Supplementary Information

# UFBoot2: Improving the Ultrafast Bootstrap Approximation

Diep Thi Hoang, Olga Chernomor, Arndt von Haeseler, Bui Quang Minh and Le Sy Vinh

## Method

### Speeding up Felsenstein's pruning algorithm

Here, we will provide a computationally more efficient version of the pruning algorithm (Felsenstein 1981). We assume a *reversible* Markov model of sequence evolution with rate matrix $Q$. W.l.o.g., we explain the method for DNA sequences with character states $\{A, C, G, T\}$ and i.i.d. rates.

### 1. Felsenstein's pruning algorithm

Given a multiple sequence alignment $\boldsymbol{A}$ with sites $A_1, \dots, A_s$, the log-likelihood of tree $T$ (with branch lengths) and rate matrix $Q$ is computed as:

$$\ell(\boldsymbol{A}\,|T,Q) = \sum_{i=1}^{s} \log\big(L(A_i|T,Q)\big), \quad (1)$$

where $L(A_i|T,Q)$ is the *site likelihood* of alignment site $i$.

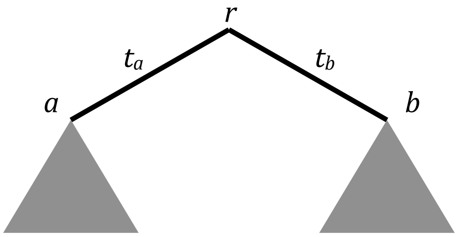

**Figure M1.** A phylogenetic tree $T$ used to illustrate Felsenstein's algorithm. It is arbitrarily rooted at an internal node $r$ with two direct descending nodes $a$ and $b$ and corresponding branch lengths $t_a$ and $t_b$.

To reduce the expensive computation of $L(A_i|T,Q)$, Felsenstein (1981) introduced a *pruning algorithm*. Here, $T$ is arbitrarily rooted at a node $r$ with two descending nodes $a$ and $b$. The pruning algorithm recursively traverses the tree to compute the so-called *partial likelihood vector* $L_i^a(x)$, where $x \in \{A, C, G, T\}$ for the subtree rooted at $a$ (Fig. M1). Similarly, one computes the partial likelihood vector at node $b$: $L_i^b(x)$. These are combined to compute the partial likelihood vector at the root:

$$L_i^r(x) = \left(\sum_y p_{xy}(t_a) L_i^a(y)\right)\left(\sum_y p_{xy}(t_b) L_i^b(y)\right), \qquad (2)$$

where

$$\left(p_{xy}(t)\right)_{x,y\in\{A,C,G,T\}} = P(t) = e^{Qt}$$

is the probability of substitution from state $x$ to $y$ during evolutionary time $t$. Note that $L_i^a(x)$ and $L_i^b(x)$ are computed in a similar manner to eq. (2) from the descendants of $a$ and $b$, respectively.

The site likelihood is then:

$$L(A_i|T,Q) = \sum_x \pi_x L_i^r(x), \qquad (3)$$

where $\pi_x$ is the equilibrium frequency of state $x$.

Because $Q$ is reversible, the site likelihood does not change as long as the branch lengths $t_a + t_b = t$ stays fixed, the Pulley principle (Felsenstein 1981). In other words, one can set $t_a = 0$ (moving $r$ to $a$). As a result, $P(t_a) = P(0)$ becomes the identity matrix and thus combining eq. (2) and (3) gives us:

$$L(A_i|T,Q) = \sum_x \pi_x\, L_i^a(x)\left(\sum_y p_{xy}(t) L_i^b(y)\right). \qquad (4)$$

In essence, Felsenstein's algorithm is a dynamic programming algorithm. It has a time complexity of $O(nsc^2)$, where *n, s* and *c* are the number of sequences, sites and character states (4 for DNA), respectively. The space complexity is $O(nsc)$ to store the partial likelihood vectors for all internal nodes of the tree.

**2. Eigen-decomposition**

Before we explain the more efficient implementation of the branch length computation we start with a particular feature of reversible rate matrices and their similarity transforms.

