Scaling species tree estimation methods to large datasets using NJMerge

Erin Molloy and Tandy Warnow

{emolloy2, warnow}@illinois.edu
University of Illinois at Urbana Champaign

2018 Phylogenomics Software Symposium
August 17, 2018
Large-scale Phylogeny Estimation is computationally intensive

- “A New View of the Tree of Life” [Hug et al., 2016]
- 3,083 genomes
- 2,596 AA positions – from concatenating 16 protein alignments
- RAxML with 156 bootstrap replicates
- 3,840 CPU hours on the CIPRES supercomputer
NJMerge can enable species tree methods (ASTRAL-III, SVDquartets, and RAxML) to analyze large and/or heterogeneous datasets using a single compute node with

- 16 cores
- 64 GB of physical memory
- 48 hours maximum wall-clock time

without sacrificing accuracy.
Divide-and-Conquer Methods

Full species set

Decompose species into overlapping subsets

[e.g., Huson et al., 1999; Nelesen et al., 2012]
Divide-and-Conquer Methods

- Decompose species into \textit{overlapping} subsets
- Build trees on each subset

[e.g., Huson et al., 1999; Nelesen et al., 2012]
Divide-and-Conquer Methods

- Decompose species into \textit{overlapping} subsets
- Build trees on each subset
- Estimate Supertree

[e.g., Huson et al., 1999; Nelesen et al., 2012]
Divide-and-Conquer Methods

[Image of a diagram showing the process of decomposing species into overlapping subsets, iterating, building trees on each subset, and estimating a supertree.]

[e.g., Huson et al., 1999; Nelesen et al., 2012]
Subset trees can be estimated using a highly accurate (but computationally intensive) method and then combined using a supertree method.
Divide-and-Conquer Methods

Decompose species into overlapping subsets

Iterate

Build trees on each subset

Estimate Supertree

Full species set

Tree on full species set

[e.g., Huson et al., 1999; Nelesen et al., 2012]
Divide-and-Conquer Methods

Full species set

Decompose species into overlapping subsets

Iterate

Build trees on each subset

Tree on full species set

NP-Hard

Estimate Supertree

[Steel, 1992; Jiang et al., 2001; Bansal et al., 2010]
Divide-and-Conquer Methods

- Decompose species into **overlapping** subsets
- Iterate
- Build trees on each subset
- Estimate Supertree

Full species set

Tree on full species set

[e.g., Huson et al., 1999; Nelesen et al., 2012]
New Divide-and-Conquer Strategy

Decompose species into disjoint subsets

Iterate

Build trees on each subset

Merge disjoint trees
New Divide-and-Conquer Strategy

Subset trees can be estimated using a highly accurate (but computationally intensive) method and then merged together using a fast (but potentially less accurate) method.
New Divide-and-Conquer Strategy

Full species set

Decompose species into disjoint subsets

Iterate

Build trees on each subset

Merge disjoint trees

Tree on full species set
Merging disjoint trees

“Merging” Joining with single edge

A
B C
D +
E
F G
H
A D
B E
F C
G H

Molloy & Warnow
NJMerge 15/48
Merging disjoint trees

Given a set $\mathcal{T}$ of pairwise disjoint trees with combined leaf set $S$, we seek a **compatibility supertree** for $\mathcal{T}$ that is close to the true tree on $S$.

NJMerge attempts to do this **by using a dissimilarity matrix $D$**.
NJMerge is a polynomial-time extension of Neighbor-Joining [Saitou and Nei, 1987] that takes a set of trees (on pairwise disjoint subsets of the leaf set) as input that operate as constraints on the output tree.

Input:
- Dissimilarity matrix $D$ on species set $S$
- Set of $k$ unrooted binary trees $\mathcal{T} = \{ T_1, T_2, \ldots, T_k \}$ on pairwise disjoint subsets of the species set $S$

Output:
- Unrooted binary tree $T$ on the species set $S$ that agrees with every tree in $\mathcal{T}$
NOTE: Distance matrix is additive for
(((A, B), (C, D)), E, (F, (G, H)));

\[ D^{ij} \]
NOTE: Distance matrix is additive for $(((A, B), (C, D)), E, (F, (G, H)))$;
NJ is an agglomerative technique that joins pairs of nodes at each iteration and reduces the number of remaining nodes by one.

