Supplementary Material to "Outperforming the Majority-Rule Consensus Tree Using Fine-Grained Dissimilarity Measures"

Yuki Takazawa[1*], Atsushi Takeda[2,3], Momoko Hayamizu[4], and Olivier Gascuel[5*]

[1] Department of Information Science and Technology, The University of Tokyo, Tokyo, Japan
[2]Graduate School of Advanced Science and Engineering, Waseda University, Tokyo, Japan
[3]Computational Bio Big-Data Open Innovation Laboratory AIST-Waseda University, Tokyo, Japan
[4]Department of Applied Mathematics, Faculty of Fundamental Science and Engineering, Waseda University, Tokyo, Japan
[5]Institut de Systématique, Evolution, Biodiversité (ISYEB, UMR7205 CNRS, Muséum National d'Histoire Naturelle, SU, EPHE, UA), Paris, France
**Corresponding authors:* yuki-takazawa@g.ecc.u-tokyo.ac.jp, oliver.gascuel@mnhn.fr

A Maximum of the unscaled transfer dissimilarity

In this section, we prove the following proposition. Note that a tree is said to be a caterpillar if there are two leaves such that all the nodes in the tree are either on the path between them or adjacent to the node on the path.

Proposition 1. *The maximum unscaled dissimilarity between two trees with n taxa is $n^2/2 - 2n + 2$ when n is even and $n^2/2 - 2n + 3/2$ when n is odd.*

Proof. First, we consider the maximum sum of sizes of internal bipartitions in a tree with n taxa. [1] derived that for any strictly increasing function $f : \{2, \dots, \lfloor n/2 \rfloor\} \to \mathbb{R}_{>0}$, the following type of quantity is (only) maximized by caterpillar trees:

$$\Phi_f(T) = \sum_{b \in B_{\text{int}}(T)} f(\text{size}(b)).$$

In particular, if we take f to be $f(x) = x$, the sum of sizes of internal bipartitions is maximized by caterpillars and is given by:

$$\sum_{i=2}^{n-2} (\min(i, n-i)) = \begin{cases} n^2/4 - 2 & \text{when } n \text{ is even,} \\ n^2/4 - 9/4 & \text{when } n \text{ is odd.} \end{cases} \tag{1}$$

Recall that the unscaled transfer dissimilarity between two trees T_1 and T_2 with n leaves are defined as follows:

$$\begin{aligned} d_{\text{transfer(unscaled)}}(T_1, T_2) = & \sum_{b \in B_{\text{int}}(T_1)} \min_{b' \in B(T_2)} \text{Transfer}(b, b') \\ & + \sum_{b \in B_{\text{int}}(T_2)} \min_{b' \in B(T_1)} \text{Transfer}(b, b'). \end{aligned} \tag{2}$$

Note that each summand cannot be larger than $\text{size}(b) - 1$ since the transfer distance between b and some appropriate external branches is equal to that amount. Combining (1) and (2), we can bound the unscaled transfer dissimilarity as follows:

$$d_{\text{transfer(unscaled)}}(T_1, T_2) \leq \sum_{b \in B_{\text{int}}(T_1)} (\text{size}(b) - 1) + \sum_{b \in B_{\text{int}}(T_2)} (\text{size}(b) - 1) \tag{3}$$

$$\leq \begin{cases} n^2/2 - 2n + 2 & \text{if } n \text{ is even} \\ n^2/2 - 2n + 3/2 & \text{if } n \text{ is odd.} \end{cases} \tag{4}$$

In particular, the last inequality (4) becomes an equality if and only if both T_1 and T_2 are caterpillars. We show in the following that you can construct two distinct caterpillar trees T_1 and T_2 that achieve this bound, i.e. achieve the equality of the inequality (3).

Suppose there are n taxa labeled as $\{1, \dots, n\}$. Take an arbitrary caterpillar tree T_1. You can assume without loss of generality that taxa 1 and n are at the two ends of the caterpillar tree, and the others are ordered accordingly: i.e. the tree in the Newick format has an expression as "$((\cdots(((1, 2), 3), 4), \cdots), n);$". Then, consider a tree T_2 that has a Newick expression as "$((\cdots(((y_1, y_2), y_3), y_4), \cdots), y_n);$", where

$$y_{2i-1} = i \qquad \text{for } i = 1, \dots, \lceil n/2 \rceil \tag{5}$$

$$y_{2i} = \lceil n/2 \rceil + i \qquad \text{for } i = 1, \dots, \lfloor n/2 \rfloor \tag{6}$$

Consider a bipartition $b = \{1, \dots, m \mid m+1, \dots, n\}$ in T_1, and without loss of generality suppose $m \leq \lfloor n/2 \rfloor$. Now, consider an arbitrary bipartition b' in T_2. Then, at least a part of $b' = A' \mid \bar{A}'$, say A', has $k \geq 1$ elements from $\{1, \dots, m\}$. Then, by construction, there are at least $k - 1$ elements in A' that are not in $\{1, \dots, m\}$. Likewise, there are at least $m - k - 1$ elements in $\bar{A}'$ that are not in $\{1, \dots, m\}$. This means that you must transfer at least $\min\{(k-1) + (m-k), k + (m-k-1)\} = m - 1$ elements from one part of b' to the other to match b. Thus, for all bipartitions b in T_1,

$$\min_{b' \in B(T_2)} \text{Transfer}(b, b') = \text{size}(b) - 1. \tag{7}$$

The same argument can be applied to bipartitions in T_2. This implies that with these two trees, the equality holds in (3), and the bound provided by (4) is achieved. $\square$

B Details on the pruning algorithm

B.1 Algorithm overview

Here, we describe the pruning algorithm for computing an approximate median tree with respect to the scaled transfer dissimilarity. The case with the unscaled transfer dissimilarity can be handled similarly.

Algorithm 1 outlines the entire pruning process. It takes as input an initial binary tree T_0, a set of input trees $\mathcal{T} = \{T_1, \dots, T_N\}$, and a hyperparameter K (positive integer), and returns the set of bipartitions E remaining after pruning the initial tree (corresponding to the pruned tree).

Algorithm 1 is divided into three parts, which are described in detail in the next section. Here, we briefly explain each part. The first part mainly computes transfer distances between the bipartitions in the initial tree and those in the input trees. Recall that the objective function to be minimized can be decomposed into false positive ($L_{\mathrm{FP}}(T, \mathcal{T})$) and false negative ($L_{\mathrm{FN}}(T, \mathcal{T})$) components:

$$L_{\mathrm{FP}}(T, \mathcal{T}) = \sum_{i=1}^{N} \sum_{b \in B_{\mathrm{int}}(T)} \left(\min_{b' \in B(T_i)} \frac{\mathrm{Transfer}(b, b')}{\mathrm{size}(b) - 1} \right), \tag{8}$$

$$L_{\mathrm{FN}}(T, \mathcal{T}) = \sum_{i=1}^{N} \sum_{b' \in B_{\mathrm{int}}(T_i)} \left(\min_{b \in B(T)} \frac{\mathrm{Transfer}(b, b')}{\mathrm{size}(b') - 1} \right), \tag{9}$$

where $B(T)$ denotes the set of bipartitions and $B_{\mathrm{int}}(T)$ denotes the set of internal bipartitions in a tree T. First, we compute the transfer support values of bipartitions in the initial tree T_0, measured with respect to the set of input trees $\mathcal{T}$. This gives the contribution of each branch b to the false positive part L_{FP} of the loss function, which remains constant throughout the pruning process.

