Bioinformatics, 35, 2019, i417–i426
doi: 10.1093/bioinformatics/btz344
ISMB/ECCB 2019

TreeMerge: a new method for improving the scalability of species tree estimation methods

Erin K. Molloy and Tandy Warnow *

Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA

*To whom correspondence should be addressed.

Abstract

Motivation: At RECOMB-CG 2018, we presented NJMerge and showed that it could be used within a divide-and-conquer framework to scale computationally intensive methods for species tree estimation to larger datasets. However, NJMerge has two significant limitations: it can fail to return a tree and, when used within the proposed divide-and-conquer framework, has $O(n^5)$ running time for datasets with n species.

Results: Here we present a new method called 'TreeMerge' that improves on NJMerge in two ways: it is guaranteed to return a tree and it has dramatically faster running time within the same divide-and-conquer framework—only $O(n^2)$ time. We use a simulation study to evaluate TreeMerge in the context of multi-locus species tree estimation with two leading methods, ASTRAL-III and RAxML. We find that the divide-and-conquer framework using TreeMerge has a minor impact on species tree accuracy, dramatically reduces running time, and enables both ASTRAL-III and RAxML to complete on datasets (that they would otherwise fail on), when given 64 GB of memory and 48 h maximum running time. Thus, TreeMerge is a step toward a larger vision of enabling researchers with limited computational resources to perform large-scale species tree estimation, which we call *Phylogenomics for All*.

Availability and implementation: TreeMerge is publicly available on Github (http://github.com/ekmolloy/treemerge).

Contact: warnow@illinois.edu

Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Introduction

Species tree estimation is a basic step in many biological studies and has traditionally been performed by selecting regions from across the genomes for a set of species, constructing multiple sequence alignments on each of these regions, and then estimating a species tree from the concatenated alignment using popular phylogeny estimation methods, such as maximum likelihood (ML) heuristics. While these concatenation analyses are standard in systematic studies, the realization that different parts of the genome can have different evolutionary histories (Maddison, 1997) has spurred the development of new approaches to species tree estimation that can address heterogeneity across the genome. Many biological processes can result in gene trees being different from each other (and from the species phylogeny), including gene duplication and loss (GDL), incomplete lineage sorting (ILS), horizontal gene transfer (HGT) and hybridizing speciation. The presence of hybridization and HGT implies that phylogenetic networks are needed rather than trees, but ILS and GDL are consistent with a species tree. The inference of phylogenetic trees in the presence of GDL is particularly difficult and is related to the problems of orthology prediction and gene tree reconciliation (Bansal and Eulenstein, 2013; Bayzid and Warnow, 2018; Boussau *et al.*, 2013; Lai *et al.*, 2012; Nakhleh, 2013; The Quest for Orthologs Consortium *et al.*, 2014; Tofigh *et al.*, 2011).

ILS, which is modeled by the multi-species coalescent (MSC) model (Kingman, 1982), has been shown to be a major challenge for estimating species trees for many biological datasets, including birds (Jarvis *et al.*, 2014) and plants (Wickett *et al.*, 2014). Concatenation analyses using ML are not statistically consistent under the MSC+Generalized Time Reversible (GTR) model [even if run in a fully partitioned mode, as shown in Roch *et al.* (2019)], where the MSC+GTR model assumes that gene trees evolve within a species tree under the MSC model, and then sequences evolve down each gene tree under the GTR model (Tavaré, 1986). Furthermore, simulation studies have shown that concatenation analyses can have poor accuracy when ILS levels are high (Kubatko and Degnan, 2007; Mirarab *et al.*, 2014b, 2016; Molloy and Warnow, 2018c).

Because gene tree heterogeneity is frequently observed in biological datasets, many new methods have been developed to estimate species trees from multi-locus datasets taking ILS into consideration, surveyed in Warnow (2017). Some of the most scalable species tree methods operate by estimating gene trees and then combining the estimated gene trees together, typically using summary statistics. Methods that combine gene trees, referred to as 'summary methods', are often provably statistically consistent under the MSC model when given true gene trees as input. However, in practice, gene trees are estimated, and standard summary methods can be statistically inconsistent (and even positively misleading) when using estimated gene trees (Roch *et al.*, 2019). Furthermore, summary methods tend to have poorer accuracy when gene tree estimation error is high (Mirarab and Warnow, 2015) and can be less accurate than concatenation analyses using ML even under high ILS conditions when gene tree estimation error is sufficiently high (Molloy and Warnow, 2018c). Note that gene tree estimation error can be impacted by the choice of sequencing methodologies and sampling strategies; for example, ultra-conserved elements (UCEs) typically have lower phylogenetic signal and thus higher gene tree estimation error (Meiklejohn *et al.*, 2016).

Many phylogenomic analyses of biological datasets estimate species trees with both concatenation analyses and summary methods. Among summary methods, one of the most popular methods is ASTRAL (Mirarab *et al.*, 2014a), which has typically produced more accurate trees than other similarly scalable summary methods in simulation studies. ASTRAL has a polynomial running time that scales with the number of species, the number of gene trees, and the degree of heterogeneity in the input gene trees (which is greater when ILS levels are high). The most popular concatenation analysis is ML, often using RAxML (Stamatakis, 2014), which is optimized for multi-locus datasets. ML is NP-hard (Roch, 2006), so RAxML uses heuristics based on hill-climbing and randomization to search for an optimal ML tree within an exponentially-sized search space, terminating when its stopping criterion is met. As a result, RAxML can be very expensive to use for datasets with large numbers of species and even for datasets with only moderate numbers of species but with long alignments that are not highly compressible.

Divide-and-conquer is a well-known approach for scaling methods to larger datasets. Recently, Molloy and Warnow (2018a) presented a divide-and-conquer framework that operates by dividing species into pairwise disjoint subsets, estimating trees on subsets, and merging subset trees together using a new method, called NJMerge. NJMerge uses an estimated distance matrix to perform the merger, treating the input subset trees as absolute constraints, so that the topology of the returned species tree must agree with the topology of each of the input subset trees. Molloy and Warnow (2018b) proved that some species tree estimation pipelines using NJMerge are statistically consistent under the MSC+GTR model, and furthermore, pipelines using NJMerge were shown to dramatically reduce the running time of ASTRAL-III and RAxML on large multi-locus datasets, while maintaining accuracy. Despite these promising results, NJMerge has two main issues that limit its utility in practice. First, NJMerge can *fail* to return a tree, and although the failure rate in these studies was low, the conditions under which NJMerge fails have not been carefully evaluated. Second, when used within the proposed divide-and-conquer framework, NJMerge has $O(n^5)$ running time, where n is the number of species.

Here, we present two new methods for merging trees on pairwise disjoint leaf sets using an estimated distance matrix: NJMerge-2, which is a minor modification to NJMerge, and TreeMerge. Both NJMerge-2 and TreeMerge are guaranteed to return a compatibility supertree on all inputs, and so never fail. When used within the proposed divide-and-conquer framework, NJMerge-2 and a slow variant of TreeMerge have the same running time as NJMerge—$O(n^5)$. In contrast, a fast variant of TreeMerge has only $O(n^2)$ running time. We establish that divide-and-conquer pipelines using NJMerge-2 and TreeMerge are provably statistically consistent under the MSC+GTR model; however, the requirements for pipelines using the fast variant of TreeMerge to be statistically consistent are stronger than the very mild requirements for pipelines using the slow variant of TreeMerge. Despite this difference in theory, our experimental results show that the less constrained divide-and-conquer pipelines using the fast variant of TreeMerge dramatically reduced the running time of both ASTRAL-III and RAxML and maintained accuracy. Most significantly, both ASTRAL-III and RAxML failed to complete on many datasets with 1000 species and 1000 genes using the available resources (64 GB of memory and 48 h maximum wall-clock time); however, the fast variant of TreeMerge enabled each method to complete on all the datasets using the same resources. Thus, TreeMerge is a promising technique for scaling species tree estimation methods to larger datasets.

The rest of the paper is organized as follows. In Section 2, we present NJMerge-2 and TreeMerge and establish their theoretical guarantees. In Section 3, we describe how TreeMerge was used in a divide-and-conquer framework for species tree estimation and the experimental study designed to evaluate this approach. In Section 4, we discuss the results of this study, and we conclude in Section 5 with suggestions for future work.

