VIETNAM NATIONAL UNIVERSITY

UNIVERSITY OF ENGINEERING AND TECHNOLOGY

REPORT OF THE STUDENT SCIENTIFIC RESEARCH CONFERENCE

YEAR 2023

Title:

COMBINING MULTIPLE TREE REARRANGEMENT OPERATORS FOR EFFICIENT PARSIMONY INFERENCE USING REINFORCEMENT LEARNING APPROACH

RESEARCH STUDENT GROUP:

1. Huỳnh Tiến Dũng – 21020007 – K66C-CLC
2. Vũ Quốc Tuấn – 21020033 – K66C-CLC
3. Nguyễn Việt Dũng – 21020043 – K66C-A-CLC2

SCIENTIFIC ADVISOR:

1. TS. Hoàng Thị Điệp

Year 2023

Combining Multiple Tree Rearrangement Operators for Efficient Parsimony Inference Using Reinforcement Learning Approach

Summary

Maximum parsimony phylogenetic bootstrapping is an important problem in bioinformatics with many applications in evolutionary biology and is efficiently solved using the approximate MPBoot approach. In this study, we propose adding the tree bisection and reconnection (TBR) transformation to MPBoot to improve sampling efficiency in the search space and propose the MPBoot-TBR algorithm. Since TBR has a cubic neighborhood, we propose optimization techniques to quickly evaluate a TBR transformation, quickly search for a neighborhood corresponding to a given cut edge, and hill climb using TBR when integrating TBR hill climbing into MPBoot. In addition, we propose the MPBoot-ACO algorithm that combines hill climbing of three transformations NNI, SPR, TBR using ant colony optimization (ACO) algorithm to improve the performance of the program. The stopping criterion of the proposed framework is adjusted because, compared to subtree pruning and regrafting (SPR), TBR requires fewer search iterations to converge to a sufficiently good MP score. MPBoot-TBR is equivalent to MPBoot in terms of bootstrap accuracy. Notably, MPBoot-ACO outperforms the original MPBoot in MP score and has an advantage over MPBoot-TBR in computation time. We have publicly implemented the proposed MPBoot-TBR algorithm at https://github.com/HynDuf/mpboot/tree/Huynh_Tien_Dung and MPBoot-ACO at https://github.com/HynDuf/mpboot/tree/ant-colony-optimization.

1 Introduction

1.1 Overview

Many computational biology investigations, such as biodiversity studies or the study of the spread of dangerous viral diseases, can only perform best when the underlying task of building the evolutionary tree is best done [1]. On a binary tree representing the evolutionary history of a given set of extant species, two closely related species are placed at two leaf nodes of proximity. Each internal node of the tree corresponds to a speciation event, which is also interpreted as the most recent common ancestor species of all leaves in the corresponding subtree.

A common need in biology is to reconstruct an evolutionary tree based on an alignment (MSA) of biological sequences of n species. It is an NP-complete combinatorial search problem [2] in the space of n-leaf binary trees. Maximum parsimony (MP) is a widely accepted criterion for evaluating the goodness of evolutionary trees, whereby the best tree explains the input MSA with the fewest substitutions [3].

The pipelines used in modern phylogenetic inference to reconstruct a tree often include subsequent bootstrap analysis [4] to assess the reliability of each branch based on its frequency in the bootstrap tree set. For simplicity, we will refer to the tree reconstruction problem with subsequent bootstrap analysis as phylogenetic bootstrapping. For a bootstrap analysis of B pseudoreplicates (typically $B = 1000$), the standard bootstrap method (SBS) constructs the bootstrap tree set by independently performing one tree search for each pseudoreplicate MSA; hence, it is computationally expensive. SBS for parsimony is implemented in TNT [5] and PAUP* [6]. The MPBoot method [7] tackles the problem of phylogenetic bootstrapping with only one tree search with the help of resampling parsimony score (REPS). On simulated data, the reliability assessed by MPBoot showed less biased than SBS. The MP score was compared among the intensive TNT method, the fast TNT method, MPBoot, and PAUP*. Since MPBoot only uses nearest neighbor interchange (NNI), subtree pruning and regrafting (SPR), and ratchet to rearrange the tree structure during the search, its MP score performance comes in second while the intensive TNT is first.

1.2 Contributions

In this paper, we propose to add TBR to MPBoot's collection of tree rearrangement operations to improve tree space sampling for MP phylogenetic bootstrapping problem. We present the MPBoot-TBR method with two options of TBR hill climbing integrated into the tree search algorithm, the bootstrap approximation algorithm of MPBoot and the MPBoot-ACO ant colony optimization algorithm combined 3 tree operations in the hill-climbing algorithm. We developed MPBoot-TBR and MPBoot-ACO with the help

of the phylogenetic likelihood library (PLL) [8] and provided the source code of MPBoot-TBR and MPBoot-ACO for free on GitHub. Moreover, we conducted experiments on benchmark datasets to compare the new methods and the MPBoot 2018 version (hereafter called "the original MPBoot") regarding MP score, bootstrap accuracy, and computation time.

2 Background

2.1 Problem Statement

The MP phylogenetic bootstrapping problem is as follows. Given a matrix A^{data} of size $n \times m$ that stores the MSA of n sequences and a number $\mathcal{B}$ specifying the number of bootstrap replicates, find the best tree T^{best} on A^{data} and set $\mathcal{B}$ of the best trees on the bootstrap MSAs A_b $(b = 1, \ldots, B)$ according to the MP principle.

By considering each branch in a tree as a bipartition for the set of leaf nodes, one can calculate the frequency in set $\mathcal{B}$ for each branch in the tree T^{best}. This is the bootstrap support value.

According to the MP principle, the objective function, or goodness function of a tree T, is the MP score, defined as the minimum number of substitutions to explain MSA A^{data} by the tree T, denoted as $MP(T|A^{data})$ (see Fig. 1) or $MP(T)$ if the context for the MSA is clear. The MP score can be calculated efficiently by the Fitch [9] or Sankoff dynamic programming algorithm [10]. In phylogenetic inference, one often works with unrooted binary trees. When calculating the MP score, we can choose to position the root on any branch for computational convenience. Which branch is selected does not affect the MP score.

Searching in the space of n-leaf binary trees for MP tree is an NP-complete problem [2].

