Title

FastTree: Computing Large Minimum-Evolution Trees with Profiles instead of a Distance Matrix

Submission type

Research Article

Authors

Morgan N. Price, Paramvir S. Dehal, Adam P. Arkin

Institution

Physical Biosciences Division, Lawrence Berkeley National Lab, 1 Cyclotron Road Mailstop 922, Berkeley California 94720, USA

Current Affiliations

All authors: Physical Biosciences Division, Lawrence Berkeley National Lab, Berkeley, USA; Virtual Institute of Microbial Stress and Survival, Lawrence Berkeley National Lab, Berkeley California, USA; A.P.A.: Department of Bioengineering, University of California, Berkeley, California, USA

Corresponding Author

Address: Morgan Price, Lawrence Berkeley National Lab, 1 Cyclotron Road Mailstop 922, Berkeley California 94720, USA
Phone: 510-643-3722
Fax: 510-643-3721
E-mail: morgannprice@yahoo.com

Key words

minimum evolution, neighbor joining, large phylogenies

Running head

FastTree

Abstract

Gene families are growing rapidly, but standard methods for inferring phylogenies do not scale to alignments with over 10,000 sequences. We present FastTree, a method for constructing large phylogenies and for estimating their reliability. Instead of storing a distance matrix, FastTree stores sequence profiles of internal nodes in the tree. FastTree uses these profiles to implement neighbor-joining and uses heuristics to quickly identify candidate joins. FastTree then uses nearest-neighbor interchanges to reduce the length of the tree. For an alignment with N sequences, L sites, and a different characters, a distance matrix requires $O(N^2)$ space and $O(N^2L)$ time, but FastTree requires just $O(NLa + N\sqrt{N})$ memory and $O(N\sqrt{N}\log(N)La)$ time. To estimate the tree's reliability, FastTree uses local bootstrapping, which gives another 100-fold speedup over a distance matrix. For example, FastTree computed a tree and support values for 158,022 distinct 16S ribosomal RNAs in 17 hours and 2.4 gigabytes of memory. Just computing pairwise Jukes-Cantor distances and storing them, without inferring a tree or bootstrapping, would require 17 hours and 50 gigabytes of memory. In simulations, FastTree was slightly more accurate than neighbor joining, BIONJ, or FastME; on genuine alignments, FastTree's topologies had higher likelihoods. FastTree is available at http://microbesonline.org/fasttree.

Introduction

Inferring phylogenies from biological sequences is the fundamental method in molecular evolution and has many applications in taxonomy and for predicting structure and biological function. In general, sequences are identified as homologous and aligned, and then a phylogeny is inferred. Large alignments can be constructed efficiently, in time linear in the number of sequences, by aligning

the sequences to a profile instead of to each other, as with position-specific blast or hmmalign (Schaffer et al. (2001); http://hmmer.janelia.org/).

Given an alignment, neighbor joining and related minimum-evolution methods are the fastest and most scalable approaches for inferring phylogenies (Saitou and Nei, 1987; Studier and Keppler, 1988; Desper and Gascuel, 2002). All of these methods rely on a distance matrix that stores an estimate of the evolutionary distance between each pair of sequences. Computing an entry in the distance matrix requires comparing the characters at each position in the alignment and hence requires $O(L)$ time, where L is the number of positions. Thus, the distance matrix takes $O(N^2L)$ time to compute, where N is the number of sequences, and $O(N^2)$ space to store.

Given a distance matrix, neighbor joining performs a greedy search for a tree of minimal length, according to a local estimate of the length of each branch (Gascuel and Steel, 2006). More specifically, neighbor joining begins with the tree as a star topology, and it iteratively refines the tree, by joining the best pair of nodes together, until the tree is fully resolved. Each step considers $O(N^2)$ possible joins, so the standard neighbor-joining algorithm requires $O(N^3)$ time to infer a tree from a distance matrix. This can be reduced to $O(N^2)$ or $O(N^2 \log N)$ time, either by using heuristics to consider fewer joins (Elias and Lagergren, 2005; Evans et al., 2006) or by using additional $O(N^2)$ memory (Simonsen et al., 2008; Zaslavsky and Tatusova, 2008). FastME is another minimum-evolution method that takes only $O(N^2)$ time (Desper and Gascuel, 2002). With any of these optimized methods, the $O(N^2L)$ time to compute the distance matrix dominates the time.

As DNA sequencing accelerates, the memory and CPU requirements of the distance-matrix approach are becoming prohibitive. For example, an alignment of full-length 16S ribosomal RNAs contains over 160,000 distinct sequences (De-

Santis et al. (2006); http://greengenes.lbl.gov). Similarly, the MicrobesOnline database, which provides phylogenies for all protein families from prokaryotic genomes, contains protein families with over 100,000 distinct sequences (Alm et al. (2005); http://www.microbesonline.org/). The distance matrix for families with 100,000-200,000 members requires 20-80 gigabytes (GB) of memory to store (a 4-byte floating point value for each of $N(N-1)/2$ pairs). Although computers with this much memory are available, the typical node in a compute cluster has an order of magnitude less memory. Furthermore, DNA sequencing technology is improving rapidly, and the distance matrix's size scales as the square of the family's size, so we expect these problems to become much more severe. Finally, most of the methods that construct a tree from a distance matrix in $O(N^2)$ time, such as FastME and the exact $O(N^2)$ implementations of neighbor joining, require additional $O(N^2)$ memory.

Whatever the method used, inferred phylogenies often contain errors, and so it is important to estimate the reliability of the result (Nei et al., 1998). The standard method to estimate reliability is to use the bootstrap: to resample the columns of the alignment, to rerun the method 100-1,000 times, to compare the resulting trees to each other or to the tree inferred from the full alignment, and to count the number of times that each split occurs in the resulting trees (Felsenstein, 1985). (A split is the two sets of leaves on either side of an internal edge.) Unfortunately, bootstrapping is a minimum of 100 times slower than the underlying phylogenetic inference, and comparing the trees to each other is also a non-trivial computation. In principle, the resampled trees could be compared to the original tree in $O(N^2)$ time and $O(N)$ space by hashing the splits in the tree. However, the tree-comparison tools that we are aware of require $O(N^3)$ time and $O(N^2)$ space.

Although building phylogenetic trees for large gene families is challenging,

it is important to do so and not just to build trees for small sets of selected homologs. Analyzing all of the sequences is important for taxonomy, for predicting gene function, for classifying environmental DNA sequences, and for identifying functional residues (Eisen, 1998; Engelhardt et al., 2005; von Mering et al., 2007; Lichtarge et al., 2003). Furthermore, omitting sequences might change the biological interpretation of the result, especially in prokaryotes: because of horizontal gene transfer, it is difficult to know which homologs are relevant without building a tree. Finally, for web sites that support interactive use of phylogenetic trees, it is desirable to compute trees for all of the genes beforehand (Li et al. (2006); http://www.treefam.org/; http://www.microbesonline.org/).