Next, we compute the reverse: for each input tree, we measure the support of each bipartition in that input tree with respect to the initial tree T_0. This allows us to compute the false negative loss of T_0. Unlike the false positive side, each bipartition's contribution to reducing the false negative loss changes during pruning. This occurs because the bipartition b that minimizes the false negative loss may change as pruning progresses. To efficiently compute each branch's contribution to reducing the false negative loss at each pruning step, we also compute the K closest bipartitions in T_0 for each bipartition in the input trees. This enables us to calculate the difference in transfer distances between the first-best and second-best matches, which can be used to assess the gain from retaining the first-matched branches (see the discussion in the main text). Furthermore, we can update these first and second matches efficiently, as long as they remain within the initial top K branches.

In the second part, we initialize the contribution of each bipartition to the false positive loss in an array FP, and the contribution of each bipartition to reducing the false negative loss in an array δ_{FN}. These are used to define an array $L = \mathrm{FP} - \delta_{\mathrm{FN}}$, where subtraction is performed elementwise. Each element of L represents the loss reduction that will be achieved by pruning each branch. We also initialize some data structures that will be used during the pruning process.

The third part is the iterative pruning process. As long as there are positive elements in L (indicating that pruning will reduce the loss), we select the branch that provides the greatest loss reduction and prune it from the tree T, which was initialized as T_0. Some data structures are updated using the top K matches returned from the first step. For certain bipartitions in the input trees, the K minimum elements may not be sufficient to find a match after pruning branches from T. In such cases, we compute the transfer distances between these bipartitions and all bipartitions in the initial tree T_0.

B.2 Details

B.2.1 Computation of transfer support and hash values

The function $\textsc{TbeSupport}(T_0, \mathcal{T})$ computes the transfer support of each bipartition in the tree T_0 with respect to the input trees $\mathcal{T}$. The algorithm introduced in [2] achieves this in $O(nN(\log n)^3)$ time. In addition to calculating support values, our approach also computes hash values for the input bipartitions and tracks the number of occurrences of each bipartition across the input trees.

For the hash function, we use the method described in [3]. The core idea involves two "universal" hash functions. Given the bitstring representation of a bipartition, $b_1, \dots, b_n$, two prime numbers M_1 and M_2, and some integers

Algorithm 1 **Greedy Transfer Pruning (Scaled)**

```
Input: Initial binary tree $T_0$, Input Trees $\mathcal{T} = \{T_1, \ldots, T_N\}$, Hyperparameter $K$ (Controls the number of matches
    returned per branch). Let $m$ denote the number of unique internal bipartitions in $\mathcal{T}$.
Step 1: Computation of transfer support and matches
 1: $\mathrm{TS}, H_1, H_2, I, C \leftarrow \textsc{TbeSupport}(T_0, \mathcal{T})$
 2: $M, S \leftarrow \textsc{TbeMatch}(\mathcal{T}, T_0, K)$

Step 2: Initial Computation
 3: $T \leftarrow T_0$                                          ▷ $T$ will be modified.
 4: $\mathrm{FP} \leftarrow (1 - \mathrm{TS}) \times N$          ▷ False Positive Losses: Array of size $n - 3$.
 5: $\delta_{\mathrm{FN}} \leftarrow$ zero array of length $n - 3$.
 6: for all $i \in \{1 \ldots m\}$ do
 7:     if $M[i,1] \neq -1$ then                                 ▷ If the first match is not the external branch
 8:         $\delta_{\mathrm{FN}}[M[i,1]] \leftarrow \delta_{\mathrm{FN}}[M[i,1]] + (S[i,2] - S[i,1]) \times C[i]$
 9:     end if
10: end for
        ▷ Now $\delta_{\mathrm{FN}}[j]$ contains the false negative loss increase induced by pruning $j$th branch of the initial tree.
11: $L \leftarrow \mathrm{FP} - \delta_{FN}$                     ▷ Array containing loss reductions by pruning each branch.
12: $E \leftarrow \{1, \ldots, n - 3\}$                         ▷ Edge IDs of consensus tree
13: $R_1 \leftarrow \underbrace{[1, \ldots, 1]}_{m}$             ▷ Array containing the “rank” of the current first match.
14: $R_2 \leftarrow \underbrace{[2, \ldots, 2]}_{m}$             ▷ Array containing the “rank” of the current second match.
15: $W_1, W_2 \leftarrow$ Array of empty sets of size $n - 3$.
16: for all $i \in \{1 \ldots m\}$ do
17:     Append $i$ to $W_1[M[i,1]]$ and $W_2[M[i,2]]$.
18: end for                                                      ▷ $W_1[b]$ now contains all branches matched to $b$.
                                                 ▷ $W_2[b]$ now contains all branches that are second matches to $b$.

Step 3: Pruning
19: while $\max L > 0$ do
20:     $b \leftarrow \operatorname{argmax} L$                    ▷ Branch that has the largest benefit of pruning.
21:     $\textsc{Remove}(E,\ b)$                                  ▷ Remove $b$ from the edge set.
22:     $\textsc{RenewMatch}(M,\ R_1,\ R_2,\ W_1,\ W_2,\ b,\ L)$
        ▷ Renew first and second matches $W_1$ and $W_2$, ranks $R_1, R_2$ and the benefit array $L$ of pruning.
23: end while
24: return E                                                     ▷ Remaining edge ID set after pruning.
```

$a_1^i, \ldots, a_n^i$ $(i = 1, 2) \leq M_i$, we compute the hash values as follows:

$$h_1(b_1, \ldots, b_n) = \sum_{i=1}^{n} a_i^1 b_i \mod M_1, \tag{10}$$

$$h_2(b_1, \ldots, b_n) = \sum_{i=1}^{n} a_i^2 b_i \mod M_2. \tag{11}$$

We prepare an array of size M_1 where each index corresponds to a value of h_1. The probability of a hash collision for h_1 is $1/M_1$. Each element in the array is a linked list that stores all distinct h_2 values for the corresponding h_1. The combined hash value, represented by the tuple (h_1, h_2), has a collision probability of $1/(M_1 M_2)$.

The memory requirement for this hash table is $O(M_1 + m)$, thus M_2 can be chosen to be a large prime number to reduce the collision probability. For example, $M_2 = 2^{31} - 1$ is a practical choice for a 32-bit unsigned integer representation. Computing the hash values for all bipartitions in an input tree takes $O(n)$ time using post-order traversal. Additionally, we store the frequency of each bipartition in the linked list and the ID of a tree containing the bipartition for possible future retrieval. As we discuss later, we need complete representations of some bipartitions for computing transfer distances to all branches in the input tree.

Currently, we do not check for hash collisions, meaning that the algorithm has a failure probability of $1/(M_1 M_2)$. We choose $M_1 = O(nN)$, and the total time and memory complexity are $O(nN(\log n)^3)$ and $O(nN)$, respectively. As an alternative safe approach, we could store the bitstring representation of each bipartition and check for collisions, but this would increase the time and memory usage to $O(n^2 N)$.

After organizing the hash table, the final output of TbeSupport is:

- TS: Array of size $n - 3$, containing the transfer support for each internal bipartition in T_0.
- H_1: Array of size m, containing the h_1 hash values.
- H_2: Array of size m, containing the h_2 hash values.
- I: Array of size m, where each element is the ID of a tree containing the bipartition.
- C: An array of size m, storing the number of occurrences of the i-th bipartition in the input trees $\mathcal{T}$.

B.2.2 Computation of first K best matches

In this step, we switch the roles of the initial tree and the input trees. The function TbeMatch$(\mathcal{T}, T_0, K)$ considers each input tree as the reference and returns, for each bipartition in the input tree, the top K matches from the reference tree T_0. Specifically, for a given bipartition b in the input tree, the algorithm identifies the K closest bipartitions in T_0 in terms of the transfer distance. This is computed efficiently by modifying the algorithm used for TbeSupport [2] to track the top K matches instead of just the best one. This process runs in $O(nNK(\log n)^3)$ time.