2 Materials and methods

2.1 Divide-and-conquer framework

The input to the species tree estimation problem is a multi-locus dataset with sequences coming from m genomic regions for a set S of n species. We use a divide-and-conquer framework, introduced in Molloy and Warnow (2018a), to estimate the species tree (i.e. an unrooted binary tree with the n leaves labeled by the species set S) as follows:

1. Decompose the species set S into pairwise disjoint subsets with a predetermined maximum size.
2. Estimate a species tree on each subset, producing a set $\mathcal{T}$ of trees on pairwise disjoint subsets of species.
3. Compute any auxiliary information (e.g. a dissimilarity matrix D on S so that $D[x, y]$ is an estimated 'distance' between species x and species y) required for step 4.
4. Merge the set $\mathcal{T}$ of trees together into a compatibility supertree (Definitions 2 and 3) using the auxiliary information.

This approach requires the user to specify methods for performing each of the five steps. Thus, users may choose to select methods that operate on gene trees or methods that operate on concatenated alignments, depending on their preferred approach. If the subset trees and the dissimilarity matrix are estimated using statistically consistent methods, then the divide-and-conquer pipeline using NJMerge is provably statistically consistent under the MSC+GTR model [Corollary 7 in Molloy and Warnow (2018b)]. As an example, the subset trees could be estimated using ASTRAL and the dissimilarity matrix could be estimated using average leaf-to-leaf distances in the input gene trees [referred to as the AGID or the USTAR distance matrix; see Liu and Yu (2011); Allman *et al.* (2018)]. However, statistical consistency is not the main objective, as achieving high accuracy in practice is typically more challenging,

especially for large heterogeneous datasets, which is the focus of this study.

2.2 Terminology

All trees in this paper are unrooted binary trees with leaves labeled by elements in a set S of species. $\mathcal{T} = \{T_1, \ldots, T_k\}$ is a set of k trees with the property that each species in S is a leaf in exactly one tree in $\mathcal{T}$; thus, we say that the trees in $\mathcal{T}$ are **leaf-disjoint**. We denote the set of leaves in a tree T or in a set of trees $\mathcal{T}$ by $\mathcal{L}(T)$ and $\mathcal{L}(\mathcal{T})$, respectively. A **dissimilarity matrix** is a square matrix D that is zero on the diagonal and symmetric (i.e. $D[i,j] = D[j,i]$). We work with dissimilarity matrices (rather than distance matrices), because most statistical methods for estimating distances between species in phylogenetic analyses fail to satisfy the triangle inequality; see Warnow (2017). Finally, given a dissimilarity matrix D, we denote the restriction of D to the rows and columns for $\mathcal{L}(T_i) \cup \mathcal{L}(T_j)$ by $D^{i,j}$.

Definition 1. *Let T be a tree with edge-weighting $w : E(T) \to \mathbb{R}^+$ and leaves labeled $1, 2, \ldots, n$. We define the $n \times n$ matrix A by setting $A[i,j]$ to be the sum of the edge weights in T between i and j; such a matrix is said to be **additive** for T. We say that an $n \times n$ matrix M is **nearly additive** for T if for all i, j, $|M[i,j] - A[i,j]| < f/2$ where f is the length of the shortest internal edge in T.*

Definition 2. *Let T be a tree on leaf set S, and let T' be a tree on leaf set $R \subseteq S$. We say that T' **agrees with** T if restricting T to leaf set R induces a binary tree that (after suppressing the internal nodes of degree 2) is isomorphic to T'.*

Definition 3. *We say that a tree T on leaf set S is a* **compatibility supertree** *for a set $\mathcal{T}$ of k unrooted binary trees if each tree $T_i \in \mathcal{T}$ agrees with T and $\cup_{i=1}^{k} \mathcal{L}(T_i) = S$.*

We now describe the input and output for **Disjoint Tree Merger** (**DTM**) methods:

- **Input:** A set $\mathcal{T}$ of unrooted binary trees that are leaf-labeled by the set S so that each $s \in S$ labels exactly one leaf in exactly one tree in $\mathcal{T}$, and auxiliary information (e.g. a dissimilarity matrix, a multiple sequence alignment, etc.)
- **Output:** A compatibility supertree T for $\mathcal{T}$ (Definition 3)

Because the trees in $\mathcal{T}$ are leaf-disjoint, a compatibility supertree always exists (Fig. 1), and the objective is to find a compatibility supertree T that is close to the (unknown) true tree T^* in polynomial time by using the auxiliary information; note that no specific dependence between the input auxiliary information and the output supertree is implied.

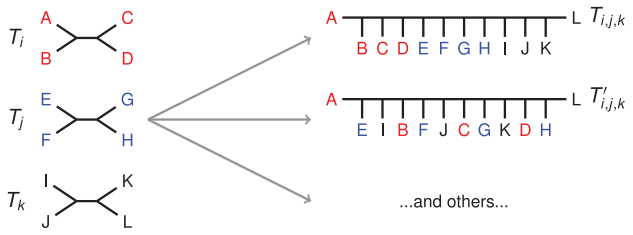

Fig. 1. Given a set of trees on pairwise disjoint leaf sets, many possible compatibility supertrees exist. Here we show two compatibility supertrees, labeled $T_{i,j,k}$ and $T'_{i,j,k}$, for $\mathcal{T} = \{T_i, T_j, T_k\}$; others can also be identified. Notably, the trees in $\mathcal{T}$ form connected subtrees in $T_{i,j,k}$ (i.e. $T_{i,j,k}$ is formed by connecting the trees in $\mathcal{T}$ by edges), but this is not the case for $T'_{i,j,k}$. We refer to the first type of compatibility supertree as *un-blended*, and the second type as *blended*

2.3 NJMerge and NJMerge-2

NJMerge is a DTM method based on the well-known Neighbor Joining (NJ) method (Saitou and Nei, 1987), but modified to address the requirement of obeying the input constraint trees. As NJMerge considers siblinghood proposals, it modifies the constraint trees and checks to see if the set of constraint trees are still compatible. Because determining the compatibility of a set of k unrooted trees on overlapping leaf sets is NP-complete (Steel, 1992), NJMerge uses a *heuristic* that can fail for $k > 2$ trees. In other words, NJMerge sometimes accepts a siblinghood proposal that causes the set of constraint trees to become incompatible, in which case NJMerge fails to return a tree. This failure is not due to insufficient computational resources and is instead an *algorithmic failure*; however, anytime that NJMerge does *not* fail, it returns a compatibility supertree for $\mathcal{T}$. A second limitation of NJMerge is its $O(kn^4)$ running time, where the input has n species divided among k leaf-disjoint trees. Thus, NJMerge has two limitations: it can fail, and its running time (although polynomial) is not fast enough to run on ultra-large datasets.

NJMerge-2 is a simple extension to NJMerge that is guaranteed to always return a compatibility supertree, addressing the first limitation of NJMerge. NJMerge-2 operates as follows:

1. Select $T_i, T_j \in \mathcal{T}$ with $i \neq j$.
2. Build a compatibility supertree $T_{i,j}$ for $\{T_i, T_j\}$ by running NJMerge on the input pair $(\{T_i, T_j\}, D^{i,j})$.
3. Update $\mathcal{T}$ by removing T_i and T_j and adding $T_{i,j}$.
4. Repeat Steps 1-3 until $|\mathcal{T}| = 1$.
5. Return $\mathcal{T}$.

Theorem 4. NJMerge-2 returns a compatibility supertree in $O(kn^4)$ time, where the input has n species divided among k leaf-disjoint subset trees. Furthermore, if every tree in $\mathcal{T}$ agrees with a tree T^* and D is nearly additive for T^*, then NJMerge-2 returns T^*.

