogram, or in our case edges in the input graph G. The operation of realization on this join sequence can be defined as the process of initializing a dendrogram of size m, iterating over the sequence, and performing JOIN operations for each element, ultimately resulting in a single tree that represents the hierarchical clustering of the data. We define two join sequences as **equivalent** if they realize to dendrograms that are equivalent.

6.2.4 Graph Formalization

We consider simple directed edge-weighted graphs $G = (V, E, w)$, where V represents the set of nodes, E represents the set of edges, and w assigns a weight to each edge.

We equip the directed graph G with an edge dissimilarity function $d : E \times E \to [0, \infty)$. This function $d(e_1, e_2)$ is defined only for pairs of edges e_1 and e_2 that are path-connected, meaning they form a directed path in G. For example, given the edges $a \to b, b \to c, a \to d$, the edges $a \to b$ and $b \to c$ are path-connected because they form the directed path $a \to b \to c$, and thus $d((a, b), (b, c))$ is defined. However, edges $a \to b$ and $a \to d$ are not path-connected, as there is no directed path in G that includes both edges; hence, $d((a, b), (a, d))$ is undefined. This type of dissimilarity function is particularly relevant in applications like communication networks, where we can model a conversation as a relay of messages. In this context, path-connected edges like $a \to b$ and $b \to c$ could represent a sequence of messages within the same conversation.

Given the above definitions, we now define the join sequence induced by this type of metric d a specific on G. The join sequence can be defined as the following sequence sorted by increasing t:

$$\{(u = e_1, v = e_2, t = d(e_1, e_2)) \mid e_1, e_2 \in E \text{ and } d(e_1, e_2) \text{ defined}\} \tag{6.1}$$

where u, v represent some edges in E and t is the dissimilarity between them. In temporal graphs, t usually represents the time difference between two edges.

Consequently, the dendrogram induced by d on G equipped with the mentioned structures is the above join sequence realized on a dendrogram of size $m = |E|$.

We note that this requirement of d on two consecutive edges is not a make-or-break assumption. Given arbitrary metric defined on adjacent edges, e.g., obtained through representation learning on edges, can also be used. It is relatively easy (e.g., random-walk based methods such as node2vec [185] on edges, or edge2vec [186], or create node embeddings and for each edge (u, v) take the average of node embeddings as edge embedding) to obtain edge embeddings on arbitrary graphs, and then the embeddings induce a metric on edges that share an incident node. Our stronger assumption of d only adds one additional assumption to our algorithm, allowing a per node $O(\deg(u) \log \deg(u))$ time complexity for join sequence construction per node in G as opposed to a $O(\deg(u)^2)$ time complexity per node.

Our work builds upon the foundational concepts and algorithms presented by Ostroski [187], with the goal of optimizing HDBSCAN for HPC environments using Chapel. We focus on parallelizing the join sequence construction and cluster generation phases of HDBSCAN, which are potential bottlenecks in the algorithm's execution. On a high level, the algorithm proceeds as follows:

- **HDBSCAN Step 1:** For a given graph $G = (V, E, w)$ equipped with an adjacent edge dissimilarity function $d : E \times E \to [0, \infty)$, we first construct the join sequence. This sequence is induced by d on G.
- **HDBSCAN Step 2:** With the join sequence established, we initialize a dendrogram of size $m = |E|$. We then iteratively apply the join operations as defined in the join sequence to form a hierarchical clustering of the edges.
- **HDBSCAN Step 3:** HDBSCAN's cluster generation algorithm is run on the dendrogram to identify clusters on the edges.
- **Convert to Overlapping Community Structure:** Finally, given node adjacent to edges labeled with clusters $c_1, c_2, \ldots, c_k$, this node is labeled with all clusters $c_1, c_2, \ldots, c_k$.

6.2.5 Parallelizable Components – Our Contributions

To optimize HDBSCAN for HPC environments using Chapel, we identify areas of HDBSCAN for improvement. Two critical components are the "join-sequence" and "cluster generation" steps. We focus on these in our contributions.

Contribution A: Parallel Join Sequence Construction in Chapel

While it's well-known in parallel computing literature that minimum spanning trees can be built in parallel, existing algorithms don't fully exploit our metric space's specific structure, which is partial and extremely local (defined not on all pairs of edges). Ostroski suggested constructing the join sequence in $O(m \log m)$ time using a parallel algorithm. We translated the algorithm and its structures to the HPC regime using Chapel. Our contribution here is practical: we implemented the bulk of HDBSCAN in Chapel, demonstrating that the join sequence can be parallelized more efficiently. This practical benefit makes Chapel advantageous for community detection under an HPC setting. Our parallel join sequence construction in Chapel significantly outperforms the original Python implementation.

Contribution B: Cluster Generation

After the join sequence is constructed, the next phase of HDBSCAN is "cluster generation," where the join sequence is translated into a dendrogram, and then the cluster generation ("cluster cutting") algorithm is run. Prior work has addressed shared-memory architectures for HDBSCAN. This segmentation isolates denser clusters, identifying "stable" clusters. The challenge in parallelizing cluster generation lies in its inherently sequential nature, requiring sequential edge joining to construct the dendrogram, and thus under concurrent or parallel settings, only one join can happen at a time. Given the challenges, we take a different approach, approximating cluster generation in a multi-locale environment. We propose a multi-locale algorithm for HDBSCAN's dendrogram construction phase and evaluate its accuracy. While less accurate than desired, it shows potential in certain concurrent settings.

6.3 MATERIALS AND METHODS

6.3.1 Notations

In general, given a graph G, we let n be the number of nodes, m be the number of edges. The size of a graph G is, for simplicity, defined as m. Let k be the number of locales in Chapel lingo, and in many cases this is equivalent to the number of physical machines present in the distributed system.

6.3.2 Notion of Accuracy

Our task is to optimize a HDBSCAN-based edge clustering algorithm that can be used also for community detection. This generates two senses of accuracy:

- Reimplementation accuracy: where we compare our clusters against the reference implementation's clusters, ignoring the actual community structure of the graph. This is a measure of how well we have implemented the algorithm.
- Community detection accuracy: where we compare our clusters against the actual community structure of the graph. This is a measure of how well the algorithm performs in practice.

In this chapter, we primarily use the first notion of accuracy. A good reimplementation of a good community detection algorithm will give good community detection accuracy, so assuming that the reference implementation is itself accurate, we can focus on reimplementation accuracy.

6.3.3 Parallel Join Sequence Construction in Chapel

Recall that given a directed edge-weighted graph $G = (V, E, w)$ with edge weights $w : E \to \mathbb{R}$ and an adjacent edge dissimilarity function $d : E \times E \to [0, \infty)$, our goal is to construct the join sequence efficiently in parallel. Recall that by adjacent, we mean that the two edges share a common endpoint and we do not care about directionality.

For a node $u \in V$, let $E(u)$ denote the "edge neighborhood" of u, containing all edges with one endpoint at u. We define the **local join sequence** $J(u)$ as the join sequence induced by the metric d on the edges in $E(u)$. The **global join sequence** $J(G)$ is the join sequence induced by d on all edges in the entire graph G.

Observation 6.1. If the metric d is the L1 metric between scalars, then the local join sequence $J(u)$ for any node u can be constructed in $O(\deg(u)) \log \deg(u)$ time, where $\deg(u)$ is the degree of u. This allows us to construct local join sequences independently and in parallel.

Proof. The L1 metric between two scalars a and b is simply the absolute difference between them, i.e., $|a - b|$. To construct the local join sequence $J(u)$, we need to compute the pairwise L1 distances between all adjacent edges in $E(u)$. Since there are $\deg(u)$ such edges, there are $\binom{\deg(u)}{2}$ pairs of adjacent edges. However, we can exploit the structure of the L1 metric to avoid computing all pairwise distances explicitly. Instead, we can sort the edges in $E(u)$ by

their weights in $O(\deg(u) \log \deg(u))$ time. Then, for each edge e_i, we only need to compute the L1 distance to its immediate successor e_{i+1} in the sorted order. This is because the L1 distance between any two edges e_i and e_j with $i < j$ can be computed as the sum of L1 distances between consecutive edges in the sorted order:

$$d(e_i, e_j) = \sum_{k=i}^{j-1} d(e_k, e_{k+1}). \tag{6.2}$$

Therefore, after sorting the edges, we can compute all pairwise L1 distances in $O(\deg(u))$ time by iterating through the sorted list and computing the L1 distance between consecutive edges. This allows us to construct the local join sequence $J(u)$ in $O(\deg(u) \log \deg(u)) + O(\deg(u)) = O(\deg(u) \log \deg(u))$ time. Furthermore, since the local join sequence construction for each node only depends on its edge neighborhood, we can perform this process independently and in parallel for all nodes in the graph. QED.

We additionally note that our above running time is pessimistic. Due to our assumption that the edge weights are in $\mathbb{Z}$, faster integer sorting algorithms exist (e.g., radix sort in practice can be very fast for integers [188]), and theoretically we have $o(n \log n)$ sorting algorithms [189] for integers, but our running time $O(n \log n)$ for integer sorting is a safe and practical assumption.

Observation 6.2. The global join sequence $J(G)$ is equivalent to the sorted sequence of the union of all local join sequences:

$$J(G) \equiv \text{Sort}\left(\bigcup_{u \in V} J(u)\right). \tag{6.3}$$

Proof. By definition, the global join sequence $J(G)$ contains all tuples $(e_1, e_2, d(e_1, e_2))$ for all edges e_1 and e_2 that share an endpoint in the graph G, sorted by increasing $d(e_1, e_2)$. Since each pair of adjacent edges is associated with a specific node (the node where the two edges meet), each tuple in $J(G)$ must also be present in the local join sequence of that corresponding node. Therefore, the union of all local join sequences contains all the tuples in $J(G)$. Furthermore, sorting the union of all local join sequences will result in a sequence that is sorted by increasing $d(e_1, e_2)$ for all edges e_1 and e_2 in the graph G, definitionally equivalent to the global join sequence $J(G)$. Therefore, the global join sequence can be constructed by taking the sorted union of all local join sequences. QED.

While these observations are not new and were presented in the original work by Ostroski [187], our contribution lies in implementing them efficiently using the Chapel programming

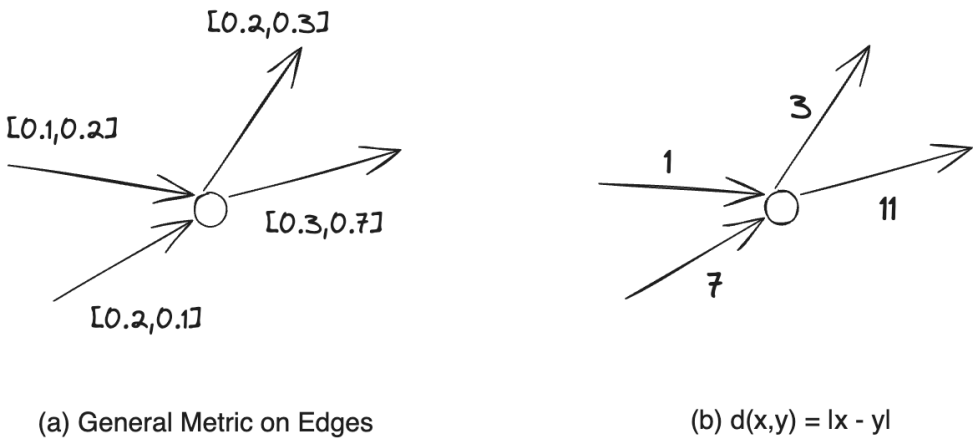


Figure 6.1: Parallelizing the join sequence construction in Chapel. (a) given general metric (represented as arbitrary vectors under an $\mathbb{R}^n$ space), it is hard to determine the local join sequence on a node in less than $O(deg(u)^2)$ time, where $deg(u)$ is the degree of node u. (b) We can exploit the structure of the metric to determine the local join sequence in $O(deg(u) \log deg(u))$ time if the metric is the L1 metric between scalars.

language and the Arachne graph framework. Our implementation leverages Chapel's parallelism and the `coforall` construct to parallelize the construction of local join sequences. If a `for` statement is used in a `coforall` loop, Chapel will automatically parallelize the loop across locales (i.e., the body of the loop will be executed concurrently, on different locales, usually different machines).

6.3.4 Algorithm for Cluster Reconstruction

Recall that HDBSCAN's first step is in obtaining a dendrogram that is then "cut" to obtain the clusters, where cutting is just an informal term for stability maximization. To motivate the question, we first give a construction for the dendrogram naively, under a shared-memory architecture.

We reuse the previous notation of $E(u) \subseteq E$ for some node $u \in V$ to be the "1-edge neighborhood" of u, which contains all the edges of form $(u, \cdot)$ or $(\cdot, u)$. Similarly, we define $Adj(e) \subseteq E$ to be all the edges "touching" $e \in E$ in G. If e is the edge (u, v), then $Adj(e) = E(u) \cup E(v)$.

For the naive, non-parallel dendrogram algorithm, first we obtain the join sequence induced by d on G using the following algorithm. This algorithm follows from the definition of the join sequence:

Algorithm 6.1 MakeJoinSequence

function MakeJoinSequence(G) ▷ *Single-locale version*

Initialize empty parallel sorted set L of tuples (u, v, t) sorted increasing by t. This L denotes a join sequence, and u, v are edges in the graph but can be thought of as points in the dendrogram.

for all node u in G in parallel **do**

for all unique pairs (e_1, e_2) in $E(u)$ such that WLOG $e_1 < e_2$ **do**

Insert $(e_1, e_2, d(e_1, e_2))$ into L

return L

Then the join sequence can be used to construct the dendrogram in a serial fashion without parallelism:

Algorithm 6.2 RealizeDendrogram

function RealizeDendrogram(L)

initialize empty dendrogram D of size $m = |E|$

for each (u, v, t) in L **do**

Join(u, v, t) on D

return D

Parallel programming offers limited ways to distribute the RealizeDendrogram process, as each join depends on the completion of the previous one, making the algorithm inherently sequential. This sequential dependency is also partly why neighbor-joining, a popular hierarchical clustering algorithm in phylogenomics, typically employs parallelization or vectorization to speed up distance calculations [190], [191]. The inherent structure of the join operations poses significant limitations for parallelization.

To proceed further, we need to set some boundaries for our distributed non-shared memory algorithm. Namely, we assume that there is a practical reason that motivated people using the distributed graph architecture in Arachne. Namely, we assume that the edges do not fit on a single machine, or else there would be a strong reason to store all the edges on the single machine to avoid all the overhead of distributed programming. To put it more formally:

Assumption 6.1. In a multi-locale environment with k locales, each locale is limited to storing at most $m/k + c$ edges, where m is the number of edges and c is a constant. Consequently, in general, no single locale is capable of holding the entire dendrogram D, of size m, at any point.

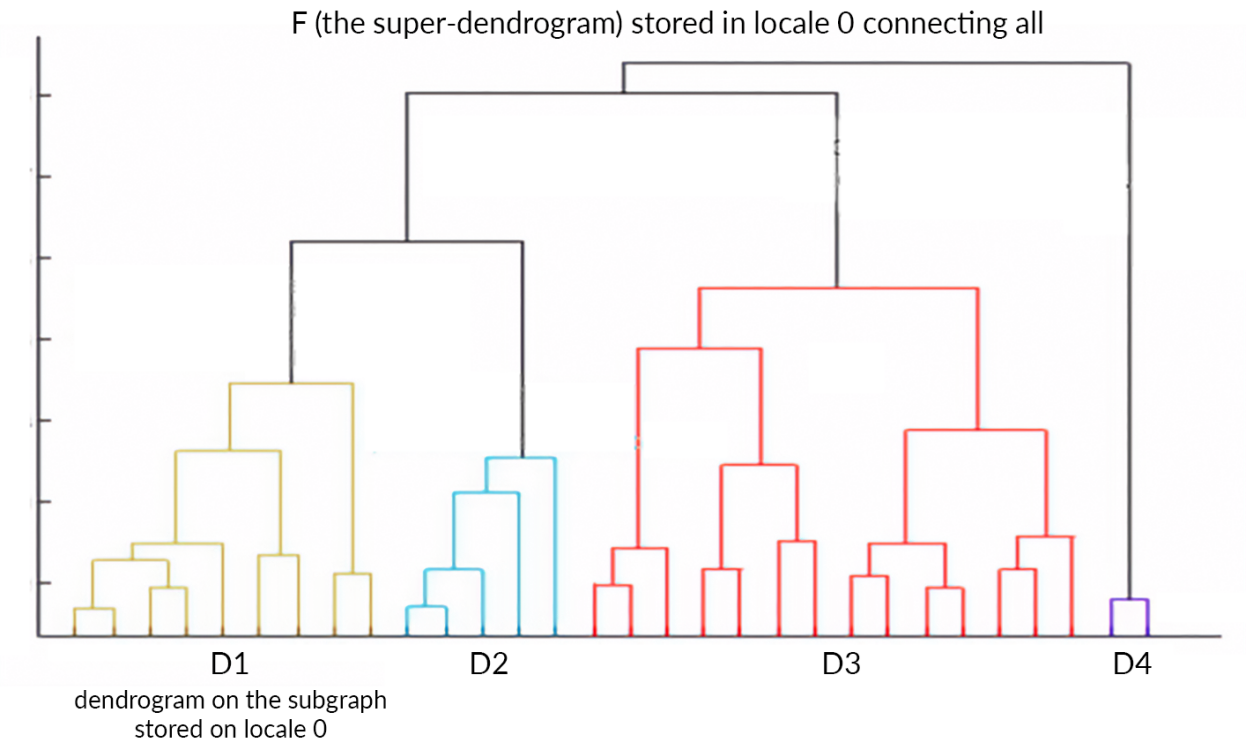


Figure 6.2: Illustration of idea of the multi-locale algorithm on four locales. The x-axis contains points in the dendrogram, equivalently edges in the Arachne graph. For instance, Arachne partitions the green edges to locale 0 in this illustration; hence we construct a dendrogram D_1 first on these edges.