NJMerge differs from NJ in that it only accepts a siblinghood proposal \((x, y)\) if and only if making \(x\) and \(y\) siblings does not violate the pairwise compatibility of the set \(\mathcal{T}\) of constraint trees.
Determining the compatibility of a set of unrooted phylogenetic trees is NP-complete [Steel, 1992; Warnow, 1994].

Thus, NJMerge uses a polynomial-time heuristic:

- Update constraint trees based on siblinghood proposal \((x, y)\)
- Test whether the set of constraint trees that contain leaf \((x, y)\) are compatible in polynomial time by rooting these trees at leaf \((x, y)\) [Aho et al., 1981]

Not every constraint tree will contain the leaf \((x, y)\), so this does not a guarantee that the set of \(k\) constraint trees is compatible.
Determining the compatibility of a set of unrooted phylogenetic trees is NP-complete [Steel, 1992; Warnow, 1994].

Thus, NJMerge uses a polynomial-time heuristic:

- Update constraint trees based on siblinghood proposal \((x, y)\)
- Test whether the set of constraint trees that contain leaf \((x, y)\) are compatible in polynomial time by rooting these trees at leaf \((x, y)\) [Aho et al., 1981]

Not every constraint tree will contain the leaf \((x, y)\), so this does not guarantee that the set of \(k\) constraint trees is compatible.

It is possible for NJMerge to fail.
Advantages

- No supertree estimation
- Polynomial Time
- Parallel, e.g., each siblinghood proposal can be evaluated in parallel

Disadvantages

- Can fail to return a tree
Species Tree Estimation

Gene 1  Gene 2  ...  Gene k

Species

Concatenation

Analyze separately

Summary Method
Using NJMerge in species tree estimation pipeline

1. Decompose full taxon set into disjoint subsets.
2. Build species trees on each subset.
3. Iterate.
4. Run NJMerge on full taxon set.

Species tree on full taxon set.
Using NJMerge in a pipeline for species tree estimation

- How should the distance matrix be computed?
- How large should the maximum subset size be?
- How should subsets be created?
- Which species tree method should be used on subsets?
Simulated Datasets

**Dataset Size**
- 100 species; 25, 100, 1000 genes
- 1000 species; 1000 genes
- Gene (alignment) length varied from 300 to 1500 bp

**Level of Incomplete Lineage Sorting (ILS)**
- Moderate: 8-10% AD
- Very high: 68-69% AD

where AD is the “Average (RF) Distance” between the true species tree and the tree gene trees.

**Sequence Type**
- Exons: lower phylogenetic signal (not shown)
- Introns: greater phylogenetic signal
How does error in the estimated distance matrix impact NJMerge?

NOTE: NJMerge failed to return a tree on 1 dataset with 100 taxa, 25 genes, and very high ILS; average is on datasets for which both methods completed.
Evaluation

Subset trees are estimated using ASTRAL-III, SVDquartets, and unpartitioned concatenation using RAxML.

Metrics

- Robinson-Foulds (RF) distance between the true and the estimated species trees
- Running time

All methods limited to

- 48 hours wall-clock time
- one compute node with 16 cores
- 64 GB of physical memory
Running time for pipeline using NJMerge

\[ T = (T_1^{Method} + \ldots + T_k^{Method}) + T^{NJMerge} \]

where

- \( k \) = number of subsets
- \( T_i^{Method} \) = time to compute tree on subset \( i \) for given method
- \( T_{NJMerge} \) = time to run NJMerge
Gene tree summary method

Finds optimal solution to search problem within a constrained search space

Computationally intensive when

- Constrained search space is large, e.g., large datasets with high levels of gene tree heterogeneity due to ILS or gene tree estimation error
NOTE: ASTRAL completed on 16/20 datasets with 1000 taxa and very high ILS; NJMerge+ASTRAL completed on 20/20 of these datasets; box plots show all datasets on which at least one method completed.
NOTE: ASTRAL completed on 16/20 datasets with 1000 taxa and very high ILS; NJMerge+ASTRAL completed on 20/20 of these datasets; bar graphs show all datasets on which at least one method completed.
SVDquartets
[Chifman and Kubatko, 2014]

