Comparing Phylogenetic Tree-Building Methods in R: A How-To Guide and Tanglegrams

Branae Craveiro

December 7, 2025

Video Link: https://youtu.be/eE3CqlUvluk

Contents

Purpose Statement:

This guide will demonstrate how to use different phylogenetic building methods in R (distance-based, parsimony, maximum likelihood), compare their results when applied to the same dataset, and how to visualize changes between trees through tanglegrams.

Background:

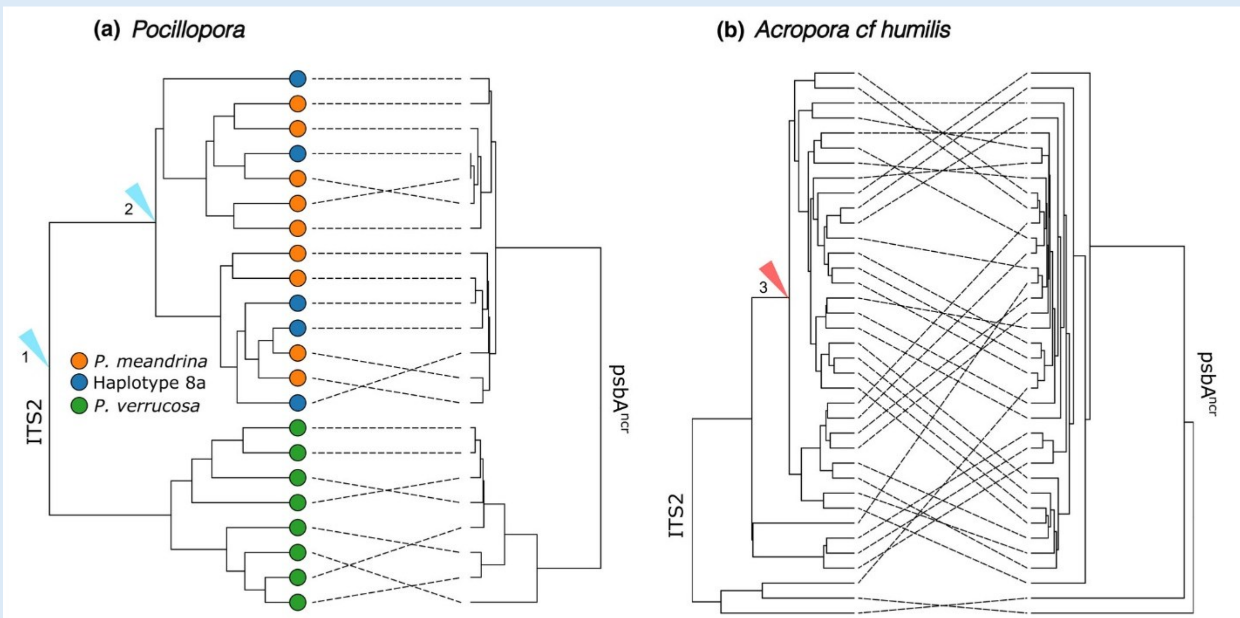
This project aims to demonstrate how different phylogenetic tree building methods (distance-based, parsimony, maximum likelihood) generate different branching patterns when applied to the same dataset. Phylogenetic trees are fundamental to biology because they reveal evolutionary relationships among species, helping researchers trace how traits, genes, and species change over time (1). Although each method seeks to reconstruct evolutionary history, they rely on different assumptions on how evolution proceeds (1) (summarized in *Table 1*): distance-based approaches infer relationships by converting sequence differences into genetic distances assuming that more similar sequences share a more recent ancestor (1,2); parsimony selects the tree requiring minimal character-state changes (1,3); and maximum likelihood identifies the tree that makes the observed DNA sequences most probable under a nucleotide substitution model (1,3). Since each approach weighs genetic information differently, they can yield trees with different branching orders or levels of resolution. Understanding these differences is important because phylogenetic trees inform biological conclusions, from species classification to conservation priorities to hypothesis about trait evolution. When methods disagree, researchers must be able to explain why the structures differ and which approach is most appropriate for their specific question. Recognizing the limitations and assumptions behind each method supports more transparent, and reproducible interpretations.

Table 1: Comparison of Major Phylogenetic Tree-Building Methods (1–3)

Method	R Package	Core Principle	How it Works	Key Assumptions	Clarifying Definition
Distance-based (e.g. Neighbor-Joining)	Ape	Groups taxa based on overall genetic similarity	Computes pairwise genetic distances between sequences and builds a tree that best fits those distances.	More similar sequences have a more recent ancestry.	<u>Genetic distance</u>: numerical measure of how different two DNA sequences are.
Maximum Parsimony	Phangorn	Chooses the tree requiring the fewest evolutionary changes	Evaluates possible trees and selects the one that minimizes the total number of character-state changes.	Evolution tends to occur following the simplest path.	<u>Character-state change</u>: when nucleotide position differs between ancestors and descendants
Maximum Likelihood	Phangorn	Selects the tree that makes the observed DNA sequences most probable	Uses statistical models to evaluate how likely each possible tree is, given patterns of nucleotide substitutions.	A specified substitution model accurately describes how DNA evolves.	<u>Nucleotide substitution model:</u> mathematical model describing how likely one nucleotide is to change to another over time

Although phylogenetic trees existed long before molecular data, this project specifically analyzes DNA sequences, which require computational tools. A bioinformatic workflow makes it possible to align nucleotide sequences accurately (4), calculate genetic distances (2,3), and apply formal evolutionary models (2,3) in a consistent and reproducible way, ensuring that observed differences between trees reflect different methodological approaches rather than inconsistencies handling data. To compare how the tree-building methods differ in their inferred evolutionary relationships, this project uses a tanglegram, a side-by-side visualization between two trees with connections between shared taxa to highlight agreement and conflict between structures (see *Figure 1* for an example) (5,6). Together, these bioinformatical approaches make it possible to process genetic data, generate phylogenetic trees under different analytical models, and visualize structural differences in a clear and reproducible way.