Let $Q$ be the rate matrix of the general time reversible model (Lanave et al. 1984) and $\Pi = \text{diag}(\pi_A, \pi_C, \pi_G, \pi_T)$ the diagonal matrix of equilibrium state frequencies. Thus, the matrix

$$Q_1 = \Pi^{1/2} \cdot Q \cdot \Pi^{-1/2}$$

is symmetric with real eigenvalues and real eigenvectors. Moreover, one can compute an orthogonal matrix $W$ of eigenvectors of $Q_1$ such that

$\Lambda = W^T \cdot Q_1 \cdot W$, with $W^T \cdot W = I = \text{diag}(1,1,1,1)$.

$\Lambda$ is the diagonal matrix of eigenvalues of $Q_1$ (and also of $Q$).

We obtain

$$\Lambda = W^T \cdot \Pi^{1/2} \cdot Q \cdot \Pi^{-1/2} \cdot W$$

Due to associativity of matrix multiplication

$$W^T \cdot \Pi^{1/2} \cdot \Pi^{-1/2} \cdot W = I.$$

Thus, $U^{-1} = W^T \cdot \Pi^{1/2}$ and $U = \Pi^{-1/2} \cdot W$and $U$ is the matrix of eigenvectors for $Q$.

$u_{xy}^{-1} = w_{xy}^T \sqrt{\pi_y}$ and $u_{yx} = w_{yx} / \sqrt{\pi_y}$. Because $w_{xy}^T = w_{yx}$ we obtain

$$u_{xy}^{-1} = \pi_y u_{yx}. \quad (5)$$

Eq. (5) will be used later.

## 3. Speeding up branch length estimation

To compute $\ell(\boldsymbol{A}|T,Q)$, one needs to estimate all branch lengths of $T$, which in turn dominates the runtime. Here, one traverses the tree to optimize each branch at a time by e.g., the Newton-Raphson method (Olsen et al. 1994). The tree traversal is repeated until the log-likelihood converges. Thus, a common operation is to compute $\ell(\boldsymbol{A}|T,Q)$ given a length $t$ for a branch $(a,b)$ connecting two nodes $a$ and $b$. Because eq. (4) is repeatedly applied when optimizing *t*, one needs to pre-compute the partial likelihood vectors $L_i^a(.)$ and $L_i^b(.)$ to save the computations. Thus, the computation cost of eq. (4) is $sc^2$ for a given branch length *t*. In the following, we present the technique to reduce this cost to $sc$, i.e., a factor of $c$ faster than the naïve application of eq. (4).

As derived above we have:

$$Q = U \cdot \Lambda \cdot U^{-1}.$$

Thus,

$$P(t) = e^{Qt} = U \cdot e^{\Lambda t} \cdot U^{-1}.$$

$e^{\Lambda t}$ is the diagonal matrix of eigenvalue exponentials. In other words, we have:

$$p_{xy}(t) = \sum_z u_{xz} e^{\lambda_z t} u_{zy}^{-1}, \quad (6)$$

for all states $x$ and $y$, where $u_{xz}$ and $u_{zy}^{-1}$ are the entries of the eigenvectors matrix $U$ and $U^{-1}$. Plugging RHS of eq. (6) into eq. (4) gives us

$$L(A_i|T,Q) = \sum_x \pi_x L_i^a(x) \left( \sum_y \sum_z u_{xz} e^{\lambda_z t} u_{zy}^{-1} L_i^b(y) \right).$$

Rearranging the sums in this equation with the observation that $\pi_x u_{xz} = u_{zx}^{-1}$ (eq. 5) gives us:

$$L(A_i|T,Q) = \sum_z e^{\lambda_z t} \left(\sum_x u_{zx}^{-1} L_i^a(x)\right)\left(\sum_y u_{zy}^{-1} L_i^b(y)\right). \qquad (7)$$

Denote the two sums in parentheses of eq. (7) as $V_i^a(z)$ and $V_i^b(z)$, we have:

$$L(A_i|T,Q) = \sum_z e^{\lambda_z t}\, V_i^a(z) V_i^b(z). \qquad (8)$$

Comparing eq. (4) with (8), we achieve a reduction from two nested sums to just one sum, given that one stores the two vectors $V_i^a(z)$ and $V_i^b(z)$ instead of the partial likelihood vectors of $L_i^a(x)$ and $L_i^b(x)$. The computational cost of *V* is twice that of the partial likelihood vectors, but in return the branch length estimation using eq. (8) is *c* times faster than eq. (4).