- Site-based coalescent method
- Use the concatenated alignment to estimate quartet trees and then combine the quartet trees
- Computationally intensive when
  - Number of species is large
SVDquartets and NJMerge

![Species Tree Error vs Level of ILS](image)

**100 species, 1000 introns**

Moderate Very High

**1000 species, 1000 introns**

0.0 0.1 0.2

**Species Tree Error**

**Level of ILS**

[Graph showing box plots for SVDquartets and NJMerge+SVDquartets]

**NOTE:** SVDquartets ran on 0/40 datasets with 1000 taxa due to segmentation faults; NJMerge+SVDquartets completed on 40/40 of these datasets; box plots show all datasets on which at least one method completed.
SVDquartets and NJMerge

### 100 species, 1000 introns
- **Level of ILS:**
  - Moderate
  - Very High
- **Running Time (m):**
  - SVDquartets: 60
  - NJMerge+SVDquartets: 20

### 1000 species, 1000 introns
- **Level of ILS:**
  - Moderate
  - Very High
- **Running Time (m):**
  - SVDquartets: 400
  - NJMerge+SVDquartets: 300

**NOTE:** SVDquartets ran on 0/40 datasets with 1000 taxa due to segmentation faults; NJMerge+SVDquartets completed on 40/40 of these datasets; bar graphs show all datasets on which **at least one** method completed.
ML tree estimation on unpartitioned concatenated alignment

Not statistically consistent under MSC model

Computationally intensive when

- Number of species is large
- Number of unique site patterns in alignment is large, i.e., cannot effectively compress the alignment
RAxML and NJMerge

NOTE: RAxML completed on 1/40 datasets with 1000 taxa due to “Out of Memory” errors; NJMerge+RAxML completed on 40/40 of these datasets; box plots show all datasets on which at least one method completed.
RAxML and NJMerge

![Bar graphs showing running time (in minutes) for different levels of ILS and dataset sizes for RAxML and NJMerge+RAxML.]

**Level of ILS**

- **Moderate**
- **Very High**

**Dataset Sizes**

- 100 species, 1000 introns
- 1000 species, 1000 introns

**Running Time (m)**

- RAxML
- NJMerge+RAxML

**NOTE**: RAxML completed on 1/40 datasets with 1000 taxa due to “Out of Memory” errors; NJMerge+RAxML completed on 40/40 of these datasets; bar graphs show all datasets on which at least one method completed.
NJMerge is a generic technique for scaling phylogeny estimation methods to large and/or heterogeneous datasets.

In our experiments, NJMerge

- improved running time without sacrificing accuracy
- enabled all three methods to run on large and/or heterogeneous when constrained by memory (64GB) and running time (48 hours)
- had a low failure rate (< 1%)
Which species tree method do I use on datasets with 1000 taxa and 1000 genes?

It depends on the dataset!
Which species tree method do I use on datasets with 1000 taxa and 1000 genes?

It depends on the dataset!
NJMerge enables species tree methods (ASTRAL-III, SVDquartets, and RAxML) to analyze large and/or heterogeneous datasets using a single compute node with

- 16 cores
- 64 GB of physical memory
- 48 hours maximum wall-clock time

without sacrificing accuracy.

How to best run NJMerge depends on the dataset!

Several algorithmic choices are left to the discretion of the user.
Funding

This work was supported by the National Science Foundation

- Graduate Research Fellowship Program (DGE-1144245)
- Blue Waters Sustained-Petascale Computing Project (OCI-0725070 and ACI-1238993)
- Graph-Theoretic Algorithms to Improve Phylogenomic Analyses (CCF-1535977)

and the Ira & Debra Cohen Graduate Fellowship in Computer Science.
Use NJMerge!


Available on Github: github.com/ekmolloy/njmerge