Our Approach

We present FastTree, which uses four ideas to reduce the space and time complexity of inferring a phylogeny from an alignment (Figure 1). First, FastTree implements neighbor joining by storing profiles for the internal nodes in the tree instead of storing a distance matrix. Each profile includes a frequency vector for each position, and the profile of an internal node is the weighted average of its childrens' profiles. For example, if we join two leaves i and j, and i has an A at a position and j has a G, then the profile of ij at that position will be 50% A and 50% G (and 0% for other characters). The intuition behind using profiles is that the average of the distances between the sequences in two subtrees A and B equals the distance between profile(A) and profile(B), because profile(A) is the average of the sequences in A. FastTree uses these profiles to compute the distances between internal nodes in the tree and also the total distance from a node to all other nodes, which is also required for neighbor joining. The profiles require a total of $O(NLa)$ space, where a is the size of the alphabet (20 for protein sequences and 4 for nucleotide sequences), instead of $O(N^2)$ space

for the distance matrix. However, the time required for neighbor-joining with exhaustive search rises from $O(N^3)$ to $O(N^3La)$, because every distance has to be recomputed on demand in $O(La)$ time.

Second, FastTree uses a combination of previously-published heuristics (Elias and Lagergren, 2005; Evans et al., 2006) and a new "top hits" heuristic to reduce the number of joins considered. Whereas traditional neighbor joining considers $O(N^3)$ possible joins, and optimized variants have considered $O(N^2)$ possible joins (the size of the distance matrix), FastTree considers $O(N\sqrt{N}\log N)$ possible joins. Thus, in theory, FastTree takes $O(N\sqrt{N}\log(N)La)$ time. In practice, FastTree is faster than computing the distance matrix. These heuristics require additional $O(N\sqrt{N})$ memory, raising the total storage requirement for FastTree to $O(NLa + N\sqrt{N})$, which is still much less than $O(N^2)$.

Third, FastTree refines the initial topology with nearest-neighbor interchanges (NNIs). Given an unrooted tree ((A,B), (C,D)), where A, B, C, and D may be sub-trees rather than individual sequences, FastTree compares the profiles of A, B, C, and D, and determines whether alternate topologies ((A,C),(B,D)) or ((A,D),(B,C)) would reduce the length of the tree. These NNIs are similar to those of FastME, although FastME uses a distance matrix (Desper and Gascuel, 2002). FastTree's NNIs take $O(N\log(N)La)$ additional time and $O(NLa)$ additional space. In practice, the NNIs take much less time than computing the initial topology, and they improve the quality of the tree.

Fourth, FastTree computes a local bootstrap value for each internal split ((A,B), (C,D)) by resampling the columns of the profiles and counting the fraction of resamples that support ((A,B), (C,D)) over the alternate topologies ((A,C), (B,D)) or ((A,D), (B,C)). The local bootstrap has been used for maximum-likelihood trees (Kishino et al., 1990), but cannot be used with distance matrices. Computing the local bootstrap takes $O(bNLa)$ time, where b

is the number of bootstrap samples. Even with 1,000 resamples, this takes less than a minute for an alignment of over 8,000 protein sequences and 394 columns. Thus, local bootstrap gives FastTree an additional 100-fold speed-up over distance matrix methods, in which the entire computation must be repeated for each sample. However, the local bootstrap should be interpreted more conservatively than the traditional bootstrap. Whereas traditional bootstrap estimates the probability that the split is correct (Efron et al., 1996), local bootstrap estimates the probability that the split is correct if we assume that A, B, C, and D are subtrees of the true tree.

Below, we describe FastTree in more detail. Then, we show that in realistic simulations, FastTree is slightly more accurate than other minimum-evolution methods such as neighbor joining, BIONJ, or FastME. On genuine alignments, FastTree topologies tend to have higher likelihoods than topologies from other minimum-evolution methods, which also suggests that FastTree gives higher-quality results. For both simulated and genuine alignments, FastTree's heuristics do not lead to any measurable reduction in quality. For large families, FastTree requires less CPU time and far less memory than computing and storing a distance matrix. Finally, we show that the local bootstrap is a good indicator of whether each split in the inferred topology is correct, and it is orders of magnitude faster than the traditional bootstrap. We believe that FastTree is the first practical method for computing accurate phylogenies, including support values, for alignments with tens or hundreds of thousands of sequences.

Materials and Methods

FastTree

A rough outline of FastTree is shown at the bottom of Figure 1. Before we explain how FastTree implements neighbor joining, we explain how it computes distances between sequences and how it computes distances between profiles. We then explain how it computes distances between internal nodes and how it calculates the neighbor joining criterion, which is used to select the best join. We also describe the heuristics that it uses to reduce the number of joins that it considers. Finally, we explain the steps after neighbor joining: nearest-neighbor interchanges, the local bootstrap, and estimating the branch lengths for the final topology. For formulas, derivations, and technical details, see Supplementary Note 1.

Distances Between Sequences

FastTree uses both corrected and uncorrected distances. FastTree corrects the distances for multiple substitutions during NNIs, computing final branch lengths, and local bootstrap, but not during neighbor-joining. For nucleotide sequences, FastTree's uncorrected distance d_u is the fraction of positions that differ, and the corrected distance is the Jukes-Cantor distance $d = -\frac{3}{4}\log(1-\frac{4}{3}d_u)$. For protein sequences, FastTree estimates distances by using the BLOSUM45 amino acid similarity matrix (Henikoff and Henikoff, 1992) and a log-correction inspired by that of scoredist (Sonnhammer and Hollich, 2005). We scaled the BLOSUM45 similarity matrix into a dissimilarity matrix such that the average dissimilarity between each amino acid and a random amino acid is 1 if we use the non-uniform amino acid frequencies of biological sequences. The uncorrected distance d_u between two sequences is the average dissimilarity among non-gap positions, and the corrected distance is $d = -1.3 \cdot \log(1 - d_u)$. The

intuitive justification is that the term within the logarithm ranges from 1 for identical sequences to an expected value of 0 for unrelated sequences, as with Jukes-Cantor distances for nucleotide sequences. For both nucleotide and protein sequences, FastTree truncates the corrected distances to a maximum of 3.0 substitutions per site, and for sequences that do not overlap because of gaps, FastTree uses this maximum distance.

Distances Between Profiles

FastTree uses profiles to estimate the average distance between the children of two nodes. The profile distance at each position is the average dissimilarity of the characters. The uncorrected distance between two profiles is then the average of these position-wise distances, weighted by the product of the proportion of non-gaps in each of the two profiles. FastTree computes the distance between two profiles in O(La) time by using the eigen decomposition of the dissimilarity matrix.

The profile distance is identical to the average distance if the distances are not corrected for multiple substitutions and if the sequences do not contain gaps. For example, if we join two sequences A and B together, then the profile distance

$$\Delta(AB, C) = \frac{d_u(A, C) + d_u(B, C)}{2}.$$

Of course, we do wish to correct for multiple substitutions, and in practice, large alignments always contain gaps. In these cases, the profile-based average becomes an approximation of the average distances used in traditional minimum-evolution methods.