***Figure 1*: Example Tanglegram Visualizing Agreement vs. Conflict Among Phylogenetic Trees**



Example of a tanglegram (adapted from Marzonie et al. 2024) used to compare two phylogenetic trees. In the original study, the authors assessed the resolution power of two molecular markers by comparing coral algal symbiont identities inferred from each marker (7). *Parallel linkages indicate strong agreement between the tree structure in the left tanglegram, whereas numerous crossing connections in the right tanglegram indicate conflicting relationships. This figure helps demonstrate how tanglegrams visually convey both agreement and disagreement between phylogenetic trees.*

Methods:

For this project multiple R packages were used to conduct and compare phylogenetic trees built using different analytical methods. These analyses rely on three major phylogenetic approaches summarized in Table 1 (distance-based, parsimony, and maximum likelihood), which differ in their underlying assumptions and influence the resulting tree structure (1). The msa package (4) was used to perform multiple sequence alignment to ensure that homologous sites were properly lined up for downstream tree construction. From this alignment, ape (2) was used to construct a distance-based tree (neighbor-joining), providing a baseline phylogeny built from overall genetic similarity. To evaluate how alternative evolutionary assumptions affect tree structure, phangorn (3) was used to implement character-based approaches, like maximum parsimony and maximum likelihood. These methods were chosen because each represents a major phylogenetic framework and comparing them directly is central to the project's goal of assessing how methodological differences influence assumed evolutionary relationships.

To visualize structural differences across methods, phytools (5) was used to generate a tanglegram, which places two trees side by side and links identical taxa to so changes in branching order can be easily compared (5,6). This visualization makes it easier to see how branching patterns different across methods, Together, these packages provide a consistent workflow for comparing multiple phylogenetic approaches on the same dataset.

Dataset:

The dataset consists of *psbA* gene sequences from rice and other grasses retrieved from the NCBI Nucleotide database. The *psbA* gene encodes a core protein of photosystem II involved in photosynthesis and is widely used as a molecular marker in plant phylogenetic due to its moderate conservation, which makes it effective for resolving species-level relationships (8).

The dataset (*Table 2*) includes a mix of closely related rice cultivars, other *Oryza* species, and more distantly related grasses (rice, wheat, barley, and oats are all part of the cereal grass family Poaceae). This combination of taxa provides both closely related and more distantly sequences, which allows distance-based, parsimony, and maximum-likelihood methods to be compared in how they resolve small versus large phylogenetic splits. The dataset is small enough to be manageable for a step-by-step analysis, yet large enough to produce distinct branching patterns under different evolutionary assumptions, making it well suited for evaluating how tree-building methods differ in the phylogenetic structure they generate.

Table 2: *psbA* Gene Accession Numbers for Selected Poaceae Species

Group	Species/Cultivar	Common Name	Accession Number
Oryza sativa cultivars	Liujiang	Rice	GQ150262.1
	Fusui	Rice	GQ150261.1
	Bairizoa	Rice	GQ150257.1
Other *Oryza* species	*Oryza* minuta	Wild rice species	KP864581.1
	Oryza officinalis	Wild rice species	KP864576.1
	Oryza australiensis	Wild rice species	KP864570.1
	Oryza eichingeri	Wild rice species	KP864571.1
Other cereal grasses	*Triticum* aestivum	Wheat	MH346224.1
	Hordeum vulgare	Barley	HQ600125.1
	Avena sativa	Oat	KJ477768.1
Outgroup	*Zea* mays	Corn	AF543684.1

Manual

Part 1: Prepare Sequences

Step 1: Install and load necessary packages in R

1. Install Bioconductor and the msa package
   The msa package (installed through Bioconductor) is used for multiple sequence alignment.

2. Install phylogenetics packages
   ape, phangorn, and phytools provide functions for building and comparing phylogenetic trees.

3. Load all packages into the R session
   Once installed, the libraries must be loaded before running the analysis.

```
#Part 1: Prepare Sequences
#Step 1: Install and load necessary packages in R
#install necessary packages

#Multiple Sequence Alignment package from Bioconductor
if (!require("BiocManager", quietly = TRUE))
  install.packages("BiocManager")

BiocManager::install("msa")

#Analyses of Phylogenetics and Evolution
install.packages("ape")

#Phylogenetic Reconstruction and Analysis
install.packages("phangorn")

#Phytools (for tanglegram)
install.packages("phytools")

#retrieving packages from library
library(ape)
library(msa)
library(phangorn)
library(phytools)
```

Step 2: Download sequences directly from GenBank using read.GenBank()

1. Create a character vector of chosen accession numbers.
   Accession numbers were first gathered from the NCBI Nucleotide database by searching for the target gene and organism (e.g. using a search query like "psbA AND Oryza [Organism] AND gene"). Once the accession numbers are selected, create a character vector that lists the accession numbers.

```
#step 2: Download sequences directly from GenBank

#make a character vector of chosen accession numbers
my_accession_numbers <- c("GQ150262.1", "GQ150261.1", "GQ150257.1", "KP864581.1",
                          "KP864576.1", "KP864570.1", "KP864571.1", "MH346224.1",
                          "HQ600125.1", "KJ477768.1", "AF543684.1")
```

2. Downloaded sequences into R using the read.GenBank() function from the ape package.
   read.GenBank() retrieves sequences from directly from GenBank and returns them as a list of DNA sequences, meaning there is no need to manually download or upload FASTA files.

```
#download FASTA sequences from GenBank
sequences <- read.GenBank(my_accession_numbers,
                          seq.names = my_accession_numbers,
                          #names given to each sequence (default is accession numbers used)
                          species.names = TRUE,
                          #returns species names of each sequence
                          as.character = TRUE)
                          #returns each nucleotide as a character vector
```

3. Assign unique species/cultivar names manually using names()

   By default, read.GenBank() assigns tip labels as accession numbers. Since some of our sequences are from the same species (e.g., Oryza sativa cultivars), duplicated labels would break downstream parsimony and likelihood analyses. To avoid this, we manually assign unique names for each sequence using names(). This ensures that each sequence has a unique label that is informative for plotting and tree building.

```
#By default, tip labels are the accession numbers, so we'll replace them with species/cultivar
#names. Duplicated names (our Oryza sativa cultivars) break parsimony and likelihood analyses,
#so a basic approach to fix this issue is to manually assign unique names to each sequence

names(sequences) <- c(
  "GQ150262.1" = "Oryza_sativa_Liujiang",
  "GQ150261.1" = "Oryza_sativa_Fusui",
  "GQ150257.1" = "Oryza_sativa_Bairizoa",
  "KP864581.1" = "Oryza_minuta",
  "KP864576.1" = "Oryza_officinalis",
  "KP864570.1" = "Oryza_australiensis",
  "KP864571.1" = "Oryza_eichingeri",
  "MH346224.1" = "Triticum_aestivum",
  "HQ600125.1" = "Hordeum_vulgare",
  "KJ477768.1" = "Avena_sativa",
  "AF543684.1" = "Zea_mays")
```