The output of TbeMatch$(\mathcal{T}, T_0, K)$ consists of:

- M: An array of size $m \times K$. The (i, j)-th entry stores the j-th best matching bipartition from T_0 to the i-th bipartition from $\mathcal{T}$.
- S: An array of size $m \times K$, storing the (scaled) transfer distance for each match.

For any entries in S that are greater than or equal to 1, we replace the value with 1 and set the corresponding entries in M to -1, reflecting the fact that the minimum scaled transfer distance for external branches is always 1.

B.2.3 Initial computation

In step 2 of Algorithm 1, we first initialize the candidate tree T to the initial tree T_0. We then compute the initial "benefit" of pruning each internal branch in T_0. First, the contribution of each internal branch i to the false positive loss can be computed as $(1 - \mathrm{TS}[i]) \times N$. We denote by FP, the array containing this false positive loss.

Next, we calculate the increase in false negative loss that will be induced by pruning a branch i from T_0, denoted as $\delta_{\mathrm{FN}}[i]$. This value depends not only on the bipartition i and input trees $\mathcal{T}$, but also on the structure of T_0. Consider a bipartition b in the input tree that is matched to bipartition i in T_0. The scaled transfer distance between i and b is $S[b, 1]$. Let j be the second closest bipartition in T_0 to b. The scaled transfer distance between

b and j is $S[b,2]$. If we prune branch i, the increase in false negative loss with respect to bipartition b is given by $(S[b,2]-S[b,1]) \times C[b]$. This implies that the total increase in false negative loss that will be induced by pruning branch i can be computed as $\delta_{\mathrm{FN}}[i]=\sum_{b \in W_1[i]}(S[b,2]-S[b,1]) \times C[b]$, where $W_1[i]$ denotes the set of bipartitions in the input trees whose first match is branch i. Thus, we compute the array $\{\delta_{\mathrm{FN}}[i]\}_{i=1}^{n-3}$ as follows: Initialize $\delta_{\mathrm{FN}}[i]=0$ for all i. Then, iterate over all bipartitions b in the input trees and add $(S[b,2]-S[b,1]) \times C[b]$ to the first match of b (if $M[b,1] \neq -1$, i.e. if the first match is not the external branch).

Finally, we define $L = \mathrm{FP} - \delta_{\mathrm{FN}}$, which represents the benefit of pruning each branch.

To efficiently compute the pruning step, we maintain two arrays, R_1 and R_2, of size m (where m is the number of bipartitions in the input trees). These arrays track the “rank” of the current first and second matches for each bipartition. For each bipartition in the input trees and its current first (or second) best match in the tree T, the “rank” refers to the original position of the current first (or second) bipartition in the sorted set of bipartitions from the initial tree T_0. Initially, R_1 is an array of ones and R_2 is an array of twos, but the arrays will change as the first and second matches for each bipartition change along the pruning process. We also maintain two arrays, W_1 and W_2, of size $n-3$, where $W_1[i]$ contains the set of bipartitions whose first match is branch i, and $W_2[i]$ contains those whose second match is branch i.

B.2.4 Pruning and updating matches

After completing the initialization, we iteratively prune the branch with the highest L value as long as it is positive. Note that after pruning a branch, the contribution to reducing the false negative loss, δ_{FN}, and the L array need to be updated, since some bipartitions may have had the pruned branch as their first or second match, which becomes unavaiable after pruning. Consequently, we also need to update arrays R_1, R_2, W_1, W_2 accordingly to reflect these changes. The following steps achieve this (RenewMatch):

1. **Update for bipartitions in** $W_1[i]$: For each bipartition b in $W_1[i]$ (the set of bipartitions that had branch i as their first match), we promote their second match to become the new first match. This is done by appending b to $W_1[M[b,R_2[b]]]$, removing b from $W_2[M[b,R_2[b]]]$, and updating $R_1[b]$ to $R_2[b]$.

2. **Find the new second match**: After promoting the second match, we need to find a new second match for bipartition b. We start from the rank $R_2[b]+1$ and look for the next available bipartition that has not been pruned. Once we find this new match, say at rank $R_2[b]+l$, we append b to $W_2[M[b,R_2[b]+l]]$ and set $R_2[b]=R_2[b]+l$.

3. **Update the false negative loss**: After updating the first and second matches for $b \in W_1[i]$, we adjust δ_{FN}. Specifically, we update $\delta_{\mathrm{FN}}[M[b,R_1[b]]]$ by adding the difference in transfer distances between the new first match and the new second match, multiplied by the number of occurrences (i.e., $(S[b,R_2[b]]-S[b,R_1[b]]) \times C[b]$).

4. **Update for bipartitions in** $W_2[i]$: Similarly, for each bipartition b in $W_2[i]$ (those that had branch i as their second match), we search for the new second match starting from rank $R_2[b]+1$. Once the new match is found, say at rank $R_2[b]+l$, we append b to $W_2[M[b,R_2[b]+l]]$ and update $R_2[b]$ to $R_2[b]+l$. We also adjust δ_{FN} for these bipartitions by updating $\delta_{\mathrm{FN}}[M[b,R_1[b]]]$ by adding the difference of transfer distances between the old second match and the new second match, multiplied by the number of occurrences (i.e., $(S[i,R_2[b]]-S[i,R_2[b]-l]) \times C[b]$).

In some cases, it may be necessary to search beyond the K-th match, which requires computation of further matches than the originally computed top K matches. When we face this situation, we recompute transfer distances to all of the branches in T_0 (or T) in $O(n)$ time, sort them in $O(n \log n)$ time, and store it as an array. By computing all the transfer distances at once, this recomputation occurs at most once per bipartition.

If we ignore the time required for this additional computation and further updating of the first and second matches beyond the K-th one, the update process takes $O(K)$ time per bipartition throughout all iteartions, resulting in a total time complexity of $O(nNK)$ for the whole $m=O(nN)$ bipartitions. This implies that if only $O(NK(logn)^2)$ branches require re-computation, the total running time can be as fast as $O(nNK(logn)^3)$. In the worst case, the total running time is still bounded by $O(n^2N(logn)^3)$. The memory requirement is also $O(nNK)$ if we ignore the memory used for storing the results of the recomputation: In the worst case, the total memory usage can be as large as $O(n^2N)$.

C Detailed results of the experiments

C.1 Detailed results of simulation experiments

C.1.1 Comparison of the fully resolved consensus approaches

In the main text, we only showed the best score of all fully resolved approaches (the extended majority, ASTRAL, MAP, MCC for the Bayesian case and the extended majority, ASTRAL, MLE for the bootstrap case). Figure S1 and Figure S2 compare the performance of these fully resolved approaches in the Bayesian simulation experiment and the bootstrap simulation experiment respectively.