## 4. The fast pruning algorithm

The new pruning algorithm will compute and store *V* instead of the partial likelihood vectors for every internal node of the tree. To this end, plugging RHS of eq. (6) into eq. (2) gives us:

$$L_i^r(x) = \left(\sum_y \sum_z u_{xz} e^{\lambda_z t_a}\, u_{zy}^{-1} L_i^a(y)\right)\left(\sum_y \sum_z u_{xz} e^{\lambda_z t_b}\, u_{zy}^{-1} L_i^b(y)\right).$$

Rearranging this equation and replacing *L* by *V*:

$$L_i^r(x) = \left(\sum_z u_{xz} e^{\lambda_z t_a} V_i^a(z)\right)\left(\sum_z u_{xz} e^{\lambda_z t_b} V_i^b(z)\right). \qquad (9)$$

And vector *V* at the root is computed as:

$$V_i^r(z) = \sum_x u_{zx}^{-1} L_i^r(x). \qquad (10)$$

Taking together the fast pruning algorithm proceeds as follows:

1. Perform a post-order tree traversal to compute vectors *V* for all nodes from *V* vectors of the descendant nodes using eqs. (9) and (10). This computation is twice more expensive than computing partial likelihood vectors (eq. 2).
2. Apply eq. (8) to estimate the branch length for any branch $(a,b)$ given that $V_i^a$ and $V_i^b$ were computed. This computation is 4, 20 and 61 times faster than applying eq. (4) for DNA, protein and codon models, respectively.

## References

Felsenstein J. 1981. Evolutionary trees from DNA sequences: a maximum likelihood approach. J. Mol. Evol. 17:368–376.

Lanave C, Preparata G, Saccone C, Serio G. 1984. A new method for calculating evolutionary substitution rates. J. Mol. Evol. 20:86–93.

Olsen GJ, Matsuda H, Hagstrom R, Overbeek R. 1994. fastDNAmL: a tool for construction of phylogenetic trees of DNA sequences using maximum likelihood. Comput. Appl. Biosci. 10:41–48.

# Figures

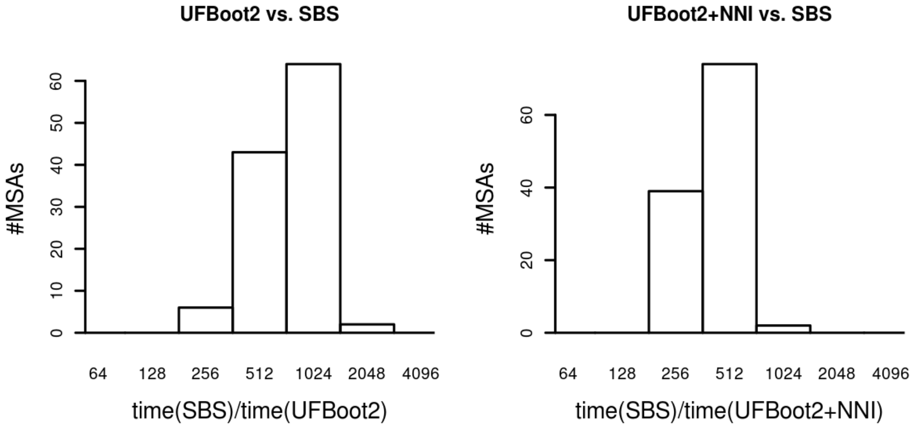

**Figure S1.** Distributions of runtime ratios between SBS and UFBoot2 (left) and between SBS and UFBoot2+NNI (right) on 115 empirical MSAs.