Step 3: Align homologous regions using msa package

1. Convert sequence lists into continuous strings

   The Read.GenBank() function returns sequences as a list of character vectors (e.g. "A","T","G","C"...). However, msa() needs a single string (e.g "ATGC"). To do this, sapply() is used to apply the paste(..., collapse = "") function on each sequence in the list, which joins all the individual letters into one string.

```
#step 3: Align homologous regions using msa package

#sequences is a list of character vectors like c("A","T","G","C",...)
#need to make them one character string per sequence for msa to work

character_sequences <- sapply(sequences,
                       function(x) paste(x, collapse=""))

#sapply applies a function to each character in a list (so each "A","T",...)
#x represents all the vector of nucleotides ("A","T",...) in each sequence
       #paste(x, collapse="") joins all letters together into one string
```

2. Convert strings into a DNAStringSet object using DNAStringSet()

   The msa() function requires sequences to be stored as a DNAStringSet object. Use DNAStringSet() from the Biostrings package (installed automatically with msa) to convert the character strings into the appropriate format.

3. Reassign names to the DNAStringSet object using names()

   The msa() function sometimes resets sequence names to the original accession numbers when converting sequences internally, which causes the neighbor-joining tree built with ape to show the

GenBank species names instead of the species or cultivar names we assigned. To keep the species names consistent across all our trees, we reassign the manually set species/cultivar names to the DNAStringSet object using names() before running the alignment. This ensures that the same species names appear in all downstream trees.

4. Perform the multiple sequence alignment using msa()
   With the sequences in a DNAStringSet, run msa() function to generate the multiple sequence alignment of homologous regions.

```
#convert the character list to a DNAStringset (format msa likes)
biostrings_sequences <- DNAStringSet(character_sequences)

#by default, msa sometimes converts seq. names back to original accession numbers,
#which makes our ape NJ tree have the GenBank names, so for consistency we can
#reassign the names we manually set earlier
names(biostrings_sequences) <- names(sequences)

#run msa
alignment <- msa(biostrings_sequences)
```

Part 2: Construct Phylogenetic Trees

Step 1: Create a Neighbor-joining tree using distance matrices from ape

1. Convert aligned sequences into DNAbin format using msaConvert().
   Ape requires sequences to be in DNAbin format. Use msaConvert() to transform the aligned sequences from a DNAStringSet object into DNAbin.

```
#Part 2: Construct Phylogenetic Trees
#step 1: Create a Neighbor-joining tree using distance matrices from ape

#now that our sequences are aligned, we need to convert the format into one ape can run

#Convert MSA to DNAbin
alignment_DNAbin <- msaConvert(alignment, type = "ape::DNAbin")
#bin = binary representation of nucleotides (A=1, C=2, G=3, T=4)
```

2. Calculate genetic distances using dist.dna().
   Compute pairwise differences between sequences with dist.dna(). For this project, the K80 model (default in ape) is used to measure genetic difference based on the assumption that different mutations have different rates of mutation (9) .

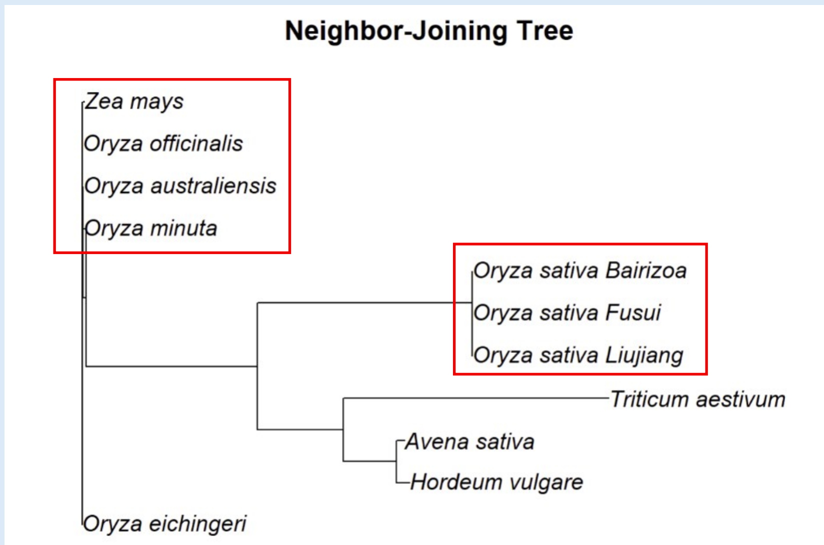
```
#calculate distances between sequences
dist_matrix_ape <- dist.dna(alignment_DNAbin, model = "K80")
#model tells R how to compare the DNA
#K80 is a statistical model that assumes equal base frequencies, but different types of
#mutations have different mutation rates
```

3. Construct the neighbor-joining tree using nj() and root the tree with root().
   Use the nj() function to build a neighbor-joining tree from the distance matrix. For the downstream tanglegram comparisons, each phylogenetic tree needs to be rooted so they all share the same baseline (6). root() places a "starting point" on the tree (a node that represents the earliest common ancestor of all taxa) which allows the tree to be compared in a consistent way (6).
4. Visualize the tree using plot().
   Visualize it with plot() to see the inferred evolutionary relationships (see Figure 2).

```
#neighbor-joining tree estimation
nj_tree_ape <- nj(dist_matrix_ape)

#root neighbor-joining tree so that Zea mays is the outgroup. Rooting fixes the
#position of the common ancestor so all trees share the same evolutionary orientation.
nj_tree_ape <- root(nj_tree_ape, outgroup = "Zea_mays", resolve.root = TRUE)

#plot tree
plot(nj_tree_ape,
     type = "phylogram",              #type of tree where branch lengths are
                                      #proportional to the amount of evolutionary change
     main = "Neighbor-Joining Tree")  #Title
```

Figure 2: Neighbor-Joining Tree (ape)

Neighbor-Joining tree constructed from aligned Poaceae sequences using genetic distance. This tree shows the overall clustering pattern but contains some unresolved (collapsed) internal nodes (red boxes). Internal nodes represent inferred ancestral branching points, and when they appear collapsed, they indicate low resolution or uncertainty in the relationships between sequences (1).