Proof. It is easy to see that the constraint trees remain leaf-disjoint (and thus compatible) during the iterative process, and because the heuristic used by NJMerge correctly determines the compatibility for exactly two trees, NJMerge returns a compatibility supertree given any two compatible trees as input. Thus, by induction on the number of constraint trees, NJMerge-2 returns a compatibility supertree given any set of leaf-disjoint trees and dissimilarity matrix as input. If, in addition, the trees in $\mathcal{T}$ are compatible with T^* and the dissimilarity matrix D is nearly additive for T^*, then NJMerge applied to the input $(\{T_i, T_j\}, D^{i,j})$ for any pair of trees $T_i, T_j \in \mathcal{T}$ returns a tree that agrees with T^* [Theorem 3 in Molloy and Warnow (2018b)]. Thus, the set $\mathcal{T}$ remains compatible with T^* during the iterative process, and, by induction on the number of constraint trees, NJMerge-2 returns T^*. For the running time analysis, we make the simplifying assumption that each tree in $\mathcal{T}$ has exactly n/k leaves, where $|S| = n$. At iteration x, for any two trees $t, t' \in \mathcal{T}$, $|\mathcal{L}(t) \cup \mathcal{L}(t')| \leq (x+1)n/k$, so NJMerge can be run on any pair of trees in $O(x^4 n^4/k^4)$ time. A total of $k-1$ iterations are required, and so the running time of NJMerge-2 scales with $(n^4/k^4)\sum_{x=2}^{k} x^4$ (note that $\sum_{x=2}^{k} x^4$ is $O(k^5)$). Thus, NJMerge-2 has $O(kn^4)$ running time. □

2.4 TreeMerge

We now present a new DTM method called TreeMerge that is inspired by PASTA (Mirarab *et al.*, 2015), a divide-and-conquer

method for co-estimating large-scale multiple sequence alignments and gene trees. A generic version of the TreeMerge algorithm is presented in the box below.

The generic TreeMerge algorithm

Input: A set $\mathcal{T}$ of leaf-disjoint trees on species set S, and auxiliary information (which includes a tree $\mathcal{G}$ with nodes labeled by $\mathcal{T}$ so that $T_i \in \mathcal{G}$ labels exactly one node in $\mathcal{G}$)
Output: A compatibility supertree T for $\mathcal{T}$
Stage 1. For each edge $(T_i, T_j) \in E(\mathcal{G})$, build a compatibility supertree $T_{i,j}$ for $\{T_i, T_j\}$ using auxiliary information.
Stage 2. Iteratively merge pairs of trees via their shared backbone tree (defined in the text), as indicated by $\mathcal{G}$.
2a. Root $\mathcal{G}$ at an arbitrary edge (T_x, T_y), and let $T = T_{x,y}$.
2b. Perform a pre-order traversal of the edges in the rooted spanning tree $\mathcal{G}$. For each edge $(T_i, T_j) \in E(\mathcal{G})$:
- Build a compatibility supertree T' for $\{T, T_{i,j}\}$ using auxiliary information.
- Let $T = T'$.
2c. Return T.

We now describe how we performed stage 1 and stage 2 of the generic TreeMerge algorithm in our simulation study, using an estimated dissimilarity matrix D on S as auxiliary information.

Stage 1. We build a compatibility supertree $T_{i,j}$ for $\{T_i, T_j\}$ by running NJMerge on the input pair $(\{T_i, T_j\}, D^{i,j})$.

Stage 2. We merge pairs of trees via their shared backbone tree using techniques similar to the Strict Consensus Merger (Swenson *et al.*, 2012; Warnow *et al.*, 2001). We describe this in the context of merging two trees $T_{i,j}$ and $T_{j,k}$ with $\mathcal{L}(T_{i,j}) = \mathcal{L}(T_i) \cup \mathcal{L}(T_j)$ and $\mathcal{L}(T_{j,k}) = \mathcal{L}(T_j) \cup \mathcal{L}(T_k)$. Because $T_{i,j}$ and $T_{j,k}$ induce a common tree topology T_j on their shared leaf set $\mathcal{L}(T_j)$, we refer to T_j as the **backbone tree**. A compatibility supertree for $\{T_{i,j}, T_{j,k}\}$ can be built by simply inserting the missing leaves (i.e. $\mathcal{L}(T_i) \cup \mathcal{L}(T_k)$) into the backbone tree T_j. Note that each edge e in T_j maps to a path p in $T_{i,j}$ and a path p' in $T_{j,k}$ (where the paths may be of length 1). When the edge maps to a path of length greater than 1, the internal nodes on those paths define subtrees that need to be inserted into the backbone tree T_j. Thus, we iterate over each edge $e \in T_j$, adding subtrees from $T_{i,j}$ and/or $T_{j,k}$ to T_j, until all missing leaves have been inserted. This process is straightforward except when both $T_{i,j}$ and $T_{j,k}$ contribute one or more subtrees to the same edge; this is called a **collision** (Warnow *et al.*, 2001). When collisions occur, there are multiple ways to merge the two trees while maintaining compatibility (Fig. 2), and picking the best one cannot be done purely using topological information in the two input trees. We propose two methods, one slow and one fast, to resolve collisions using the dissimilarity matrix D, referred to as 'TreeMerge-slow' and 'TreeMerge-fast', respectively.

TreeMerge-slow. TreeMerge-slow resolves edge collisions between two trees as follows. Let $e = (X, Y)$ be an edge in the backbone tree involved in a collision, and select two leaves $x, y \in T_j$ on opposite sides of e. Let $T_{i,j}|_e$ (and $T_{j,k}|_e$) denote the set of subtrees in $T_{i,j}$ (and $T_{j,k}$) that need to be attached to edge $e \in T_j$, and define constraint trees t and t' by restricting $T_{i,j}$ to leaf set $\{x, y\} \cup \mathcal{L}(T_{i,j}|_e)$ and by restricting $T_{j,k}$ to leaf set $\{x, y\} \cup \mathcal{L}(T_{j,k}|_e)$. Because t and t' share only two leaves in common (and thus are compatible), a compatibility supertree for $\{t, t'\}$ can be computed in polynomial time by running NJMerge on the input pair $(\{t, t'\}, D')$, where D' is D restricted to $L(t) \cup L(t')$. The path from x to y in the resulting tree indicates how this compatibility supertree for $\{t, t'\}$ should be inserted into $e \in T_j$. Note that this technique enables t and t' to **blend**, as shown in Figure 2.

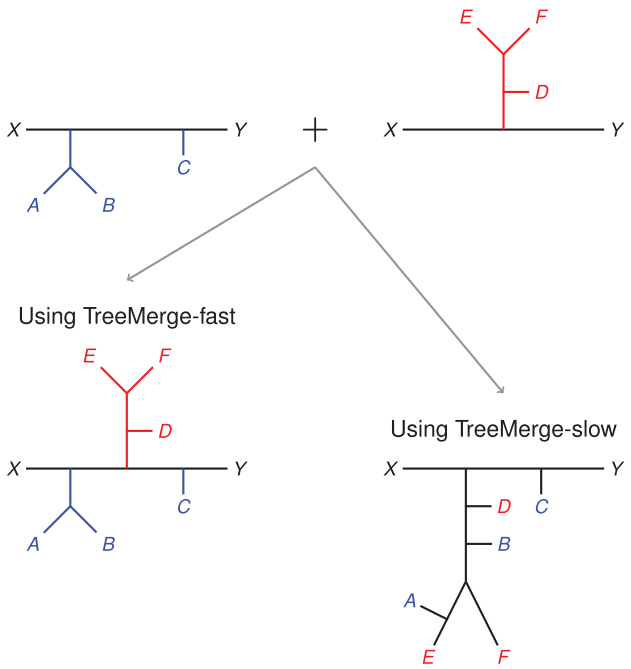

Fig. 2. TreeMerge-slow versus TreeMerge-fast. TreeMerge-slow and TreeMerge-fast differ with respect to how they resolve edge collisions. Consider the case where we have two compatible trees t and t' involved in a collision on edge (X, Y) in the backbone tree. We label the corresponding internal nodes in t and t' also by X and Y and then restrict the trees to just the subtrees between X and Y to illustrate how the two methods perform on this input. Note that the subtree of t shown on the upper left needs to insert two subtrees [trees (A, B) and C] and the subtree of t' shown on the upper right needs to insert one subtree [tree $(D, (E, F))$] onto edge (X, Y). TreeMerge-slow runs NJMerge on the restricted versions of t' and t' (as described in the main text), resulting in a 'blended' compatibility supertree that has only two subtrees inserted into the backbone edge. In contrast, TreeMerge-fast uses the branch lengths in t and t' and rescales them (if necessary) so that the lengths of the paths from X to Y in the two trees are the same (as shown here). These paths can then be superimposed, so that the ordering in which the subtrees appear along the path defines how the two trees are merged. Thus, TreeMerge-fast maintains the separation between the subtrees and does not allow 'blending' when resolving collisions

TreeMerge-fast. TreeMerge-fast resolves edge collisions between two trees as follows. First, TreeMerge-fast fits branch lengths to all trees $T_{i,j}$ (computed in step 2) using the $O(p^2)$ time algorithm from Bryant and Waddell (1998) that takes as input a tree topology with p leaves and a $p \times p$ dissimilarity matrix and computes the optimal branch lengths for T using a least squares approach. For two trees $T_{i,j}$ and $T_{j,k}$, each contributing one or more subtrees to an edge e in the backbone tree T_j, TreeMerge-fast rescales the branch lengths so that the associated paths (corresponding to edge e) in the two trees have the same length. Then, the rescaled branch lengths determine the order in which the subtrees are inserted into e (Fig. 2). Note that this technique never produces a blended compatibility supertree, so the subtrees added define clades (subtrees defined by edges in the tree) in the resulting compatibility supertree.