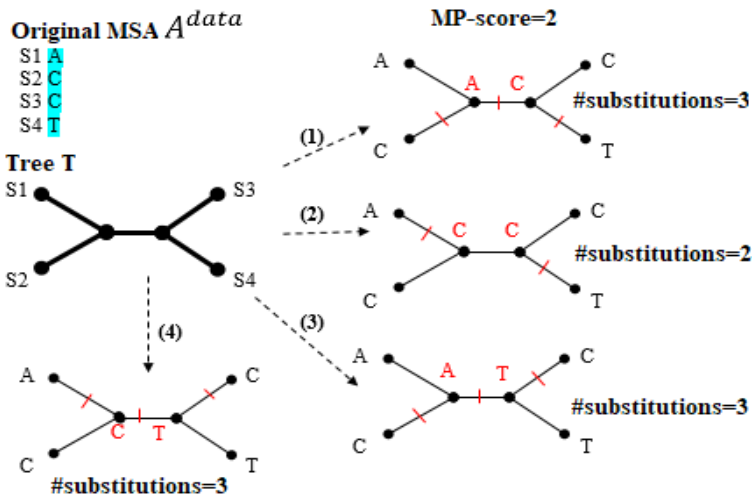

Fig. 1 Illustration for the fewest substitutions (i.e. the MP-score) required to explain the example A^{data} by the tree T. Here $MP(T|A^{data}) = 2$.

2.2 Common Tree Rearrangement Techniques

The effective sampling of the space of n-leaf binary trees for big n is achieved by various methods, each of which employs a unique approximation strategy. However, most rely on three tree rearrangement operations: the nearest neighbor interchange (NNI), the subtree pruning and regrafting (SPR), and the tree bisection and reconnection (TBR) [11].

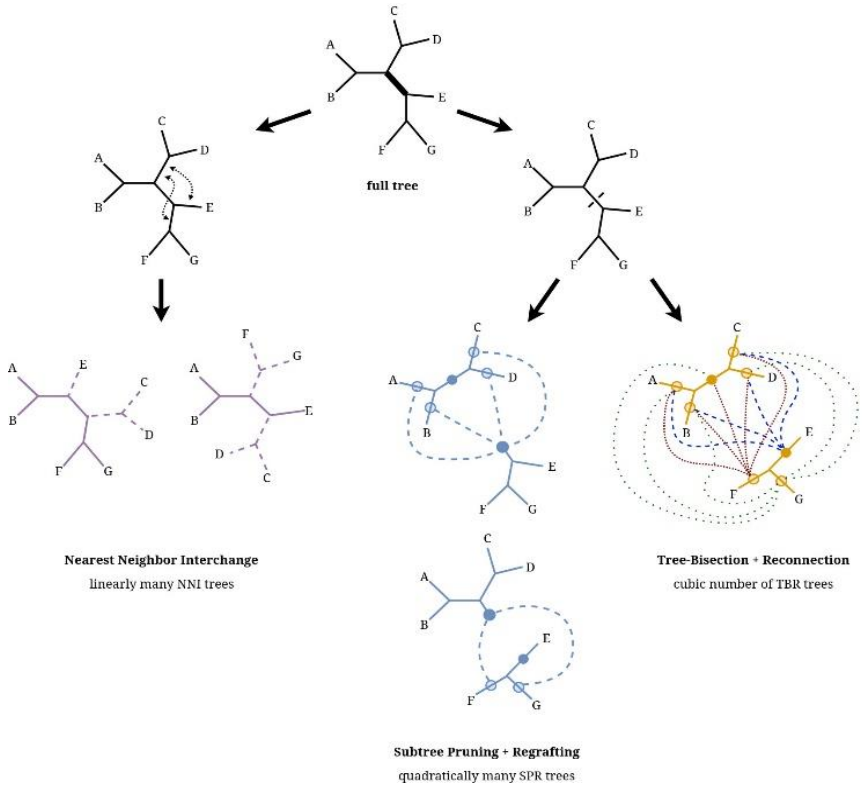

Fig. 2 The three tree rearrangements on an example tree of 7 taxa.

On a given branch, the NNI operation generates $O(1)$ neighbor solutions by swapping pairs of adjacent subtrees at the two endpoints of the selected branch (see Fig. 2). On a given remove-branch, the SPR operation generates $O(n)$ neighbor solutions by grafting the root from one side of the tree onto a branch from the other (see Fig. 2). On a given remove-branch, the TBR operation generates $O(n^2)$ neighbor solutions by rerooting on one of the branches of one side and then grafting it to one of the branches of the other side (see Fig. 2).

Because of the computational cost of SPR and TBR, approximation algorithms usually consider the relaxed variant, which examines the grafting points within a certain radius threshold relative to the pruning point. The SPR radius is the number of nodes

between the pruning and grafting points. The TBR radius counts on both sides the number of nodes between the pruning point and the grafting point.

2.3 Ant Colony Optimization

Ant Colony Optimization (ACO) algorithm is a method inspired by simulating the behavior of ants in nature, aimed at solving complex optimization problems in practice. Ants exchange information on the path they take through the pheromone trail left on the path. Paths with lower pheromone concentration will be eliminated, and ultimately, all ants will travel on the path with the potential to become the shortest path from the nest to the food source.

2.4 Related Works

SBS, as implemented in TNT and PAUP* tends to underestimate the probability of a branch being true, while the MPBoot support values appear less biased [7]. Therefore, the bootstrap support value by MPBoot is easier to interpret. A branch on T^{best} is considered reliable when the SBS bootstrap support value is greater than 70%, or the MPBoot bootstrap support value is greater than 95%.

Compared with the intensive TNT method, the fast TNT method, and PAUP* in terms of the best-found MP scores on the real benchmark dataset, MPBoot is right behind intensive TNT [7]. The intensive TNT method uses TBR and multiple advanced techniques such as ratchet, sectorial searches, tree fusing, and tree drifting for tree space exploration. MPBoot employs NNI, SPR within a given radius, and ratchet.

3 Methods

We propose to add TBR tree rearrangement to MPBoot. The integration is done through specific tasks: fast evaluation of a TBR move, fast TBR neighborhood search within a range of radius, efficient TBR hill-climbing, TBR integrated into MPBoot, and combining three hill-climbing operations using NNI, SPR, and TBR with the ant colony optimization (ACO) algorithm.