Step 2: Build a maximum parsimony tree using phangorn

1. Convert the aligned sequences into phyDat format using as.phyDat().
   Phangorn requires DNA sequences to be stored as a phyDAT object. Use as.phyDat() to convert the alignment into this format so that functions in the phangorn package can evaluate character-based changes across sites.

```
#step 2: Build a maximum parsimony tree using phangorn

#again we need to convert aligned sequences into a format phangorn can read

#Convert MSA to phyDAT
alignment_phangorn <- as.phyDat(alignment, type = "DNA")    #phyDAT = phylogenetic data
```

2. Generate a starting tree using NJ().
   Maximum parsimony searches need an initial tree to optimize from (1). Otherwise, phangorn would need to evaluate every possible structure, which would take a long time. Create a starting tree by computing a distance matrix with dist.ml() and building a quick neighbor-joining tree using NJ().

```
#create a Neighbor Joining tree (quick starting point for phangorn to build off of)
dist_matrix_phang <- dist.ml(alignment_phangorn)     # distance matrix using ML model
nj_tree_phang <- NJ(dist_matrix_phang)        # basic tree structure for maximum parsimony search
```

3. Run the maximum parsimony search using optim.parsimony().
   Use optim.parsimony() to refine the neighbor-joining tree. This function adjusts branch arrangements to find the tree that minimizes the total number of evolutionary changes across all sites (1,3).

```
#perform maximum parsimony tree search
mp_tree <- optim.parsimony(nj_tree_phang, alignment_phangorn)

#running phangorn's maximum parismony or maximum likelihood searches needs a starting tree,
#The NJ tree acts as a rough initial guess to begin optimizing from
#otherwise, it would have to test all possible trees, which would take a long time.
```

4. Root the tree using root() and visualize the resulting maximum parsimony tree using plot().
   Root the tree with Zea Mays as the outgroup using root() for consistent evolutionary orientation between trees and plot the maximum parsimony tree using plot() to view branching structure (see Figure 3).

```
#root maximum parismony tree so that Zea mays is the outgroup.
mp_tree <- root(mp_tree, outgroup = "Zea_mays", resolve.root = TRUE)

#plot maximum parismony tree
plot(mp_tree,
     type = "phylogram",              #type of tree where branch lengths are
                                      #proportional to the amount of evolutionary change
     main = "Maximum Parismony Tree") #Title
```

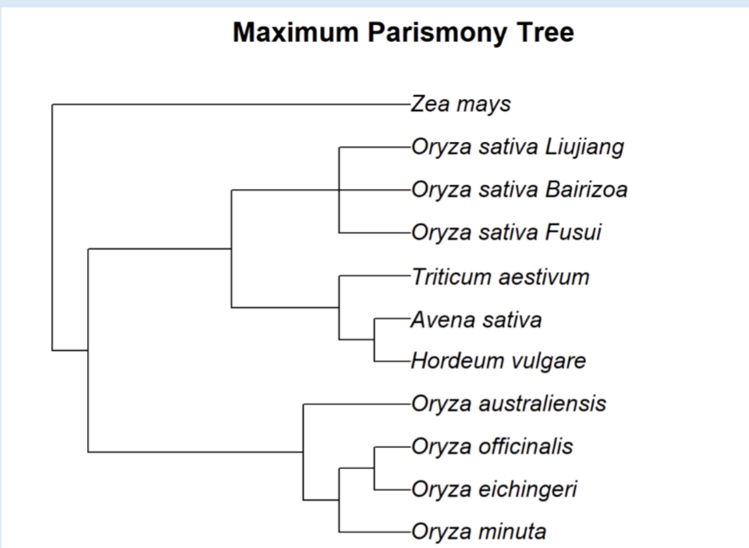


***Figure 3:* Maximum Parsimony Tree (phangorn)**

A Maximum Parsimony tree constructed using the phangorn package where the total number of evolutionary changes is minimized. This method produced a fully resolved branching pattern with no collapsed internal nodes, giving it the highest apparent resolution among the three tree-building approaches.

Step 3: create a maximum likelihood tree using phangorn

1. Bundle the starting Neighbor-Joining tree and alignment into a pml object using pml()
   Similar to creating a maximum parsimony tree, maximum likelihood also uses a neighbor joining tree (1). The first part of creating a maximum likelihood tree is gathering the materials needed to run the statistical model in one spot. We can do this using pml(), which combines the NJ tree created earlier, and the multiple sequence alignment into a single "analysis object" that stores the starting tree, branch lengths, nucleotide frequencies, and the chosen substitution model (3,10). By default, phangorn uses the GTR (General Time Reversible) model, which is widely used

in evolutionary biology because it allows each nucleotide substitution type to have its own rate and estimates the base frequencies from the sequences (1).

```
#step 3: build maximum likelihood tree using phangorn

#a maximum likelihood is a bit more complicated to build than our last two trees
#However you can think of the steps as first loading the tree & alignment into a “calculator”,
#and then tell the “calculator” to make the tree as statistically correct as possible.

#first we need to bundle our starting tree and our alignment together
#like our maximum parsimony tree, our maximum likelihood tree also needs a NJ starting tree
ml_bundle <- pml(nj_tree_phang, data = alignment_phangorn)

#pml() sets up the tree structure, branch lengths, substitution model (default = GTR),
#and the alignment all in a single object
#GTR (General Time Reversible) model is a flexible DNA-evolution model that allows each type
#of nucleotide substitution to occur at its own rate while also estimating the base frequencies.
#It’s widely used because it realistically models sequence evolution and makes maximum
#likelihood tree building mathematically workable.
```

2. Optimize the tree using optim.pml()
   optim.pml() procedure adjusts the tree structure and model parameters to maximize the probability of observing the given DNA alignment. These additional optimizations make the maximum likelihood tree more realistic and statistically supported compared to simpler tree-building methods:
   - *optNni: swap branches to fix incorrect tree shapes, explore new tree structures instead of being stuck with the neighbor-joining guess (10).*
   - *optGamma: estimate gamma rate variation, which accounts for some DNA sites mutating faster/slower than others (10).*
   - *optBf: estimate base frequencies (A/T/C/G proportions) from the data, without this, the model incorrectly assumes all bases are equally common (10).*

```
#now we make the tree as statistically likely as possible given the DNA
ml_tree <- optim.pml(
  ml_bundle,
  optNni = TRUE,   #allows the function to swap branches to fix incorrect tree shapes
  #this lets ML explore new tree structures instead of being stuck with the NJ guess
  optGamma = TRUE, #estimate gamma rate variation
  #accounts for some DNA sites mutating faster/slower than others, makes model more realistic.
  optBf = TRUE)    #estimate base frequencies (A/T/C/G proportions) from the data
  #without this, the model incorrectly assumes all bases are equally common
```

