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

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

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

morgannprice@yahoo.com

## Abstract

As DNA sequencing accelerates, gene families are growing rapidly, but standard methods for inferring phylogenies become computationally demanding for alignments of thousands of 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 refines the topology with nearest-neighbor interchanges according to the minimum-evolution criterion. Compared to using a distance matrix, FastTree reduces the memory required from $O(N^2)$ to $O(NLa + N\sqrt{N})$ and reduces the computation time from $O(N^2L)$ to $O(N\sqrt{N}\log(N)La)$, where $N$ is the number of sequences, $L$ is the width of the alignment, and $a$ is the size of the alphabet. To estimate the tree's reliability, FastTree uses local bootstrapping, which gives another 100-fold speedup over distance matrix approaches. FastTree constructed trees, including support values, for biological alignments with 39,092 or 158,022 distinct sequences in less time than it takes to compute a distance matrix and in a fraction of the space. Traditional neighbor joining with 100 bootstraps would be 10,000 times slower and would require 50 gigabytes of memory. In simulations, FastTree is slightly more accurate than other minimum-evolution methods such as neighbor joining, BIONJ, or FastME, and on genuine alignments, FastTree produces topologies with 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. They then search for the topology that minimizes the total length of the tree, using a local estimate of the length of each branch (Gascuel and Steel, 2006). 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. The standard neighbor-joining algorithm requires a further $O(N^3)$ time to build the tree, but there are implementations that take $O(N^2)$ or $O(N^2 \log N)$ time, either by using heuristics to speed the search (Elias and Lagergren, 2005; Evans et al., 2006) or by using additional $O(N^2)$ memory (Simonsen et al., 2008; Zaslavsky and Tatusova, 2008). There are also other minimum-evolution methods that take only $O(N^2)$ time, such as FastME (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. Many families already contain 100,000-200,000 members. For example, the MicrobesOnline database, which provides phylogenies for all protein families from prokaryotic genomes, already contains 100 protein families that contain over 100,000 distinct sequences (Alm et al. (2005); http://www.microbesonline.org/). Similarly, an alignment of full-length 16S ribosomal RNAs contains over 160,000 distinct sequences (DeSantis et al. (2006); http://greengenes.lbl.gov). 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 to 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 neighbor joining with bootstrap takes

days of CPU time for just 10,000 sequences.

Although building phylogenetic trees for large gene families is challenging, we argue that it is important to do so and not just to build trees for small sets of selected homologs. First, building smaller trees means not considering all of the data, which reduces the accuracy of the resulting tree (Zwickl and Hillis, 2002). Second, not considering all taxa can change the biological interpretation of the result. This problem is particularly severe in prokaryotes: because horizontal gene transfer is pervasive, it is difficult to know which homologs are relevant without building a tree. Third, phylogenetic trees have many applications, most obviously in taxonomy, but also in predicting gene function, identifying functional residues, and classifying environmental DNA sequences (Eisen, 1998; Engelhardt et al., 2005; Lichtarge et al., 2003; von Mering et al., 2007). For these applications, it is preferable to analyze all of the sequences. 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.microbesonline.org/). One alternative to building large gene trees that we have explored is to identify orthology groups and to build smaller trees for these subfamilies, without building larger trees (Dehal and Boore, 2006). However, we doubt whether this approach will scale to thousands of genomes because the time to identify orthology groups is quadratic in the number of genomes.

## 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 stores 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. FastTree uses these profiles to compute the distances between internal nodes. For example, in traditional neighbor joining, when one joins two nodes A and B to make a new internal node AB, one computes and stores the distances of the new node to other nodes $d(AB, C) = (d(A, C) + d(B, C) - d(A, B))/2$. In BIONJ, a popular refinement of neighbor-joining, the joins are weighted, so that $d(AB, C) = \lambda(d(A, B) - d(A, AB)) + (1-\lambda)(d(A, B) - d(B, AB))$, where $\lambda$ is the weight (Gascuel, 1997). FastTree instead stores a profile $P(AB) = \lambda P(A) + (1-\lambda)P(B)$, as well as an "up-distance" to account for subtracting the $d(A, AB)$ and $d(B, AB)$ terms. 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). Given the profiles of AB and C and the up-distances, FastTree can compute the distance $d(AB, C)$ as needed. 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. Nearest-neighbor interchanges have previously been used in a minimum-evolution framework in FastME, which 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 that 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 examining the profiles of A, B, C, and D. The local bootstrap has been used for maximum-likelihood trees (Kishino et al., 1990) but cannot be used with distance matrix methods. Computing the local bootstrap takes $O(bNLa)$ time, where $b$ is the number of bootstrap samples. Even with 1,000 samples, 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.