3.1 Uses of Tree Rearrangements in the Original MPBoot Algorithm

The MPBoot algorithm maintains a candidate set C consisting of n_C best-found trees for the original MSA A^{data}. This set is formed during the initialization phase (phase 1) by executing 100 times the randomized stepwise addition, followed by SPR hill-climbing, and then selecting the top n_C trees. Set C is improved during the exploration phase (phase 2) by repeatedly perturbing a randomly picked candidate tree and then performing SPR hill-climbing to the resultant tree. The perturbation is implemented by alternating (i) random NNI and (ii) ratchet with SPR hill-climbing. In addition, the bootstrap tree set $\mathcal{B}$ is updated along with the tree search. Finally, during

the refinement phase (phase 3), each bootstrap tree will be enhanced using SPR hill-climbing performed separately on the corresponding bootstrap MSA.

SPR tree rearrangement was used throughout the MPBoot. As a result, bringing a more comprehensive tree rearrangement than SPR to the framework may result in a better MP tree and a better bootstrap tree set.

3.2 Proposal for Fast Evaluation of a TBR Move

Consider a TBR move on tree T^{lst} (see Fig. 3A). Let R be the remove-branch of the TBR move. In general, cutting T^{lst} at R produces three separated parts, that is, two subtrees T_1, T_2 and branch R (see Fig. 3B). Following the cut, branch R will be used as an intermediate branch to connect T_1 and T_2 (see Fig. 3C). Suppose that I_1 and I_2 are the insert-branches (I_1 belongs to T_1, and I_2 belongs to T_2). Let T^* be the resulting tree after connecting I_1 and I_2.

In the straightforward approach, the evaluation of T^* is conducted by a full post-order traversal to calculate MP score by Fitch or Sankoff. However, many subtrees stay unchanged after the TBR move, therefore, their MP scores need not re-evaluated.

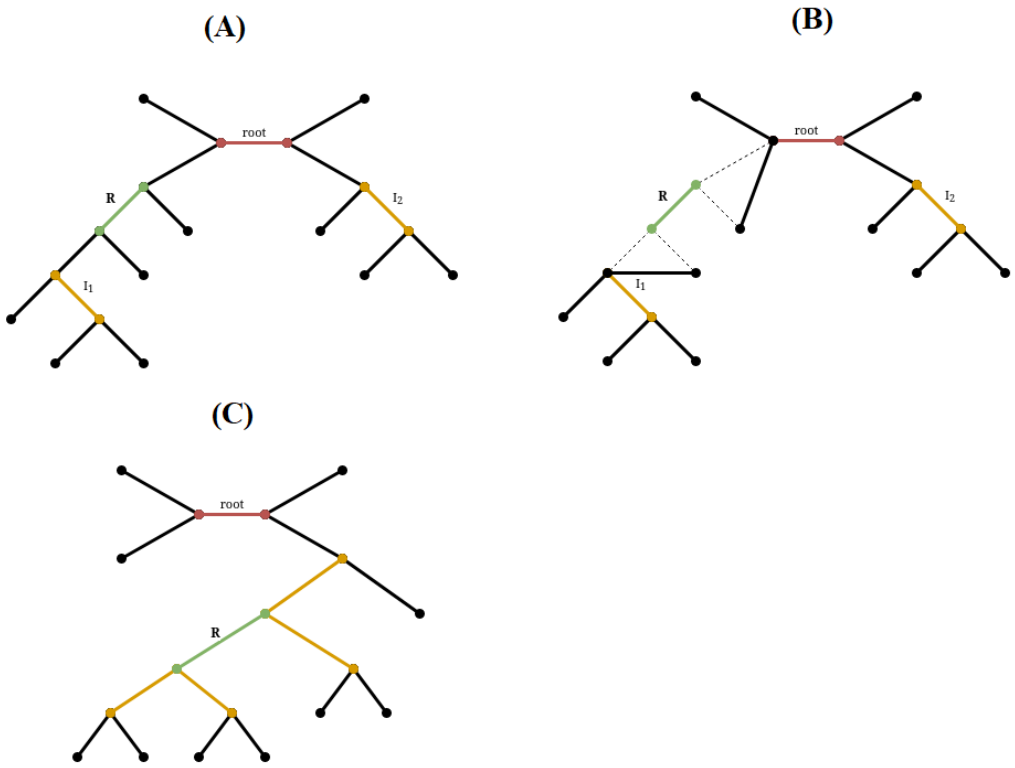

Fig. 3 A TBR move with remove-branch R and two insert branches I_1 and I_2.

Here, we propose to re-evaluate only the nodes whose score was changed. Suppose that T^{lst} has its MP score calculated when rooted at the branch $root$ that connects $root_1$ and $root_2$. Every node maintains the score of its corresponding subtree (see Fig. 4A). Now consider rooting T^* at branch R. The nodes whose score needs recomputation will lie on the path from I_1 to $root$ and from I_2 to $root$ (see Fig. 4B). To identify such nodes, we modify the data structure of a node to keep an extra pointer to its parent node that associates with T^{lst}. Thereby, we proceed by recursively jumping from I_1 and I_2 in tree T^{lst} to their parents until reaching the root of T^{lst} and marking every node in the path for recomputation. Finally, on tree T^* with the root placed at branch R, we will evaluate the MP score of the marked nodes and update their parent pointer accordingly.

Tree T^* will be served as tree T^{lst} for the subsequent TBR move.

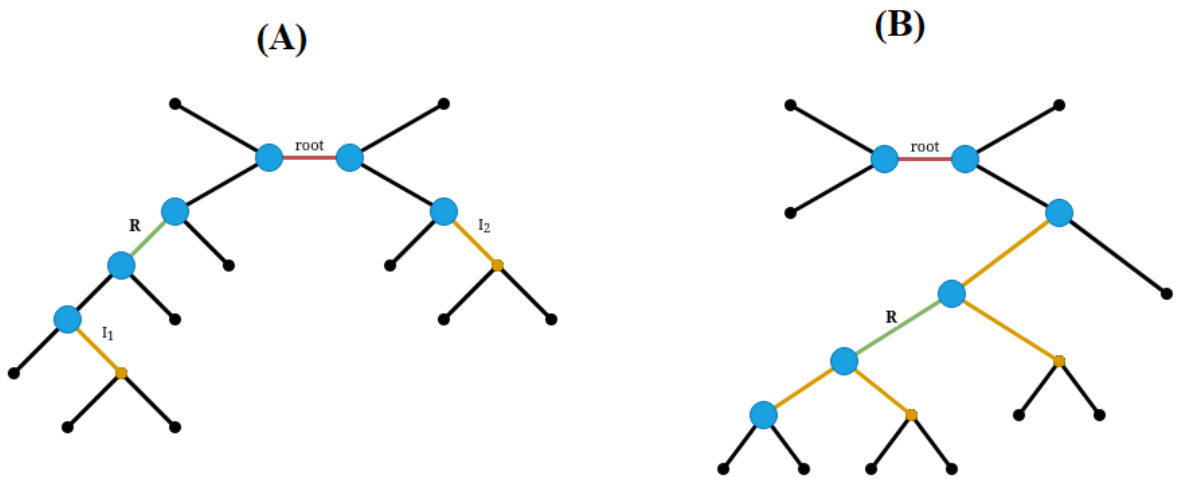

Fig. 4 Identifying nodes whose scores need recomputing with R as remove-branch and I_1 and I_2 as insert-branches.