3. Root the tree with root() and visualize the final maximum likelihood tree using plot()
   optim.pml() returns a large object containing many statistical outputs, so the actual phylogenetic tree must be extracted using $tree. Use root() to place Zea mays as the outgroup, giving the tree the same ancestral reference point as the others. The maximum likelihood tree can then be displayed using plot() to visualize the most probable evolutionary relationships among the sequences (Figure 4).

```
#root maximum likelihood tree
ml_tree_rooted <- root(ml_tree$tree, outgroup = "Zea_mays", resolve.root = TRUE)

# We use ml_tree$tree instead of ml_tree because optim.pml() returns a big object
# that contains the tree plus extra statistical information. Rooting the whole object
# can make the tree draw weird, so we extract just the actual phylo tree (ml_tree$tree)

#plot maximum likelihood tree
plot(ml_tree$tree, type="phylogram", main="Maximum Likelihood Tree")
```

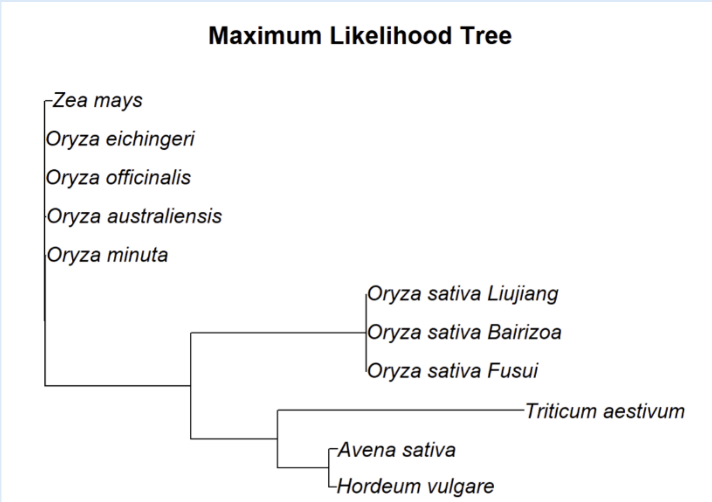


Figure 4: Maximum Likelihood Tree (phangorn)

Maximum Likelihood tree constructed using phangorn optimized under the GTR model with rate variation among nucleotides (1), a method that provides the statistically most probable tree given the sequence data. This tree shows improved structure within clades relative to the Neighbor-Joining tree but still contains a few unresolved internal nodes.

Part 3: Tanglegram Visualization

Step 1: Perform Tanglegram Analysis to Compare Tree Structures

1. Fix collapsed internal nodes using multi2di() so the trees can be compared correctly
   Sometimes trees have "collapsed" internal nodes, which show up as straight vertical lines instead of a normal branching point. When this happens, cophylo() cannot rotate the branches, and the tanglegram can look more tangled than it should. To fix this, we run multi2di() on each tree. This adds tiny branches to any collapsed areas, so the trees have proper structure. This prevents misleadingly high tanglegram disagreement.

```
#some internal nodes in the original trees were collapsed (shown as straight lines),

#which makes it hard for cophylo() to rotate the trees properly.
#this can make the tanglement score look higher than it really is.

#multi2di() adds tiny branches to any collapsed nodes, giving them proper structure
#This ensures that cophylo() can rotate trees correctly

nj_tree_ape <- multi2di(nj_tree_ape)
mp_tree <- multi2di(mp_tree)
ml_tree <- multi2di(ml_tree$tree)
```

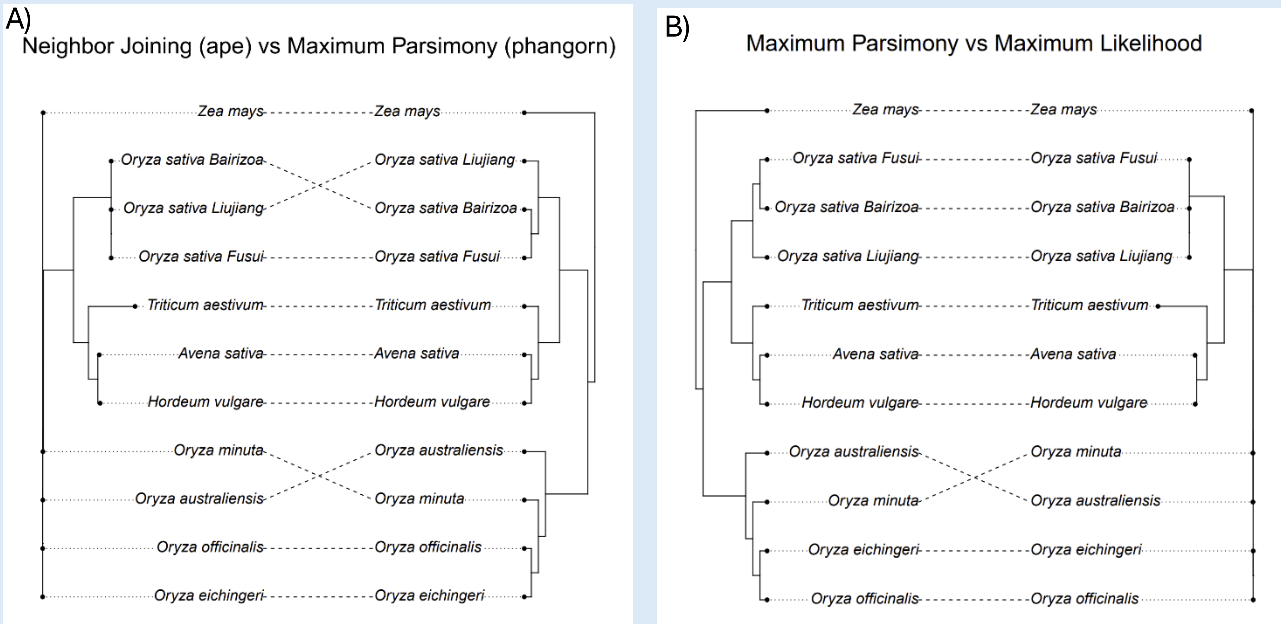
2. Use cophylo() to line up the two trees for comparison.
   Two trees can show the same evolutionary groups but still look different if their tips are in a different order. The cophylo() function rotates branches so matching clades line up on the left and right trees. This makes sure that any differences we see in the tanglegram come from the actual tree shapes, not just how the tips were arranged.

```
#align trees based on matching branching
co_phy_NJ_MP <- cophylo(nj_tree_ape, mp_tree, rotate = TRUE)
co_phy_MP_ML <- cophylo(mp_tree, ml_tree, rotate = TRUE)
co_phy_NJ_ML <- cophylo(mp_tree, ml_tree, rotate = TRUE)

#rotate = TRUE lets cophylo() rotate internal nodes to arrange the trees so they
#show the same clades in the same left/right positions whenever possible.
#This way, any differences you see come from the actual tree structure,
#not the order of tips.
```