This assumption significantly influences our approach, especially since it precludes the use of MapReduce[192]-based techniques. In the standard MapReduce framework, the reduce stage aggregates the dendrogram D into a single machine. However, our assumption explicitly prevents storing D entirely within any single locale.

Our additional assumption is that we can sacrifice accuracy with respect to the single locale version in the construction of D if the decrease in accuracy means significantly less cross-locale communication. This is our approximation aspect. To be explicit:

Assumption 6.2. We can sacrifice accuracy in the construction of D if the decrease in accuracy means significantly less cross-locale communication.

These two assumptions lead to a straightforward multi-locale modification of the single-locale version of the algorithm. Because in the single-locale version, the dendrogram is constructed on all the edges of the graph, in the multi-locale version, we can have each locale construct a dendrogram on all the edges of the graph this locale owns, as the Arachne graph framework partitions the edges of the graph across locales.

In other words, each locale owns a "local dendrogram". Then we construct a "super-dendrogram" on top of all the local dendrograms (see also Figure 6.2). The local dendrograms only clusters the edges that are stored in that locale. The pseudocode of this modified version is given below.

Algorithm 6.3 NaiveDistributedMakeJoinSequence

function NaiveDistributedMakeJoinSequence(G) ▷ *Multi-locale version*
  Initialize empty parallel sorted set $L_1, L_2, ..., L_k$ of tuples (u, v, t) sorted increasing by t ▷ *these sets are stored in different locales, and each locale owns a join sequence.*
  for $i = 1$ to k **do** ▷ *coforall*
    for all node u in stored in locale i **do** ▷ *Can be in parallel*
      for all unique pairs (e_1, e_2) in $E(u)$ such that $e_1 < e_2$ and both e_1, e_2 are stored locally in locale i **do**
        Insert $(e_1, e_2, d(e_1, e_2))$ into L_i
  return $L_1, L_2, ..., L_k$

Algorithm 6.4 NaiveDistributedRealizeDendrogram

function NaiveDistributedRealizeDendrogram($\{L_i\}$)
  initialize empty local dendrograms $D_1, D_2, ..., D_k$
  for $i = 1$ to k **do** ▷ *coforall*
    for each (u, v, t) in L_i **do**
      Join(u, v, t) on D_i
  Initialize a new dendrogram F of size k, representing a "super" dendrogram joining $D_1 \dots D_k$ on locale 0
  Construct F in by iterating through all pairs of e_1, e_2 s.t. e_1 and e_2 are stored in different locales and e_1, e_2 adjacent
  return $F, D_1, \dots, D_k$

To run the algorithm from start to finish given input G, one compose the above two algorithms, i.e., NaiveDistributedMakeJoinSequence (which produces the join sequences on each locale) followed by NaiveDistributedRealizeDendrogram (which constructs the dendrogram on each locale and then constructs the super-dendrogram F).

The above naive multi-locale algorithm is very efficient because each locale can run entirely independently producing $D_1, \dots, D_k$ before we construct the super-dendrogram F. The main problem is loss of (re-implementation) accuracy, each local dendrogram can have discrepancies with the "true" dendrogram, defined as the dendrogram obtained by the non-distributed

version of the algorithm considering all m edges of the graph. This is because each local dendrogram only considers pairwise relationships on edges it owns in its locale, which will lose information.

An additional benefit from the above multi-locale version makes top-down or bottom-up traversals relatively easy on the dendrogram formed by $F, D_1, \ldots, D_k$. For example, when doing top-down traversal (e.g., post-order), one can traverse F first in a non-distributed fashion, and then traverse $D_1, \ldots, D_k$ distributed.

Algorithm 6.5 Bucketed Distributed Hierarchical Clustering

```
function BucketedDistributedClustering({L_i}, b) ▷ Replacement of NaiveDistributedRealizeDendrogram
    initialize empty local dendrograms D_1, D_2, ..., D_k        ▷ k is the number of locales
    initialize empty global dendrogram G
    divide t values into b uniform buckets in sorted order
    for i = 1 to b do                                            ▷ Processing each bucket
        for all locales do                                       ▷ coforall, parallel processing
            Local Stage:
            for each (u, v, t) in L_i belonging to bucket i do
                LocalJoin(u, v, t) on D_i
            Global Stage:
            for each (u, v, t) in L_i belonging to bucket i do
                RemoteJoin(u, v, t) on G
    return G, D_1, ..., D_k
```

Remote joins can be defined as the number of times a locale queries another locale for cluster component determination. A remote join requires communication cross-node, and hence is slower than a local join. The number of remote joins thus acts as a hardware independent proxy for the speed of the algorithm, and the lower the number of remote joins, the faster the algorithm.

Bucketing

The bucketing approach, detailed in Algorithm 6.5, segments the processing of edge pairs based on their distance measure (`t` values) into distinct intervals or `buckets`. Given 1 bucket, bucketing is equivalent to no bucketing at all. The number of buckets b is a tunable parameter that can be adjusted to balance the trade-off between the accuracy of our approximation and

performance. A higher number of buckets can lead to more accurate results but will most likely increase the number of remote joins, slowing down the algorithm.

6.4 EXPERIMENTAL SETUP

This study investigates two research questions:

- **RQ1:** Does a shared-memory parallelism implementation in Chapel significantly improve the runtime of the HDBSCAN algorithm compared to a Python implementation?
- **RQ2:** Does our proposed multi-locale (non-shared-memory) algorithm for the hierarchical clustering phase of HDBSCAN produce accurate approximations of the clustering results?

We perform two experiments using the SNAP dataset collection [193] to answer these questions. We aim to ensure that the Chapel implementation achieves comparable accuracy to the Python version, despite potential discrepancies arising from differences in implementation details like tie-breaking or floating-point precision. For the second experiments, we assess the preservation of accuracy of our multi-locale cluster generation algorithm by varying the number of locales and buckets, focusing on algorithm precision rather than just performance under our approximation scheme.

6.4.1 Metrics

Our evaluation uses the Adjusted Rand Index (ARI) and Adjusted Mutual Information (AMI) to compare the accuracy and mutual information of our algorithm's edge clusters against a reference implementation, either the original Python implementation or our Chapel implementation (that is later proven to be practically identical to the Python implementation). Both metrics are calculated using functions in scikit-learn [194], using `adjusted_mutual_info_score` and `adjusted_rand_score` under `sklearn.metrics`.

Due to their high correlation, we focus primarily on ARI in our reporting. ARI ranges from -0.5 to 1, where 1 indicates identical clusterings and 0 indicates random clusterings (values close to -0.5 indicate very discordant clusterings). AMI can be at most 1, where 1 indicates identical clusterings and values close to 0 indicates independent clusterings.

Experiment 1: Performance Evaluation

We measure the wall-clock running time to assess the performance improvement of our Chapel implementation. We also analyze scalability by testing different network sizes from the SNAP dataset collection and ensure accuracy by comparing the ARI and AMI between our implementation and the original.

Experiment 2: Accuracy Evaluation

We focus on the Adjusted Rand Index (ARI) as our primary metric to reflect the similarity of our algorithm's clustering with the original. We still show AMI and due to the high correlation between ARI and AMI, we omit dwelling on AMI for simplicity.

6.4.2 Datasets

In Experiment 1, we test on a variety of temporal networks from SNAP, reflecting diverse sizes and densities. Experiment 2 focuses on the CollegeMsg network due to its moderate size and the large number of parameters. Moreover, here in Experiment 2, our primary concern is accuracy, but we also run our results on larger networks to validate that our results from this moderately-sized CollegeMsg network generalize. Table 6.1 summarizes the key properties of the selected datasets.

6.4.3 Experimental Environment

We note that our experiments were conducted on a shared-memory machine with 32 cores and at least 64GB of memory on the Illinois Campus Cluster, a heterogeneous cluster that has a four-hour running time limit, with at least 64 GB memory available. Given the heterogeneity, the running times are not directly comparable to a homogeneous cluster.

6.4.4 Results & Discussion

Experiment 1: Chapel vs Python on SNAP

We compared the performance of our Chapel implementation with the original Python implementation on the SNAP dataset collection. We measured the time taken to construct the join sequence and the dendrogram, which are key steps in the HDBSCAN algorithm.

Name	Nodes	Temporal Edges	Static Edges
sx-stackoverflow	2,601,977	63,497,050	36,233,450
wiki-talk-temporal	1,140,149	7,833,140	3,309,592
sx-superuser	194,085	1,443,339	924,886
sx-askubuntu	159,316	964,437	596,933
soc-RedditHyperlinks	55,863	858,490	858,490
sx-mathoverflow	24,818	506,550	239,978
CollegeMsg	1,899	20,296	59,835
email-Eu-core-temporal	986	332,334	24,929

Table 6.1: Summary of SNAP Dataset Collections Used in Experiment 1. Static edges are the number of edges in the static graph that do not contain temporal information (i.e., not representing communication, one-directional relationships). Both the Chapel and Python version are multithreaded (or due to Python's limitation of the Global Interpreter Lock, multi-processed) using 16 threads.

Due to potential floating-point discrepancies, we used the Adjusted Rand Index (ARI) to assess the similarity between the clusters produced by both implementations.

Dataset	Python (time, s)	Arachne (time, s)	AMI	ARI
sx-stackoverflow	1,200.00	31.31	0.995	0.994
wiki-talk-temporal	302.40	8.29	0.995	0.994
sx-superuser	86.72	2.12	0.996	0.995
sx-askubuntu	43.32	1.50	0.996	0.995
soc-RedditHyperlinks	73.41	2.15	0.994	0.993
sx-mathoverflow	19.30	0.60	0.993	0.991
CollegeMsg	5.61	0.15	0.996	0.995
email-Eu-core-temporal	2.34	0.12	0.996	0.995

Table 6.2: Running times of Chapel vs Python on SNAP datasets. All times are in seconds, and using the Python version as reference, Arachne's ARI and AMI has consistently achieved > 0.99 when comparing to the reference clusterings produced using Python (practically identical).

We show the running time results in Table 6.2. As shown from the table, our Chapel implementation consistently outperforms the Python implementation, achieving significant speedups across all datasets. In most cases, the Chapel implementation is more than an order of magnitude faster than its Python counterpart. As expected, our implementation is *correct* – these performance enhancements do not compromise the faithfulness of our algorithms. The Adjusted Rand Index (ARI) values, comparing the Chapel and Python implementations, consistently exceed 0.99, same as AMI, indicating an almost perfect agreement in the identified

community structures. AMI is consistently higher than ARI, possibly due to AMI's range is from 1 to around 0 (can be slightly negative) while ARI ranges in $[-0.5, 1]$.

The relative speedup provided by the Chapel version does not seem to correlate with the size (number of temporal edges) of the dataset, which is consistent as our level of parallelism is not directly related to the size of the dataset.

Experiment 2: Evaluating the Accuracy of the Multi-Locale Algorithm

We implemented both the naive and bucketed versions of the multi-locale HDBSCAN algorithm in Chapel. The naive version serves as a baseline, while the bucketed version introduces a strategy to improve accuracy by interjecting remote joins between phases of local joins. We evaluated the performance of both algorithms on the CollegeMsg network from SNAP, which comprises 1899 nodes and 59835 edges. Although this is a small network, it provides a representative dataset for our accuracy evaluation (a "training" dataset, in some sense). We later ran selected settings on larger scale networks, namely including our results on sx-stackoverflow, and the results were consistent with the CollegeMsg network.

We used the Adjusted Rand Index (ARI) to compare the clustering results of our multi-locale algorithms with the original single-locale HDBSCAN implementation, treating the latter as the ground truth. We also calculated AMI to ensure that the results are consistent across different evaluation metrics, but as mentioned we focus our discussion on ARI due to its high correlation with AMI.

Experiment 2(a): Investigating the Impact of Locale Increase on Naive Version Accuracy

The results shown in Table 6.3 confirm an obvious decline in the accuracy of the naive algorithm as the number of locales increases, where accuracy is defined as the faithfulness to the reference implementation in Chapel. At two locales, the Adjusted Rand Index (ARI) is 0.576, indicating a moderate agreement with the ground truth. However, this figure significantly decreases to 0.096 at 128 locales, highlighting a substantial loss in clustering accuracy. Similarly, the Adjusted Mutual Information (AMI) values decrease from 0.708 at two locales to just 0.137 at 128 locales. This downward trend in both ARI and AMI with an increase in locales highlights the challenges associated with managing a higher number of locales, with graphs stored on different locales leading to a decrease in clustering accuracy.

Experiment 2(b): Assessing the Effectiveness of Bucketing on Accuracy

We evaluated the bucketed version of the algorithm with different bucket counts: 100, 10,000, and 1,000,000. The highest bucket count essentially assigns each edge pair to its own bucket, maximizing the frequency of remote joins and potentially leading to higher accuracy.

The results are shown in Table 6.4. The configuration with 100 buckets consistently underperforms compared to the single-bucket (naive) setup, indicating that too frequent switching between local and remote joins can be detrimental to accuracy. The 10,000 bucket configuration achieves the highest accuracy among practical settings, particularly with 32 locales. However, the improvement over the single-bucket setup is not substantial.

The 1,000,000 bucket configuration, while impractical in real-world scenarios due to the high communication overhead, achieves near-perfect clustering accuracy (ARI of 0.999) at 32 locales. This suggests that the bucketing strategy has the potential to achieve high accuracy if the communication overhead can be effectively managed.

Generalized results for larger networks

We also ran the selected settings on larger networks, including the sx-stackoverflow network. The results are shown in Table 6.5 and Table 6.6. The comparison of Table 6.5 and Table 6.6 reinforces the trends observed in smaller networks, with bucketing consistently impacting accuracy and network efficiency. Notably, the non-bucketed configurations exhibit a clear degradation in accuracy as the number of locales increases. Conversely, the bucketed configurations maintain or even improve accuracy in this larger network.

Num locales	**ARI**	**AMI**	**Proportion of remote joins**
1	1.000	1.000	0.000
2	0.576	0.708	0.085
4	0.286	0.487	0.175
8	0.136	0.333	0.263
16	0.099	0.251	0.337
32	0.097	0.197	0.406
128	0.096	0.137	0.509

Table 6.3: Accuracy of the reimplementation (non-approximated version vs. approximated version) and remote joins per number of locales for the multi-locale HDBSCAN, showing the results for accuracy (ARI and AMI) and network structure (Proportion of remote joins). A higher ARI and AMI indicate better clustering accuracy. Dataset: CollegeMsg (from SNAP, 1899 nodes and 59835 edges).

Num locales	Num buckets	ARI	AMI	Proportion of remote joins
4	1	0.286	0.487	0.175
4	100	-0.011	-0.002	0.175
4	10,000	0.006	0.249	0.175
4	1,000,000	1.000	1.000	0.175
16	1	0.099	0.251	0.337
16	100	-0.011	-0.002	0.337
16	10,000	-0.022	0.105	0.337
16	1,000,000	0.999	0.999	0.337
32	1	0.097	0.197	0.406
32	100	-0.011	-0.002	0.406
32	10,000	0.503	0.670	0.406
32	1,000,000	0.999	0.999	0.406
128	1	0.096	0.137	0.509
128	100	-0.016	0.100	0.509
128	10,000	-0.027	0.189	0.509
128	1,000,000	0.999	0.999	0.509

Table 6.4: Evaluation of accuracy across different bucket configurations in multi-locale HDBSCAN, integrating both ARI and AMI measures. This table now includes the effect of bucket granularity on both the clustering accuracy (ARI and AMI) and the network efficiency (Proportion of remote joins).

Num locales	ARI	AMI	Proportion of remote joins
1	1.0000	1.0000	0.0000
2	0.0612	0.1525	0.3981
4	0.0015	0.0631	0.5505
8	0.0104	0.0510	0.6216
16	0.0130	0.0442	0.6837
32	0.0123	0.0407	0.7289
128	0.0107	0.0363	0.8019

Table 6.5: Evaluation of accuracy and network efficiency with no bucketing across different configurations on the sx-stackoverflow dataset (2,601,977 nodes & 63,497,050 edges), using our HDBSCAN-based approach.