3.3 Proposals for Search the TBR Neighborhood for a Given Remove-Branch and within a Radius Range

3.3.1 Best improvement strategy

We propose the TBR neighborhood search algorithm starting at tree T with remove-branch R as follows:

- Cut the tree at remove-branch R.
- Consider all of the insert-branch pairs (I_1, I_2) such that I_1 belongs to T_1, I_2 belongs to T_2, and the distance between I_1 and I_2 on the starting tree is in the given range $[mintrav, maxtrav]$ where $mintrav$ and $maxtrav$ specify the lower and upper bound for the TBR radius, respectively.

– Connect I_1 and I_2 via branch R (see Fig. 3C).

– T^* is the resulting tree. Evaluate the MP-score of tree T^* with the method proposed in **3.2**, then update the best-found neighbor tree $T^{bestNei}$.

– Cut the tree at remove-branch R again to examine the next (I_1, I_2) pairs.

- When the search process is complete, we obtain the best tree $T^{bestNei}$ in the TBR neighborhood of tree T with respect to remove-branch R.

If we reconnect R to its original position, we obtain the starting tree T.

The algorithm is described by pseudocode in the **Algorithm 1**.

Algorithm 1 Evaluating TBR moves with given remove-branch R on tree T

Input: Tree T,
Remove-branch R,
Radius criteria mintrav and maxtrav for insert-branch I_1 and I_2

Output: Best found tree $T^{bestNei}$ (best (I_1,I_2)) with remove-branch R

```
computeTBR(R, mintrav, maxtrav)
Function testTBRMove(I1,I2)
    # T is already cut into T1, T2 and R when called testTBRMove()
    Connect branch I1 and I2 using R, result in T*
    Evaluate parsimony score MP(T*) of T*
    if MP(T*) < MP(T^bestNei) then
        T^bestNei := T*
        I1^bestNei := I1
        I2^bestNei := I2
    end if
    Remove branch R, rollback the changes
Function computeTBR(R, mintrav, maxtrav)
    Remove branch R from tree T
    # Find all valid (I1,I2) can be done recursively via DFS
    for each (I1,I2) satisfied
        testTBRMove(I1,I2)
    end for
    Reconnect branch R, rollback to T
    T = applyTBR(R, I1^bestNei, I2^bestNei)
```

Therefore, with starting tree T, remove-branch R, TBR radius range $[mintrav, maxtrav]$, we will get the tree $T^{bestNei}$ and its associating insert-branch pair (I_1, I_2) after performing the procedure `computeTBR(R, mintrav, maxtrav)`.

Special cases where R is not an internal edge (connected to a leaf vertex) are handled separately due to the differences in the edge cutting and connecting process.

Furthermore, to quickly determine the score $MP(T^*)$, we change the root of tree T^{lst} to branch R after the first call to `testTBRMove` and preserve it until the assessment of the next remove-branch. This one is to ensure that the number of nodes to be recalculated does not exceed $O(maxtrav)$ (the first call does not count because $root$ could be different from R).

3.3.2 Better improvement strategy

To reach trees of greater distance compared to the best improvement strategy, we propose another TBR neighborhood search. In **Algorithm 1**, the current tree will be updated at most once for each remove-branch R. In this version **(Algorithm 2)**, the current tree will be updated at most once for each pairs of remove-branch R and insert-branch I_1. We consider all of the appropriate insert-branch I_2, find the best neighbor tree $T^{bestNei}$ and update the current tree if the score of $T^{bestNei}$ is better. When examining the next insert-branch I_1, tree $T^{bestNei}$ will be used as the starting tree.

Algorithm 2 Better improvement strategy

```
1: Function computeTBR(R, mintrav, maxtrav)
2:     for each I1 satisfied
3:         Remove branch R from tree T
4:         for each I2 satisfied
5:             testTBRMove(I1,I2)
6:         end for
7:         Reconnect branch R, rollback to T
8:         T = applyTBR(R, I1, I2^bestNei)
9:     end for
```

3.4 Proposal for the TBR Hill-Climbing

Our TBR hill-climbing algorithm updates tree T with $T^{bestNei}$ found (if $T^{bestNei}$ yields a better score) for each remove-branch R examined by **Algorithm 1** or **Algorithm 2**. The search algorithm repeats as long as the MP score of tree T still improves.

The algorithm is described with pseudocode in **Algorithm 3**.

Algorithm 3 Hill-climbing algorithm with TBR moves on tree T

Input: Tree T,
Radius criteria mintrav and maxtrav

Output: Tree T updated to best found neighbor tree $T^{bestNei}$ consider every remove-branch R

```
1: do
2:     for each branch R in T
3:         T^bestNei := NULL
4:         computeTBR(R, mintrav, maxtrav)
```

if $MP(T^{bestNei}) < MP(T)$ **then**
$T := T^{bestNei}$
end if
end for
while $MP(T)$ still improves
end do

Hill-climbing with two different types of TBR neighborhood search (which will be called two types of hill-climbing for simplicity) results in different search space sampling.

3.5 The MPBoot-TBR Framework Proposal

We propose MPBoot-TBR (**Algorithm 4**) by entirely substituting SPR hill-climbing with TBR hill-climbing and investigate two strategies of neighborhood search. If a search iteration of phase 2 fails to find a tree with an MP score strictly smaller than that of $T^{bestNei}$, it is called unsuccessful. The algorithm maintains the variable $n_{unsuccess}$ to track the number of consecutive unsuccessful iterations. The original MPBoot stops if $n_{unsuccess}$ meets n', the rounded-up value to the nearest hundredth of n. Due to the wide coverage of TBR in search space, in MPBoot-TBR, we adjust threshold $n' = 100$ following the similar framework of IQ-TREE [12].