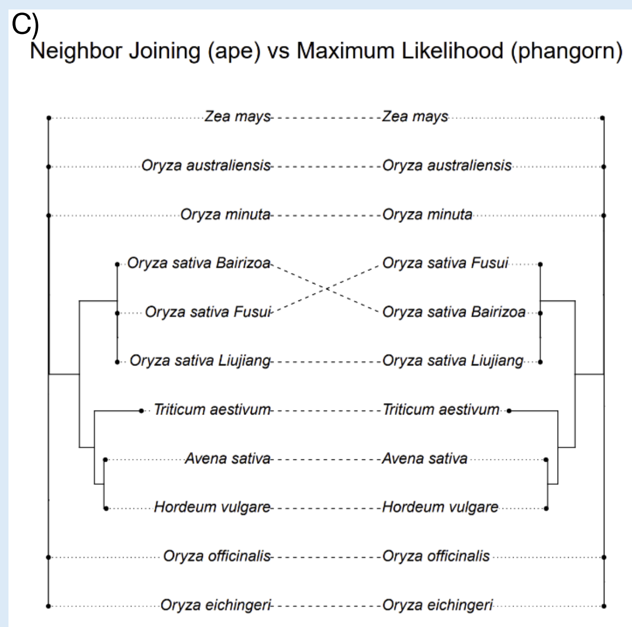
3. Plot the tanglegrams and add titles using mtext().
   Plotting the tanglegrams in the console often cuts off parts of the trees, so the plot is saved as a PDF using pdf() for full visibility. The title can also get cut off, so par(oma = ...) is used to increase the top outer margin, and mtext() adds the title in the outer margin instead of inside the plot. This keeps the title readable and prevents it from overlapping with the trees. The final tanglegram shows the two trees side-by-side with dashed lines connecting matching taxa (see Figure 5).

```
#plot tanglegrams

pdf("cophylo_plot_NJ_MP.pdf")
par(oma = c(1, 1, 5, 1)) # setting outer margins, order goes bottom, left, top, right
#this ensures title is not cut off in plot
plot(co_phy_NJ_MP) mtext("Neighbor Joining (ape) vs Maximum Parsimony (phangorn)",
      side = 3,          #top of pdf
      line = 2,          #move it down a bit so it is not cut off
      cex = 1.5)         #larger text
dev.off()

pdf("cophylo_plot_MP_ML.pdf")
par(oma = c(1, 1, 5, 1)) #setting outer margins, order goes bottom, left, top, right
#this ensures title is not cut off in plot
plot(co_phy_MP_ML)
mtext("Maximum Parsimony vs Maximum Likelihood",
      #putting title in margin
      side = 3,          #top of pdf
      line = 2,          #move it down a bit so it is not cut off
      cex = 1.5)         #larger text

dev.off()


pdf("cophylo_plot_NJ_ML.pdf")

par(oma = c(1, 1, 5, 1)) #setting outer margins, order goes bottom, left, top, right
#this ensures title is not cut off in plot
plot(co_phy_NJ_ML)
mtext("Neighbor Joining (ape) vs Maximum Likelihood (phangorn)",
      #putting title in margin
      side = 3,          #top of pdf
      line = 2,          #move it down a bit so it is not cut off
      cex = 1.5)         #larger text
dev.off()
```

Figure 5: **Tanglegrams comparing neighbor joining, maximum parsimony, and maximum likelihood phylogenetic tree to illustrate how similarly the three methods resolve relationships among the Poaceae sequences.**



__A)__ Tanglegram comparing neighbor-joining tree (build using ape) and maximum parsimony tree (built using phangorn). ***B)*** *Tanglegram comparing maximum parsimony and maximum likelihood trees (both built using the phangorn).* ***C)*** *Tanglegram comparing neighbor-joining tree (ape) and maximum likelihood tree (phangorn).*
Across all panels, the major phylogenetic structure is highly consistent between methods: the Oryza sativa cultivars, the wild Oryza species, the cereal grasses (wheat, barley, oat) all cluster together. All line crossings result from the tanglegram arrangement of unresolved nodes, where multiple branch orientations are equally plausible, rather than true evolutionary disagreement.



Conclusion

Across all phylogenetic methods, the tanglegrams show that the major evolutionary relationships are consistent: the cereal grasses cluster together, and the *Oryza* sativa cultivars group, and the wild *Oryza* species form a cluster (except for one wild species in the neighbor-joining tree, which is placed outside the main wild-rice group). The agreement across approaches indicates that the broad splits in the dataset are well supported regardless of tree-building approach.

The differences that do appear occur in the fine-scale resolution of closely related taxa. Both neighbor-joining and maximum likelihood under-resolve the *Oryza sativa* cultivars and wild *Oryza* species, recovering the group but showing minimal internal structure within it. Neighbor-joining also places one wild *Oryza* species slightly outside the main wild-rice cluster, which may reflect limitations of distance-based reconstruction for very closely related sequences. Maximum likelihood avoids this misplacement, but *Oryza sativa* cultivars and wild *Oryza* clusters still have internal nodes, indicating weak statistical support for resolving relationships in these highly similar sequences (1). Maximum parsimony provides the highest internal resolution, distinguishing more relationships between cultivars and wild *Oryza* species. This additional resolution most likely results from maximum parsimony's sensitivity to small character-state differences (1).