First, consider the issue of correcting distances for multiple substitutions with a formula of the form $d \propto -\log(1 - d_u)$. The average corrected distance between A and BC is $(d(A, B) + d(A, C))/2$, or the average of two logarithms.

However, FastTree cannot compute this average of logarithms from the profiles. Instead, FastTree uses the logarithm of averages. This is a close approximation if the distances are short or if the distances are similar. If the distances are large, then distances between profiles may be more accurate than averages of distances (Müller et al., 2004).

Second, consider what happens if the sequences contain gaps. FastTree records the fraction of gaps at each profile position, and when computing distances, FastTree weights positions by their proportion of non-gaps. Traditional neighbor-joining implicitly weights the ungapped columns more highly. For example, consider an alignment with A=C-, B=GG, and C=CC: $\Delta(AB, C) = 2/3$, but $(d_u(A, B)+d_u(A, C))/2 = 1/2$. Both approaches treat gaps as missing data, and it is not obvious which is preferable.

Distances Between Internal Nodes

Neighbor joining operates on distances between internal nodes rather than on average distances between the members of subtrees. For example, after joining nodes A and B, neighbor joining sets

$$d_u(AB, C) = \frac{d_u(A, C) + d_u(B, C) - d_u(A, B)}{2}.$$

FastTree instead sets the profile of AB to $\vec{P}(AB) = (\vec{P}(A) + \vec{P}(B))/2$, and computes the distance between nodes with

$$d_u(i, j) = \Delta(i, j) - u(i) - u(j),$$

where $\Delta(i, j)$ is the profile distance and $u(i)$ is the "up-distance," or the average distance of the node from its children. $u(i) = 0$ for leaves, and for balanced joins, $u(ij) = \Delta(i, j)/2$. This profile-based computation gives the exact same value of

$d_u(i,j)$ as neighbor joining after any number of joins, as long as distances are not corrected for multiple substitutions and the sequences contain no gaps.

FastTree actually uses weighted joins, as in BIONJ (Gascuel, 1997), rather than the balanced joins. In BIONJ, the weight of each join depends on the variance of the distance between two joined nodes, which can also be computed from the profiles. Also, with weighted joins, the formula for the up-distances becomes more complicated.

Calculating the Neighbor-Joining Criterion

Given the distances between nodes, neighbor joining selects the join that minimizes the criterion $d_u(i,j) - r(i) - r(j)$, where i, j, k are indices of active nodes that have not yet been joined, $d_u(i,j)$ is the distance between nodes i and j, n is the number of active nodes and

$$r(i) \equiv \sum_{k \neq i} d_u(i,k)/(n-2).$$

$r(i)$ can be thought of as the average "out-distance" of i to other active nodes (although the denominator is $n-2$, not $n-1$). Traditional neighbor joining computes all N out-distances before doing any joins, which takes $O(N^2)$ time, and updates each out-distance after each join, which also takes $O(N^2)$ time overall. To avoid this work, FastTree computes each out-distance as needed in $O(La)$ time by using a "total profile" T which is the average of all active nodes' profiles, as implied by

$$\sum_{k \neq i} \Delta(i,k) = n \cdot \Delta(i,T) - \Delta(i,i).$$

($\Delta(i,i)$ is the average distance between children of i, including self-comparisons.) If there are gaps, then this is an approximation. FastTree computes the total

profile at the beginning of neighbor joining in $O(NLa)$ time, updates it incrementally in $O(La)$ time, and recomputes it every 200 joins to avoid round-off error.

Notice that FastTree does not log-correct the distances during neighbor joining. We considered doing so, but it reduced FastTree's accuracy. Perhaps the profile-based out-distances become inaccurate: the out-distance is an average of both far and small values, and so the log-correction of the average distance is a poor estimate of the average of the log-corrected distances.

Selecting the Best Join

FastTree uses heuristics to reduce the number of joins considered at each step to less than $O(n)$. We first explain the "top hits" heuristic. For each node, FastTree records a top-hits list: the nodes that are the closest m neighbors of that node, according to the neighbor-joining criterion. By default, $m = \sqrt{N}$. Before doing any joins, FastTree estimates these lists for all N sequences by assuming that if A and B have similar sequences, then the top-hits lists of A and B will largely overlap. More precisely, FastTree computes the $2m$ top hits of A, where the factor of two is a safety factor. Then, for each node B within the top m hits of A that does not already have a top-hits list, FastTree estimates the top-hits of B by comparing B to the top $2m$ hits of A. In theory, this takes a total of $O(N^2L/m + NmL) = O(N\sqrt{N}L)$ time to compute and $O(Nm) = O(N\sqrt{N})$ space to store.

FastTree restricts the top hits heuristic to ensure that a sequence's top hits are only inferred from the top hits of "close enough" neighbor. Because of these restrictions, it is not clear how many sequences will have $O(m)$ close neighbors, and it is not clear if the initial computation of top-hits lists will truly take $O(N\sqrt{N}L)$ time. However, for large alignments, it takes less time than computing the distance matrix, so in practice it takes less than $O(N^2L)$ time.

FastTree maintains these top hits lists during neighbor joining. First, after a join, FastTree computes the top-hits list for the new node in $O(mLa)$ time by comparing the node to all entries in the top hits lists of its children. Second, after a join, some of the other nodes' top hits may point to an inactive (joined) node. When FastTree encounters these entries, it replaces them with the active ancestor. Finally, as the algorithm progresses, the top-hits lists will gradually become shorter, as joined nodes become absent from lists. Thus, FastTree periodically "refreshes" the top hit list by comparing the new node to all other nodes, and also by comparing each of the new node's top hits to each other. Each refresh takes $O(nLa + m^2La)$ time and ensures that the top-hits lists of $O(m)$ other nodes are of full length and up-to-date, so FastTree performs $O(\sqrt{N})$ refreshes, and they take a total of $O(N\sqrt{N}La)$ time.

Besides storing the list of top hits for each node, FastTree also remembers the best known join for each node, as in FastNJ (Elias and Lagergren, 2005). FastTree updates the best known join whenever it considers a join that involves that node. For example, while computing the top hits of A, it may discover that A,B is a better join than B,best(B).

Based on the best joins and the top-hits lists, FastTree can quickly select a join. First, FastTree finds the best m joins among the best known joins of the n active nodes, without recomputing the neighbor-joining criterion to reflect the current out-distances. In principle, this can be implemented in $O(m \log N)$ time per join by using a priority queue. (FastTree simply sorts the entries, which adds $O(N \log N)$ time per join or $O(N^2 \log N)$ time overall.) For those m candidates, FastTree recomputes the neighbor-joining criterion, which takes $O(mLa)$ time, and selects the best. Furthermore, FastTree does a local hill-climbing search to find a better join, as in relaxed neighbor-joining (Evans et al., 2006): given a join AB, it considers all joins AC or BD, where C is in top-

hits(A) or D is in top-hits(B). This can be beneficial because the out-distances change after every join, so the best join for a node can change as well. In theory, this takes $O(\log n)$ iterations (Evans et al., 2006), $O(m \log(n) La)$ time per join, or $O(N\sqrt{N} \log(N) La)$ time overall. Thus, it takes FastTree a total of $O(N\sqrt{N} \log(N) La)$ time to maintain the top-hits lists and to select all of the joins.