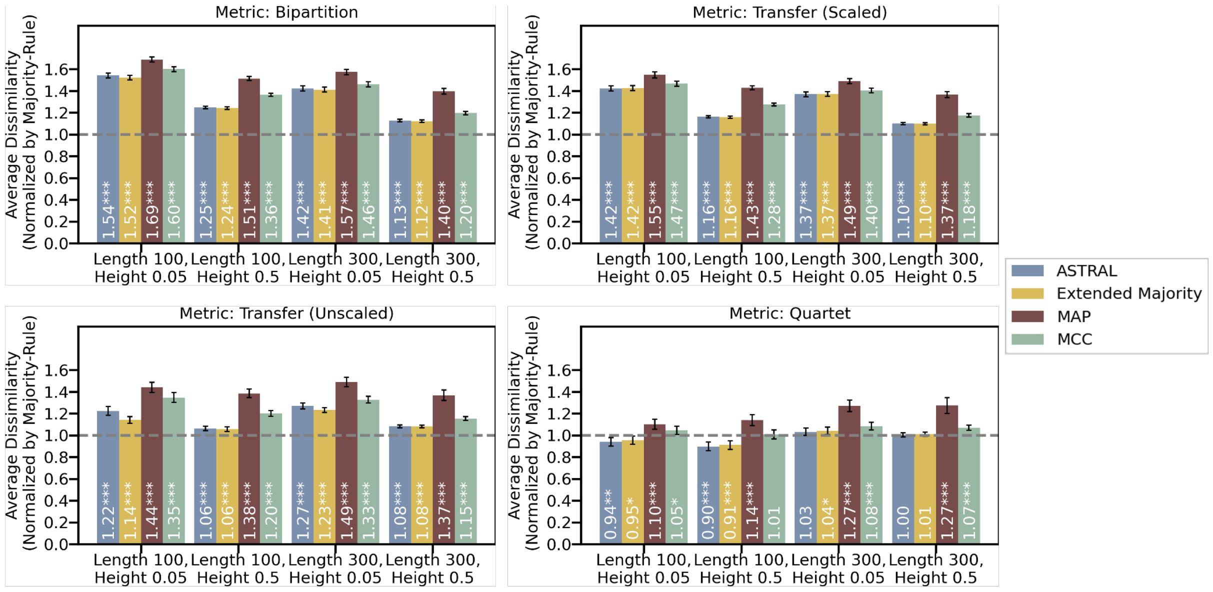
For the Bayesian case, MAP is the worst approach in all settings. This implies that MAP is not a good tree estimate in terms of the topology metric. In addition, the commonly-used MCC is also worse than ASTRAL or the extended majority in all of the cases. This suggests that MCC is not suitable for the consensus of posterior draws in terms of tree topology metrics. The best approach is either ASTRAL or the extended majority, for both as a summary of posterior draws (average dissimilarity against input trees) and as a tree estimate of the true phylogeny. Likewise, in the bootstrap case, MLE is mostly worse than the extended majority and ASTRAL.

C.1.2 Comparison of the pruning approaches with different support types and thresholds used for MLE

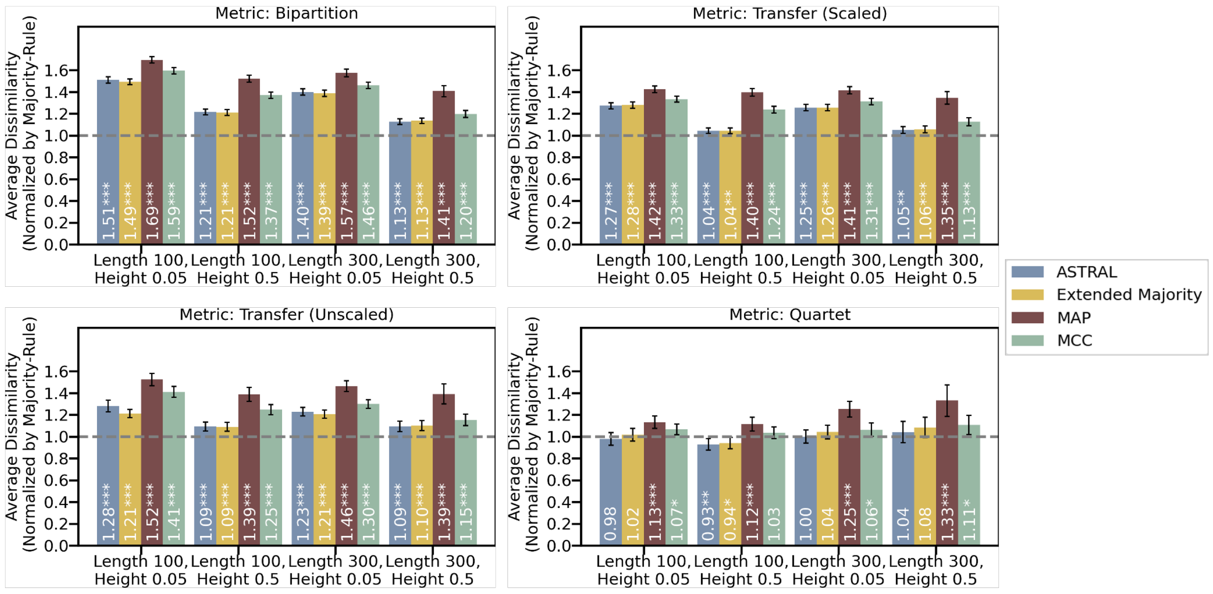
In the main text, we only showed the best score of all pruning approaches from MLE (alternating support types between transfer support and the branch support, as well as support thresholds from 5% to 95% in the increments of 5%). We show below the results of some common thresholds (50%, 70%, 95%), the most ambitious approach without thresholding (MLE) as well as an intermediate value (25%) with both types of support.

With Felsenstein's bootstrap support value, we can see that in both the average dissimilarity against bootstrap trees and the true tree, the more resolved, ambitious option of using 25 % performs well in most cases with respect to the three proposed dissimilarity measures. The exceptions are for the scaled transfer dissimilarity, where it performs slightly worse than the 50% threshold ($\sim$ the majority rule). In all cases with all dissimilarity measures, the 70% threshold performs worse than the majority rule. MLE without thresholding performs well with respect to the quartet distance and comparatively to majority rule in the unscaled transfer. However, its loss in terms of the unweighted measures is significantly higher than that of the majority-rule.

Pruning approaches using transfer support (transfer bootstrap expectation) thresholds perform well in terms of weighted measures. However, they mostly have worse results than the majority rule in terms of the unweighted measures. Because the transfer support always gives larger or equal support values to each branch, the resolution of the obtained trees are higher for transfer support approach.