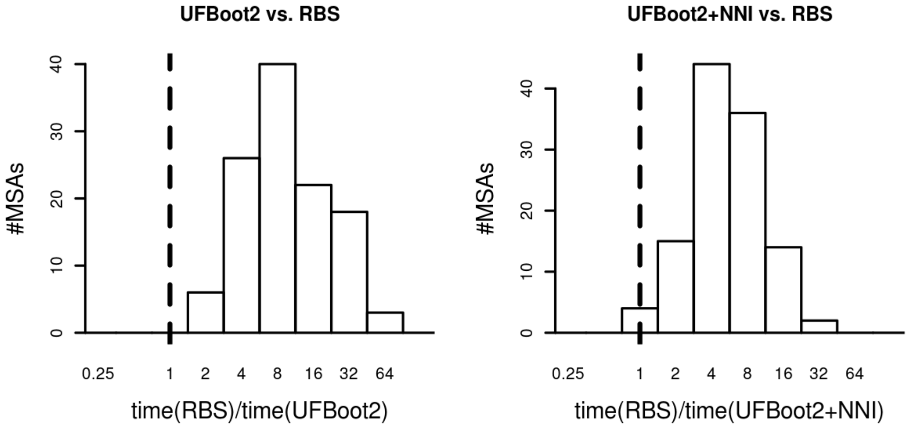

**Figure S2.** Distributions of runtime ratios between RBS and UFBoot2 (left) and between RBS and UFBoot2+NNI (right) on 115 empirical MSAs.

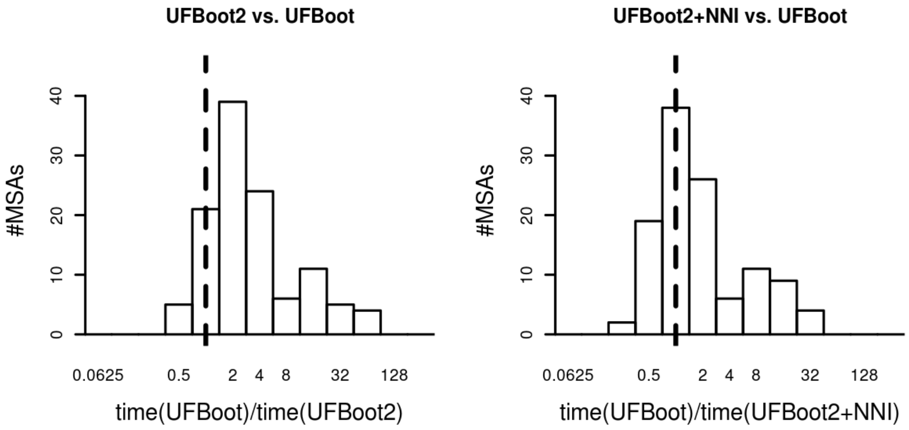

**Figure S3.** Distributions of runtime ratios between the original UFBoot and UFBoot2 (left) and between original UFBoot and UFBoot2+NNI (right) on 115 empirical MSAs.

# Table

**Table S1.** Command-lines for running IQ-TREE and RAxML bootstrap methods used in this study.

| Method | Software version | Example command-line |
|---|---|---|
| Original UFBoot | IQ-TREE 0.9.6 | iqtree -s example.phy -m GTR+G -bb 1000 -p 0.5 |
| UFBoot2 | IQ-TREE 1.6.beta5 | iqtree -s example.phy -m GTR+G -bb 1000 |
| UFBoot2+NNI | IQ-TREE 1.6.beta5 | iqtree -s example.phy -m GTR+G -bb 1000 -bnni |
| SBS | IQ-TREE 1.6.beta5 | iqtree -s example.phy -m GTR+G -b 100 |
| RAxML search | RAxML 8.2.9 | raxmlHPC-SSE3 -f d -m GTRGAMMA -p $RANDOM -s example.phy -n raxsearch.example.phy |
| RAxML rapid bootstrap with bootstopping | RAxML 8.2.9 | raxmlHPC-SSE3 -s example.phy -m GTRGAMMA -n rbs.example.phy -x $RANDOM -N autoMRE -p $RANDOM |