Nearest-neighbor Interchanges

After FastTree constructs an initial tree with neighbor joining, it uses nearest-neighbor interchanges to improve the tree topology. During each round, FastTree tests and possibly rearranges each split in the tree, and it recomputes the profile of each internal node. The profiles can change even if the topology does not change because FastTree recomputes the weighting of the joins.

By default, FastTree does $\log_2(N) + 1$ rounds of NNIs. We chose a fixed number of rounds, instead of iterating until no more NNIs occur, to ensure fast completion. We chose roughly $\log_2(N)$ rounds so that, on a balanced topology, a misplaced node could migrate all of the way across the tree.

The minimum evolution criterion prefers ((A,B), (C,D)) over alternate topologies ((A,C), (B,D)) or ((A,D), (B,C)) if $d(A,B) + d(C,D) < d(A,C) + d(B,D)$ and $d(A,B) + d(C,D) < d(A,D) + d(B,C)$. Here, FastTree uses log-corrected profile distances, rather than distances between nodes. The profile distances do not account for the distances within the nodes, but this does not affect the minimum evolution criterion, as it increases all distances $d(A,\cdot)$ by the same amount.

For larger topologies, FastTree must compute profiles for additional subtrees before doing this computation. For example, consider the topology ((A,(B,C)), D, E). After neighbor-joining, FastTree has profiles for the internal nodes BC and ABC as well as for the leaves, but to test the split BC versus ADE requires

the profile for DE. FastTree computes the profile for DE by doing a weighted join of D and E, using the weighting of BIONJ for a 4-leaf tree (Gascuel, 1997). FastTree stores these additional profiles along the path to the root and reuses them when possible. (FastTree computes an unrooted tree but stores it as a rooted tree.) To ensure that a round of NNIs takes O(NLa) time and at most O(NLa) additional space, FastTree visits nodes in post-order (it visits children before their parents).

Local Bootstrap

To estimate the support for each split, FastTree resamples the alignment's columns with Knuth's 2002 random number generator (http://www-cs-faculty.stanford.edu/~knuth/programs/rng.c). FastTree counts the fraction of resamples that support a split over the two potential NNIs around that node, much as it does while using NNIs to improve the topology. If a resample's minimum-evolution criterion gives a tie then that resample is counted as not supporting the split.

Branch Lengths

Once the topology is complete, FastTree computes branch lengths, with

$$d(AB,CD) = \frac{d(A,C)+d(A,D)+d(B,C)+d(B,D)}{4} - \frac{d(A,B)+d(C,D)}{2}$$

for internal branches and

$$d(A,BC) = \frac{d(A,B)+d(A,C)-d(B,C)}{2}$$

for the branch leading to leaf A, where d are log-corrected profile distances.

Unique Sequences

Large alignments often contain many sequences that are exactly identical to each other (Howe et al., 2002). Before inferring a tree, FastTree uses hashing to quickly identify redundant sequences. It constructs a tree for the unique subset of sequences, and then creates multifurcating nodes, without support values, as parents of the redundant sequences.

Testing FastTree

Sources of Alignments

We obtained sequences of members of COG gene families (Tatusov et al., 2001) and members of Pfam PF00005 (Finn et al., 2006) from the fall 2007 release of the MicrobesOnline database (http://www.microbesonline.org/). We aligned the sequences to the family's profile, using reverse position-specific blast for the COG alignment (Schaffer et al., 2001) and hmmalign for the PF00005 alignment (http://hmmer.janelia.org/). As the profiles only include positions that are present in many members of the family, these alignments do not contain all positions from the original sequences. The 16S rRNA alignment is from greengenes and is trimmed with the greengenes mask (DeSantis et al. (2006); http://greengenes.lbl.gov).

To simulate alignments with realistic phylogenies and realistic gaps, we used the COG alignments. In each simulation, we selected the desired number of sequences from a COG alignment, we removed positions that were over 25% gaps, we estimated a topology and branch lengths with PhyML (Guindon and Gascuel, 2003), we estimated evolutionary rates across sites with phylip's proml (http://evolution.genetics.washington.edu/phylip.htm), we simulated sequences with Rose (Stoye et al., 1998), and we re-introduced the gaps from the original alignment. For simulations of 5,000 sequences, we used FastTree instead

of PhyML and we assigned evolutionary rates at random. For $N = 10$, we simulated 3,100 alignments (10 independent runs per family); for $N = 50$, we simulated 3,099 alignments; for $N = 250$, we simulated 308 alignments; for $N = 1,250$, we simulated only 92 alignments because some PhyML jobs did not complete, and for $N = 5,000$, we simulated 7 alignments, as only 7 families contained enough non-redundant sequences. See Supplementary Note 2 for technical details.

CPU Timings

All programs used a single thread of execution. We used a computer with 2 dual-core 2.6 GHz AMD Opteron processors and 32 GB of RAM. However, for the two long-running maximum-likelihood jobs in Table 6, we used a computer with a 2.4 GHz Intel Q6600 quad-core processor and 8 GB of RAM. The two machines have similar performance (about 20% different for FastTree).

To estimate performance on large alignments, we extrapolated from the largest feasible alignment for that method and its theoretical complexity. Inferring a tree from a distance matrix requires $O(N^2)$ space and either $O(N^2)$ time (FastME and RapidNJ (Simonsen et al., 2008)), $O(N^2 \log N)$ time (Clearcut), or $O(N^3)$ time (QuickTree and BIONJ). Computing bootstrap values from resampled trees with phylip's consense or with QuickTree's built-in bootstrap requires $O(N^2)$ space and $O(N^3)$ time. For QuickTree, which identifies and removes duplicate sequences, we used the number of unique sequences rather than the total number.

Results

Topological Accuracy in Simulations

We tested FastTree and other methods for inferring phylogenies on simulated protein alignments with realistic topologies, realistic gaps, varying evolutionary rates across sites, and between 10 and 5,000 sequences. The simulated alignments ranged from 64-1,009 positions (median 304), with 9% gaps, and on average, pairs of sequences within these alignments were 33% identical. For each alignment and for each method, we counted the proportion of splits that were correctly inferred.

As shown in Table 1, FastTree was significantly more accurate other minimum-evolution methods but was 1-2% less accurate than PhyML, a maximum-likelihood method (Guindon and Gascuel, 2003). We will show that FastTree scales to far larger alignments than current maximum-likelihood methods can handle. Furthermore, most of the splits that disagree between minimum-evolution and maximum-likelihood trees are poorly supported (Nei et al., 1998). This is true in our simulations as well, even for the splits that PhyML inferred correctly but FastTree missed (data not shown). Thus, the practical effect of these differences may be much less than 1-2%.