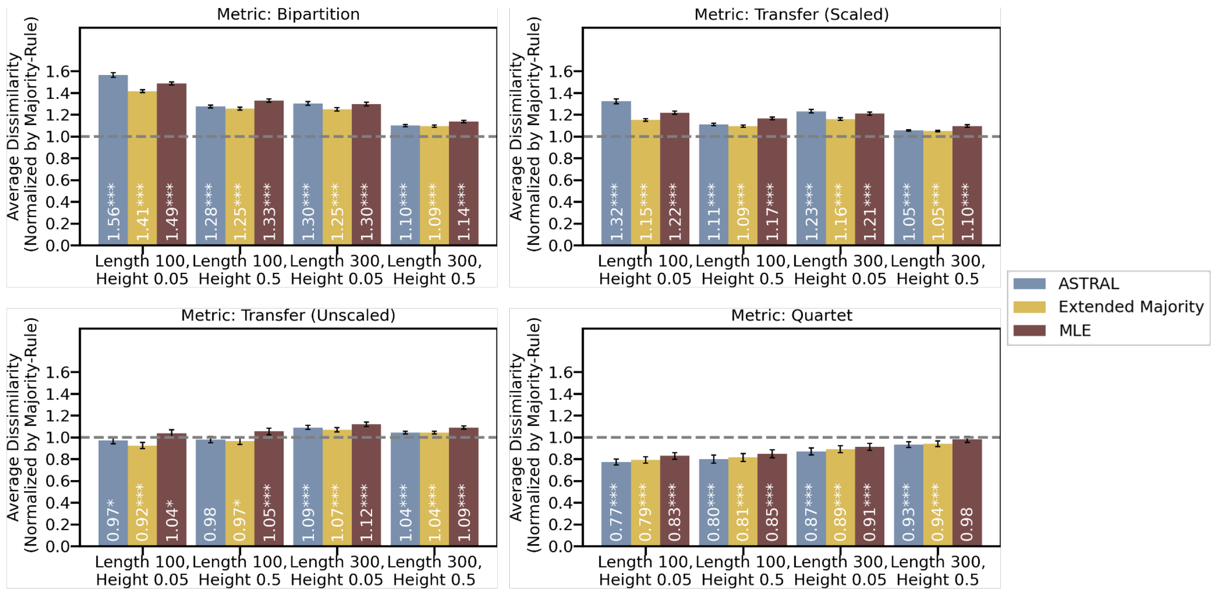


(a) Average Dissimilarity to the **Input** Trees

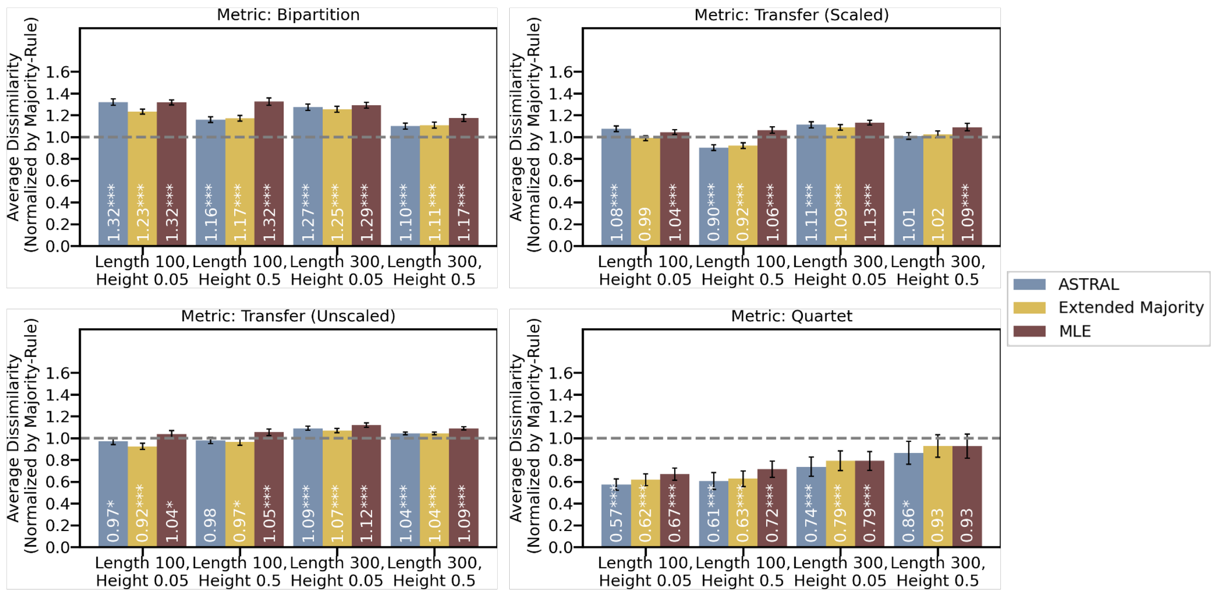


(b) Average Dissimilarity to the **True** Trees

Figure S1: Performance of fully resolved trees in the Bayesian simulation experiment, in comparison to the majority rule: (a) Average dissimilarities to the input trees for 100 datasets, (b): Average dissimilarities to the true trees for 100 datasets. The error bars are 95% confidence intervals, and asterisks indicate p-values (*$p < 0.05$, **$p < 0.01$, ***$p < 0.001$) based on paired t-tests.