THEOREM 5. TreeMerge-slow and TreeMerge-fast return a compatibility supertree in $O(kn^4)$ and $O(nk + n^2/k + n^4/k^3)$ time, respectively, where the input has n species divided among k leaf-disjoint subset trees. TreeMerge-slow and TreeMerge-fast require $O(n^2)$ and $O(n^2/k)$ storage, respectively.

PROOF. The proof that TreeMerge-slow and TreeMerge-fast return a compatibility supertree follows from NJMerge being guaranteed to return a compatibility supertree for two leaf-disjoint trees and from the techniques for handling collisions (i.e. using either NJMerge on pairs of compatible constraint trees or using branch lengths) being guaranteed to return a compatibility supertree for two trees that agree on their shared leaf set.

For the running time analysis, we make the simplifying assumption that each tree in $\mathcal{T}$ has exactly n/k leaves, where $|S| = n$. In stage 1, both TreeMerge-slow and TreeMerge-fast run NJMerge on input pair $(\{T_i, T_j\}, D^{i,j})$ for each of the k-1 edges in the spanning tree $\mathcal{G}$. All input constraint trees have n/k leaves, so stage 1 uses $O(n^4/k^3)$ time and $O(n^2/k)$ storage.

TreeMerge-fast and TreeMerge-slow address stage 2 differently; we discuss these methods separately, using the same notation presented in the box above. We begin by analyzing TreeMerge-slow's running time for stage 2. At iteration x, TreeMerge-slow merges trees $T_{i,j}$ and T using a shared backbone tree t_B, where $|\mathcal{L}(T_{i,j})| = 2n/k$, $|\mathcal{L}(T)| = (x+1)n/k$, and $|\mathcal{L}(t_B)| = n/k$. Thus, there are $(x+1)n/k$ possible leaves that could be contributed to the $n/k - 3$ edges in t_B. In the worst case analysis, $(x+1)n/k$ leaves are contributed to a single edge in t_B, and then NJMerge runs in $O(x^4n^4/k^4)$ time. A total of $k-2$ iterations are required, and so the running time scales with $(n^4/k^4)\sum_{x=3}^{k} x^4$ (note that $\sum_{x=3}^{k} x^4$ is $O(k^5)$). In the final iteration, NJMerge requires $O(n^2)$ storage in the worst case analysis. Thus, TreeMerge-slow has $O(kn^4)$ time and $O(n^2)$ storage for stage 2. We now analyze TreeMerge-fast's running time for stage 2. First, TreeMerge-fast computes branch lengths for the $k-1$ trees (each with $2n/k$ leaves) from stage 1 using the quadratic time and space algorithm from Bryant and Waddell (1998). Then, at iteration x, TreeMerge-fast merges trees $T_{i,j}$ and T using a shared backbone tree t_B and branch lengths; note that this approach scales linearly with the number of leaves in the merged tree (i.e. $|\mathcal{L}(T_{i,j}) \cup \mathcal{L}(T)| = (x+2)n/k$). A total of $k-2$ iterations are required, and so the running time scales with $(n/k)\sum_{x=3}^{k} x$ [note that $\sum_{x=3}^{k} x$ is $O(k^2)$]. Thus, TreeMerge-fast has $O(kn + n^2/k)$ time and $O(n^2/k)$ space for stage 2. Overall, TreeMerge-fast has $O(nk + n^2/k + n^4/k^3)$ time and $O(n^2/k)$ storage, while TreeMerge-slow has $O(kn^4)$ time and requires $O(n^2)$ storage. □

Corollary 6. *When run within the divide-and-conquer framework proposed in Section 2.1, NJMerge, NJMerge-2 and TreeMerge-slow use $O(n^5)$ time. In contrast, TreeMerge-fast uses $O(n^2)$ time when run within this same divide-and-conquer framework.*

PROOF. In the divide-and-conquer framework, the input dataset of n species is divided into k pairwise disjoint subsets of bounded size, so that $k = O(n)$. The time to run NJMerge, NJMerge-2 or TreeMerge-slow given the subset trees and the dissimilarity matrix as input is $O(kn^4)$, and thus the total time to run NJMerge, NJMerge-2 or TreeMerge-slow within the divide-and-conquer framework is $O(n^5)$. In contrast, the time to run TreeMerge-fast given the subset trees, the dissimilarity matrix, and the spanning tree on vertex set $\mathcal{T}$ as input is $O(nk + n^2/k + n^4/k^3)$, and thus the total time to run TreeMerge-fast within the divide-and-conquer framework is $O(n^2)$. □

THEOREM 7. Suppose every tree in $\mathcal{T}$ agrees with a tree T^* and D is nearly additive for T^*. Then TreeMerge-slow returns T^*.

PROOF. Let T_x and T_y be any two trees in $\mathcal{T}$. By Theorem 3 in Molloy and Warnow (2018b), NJMerge applied to the input $(\{T_x, T_y\}, D^{x,y})$ returns a compatibility supertree tree $T_{x,y}$ that agrees with T^*. Since TreeMerge-slow performs all its mergers using NJMerge, the result follows by induction on the number of mergers. □

Corollary 8 follows easily from Theorems 4 and 7 [see also Corollary 7 in Molloy and Warnow (2018b)], and its proof is omitted due to space constraints.

Corollary 8. *Any divide-and-conquer pipeline following the protocol in Section 2.1 and using NJMerge-2 or TreeMerge-slow is statistically consistent under the MSC+GTR model when Step 2 is performed with a statistically consistent method for estimating species trees on subsets and Step 3 is performed with a statistically consistent method for estimating distances.*

Unfortunately, TreeMerge-fast is not guaranteed to be statistically consistent for these general pipelines, because it can fail in some rare conditions to correctly combine trees in the presence of collisions. (The simplest example is where the decomposition strategy produces three subsets by deleting three edges all sharing a common endpoint, creating a condition where the correct merger technique must detect that the backbone edge should be subdivided into exactly two edges and not into three, which is what the current strategy in TreeMerge-fast would do.) However, a simple modification to the pipeline is sufficient to guarantee statistical consistency, as we now show.

THEOREM 9. Consider the pipeline where gene trees are computed using a statistically consistent method, the distance matrix D is the AGID matrix, and the starting tree T_0 is computed using a statistically consistent method (e.g. NJ on the AGID matrix). Suppose that the starting tree is decomposed into disjoint subsets $A_1, A_2, \ldots, A_k$ by removing a set E_0 of edges from T_0 so that every pair of edges in E_0 is separated by at least two edges in T_0. For $i = 1, 2, \ldots, k$, let t_i denote the constraint tree computed on subset A_i using a statistically consistent method (e.g. ASTRAL). Given node v in T_0, label v by i where v is on a path between two leaves that are both in A_i. Define the spanning tree $\mathcal{G}$ on the constraint trees by making t_i and t_j adjacent in $\mathcal{G}$ if and only if there is an edge in T_0 whose endpoints are labelled by i and j. Then as the number of genes and number of sites per gene both increase, the species tree computed using TreeMerge-fast on input $(\mathcal{T} = \{t_1, t_2, \ldots, t_k\}, D, \mathcal{G})$ will converge to the true species tree. In other words, this pipeline, using TreeMerge-fast to combine subset trees, is statistically consistent under the MSC+GTR model.