Num locales	Num buckets	ARI	AMI	Proportion of remote joins
4	1	0.0015	0.0631	0.5505
4	100	0.0397	0.0003	0.5505
4	10000	0.0843	0.0127	0.5505
4	1000000	1.0000	1.0000	0.5505
16	1	0.0130	0.0442	0.6837
16	100	0.0391	0.0003	0.6837
16	10000	0.0843	0.0127	0.6837
16	1000000	1.0000	1.0000	0.6837
32	1	0.0123	0.0407	0.7289
32	100	0.0517	0.0023	0.7289
32	10000	0.0842	0.0127	0.7289
32	1000000	1.0000	1.0000	0.7289
128	1	0.0107	0.0363	0.8019
128	100	0.0516	0.0023	0.8019
128	10000	0.0953	0.0185	0.8019
128	1000000	1.0000	1.0000	0.8019

Table 6.6: Evaluation of accuracy and network efficiency with varying numbers of buckets across different configurations on the sx-stackoverflow dataset (2,601,977 nodes & 63,497,050 edges), using our HDBSCAN-based edge clustering approach.

6.5 CONCLUSION & FUTURE WORK

We set out to perform overlapping community detection, and our technical contributions lie in our exploration of both shared memory and distributed memory of the HDSBCAN algorithm. We have shown that Chapel can significantly accelerate the HDBSCAN algorithm for overlapping community detection in large-scale networks in the shared-memory case, and we have proposed a multi-locale algorithm for approximating HDBSCAN, which can be used in a distributed memory setting. While the multi-locale algorithm can be accurate, the naive version is not accurate enough, and the bucketed version can be accurate but is not scalable.

While our work has focused on a specific type of networks, the results should generalize. Technique-wise, our implementation of HDBSCAN in Chapel can be used in other clustering applications, beyond clustering edges of a network. Methods to deal with other types of graphs (e.g., undirected, unweighted) have also been suggested, namely to add bidirectional edges to the graph, or use off-the-shelf methods to obtain edge embeddings.

Distributing HDBSCAN itself is a research problem. Many existing approaches tackled distributing DBSCAN, e.g., [195], [196], but the literature on distributing HDBSCAN is comparatively sparse (for shared memory we have [197]; MapReduce-based methods include [198], [199]). Our work is a step towards distributed HDBSCAN, but clearly more work is

needed to make it practical.

Future work should aim at refining the scalability of the multi-locale algorithm and its accuracy with respect to the single-locale algorithm, particularly focusing on the bucketing strategy. This might include investigating alternative bucketing methods that could consider adaptive strategies or base the bucketing on network properties such as edge weights, thereby optimizing accuracy while minimizing communication overhead. Another avenue is the development of more sophisticated communication protocols to efficiently manage data exchange between locales during the global join phases.

Recent advancements in Chapel, particularly in GPU programming, present an exciting opportunity for further performance enhancements. The inherently parallel nature of the HDBSCAN algorithm, along with the growing capabilities of CUDA in graph algorithms, suggests that GPU acceleration could significantly improve scalability and efficiency. Exploring other high-performance computing languages, like Julia, known for their robust support in multi-locale programming and increasing popularity in scientific computing, may also yield additional advantages for our multi-locale HDBSCAN implementation.

Chapter 7: Conclusion

This thesis has explored the development of algorithms for phylogenomics, specifically focusing on species tree inference and multiple sequence alignment. Our focus has been on designing algorithms that are not only accurate but also efficient, capable of scaling to large genomic data. Finally, we extended our research to parallel and distributed community detection in large networks.

In phylogenomics, we introduced Weighted ASTRID, an improved version of the ASTRID method that integrates gene tree uncertainty into species tree estimation. This approach significantly improves accuracy, particularly under realistic conditions of gene tree reconstruction error, compensating for a notable limitation of traditional summary methods. Additionally, we ventured into constrained species tree inference with the introduction of two new methods, NJst-J and FASTRAL-J, integrating prior species tree knowledge for enhanced refinement. We also introduced WITCH-NG, an efficient and algorithmically better motivated version of WITCH, for multiple sequence alignment under sequence length heterogeneity. Our common thread is the development of both theoretically efficient algorithms, but these algorithms are also efficiently implemented.

Beyond phylogenomics, our research extended to the investigation of overlapping community detection in large networks using high-performance computing (HPC) techniques. We developed parallel algorithms for key steps in the HDBSCAN clustering method, aiding efficient analysis of large networks to discover commonly occurring community structures.

As we discuss future work, it is useful to gauge how much phylogenomics has advanced. Summary methods, in particular, having evolved considerably since the introduction of early techniques such as GLASS [70] and BUCKy [200]. Today's summary methods are more accurate and faster than the previous generation of methods.

Our work paves the way for several exciting directions for future research. First, it is important to note that distance-based species tree estimation remains an active area of research. This line of inquiry, originating with approaches like GLASS and STAR [146], has recently seen further developments [201], [202] alongside our work. One exciting implication of weighted ASTRID is its demonstration that incorporating branch support from gene trees can enhance the accuracy of distance-based species tree estimation, and therefore it is not just quartet-based methods that can benefit from this approach, and in fact, it was conceptually very easy to incorporate gene tree support in ASTRID.

Performance has been a major theme throughout this dissertation. The trend towards increased efficiency in species tree estimation is evident. As discussed, the traditional and

popular concatenation method is less efficient than summary methods for large phylogenomic datasets [4], [59]. ASTRAL has shown improved performance over previous quartet-based methods and prior summary methods [51], and wASTRAL is even faster. ASTRID essentially serves as an efficient reinterpretation of the earlier summary method NJst, and wASTRID offers a more efficient reworking of ASTRID in addition to accuracy improvements. Remarkably, wASTRID completed the analysis of the Avian phylogenomics data in approximately one second — that is, 14,446 gene trees on 48 taxa analyzed in one second (of course, inferring the gene trees also takes lots of time not accounted in this one second). This marks substantial progress in performance throughout the years, not to mention the accuracy improvements. Furthermore, we demonstrated that, without a more efficient big-O notation algorithm, we could accelerate the computation of wASTRID over ASTRID by more than a factor of five on this important Avian biological dataset. It is conceivable that similar techniques could be applied to widely used computational biology software to speed-up computations by significant constant factors. However, considering that ASTRID required no more than seven seconds and wASTRID about one second, one might question the practical value of further performance enhancements, and further methods might find better use of using the extra time-budget to improve accuracy or incorporate more complex models.

Another curious note on performance is the recent convergence of Python as the dominant language used by both computational biology and deep learning. Although Python is a language traditionally critiqued for its performance, deep learning-focused libraries such as Jax [203] have empowered Python with the ability to efficiently manage computations on GPUs or through multi-threaded processes, minimizing the need for manual intervention (e.g., no manual writing of CUDA kernels or multithreading). This capability could allow bioinformatics software relying on Numpy [204] for performance (e.g., HMMerge [205], APPLES [206]) to transition with low friction to Jax for enhanced performance on CPUs or GPUs, because Jax offers a Numpy-compatible API. Relatedly, this dissertation has focused on using performant languages to speed up computations; however, for highly parallelized tasks, the trend might shift towards using specialized computing devices (GPUs, TPUs) with Python acting as the intermediary, similar to trends observed in deep learning. Additionally, much of our performance engineering efforts for WITCH-NG involved manual parallelization and algorithmic block reshuffling to facilitate compiler auto-vectorization. With the recent shift towards GPU-centric computing power, reimplementing WITCH-NG in a framework like Jax or PyTorch [207] could yield substantial performance gains while maintaining the ability to operate on accelerators. Ease writing fast scientific software is a significant field of research of its own [208], [209].

Furthermore, the field of multiple sequence alignment has recently enjoyed the emergence

of many methods, particularly those tailored for sequence length heterogeneity [39], [41]–[45], including our own contributions with WITCH-NG. With MAGUS [177] also demonstrating good sequence alignment accuracy in various cases [42], choosing among these diverse methods can be daunting to practitioners. It would be intriguing to develop a unifying or "meta-ensemble" method that integrates these techniques, as they exhibit varying strengths. Even low-hanging fruits, such as creating a unified interface or prepackaging all mentioned software inside one package could be highly beneficial to the academic community.

On a wider note, given the many tradeoffs between different methods, e.g. wASTRID vs. wASTRAL, which is a continuation on the continuing discussion of performance between ASTRID and ASTRAL, and WITCH-NG vs. the many other methods, understanding on a deeper level the trade offs and why different methods perform differently is crucial, which connects us back to the eternal question of having good simulated data to test these hypotheses. With even more choices (which weighted method; should we estimate directly from alignments; which branch support to choose; which alignment method to use), the need for a comprehensive benchmarking suite and guidelines for selecting the most appropriate method for a given dataset is more pressing than ever.

On the topic of data, as genome-scale data continues to expand, the underlying data infrastructure increasingly reveals its limitations. Both WITCH-NG and wASTRID required the conversion of inefficient on-disk data formats to more efficient in-memory formats, a challenge not unique to our projects (cf., the recent codebases of [66], [177], [210]). A more standardized and efficient data format could unify this domain for better interoperability and alleviate performance bottlenecks associated with parsing these formats in slower programming environments like Python.

Many components of our work have broad applicability. For example, wASTRID's improved performance is attributable to its cache-efficient tree structure, and WITCH-NG incorporates a more efficient algorithm for transitive alignment merging. These enhancements could be readily integrated into other software.

Technique-wise, a notable direction for accuracy is the hybrid weighting in Weighted ASTRID, where incorporating branch lengths into the weighting scheme could potentially enhance accuracy, especially under specific conditions where it closely rivals wASTRAL-h. Additionally, the extension of Weighted ASTRID and the constrained species tree methods to handle gene duplication and loss events would not only broaden their applicability but also potentially improve accuracy in under the presence of multi-copy gene families.

Another promising area is the application of Weighted ASTRID in supertree estimation. This could provide a fast and accurate method for a notoriously challenging problem that can be key to piecing together the tree of life. Similarly, the integration of Weighted ASTRID

into the FASTRAL pipeline might lead to more precise and efficient methods for large-scale species tree estimation. Moving beyond traditional accuracy metrics, exploring alternative criteria like quartet-similarity could prove valuable, particularly for evaluating species tree estimation methods in the presence of rogue taxa and in very large datasets. Moreover, extending the scalability of Weighted ASTRID and other methods to handle even larger phylogenomic datasets can prove beneficial. This might involve investigating parallelization strategies and more efficient data structures.

For overlapping community detection, the exploration of GPU acceleration for key steps in the HDBSCAN algorithm could significantly enhance performance by leveraging the parallelism and recent advancements in GPU computing for graph algorithms. Finally, examining the use of alternative high-performance computing languages, such as Julia, known for their strong support in multi-locale programming and growing popularity in scientific computing, could offer substantial benefits for our multi-locale HDBSCAN implementation.

Appendix A: Additional Results and Materials for Weighted ASTRID

A.1 COMMANDS

A.1.1 Weighted ASTRID

The commands differ depending on the type of support annotated on the gene trees. For FastTree SH-like, bootstrap support, and aBayes support, the commands for wASTRID-s are respectively:

```
wastrid -i $genes -o $output # FastTree SH-like
wastrid -x 100 -i $genes -o $output # bootstrap
wastrid -n 0.333 -i $genes -o $output # aBayes
```

For the final setting of wASTRID-pl (distance defined by path lengths, with branch lengths normalized by the maximum path-length in tree), the above commands need to be appended with an additional flag: `-m n-length` (distance using "normalized branch length" instead of the default `-m support`).

A.1.2 Other Summary Methods

Weighted ASTRAL

We ran hybrid-weighted ASTRAL (v1.4.2.3) using the following command on trees annotated with aBayes support:

```
astral-hybrid -x 1 -n 0.333 -i $genes -o $output
```

ASTRID

For all experiments except for Experiment 4, we ran ASTRID-2 compiled from scratch from [143] without missing data imputation, using the following command:

```
ASTRID -s -i $genes -o $output
```

In Experiment 4, we instead used the following command:

```
ASTRID -u -n -s -i $genes -o $output
```

ASTRAL

We ran ASTRAL (v5.7.8) using the following command:

```
java -jar astral.5.7.8.jar -i $genes -o $output
```

FastME

ASTRID and wASTRID bind into and call subroutines inside FastME directly instead of calling FastME's CLI interface. We bundle our software with FastME (v2.1.5). See `https://github.com/RuneBlaze/internode/blob/3791b7e344869f3beb9f11e3d588e5e4eff788ba/src/internode.rs#L345` for how wASTRID binds into FastME, which is a translation of the C++ code from ASTRID.

A.1.3 Obtaining Gene Tree Branch Support

Approximate Bayesian support

We computed IQ-TREE (v2.1.2) aBayes support using the following command:

```
iqtree2 -s $aln -te $gtree -m GTR+G -abayes
```

where `$aln` is the alignment file for the gene tree, and `$gtree` is the path to the gene tree topology.

Bootstrap support

We computed bootstrap support (on the training dataset) using FastTree (v2.1.11) on bootstrap replicates generated by Goalign (v0.3.5) [211].

The following command was used to generate bootstrap replicates (`$aln` is the original gene alignment):

```
goalign build seqboot -i $aln -S -n 100
```

The FastTree command used for inferring a tree from one bootstrap replicate is (`$bs_aln` is one bootstrap alignment replicate generated by Goalign):

```
FastTree -nt -gtr -nosupport $bs_aln > $output
```

The FastTree bootstrap trees are then randomly resolved to eliminate polytomies, and then mapped by RAxML-NG [66] to the original gene trees using the following command:

```
raxml-ng --support --tree $gtree --bs-trees $bs_trees --bs-metric fbp
```

where `$gtree` is the path to the original gene tree, and `$bs_trees` contains new-line separated newick trees inferred on the bootstrap replicates.

A.2 SUPPLEMENTARY TEXT

A.2.1 Failures to Complete

ASTRAL failed to complete 24/50 replicates (replicates 2, 4, 6, 8, 9, 11, 13, 15, 16, 17, 20, 23, 27, 28, 29, 30, 32, 36, 38, 39, 40, 41, 48, and 49) on the MC11H condition ($n = 1001$) under $k = 1000$. The 24 log files indicate that ASTRAL timed out on each of the replicates. An example of the last three lines of the log files is attached:

```
Calculated 700000 weights; time (seconds): 1549
Calculated 800000 weights; time (seconds): 1544
slurmstepd: error: *** JOB 5328647 ON golub041\
CANCELLED AT 2022-05-08T05:24:48 DUE TO TIME LIMIT ***
```

A.2.2 Branch Support

Many techniques are available to assess branch support. In the following paragraph, let T denote the gene tree that we aim to annotate each non-leaf edge $e \in E(T)$ with support, and let A be the gene alignment that we used to infer T.

The most common approach is the bootstrap method first proposed by Joe Felsenstein [90]. Briefly, a fixed number of "bootstrap alignments" $\{A_1, \ldots, A_k\}$ (k a parameter commonly set to 100 or 1000) are sampled with replacement from columns of A. Each bootstrap alignment A_i has same number of columns as A. Bootstrap trees $\{T_i\}$ are then inferred from the bootstrap alignments using the same method used to infer T. Each internal edge $e \in E(T)$ has support defined as $s(e) = |\{\pi_e \in C(T_i) \mid 1 \leqslant i \leqslant k\}|/k$. For example, if $s(e) = 1$, the bipartition associated with e appears in every bootstrap tree and so has bootstrap support of 1.0. Variants of Felsenstein's support exists, including Rapid Bootstrap Support [212] and Ultrafast Bootstrap [213] designed for speed. A different approach, the Transfer

Bootstrap Expectation [53], was proposed in order to enable robustness to rogue taxa in large phylogenies.

This study extensively uses the approximate Bayes branch support [144], available in the IQ-TREE [67] software, which is an approximated version of the posterior probability of an NNI configuration at a particular branch. Similarly only exploring the NNI configurations around a branch, FastTree SH-like support, which relies on a Shimodaira-Hasegawa-like procedure [214], [215], was also explored in this study.

The Bayesian posterior probability support [216], a well known form of support approximating the posterior probability of the bipartition, seems computationally infeasible for datasets in this study ($n \geqslant 200$) and is thought to be an overconfident form of support [217].

A.3 SUPPLEMENTARY FIGURES & TABLES

Table A.1: Wall-clock running times of ASTRID-2 and ASTRID-3, which differ only in the calculation of the distance matrix. Time are shown in seconds, median across 50 replicates.