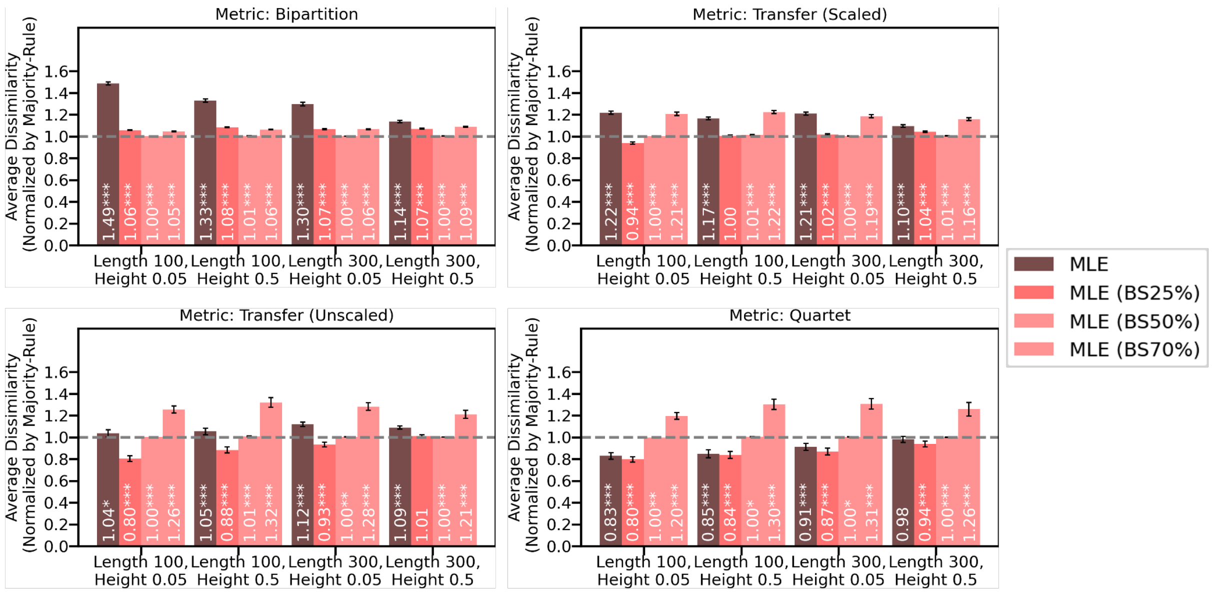


(a) Average Dissimilarity to the **Input** Trees

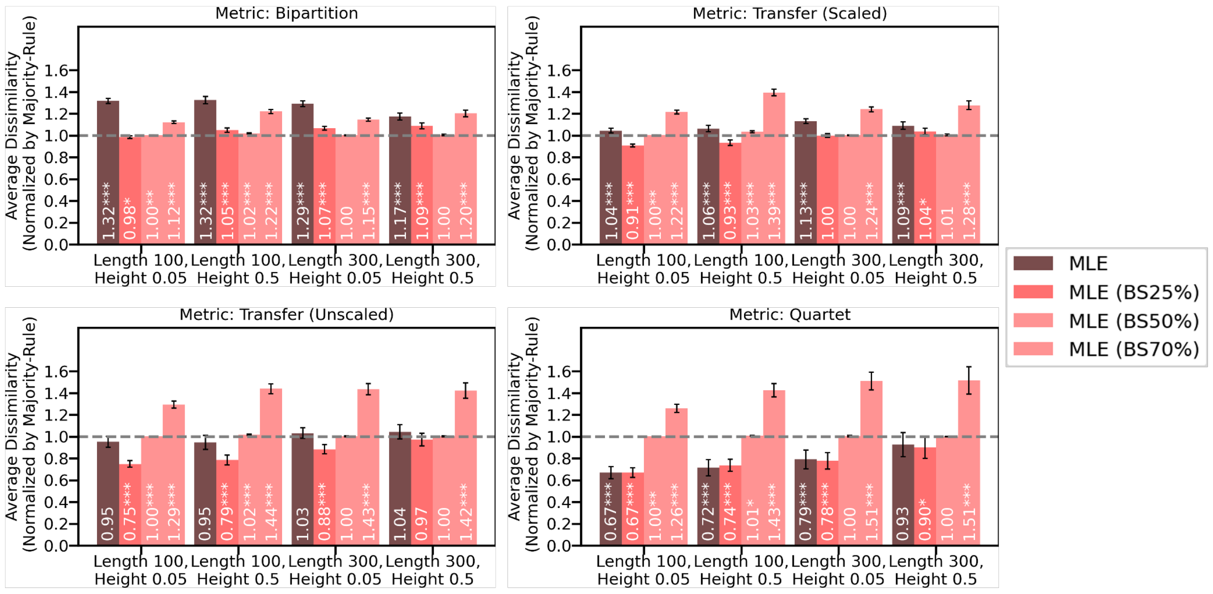


(b) Average Dissimilarity to the **True** Trees

Figure S2: Performance of fully resolved trees in the bootstrap simulation experiment, in comparison to the majority rule: (a) Average dissimilarities to the input trees for 100 datasets, (b): Average dissimilarities to the true trees for 100 datasets. The error bars are 95% confidence intervals, and asterisks indicate p-values (*$p < 0.05$, **$p < 0.01$, ***$p < 0.001$) based on paired t-tests.

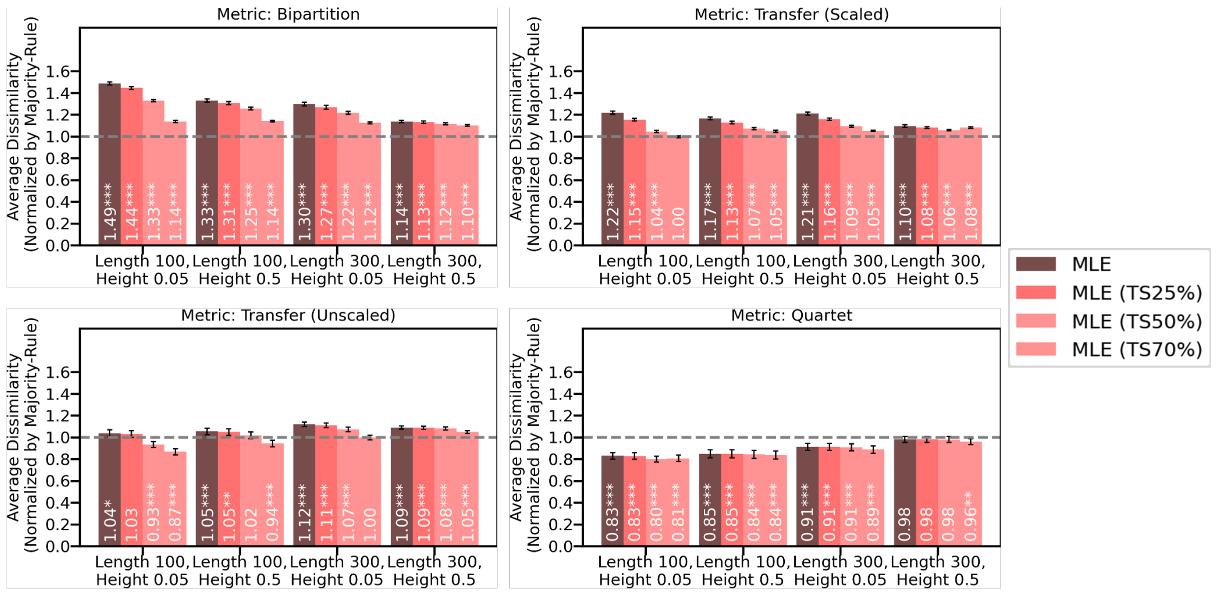


(a) Average Dissimilarity to the **Input** Trees

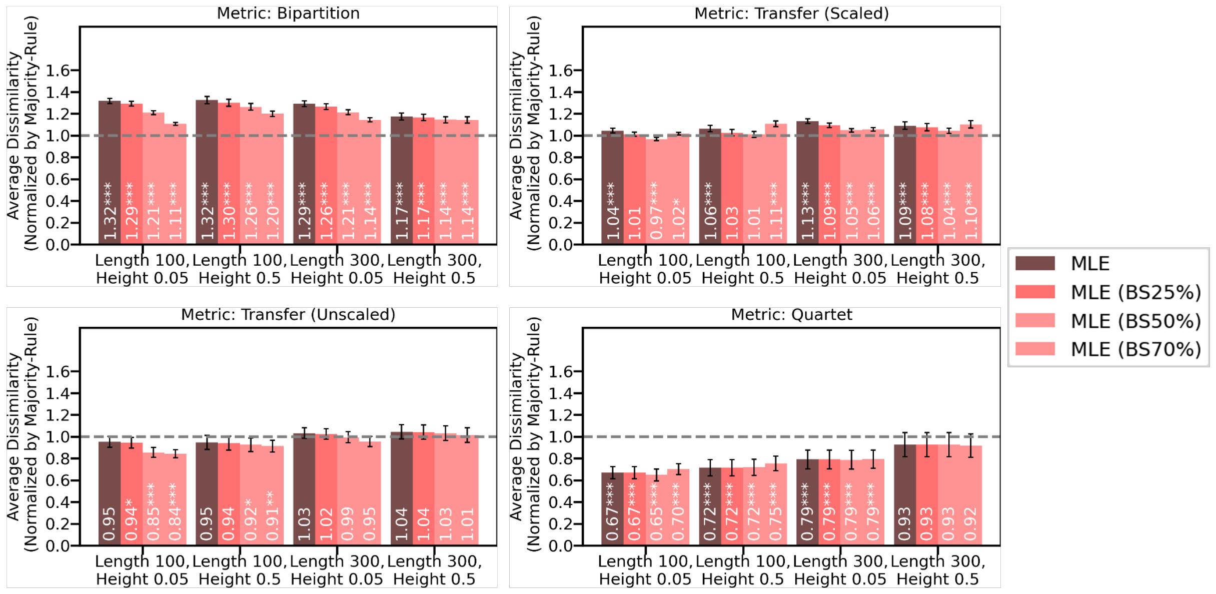


(b) Average Dissimilarity to the **True** Trees

Figure S3: Performance of MLE with different branch support selection thresholds in the Bayesian simulation experiment, in comparison to the majority rule: (a) Average dissimilarities to the input trees for 100 datasets, (b): Average dissimilarities to the true trees for 100 datasets. The error bars are 95% confidence intervals, and asterisks indicate p-values (*$p < 0.05$, **$p < 0.01$, ***$p < 0.001$) based on paired t-tests.