Algorithm 4 The MPBoot-TBR method for bootstrap approximation

Input: an MSA A^{data} with n sequences,
the number of bootstrap MSAs B,
upperbound for TBR radius $maxtrav$

Output: A tree T^{best} with best found $MP(T^{best}|A^{data})$ and a set $\mathcal{B}$ of bootstrap trees $\{T_1, T_2, \dots, T_B\}$

Phase 1. Initialization

Generate bootstrap MSAs and initialize bootstrap tree set $\mathcal{B}$.
Initialize the threshold $MP_{max} := +\infty$.
Initialize the candidate set C for A^{data} with 100 random stepwise addition procedures followed by TBR hill-climbing

Phase 2. Exploration

do
Improve C by performing the perturbation on a randomly selected tree from the candidate set C and a subsequent TBR hill-climbing step. Every time a new tree, T, with $MP(T|A^{data}) < MP_{max}$ is encountered, execute REPS to update bootstrap tree set $\mathcal{B}$.
Update T^{best}, MP_{max}, $n_{unsuccess}$
while $n_{unsuccess} < n'$
end do

Phase 3. Bootstrap refinement

For each MP-tree T_b , do a hill-climbing TBR search and replace T_b by the new MP tree, if a better parsimony score is found.
output T^{best} the best MP tree that was found for A^{data}.

11: **output** set $\mathcal{B}$ and/or map the support values onto T^{best}.

3.6 The MPBoot-ACO Framework Proposal

We propose MPBoot-ACO as a replacement for the use of only SPR or TBR in hill climbing algorithm by introducing ant colony optimization (ACO) to determine the appropriate hill climbing operations (among NNI, SPR, TBR) in each hill climbing round. The algorithm maintains three "pheromone" values for the three hill climbing operations, respectively $\tau_{NNI}, \tau_{SPR}, \tau_{TBR}$. These "pheromone" values are large when the corresponding hill climbing operation has been effective in recent hill climbing rounds. In addition, three heuristic parameters $\eta_{NNI}, \eta_{SPR}, \eta_{TBR}$ are initialized to represent the initial priority for each hill climbing operation. Then, in each hill climbing round, the selection of the corresponding tree transformation operation follows the following probability function:

$$p_{NNI} = \frac{\tau_{NNI}\,\eta_{NNI}}{\Sigma\,\tau_i\,\eta_i}; \qquad p_{SPR} = \frac{\tau_{SPR}\,\eta_{SPR}}{\Sigma\,\tau_i\,\eta_i}; \qquad p_{TBR} = \frac{\tau_{TBR}\,\eta_{TBR}}{\Sigma\,\tau_i\,\eta_i}$$

After every $UPDATE\_ITERS$ hill-climbing iterations, the pheromone of the 3 tree transformations will be updated as follows: $\tau_i = (1-\rho)\tau_i + \Delta\tau_i$, where ρ is the pheromone evaporation coefficient and $\Delta\tau_i$ is the amount of pheromone change after UPDATE_ITERS hill-climbing iterations. Assuming the hill-climbing iteration uses SPR (similarly for other transformations) and the result finds a tree with a better MP score than the best score so far. Then, $\Delta\tau_{SPR} = \Delta\tau_{SPR} + 1$; otherwise, when SPR does not improve the score, update the pheromone for the remaining 2 transformations $\Delta\tau_{TBR} = \Delta\tau_{TBR} + 0.5$ and $\Delta\tau_{NNI} = \Delta\tau_{NNI} + 0.5$. All $\Delta\tau_i$ values are reset to 0 after each pheromone update (or after every $UPDATE\_ITERS$ hill-climbing iterations).

3.7 Implementation

We implemented MPBoot-TBR with C/C++ as open-source command line software, based on the open-source code of MPBoot and tree structures of the PLL library [8]. The command line to run MPBoot-TBR parsimony tree search on the original MSA given by <alignment_file> and use the TBR-best hill-climbing, has the following syntax:

```
mpboot -s <alignment_file> -tbr_pars
```

The syntax to search with the TBR-better hill-climbing is:

```
mpboot -s <alignment_file> -tbr_pars -tbr_restore_ver2
```

The syntax to search with MPBoot-ACO (default using TBR-best) is:

```
mpboot -s <alignment_file> -do_ant_colony
```

To perform the bootstrap approximation with B pseudo-replicates, add the "`-bb <B>`" to the command line.

4 Experimental evaluation

4.1 Experimental settings

We compare the original MPBoot with two MPBoot-TBR variants: one based on the best neighborhood search strategy (denoted as MPBoot-TBR-best) and the other based on a better neighborhood search strategy (denoted as MPBoot-TBR-better). Experiments were conducted for simulation and biological data on the high-performance computing system of the VNU University of Engineering and Technology. MPBoot-TBR was examined with $mintrav = 1, maxtrav = 5$, and the old stopping condition (as in the original MPBoot) because we observed that this radius settings took equivalent runtime as the original MPBoot. Besides, we also run MPBoot-TBR with $n' = 100$. For convenience, we append "sc100" to the method name to indicate the use of this stopping condition. The original MPBoot used a hill-climbing based on an SPR radius of 6. For all methods, parsimony ratchet for tree perturbing is enabled in even iterations.

In addition, we compared MPBoot-ACO with MPBoot-TBR-best (abbreviated as MPBoot-TBR in the results section) and the original MPBoot. The experiment was conducted on the TreeBASE biological dataset, taking the average of the scores and execution time after 5 random runs for each method. We ran the tree search algorithm (not bootstrap) and used the stopping condition of the original MPBoot. The initial parameters for running MPBoot-ACO included the number of hill-climbing iterations before each update, $UPDATE\_ITERS = 30$, the pheromone evaporation rate $\rho = EVAPORATION\_RATE = 0.6$, and the priority $\eta_{NNI}, \eta_{SPR}, \eta_{TBR} = 0.3, 0.35, 0.35$. We have explored MPBoot-ACO with many different parameter sets, and the above values provided the best results in terms of score and time. The parameters for running MPBoot-TBR-best (MPBoot-TBR) and the original MPBoot were set as mentioned in the above section.

4.2 Datasets

4.2.1 Simulated datasets

Using Seq-gen tool [13], we obtained the benchmark simulated datasets, which consists of three DNA datasets and two protein datasets, each having 200 MSAs (Table 1), evolved from trees following the Yule-Harding model [14]. This pipeline for assessing bootstrap accuracy has been utilized in previous studies [15]–[17].