Running Time (s)	**Distance Matrix**		**FastME**	
	ASTRID-2	**ASTRID-3**	**ASTRID-2**	**ASTRID-3**
S101 (seqlen = 400, n = 101)	0.32	0.07	0.00	0.00
MC5 (n = 201)	0.53	0.15	0.01	0.01
MC11H (n = 1001)	6.37	1.75	1.82	1.66

Appendix B: Additional Results and Materials for Fast and Scalable Species Tree Inference with Partial Constraints

B.1 COMMANDS AND SOFTWARE AVAILABILITY

B.1.1 ASTRAL with Constrained Search (ASTRAL-J)

We used ASTRAL with constrained search version 5.6.9 (from `https://github.com/maryamrabiee/Constrained-search`), and ran the following command for inferring species trees:

```
java -jar astral.5.6.9.jar -i $genes -j $backbone -o $output
```

B.1.2 ASTRAL

We used ASTRAL version 5.7.7 and ran the following command for inferring species trees:

```
java -jar astral.5.7.7.jar -i $genes -o $output
```

For getting the localPP [154] branch support values, we used the following command:

```
java -jar astral.5.7.7.jar -i $genes -q $tree -o $scored_tree
```

B.1.3 NJst-J

Our implementation of NJst-J is available at https://github.com/RuneBlaze/NJst-constrained for Linux x86_64 systems. We used the following command for running NJst-J:

```
python3 njst_constrained.py -i $genes -j $backbone -o $output
```

NJst-J depends on Python 3, TreeSwift (version 1.1.19) [148], and optionally LuaJIT for speed. We used Python 3.7.0 and LuaJIT 2.1.0-beta3 for the experiments. The other Python software packages that are required are listed in the README in the GitHub repository linked. NJst-J internally uses code from ASTRID [5] in its computation of the distance matrix.

B.1.4 FASTRAL-J

Our implementation of FASTRAL-J is available at `https://github.com/RuneBlaze/FASTRAL-constrained` for Linux x86_64 systems. We used the following command for

running FASTRAL-J:

```
python3 fastral_constrained.py -i $genes -j $backbone -o $output
```

FASTRAL-J depends on Python 3 and requires Java for invoking subroutines from both ASTRAL and constrained ASTRAL. The other Python software packages that are required are listed in the README in the GitHub repository linked. FASTRAL-J internally uses code from ASTRID [5].

B.1.5 Re-estimating the Gene Trees

The gene trees were re-estimated from the alignments using FastTree version 2.1.11 with the following command:

```
FastTree -nt -gtr < $input_aln > $output_tree
```

During re-estimation, the input alignments to FastTree were shortened versions of the original gene alignments truncated to half length (rounded up). To truncate the alignments, we used the script at `https://git.io/Jy0IV` with the following command (with `$input` as the path to the original gene alignment):

```
python3 truncate_aln.py -i $input -o $output
```

B.1.6 Contracting the Shortest Branches

To generate the backbone trees for the simulation experiments, we used the following script to contract the shortest branches of the species trees: `https://gist.github.com/RuneBlaze/8cdf3f493f599c148180fca4b14c85d6`, with the following command:

```
python3 contracting_by_length.py -i $input -v $amount -o $output
```

The `$amount` argument is the portion of branches that should be contracted. For obtaining the T_{25} tree, `$amount` should be set to 0.75.

B.1.7 RF Topological Error

For computing the RF topological error, we used a script written by Erin Molloy. See the FastMulRFS GitHub repository for the scripts: `https://github.com/ekmolloy/fastmulrfs`.

B.1.8 Wall-Clock Running Time

We used the following command for tracking the wall-clock running time:

```
/usr/bin/time -f '%e;%M' -o $output $command
```

B.2 COMPLETE FIGURES FOR THE SIMULATED EXPERIMENTS

We include the complete figures for accuracy and running time for the simulated experiments, showing the omitted backbone = T_{50} condition here:

B.2.1 ASTRAL-II Datasets

B.2.2 Avian Simulated Datasets

B.3 CONSTRAINT TREE FOR THE BIOLOGICAL EXPERIMENT

Here (in Figure B.5) we show the constraint (backbone) tree encoding the "Magnificent Seven" [170] used for the biological experiment.

B.4 DATASET AVAILABILITY

Our data for the MC6H and MC11H datasets are available in the Illinois Data Bank at `https://databank.illinois.edu/datasets/IDB-2566000`.

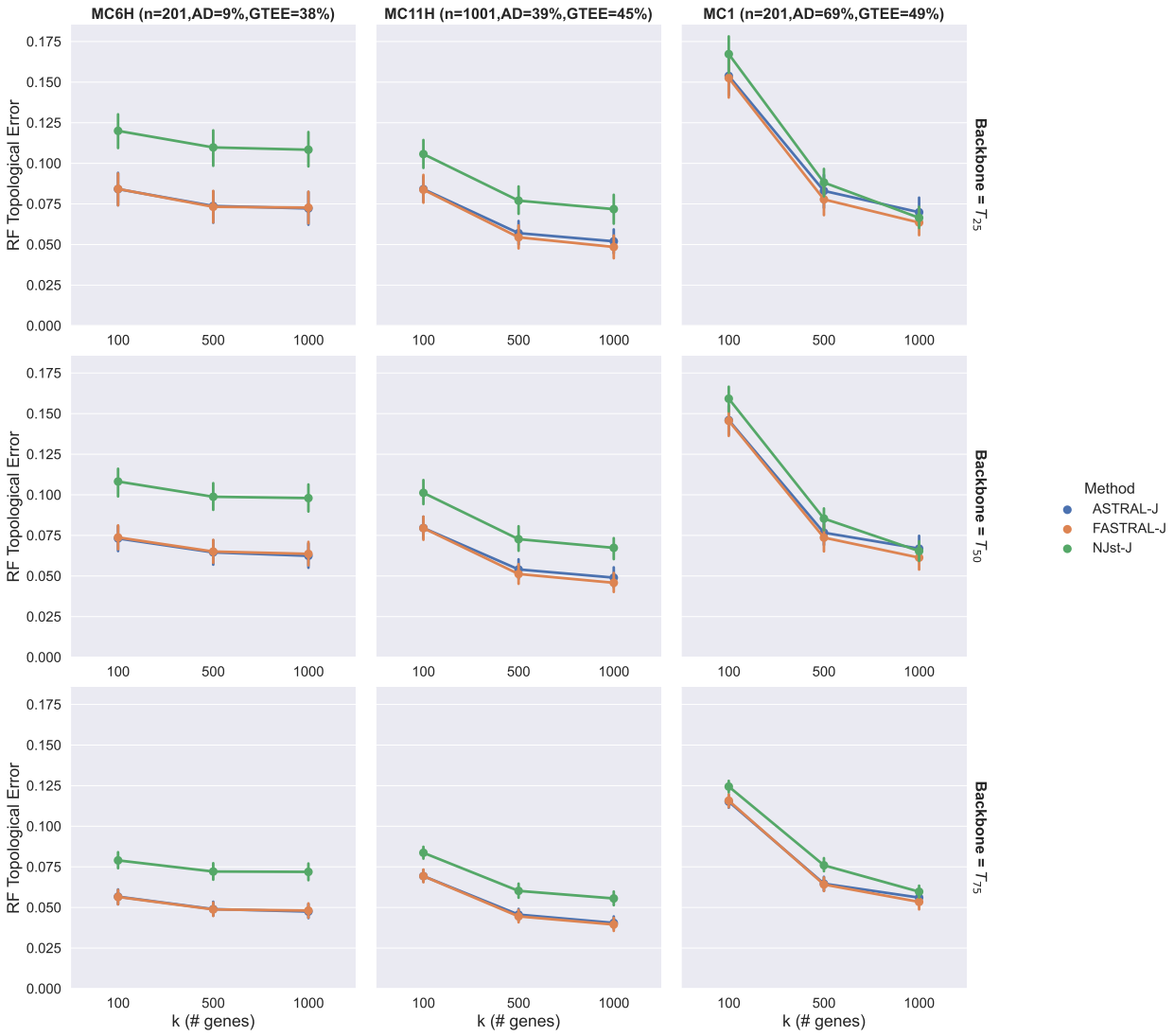


Figure B.1: The accuracy of ASTRAL-J, FASTRAL-J, and NJst-J under the MC6H, MC11H, MC1 datasets when the shortest branches are contracted. T_z denotes the true species tree contracting all branches except the longest z% branches. The x-axis varies in k (number of genes). The RF topological error shown is the mean across 50 replicates under MC6H and MC1. Under MC11H, when backbone is T_{25}, ASTRAL-J timed out on 16 replicates, and thus for MC11H the average on the completed 34 replicates is shown for all the methods. The error bars indicate standard error.

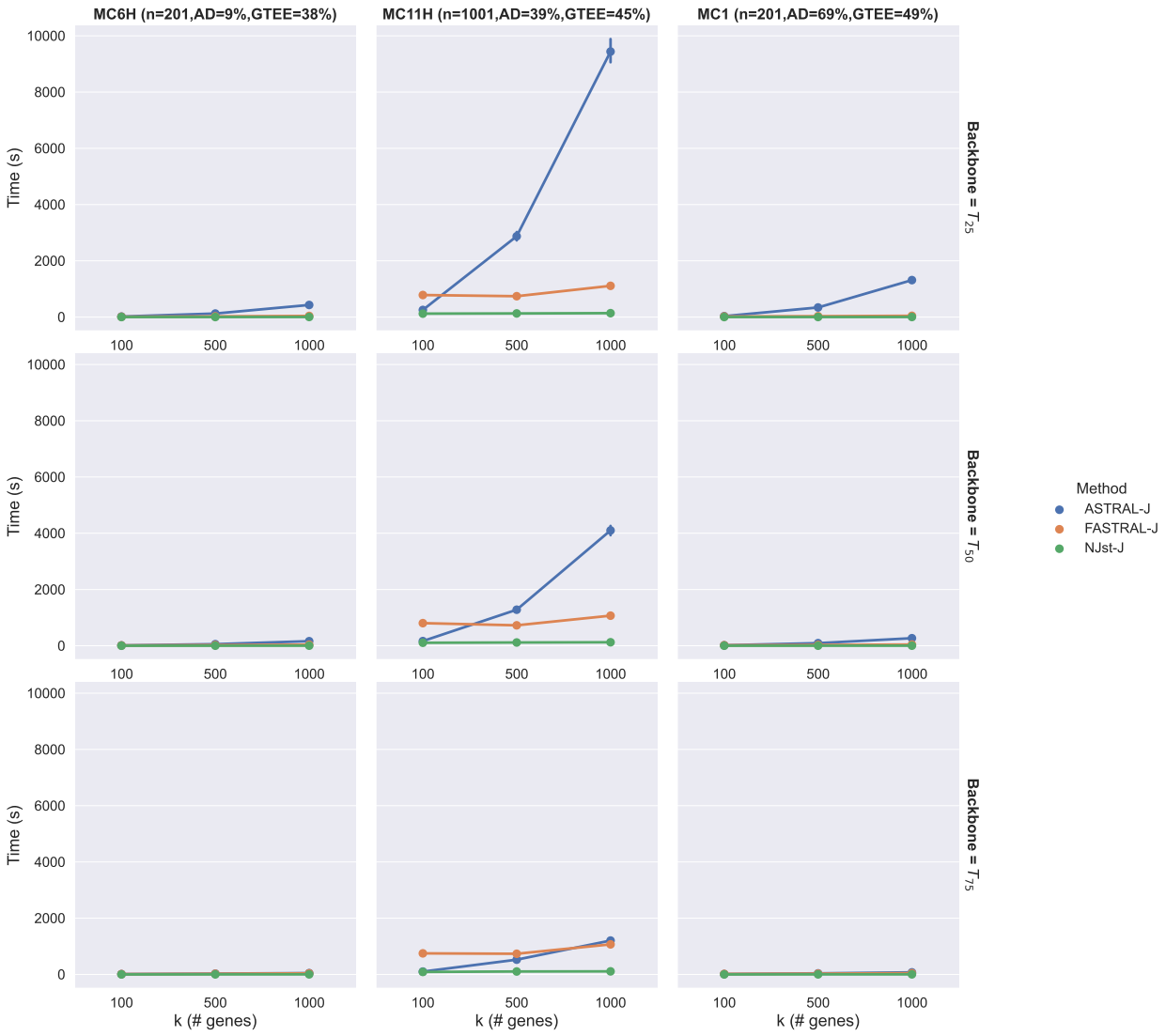


Figure B.2: The wall-clock running time of ASTRAL-J, FASTRAL-J, and NJst-J under the MC6, MC11, and MC1 datasets when the shortest branches are contracted. T_z denotes the true species tree contracting all branches except the longest $z\%$ branches. The x-axis varies in k (number of genes). The running time shown is the mean across 50 replicates under MC6H and MC1. Under MC11H, when backbone is T_{25}, ASTRAL-J timed out on 16 replicates, and thus for MC11H the average on the completed 34 replicates is shown for all the methods. The error bars indicate standard error.

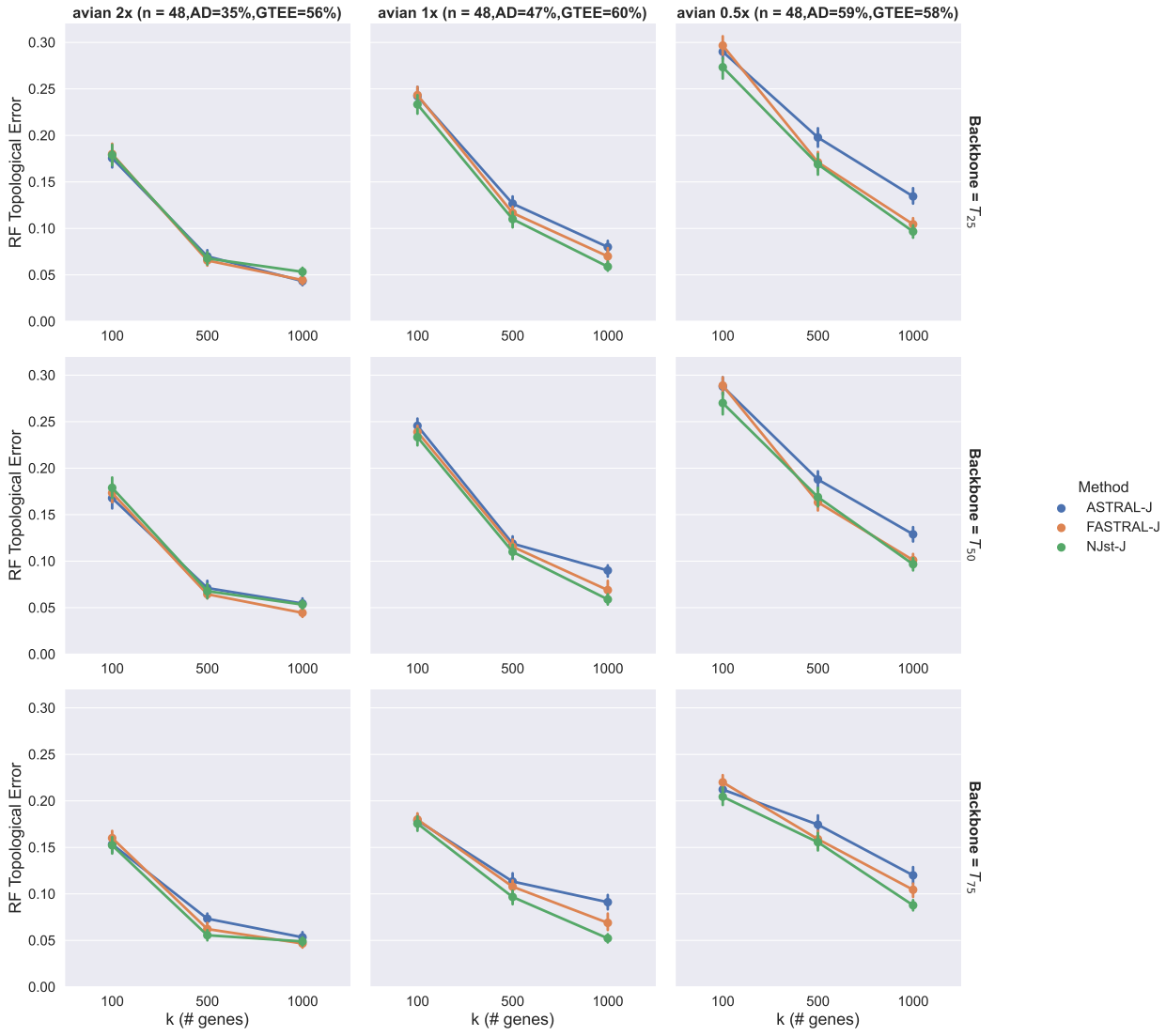


Figure B.3: The accuracy of ASTRAL-J, FASTRAL-J, and NJst-J under the avian simulated dataset when the shortest branches are contracted. T_z denotes the true species tree contracting all branches except the longest $z\%$ branches. The x-axis varies in k (number of genes). The RF topological error shown is the mean across 20 replicates. The error bars indicate standard error.