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. Furthermore, we show that the local bootstrap is a good indicator of whether each split in the inferred topology is correct. 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. Finally, we show that for large families, FastTree requires less CPU time and far less memory than computing and storing a distance matrix. Thus, 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

### FastTree's Distance Measure

Whereas traditional distance matrix methods take the distance matrix as input, FastTree computes the distances itself. For nucleotide sequences, FastTree uses Jukes-Cantor distance $d = -\frac{3}{4}\log(1-\frac{4}{3}p)$, where $p$ is the proportion of non-gap positions that differ.

For protein sequences, FastTree estimates distances by using the BLOSUM45 amino acid similarity matrix (Henikoff and Henikoff, 1992) and a log-correction similar to 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. Before correcting for multiple substitutions, the distance between two sequences is the average dissimilarity among non-gap positions. To correct for multiple substitutions, FastTree uses the formula $d = -1.3\log(1-d_u)$, where $d_u$ is the uncorrected distance. As in scoredist, FastTree truncates distances to values no greater than 3.0, and for sequences that do not overlap because of gaps, FastTree uses this maximum distance. In the Results, we will show that FastTree's log-corrected distances lead to the same quality of trees as maximum-likelihood distances from protdist (http://evolution.genetics.washington.edu/phylip.htm), but protdist is 1,000 times slower.

In preliminary testing, correcting for multiple substitutions decreased the accuracy of the initial neighbor-joining phase of FastTree (data not shown). This might be because of the average-of-logs approximation described in the next section. Neighbor-joining requires averaging over distances that vary widely, so that the approximation performs poorly. In contrast, during the nearest-neighbor interchanges, FastTree only averages distances between adjacent nodes in the tree. In any case, FastTree corrects the distances during nearest-neighbor interchanges and during local bootstrap but does not correct for multiple substitutions during the neighbor-joining phase.

### Computing Average Distances with Profiles

The intuition behind using profiles is that the average of the distances between the sequences in two subtrees A and B is the same as the distance between profile(A) and profile(B), because profile(A) can be thought of as the average of the sequences in A. We define the uncorrected distance between two profiles at position $l$ to be the weighted average over all pairs of characters of the dissimilarity of those characters, weighted by the product of the frequencies of the characters in the two profiles, or $P_l(A,B) = \sum_\alpha \sum_\beta f_{Al}(\alpha) f_{Bl}(\beta) D(\alpha,\beta)$, where $f$ represent frequencies and $D$ the dissimilarity matrix. The uncorrected distance between the two profiles is then $P(A,B) = \sum_l w_{Al} w_{Bl} P_l(A,B) / (\sum_l w_{Al} w_{Bl})$, where $w_{Al}$ is the proportion of non-gaps in the profile of A at position $l$. Naively,

the distance between two profiles takes $O(La^2)$ time to compute, but this can be reduced to $O(La)$ 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. 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 neighbor joining. In the Results, we show that these approximations are effective.

First, consider the issue of correcting for multiple substitutions with the formula $d = -1.3 \cdot \log(1 - d_u)$. Neighbor-joining uses the average distances $(d(A,B) + d(A,C))/2$, while FastTree uses the corrected profile distance $-1.3 \cdot \log(1 - P(A,BC)) = -1.3 \cdot \log(1 - (d_u(A,B) + d_u(A,C))/2)$. This is a good approximation if the distances are under 0.4 (short branches) or if the distances are near each other ("clock-like" evolution). For example, if $d_u(A,B) = 0.2$ and $d_u(A,C) = 0.3$ then the two formulas give average distances of 0.377 and 0.374, respectively. FastTree actually uses weighted joins, so that the long branch will contribute less to the average, which also makes the approximation more accurate. The same argument applies to Jukes-Cantor distances.