After FastTree, the next best method was FastME, which like FastTree uses nearest-neighbor interchanges according to the minimum evolution criterion (Desper and Gascuel, 2002). Depending on the number of sequences, FastTree was slightly but significantly more accurate than FastME, or the two methods were tied. FastTree was up to 4% more accurate than BIONJ, a weighted variant of neighbor joining (Gascuel, 1997), when run with FastTree's log-corrected distances. BIONJ with log-corrected distances was about as accurate as BIONJ with maximum-likelihood distances from phylip's protdist, so FastTree's distance measure is adequate. Maximum-likelihood distances that were estimated

using a model with gamma-distributed rates gave poor results. FastTree was 1-5% more accurate than QuickTree, an implementation of traditional neighbor joining (Howe et al., 2002), and 4-6% more accurate that Clearcut, an implementation of relaxed neighbor joining (Evans et al., 2006). Clearcut is more scalable than the other distance-matrix methods but not as scalable as FastTree (see below).

We obtained similar results with a standard set of simulations of ungapped nucleotide alignments (Desper and Gascuel, 2002) or with ungapped protein simulations (Supplementary Tables 1 & 2). Furthermore, FastTree was more accurate than BIONJ regardless of how strongly the tree deviated from the molecular clock or how divergent the sequences were (Supplementary Figure 1).

These simulations also confirm that topologies can be inferred even when there are many more sequences than sites (Bininda-Emonds et al., 2001). The alignments with 5,000 sequences contained just 197-384 sites, yet FastTree identified 76.3% of the splits correctly.

Effectiveness of FastTree's Approximations and Heuristics

The simulations also let us test the internals of FastTree. First, FastTree's neighbor-joining phase should give roughly the same results as BIONJ with uncorrected distances. In practice, the two methods had very similar accuracies, as did FastTree's neighbor-joining with exhaustive search (Table 2). Thus, FastTree's accuracy was not affected by its approximations to handle gaps or by its heuristics to reduce the number of joins considered. Heuristic search was also over 100 times faster: for an alignment of 1,250 proteins with 338 positions, the neighbor-joining phase of FastTree took 1,551 seconds with exhaustive search but only 8 seconds with heuristic search.

Second, using uncorrected distances only reduced the accuracy of BIONJ by

around 3% (Table 2). This is consistent with a previous simulation study of realistic topologies and protein alignments (Hollich et al., 2005). Because using uncorrected distances leads to relatively few errors, FastTree can correct these errors by doing a few rounds of NNIs. Adding more rounds of NNIs did not increase accuracy (Table 2).

Quality of Trees for Genuine Alignments

To test the quality of FastTree's results on genuine protein families, we inferred topologies for alignments of 500 randomly selected sequences from large COGs. These alignments ranged from 65 to 1,009 positions, and within each alignment, the average pair of sequences were 27% identical. To quantify the quality of each topology, we used PhyML to optimize the branch lengths and compute the log-likelihood. We ran PhyML with the JTT model of amino acid substitution and four categories of gamma-distributed rates.

In Table 3, we report the average difference in log-likelihood between that method's trees and FastTree's trees. The methods are sorted by the average difference. All of the distance-matrix methods gave significantly worse average likelihoods than FastTree (paired t test, all $P < 10^{-20}$). Furthermore, as in the simulations, FastTree's approximations and heuristics did not reduce the quality of the trees (Supplementary Table 3). Overall, we found that for these genuine alignments, FastTree's topologies were of high quality.

We also tested the quality of FastTree trees for sets of 500 non-redundant sequences from a large 16S ribosomal RNA alignment (DeSantis et al. (2006); http://greengenes.lbl.gov). To quantify the quality of each topology, we used PhyML with the HKY85 model, which accounts for the higher rate of transitions over transversions, and four categories of gamma-distributed rates. FastTree found topologies with higher likelihoods than most of the distance-matrix

methods (Table 4). FastME did outperform FastTree slightly if given maximum-likelihood distances that account for the higher rate of transitions than transversions. Distinguishing transitions from transversions might further improve FastTree's topologies.

CPU Time and Memory Required to Infer Trees

We tested FastTree and other methods on a protein alignment from the COG database (COG2814), a domain alignment from PFam (PF00005), and a trimmed alignment of full-length 16S rRNAs (Tatusov et al. (2001); Finn et al. (2006); http://greengenes.lbl.gov). These alignments contain roughly 8,000-150,000 distinct sequences (Table 5). Running the distance-matrix methods on the larger alignments was not feasible, so we extrapolated from smaller alignments (see Methods). The actual or estimated CPU time and memory usage are shown in Table 6.

The maximum-likelihood methods we tested, PhyML 3 (Guindon and Gascuel, 2003) and RAxML VI (Stamatakis, 2006), did not complete in 50 days on the smallest of these problems, which took FastTree about 3 minutes. (Despite the high usage of virtual memory by PhyML, both PhyML and RAxML ran at over 99% CPU utilization.) Even for COG alignments of just 1,250 proteins, PhyML 3 typically took over a week. Thus, current maximum-likelihood methods do not scale.

Most of the methods require a distance matrix as input, so in practice, the running time is the time to compute a distance matrix plus the time to infer a tree. As shown in Table 6, FastTree is over 1,000 times faster than computing maximum-likelihood protein distances. For the 16S rRNA alignment, FastTree is as fast as computing Jukes-Cantor distances and over 100 times faster than computing maximum-likelihood distances with gamma-distributed rates.

For the 16S alignment, the only method other than FastTree that seems practical is Clearcut: all of the other methods would require over 1,000 hours or over 500 gigabytes of memory. Clearcut itself is very fast – we estimate that it might take only 12 hours to infer a tree from the 16S distance matrix. However, Clearcut requires a distance matrix, and FastTree is faster than Clearcut once the cost of computing the distance matrix is included. Clearcut would also require over 50 gigabytes of memory – 20 times as much as FastTree – which makes it impractical for us to run. Furthermore, Clearcut seems to be less accurate than FastTree (Tables 1, 3, & 4).

Effectiveness and Speed of the Local Bootstrap

To test whether FastTree's local bootstrap can identify which splits are reliable, we used the protein simulations with 250 sequences, We also computed the traditional bootstrap: we used phylip's seqboot to generate resampled alignments, we ran FastTree on each resample, and we counted how often each split in the original tree was present in the resampled trees. For both methods, we used 1,000 resamples. As shown in Figure 2, both methods were effective in identifying correct splits. If we define "strongly supported" as a local bootstrap of $\geq 95\%$, then 65% of the correct splits were strongly supported. Conversely, 97% of the strongly-supported splits were correct.