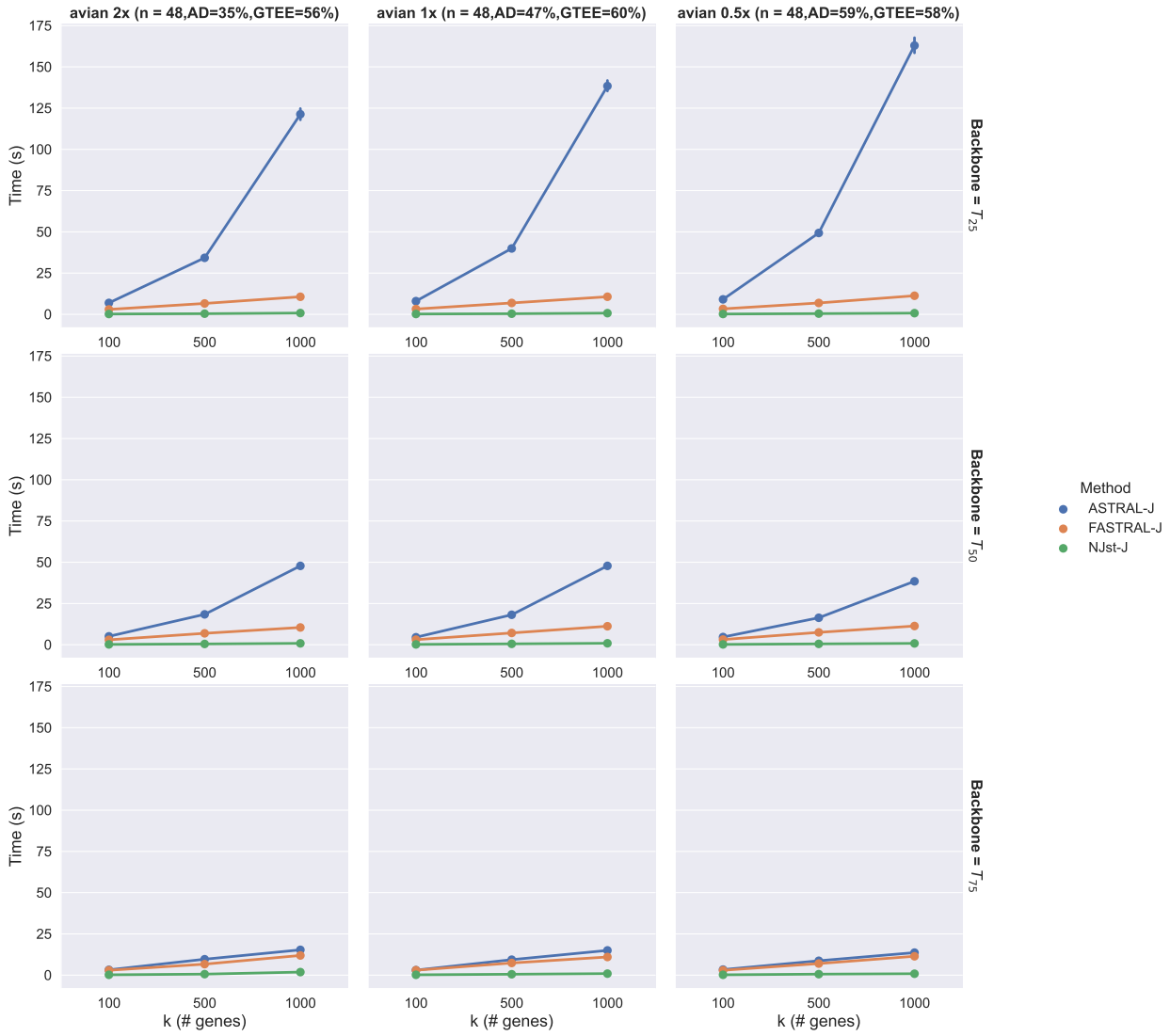


Figure B.4: The wall-clock running time of ASTRAL-J, FASTRAL-J, and NJst-J under the avian simulated dataset when the shortest branches are contracted. T_z denotes the true species tree contracting all branches except the longest $z\%$ branches. The x-axis varies in k (number of genes). The running time shown is the mean across 20 replicates. The error bars indicate standard error. All methods completed on all replicates successfully.

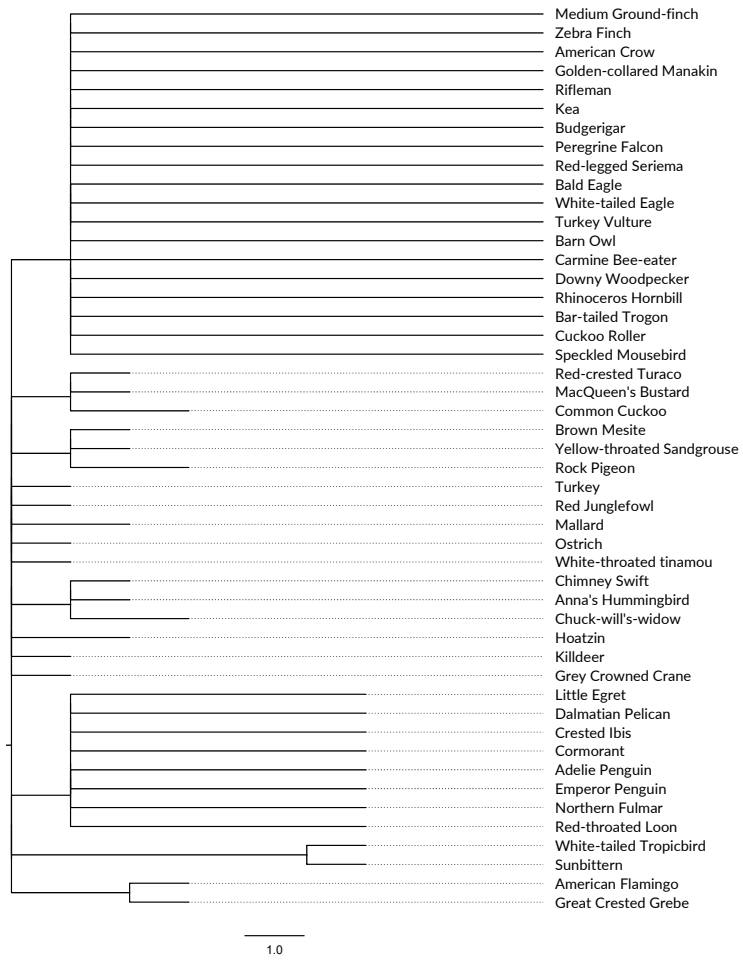


Figure B.5: The constraint tree encoding the "Magnificent Seven" clades from [170]. This tree is used with ASTRAL-J, FASTRAL-J, and NJst-J on the biological experiment.

Appendix C: Additional Results and Materials for WITCH-NG

C.1 SOFTWARE VERSIONS AND COMMANDS

C.1.1 Methods for Adding Query Sequences to Alignments

These methods assume the existence of the following three files:

- `$bb_aln`, (path to) the backbone alignment
- `$bb_tre`, the backbone tree
- `$q`, the query sequences

In addition, WITCH and UPP require specifying the type of "molecule" in the input, either "`dna`", "`rna`", or "`amino`". Let `$molecule` be this variable. Both WITCH and UPP require a working directory, which we denote by `$work_dir`. The number 16 that occurs across the commands is the number of threads we used.

We ran UPP version 4.5.2, with the following command:

```
python3 run_upp.py -m $molecule -x 16 -s $q -a $bb_aln -t $bb_tre -d
   $work_dir -o $output_suffix
```

We ran WITCH version 0.2.1, with the following command:

```
python3 witch.py --molecule $molecule -t 16 -q $q -b $bb_aln -e $bb_tre -d
    $work_dir
```

We ran WITCH-NG version 0.0.2, with the following command:

```
witch-ng add --threads 16 -i $q -b $bb_aln -t $bb_tre -o $output_path
```

C.1.2 HMMER Commands

We invoke HMMER (version 3.1b2 to keep it consistent with WITCH) inside WITCH-NG as subprocesses. The commands are given below:

```
hmmbuild --cpu 0 --informat afa --ere 0.59 --symfrac 0.0 $hmm_outpath -
hmmsearch --cpu 0 --noali --max -E 999999999 $hmm_path -
hmmalign --informat fasta --outformat afa $hmm_path -
```

C.1.3 Other Commands

For computing the alignment error statistics, we used FastSP [180] (v1.7.1) with the following command:

```
java -jar FastSP.jar -ml -e $est -r $ref
```

For the "WITCH(Smith-Waterman)" method, we used a custom version of WITCH, located at [218], with the exact same commands used for WITCH listed above.

For aligning the backbone alignment and to infer trees on the backbone alignment, we either directly ran WITCH (which will as a first step split the dataset and align the backbone alignment, then inferring a tree on the backbone alignment) terminating it once the backbone alignment and tree is produced, or we manually align the backbone sequences and infer trees on the backbone alignment using the same methods (MAGUS and FastTree).

For running WITCH to separate the backbone and queries, and to produce the backbone alignment and tree, we used:

```
python3 witch.py -t 16 -i $unaln_seqs -d $work_dir
```

For running MAGUS (0.1.0b2), we used:

```
python3 magus.py -np 16 -i $unaln_seqs -d $work_dir -o $output_aln_path
```

For running FastTree (v2.1.11 SSE3), we used:

```
FastTreeMP -lg $bb_aln > $bb_tre # for AA data
FastTreeMP -nt -gtr $bb_aln > $bb_tre # for NT data
```

C.2 DATASET AVAILABILITY

We list the sources of our datasets here, with links to the download pages.

- 1000M-HF: [219]
- RNASim-500bp, originally generated in [220] and preprocessed by [39]: [221]
- CRW [35]
  - 5S series: [222]
  - 16S series: [223]
- 10AA (based on [178], [179]): [221]
- HomFam: [224]

Backbone Size	Query Size	Method	SPFN	SPFP	Avg. Error
500	1000	WITCH	**0.100**	**0.097**	**0.099**
		WITCH-NG	**0.100**	**0.097**	**0.099**
		UPP	0.102	**0.097**	**0.099**
	2000	WITCH	**0.099**	**0.094**	**0.097**
		WITCH-NG	**0.099**	**0.095**	**0.097**
		UPP	0.102	**0.094**	**0.098**
	5000	WITCH	**0.099**	**0.093**	**0.096**
		WITCH-NG	**0.098**	**0.093**	**0.096**
		UPP	0.102	**0.093**	0.097
1000	1000	WITCH	**0.096**	**0.093**	**0.095**
		WITCH-NG	**0.096**	**0.093**	**0.095**
		UPP	**0.097**	**0.093**	**0.095**
	2000	WITCH	**0.096**	**0.091**	**0.094**
		WITCH-NG	**0.096**	**0.092**	**0.094**
		UPP	0.098	0.091	**0.094**
	5000	WITCH	**0.096**	**0.090**	**0.093**
		WITCH-NG	**0.095**	**0.090**	**0.093**
		UPP	0.098	**0.090**	**0.094**
2000	1000	WITCH	**0.093**	**0.088**	**0.091**
		WITCH-NG	**0.093**	**0.088**	**0.091**
		UPP	**0.094**	**0.088**	**0.091**
	2000	WITCH	**0.093**	**0.088**	**0.090**
		WITCH-NG	**0.093**	**0.088**	**0.090**
		UPP	**0.094**	**0.087**	**0.091**
	5000	WITCH-NG	**0.092**	**0.086**	**0.089**
		WITCH	**0.092**	**0.086**	**0.089**
		UPP	0.094	**0.086**	0.090

Table C.1: Alignment error rates on RNASim-500bp, averaged over five replicates. WITCH timed out on 1/5 replicates on the 200 backbone 5000 query configuration, hence for that configuration all methods are averaged across four replicates. SPFN (sum-of-pairs false negative) refers to the proportion of pairwise homologies in the reference alignment missing in the estimated alignment, SPFP (sum-of-pairs false positive) refers to the proportion of homologies in the estimated alignment that are not in the reference alignment, and "Avg. Error" is the average of these two values. Best values (with ties within 0.001) are boldfaced.

Dataset	Method	SPFN	SPFP	Avg. Error
PDZ	WITCH	**0.211**	**0.096**	**0.154**
	WITCH-NG	**0.211**	**0.096**	**0.154**
	UPP	0.239	0.107	0.173
blmb	WITCH	**0.300**	0.139	0.220
	WITCH-NG	**0.300**	0.142	0.221
	UPP	0.310	**0.122**	**0.216**
p450	WITCH	0.279	0.183	**0.231**
	WITCH-NG	**0.277**	0.186	**0.231**
	UPP	0.298	**0.175**	0.237
adh	WITCH	**0.638**	**0.003**	**0.321**
	WITCH-NG	**0.638**	**0.003**	**0.321**
	UPP	0.641	**0.003**	**0.322**
aat	WITCH	**0.182**	**0.106**	**0.144**
	WITCH-NG	**0.182**	**0.106**	**0.144**
	UPP	**0.183**	**0.106**	**0.145**
rrm	WITCH	**0.246**	**0.134**	**0.190**
	WITCH-NG	**0.247**	0.136	**0.191**
	UPP	0.253	**0.134**	0.194
Acetyltransf	WITCH	**0.548**	0.167	0.358
	WITCH-NG	**0.548**	0.157	0.352
	UPP	0.550	**0.146**	**0.348**
sdr	WITCH	**0.404**	**0.140**	**0.272**
	WITCH-NG	**0.405**	0.148	0.276
	UPP	0.421	0.147	0.284
zf-CCHH	WITCH	**0.186**	**0.055**	**0.120**
	WITCH-NG	**0.186**	**0.055**	**0.120**
	UPP	**0.186**	**0.056**	**0.121**
rvp	WITCH	**0.273**	**0.218**	**0.246**
	WITCH-NG	**0.273**	**0.218**	**0.246**
	UPP	0.281	0.227	0.254
Average across ten	WITCH	**0.327**	0.124	**0.225**
	WITCH-NG	**0.327**	0.125	**0.226**
	UPP	0.336	**0.122**	0.229

Table C.2: Alignment error rates on the ten largest HomFam datasets. SPFN (sum-of-pairs false negative) refers to the proportion of pairwise homologies in the reference alignment missing in the estimated alignment, SPFP (sum-of-pairs false positive) refers to the proportion of homologies in the estimated alignment that are not in the reference alignment, and "Avg. Error" is the average of these two values. Best values (with ties within 0.001) are boldfaced.

Dataset	Method	Time (s)	SPFN	SPFP	Avg. Error
1GADBL_100	UPP	7.22	**0.033**	**0.034**	**0.034**
	WITCH	9.75	**0.033**	**0.034**	**0.034**
	WITCH-NG	**4.09**	**0.033**	**0.034**	**0.034**
coli_epi_100	UPP	2.95	**0.036**	**0.039**	**0.038**
	WITCH	4.73	**0.036**	**0.039**	**0.038**
	WITCH-NG	**1.29**	**0.036**	**0.039**	**0.038**
RV100_BBA0039	UPP	20.42	**0.071**	0.094	0.083
	WITCH	37.75	0.075	**0.086**	**0.080**
	WITCH-NG	**11.6**	0.075	**0.086**	**0.080**
RV100_BBA0067	UPP	15.37	**0.220**	**0.276**	**0.248**
	WITCH	40.46	**0.226**	0.274	0.250
	WITCH-NG	**12.27**	**0.226**	0.273	0.250
RV100_BBA0081	UPP	**20.59**	**0.494**	0.683	**0.589**
	WITCH	46.71	0.499	**0.681**	0.590
	WITCH-NG	26.36	0.499	**0.681**	0.590
RV100_BBA0101	UPP	21.49	**0.251**	0.324	**0.287**
	WITCH	70.77	0.257	**0.322**	0.290
	WITCH-NG	**20.87**	0.257	**0.322**	0.290
RV100_BBA0117	UPP	4.36	**0.147**	0.155	**0.151**
	WITCH	13.64	0.150	**0.151**	**0.151**
	WITCH-NG	**1.15**	**0.151**	0.151	**0.151**
RV100_BBA0134	UPP	**36.46**	**0.167**	0.253	**0.210**
	WITCH	108.54	0.171	**0.253**	0.212
	WITCH-NG	48.96	0.178	**0.255**	0.216
RV100_BBA0154	UPP	8.09	**0.201**	**0.233**	**0.217**
	WITCH	10.23	0.202	**0.233**	**0.217**
	WITCH-NG	**4.37**	0.202	**0.233**	**0.217**
RV100_BBA0190	UPP	22.06	**0.219**	**0.249**	**0.234**
	WITCH	34.37	**0.219**	**0.249**	**0.234**
	WITCH-NG	**16.73**	**0.219**	**0.249**	**0.234**

Table C.3: Alignment error rates and running times on the 10AA datasets. Runtime is restricted to Phases 1 – 4 , and so does not include the time to compute the backbone alignment and tree (which is common to all three methods).