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. This seems intuitively reasonable, and it is not clear if this is better or worse than averaging the distances as in traditional neighbor joining. In both approaches, the gaps are treated as missing data, but the weighting of the data is different. For example, consider this alignment of three sequences and two positions with the alphabet {0,1}:

```
1- (A)
00 (B)
11 (C)
```

so that $d(A,C) = 0/1$ and $d(B,C) = 2/2$. By the profile approach, the two positions have weights 1 and 1/2, and $P(AB,C) = (1 \cdot 0.5 + 0.5 \cdot 1)/(1 + 0.5) = (0.5 + 0.5)/1.5 = 2/3 \neq (d(A,C) + d(B,C))/2 = 1/2$. As this example illustrates, traditional neighbor joining places more weight on the ungapped column (in this case, the first column).

### Neighbor Joining Using Profiles

Neighbor joining operates on distances between internal nodes rather than on average distances between the members of subtrees. For example, if we do a weighted join between two nodes $A$ and $B$, as in BIONJ, then $d(AB,C) = \lambda(d(A,B) - d(A,AB)) + (1 - \lambda)(d(A,B) - d(B,AB))$ (Gascuel, 1997). To compute these distances between nodes, FastTree sets the profile of $AB$ to $P(AB) = \lambda P(A) + (1 - \lambda)P(B)$, and for any pair of nodes $i$ and $j$, we write $d(i,j) = P(i,j) - u(i) - u(j)$, where $P(i,j)$ is the profile distance and $u(i)$ is the "up-distance." We set $u(i) = 0$ for leaves, and for joined nodes we set

$u(ij) = \lambda(u(i) + d(i,ij)) + (1-\lambda)(u(j) + d(j,ij))$. It is easy to show that this profile-based computation gives the exact same value of $d(i,j)$ as BIONJ after any number of joins, as long as distances are not corrected for multiple substitutions and the sequences contain no gaps (Supplementary Note 1).

Another complication is that neighbor joining selects the best join by minimizing the criterion $d(i,j) - r(i) - r(j)$, where $i$, $j$, $k$ are indices of active nodes that have not yet been joined, $d(i,j)$ is the distance between nodes $i$ and $j$, $n$ is the number of active nodes and $r(i) \equiv \sum_{k \neq i} d(i,k)/(n-2)$. $r(i)$ can be thought of as the average "out-distance" of $i$ to other active nodes (although the denominator is $n-2$, not $n-1$). The out-distance for a node can be computed in $O(La)$ time if we store a "total profile" which is the average over all active nodes. A derivation for the formula $(n-2)r(i) = nP(i,T) - P(i,i) - (n-2)u(i) - \sum_j u(j)$, where $T$ is the total profile, is given in Supplementary Note 1. ($P(i,i)$ equals the average distance between children of $i$, and is usually above zero unless $i$ is a leaf.) We store the total up-distance $\sum_j u(j)$ and update it after every join, so this formula takes only $O(La)$ time to evaluate. In the Supplementary Note, we also present a more complicated formula that gives better results in the presence of gaps.

At the start of neighbor joining, computing the total profile takes $O(NLa)$ time. After each join, FastTree updates the total profile in $O(La)$ time by subtracting the contribution of the children and adding the contribution of the new node. FastTree also recomputes the total profile from scratch every 200 joins to prevent roundoff errors from accumulating. This adds a total of $O(N^2La)$ time, but the constant factor is very small.

To compute the weights for each join, we also need to compute the variance of the distance between two joined nodes. Formulas and derivations to compute these variances and to compute the weight for each join by using profiles are given in Supplementary Note 1.

### Heuristics for Selecting the Join

Traditional neighbor joining considers $O(n^2)$ possible joins to do at each step, where $n$ is the number of active nodes, and then selects the best join. Thus, selecting all of the joins takes $O(N^3)$ time. Here we explain how FastTree reduces the number of joins considered at each step to less than $O(n)$. FastTree uses three heuristics. First, it remembers the best known join for each node, as in FastNJ (Elias and Lagergren, 2005). Then, before accepting the best known join, it does a local hill-climbing search to find a better join, as in relaxed neighbor joining (Evans et al., 2006). Because the out-distances change after every join, the best join for a node can change as well. Finally, FastTree uses a "top-hits" heuristic which we explain below. Together, these three heuristics reduce the total time for neighbor joining with profiles from $O(N^3La)$ to $O(N\sqrt{N}\log(N)La)$. In the Results, we show that these heuristics do not reduce the accuracy of the neighbor-joining phase of FastTree.