PROOF. As the number of genes and sites per gene increases, all estimated gene trees will converge to the true gene trees, the AGID distance matrix will converge to an additive matrix defined by the true species tree, the starting tree will converge to the true species tree, and each of the constraint trees will converge to a tree that agrees with the true species tree. Hence, for a large enough number of genes and sites per gene, with high probability (i) the deletion of the selected set of edges from the starting tree will partition the species set into subsets so that the labelling described above is unique for each internal node in the tree (i.e. the labelling is 'convex' on the starting tree), (ii) constraint trees computed on each subset will be equal to the true species tree on the subset, and (iii) constraint trees that are adjacent in the spanning tree will be adjacent in the true species tree as well. Under these conditions, it is then easy to see that applying TreeMerge-fast will return the true species tree, because neither conflicts nor collisions will ever occur. □

3 Performance study

3.1 Overview

The main goal of this paper is empirical performance on large multilocus datasets, and so we focus our attention on TreeMerge-fast. We

include a comparison to NJMerge and NJMerge-2, noting that TreeMerge-slow and NJMerge-2 have the same theoretical performance (i.e. asymptotic computational complexity and statistical consistency) and NJMerge-2 is simpler to implement. We explore species tree estimation using ASTRAL v5.6.1 (i.e. ASTRAL-III) and RAxML v8.2.12 (with SSE3 and pthreads). We computed species trees using these two methods *de novo* as well as within a divide-and-conquer framework, where they were used to estimate species trees on subsets and then the estimated subset (species) trees were combined using NJMerge, NJMerge-2, and TreeMerge-fast. We used the exact same commands (for ASTRAL-III and RAxML) to estimate trees on the full species set as well as on the subsets. We evaluated performance with respect to algorithmic failure rate (for NJMerge), computational failure rate (i.e. failure to complete due to insufficient computational resources), running time, and topological accuracy. All datasets used in this study are publicly available on the Illinois Data Bank (https://databank.illinois.edu/datasets/IDB-9570561), and the exact software commands used in this study are in the Supplementary Materials.

3.2 Datasets

We used datasets with 1000 species and 2000 genes from a prior study (Molloy and Warnow, 2018a) and describe the simulation protocol below. SimPhy (Mallo *et al.*, 2016) was used to generate gene trees within species trees under the MSC model. By holding the effective population size constant and varying the species tree height (in generations), model conditions with two different levels of ILS were created, each with 20 replicate datasets. The ILS level was measured by the average normalized Robinson–Foulds (RF) distance (Robinson and Foulds, 1981) between the true species tree and the true gene trees (called AD). For the two model conditions, the average AD was 8–10% and 68–69%, and we refer to these conditions as 'low/moderate ILS' and 'very high ILS', respectively.

Sequence alignments were simulated for each true gene tree using INDELible (Fletcher and Yang, 2009) under the GTR + Γ model of evolution. Sequence lengths were drawn from a distribution (varying from 300 to 1500 bp). GTR + Γ model parameters were also drawn from distributions [see Supplementary Table S2 in Molloy and Warnow (2018a) for details] to simulate 1000 exon-like sequences and 1000 intron-like sequences. Exons were characterized by slower rates of evolution and thus had less phylogenetic signal than introns. Exons and introns were analyzed separately, so all datasets had 1000 species and 1000 genes.

Finally, summary methods take gene trees as input, and we used estimated gene trees that had previously been computed by Molloy and Warnow (2018a) with FastTree-2 (Price *et al.*, 2010). The average gene tree estimation error across all replicate datasets was from 26–51% for introns and 38–64% for exons.

3.3 Details for running divide-and-conquer pipelines

Subset decomposition. We evaluated divide-and-conquer pipelines that used an estimated starting tree to divide the species set into pairwise disjoint subsets. Specifically, the set of 1000 species was divided into 10–15 subsets by repeatedly deleting 'centroid' edges (i.e. edges whose deletions divide the leaf set roughly in half) in the starting tree until each subset was smaller than the maximum size of 120 species.

Running TreeMerge-fast. Because we used a starting tree T^s to decompose the species set into subsets, we also used T^s to construct a spanning tree on vertex set $\mathcal{T}$ (i.e. the tree $\mathcal{G}$ has vertices labeled by $\mathcal{T}$ so that $T_i \in \mathcal{G}$ labels exactly one node in $\mathcal{G}$) for TreeMerge-fast as follows: first, we randomly selected one leaf from each tree in $\mathcal{T}$ and deleted all remaining leaves from T^s (suppressing the internal nodes of degree 2), thus producing a tree T that has one leaf for every tree in $\mathcal{T}$. Second, we built a complete graph $\mathcal{G}^s$ with nodes labeled by the trees in $\mathcal{T}$ and edges (T_i, T_j) weighted by the path distance between leaves labeled T_i and T_j in T. Third, we computed a minimum spanning tree $\mathcal{G}$ on $\mathcal{G}^s$ using Kruskal (1956). Finally, TreeMerge-fast used PAUP* (Swofford, 2019) v4a163 to fit least squares branch lengths to trees using a distance matrix, prohibiting negative branch lengths.

Divide-and-conquer pipeline using ASTRAL-III. We evaluated these pipelines by giving NJMerge, NJMerge-2 and TreeMerge the following inputs:

- *Distance matrix:* AGID matrix computed using ASTRID (Vachaspati and Warnow, 2015) v1.4 given estimated gene trees as input
- *Starting tree:* NJ tree computed using FastMe (Lefort *et al.*, 2015) v2.1.5 on the AGID distance matrix [i.e. the NJst tree from Liu and Yu (2011)]
- *Constraint trees:* Species trees computed using ASTRAL v5.6.1 (i.e. ASTRAL-III) on each subset given the induced subtrees from the estimated gene trees as input

Divide-and-conquer pipeline using RAxML. We evaluated these pipelines by giving NJMerge, NJMerge-2 and TreeMerge the following inputs:

- *Distance matrix:* Matrix of log-det distances (Steel, 1994) computed using PAUP* v4a163 given the concatenated alignment as input
- *Starting tree:* Greedy maximum parsimony tree based on a random taxon addition order computed using RAxML v8.2.12 (with SSE3 and pthreads) given the concatenated alignment as input
- *Constraint trees:* Species trees computed under the GTR+Γ model of evolution using RAxML v2.12 (with SSE3 and pthreads) on each subset given the unpartitioned concatenated alignment for those species as input

3.4 Evaluation criteria

Species tree error. Species tree error was measured as the RF error rate (i.e. the normalized RF distance between the true and the estimated species trees) using Dendropy (Sukumaran and Holder, 2010).

Running time. All analyses were performed using a single Blue Waters compute node (XE6 dual-socket nodes with 64 GB of physical memory and two AMD Interlagos model 6276 CPU processors, i.e. one per socket each with eight floating-point cores) and a maximum wall-clock time of 48 h. All methods were given access to 16 threads with 1 thread per bulldozer (floating-point) core; however, only RAxML was implemented with multi-threading at the time of this study. We used the last checkpoint file written by RAxML to evaluate species tree estimation error and running time; as a result the running time for RAxML was measured as the time between the info file and the last checkpoint file being written. Because we assumed that researchers had access to a single compute node, we approximated the total running time of divide-and-conquer pipelines as

$$t(\Phi_D(S)) + \sum_{i=1}^{k} t(\Phi_T(S_i)) + t(\Phi_M(\mathcal{T}, D)), \quad (1)$$

where Φ_D is the method used to estimate a distance matrix, Φ_T is the method used to estimate constraint trees, and Φ_M is the method used

to merge the set of constraint trees using a distance matrix. The approximate running time does not include the time required to estimate starting trees, as these computations were relative fast, requiring only a few minutes.