Name	Type	# seqs.	Seq. length	Align length	% gappy	p-dist.	
						avg.	max
1000M1-HF	Sim NT	1000	631.3	3960	84.0	0.694	1.0
1000M2-HF	Sim NT	1000	634.3	3972	83.8	0.683	1.0
1000M3-HF	Sim NT	1000	629.6	2723	76.7	0.660	1.0
1000M4-HF	Sim NT	1000	629.6	2571	75.3	0.495	1.0
RNASim-LF	Sim NT	1500-7000	1025.44	21946	95.3	0.408	1.0

Table C.4: Expanded dataset statistics for simulated datasets.

Name	Type	Align length	# seqs.	Avg. p-dist
1GADBL_100	Bio AA	490	561	0.4573
coli_epi_100	Bio AA	150	320	0.5829
RV100_BBA0039	Bio AA	2696	807	0.4153
RV100_BBA0067	Bio AA	1092	410	0.7827
RV100_BBA0081	Bio AA	1693	353	0.8631
RV100_BBA0101	Bio AA	4214	509	0.7844
RV100_BBA0117	Bio AA	110	460	0.7537
RV100_BBA0134	Bio AA	3186	717	0.7286
RV100_BBA0154	Bio AA	1275	303	0.6598
RV100_BBA0190	Bio AA	2547	397	0.6884

Table C.5: Dataset properties for the 10AA datasets, focusing on alignment length, sequence count, and average pairwise distance (Avg. p-dist).

Name	Type	Align length	# seqs.	Avg. p-dist
PDZ	Bio AA	3730	14944	0.7552
blmb	Bio AA	13246	17194	0.8008
p450	Bio AA	6848	21001	0.7519
adh	Bio AA	5761	21326	0.7792
aat	Bio AA	12667	25090	0.8090
rrm	Bio AA	3138	27590	0.7604
Acetyltransf	Bio AA	16044	46279	0.8032
sdr	Bio AA	14468	50144	0.7444
zf-CCHH	Bio AA	2287	88330	0.6332
rvp	Bio AA	569	93675	0.1275

Table C.6: Dataset properties of the individual HomFam datasets, focusing on alignment length, sequence count, and average pairwise distance (Avg. p-dist).

Name	Type	# seqs.	Seq. length	Align length	% gappy	p-dist.	
						avg.	max
5S.3	Bio NT	5507	105.6	414	74.5	0.418	1.0
5S.T	Bio NT	5751	106.2	436	75.6	0.425	1.0
16S.3	Bio NT	6323	1557.2	8716	82.1	0.315	0.833
16S.T	Bio NT	7350	1492.1	11856	87.4	0.345	0.901
16S.B.ALL	Bio NT	27643	1371.9	6857	80.0	0.210	0.769

Table C.7: Expanded dataset statistics for CRW datasets, based on the reference alignments from [35].

		Time (min)
Dataset	Method	
1000M1	WITCH	3.4
	WITCH-NG	**0.5**
	UPP	0.7
1000M2	WITCH	3.3
	WITCH-NG	**0.4**
	UPP	0.7
1000M3	WITCH	2.8
	WITCH-NG	**0.4**
	UPP	0.6
1000M4	WITCH	2.8
	WITCH-NG	**0.6**
	UPP	1.1
5S.3	WITCH	5.8
	WITCH-NG	1.2
	UPP	**1.1**
5S.T	WITCH	5.9
	WITCH-NG	1.4
	UPP	**1.3**

Table C.8: Running times for six nucleotide datasets (4 simulated and 2 biological) that had short (less than ten minutes) in running time for Phases 1–4.

Dataset	SPFN	SPFP	Avg. Error
CRW (5)	0.000	0.000	0.000
10AA (10)	0.000	0.000	0.000
HomFam (10)	0.000	0.001	0.001

Table C.9: Average SPFN and SPFP rates comparing WITCH and WITCH-NG alignments on the biological datasets, treating WITCH as the reference alignments with WITCH-NG as the estimated alignments.

References

[1] T. Dobzhansky, "Nothing in biology makes sense except in the light of evolution," *The American Biology Teacher*, vol. 75, no. 2, pp. 87–91, 2013.

[2] E. Pennisi, "Modernizing the Tree of Life," *Science*, vol. 300, no. 5626, pp. 1692–1697, 2003. DOI: 10.1126/science.300.5626.1692.

[3] C. R. Woese, "On the evolution of cells," *Proceedings of the National Academy of Sciences*, vol. 99, no. 13, pp. 8742–8747, 2002. DOI: 10.1073/pnas.132266999.

[4] S. Mirarab, "Species Tree Estimation Using ASTRAL: Practical Considerations," in *Species Tree Inference: A Guide to Methods and Applications*, L. Knowles and L. Kubatko, Eds., Princeton, NJ: Princeton University Press, 2019, pp. 68–88.

[5] P. Vachaspati and T. Warnow, "ASTRID: Accurate Species TRees from Internode Distances," *BMC Genomics*, vol. 16, no. 10, S3, Oct. 2, 2015, ISSN: 1471-2164. DOI: 10.1186/1471-2164-16-S10-S3.

[6] E. L. Braun, C. H. Oliveros, N. D. White Carreiro, *et al.*, "Testing the mettle of METAL: A comparison of phylogenomic methods using a challenging but well-resolved phylogeny," *bioRxiv*, pp. 2024–02, 2024.

[7] C. Shen, M. Park, and T. Warnow, "WITCH: Improved multiple sequence alignment through weighted consensus hidden markov model alignment," *Journal of Computational Biology*, vol. 29, no. 8, pp. 782–801, 2022.

[8] B. Liu and T. Warnow, "WITCH-NG: Efficient and Accurate Alignment of Datasets with Sequence Length Heterogeneity," *Bioinformatics Advances*, vol. 3, no. 1, vbad024, Mar. 2023, ISSN: 2635-0041. DOI: 10.1093/bioadv/vbad024.

[9] R. J. Campello, D. Moulavi, and J. Sander, "Density-based clustering based on hierarchical density estimates," in *Advances in Knowledge Discovery and Data Mining: 17th Pacific-Asia Conference, PAKDD 2013, Gold Coast, Australia, April 14-17, 2013, Proceedings, Part II 17*, Springer, 2013, pp. 160–172.

[10] 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.

[11] J. Achiam, S. Adler, S. Agarwal, *et al.*, "GPT-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[12] M. Reid, N. Savinov, D. Teplyashin, *et al.*, "Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context," *arXiv preprint arXiv:2403.05530*, 2024.

[13] A. Q. Jiang, A. Sablayrolles, A. Mensch, *et al.*, "Mistral 7B," *arXiv preprint arXiv:2310.06825*, 2023.

[14] S. Ovchinnikov, H. Park, N. Varghese, *et al.*, "Protein structure determination using metagenome sequence data," *Science*, vol. 355, no. 6322, pp. 294–298, 2017.

[15] J. Jumper, R. Evans, A. Pritzel, *et al.*, "Highly accurate protein structure prediction with AlphaFold," *Nature*, vol. 596, no. 7873, pp. 583–589, 2021.

[16] P. Kapli, Z. Yang, and M. J. Telford, "Phylogenetic tree building in the genomic age," *Nature Reviews Genetics*, vol. 21, no. 7, pp. 428–444, 2020.

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

[18] D. A. Morrison, "Multiple sequence alignment for phylogenetic purposes," *Australian Systematic Botany*, vol. 19, no. 6, pp. 479–539, 2006.

[19] H. Carrillo and D. Lipman, "The multiple sequence alignment problem in biology," *SIAM Journal on Applied Mathematics*, vol. 48, no. 5, pp. 1073–1082, 1988.

[20] J. D. Thompson, F. Plewniak, and O. Poch, "A comprehensive comparison of multiple sequence alignment programs," *Nucleic Acids Research*, vol. 27, no. 13, pp. 2682–2690, 1999.

[21] J. L. Thorne, H. Kishino, and J. Felsenstein, "An evolutionary model for maximum likelihood alignment of dna sequences," *Journal of Molecular Evolution*, vol. 33, pp. 114–124, 1991.

[22] I. Holmes and W. J. Bruno, "Evolutionary HMMs: A Bayesian approach to multiple alignment," *Bioinformatics*, vol. 17, no. 9, pp. 803–820, 2001.

[23] J. Kececioglu, "The maximum weight trace problem in multiple sequence alignment," in *Annual Symposium on Combinatorial Pattern Matching*, Springer, 1993, pp. 106–119.

[24] S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *Journal of Molecular Biology*, vol. 48, no. 3, pp. 443–453, 1970.

[25] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences," *Journal of Molecular Biology*, vol. 147, no. 1, pp. 195–197, 1981.

[26] W. Just, "Computational complexity of multiple sequence alignment with SP-score," *Journal of Computational Biology*, vol. 8, no. 6, pp. 615–623, 2001.

[27] R. Durbin, S. R. Eddy, A. Krogh, and G. Mitchison, *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge University Press, 1998.

[28] R. D. Finn, J. Clements, and S. R. Eddy, "HMMER web server: Interactive sequence similarity searching," *Nucleic Acids Research*, vol. 39, no. Web Server issue, W29–W37, Jul. 2011, ISSN: 0305-1048. DOI: 10.1093/nar/gkr367.

[29] J. Mistry, S. Chuguransky, L. Williams, *et al.*, "Pfam: The protein families database in 2021," *Nucleic acids research*, vol. 49, no. D1, pp. D412–D419, 2021.

[30] A. Löytynoja, A. J. Vilella, and N. Goldman, "Accurate extension of multiple sequence alignments using a phylogeny-aware graph algorithm," *Bioinformatics*, vol. 28, no. 13, pp. 1684–1691, 2012.

[31] S. A. Berger and A. Stamatakis, "Aligning short reads to reference alignments and trees," *Bioinformatics*, vol. 27, no. 15, pp. 2068–2075, 2011.

[32] S. A. Berger, D. Krompass, and A. Stamatakis, "Performance, accuracy, and web server for evolutionary placement of short sequence reads under maximum likelihood," *Systematic biology*, vol. 60, no. 3, pp. 291–302, 2011.

[33] S. R. Eddy, "Accelerated Profile HMM searches," *PLoS Computational Biology*, vol. 7, no. 10, e1002195, 2011.

[34] F. Sievers, A. Wilm, D. Dineen, *et al.*, "Fast, scalable generation of high-quality protein multiple sequence alignments using Clustal Omega," *Molecular Systems Biology*, vol. 7, p. 539, Oct. 2011, ISSN: 1744-4292. DOI: 10.1038/msb.2011.75.

[35] J. J. Cannone, S. Subramanian, M. N. Schnare, *et al.*, "The Comparative RNA Web (CRW) Site: An online database of comparative sequence and structure information for ribosomal, intron, and other RNAs," *BMC Bioinformatics*, vol. 3, no. 1, p. 2, Jan. 2002, ISSN: 1471-2105. DOI: 10.1186/1471-2105-3-2.

[36] I. Holmes and R. Durbin, "Dynamic programming alignment accuracy," in *Proceedings of the second annual international conference on Computational molecular biology*, 1998, pp. 102–108.

[37] V. Smirnov and T. Warnow, "Phylogeny Estimation Given Sequence Length Heterogeneity," *Systematic Biology*, vol. 70, no. 2, pp. 268–282, Mar. 2021, ISSN: 1063-5157. DOI: 10.1093/sysbio/syaa058.

[38] 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, Apr. 2013, ISSN: 0737-4038. DOI: 10.1093/molbev/mst010.

[39] N.-p. Nguyen, S. Mirarab, K. Kumar, and T. Warnow, "Ultra-large alignments using phylogeny-aware profiles," *Genome Biology*, vol. 16, no. 1, p. 124, Jun. 2015, ISSN: 1465-6906. DOI: 10.1186/s13059-015-0688-z.

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

[41] M. Park, S. Ivanovic, G. Chu, C. Shen, and T. Warnow, "UPP2: fast and accurate alignment of datasets with fragmentary sequences," *Bioinformatics*, vol. 39, 1 2023, https://doi.org/10.1093/bioinformatics/btad007.

[42] C. Shen, P. Zaharias, and T. Warnow, "MAGUS+ eHMMs: Improved multiple sequence alignment accuracy for fragmentary sequences," *Bioinformatics*, vol. 38, no. 4, pp. 918–924, 2022.

[43] C. Shen, M. Park, and T. Warnow, "WITCH: Improved Multiple Sequence Alignment Through Weighted Consensus Hidden Markov Model Alignment," *Journal of Computational Biology*, vol. 29, no. 8, pp. 782–801, Aug. 2022, ISSN: 1557-8666. DOI: 10.1089/cmb.2021.0585.

[44] M. Park and T. Warnow, "HMMerge: an ensemble method for improving multiple sequence alignment," *bioRxiv*, 2022.

[45] C. Shen, B. Liu, K. P. Williams, and T. Warnow, "EMMA: A new method for computing multiple sequence alignments given a constraint subset alignment," *Algorithms for Molecular Biology*, vol. 18, no. 1, p. 21, 2023.

[46] S. A. Berger and A. Stamatakis, "PaPaRa 2.0: a vectorized algorithm for probabilistic phylogeny-aware alignment extension," *Heidelberg Institute for Theoretical Studies*, vol. 12, 2012.

[47] J. Prado-Martinez, P. H. Sudmant, J. M. Kidd, *et al.*, "Great ape genetic diversity and population history," *Nature*, vol. 499, no. 7459, pp. 471–475, 2013.

[48] P. Vachaspati, "Large scale phylogenomic estimation," 2019, PhD dissertation, University of Illinois Urbana-Champaign.

[49] J. Felsenstein, "Evolutionary trees from dna sequences: A maximum likelihood approach," *Journal of Molecular Evolution*, vol. 17, pp. 368–376, 1981.

[50] T. Warnow, *Computational phylogenetics: an introduction to designing methods for phylogeny estimation*. Cambridge University Press, 2017.

[51] S. Mirarab, R. Reaz, M. S. Bayzid, T. Zimmermann, M. S. Swenson, and T. Warnow, "ASTRAL: Genome-scale coalescent-based species tree estimation," *Bioinformatics*, vol. 30, no. 17, pp. i541–i548, Sep. 2014, ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btu462.

[52] D. F. Robinson and L. R. Foulds, "Comparison of phylogenetic trees," *Mathematical Biosciences*, vol. 53, no. 1-2, pp. 131–147, 1981.

[53] F. Lemoine, J.-B. Domelevo Entfellner, E. Wilkinson, *et al.*, "Renewing felsenstein's phylogenetic bootstrap in the era of big data," *Nature*, vol. 556, no. 7702, pp. 452–456, 2018.

[54] M. R. Smith, "Robust analysis of phylogenetic tree space," *Systematic Biology*, vol. 71, no. 5, pp. 1255–1270, 2022.

[55] N. Takahata, "Gene genealogy in three related populations: Consistency probability between gene and population trees.," *Genetics*, vol. 122, no. 4, pp. 957–966, Aug. 1989, ISSN: 1943-2631. DOI: 10.1093/genetics/122.4.957.

[56] W. P. Maddison, "Gene Trees in Species Trees," *Systematic Biology*, vol. 46, no. 3, pp. 523–536, Sep. 1997, ISSN: 1063-5157. DOI: 10.1093/sysbio/46.3.523.

[57] J. F. C. Kingman, "The coalescent," *Stochastic Processes and their Applications*, vol. 13, no. 3, pp. 235–248, Sep. 1982, ISSN: 0304-4149. DOI: 10.1016/0304-4149(82)90011-4.

[58] J. H. Degnan, "Anomalous unrooted gene trees," *Systematic Biology*, vol. 62, no. 4, pp. 574–590, 2013.

[59] S. Mirarab, M. S. Bayzid, and T. Warnow, "Evaluating summary methods for multilocus species tree estimation in the presence of incomplete lineage sorting," *Systematic Biology*, vol. 65, no. 3, pp. 366–380, 2016.

[60] E. K. Molloy and T. Warnow, "To Include or Not to Include: The Impact of Gene Filtering on Species Tree Estimation Methods," *Systematic Biology*, vol. 67, no. 2, pp. 285–303, Mar. 2018, ISSN: 1063-5157. DOI: 10.1093/sysbio/syx077.

[61] D. Bryant and M. W. Hahn, "The concatenation question," in *Phylogenetics in the Genomic Era*, C. Scornavacca, F. Delsuc, and N. Galtier, Eds., No commercial publisher — Authors open access book, 2020, ch. 3.4, 3.4:1–3.4:23.

[62] S. Roch and M. Steel, "Likelihood-based tree reconstruction on a concatenation of aligned sequence data sets can be statistically inconsistent," *Theoretical Population Biology*, vol. 100, pp. 56–62, Mar. 2015, ISSN: 0040-5809. DOI: 10.1016/j.tpb.2014.12.005.