For each node, FastTree records a top-hits list of nodes that are the closest neighbors of that node, according to the neighbor-joining criterion. We use

lists of a fixed size $m$, and by default, $m = \sqrt{N}$. Before we begin joining, we can approximate these lists for all $N$ sequences by assuming that if A and B have similar sequences or profiles, then the top-hits lists of A and B will largely overlap. More precisely, we compute the $2m$ top hits of A, where the factor of two is an arbitrary safety factor. Then, for each node B within the top $m$ hits of A that does not already have a top-hits list, we estimate the top-hits of B by comparing B to the top $2m$ hits of A. 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.

During joining, we can compute the top-hits list for a new joined node from the top-hits lists of its children in $O(mLa)$ time by comparing the new node to all of the nodes in the two top-hits lists. However, when we do a join, the top-hit lists will gradually become shorter. For example, if we join nodes A and B with identical top-hits lists, then the top-hit list of AB will be the same list but without A or B, or one shorter. Similarly, if A is in the top-hit list of B, and then we join A to C, then we replace A and C with AC when we next examine the top-hit list of B, but the list will now be shorter. When the list becomes too short (by default $< 0.8 \cdot m$ in length), or if we have done more than $\log_2 m$ joins since we updated the top-hit list of a node, then we "refresh" the top hit list. This involves comparing the newly joined node AB to all other nodes and also updating the top-hits lists of the top $2m$ hits of AB by comparing those neighbors to all of the top $m$ hits of AB. 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 these refreshes take a total of $O(N\sqrt{N}La)$ time.

Given these top-hits lists, we can find the best join in $O(m \log(N)La)$ time. First, we find the best $m$ joins among the best-hit entries for 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, we compute the current value of the neighbor-joining criterion, which takes $O(mLa)$ time, and we select the best one. Then, we do a local hill-climbing search using the best-hit lists: given a join AB, we consider all joins AC or BD, where C is in top-hits(A) or D is in top-hits(B). In theory, this takes $O(\log n)$ iterations (Evans et al., 2006), $O(m \log(n)La)$ time per join, or $O(N\sqrt{N} \log NLa)$ time overall. In practice, because we already start with a good join, hill-climbing rarely changes the join, even for alignments with 40,000 sequences (data not shown).

### Restrictions on the Top-Hits Heuristic

To ensure accuracy, FastTree restricts the top-hits heuristic and only estimates the top $m$ hits of B from the top $2m$ hits of A if A and B are similar enough. By default, FastTree requires that $d_u(A, B) \leq 0.75 \cdot d_u(A, H_{2m})$, where $H_{2m}$ is A's $2m$-th best hit, and the factor of 0.75 represents a compromise between speed and accuracy. (Note that we are using uncorrected distances here, not the neighbor-joining criterion.) If we used a factor of 0.5, then the top-hits list

would be guaranteed to include the top hit because of the triangle inequality $d_u(A,B) \leq d_u(A,C)+d_u(C,B)$. On the other hand, for a perfectly balanced tree with clock-like evolution, we expect the distance of hit $m$ to be proportionate to $\log_2 m$ and the distance of hit $2m$ to proportionate to $1+\log_2 m$, so with a factor of $\log_2 m/(1+\log_2 m)$ we would expect $O(m)$ close neighbors of A.

Before it estimates the top-hits of B from the top-hits of A, FastTree also requires that A and B have similar patterns of gaps. This is to avoid cases where the sequences overlap in only a few positions, so that they might be identical or nearly so (for the positions considered) even though they have very different top-hits lists. Specifically, we require that the total number of non-gap positions in the comparison between A and B must be at least $1-d_u/2$ times the number of non-gap positions in B or at least $1-2d_u/3$ times the average shared positions between A and its top $2m$ hits, where $d_u$ is the maximum distance allowed between A and B.

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. In the Results, we show that FastTree takes less time that computing the distance matrix, which suggests that this does take less than $O(N^2L)$ time.