4 Results and discussion

4.1 How does TreeMerge compare with NJMerge and NJMerge-2?

A comparison between these methods with respect to their theoretical properties is provided in Table 1; here we explore their empirical performance. When RAxML was used to construct subset trees, NJMerge failed to return a tree on 6 out of 80 datasets (i.e. the failure rate was 7.5%); note that these were algorithmic failures, not computational failures. For the same analyses in Molloy and Warnow (2018a), only 2 out of 80 datasets resulted in failures (i.e. the failure rate was 2.5%). The only difference between the analyses here and the ones in Molloy and Warnow (2018a) is the starting tree: here we used the randomized parsimony tree from RAxML, whereas in Molloy and Warnow (2018a) starting trees were computed using NJ on the log-det distance matrix. This finding suggests that the choice of starting tree may be a factor in whether or not NJMerge fails. In contrast, NJMerge-2 and TreeMerge-fast completed on all datasets within the allowed time using the allowed memory. Finally, we compared the accuracy of NJMerge, NJMerge-2, and TreeMerge-fast on the replicate datasets for which NJMerge returned a tree. In this analysis, NJMerge-2 produced trees with the same average error as NJMerge, while trees produced by TreeMerge-fast had at most 1% greater error on average than those produced by NJMerge (Supplementary Table S1).

That TreeMerge-fast had similar performance compared with NJMerge and NJMerge-2 is noteworthy, as the ASTRAL-III pipeline guarantees statistical consistency for NJMerge and NJMerge-2, but does *not* for TreeMerge-fast. (The centroid edge decomposition does not prohibit the case described above Theorem 9.) There are two conclusions to be drawn from this result. First, it is a reminder that statistical consistency is a theoretical guarantee for what happens in the limit, as the amount of data increases, and does not predict performance given finite data. Second, it suggests the possibility that TreeMerge-fast might be even more accurate when used within pipelines that do provide a guarantee of statistical consistency, which we note in Section 5 as a topic for future research.

Table 1. Theoretical properties of NJMerge, NJMerge-2, TreeMerge-slow, and TreeMerge-fast

	NJMerge	NJMerge-2	TreeMerge-slow	TreeMerge-fast
Can fail?	Yes	No	No	No
Consistent?	Yes	Yes	Yes	Yes*
Runtime	$O(kn^4)$	$O(kn^4)$	$O(kn^4)$	$O(nk + n^4/k^3)$
Storage	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^2/k)$
D&C runtime	$O(n^5)$	$O(n^5)$	$O(n^5)$	$O(n^2)$

Note: 'Can Fail?' means that the method can, on some inputs, fail to return a tree due to algorithmic issues (rather than limited computational resources). 'Consistent?' means that the method is statistically consistent under the MSC+GTR model, when used within the divide-and-conquer framework described in Section 2.1. Runtime and storage are for worst-case analysis. 'D&C runtime' is the runtime of the method, when used within the divide-and-conquer framework described in Section 2.1. 'Yes*' indicates that a slightly modified divide-and-conquer pipeline is needed to ensure statistical consistency for TreeMerge-fast; see Theorem 9.

4.2 What is the impact of using TreeMerge-fast on ASTRAL-III and RAxML?

Failure rate. In our experiments, ASTRAL-III and RAxML failed to complete analyses on many datasets, though for different reasons. ASTRAL-III failed to complete its analyses within 48 h using the available computational resources on 19 (out of 40) exon datasets and 4 (out of 40) intron datasets (i.e. the failure rate was 29%), and note that all failures occurred on datasets with very high ILS. RAxML reported Out Of Memory (OOM) errors on 39 (out of 40) intron datasets and on 3 (out of 40) exon datasets (i.e. the failure rate was 53%). In contrast, when run within divide-and-conquer pipelines using TreeMerge-fast, all analyses with ASTRAL-III and RAxML completed. Thus, TreeMerge-fast enabled both RAxML and ASTRAL-III to complete analyses on large datasets, when given only 64 GB of memory and 48 h wall-clock time.

The failure rate for ASTRAL-III and RAxML depended on the model condition. RAxML failed (due to running out of memory) on more intron datasets than exon datasets. Exon-like sequences, which evolved more slowly than intron-like sequences, had fewer distinct alignment patterns, and so could be compressed. When the alignments could not be effectively compressed (as was the case for the intron datasets), RAxML was more likely to run out of memory. A distributed-memory version of RAxML, called ExaML (Stamatakis and Aberer, 2013), can be used to estimate trees when RAxML runs out of memory, provided the user has access to a distributed-memory system; however, in our study, we explicitly limited all methods to a single compute node. ASTRAL-III failed (due to running longer than 48 h) on datasets with very high ILS. High ILS datasets are characterized by true gene trees that are topologically very different from the true species tree and from each other, resulting in a very large number of distinct bipartitions. The running time of ASTRAL-III scales with the number of distinct bipartitions in the estimated gene trees (Zhang *et al.*, 2018a), explaining why ASTRAL-III failed to complete within 48 h on many of the high ILS datasets.

Species tree error. When ASTRAL-III or RAxML completed, we were able to compare the accuracy of species trees estimated by these base methods *de novo* or *within the divide-and-conquer pipeline*, where TreeMerge-fast was used to combine subset trees. We found that using the divide-and-conquer pipeline had little impact on the accuracy of ASTRAL-III and RAxML, with error rates differing on average by at most 1% (Figs 3 and 4). In general, ASTRAL-III run *de novo* was on average 1% more accurate than when run within the divide-and-conquer pipeline. The impact on RAxML was approximately the same, with a difference of at most 1% on average, and for some datasets with very high ILS, the *de novo* approach was less accurate than the divide-and-conquer approach.

Running time. The divide-and-conquer pipeline using TreeMerge-fast reduced running time for both ASTRAL-III and RAxML, often dramatically (Figs 3 and 4). For example, when the level of ILS was very high, ASTRAL-III run *de novo* used on average 42 h, whereas ASTRAL-III run within the divide-and-conquer pipeline used just under 4 h on average. On exon datasets, RAxML run *de novo* used 43 h on average, whereas RAxML used 11 h on average within the divide-and-conquer pipeline. Furthermore, the total time spent merging trees using TreeMerge-fast was always low: on average only 32 min and never more than 46 min, which was just a small percentage (on average 3–7%) of the time required to estimate subset trees using RAxML (Supplementary Table S2). We prototyped TreeMerge-fast (without parallelism) in Python using Dendropy (Sukumaran and Holder, 2010), and so an optimized

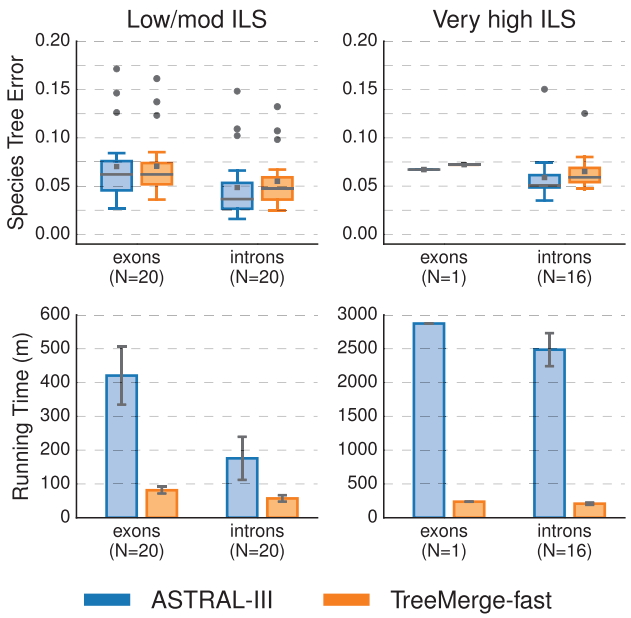

Fig. 3. Impact of using TreeMerge-fast with ASTRAL-III. The top row shows species tree estimation error for datasets with 1000 taxa and 1000 genes; gray bars represent medians, gray squares represent means, gray circles represent outliers, box plots extend from the first to the third quartiles, and whiskers extend to plus/minus 1.5 times the interquartile distance (unless greater/less than the maximum/minimum value). The bottom row shows running time (in minutes); bars represent means and error bars represent standard deviations across replicate datasets. The running time of TreeMerge-fast is the time to estimate the distance matrix, to estimate each subset tree using ASTRAL-III, and to combine the subset trees using TreeMerge-fast (Equation 1). The number N of replicates on which ASTRAL-III completed is shown on the x-axis; note that averages are taken across the replicates on which ASTRAL-III completed. When ASTRAL-III did not complete, it was due to running longer than the 48-h maximum wall-clock time