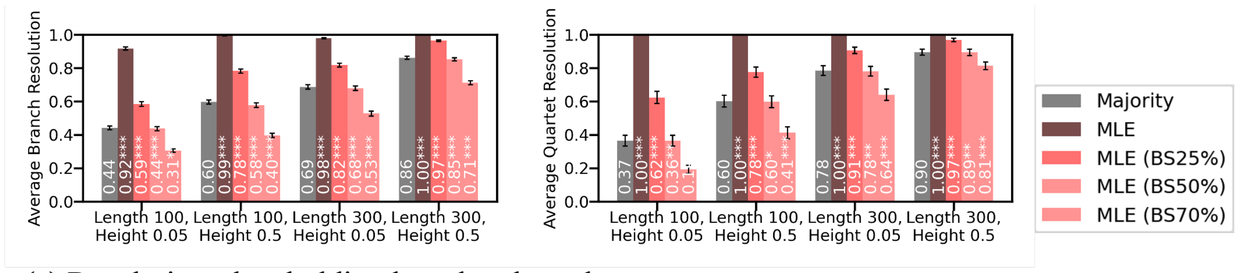


(a) Average Dissimilarity to the **Input** Trees



(b) Average Dissimilarity to the **True** Trees

Figure S4: Performance of MLE with different transfer support selection thresholds in the Bayesian simulation experiment, in comparison to the majority rule: (a) Average dissimilarities to the input trees for 100 datasets, (b): Average dissimilarities to the true trees for 100 datasets. The error bars are 95% confidence intervals, and asterisks indicate p-values (*$p < 0.05$, **$p < 0.01$, ***$p < 0.001$) based on paired t-tests.


(a) Resolution: thresholding based on branch supports.

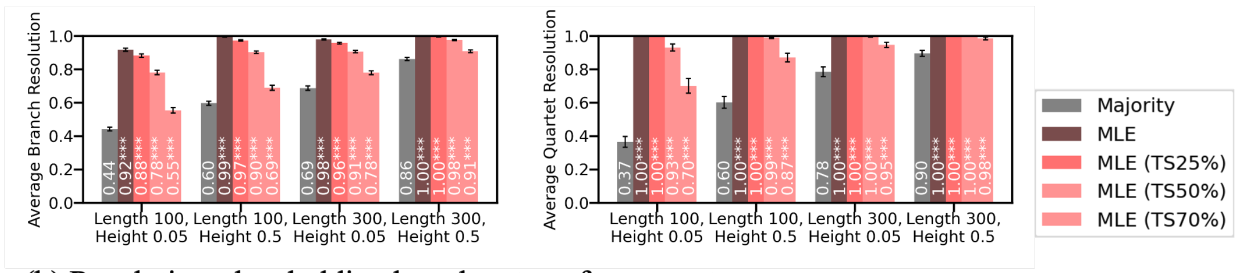

(b) Resolution: thresholding based on transfer supports.

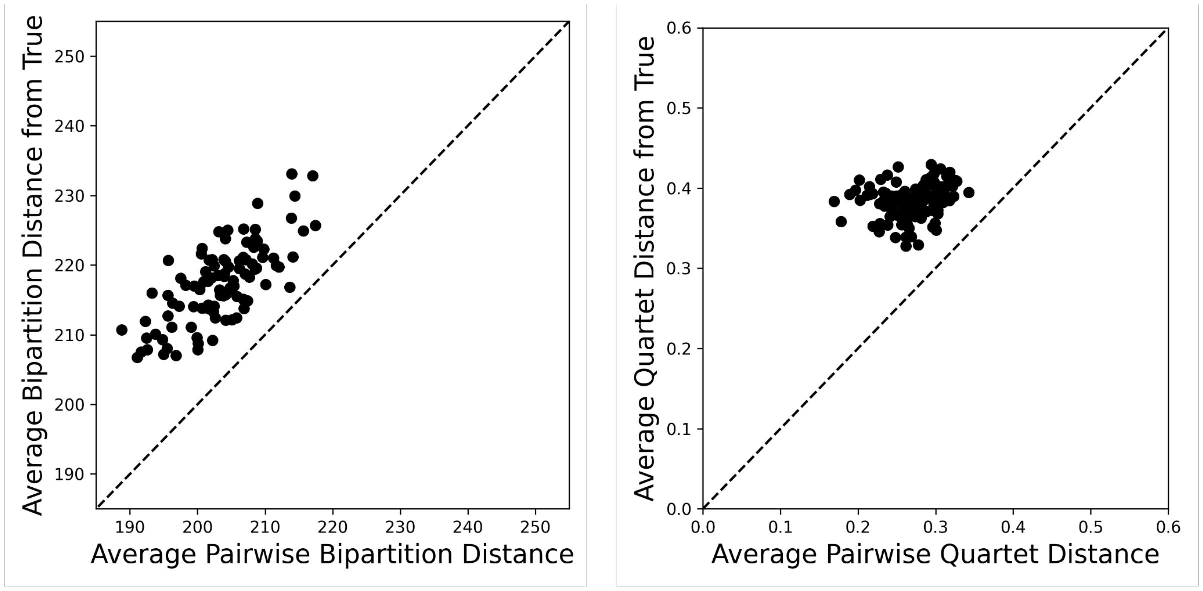
Figure S5: Resolution of MLE with different support thresholds and the majority-rule consensus tree: (a) Branch resolution (left) and quartet resolution (right) with branch support thresholds (b) Branch resolution (left) and quartet resolution (right) with transfer support thresholds. The error bars correspond to 95% confidence intervals of the average resolution of each estimator. The asterisks indicate p-values ($*p < 0.05$, $**p < 0.01$, $***p < 0.001$) of paired t-tests of resolution differences between each estimator and the majority-rule consensus tree.

C.2 Results of Bayesian analysis of subsampled mammals data

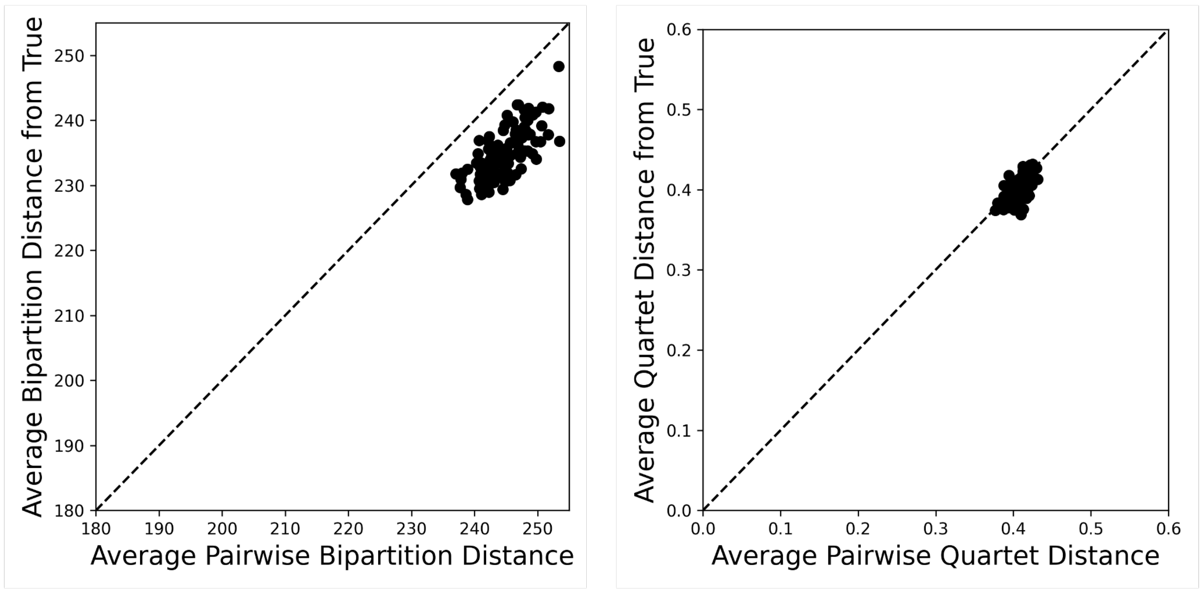
Figure S6 compares the average pairwise distance among input trees to the average distance between input trees and the true (NCBI) trees. In the Bayesian setting, both the bipartition and quartet distances show that the average distance to the true tree is consistently higher than the average pairwise distance among input trees across all datasets. This suggests that the true trees are not located at the center of the input tree clouds in either metric space.