These small differences reflect each method's underlying assumptions: neighbor-joining relies on genetic distances, maximum parsimony minimizes character changes, and maximum-likelihood evaluates trees under nucleotide substitution model. The tanglegrams highlight where these methods agree and where branching order shifts, but they also show some apparent disagreements arise from nodes that are weakly supported or collapsed. In these regions, the branching order is not fixed and can be arranged in multiple valid ways. Because their orientation is not always matched between trees, the tanglegram may display artificial disagreement that does not reflect true evolutionary conflict.

Overall, the analysis shows that while method choice influences fine-scale branching patterns and node resolution among closely related species, all three approaches consistently recover the major evolutionary groupings in the Poaceae dataset. This consistency across neighbor-joining, maximum parsimony, and maximum likelihood supports confidence in the broad evolutionary relationships inferred from the data. However, finer-scale interpretations require careful evaluation, as unresolved nodes can introduce visually misleading discrepancy in comparative tree visualizations such as tanglegrams.

References

1. Zou Y, Zhang Z, Zeng Y, Hu H, Hao Y, Huang S, et al. Common Methods for Phylogenetic Tree Construction and Their Implementation in R. Bioengineering. 2024 May;11(5):480.

2. Paradis E, Schliep K. ape 5.0: an environment for modern phylogenetics and evolutionary analyses in R. Bioinformatics. 2019 Feb 1;35(3):526–8.

3. Schliep KP. phangorn: phylogenetic analysis in R. Bioinformatics. 2011 Feb 15;27(4):592–3.

4. Bodenhofer U, Bonatesta E, Horejš-Kainrath C, Hochreiter S. msa: an R package for multiple sequence alignment. Bioinformatics. 2015 Dec 15;31(24):3997–9.

5. Revell LJ. phytools: an R package for phylogenetic comparative biology (and other things). Methods in Ecology and Evolution. 2012;3(2):217–23.

6. Scornavacca C, Zickmann F, Huson DH. Tanglegrams for rooted phylogenetic trees and networks. Bioinformatics. 2011 July 1;27(13):i248–56.

7. Marzonie MR, Nitschke MR, Bay LK, Bourne DG, Harrison HB. Symbiodiniaceae diversity varies by host and environment across thermally distinct reefs. Molecular Ecology. 2024;33(9):e17342.

8. LaJeunesse TC, Thornhill DJ. Improved Resolution of Reef-Coral Endosymbiont (Symbiodinium) Species Diversity, Ecology, and Evolution through psbA Non-Coding Region Genotyping. PLOS ONE. 2011 Dec 28;6(12):e29013.

9. Kimura M. A simple method for estimating evolutionary rates of base substitutions through comparative studies of nucleotide sequences. J Mol Evol. 1980 Dec;16(2):111–20.

10. Likelihood of a tree. — as.pml [Internet]. [cited 2025 Dec 4]. Available from: https://klausvigo.github.io/phangorn/reference/pml.html