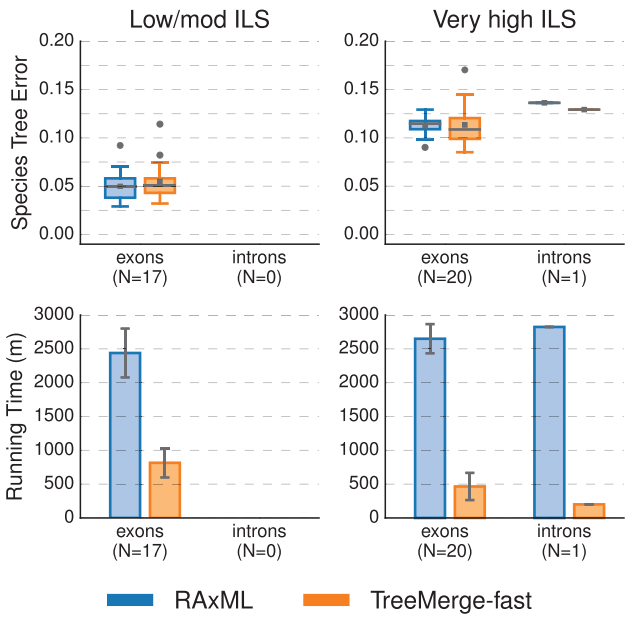

Fig. 4. Impact of using TreeMerge-fast with RAxML. The top row shows species tree estimation error for datasets with 1000 taxa and 1000 genes; note that gray bars represent medians, gray squares represent means, gray circles represent outliers, box plots extend from the first to the third quartiles, and whiskers extend to plus/minus 1.5 times the interquartile distance (unless greater/less than the maximum/minimum value). The bottom row shows running time (in minutes); bars represent means and error bars represent standard deviations across replicate datasets. The running time of TreeMerge-fast is the time to estimate the distance matrix, to estimate each subset tree using RAxML, and to combine the subset trees using TreeMerge-fast (Equation 1). The number N of replicates on which RAxML completed is shown on the x-axis; note that averages are taken across the replicates on which RAxML completed. When RAxML did not complete, it was due to OOM errors; otherwise the last checkpoint written by RAxML was evaluated

implementation of TreeMerge-fast with multi-threading would produce even better results.

Many phylogenomic studies [e.g. Prum *et al.* (2015) and Wickett *et al.* (2014)] have analyzed multi-locus datasets using *both* ASTRAL and RAxML, so that differences in computational requirements are of interest. The timings we report for ASTRAL-III and RAxML are *not* directly comparable, because the ASTRAL-III timings do not include the time required for gene tree estimation with FastTree-2, which has a worst case running time of $O(n^{1.5}\log(n)L)$, where n is the number of species and L is the length of the alignment (ML gene tree estimation using other methods would likely require more time). In our study, the average time ($\pm$ standard deviation) to estimate the full set of 1000 gene trees, each with 1000 species, was 217 ± 20 min using a single Blue Waters compute node (with 64 GB of memory and 16 floating-point cores). Thus, the amount time spent estimating 1000 gene trees (3.6 h on average) was small in comparison to the amount of time spent running ASTRAL-III on datasets with very high ILS (42 h on average). However, if the time for gene tree estimation were included, then the *percent* difference between running ASTRAL-III *de novo* or within the divide-and-conquer pipeline would decrease.

Recall that we earlier established that the component of the pipeline that uses TreeMerge-fast has a running time of only $O(n^2)$; here we discuss the big-O complexity of the steps that proceed it. In the preprocessing component of the ASTRAL-III pipeline, m gene trees (each with n species) are computed, and the most expensive parts, after computing gene trees, are the calculation of the AGID matrix, which uses $O(mn^2)$ time, and the calculation of the starting tree using NJ, which uses $O(n^3)$ time (although this could be reduced by using FastME instead of NJ, as done in ASTRID). Thus the preprocessing component of the pipeline is more expensive than the merger of the subset trees using TreeMerge-fast.

5 Conclusion

TreeMerge is a new technique for merging a set of leaf-disjoint trees through the use of an auxiliary dissimilarity matrix. TreeMerge is motivated by the success of NJMerge (Molloy and Warnow, 2018a,b) and addresses two important limitations: NJMerge can fail to return a tree on some datasets due to algorithmic failure (rather than computational issues), and NJMerge has asymptotic complexity that scales with kn^4 (where the input has n species divided among k leaf-disjoint trees), and so scales as n^5 within the divide-and-conquer framework described in Section 2.1. In contrast, TreeMerge-fast runs in $O(n^2)$ time within the same divide-and-conquer framework. Our study shows that the impact of TreeMerge-fast is greatest for those datasets on which ASTRAL-III or RAxML fails to complete, either due to limited running time (for ASTRAL-III) or limited memory (for RAxML). In practice, the computational requirements for ML analyses can be very large, even on 'small' numbers of species, when using genome-scale data; for example, the Avian phylogeomics project with whole genomes for 48 birds used 1TB of memory and took more than 200 CPU years to complete. Thus, TreeMerge-fast could make it computationally

feasible for researchers with limited resources to analyze large multi-locus datasets, enabling 'phylogenomics for all'.

Our study suggests several directions for future research. For example, while we examined four model conditions (two levels of ILS and two types of sequence data), a more extensive study should evaluate performance on biological datasets as well as datasets simulated under different model conditions (especially, considering other sources of gene tree discord). Variations to the pipeline should be explicitly tested; for example, the dissimilarity matrix, the subset decomposition, and the spanning tree on the subsets could all be computed using different approaches; in particular, variants that ensure statistical consistency for TreeMerge-fast (as discussed in Theorem 9) may be useful to evaluate. Explicit testing of robustness to the starting tree (which impacts the subset decomposition and the spanning tree) is another important direction for future work. It is also worth evaluating this divide-and-conquer approach combined with iteration, which has been used successfully in similar applications (Liu *et al.*, 2009, 2012; Mirarab *et al.*, 2015; Nelesen *et al.*, 2012), and may result in improved accuracy (and robustness to the starting tree), at an increase in running time. We tested the divide-and-conquer framework with ASTRAL-III and RAxML, but other species tree estimation methods could be explored. Other extensions include evaluating the divide-and-conquer framework in a distributed-memory computing environment to scale Bayesian methods, such as StarBEAST-2, to larger datasets. Finally, Zhang *et al.* (2018b) recently presented constrained-INC, a new algorithm for merging leaf-disjoint trees in the context of gene tree estimation, and Le *et al.* (2019) implemented and evaluated variants of the algorithm on simulated data within a similar pipeline. It would be interesting to see whether divide-and-conquer using constrained-INC is suitable for species tree estimation.

Acknowledgements

The authors thank Sarah Christensen, William Gropp, Thien Le, Luay Nakhleh, Mike Nute, Srilakshmi Pattabiraman, Marc Snir, Pranjal Vachaspati and four anonymous reviewers for helpful comments that led to improvements in the quality of this work.

Funding

This work was supported by the U.S. National Science Foundation [Award No. CCF-1535977] to T.W. E.K.M. was supported by the NSF Graduate Research Fellowship [Award No. DGE-1144245] and the Ira and Debra Cohen Graduate Fellowship in Computer Science. Computational experiments were performed on Blue Waters. This research is part of the Blue Waters sustained-petascale computing project, which is supported by the NSF [Award Nos. OCI-0725070 and ACI-1238993] and the state of Illinois. Blue Waters is a joint effort of the University of Illinois at Urbana-Champaign and its National Center for Supercomputing Applications.

Conflict of Interest: none declared.

References

Allman,E.S. *et al.* (2018) Species tree inference from gene splits by unrooted STAR methods. *IEEE/ACM Trans. Comput. Biol. Bioinf.*, **15**, 337–342.

Bansal,M. and Eulenstein,O. (2013) Algorithms for genome-scale phylogenetics using gene tree parsimony. *IEEE/ACM Trans. Comput. Biol. Bioinf.*, **10**, 939–956.

Bayzid,M.S. and Warnow,T. (2018) Gene tree parsimony for incomplete gene trees: addressing true biological loss. *Algorithms Mol. Biol.*, **13**, 1.

Boussau,B. *et al.* (2013) Genome-scale coestimation of species and gene trees. *Genome Res.*, **23**, 323–330.