[63] S. Roch, M. Nute, and T. Warnow, "Long-branch attraction in species tree estimation: Inconsistency of partitioned likelihood and topology-based summary methods," *Systematic Biology*, vol. 68, no. 2, pp. 281–297, 2019.

[64] 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.

[65] A. Stamatakis, "RAxML version 8: A tool for phylogenetic analysis and post-analysis of large phylogenies," *Bioinformatics*, vol. 30, no. 9, pp. 1312–1313, May 2014, ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btu033.

[66] A. M. Kozlov, D. Darriba, T. Flouri, B. Morel, and A. Stamatakis, "RAxML-NG: a fast, scalable and user-friendly tool for maximum likelihood phylogenetic inference," *Bioinformatics*, vol. 35, no. 21, pp. 4453–4455, May 2019, ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btz305.

[67] B. Q. Minh, H. A. Schmidt, O. Chernomor, *et al.*, "IQ-TREE 2: new models and efficient methods for phylogenetic inference in the genomic era," *Molecular Biology and Evolution*, vol. 37, no. 5, pp. 1530–1534, 2020.

[68] E. D. Jarvis, S. Mirarab, A. J. Aberer, *et al.*, "Whole-genome analyses resolve early branches in the tree of life of modern birds," *Science*, vol. 346, no. 6215, pp. 1320–1331, 2014.

[69] S. Mirarab, M. S. Bayzid, B. Boussau, and T. Warnow, "Statistical binning enables an accurate coalescent-based estimation of the avian tree," *Science*, vol. 346, no. 6215, p. 1 250 463, Dec. 2014. DOI: 10.1126/science.1250463.

[70] L. Liu, L. Yu, and S. V. Edwards, "A maximum pseudo-likelihood approach for estimating species trees under the coalescent model," *BMC Evolutionary Biology*, vol. 10, no. 1, p. 302, Oct. 2010, ISSN: 1471-2148. DOI: 10.1186/1471-2148-10-302.

[71] S. Mirarab and T. Warnow, "ASTRAL-II: Coalescent-based species tree estimation with many hundreds of taxa and thousands of genes," *Bioinformatics*, vol. 31, no. 12, pp. i44–i52, Jun. 15, 2015, ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btv234.

[72] C. Zhang and S. Mirarab, "ASTRAL-Pro 2: Ultrafast species tree reconstruction from multi-copy gene family trees," *Bioinformatics*, 2022.

[73] M. Lafond and C. Scornavacca, "On the weighted quartet consensus problem," *Theoretical Computer Science*, vol. 769, pp. 1–17, 2019.

[74] D. E. Chavez, I. Gronau, T. Hains, S. Kliver, K.-P. Koepfli, and R. K. Wayne, "Comparative genomics provides new insights into the remarkable adaptations of the african wild dog (lycaon pictus)," *Scientific Reports*, vol. 9, no. 1, p. 8329, 2019.

[75] Q. Zhu, U. Mai, W. Pfeiffer, *et al.*, "Phylogenomics of 10,575 genomes reveals evolutionary proximity between domains Bacteria and Archaea," *Nature Communications*, vol. 10, no. 1, p. 5477, Dec. 2019, ISSN: 2041-1723. DOI: 10.1038/s41467-019-13443-4.

[76] J. Stiller, S. Feng, A.-A. Chowdhury, *et al.*, "Complexity of avian evolution revealed by family-level genomes," *Nature*, pp. 1–3, 2024.

[77] L. Liu and L. Yu, "Estimating Species Trees from Unrooted Gene Trees," *Systematic Biology*, vol. 60, no. 5, pp. 661–667, Oct. 1, 2011, ISSN: 1063-5157. DOI: 10.1093/sysbio/syr027.

[78] N. Saitou and M. Nei, "The neighbor-joining method: A new method for reconstructing phylogenetic trees.," *Molecular Biology and Evolution*, vol. 4, no. 4, pp. 406–425, 1987.

[79] E. S. Allman, J. H. Degnan, and J. A. Rhodes, "Species tree inference from gene splits by unrooted STAR methods," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 15, no. 1, pp. 337–342, 2018. DOI: 10.1109/TCBB.2016.2604812.

[80] K. Atteson, "The performance of neighbor-joining methods of phylogenetic reconstruction," *Algorithmica*, vol. 25, no. 2, pp. 251–278, 1999.

[81] M. Bordewich, O. Gascuel, K. T. Huber, and V. Moulton, "Consistency of topological moves based on the balanced minimum evolution principle of phylogenetic inference," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 6, no. 1, pp. 110–117, 2008.

[82] M. Bordewich and R. Mihaescu, "Accuracy guarantees for phylogeny reconstruction algorithms based on balanced minimum evolution," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 10, no. 3, pp. 576–583, 2013.

[83] V. Lefort, R. Desper, and O. Gascuel, "FastME 2.0: A Comprehensive, Accurate, and Fast Distance-Based Phylogeny Inference Program," *Molecular Biology and Evolution*, vol. 32, no. 10, pp. 2798–2800, Oct. 2015, ISSN: 0737-4038. DOI: 10.1093/molbev/msv150.

[84] S. Fiorini and G. Joret, "Approximating the balanced minimum evolution problem," *Operations Research Letters*, vol. 40, no. 1, pp. 31–35, 2012.

[85] C. Zhang and S. Mirarab, "Weighting by Gene Tree Uncertainty Improves Accuracy of Quartet-based Species Trees," *Molecular Biology and Evolution*, vol. 39, no. 12, Oct. 2022, msac215, ISSN: 1537-1719. DOI: 10.1093/molbev/msac215.

[86] C. Zhang and S. Mirarab, *Accurate Species Tree EstimatoR (ASTER*)*, https://github.com/chaoszhang/ASTER, 2022.

[87] P. Dibaeinia, S. Tabe-Bordbar, and T. Warnow, "FASTRAL: Improving scalability of phylogenomic analysis," *Bioinformatics*, vol. 37, no. 16, pp. 2317–2324, Aug. 2021, ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btab093.

[88] D. Mallo, L. de Oliveira Martins, and D. Posada, "SimPhy: Phylogenomic simulation of gene, locus, and species trees," *Systematic biology*, vol. 65, no. 2, pp. 334–344, 2016.

[89] W. Fletcher and Z. Yang, "INDELible: A flexible simulator of biological sequence evolution," *Molecular Biology and Evolution*, vol. 26, no. 8, pp. 1879–1888, 2009.

[90] J. Felsenstein, "Confidence Limits on Phylogenies: An Approach Using the Bootstrap," *Evolution*, vol. 39, no. 4, pp. 783–791, 1985, ISSN: 1558-5646. DOI: 10.1111/j.1558-5646.1985.tb00420.x.

[91] L. Arvestad, J. Lagergren, and B. Sennblad, "The gene evolution model and computing its associated probabilities," *Journal of the ACM*, vol. 56, no. 2, 7:1–7:44, Apr. 17, 2009, ISSN: 0004-5411. DOI: 10.1145/1502793.1502796.

[92] K. T. Huber and V. Moulton, "Phylogenetic networks from multi-labelled trees," *Journal of Mathematical Biology*, vol. 52, pp. 613–632, 2006.

[93] C. Zhang, C. Scornavacca, E. K. Molloy, *et al.*, "ASTRAL-Pro: Quartet-Based Species-Tree Inference despite Paralogy," *Molecular Biology and Evolution*, vol. 37, no. 11, pp. 3292–3307, Nov. 1, 2020.

[94] J. Willson, M. S. Roddur, B. Liu, P. Zaharias, and T. Warnow, "DISCO: Species Tree Inference using Multicopy Gene Family Tree Decomposition," *Systematic Biology*, vol. 71, no. 3, pp. 610–629, May 2022, ISSN: 1063-5157. DOI: 10.1093/sysbio/syab070.

[95] E. Avni, R. Cohen, and S. Snir, "Weighted quartets phylogenetics," *Systematic Biology*, vol. 64, no. 2, pp. 233–242, 2015.

[96] R. Davidson, P. Vachaspati, S. Mirarab, and T. Warnow, "Phylogenomic species tree estimation in the presence of incomplete lineage sorting and horizontal gene transfer," *BMC Genomics*, vol. 16, no. 10, S1, Oct. 2015, ISSN: 1471-2164. DOI: 10.1186/1471-2164-16-S10-S1.

[97] J. Chifman and L. Kubatko, "Quartet Inference from SNP Data Under the Coalescent Model," *Bioinformatics*, vol. 30, no. 23, pp. 3317–3324, Dec. 2014, ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btu530.

[98] P. Vachaspati and T. Warnow, "SVDquest: Improving SVDquartets species tree estimation using exact optimization within a constrained search space," *Molecular Phylogenetics and Evolution*, vol. 124, pp. 122–136, 2018.

[99] H. A. Ogilvie, R. R. Bouckaert, and A. J. Drummond, "StarBEAST2 Brings Faster Species Tree Inference and Accurate Estimates of Substitution Rates," *Molecular Biology and Evolution*, vol. 34, no. 8, pp. 2101–2114, Aug. 2017, ISSN: 0737-4038. DOI: 10.1093/molbev/msx126.

[100] M. Rabiee and S. Mirarab, "QuCo: Quartet-based co-estimation of species trees and gene trees," *Bioinformatics*, vol. 38, no. Supplement_1, pp. i413–i421, 2022.

[101] M. Girvan and M. E. Newman, "Community structure in social and biological networks," *Proceedings of the National Academy of Sciences*, vol. 99, no. 12, pp. 7821–7826, 2002.

[102] S. Fortunato, "Community detection in graphs," *Physics Reports*, vol. 486, no. 3-5, pp. 75–174, 2010.

[103] S. Fortunato and D. Hric, "Community detection in networks: A user guide," *Physics Reports*, vol. 659, pp. 1–44, 2016.

[104] S. Sobolevsky, R. Campari, A. Belyi, and C. Ratti, "General optimization technique for high-quality community detection in complex networks," *Physical Review E*, vol. 90, no. 1, p. 012 811, 2014.

[105] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2008, no. 10, P10008, 2008.

[106] U. Brandes, D. Delling, M. Gaertler, *et al.*, "On modularity clustering," *IEEE Transactions on Knowledge and Data Engineering*, vol. 20, no. 2, pp. 172–188, 2007.

[107] M. E. Newman, "Detecting community structure in networks," *The European physical journal B*, vol. 38, pp. 321–330, 2004.

[108] S. Fortunato and M. Barthelemy, "Resolution limit in community detection," *Proceedings of the National Academy of Sciences*, vol. 104, no. 1, pp. 36–41, 2007.

[109] V. A. Traag, P. Van Dooren, and Y. Nesterov, "Narrow scope for resolution-limit-free community detection," *Physical Review E*, vol. 84, no. 1, p. 016 114, 2011.

[110] V. A. Traag, L. Waltman, and N. J. Van Eck, "From Louvain to Leiden: Guaranteeing well-connected communities," *Scientific Reports*, vol. 9, no. 1, p. 5233, 2019.

[111] U. N. Raghavan, R. Albert, and S. Kumara, "Near linear time algorithm to detect community structures in large-scale networks," *Physical Review E*, vol. 76, no. 3, p. 036 106, 2007.

[112] S. Guattery and G. L. Miller, "On the performance of spectral graph partitioning methods," in *SODA*, Citeseer, vol. 95, 1995, pp. 233–242.

[113] A. Ng, M. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," *Advances in Neural Information Processing Systems*, vol. 14, 2001.

[114] M. Rosvall and C. T. Bergstrom, "Maps of random walks on complex networks reveal community structure," *Proceedings of the national academy of sciences*, vol. 105, no. 4, pp. 1118–1123, 2008.

[115] B. Rozemberczki, O. Kiss, and R. Sarkar, "Karate Club: An API Oriented Open-source Python Framework for Unsupervised Learning on Graphs," in *Proceedings of the 29th ACM international conference on Information & Knowledge management*, 2020, pp. 3125–3132.

[116] X. Su, S. Xue, F. Liu, *et al.*, "A comprehensive survey on community detection with deep learning," *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[117] P.-Z. Li, L. Huang, C.-D. Wang, and J.-H. Lai, "EdMot: An Edge Enhancement Approach for Motif-aware Community Detection," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019, pp. 479–487.

[118] A. Jakatdar, B. Liu, T. Warnow, and G. Chacko, "AOC: Assembling overlapping communities," *Quantitative Science Studies*, vol. 3, no. 4, pp. 1–28, 2022.

[119] A. Epasto, S. Lattanzi, and R. Paes Leme, "Ego-splitting framework: From non-overlapping to overlapping clusters," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017, pp. 145–154.

[120] J. Yang and J. Leskovec, "Overlapping community detection at scale: A nonnegative matrix factorization approach," in *Proceedings of the sixth ACM International Conference on Web Search and Data Mining*, 2013, pp. 587–596.

[121] J. Xie, B. K. Szymanski, and X. Liu, "SLPA: Uncovering overlapping communities in social networks via a speaker-listener interaction dynamic process," in *2011 IEEE 11th international conference on data mining workshops*, IEEE, 2011, pp. 344–349.

[122] L. McInnes, J. Healy, and S. Astels, "Hdbscan: Hierarchical density based clustering.," *J. Open Source Softw.*, vol. 2, no. 11, p. 205, 2017.

[123] L. Van der Maaten and G. Hinton, "Visualizing data using t-SNE.," *Journal of machine learning research*, vol. 9, no. 11, 2008.

[124] P. Cui, X. Wang, J. Pei, and W. Zhu, "A survey on network embedding," *IEEE transactions on knowledge and data engineering*, vol. 31, no. 5, pp. 833–852, 2018.

[125] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[126] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[127] B. Perozzi, R. Al-Rfou, and S. Skiena, "DeepWalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2014, pp. 701–710.

[128] J. You, J. M. Gomes-Selman, R. Ying, and J. Leskovec, "Identity-aware graph neural networks," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 35, 2021, pp. 10 737–10 745.

[129] S. Cavallari, V. W. Zheng, H. Cai, K. C.-C. Chang, and E. Cambria, "Learning community embedding with community detection and node embedding on graphs," in *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, 2017, pp. 377–386.

[130] O. Shchur and S. Günnemann, "Overlapping community detection with graph neural networks," *arXiv preprint arXiv:1909.12201*, 2019.

[131] K. Chaudhuri and S. Dasgupta, "Rates of convergence for the cluster tree," *Advances in Neural Information Processing Systems*, vol. 23, 2010.

[132] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "Density-based spatial clustering of applications with noise," in *Int. Conf. knowledge discovery and data mining*, vol. 240, 1996.

[133] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander, "OPTICS: Ordering points to identify the clustering structure," *ACM SIGMOD record*, vol. 28, no. 2, pp. 49–60, 1999.

[134] S. Halim, F. Halim, S. S. Skiena, and M. A. Revilla, "Chapter 3: Problem Solving Paradigms," in *Competitive Programming 3*. Citeseer, 2013, ch. 3.

[135] R. J. Lipton, K. W. Regan, R. J. Lipton, and K. W. Regan, "David Johnson: Galactic algorithms," *People, Problems, and Proofs: Essays from Gödel's Lost Letter: 2010*, pp. 109–112, 2013.

[136] 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, pp. 1–10, Mar. 2010. DOI: 10.1371/journal.pone.0009490.

[137] C. Zhang, M. Rabiee, E. Sayyari, and S. Mirarab, *Software of ASTRAL-III: Polynomial time species tree reconstruction from partially resolved gene trees*, https://github.com/smirarab/ASTRAL, 2018.

[138] U. Drepper, "What every programmer should know about memory," *Red Hat, Inc*, 2007.

[139] J. Dean, "Software engineering advice from building large-scale distributed systems," *CS295 Lecture at Stanford University*, vol. 1, no. 2.1, pp. 1–2, 2007.

[140] B. Liu and T. Warnow, "Weighted ASTRID: Fast and accurate species trees from weighted internode distances," *Algorithms for Molecular Biology*, vol. 18, no. 1, p. 6, 2023.

[141] A. Criscuolo and O. Gascuel, "Fast NJ-like algorithms to deal with incomplete distance matrices," *BMC Bioinformatics*, vol. 9, pp. 1–16, 2008.

[142] C. D. Michener and R. R. Sokal, "A quantitative approach to a problem in classification," *Evolution*, vol. 11, no. 2, pp. 130–162, 1957.

[143] P. Vachaspati, *ASTRID2 GitHub site*, https://pranjalv123.github.io/ASTRID/, 2023.

[144] M. Anisimova, M. Gil, J.-F. Dufayard, C. Dessimoz, and O. Gascuel, "Survey of Branch Support Methods Demonstrates Accuracy, Power, and Robustness of Fast Likelihood-based Approximation Schemes," *Systematic Biology*, vol. 60, no. 5, pp. 685–699, Oct. 2011, ISSN: 1063-5157. DOI: 10.1093/sysbio/syr041.