**Nearest-neighbor Interchanges**

After FastTree constructs an initial tree with neighbor joinining, it uses nearest-neighbor interchanges to improve the tree topology. FastTree does the NNIs in rounds. During each round, it 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. Finally, after the NNIs are complete, FastTree computes branch lengths for the new topology.

According to the minimum evolution criterion, the topology ((A,B),(C,D)) preferable 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 $d(A,B)+d(C,D) < d(A,D)+d(B,C)$. For larger topologies, we need to compute a profile for additional subtrees in order to do this computation. For example, consider the topology ((A,(B,C)),D,E). After neighbor-joining, we have profiles for the internal nodes BC and ABC as well as for the leaves (these profiles are just sequences). To test the split ABC versus DE, we need profiles for A, BC, D, and E, which we already have. To test the split BC versus ADE, we need profiles for B, C, A, and DE. We compute the profile for DE by doing a weighted join of D and E (see below). We store these additional profiles along the path to the root and reuse them when possible. (FastTree computes an unrooted tree but stores it as a rooted tree.) FastTree uses a "post-order" traversal of the tree, in which children are always visited before their parents, within each round. This ensures that we compute at most $O(N)$ additional profiles, that we store at most $O(d)$ profiles, where $d < N$ is the depth of the tree, and that none of the additional profiles is needed after we visit a node in the profile and potentially change its topology (which could lead

to inconsistencies). Thus, a round of NNIs takes O($NLa$) time and O($NLa$) additional space.

To compute the weighted join of A and B in a quartet ABCD, we use the BIONJ weighting $\lambda = 0.5 + (d(B,C) + d(B,D) - d(A,C) - d(A,D))/(4 \cdot d(A,B))$ and $P(AB) = \lambda A + (1-\lambda)B$.

By default, FastTree does $\log_2(N) + 1$ rounds of NNIs. We use a fixed number of rounds, instead of iterating until no more NNIs occur, to ensure convergence in a reasonable amount of time. The motivation for using roughly $\log_2(N)$ rounds is that, on a balanced topology, a misplaced leaf can migrate all of the way across the tree.

Finally, once the topology is complete, FastTree recomputes lengths for all branches using the formula $d(AB,CD) = (d(A,C) + d(A,D) + d(B,C) + d(B,D))/4 - (d(A,B) + d(C,D))/2$ for internal branches and $d(A,BC) = (d(A,B) + d(A,C) - d(B,C)/2$ for branches leading to leaves.

### Local Bootstrap

To estimate the support for each split, FastTree examines the same four profiles and uses the same minimum evolution criterion as for the nearest-neighbor interchanges. FastTree resamples the columns using Knuth's 2002 random number generator (http://www-cs-faculty.stanford.edu/~knuth/programs/rng.c).

### Unique sequences

Large alignments often contain many sequences that are exactly identical to each other (Howe et al., 2002). FastTree uses hashing to quickly identify redundant sequences, 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 tested FastTree on both genuine alignments and simulated 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 according to the greengenes mask (DeSantis et al. (2006); http://greengenes.lbl.gov)

To simulate alignments with realistic phylogenies, we used 310 full-length protein families from the COG database with at least 1,000 distinct sequences

in MicrobesOnline. Given an actual family and alignment, we removed duplicate sequences, we selected the desired number of family members at random, and we removed alignment positions that were over 25% gaps. For alignments of up to 1,250 sequences, we inferred a maximum-likelihood phylogeny using PhyML 3.0 with the JTT model and no rate variation across sites (Guindon and Gascuel, 2003). Given the topology, we inferred the evolutionary rate of each site using proml from the phylip package, gamma-distributed rates (8 categories), and a coefficient of variation of 1 (http://evolution.genetics.washington.edu/phylip.htm). The inferred rate categories were biased downwards (the average rate was less than one), so we normalized the rates so that their average was 1. We then used the branch lengths (from proml) and the evolutionary rate of each site to simulate an ungapped alignment with Rose (Stoye et al., 1998). Finally, we re-introduced the gaps that were in the genuine alignment so that the simulated alignment had the same pattern of gaps. Thus, both the topology and the placement of gaps should be realistic.