To quantify how effective the measures were in distinguishing correct splits, we used the area under the receiver operating characteristic curve (AOC, DeLong and Clarke-Pearson (1998)). The AOC is the probability that a true split will have a higher support value than an incorrect split, so a perfect predictor has AOC=1 and a random predictor has AOC=1/2. The traditional bootstrap had an AOC of 0.933, versus 0.875 for the local bootstrap. Overall, the local bootstrap is not quite as sensitive as the traditional bootstrap, but it is a strong

indicator of which splits are correct.

The local bootstrap was far faster than the traditional bootstrap and required far less memory. The traditional bootstrap takes 100 times longer than tree inference plus the time to compare the trees to each other. For the 16S rRNA alignment, performing the tree comparisons with phylip's consense would take months and would require over 90 GB of memory (Table 6). In contrast, FastTree computed the local bootstrap in an hour and 2.4 GB.

Discussion

Large Alignments

We have relied on profile-based multiple sequence alignment as the most practical method for large families. However, profile-based alignment is believed to be less accurate than progressive alignment. Thus, whenever possible, biological inferences from these large trees should be confirmed with smaller, higher-quality alignments. This also allows the use of slower but more accurate tree-building methods and tests. For example, MicrobesOnline.org includes interactive tools for browsing large trees, for selecting relevant sequences, and for building progressive alignments and maximum-likelihood trees with those sequences.

Scaling to a Million Sequences

FastTree computes trees for the largest existing alignments, with on the order of 100,000 sequences, in under a day. However, given the rapid rate of DNA sequencing, we expect that alignments with 1,000,000 sequences will soon exist. For such large alignments, the major memory requirement will be the top-hits lists, which take $O(N\sqrt{N})$ space. For 1 million sequences, this will be about 20 gigabytes. In contrast, the distance matrix for a million sequences would

take 2 terabytes of memory. FastTree's running time should scale by between $O(N \log N \sqrt{N})$ and $O(N^2)$, so inferring a tree for a million rRNA sequences should take 2-4 weeks. Tuning the top-hits heuristic might reduce this time.

Conclusions

FastTree makes it practical to infer accurate phylogenies, including support values, for families with tens or hundreds of thousands of sequences. These phylogenies should be useful for reconstructing the tree of life and for predicting functions for the millions of uncharacterized proteins that are being identified by large-scale DNA sequencing. FastTree executables and source code are available at http://www.microbesonline.org/fasttree; FastTree trees for every prokaryotic gene family are available in the MicrobesOnline tree-browser (http://www.microbesonline.org/); and a FastTree tree for all sequenced full-length 16S ribosomal RNAs is available from the FastTree web site and will be included in the next release of greengenes (http://greengenes.lbl.gov).

Acknowledgments

This work was supported by a grant from the US Department of Energy Genomics:GTL program (DE-AC02-05CH11231).

References

Alm, E. J., K. H. Huang, M. N. Price, R. P. Koche, K. Keller, I. L. Dubchak, and A. P. Arkin. 2005. The MicrobesOnline Web site for comparative genomics. Genome Res. **15**:1015–22.

Bininda-Emonds, O. R., S. G. Brady, J. Kim, and M. J. Sanderson. 2001. Scaling of accuracy in extremely large phylogenetic trees. Pac Symp Biocomput. :547–58.

DeLong, E. R., and D. L. Clarke-Pearson. 1998. Comparing the areas under two or more correlated receiver operating characteristic curves: a nonparametric approach. Biometrics **44**:837–45.

DeSantis, T. Z., P. Hugenholtz, N. Larsen, M. Rojas, E. L. Brodie, K. Keller, T. Huber, D. Dalevi, P. Hu, and G. L. Andersen. 2006. Greengenes, a chimera-checked 16S rRNA gene database and workbench compatible with ARB. Appl Environ Microbiol **72**:5069–5072.

Desper, R., and O. Gascuel. 2002. Fast and accurate phylogeny reconstruction algorithms based on the minimum-evolution principle. Journal of Computational Biology **9**:687–705.

Efron, B., E. Halloran, and S. Holmes. 1996. Bootstrap confidence levels for phylogenetic trees. Proc Natl Acad Sci USA **93**:13429–34.

Eisen, J. A. 1998. Phylogenomics: Improving functional predictions for uncharacterized genes by evolutionary analysis. Genome Res. **8**:163–167.

Elias, I., and J. Lagergren. 2005. Fast neighbor joining. In: Proc. of the 32nd International Colloquium on Automata, Languages and Programming (ICALP'05), volume 3580 of *Lecture Notes in Computer Science*. Springer-Verlag, 1263–1274.

Engelhardt, B. E., M. I. Jordan, K. E. Muratore, and S. E. Brenner. 2005. Protein molecular function prediction by bayesian phylogenomics. PLoS Comput. Biol. **1**:e45.

Evans, J., L. Sheneman, and J. Foster. 2006. Relaxed neighbor joining: a fast distance-based phylogenetic tree construction method. J Mol Evol. **62**:785–92.

Felsenstein, J. 1985. Confidence limits on phylogenies: an approach using the bootstrap. Evolution **39**:783–791.

Finn, R. D., J. Mistry, B. Schuster-Böckler, et al. 2006. Pfam: clans, web tools and services. Nucleic Acids Res. **34**:D247–51.

Gascuel, O. 1997. BIONJ: an improved version of the NJ algorithm based on a simple model of sequence data. Mol Biol Evol. **14**:685–695.

Gascuel, O., and M. Steel. 2006. Neighbor-joining revealed. Mol Biol Evol. **23**:1997–2000.

Guindon, S., and O. Gascuel. 2003. A simple, fast, and accurate algorithm to estimate large phylogenies by maximum likelihood. Syst Biol. **52**:696–704.

Henikoff, S., and J. G. Henikoff. 1992. Amino acid substitution matrices from protein blocks. Proc. Natl. Acad. Sci. USA **89**.

Hollich, V., L. Milchert, L. Arvestad, and E. L. Sonnhammer. 2005. Assessment of protein distance measures and tree-building methods for phylogenetic tree reconstruction. Mol Biol Evol. **22**:2257–64.

Howe, K., A. Bateman, and R. Durbin. 2002. QuickTree: building huge neighbour-joining trees of protein sequences. Bioinformatics **18**:1546–7.

Kishino, H., T. Miyata, and M. Hasegawa. 1990. Maximum likelihood inference of protein phylogeny and the origin of chloroplasts. J. Mol. Evol. **31**:151–160.

Li, H., A. Coghlan, J. Ruan, et al. 2006. TreeFam: a curated database of phylogenetic trees of animal gene families. Nucleic Acids Res. **34**:D572–D580.

Lichtarge, O., H. Yao, D. M. Kristensen, S. Madabushi, and I. Mihalek. 2003. Accurate and scalable identification of functional sites by evolutionary tracing. J. Struct. Funct. Genomics **4**:159–66.

Müller, T., S. Rahmann, T. Dandekar, and M. Wolf. 2004. Accurate and robust phylogeny estimation based on profile distances: a study of the Chlorophyceae (Chlorophyta). BMC Evol Biol. **4**.