Bryant,D. and Waddell,P. (1998) Rapid evaluation of least-squares and minimum-evolution criteria on phylogenetic trees. *Mol. Biol. Evol.*, **15**, 1346.

Fletcher,W. and Yang,Z. (2009) INDELible: a flexible simulator of biological sequence evolution. *Mol. Biol. Evol.*, **26**, 1879–1888.

Jarvis,E.D. *et al.* (2014) Whole-genome analyses resolve early branches in the tree of life of modern birds. *Science*, **346**, 1320–1331.

Kingman,J.F.C. (1982) The coalescent. *Stoch. Process. Appl.*, **13**, 235–248.

Kruskal,J.B. (1956) On the shortest spanning subtree of a graph and the traveling salesman problem. *Proc. Am. Math. Soc.*, **7**, 48–50.

Kubatko,L. and Degnan,J. (2007) Inconsistency of phylogenetic estimates from concatenated data under coalescence. *Syst. Biol.*, **56**, 17–24.

Lai,H. *et al.* (2012) Inferring duplications, losses, transfers and incomplete lineage sorting with nonbinary species trees. *Bioinformatics*, **28**, i409–i415.

Le,T. *et al.* (2019) Using INC within divide-and-conquer phylogeny estimation. In: *6th International Conference on Algorithms for Computational Biology, AlCoB 2019, May 28–30, 2019*, Berkeley, CA, USA, in press.

Lefort,V. *et al.* (2015) FastME 2.0: a comprehensive, accurate, and fast distance-based phylogeny inference program. *Mol. Biol. Evol.*, **32**, 2798–2800.

Liu,K. *et al.* (2009) Rapid and accurate large-scale coestimation of sequence alignments and phylogenetic trees. *Science*, **324**, 1561–1564.

Liu,K. *et al.* (2012) SATé-II: very fast and accurate simultaneous estimation of multiple sequence alignments and phylogenetic trees. *Syst. Biol.*, **61**, 90–106.

Liu,L. and Yu,L. (2011) Estimating species trees from unrooted gene trees. *Syst. Biol.*, **60**, 661–667.

Maddison,W.P. (1997) Gene trees in species trees. *Syst. Biol.*, **46**, 523–536.

Mallo,D. *et al.* (2016) SimPhy: phylogenomic simulation of gene, locus, and species trees. *Syst. Biol.*, **65**, 334–344.

Meiklejohn,K.A. *et al.* (2016) Analysis of a rapid evolutionary radiation using ultraconserved elements: evidence for a bias in some multispecies coalescent methods. *Syst. Biol.*, **65**, 612–627.

Mirarab,S. and Warnow,T. (2015) ASTRAL-II: coalescent-based species tree estimation with many hundreds of taxa and thousands of genes. *Bioinformatics*, **31**, i44–i52.

Mirarab,S. *et al.* (2014a) ASTRAL: genome-scale coalescent-based species tree estimation. *Bioinformatics*, **30**, i541–i548.

Mirarab,S. *et al.* (2014b) Statistical binning enables an accurate coalescent-based estimation of the avian tree. *Science*, **346**, 1250463.

Mirarab,S. *et al.* (2015) PASTA: ultra-large multiple sequence alignment for nucleotide and amino-acid sequences. *J. Comput. Biol.*, **22**, 377–386.

Mirarab,S. *et al.* (2016) Evaluating summary methods for multi-locus species tree estimation in the presence of incomplete lineage sorting. *Syst. Biol.*, **65**, 366–380.

Molloy,E.K. and Warnow,T. (2018a) NJMerge: a generic technique for scaling phylogeny estimation methods and its application to species trees. In: Blanchette M. and Ouangraoua A. (eds.) *Comparative Genomics. RECOMB-CG 2018. Lecture Notes in Computer Science*. Vol. **11183**, Springer, Cham.

Molloy,E.K. and Warnow,T. (2018b) Statistically consistent divide-and-conquer pipelines for phylogeny estimation using NJMerge. *Algorithms Mol. Biol.*, in press.

Molloy,E.K. and Warnow,T. (2018c) To include or not to include: the impact of gene filtering on species tree estimation methods. *Syst. Biol.*, **67**, 285–303.

Nakhleh,L. (2013) Computational approaches to species phylogeny inference and gene tree reconciliation. *Trends Ecol. Evol.*, **28**, 719–728.

Nelesen,S. *et al.* (2012) DACTAL: divide-and-conquer trees (almost) without alignments. *Bioinformatics*, **28**, i274–i282.

Price,M.N. *et al.* (2010) FastTree 2—approximately maximum-likelihood trees for large alignments. *PLOS One*, **5**, 1–10.

Prum,R.O. *et al.* (2015) A comprehensive phylogeny of birds (Aves) using targeted next-generation DNA sequencing. *Nature*, **526**, 569–573.

Robinson,D. and Foulds,L. (1981) Comparison of phylogenetic trees. *Math. Biosci.*, **53**, 131–147.

Roch,S. (2006) A short proof that phylogenetic tree reconstruction by maximum likelihood is hard. *IEEE/ACM Trans. Comput. Biol. Bioinf.*, **3**, 92–94.

Roch,S. *et al*. (2019) Long-branch attraction in species tree estimation: inconsistency of partitioned likelihood and topology-based summary methods. *Syst. Biol*., **68**, 281–297.

Saitou,N. and Nei,M. (1987) The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Mol. Biol. Evol*., **4**, 406–425.

Stamatakis,A. (2014) RAxML version 8: a tool for phylogenetic analysis and post-analysis of large phylogenies. *Bioinformatics*, **30**, 1312–1313.

Stamatakis,A. and Aberer,A.J. (2013) Novel parallelization schemes for large-scale likelihood-based phylogenetic inference. In: *2013 IEEE 27th International Symposium on Parallel and Distributed Processing, IPDPS 2013, May 20–24, 2013*, Boston, MA, USA, pp. 1195–1204.

Steel,M. (1992) The complexity of reconstructing trees from qualitative characters and subtrees. *J. Classif*., **9**, 91–116.

Steel,M. (1994) Recovering a tree from the leaf colourations it generates under a Markov model. *Appl. Math. Lett*., **7**, 19–24.

Sukumaran,J. and Holder,M.T. (2010) DendroPy: a Python library for phylogenetic computing. *Bioinformatics*, **26**, 1569–1571.

Swenson,M. *et al*. (2012) SuperFine: fast and accurate supertree estimation. *Syst. Biol*., **61**, 214–227.

Swofford,D.L. (2019) PAUP* (*Phylogenetic Analysis Using PAUP). http://phylosolutions.com/paup-test/.

Tavaré,S. (1986) Some probabilistic and statistical problems in the analysis of DNA sequences. *Lect. Math. Life Sci*., **17**, 57–86.

The Quest for Orthologs Consortium (2014) Big data and other challenges in the quest for orthologs. *Bioinformatics*, **30**, 2993–2998.

Tofigh,A. *et al*. (2011) Simultaneous identification of duplications and lateral gene transfers. *IEEE/ACM Trans. Comput. Biol. Bioinf*., **8**, 517–535.

Vachaspati,P. and Warnow,T. (2015) ASTRID: accurate species trees from internode distances. *BMC Genomics*, **16**, S3.

Warnow,T. (2017) *Computational Phylogenetics: An Introduction to Designing Methods for Phylogeny Estimation*. Cambridge University Press, Cambridge, UK.

Warnow,T. *et al*. (2001) Absolute convergence: true trees from short sequences. In: *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2001, January 7–9, 2001*, Washington, DC, USA, pp. 186–195.

Wickett,N.J. *et al*. (2014) Phylotranscriptomic analysis of the origin and early diversification of land plants. *Proc. Natl. Acad. Sci. USA*, **111**, E4859–E4868.

Zhang,C. *et al*. (2018a) ASTRAL-III: polynomial time species tree reconstruction from partially resolved gene trees. *BMC Bioinformatics*, **19**, 153.

Zhang,Q.R. *et al*. (2018b) New absolute fast converging phylogeny estimation methods with improved scalability and accuracy. In: *18th International Workshop on Algorithms in Bioinformatics, WABI 2018, August 20–22, 2018, Helsinki, Finland*, pp. 8:1–8:12.