For simulations of 5,000 sequences, the above approach was not computationally feasible. Instead, we inferred the topology and branch lengths using FastTree and we assigned the evolutionary rate of each site randomly among 16 categories. These categories approximated a gamma distribution with a coefficient of variation of 0.7. For comparison, the coefficient of variation of the inferred rates for the genuine alignments of 10-250 sequences was typically 0.6-0.8. Also, before selecting sequences from the genuine alignment, we made a 99% non-redundant subset of sequences with CD-HIT (Li et al., 2002). This avoided inferring a tree with many very-short branch lengths and hence simulating large numbers of non-unique sequences

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 of computational restrictions for inferring a realistic topology for the genuine alignment; and for $N = 5,000$, we simulated 7 alignments, as only 7 families contained enough non-redundant sequences.

### Software Tools

To compare the performance of FastTree to other methods, we used a variety of tools. We tested FastTree 1.0.0, a C implementation of BIONJ (http://www.lirmm.fr/~w3ifa/MAAS/BIONJ/BIONJ.c), QuickTree 1.1, Clearcut 1.0.8, RapidNJ 1.0.0 (Simonsen et al. (2008); http://birc.au.dk/Software/RapidNJ/), FastME 1.1, PhyML 3.0, RAxML VI version 1.0 (Stamatakis, 2006), and protdist and seqboot from version 3.65 of the phylip package. We also tried to run quick-join 1.0.10 (Mailund et al., 2006) on COG2814, but it crashed after using 4 GB of memory. Similarly, the only available implementation of scoredist, from the Belvu alignment viewer, required too much memory. Executables were obtained from the authors' web sites or were compiled with gcc version 3.4.6 and the -O2 optimization setting. When using protdist to estimate maximum likelihood distances, we replaced negative distances (from non-overlapping se-

quences) and distances above 3 substitutions per site with 3, as in scoredist. This improved the likelihood of the trees inferred by BIONJ from protdist distances (data not shown).

For most timings, we used a computer with 2 dual-core 2.6 GHz AMD Opteron processors and 32 GB of RAM. For two long-running jobs to infer a maximum-likelihood topology for an alignment with around 10,000 sequences, we used a different 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). All programs used a single thread of execution.

The time to compute a distance matrix for log-corrected distances was measured by running FastTree with the -makematrix option, so that it computes distances between all pairs of sequences. It compares all $N^2$ pairs of sequences rather than the $N(N-1)/2$ unique pairs in order to save memory.

To estimate the performance of the distance matrix methods on the larger families, we extrapolated from the largest feasible problem and we used the scaling behavior of each algorithm. We assumed that BIONJ scales as $O(N^3)$ time and $O(N^2)$ space, that Clearcut scales as $O(N^2 \log N)$ time and $O(N^2)$ space, that RapidNJ scales as $O(N^2)$ time and $O(N^2)$ space, and that protdist scales as $O(N^2 L)$ time. The scaling of QuickTree is cubic in time and quadratic in space, but we scaled by the number of distinct sequences, as QuickTree includes an optimization to remove redundant sequences (as does FastTree).

To compare the FastTree's local bootstrap to the traditional bootstrap, we used phylip's seqboot to generate resampled alignments and we ran FastTree on each resampled alignment. We recorded how often each split from the tree that FastTree inferred with the full data appeared in the resampled trees.

# Results

## Topological Accuracy in Simulations

We tested FastTree and other methods for inferring phylogenies on simulated protein alignments with 10, 50, 250, 1,250, and 5,000 sequences. The simulated alignments were derived from genuine alignments of large gene families (from COG, Tatusov et al. (2001)) by inferring a phylogeny, simulating sequences with that phylogeny (Stoye et al., 1998), and placing gaps in the simulated sequences in the same pattern that they appear in the genuine sequences. Although positions with gaps and other ambiguously aligned positions are usually removed before computing a phylogeny, for large alignments, this is impossible, because few or no positions will remain after trimming. Thus, we removed positions that were >25% gaps but we did not try to remove all of the gaps. For alignments of up to 1,250 sequences, we also estimated the evolutionary rate of each site in the genuine alignment, and we used these rates in the simulations (see Methods for details). The simulated alignments ranged from 64-1,009 positions (median 304). On average, the sequences in the simulated alignments were 33% identical, and 9% of positions were gaps.