4.2.2 Biological datasets

For real biological data we used the TreeBASE dataset of 70 DNA and 45 protein MSAs (Table 1) collected and selected from the TreeBASE online database [18] as described in [19].

Table 1 Summary of the datasets

Dataset	Sub-dataset	Data type	#MSAs	#sequences	#sites
YuleHarding	YH1	DNA	200	100	500
	YH2		200	200	1000
	YH3		200	500	1000
	YH4	Protein	200	100	300
	YH5		200	200	500
TreeBASE	TreeBASE-dna	DNA	70	201-767	976-61199
	TreeBASE-prot	Protein	45	50-194	126-22426

4.3 Evaluation metrics

4.3.1 MP Score

We first compare the methods by the MP score of the tree T^{best}. To this end, with method X and dataset Y, we calculate the percentage of MSAs in Y that method X obtains the best score among examined methods.

4.3.2 Bootstrap Accuracy

Compare the quality of set B of the bootstrap trees are achieved by analyzing the simulation data. For a bootstrap method, Z, we computed $f_Z(v)$ as the fraction of true branches out of all branches with bootstrap support values $v\%$ (counting across trees T^{best}) [20].

4.3.3 Computational Time

Here, we observe for each method the total runtime in hours for bootstrap approximation aggregated on the whole dataset.

4.4 Results

4.4.1 MP Score

As summarized in Table 2, on the Yule-Harding dataset, MPBoot-TBR variants and MPBoot obtain identical MP scores. On the TreeBASE dataset, MPBoot-TBR variants obtain better MP scores. Notably, MPBoot-TBR-sc100 frequencies are comparable to MPBoot-TBR.

Table 2 Frequency of obtaining best MP Score among examined MPBoot versions

Dataset	Sub-dataset	MPBoot	MPBoot-TBR-best	MPBoot-TBR-better	MPBoot-TBR-best-sc100	MPBoot-TBR-better-sc100
YuleHarding	YH1	100%	100%	100%	100%	100%
	YH2	100%	100%	100%	100%	100%
	YH3	100%	100%	100%	100%	100%
	YH4	100%	100%	100%	100%	100%
	YH5	100%	100%	100%	100%	100%
TreeBASE	TreeBASE-dna	63%	80%	74%	70%	74%
	TreeBASE-prot	87%	89%	96%	98%	93%

4.4.2 Bootstrap Accuracy

Function $f_{MPBoot}(v)$ (black curve), function $f_{MPBoot-TBR-best-sc100}(v)$ (orange curve), and function $f_{MPBoot-TBR-better-sc100}(v)$ (blue curve) for each Yule-Harding setting are visualized in Fig. 5. In all settings, the three curves appeared close to each other and lie above the diagonal indicating that the new version obtains the same bootstrap accuracy as MPBoot.

4.4.3 Computational Time

The total runtime of each method on a dataset (Table 3) shows a notable result that MPBoot-TBR-best-sc100 took less amount of time than the original MPBoot on real data. On simulated data, where the MP score can be found easier, MPBoot-TBR-best-sc100 appears comparable to the original MPBoot.

Table 3 Computational time of MPBoot and MPBoot-TBR

Dataset	Time summation over all MSAs (in hours)				
	Original MPBoot	TBR-best	TBR-better	TBR-best-sc100	TBR-better-sc100
YuleHarding	38.9	81.4	104.5	39	48.7
TreeBASE	27.2	56.1	59.9	22.7	28.1

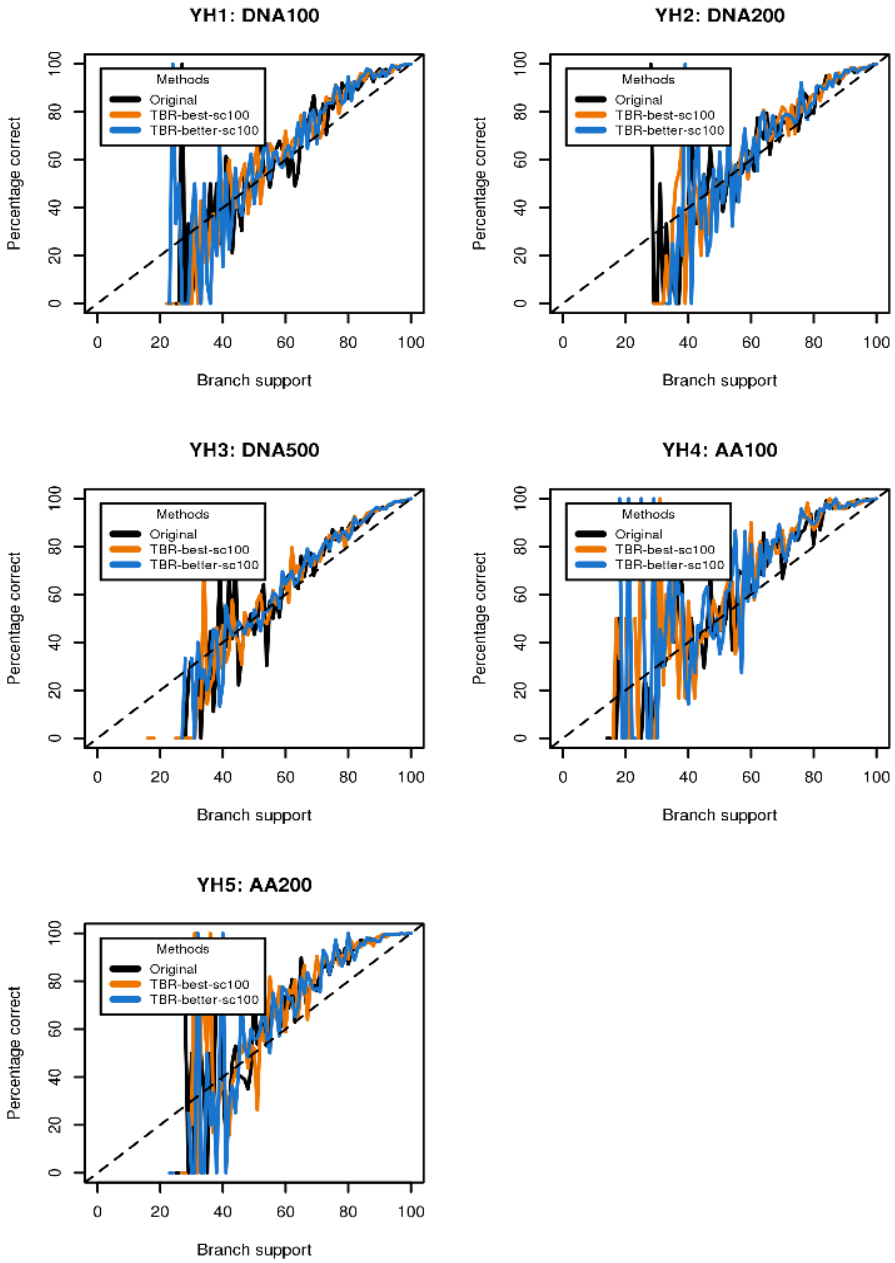

Fig. 5 Visualization for bootstrap accuracy on the 5 Yule-Harding datasets of the original MPBoot and MPBoot-TBR variants.