Nei, M., S. Kumar, and K. Takahashi. 1998. The optimization principle in phylogenetic analysis tends to give incorrect topologies when the number of nucleotides or amino acids used is small. Proc Natl Acad Sci USA **95**:12390–7.

Saitou, N., and M. Nei. 1987. The neighbor-joining method: a new method for reconstructing phylogenetic trees. Mol. Biol. Evol. **4**:406–425.

Schaffer, A. A., L. Aravind, T. L. Madden, S. Shavirin, J. L. Spouge, et al. 2001. Improving the accuracy of PSI-BLAST protein database searches with composition-based statistics and other refinements. Nucleic Acids Res. **29**:2994–3005.

Simonsen, M., T. Mailund, and C. N. S. Pedersen. 2008. Rapid neighbor-joining. In: Proc. of the 8th Workshop on Algorithms in Bioinformatics (WABI 2008). in press.

Sonnhammer, E. L. L., and V. Hollich. 2005. Scoredist: A simple and robust protein sequence distance estimator. BMC Bioinformatics **6**:108.

Stamatakis, A. 2006. RAxML-VI-HPC: maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models. Bioinformatics **22**:2688–2690.

Stoye, J., D. Evers, and F. Meyer. 1998. Rose: generating sequence families. Bioinformatics **14**:157–163.

Studier, J. A., and K. J. Keppler. 1988. A note on the neighbor-joining algorithm of Saitou and Nei. Mol Biol Evol. **5**:729–31.

Tatusov, R. L., D. A. Natale, I. V. Garkavtsev, T. A. Tatusova, U. T. Shankavaram, B. S. Rao, B. Kiryutin, M. Y. Galperin, N. D. Fedorova, and E. V. Koonin. 2001. The COG database: new developments in phylogenetic classification of proteins from complete genomes. Nucleic Acids Res. **29**:22–8.

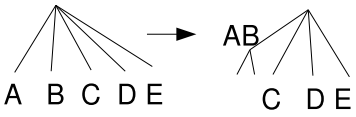
von Mering, C., P. Hugenholtz, J. Raes, S. G. Tringe, T. Doerks, L. J. Jensen, N. Ward, and P. Bork. 2007. Quantitative phylogenetic assessment of microbial communities in diverse environments. science **315**:1126–1130.

Zaslavsky, L., and T. A. Tatusova. 2008. Accelerating the neighbor-joining algorithm using the adaptive bucket data structure. Lect Notes Comput Sci. :122–133.

Figures

Figure 1 - Overview of FastTree.

Neighbor Joining with Profiles




```
Profile(A)    ACGTACGTACGT
Profile(B)    A-CGACGTAC-T

Profile(AB)   A[ccg/-gt]ACGTAC[g/-]T
```

O(NLa) space instead of $O(N^2)$ space

Neighbor-joining Criterion:

find the join that minimizes d(A,B) – r(A) – r(B)

Distances to joined nodes:

Neighbor joining: d(AB,C) = (d(A,C)+d(B,C))/2 – d(A,B)/2

FastTree: = **Δ**(AB,C) – u(C) – u(AB) ← "up-distance" u

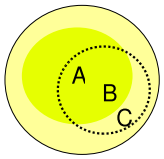
"profile-distance" **Δ**

Average out-distances:

Neighbor joining: r(A) = Σ d(A,X)/(n-2)

FastTree: $= \frac{n^{*}\mathbf{\Delta}(A, \sum X/n) - \mathbf{\Delta}(A,A) - \sum (u(A)+u(X))}{n-2}$

Top-hits Heuristic



If B is close to A, then the best join for B is also close to A:

TopHits(B) ⊂ TopHits(A) with a larger radius

When we do a join:

TopHits(AB) ⊂ TopHits(A) ∪ TopHits(B)

Nearest-neighbor interchanges (NNIs) & Local bootstrap

Consider three local topologies:

Minimum evolution criterion:

prefer AB|CD if $d_{AB} + d_{CD} < \min(d_{AC} + d_{BD}, d_{AD} + d_{BC})$

where d_{AB} = LogCorrection(**Δ**(A,B))

Steps in FastTree

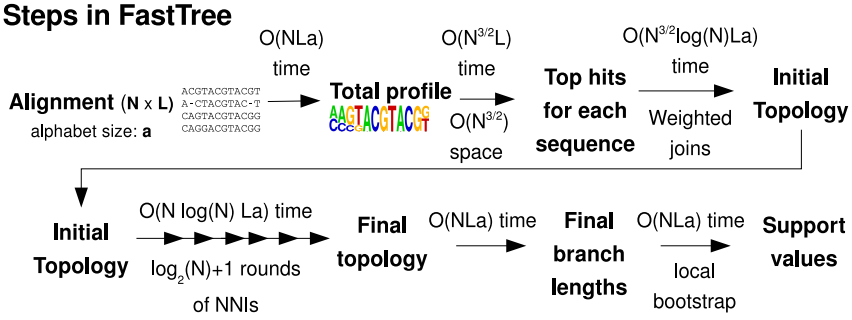
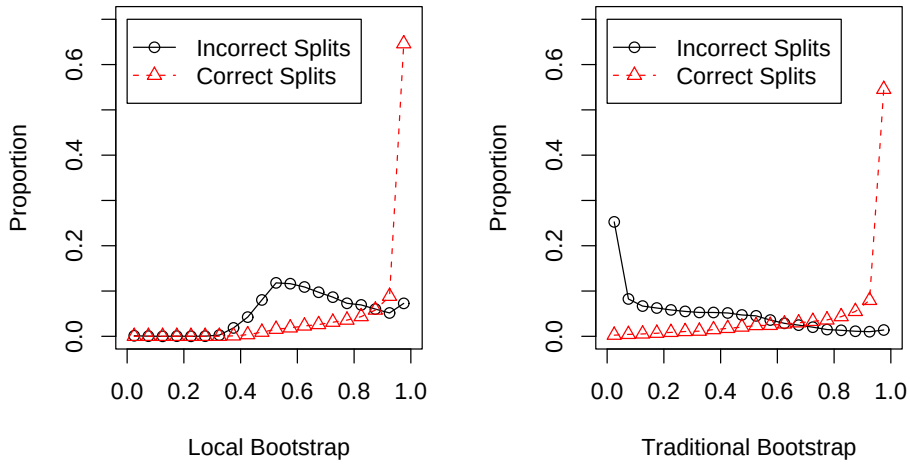


Figure 2 - Distribution of support values for simulated alignments of 250 protein sequences with gaps.

We compare the distribution of FastTree's local bootstrap and the traditional (global) bootstrap for correctly- and incorrectly-inferred splits. The right-most bin contains the strongly-supported splits (0.95-1.0).