Supplemental: Full Code for Quick Copy & Paste

```
#Part 1: Prepare Sequences
#Step 1: Install and load necessary packages in R
#install necessary packages

#Multiple Sequence Alignment package from Bioconductor
if (!require("BiocManager", quietly = TRUE))
  install.packages("BiocManager")

BiocManager::install("msa")

#Analyses of Phylogenetics and Evolution
install.packages("ape")

#Phylogenetic Reconstruction and Analysis
install.packages("phangorn")

#Phytools (for tanglegram)
install.packages("phytools")

#retrieving packages from library
library(ape)
library(msa)
library(phangorn)
library(phytools)

#step 2: Download sequences directly from GenBank

#make a character vector of chosen accession numbers
my_accession_numbers <- c("GQ150262.1", "GQ150261.1", "GQ150257.1", "KP864581.1",
                          "KP864576.1", "KP864570.1", "KP864571.1", "MH346224.1",
                          "HQ600125.1", "KJ477768.1", "AF543684.1")

#download FASTA data from GenBank
sequences <- read.GenBank(my_accession_numbers,
                          seq.names = my_accession_numbers,
                          #names given to each sequence (default is accession numbers used)
                          species.names = TRUE,
                          #returns species names of each sequence
                          as.character = TRUE)
                          #returns each nucleotide as a character vector
```

```
#By default, tip labels are the accession numbers, so we'll replace them with species/cultivar
#names. Duplicated names (our Oryza sativa cultivars) break parsimony and likelihood analyses,
#so a basic approach to fix this issue is to manually assign unique names to each sequence

names(sequences) <- c(
  "GQ150262.1" = "Oryza_sativa_Liujiang",
  "GQ150261.1" = "Oryza_sativa_Fusui",
  "GQ150257.1" = "Oryza_sativa_Bairizoa",
  "KP864581.1" = "Oryza_minuta",
  "KP864576.1" = "Oryza_officinalis",
  "KP864570.1" = "Oryza_australiensis",
  "KP864571.1" = "Oryza_eichingeri",
  "MH346224.1" = "Triticum_aestivum",
  "HQ600125.1" = "Hordeum_vulgare",
  "KJ477768.1" = "Avena_sativa",
  "AF543684.1" = "Zea_mays")

#step 3: Align homologous regions using msa package

#sequences is a list of character vectors like c("A","T","G","C",...)
#need to make them one character string per sequence for msa to work

character_sequences <- sapply(sequences,
                              function(x) paste(x, collapse=""))

#sapply applies a function to each character in a list (so each "A","T",...)
#x represents all the vector of nucleotides ("A","T",...) in each sequence
#paste(x, collapse="") joins all letters together into one string

#convert the character list to a DNAStringset (format msa likes)
biostrings_sequences <- DNAStringSet(character_sequences)

#by default, msa sometimes converts seq. names back to original accession numbers,
#which makes our ape NJ tree have the GenBank names, so for consistency we can
#reassign the names we manually set earlier
names(biostrings_sequences) <- names(sequences)

#run msa
alignment <- msa(biostrings_sequences)

#Part 2: Construct Phylogenetic Trees
#step 1: Create a Neighbor-joining tree using distance matrices from ape

#now that our sequences are aligned, we need to convert the format into one ape can run
#Convert MSA to DNAbin

alignment_DNAbin <- msaConvert(alignment, type = "ape::DNAbin")
#bin = binary representation of nucleotides (A=1, C=2, G=3, T=4)

#calculate distances between sequences
dist_matrix_ape <- dist.dna(alignment_DNAbin, model = "K80")
#model tells R how to compare the DNA
#K80 is a statistical model that assumes equal base frequencies, but different types of
#mutations have different mutation rates

#neighbor-joining tree estimation
nj_tree_ape <- nj(dist_matrix_ape)

#root neighbor-joining tree so that Zea mays is the outgroup. Rooting fixes the
#position of the common ancestor so all trees share the same evolutionary orientation.
nj_tree_ape <- root(nj_tree_ape, outgroup = "Zea_mays", resolve.root = TRUE)

#plot tree
plot(nj_tree_ape,
     type = "phylogram",              #type of tree where branch lengths are
                                      #proportional to the amount of evolutionary change
     main = "Neighbor-Joining Tree")  #Title

#step 2: Build a maximum parsimony tree using phangorn
```

```
#again we need to convert aligned sequences into a format phangorn can read

#Convert MSA to phyDAT
alignment_phangorn <- as.phyDat(alignment, type = "DNA")    #phyDAT = phylogenetic data

#create a Neighbor Joining tree (quick starting point for phangorn to build off of)
dist_matrix_phang <- dist.ml(alignment_phangorn)     # distance matrix using ML model
nj_tree_phang <- NJ(dist_matrix_phang)               # basic tree structure for maximum parismony
search

#perform maximum parsimony tree search
mp_tree <- optim.parsimony(nj_tree_phang, alignment_phangorn)

#running phangorn’s maximum parismony or maximum likelihood searches needs a starting tree,
#The NJ tree acts as a rough initial guess to begin optimizing from
#otherwise, it would have to test all possible trees which would take a long time.

#root maximum parsimony tree so that Zea mays is the outgroup.
mp_tree <- root(mp_tree, outgroup = "Zea_mays", resolve.root = TRUE)

#plot maximum parsimony tree
plot(mp_tree,
     type = "phylogram",              #type of tree where branch lengths are
                                      #proportional to the amount of evolutionary change
     main = "Maximum Parsimony Tree") #Title

#step 3: build maximum likelihood tree using phangorn

#a maximum likelihood is a bit more complicated to build than our last two trees
#However you can think of the steps as first loading the tree & alignment into a “calculator”,
#and then tell the “calculator” to make the tree as statistically correct as possible.

#first we need to bundle our starting tree and our alignment together
#like our maximum parsimony tree, our maximum likelihood tree also needs a NJ starting tree
ml_bundle <- pml(nj_tree_phang, data = alignment_phangorn)

#pml() sets up the tree structure, branch lengths, substitution model (default = GTR),
#and the alignment all in a single object

#GTR (General Time Reversible) model is a flexible DNA-evolution model that allows each type
#of nucleotide substitution to occur at its own rate while also estimating the base frequencies.
#It’s widely used because it realistically models sequence evolution and makes maximum
#likelihood tree building mathematically workable.

#now we make the tree as statistically likely as possible given the DNA
ml_tree <- optim.pml(
  ml_bundle,
  optNni = TRUE,   #allows the function to swap branches to fix incorrect tree shapes
  #this lets ML explore new tree structures instead of being stuck with the NJ guess
  optGamma = TRUE, #estimate gamma rate variation
  #accounts for some DNA sites mutating faster/slower than others, makes model more realistic.
  optBf = TRUE)    #estimate base frequencies (A/T/C/G proportions) from the data
  #without this, the model incorrectly assumes all bases are equally common

#root maximum likelihood tree
ml_tree_rooted <- root(ml_tree$tree, outgroup = "Zea_mays", resolve.root = TRUE)

# We use ml_tree$tree instead of ml_tree because optim.pml() returns a big object
# that contains the tree plus extra statistical information. Rooting the whole object
# can make the tree draw weird, so we extract just the actual phylo tree (ml_tree$tree)

#plot maximum likelihood tree
plot(ml_tree_rooted, type="phylogram", main="Maximum Likelihood Tree")

#Part 3: Tanglegram Visualization
#prepare trees to be compared

#some internal nodes in the original trees were collapsed (shown as straight lines),
#which makes it hard for cophylo() to rotate the trees properly.
#this can make the tanglement score look higher than it really is.
```

```r
#multi2di() adds tiny branches to any collapsed nodes, giving them proper structure
#This ensures that cophylo() can rotate trees correctly

nj_tree_ape <- multi2di(nj_tree_ape)
mp_tree <- multi2di(mp_tree)
ml_tree <- multi2di(ml_tree$tree)

#align trees based on matching branching
co_phy_NJ_MP <- cophylo(nj_tree_ape, mp_tree, rotate = TRUE)
co_phy_MP_ML <- cophylo(mp_tree, ml_tree, rotate = TRUE)
co_phy_NJ_ML <- cophylo(nj_tree_ape, ml_tree, rotate = TRUE)

#rotate = TRUE lets cophylo() rotate internal nodes to arrange the trees so they
#show the same clades in the same left/right positions whenever possible.
#This way, any differences you see come from the actual tree structure,
#not the order of tips.

#plot tanglegrams
pdf("cophylo_plot_NJ_MP.pdf")
par(oma = c(1, 1, 5, 1)) #setting outer margins, order goes bottom, left, top, right
#this ensures title is not cut off in plot
plot(co_phy_NJ_MP)
mtext("Neighbor Joining (ape) vs Maximum Parsimony (phangorn)",
      side = 3,          #top of pdf
      line = 2,          #move it down a bit so it is not cut off
      cex = 1.5)         #larger text
dev.off()

pdf("cophylo_plot_MP_ML.pdf")
par(oma = c(1, 1, 5, 1)) # setting outer margins, order goes bottom, left, top, right
#this ensures title is not cut off in plot
plot(co_phy_MP_ML)
mtext("Maximum Parsimony vs Maximum Likelihood",
      #putting title in margin
      side = 3,          #top of pdf
      line = 2,          #move it down a bit so it is not cut off
      cex = 1.5)         #larger text
dev.off()

pdf("cophylo_plot_NJ_ML.pdf")
par(oma = c(1, 1, 5, 1)) #setting outer margins, order goes bottom, left, top, right
#this ensures title is not cut off in plot
plot(co_phy_NJ_ML)
mtext("Neighbor Joining (ape) vs Maximum Likelihood (phangorn)",
      #putting title in margin
      side = 3,          #top of pdf
      line = 2,          #move it down a bit so it is not cut off
      cex = 1.5)         #larger text
dev.off()
```