In contrast, the bootstrap setting reveals a different pattern. For the bipartition distance, the average distance to the true tree is lower than the average pairwise distance among input trees in all datasets. When using the quartet distance, most datasets also show lower average distances to the true tree compared to the average pairwise distances, although some datasets deviate from this trend. Additionally, the bootstrap setting exhibits higher average pairwise distances among input trees, indicating greater dispersion (or variance) in the bootstrap distributions. These findings suggest that the bootstrap distribution for the subsampled mammals data has a less pronounced bias, although they have a broader dispersion of input trees.

Figure S7 shows the result of the subsampled analysis of the mammals data using the Bayesian approach. We see the evident discrepancies of between the average dissimilarity to the input trees and the average dissimilarity to the true trees: the proposed approaches successfully improve the majority rule with respect to the three fine-grained dissimilarity measures (the scaled transfer, the unscaled transfer, and the quartet distance) to the input trees, but they do not improve the dissimilarities to the true trees. This discrepancy can be attributed to the bias of the posterior distribution described above.

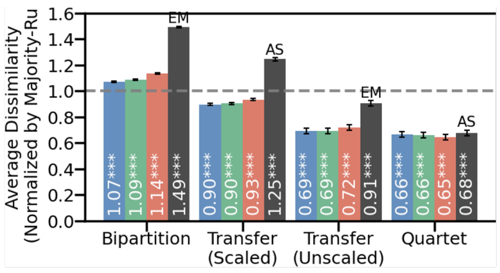


(a) Bayesian Setting

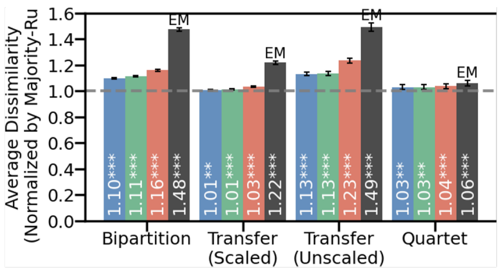


(b) Bootstrap Setting

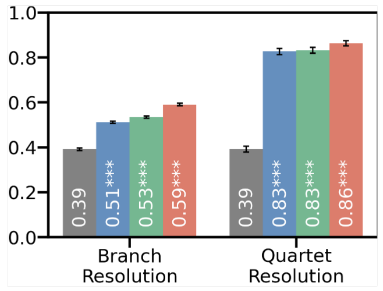
Figure S6: Scatter plots of the average pairwise distance between input trees vs. the average distance between input trees and the true (NCBI) trees for 100 datasets in the subsampled analysis of mammals data. (a): In the Bayesian setting. (b) In the bootstrap setting. The left panel uses bipartition distance as the metric, while the right panel uses normalized quartet distance.



(a) Average Dissimilarity to the **Input** Trees



(b) Average Dissimilarity to the **True** Trees



(c) Average Resolution

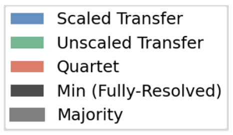


Figure S7: Results of the subsampled bootstrap analysis of Mammals datasets (a): Average dissimilarity between consensus methods and input trees for 100 datasets. (b): Average dissimilarity between consensus methods and the true tree for 100 datasets. (c): Average branch and quartet resolution for 100 datasets. The black bar corresponds to the minimum loss of four fully resolved approaches (EM: the extended majority, AS: ASTRAL, MAP, MCC). The name of the method that takes the minimum is shown above each bar. The error bars are 95% confidence intervals. The asterisks indicate p-values (*$p < 0.05$, **$p < 0.01$, ***$p < 0.001$) of paired t-tests of differences between each estimator and the majority-rule consensus tree.

C.3 Performance of the majority-rule consensus tree

All the dissimilarity results given in the main text are normalized by the performance of the majority-rule consensus. In this subsection, we list the performance of the majority-rule consensus tree in the following tables. Note that each value is normalized in the following way: the bipartition distance, the scaled dissimilarity, and the unscaled dissimilarity are divided by $2(n-3)$, where n is the number of taxa, and the quartet distance is divided by $2\binom{n}{4}$.

Table S1: The normalized average dissimilarity between the majority-rule consensus tree and the **input** trees in the **Bayesian simulation experiments**.

Dissimilarity	Length 100, Height 0.05	Length 100, Height 0.5	Length 300, Height 0.05	Length 300, Height 0.5
Bipartition distance	0.339	0.250	0.226	0.130
Scaled transfer dissimilarity	0.203	0.136	0.127	0.066
Unscaled transfer dissimilarity	1.112	0.648	0.507	0.270
Quartet distance	0.194	0.117	0.083	0.051

Table S2: The normalized average dissimilarity between the majority-rule consensus tree and the **true** trees in the **Bayesian simulation experiments**.

Dissimilarity	Length 100, Height 0.05	Length 100, Height 0.5	Length 300, Height 0.05	Length 300, Height 0.5
Bipartition distance	0.334	0.250	0.229	0.132
Scaled transfer dissimilarity	0.234	0.144	0.142	0.069
Unscaled transfer dissimilarity	1.028	0.703	0.530	0.279
Quartet distance	0.199	0.139	0.089	0.052

Table S3: The normalized average dissimilarity between the majority-rule consensus tree and the **input** trees in the **bootstrap simulation experiments**.

Dissimilarity	Length 100, Height 0.05	Length 100, Height 0.5	Length 300, Height 0.05	Length 300, Height 0.5
Bipartition distance	0.318	0.323	0.261	0.176
Scaled transfer dissimilarity	0.196	0.185	0.136	0.087
Unscaled transfer dissimilarity	1.803	1.240	0.781	0.428
Quartet distance	0.341	0.233	0.150	0.086

Table S4: The normalized average dissimilarity between the majority-rule consensus tree and the **true** trees in the **bootstrap simulation experiments**.

Dissimilarity	Length 100, Height 0.05	Length 100, Height 0.5	Length 300, Height 0.05	Length 300, Height 0.5
Bipartition distance	0.374	0.261	0.243	0.136
Scaled transfer dissimilarity	0.273	0.166	0.152	0.071
Unscaled transfer dissimilarity	1.461	0.960	0.637	0.316
Quartet distance	0.323	0.206	0.120	0.063

Table S5: The normalized average dissimilarity between the majority-rule consensus tree and the **input** trees or the true trees in the **experiments with subsampled mammals dataset**. Each column name specifies the type of analysis (Bayesian or bootstrap) and the compared trees (the input trees or the true trees).

Dissimilarity	Bayesian, input	Bayesian, true	Bootstrap, input	Bootstrap, true
Bipartition distance	0.357	0.396	0.420	0.396
Scaled transfer dissimilarity	0.233	0.322	0.325	0.348
Unscaled transfer dissimilarity	1.818	1.493	3.312	1.724
Quartet distance	0.311	0.338	0.388	0.382

References

[1] Mareike Fischer and Volkmar Liebscher. On the balance of unrooted trees. *Journal of Graph Algorithms and Applications*, 25(1):133–150, 2021.

[2] Jakub Truszkowski, Olivier Gascuel, and Krister M. Swenson. Rapidly computing the phylogenetic transfer index. In *19th International Workshop on Algorithms in Bioinformatics (WABI 2019)*, pages 20:1–20:12, 2019.

[3] Nina Amenta, Frederick Clarke, and Katherine St. John. A linear-time majority tree algorithm. In *Algorithms in Bioinformatics: Third International Workshop, WABI 2003*, pages 216–227, 2003.