Incorrect Splits
Correct Splits
Proportion
0.0
0.2
0.4
0.6
0.0
0.2
0.4
0.6
0.8
1.0
Local Bootstrap
Incorrect Splits
Correct Splits
Proportion
0.0
0.2
0.4
0.6
0.0
0.2
0.4
0.6
0.8
1.0
Traditional Bootstrap

Tables

Table 1 - Topological accuracy of tree-building methods on simulated protein alignments with gaps.

		Topological Accuracy				
Method	**Distances**	**N=10**	**N=50**	**N=250**	**N=1,250**	**N=5,000**
PhyML	JTT	0.744^{+}	0.771^{+}	0.817^{+}	0.801^{+}	–
FastTree	log-corrected	0.724^{0}	0.763^{0}	0.797^{0}	0.778^{0}	0.763^{0}
FastME	log-corrected	0.716^{-}	0.754^{-}	0.796^{0}	0.777^{0}	0.753^{-}
BIONJ	log-corrected	0.725^{0}	0.754^{-}	0.766^{-}	0.730^{-}	0.723^{-}
BIONJ	JTT	0.701^{-}	0.758^{-}	0.777^{-}	0.737^{-}	0.731^{-}
BIONJ	JTT+Γ	0.567^{-}	0.625^{-}	0.737^{-}	0.697^{-}	–
QuickTree	log-corrected	0.716^{-}	0.746^{-}	0.760^{-}	0.726^{-}	0.716^{-}
QuickTree	%different	0.673^{-}	0.678^{-}	0.699^{-}	0.672^{-}	0.655^{-}
Clearcut	log-corrected	0.682^{-}	0.733^{-}	0.755^{-}	0.723^{-}	0.715^{-}

$^{+}$ Significantly more accurate than FastTree ($P < 0.01$, paired t test)
^{0} Not significantly different from FastTree ($P > 0.01$, paired t test)
$^{-}$ Significantly less accurate than FastTree ($P < 0.01$, paired t test)

Table 2 - The topological accuracy of variants of FastTree on simulated protein alignments with gaps.

	Topological Accuracy		
Method	**N=250**	**N=1,250**	**N=5,000**
FastTree, Default settings	0.797	0.778	0.763
FastTree + Extra NNI (20 rounds)	0.797	0.778	0.763
FastTree's Neighbor-joining (No NNI)	0.734	0.702	0.698
FastTree, Exhaustive search, No NNI	0.733	0.701	–
BIONJ, uncorrected distances	0.731	0.699	0.694
BIONJ, log-corrected distances	0.766	0.730	0.723

Table 3 - The relative log-likelihoods of topologies inferred for 310 genuine protein alignments of 500 sequences each.

Method	Distances/ Model	Average Log-Lik.	Lower Lik. than FastTree
PhyML/FastTree[a]	JTT+Γ_4[b]	440.7	0%
FastTree	log-corrected	0.0	–
FastME	log-corrected	-165.2	86%
BIONJ	JTT	-404.3	95%
BIONJ	log-corrected	-426.1	>99%
QuickTree	log-corrected	-495.3	>99%
Clearcut	log-corrected	-532.2	99%
QuickTree	%different	-667.0	100%
BIONJ	JTT+Γ	-1576.1	99%

[a] PhyML 3 with FastTree as the starting tree.

[b] Γ_4 means four categories of sites with gamma-distributed rates.

Table 4 - The relative log-likelihoods of topologies inferred for 100 genuine 16S ribosomal RNA alignments of 500 sequences each.

Method	Distances/ Model	Average Log-Lik.	Lower Lik. than FastTree
PhyML	HKY85+Γ_4	510.4	0%
PhyML	HKY85	358.4	5%
FastME	F84+Γ	59.9	34%
FastTree	Jukes-Cantor	0.0	–
FastME	Kimura	-7.4	53%
FastME	Jukes-Cantor	-71.7	70%
BIONJ	F84+Γ	-749.1	100%
BIONJ	Kimura	-781.0	100%
BIONJ	Jukes-Cantor	-843.9	100%
QuickTree	F84+Γ	-878.8	100%
Clearcut	F84+Γ	-905.1	100%
QuickTree	Jukes-Cantor	-941.1	100%
Clearcut	Jukes-Cantor	-982.3	100%

Table 5 - Genuine alignments for performance testing.

Alignment	COG2814	PF00005	16S rRNA
Type	protein	protein	nucleotide
#Sequences	10,610	52,927	167,547
#Distinct	8,362	39,092	158,022
#Columns	394	214	1,287
%Gaps	10.8%	15.2%	4.3%

Table 6 - CPU time and memory usage for computing distances, trees, and support values.

		COG2814		PF00005		16S rRNA	
Program	**Support**	**hours**	**GB**	**hours**	**GB**	**hours**	**GB**
FastTree 1.0	none	0.06	0.16	0.52	0.3	16.3	2.4
FastTree 1.0	local 1,000	0.08	0.16	0.56	0.3	17.3	2.4
Log-corrected Distances[a]		0.05	0.13	0.71	2.8	33.1	49.9
Max-lik. Distances[b]		138	0.72	$\approx 3,000$	–	$\approx 5,000$	–
Clearcut 1.0.8 [c]	none	0.06	0.22	1.44	5.2	≈ 28.6	≈ 52
RapidNJ 1.0.0 [c]	none	0.05	2.2	≈ 0.9	≈ 55	≈ 22.1	≈ 549
FastME 1.1 [c]	none	0.51	4.2	≈ 12.5	≈ 105	≈ 138	$\approx 1,000$
QuickTree 1.1 [c]	none	0.24	0.16	22.7	2.9	$\approx 1,500$	≈ 47
QuickTree 1.1 [d]	boot 100	63.5	0.71	$\approx 10^4$	≈ 15.5	$\approx 10^5$	≈ 254
BIONJ[c]	none	32.9	0.44	≈ 820	≈ 10.9	$\approx 10^5$	≈ 110
PhyML 3[e]	aLRT	>1,000	9.5	–	–	–	–
RAxML VI 1.0[f]	none	>1,000	0.70	–	–	–	–
consense[g]	boot 100	1.09	0.36	118	9.4	$\approx 3,700$	≈ 94

[a] The time to compute the distances between all N^2 pairs of sequences in the alignment, as implemented by the authors, and the space required to store the $N(N-1)/2$ distinct entries of the distance matrix. For nucleotide sequences, these are the same as Jukes-Cantor distances.

[b] For protein sequences, we used phylip's protdist and default options (JTT model, no variation of rates across sites). For nucleotide sequences, we used phylip's dnadist with the F84 model and gamma-distributed rates.

[c] These timings include half of the time to compute N^2 log-corrected distances because the method requires a distance matrix but each pair of sequences only needs to be considered once.

[d] Using QuickTree's built-in implementation of %different distances and of global bootstrap.

[e] For best performance, we used no variation of rates across sites.

[f] For best performance, we used no variation of rates across sites and the fast hill-climbing option (-f d). For an initial topology, we used the BIONJ tree.

[g] This does not include the time to compute the resampled trees.