4.4.4 Comparison of MPBoot-ACO, MPBoot and MPBoot-TBR

The accuracy in terms of MP score and the total execution time averaged over 115 TreeBASE datasets with MPBoot and MPBoot-TBR-best (abbreviated as MPBoot-TBR) is summarized in Table 4.

MPBoot-ACO performs better in terms of MP score than both MPBoot and MPBoot-TBR, with a slower runtime than MPBoot but faster than MPBoot-TBR, demonstrating superiority over MPBoot-TBR.

Table 4 MP score accuracy and average total running time on the TreeBASE dataset of MPBoot, MPBoot-TBR, and MPBoot-ACO.

	MPBoot	MPBoot-TBR	MPBoot-ACO
Number of datasets with the best average score (out of 115 datasets)	90	92	96
Average total time (minutes)	227.1	279.1	249.8

In Table 5, by statistically analyzing the usage rate of hill-climbing operators in MPBoot-ACO on datasets with different characteristics, we can see the advantages of MPBoot-ACO. On average, MPBoot-ACO uses 16.4% NNI, 39.1% SPR, and 44.4% TBR. In datasets where MPBoot-ACO performs better than MPBoot-TBR, MPBoot-ACO uses more SPR hill-climbing and fewer NNI hill-climbing iterations. In datasets where the scores are equal and MPBoot-ACO has faster computation time than MPBoot-TBR, the usage rate of NNI hill-climbing iterations increases while SPR and TBR decrease. In particular, in protein datasets (which are simpler datasets in TreeBASE) where MPBoot-ACO achieves equivalent scores and faster computation time than MPBoot-TBR, the usage rate of NNI hill-climbing iterations increases significantly compared to the average. This demonstrates that MPBoot-ACO selects appropriate hill-climbing operators for the difficulty level of the data, thereby increasing the program's efficiency.

Table 5 Details of the proportion of the use of different hill-climbing methods in the TreeBASE dataset of MPBoot-ACO.

Evaluation Sample	%NNI	%SPR	%TBR
On the entire TreeBASE	16.4	39.1	44.4

On datasets where ACO is better than TBR	13.9	43.4	42.7
On datasets where ACO is equal to TBR and ACO is faster than TBR	19.7	39.7	40.6
On protein datasets where ACO is equal to TBR and ACO is faster than TBR	22.1	39.3	38.6

5 Discussion and Conclusions

We proposed the MPBoot-TBR algorithm by thoroughly investigating the TBR rearrangement operation and employing TBR to improve the tree sampling performance of the MPBoot framework [7]. Accordingly, the new algorithm aims at improving both the best-found MP tree and the bootstrap tree set. Moreover, to alleviate the computational cost of TBR for a scalable framework, we considered the general forms of TBR neighborhood search with parameterized radius. Furthermore, we propose the MPBoot-ACO algorithm based on the TBR transformation in MPBoot-TBR and the two available transformations NNI and SPR. This combined algorithm aims to improve the score and running time by exploiting the strengths of different hill-climbing options.

The proposed algorithm, MPBoot-TBR, incorporates a number of algorithmic solutions for specific tasks, including fast evaluation of a TBR operation, fast TBR neighborhood search for a given remove-branch, fast TBR hill-climbing, and a revised bootstrap approximation framework integrating TBR hill-climbing. For fast evaluation of a TBR move, we proposed to combine the idea of marking the nodes for recomputation and the idea of re-rooting. The re-rooting technique was also utilized to significantly reduce the number of nodes having MP score recomputed between two consecutive re-graftings, from $O(n)$ to $O(maxtrav)$. We investigated two neighborhood search strategies, namely TBR-best and TBR-better and adjusted the stopping condition of the framework to adapt for TBR convergence. Since the original MPBoot was thoroughly compared with other tools in [7], this paper focuses on comparing the MPBoot-TBR variants with the original MPBoot to assess the impact of the TBR-related proposals.

The bootstrap accuracy of the MPBoot-TBR was comparable to that of MPBoot on all YH settings. On both simulated and real benchmark datasets, the MPBoot-TBR variants yielded equivalent or better MP scores than MPBoot. The YH datasets were simulated under a simple evolutionary model to assess parsimony bootstrap accuracy under different data types and input sizes rather than to assess tree search performance. Therefore, it is easier to find the MP scores for YH than for the real dataset TreeBASE.

All proposed methods obtained 100% of the best-known MP scores on the YH indicate that they passed the sanity check. Notably, MPBoot-TBR variants with $n' = 100$ and the best improvement strategy (MPBoot-TBR-best-sc100) performed with less computing time than the original did on real data. We chose radius $maxtrav = 5$ to balance the performance in terms of MP score and runtime after examining the TBR-best under other radii. Radius 3 analyzed TreeBASE in 11.7 hours, twice as fast but with worse scores than the original MPBoot. Radius 7 analyzed TreeBASE in 54.6 hours, twice as slow as the original MPBoot, but the score was equivalent to radius 5 with the old n'.

MPBoot-ACO yields superior results compared to both original MPBoot and MPBoot-TBR-best in terms of MP scores. Moreover, MPBoot-ACO also outperforms MPBoot-TBR-best in terms of running time. In easier datasets, MPBoot-ACO selects hill-climbing moves to improve program efficiency (since strong hill-climbing moves will not be effective after finding the optimal MP score). This demonstrates the reasonable exploitation of the strengths of tree transformation operations in MPBoot-ACO.