[145] J. J. Wiens, C. A. Kuczynski, S. A. Smith, *et al.*, "Branch lengths, support, and congruence: Testing the phylogenomic approach with 20 nuclear loci in snakes," *Systematic Biology*, vol. 57, no. 3, pp. 420–431, 2008.

[146] L. Liu, L. Yu, D. K. Pearl, and S. V. Edwards, "Estimating Species Phylogenies Using Coalescence Times among Sequences," *Systematic Biology*, vol. 58, no. 5, pp. 468–477, Oct. 2009, ISSN: 1063-5157. DOI: 10.1093/sysbio/syp031.

[147] R. Desper and O. Gascuel, "Fast and accurate phylogeny reconstruction algorithms based on the minimum-evolution principle," *Journal of Computational Biology*, vol. 9, no. 5, pp. 687–705, 2002. DOI: 10.1089/106652702761034136.

[148] N. Moshiri, "TreeSwift: A massively scalable python tree package," *SoftwareX*, vol. 11, p. 100 436, 2020.

[149] E. D. Jarvis, S. Mirarab, A. J. Aberer, *et al.*, "Whole-genome analyses resolve early branches in the tree of life of modern birds," *Science*, vol. 346, no. 6215, pp. 1320–1331, Dec. 2014. DOI: 10.1126/science.1253451.

[150] M. Nute, J. Chou, E. K. Molloy, *et al.*, "The performance of coalescent-based species tree estimation methods under models of missing data," *BMC Genomics*, vol. 19, no. 5, p. 286, 2018.

[151] C. Zhang, M. Rabiee, E. Sayyari, and S. Mirarab, "ASTRAL-III: Polynomial time species tree reconstruction from partially resolved gene trees," *BMC Bioinformatics*, vol. 19, no. 6, pp. 15–30, 2018.

[152] B. Liu and T. Warnow, *Data from Scalable Species Tree Inference with External Constraints*, University of Illinois at Urbana-Champaign, 2021. DOI: 10.13012/B2IDB-2566000_V1.

[153] B. Liu, *Internode: Fast implementation of ASTRID-like methods*, https://github.com/RuneBlaze/internode, 2023.

[154] E. Sayyari and S. Mirarab, "Fast Coalescent-Based Computation of Local Branch Support from Quartet Frequencies," *Molecular Biology and Evolution*, vol. 33, no. 7, pp. 1654–1668, Jul. 2016, ISSN: 0737-4038. DOI: 10.1093/molbev/msw079.

[155] E. L. Braun and R. T. Kimball, "Data types and the phylogeny of neoaves," *Birds*, vol. 2, no. 1, pp. 1–22, 2021, ISSN: 2673-6004. DOI: 10.3390/birds2010001.

[156] M. Rabiee and S. Mirarab, "Forcing external constraints on tree inference using astral," *BMC Genomics*, vol. 21, no. 2, pp. 1–13, 2020.

[157] J. A. Rhodes, M. G. Nute, and T. Warnow, *NJst and ASTRID are not statistically consistent under a random model of missing data*, 2020. arXiv: 2001.07844 [q-bio.PE].

[158] B. Legried, E. K. Molloy, T. Warnow, and S. Roch, "Polynomial-time statistical estimation of species trees under gene duplication and loss," *Journal of Computational Biology*, vol. 28, no. 5, pp. 452–468, 2021.

[159] A. Markin and O. Eulenstein, "Quartet-based inference is statistically consistent under the unified duplication-loss-coalescence model," *Bioinformatics*, vol. 37, no. 22, pp. 4064–4074, 2021.

[160] C. Daskalakis and S. Roch, "Species trees from gene trees despite a high rate of lateral genetic transfer: A tight bound," in *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SIAM, 2016, pp. 1621–1630.

[161] M. Hill, B. Legried, and S. Roch, "Species tree estimation under joint modeling of coalescence and duplication: Sample complexity of quartet methods," *arXiv preprint arXiv:2007.06697*, vol. 32, no. 6, pp. 4681–4705, 2020.

[162] P. Vachaspati and T. Warnow, "FastRFS: fast and accurate Robinson-Foulds Supertrees using constrained exact optimization," *Bioinformatics*, vol. 33, no. 5, pp. 631–639, Sep. 2016, ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btw600.

[163] P. Zaharias, F. Lemoine, and O. Gascuel, "Robustness of Felsenstein's Versus Transfer Bootstrap Supports With Respect to Taxon Sampling," *Systematic Biology*, vol. 72, no. 6, pp. 1280–1295, Aug. 2023, ISSN: 1063-5157. DOI: 10.1093/sysbio/syad052.

[164] B. Liu and T. Warnow, "Scalable Species Tree Inference with External Constraints," *Journal of Computational Biology*, vol. 29, no. 7, pp. 664–678, Jul. 2022. DOI: 10.1089/cmb.2021.0543.

[165] M. Rabiee and S. Mirarab, "Forcing external constraints on tree inference using ASTRAL," *BMC Genomics*, vol. 21, no. 2, p. 218, Apr. 16, 2020.

[166] M. S. Bansal, "Linear-time algorithms for phylogenetic tree completion under Robinson–Foulds distance," *Algorithms for Molecular Biology*, vol. 15, no. 1, pp. 1–15, Dec. 2020.

[167] S. Christensen, E. K. Molloy, P. Vachaspati, *et al.*, "OCTAL: Optimal completion of gene trees in polynomial time," *Algorithms for Molecular Biology*, vol. 13, no. 1, p. 6, 2018.

[168] E. S. Allman, J. H. Degnan, and J. A. Rhodes, "Species tree inference from gene splits by unrooted STAR methods," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 15, no. 1, pp. 337–342, Jan. 2018.

[169] R. O. Prum, J. S. Berv, A. Dornburg, *et al.*, "A comprehensive phylogeny of birds (Aves) using targeted next-generation DNA sequencing," *Nature*, vol. 526, no. 7574, pp. 569–573, Oct. 2015.

[170] S. Reddy, R. T. Kimball, A. Pandey, *et al.*, "Why Do Phylogenomic Data Sets Yield Conflicting Trees? Data Type Influences the Avian Tree of Life more than Taxon Sampling," *Systematic Biology*, vol. 66, no. 5, pp. 857–879, Sep. 2017, ISSN: 1063-5157. DOI: 10.1093/sysbio/syx041.

[171] E. L. Braun and R. T. Kimball, "Data types and the phylogeny of Neoaves," *Birds*, vol. 2, no. 1, pp. 1–22, 2021.

[172] P. Houde, E. L. Braun, N. Narula, *et al.*, "Phylogenetic signal of indels and the Neoavian radiation," *Diversity*, vol. 11, no. 7, p. 108, 2019.

[173] L. S. Kubatko and J. H. Degnan, "Inconsistency of Phylogenetic Estimates from Concatenated Data under Coalescence," *Systematic Biology*, vol. 56, no. 1, pp. 17–24, Feb. 2007, ISSN: 1063-5157. DOI: 10.1080/10635150601146041.

[174] V. Lefort, R. Desper, and O. Gascuel, "FastME 2.0: A comprehensive, accurate, and fast distance-based phylogeny inference program," *Molecular Biology and Evolution*, vol. 32, no. 10, pp. 2798–2800, 2015.

[175] S. V. Dongen, "Graph clustering by flow simulation," *PhD thesis, University of Utrecht*, 2000.

[176] P. Zaharias, V. Smirnov, and T. Warnow, "Large-scale multiple sequence alignment and the maximum weight trace alignment merging problem," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, pp. 1–13, 2022. DOI: 10.1109/TCBB.2022.3191848.

[177] V. Smirnov and T. Warnow, "MAGUS: Multiple sequence Alignment using Graph clUStering," *Bioinformatics*, vol. 37, no. 12, pp. 1666–1672, Jun. 15, 2021, ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btaa992.

[178] J. D. Thompson, B. Linard, O. Lecompte, and O. Poch, "A Comprehensive Benchmark Study of Multiple Sequence Alignment Methods: Current Challenges and Future Perspectives," *PLOS ONE*, vol. 6, no. 3, pp. 1–14, Mar. 2011. DOI: 10.1371/journal.pone.0018093.

[179] G. B. Gloor, L. C. Martin, L. M. Wahl, and S. D. Dunn, "Mutual information in protein multiple sequence alignments reveals two classes of coevolving positions," *Biochemistry*, vol. 44, no. 19, pp. 7156–7165, 2005. DOI: 10.1021/bi050293e.

[180] S. Mirarab and T. Warnow, "FastSP: Linear time calculation of alignment accuracy," *Bioinformatics*, vol. 27, no. 23, pp. 3250–3258, 2011.

[181] C. Shen, P. Zaharias, and T. Warnow, "MAGUS+eHMMs: Improved multiple sequence alignment accuracy for fragmentary sequences," *Bioinformatics*, vol. 38, no. 4, pp. 918–924, Feb. 2022, ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btab788.

[182] M. Farrar, "Striped Smith–Waterman speeds database searches six times over other SIMD implementations," *Bioinformatics*, vol. 23, no. 2, pp. 156–161, 2007.

[183] O. A. Rodriguez, Z. Du, J. Patchett, F. Li, and D. A. Bader, "Arachne: An Arkouda package for large-scale graph analytics," in *2022 IEEE High Performance Extreme Computing Conference (HPEC)*, IEEE, 2022, pp. 1–7.

[184] R. R. Sokal and C. D. Michener, "A statistical method for evaluating systematic relationships," 1958.

[185] A. Grover and J. Leskovec, "Node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 855–864.

[186] C. Wang, C. Wang, Z. Wang, X. Ye, and P. S. Yu, "Edge2vec: Edge-based social network embedding," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 14, no. 4, pp. 1–24, 2020.

[187] M. Ostroski, "Dynamic graph edge clustering: The art of conversation mining," in *Proceedings of the AMS Special Session at the Joint Mathematics Meetings*, Department of Defense, Baltimore, MD, Jan. 9, 2019.

[188] D. Lemire, *How fast can you sort arrays of integers in Java? – Daniel Lemire's blog*, en-US, Apr. 2021. [Online]. Available: https://lemire.me/blog/2021/04/09/how-fast-can-you-sort-arrays-of-integers-in-java/ (visited on 05/09/2024).

[189] Y. Han and M. Thorup, "Integer Sorting in $O(n\sqrt{\log\log n})$ Expected time and Linear Space," in *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.*, IEEE, 2002, pp. 135–144.

[190] T. J. Wheeler, "Large-scale neighbor-joining with NINJA," in *Algorithms in Bioinformatics: 9th International Workshop, WABI 2009, Philadelphia, PA, USA, September 12-13, 2009. Proceedings 9*, Springer, 2009, pp. 375–389.

[191] M. Simonsen, T. Mailund, and C. N. Pedersen, "Inference of large phylogenies using neighbour-joining," in *Biomedical Engineering Systems and Technologies: Third International Joint Conference, BIOSTEC 2010, Valencia, Spain, January 20-23, 2010, Revised Selected Papers 3*, Springer, 2011, pp. 334–344.

[192] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[193] J. Leskovec and A. Krevl, *SNAP Datasets: Stanford large network dataset collection*, `http://snap.stanford.edu/data`, Jun. 2014.

[194] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[195] Y. Wang, Y. Gu, and J. Shun, "Theoretically-efficient and practical parallel DBSCAN," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020, pp. 2555–2571.

[196] G. Wu, L. Cao, H. Tian, and W. Wang, "HY-DBSCAN: A hybrid parallel DBSCAN clustering algorithm scalable on distributed-memory computers," *Journal of Parallel and Distributed Computing*, vol. 168, pp. 57–69, 2022.

[197] Y. Wang, S. Yu, Y. Gu, and J. Shun, "Fast parallel algorithms for euclidean minimum spanning tree and hierarchical spatial clustering," in *Proceedings of the 2021 international conference on management of data*, 2021, pp. 1982–1995.

[198] T. I. Syed, "Parallelization of hierarchical density-based clustering using MapReduce," 2015.

[199] J. A. dos Santos, T. I. Syed, M. C. Naldi, R. J. Campello, and J. Sander, "Hierarchical density-based clustering using MapReduce," *IEEE Transactions on Big Data*, vol. 7, no. 1, pp. 102–114, 2019.

[200] B. R. Larget, S. K. Kotha, C. N. Dewey, and C. Ané, "BUCKy: Gene tree/species tree reconciliation with Bayesian concordance analysis," *Bioinformatics*, vol. 26, no. 22, pp. 2910–2911, 2010.

[201] S. Yourdkhani and J. A. Rhodes, "Inferring metric trees from weighted quartets via an intertaxon distance," *Bulletin of Mathematical Biology*, vol. 82, pp. 1–22, 2020.

[202] B. Morel, T. A. Williams, and A. Stamatakis, "Asteroid: A new algorithm to infer species trees from gene trees under high proportions of missing data," *Bioinformatics*, vol. 39, no. 1, btac832, 2023.

[203] J. Bradbury, R. Frostig, P. Hawkins, *et al.*, *JAX: Composable transformations of Python+NumPy programs*, version 0.3.13, 2018. [Online]. Available: `http://github.com/google/jax`.

[204] C. R. Harris, K. J. Millman, S. J. Van Der Walt, *et al.*, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, 2020.

[205] M. Park and T. Warnow, "HMMerge: An ensemble method for improving multiple sequence alignment," *bioRxiv*, 2022. DOI: `10.1101/2022.05.29.493880`.

[206] 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.

[207] A. Paszke, S. Gross, F. Massa, *et al.*, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[208] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A fresh approach to numerical computing," *SIAM Review*, vol. 59, no. 1, pp. 65–98, 2017.

[209] S. K. Lam, A. Pitrou, and S. Seibert, "Numba: A LLVM-based python JIT compiler," in *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, 2015, pp. 1–6.

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

[211] F. Lemoine and O. Gascuel, "Gotree/Goalign: toolkit and Go API to facilitate the development of phylogenetic workflows," *NAR Genomics and Bioinformatics*, vol. 3, no. 3, Aug. 2021, ISSN: 2631-9268. DOI: `10.1093/nargab/lqab075`.

[212] A. Stamatakis, P. Hoover, and J. Rougemont, "A rapid bootstrap algorithm for the RAxML web servers," *Systematic Biology*, vol. 57, no. 5, pp. 758–771, 2008.

[213] D. T. Hoang, O. Chernomor, A. Von Haeseler, B. Q. Minh, and L. S. Vinh, "UFBoot2: Improving the ultrafast bootstrap approximation," *Molecular Biology and Evolution*, vol. 35, no. 2, pp. 518–522, 2018.

[214] H. Shimodaira and M. Hasegawa, "Multiple comparisons of log-likelihoods with applications to phylogenetic inference," *Molecular Biology and Evolution*, vol. 16, no. 8, p. 1114, 1999.

[215] S. Guindon, F. Delsuc, J.-F. Dufayard, and O. Gascuel, "Estimating maximum likelihood phylogenies with PhyML," *Bioinformatics for DNA sequence analysis*, pp. 113–137, 2009.

[216] B. Rannala and Z. Yang, "Probability distribution of molecular evolutionary trees: A new method of phylogenetic inference," *Journal of Molecular Evolution*, vol. 43, pp. 304–311, 1996.

[217] M. P. Simmons, K. M. Pickett, and M. Miya, "How meaningful are Bayesian support values?" *Molecular Biology and Evolution*, vol. 21, no. 1, pp. 188–199, 2004.

[218] B. Liu, *GitHub site for WITCH-NG using Smith-Waterman*, Available at `https://github.com/RuneBlaze/WITCH`, 2023.

[219] V. Smirnov and T. Warnow, *Datasets for publication "Phylogeny estimation given sequence length heterogeneity"*, `https://doi.org/10.5061/dryad.95x69p8h8`, 2020.

[220] 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, May 2015. DOI: `10.1089/cmb.2014.0156`.

[221] N.-p. Nguyen, S. Mirarab, K. Kumar, and T. Warnow, *Datasets for publication "Ultra-large alignments using phylogeny-aware profiles"*, `https://sites.google.com/eng.ucsd.edu/datasets/alignment/pastaupp`, Jun. 2015.

[222] C. Shen, P. Zaharias, and T. Warnow, *Datasets for publication "MAGUS+eHMMs: Improved multiple sequence alignment accuracy for fragmentary sequences"*, `https://doi.org/10.13012/B2IDB-2419626_V1`, 2021.

[223] N.-p. Nguyen, S. Mirarab, K. Kumar, and T. Warnow, *Preprocessed 16s and 23s datasets*, `https://sites.google.com/eng.ucsd.edu/datasets/alignment/16s23s`, Jun. 2015.

[224] F. Sievers, A. Wilm, D. Dineen, *et al.*, *Homfam Datasets, for publication "Fast, scalable generation of high-quality protein multiple sequence alignments using Clustal Omega"*, `http://www.clustal.org/omega/homfam-20110613-25.tar.gz`, Oct. 2011.