In future studies, we will investigate improvements to MPBoot-TBR in both MP score and running time. Firstly, we can optimize the recalculation of tree score so that the computational complexity does not depend on the $maxtrav$ radius, which can explore distant branches without additional cost, and the optimization can be applied to both TBR and SPR. Secondly, we can study the improvement of hill-climbing traversal to avoid exploring multiple TBR moves leading to the same tree structure. Thirdly, we can study the adjustment of the limit on the set of trees used as REPS to ensure bootstrap accuracy while exploring fewer trees.

We have implemented the proposed algorithm based on open-source software MPBoot, available at https://github.com/HynDuf/mpboot/tree/Huynh_Tien_Dung for MPBoot-TBR, and https://github.com/HynDuf/mpboot/tree/ant-colony-optimization for MPBoot-ACO.

Related Publications

- T. D. Huynh, Q. T. Vu, V. D. Nguyen and D. T. Hoang, "Employing tree bisection and reconnection rearrangement for parsimony inference in MPBoot," *2022 14th International Conference on Knowledge and Systems Engineering (KSE)*, Nha Trang, Vietnam, 2022, pp. 1-6, doi: 10.1109/KSE56063.2022.9953773.

References

[1] J. Herron and S. Freeman, *Evolutionary Analysis*, 5th ed. Pearson, 2013. doi: 10.1163/156854009X409126.

[2] R. L. Graham and L. R. Foulds, "Unlikelihood that minimal phylogenies for a realistic biological study can be constructed in reasonable computational time," *Math Biosci*, vol. 60, no. 2, pp. 133–142, Aug. 1982, doi: 10.1016/0025-5564(82)90125-0.

[3] S. Wooding, "Inferring Phylogenies," *Am J Hum Genet*, vol. 74, p. 1074, May 2004.

[4] J. Felsenstein, "CONFIDENCE LIMITS ON PHYLOGENIES: AN APPROACH USING THE BOOTSTRAP," *Evolution (N Y)*, vol. 39, no. 4, pp. 783–791, Jul. 1985, doi: 10.1111/j.1558-5646.1985.tb00420.x.

[5] P. A. Goloboff, J. S. Farris, and K. C. Nixon, "TNT, a free program for phylogenetic analysis," *Cladistics*, vol. 24, no. 5, pp. 774–786, Oct. 2008, doi: 10.1111/j.1096-0031.2008.00217.x.

[6] D. Swofford, "PAUP*. Phylogenetic Analysis Using Parsimony (*and Other Methods)," 2002, doi: 10.1111/j.0014-3820.2002.tb00191.x.

[7] D. T. Hoang, L. S. Vinh, T. Flouri, A. Stamatakis, A. von Haeseler, and B. Q. Minh, "MPBoot: fast phylogenetic maximum parsimony tree inference and bootstrap approximation," *BMC Evol Biol*, vol. 18, no. 1, p. 11, Dec. 2018, doi: 10.1186/s12862-018-1131-3.

[8] T. Flouri *et al.*, "The Phylogenetic Likelihood Library," *Syst Biol*, vol. 64, no. 2, pp. 356–362, Mar. 2015, doi: 10.1093/sysbio/syu084.

[9] W. M. Fitch, "Toward Defining the Course of Evolution: Minimum Change for a Specific Tree Topology," *Syst Zool*, vol. 20, no. 4, p. 406, Dec. 1971, doi: 10.2307/2412116.

[10] D. Sankoff, "Minimal Mutation Trees of Sequences," *SIAM J Appl Math*, vol. 28, no. 1, pp. 35–42, Jan. 1975, doi: 10.1137/0128004.

[11] P. Lemey, *The Phylogenetic Handbook*, 2nd ed. Cambridge: Cambridge University Press, 2009. doi: 10.1017/CBO9780511819049.

[12] B. Q. Minh *et al.*, “IQ-TREE 2: New Models and Efficient Methods for Phylogenetic Inference in the Genomic Era,” *Mol Biol Evol*, vol. 37, no. 5, pp. 1530–1534, May 2020, doi: 10.1093/molbev/msaa015.

[13] A. Rambaut and N. C. Grass, “Seq-Gen: an application for the Monte Carlo simulation of DNA sequence evolution along phylogenetic trees,” *Bioinformatics*, vol. 13, no. 3, pp. 235–238, 1997, doi: 10.1093/bioinformatics/13.3.235.

[14] E. F. Harding, “The probabilities of rooted tree-shapes generated by random bifurcation,” *Adv Appl Probab*, vol. 3, no. 1, pp. 44–77, Jul. 1971, doi: 10.2307/1426329.

[15] B. Q. Minh, M. A. T. Nguyen, and A. von Haeseler, “Ultrafast Approximation for Phylogenetic Bootstrap,” *Mol Biol Evol*, vol. 30, no. 5, pp. 1188–1195, May 2013, doi: 10.1093/molbev/mst024.

[16] D. T. Hoang, L. S. Vinh, T. Flouri, A. Stamatakis, A. von Haeseler, and B. Q. Minh, “A new phylogenetic tree sampling method for maximum parsimony bootstrapping and proof-of-concept implementation,” in *2016 Eighth International Conference on Knowledge and Systems Engineering (KSE)*, Oct. 2016, pp. 1–6. doi: 10.1109/KSE.2016.7758020.

[17] N. Pham and D. T. Hoang, “A distributed algorithm for the parsimony bootstrap approximation,” in *2021 13th International Conference on Knowledge and Systems Engineering (KSE)*, Nov. 2021, pp. 1–6. doi: 10.1109/KSE53942.2021.9648648.

[18] M. J. Sanderson, M. J. Donoghue, W. Piel, and T. Eriksson, “TreeBASE: a prototype database of phylogenetic analyses and an interactive tool for browsing the phylogeny of life,” *Am J Bot*, vol. 81, p. 183, 1994.

[19] L.-T. Nguyen, H. A. Schmidt, A. von Haeseler, and B. Q. Minh, “IQ-TREE: A Fast and Effective Stochastic Algorithm for Estimating Maximum-Likelihood Phylogenies,” *Mol Biol Evol*, vol. 32, no. 1, pp. 268–274, Jan. 2015, doi: 10.1093/molbev/msu300.

[20] D. M. Hillis and J. J. Bull, “An Empirical Test of Bootstrapping as a Method for Assessing Confidence in Phylogenetic Analysis,” *Syst Biol*, vol. 42, no. 2, pp. 182–192, Jun. 1993, doi: 10.1093/sysbio/42.2.182.