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 than other distance matrix methods but was 1-2% less accurate than PhyML, a maximum likelihood method (Guindon and Gascuel, 2003). The next best method was FastME, which like FastTree uses nearest-neighbor interchanges according to the minimum evolution criterion to improve the topology (Desper and Gascuel, 2002). For 10 or 50 sequences, FastTree was about 1% more accurate than FastME, but in larger simulations, the accuracy of FastME and FastTree was not significantly different ($P > 0.01$, paired $t$ test). FastTree was significantly more accurate than BIONJ, a weighted variant of neighbor joining (Gascuel, 1997), with either log-corrected distances (as in FastTree) or maximum likelihood distances from protdist. For 10 or 50 sequences, the difference in accuracy was slight, but for 250 to 5,000 sequences, FastTree was 2-4% more accurate. BIONJ with maximum-likelihood distances that were estimated using a model with gamma-distributed rates (again using protdist) gave poor results. BIONJ with maximum-likelihood distances that were estimated with a single rate category had about the same accuracy as BIONJ with log-corrected distances, which shows that FastTree's method for estimating distances is adequate. FastTree was 1-5% more accurate than QuickTree, an implementation of traditional neighbor joining (Howe et al., 2002). Finally, FastTree was 4-6% more accurate that Clearcut, an implementation of relaxed neighbor joining (Evans et al., 2006) that is more scalable than the other distance matrix methods (see below).

It is not clear if these differences in accuracy are significant in practice. Most of the erroneous splits found by FastTree have poor support (see below). Conversely, we suspected that most splits missed by FastTree but found by PhyML have poor support. Indeed, it has previously been reported that maximum likelihood methods are more accurate than neighbor joining, but most of those additional correct splits have poor support (Nei et al., 1998). To identify strongly supported splits, we used PhyML 3.0's approximate likelihood ratio test (aLRT) with the minimum of SH-like and $\chi^2$ supports (Anisimova and Gascuel, 2006). We considered splits with support values of 90% or higher to be strongly supported. In the simulations with 10 sequences, only 10% of the splits that FastTree missed but PhyML found were strongly supported. In contrast, 64% of splits that were correctly identified by both PhyML and FastTree were strongly supported. Similarly, in simulations with 50 sequences, only 23% of the splits that FastTree missed but PhyML found were strongly supported, but 79% of splits that were correctly identified by both methods were strongly supported. Thus, most of the additional correct splits found by PhyML are poorly supported and might not be used to draw biological conclusions.

## Effectiveness of FastTree's Approximations and Heuristics

We also used the simulations to test whether the FastTree's approximations and heuristics were effective. First, FastTree uses a limited number of rounds of nearest-neighbor interchanges (e.g., 10 rounds for 250 sequences and 13 rounds

for 5,000 sequences). As shown in Table 2, increasing the number of rounds to 20 ("extra NNI") had no effect on accuracy. Second, FastTree uses heuristics to speed up the neighbor-joining phase. The accuracy of the neighbor-joining phase was virtually unchanged by these heuristics (compare "no NNI" to "exhaustive"). In any case, the NNIs should correct any errors that are due to the heuristics. Heuristic search was also over 100 times faster: for example, 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. Third, in the absence of gaps, FastTree's neighbor-joining phase should give the same results as BIONJ with uncorrected FastTree distances. In ungapped simulations, this was indeed the case (Supplementary Table 1). With gaps, FastTree uses an approximation. However, BIONJ with uncorrected distances was no more accurate than FastTree's neighbor-joining phase, which shows that FastTree's approximation is effective.

The loss of accuracy due to using uncorrected distances during the neighor-joining phase was modest. In the simulations, BIONJ with corrected distances was about 2% more accurate than BIONJ with uncorrected distances. This is consistent with a previous simulation study of realistic topologies and protein alignments (Hollich et al., 2005). Because the number of errors due to uncorrected distances is small, it is not suprising that FastTree can correct these errors by doing a few rounds of NNIs.

After doing nearest-neighbor interchanges, it made little difference if the joins were weighted, as in the default settings of FastTree, or not (see "balanced joins" in Table 2). This was surprising to us because weighting the joins did improve the accuracy of the neighbor joining methods (compare QuickTree and BIONJ with log-corrected distances in Table 1). Again, the NNIs probably make up for any deficiencies in the initial neighbor-joining tree.

Finally, we wondered how the presence of gaps in the alignments was affecting our results. For the simulated families with 10-1,250 sequences, we ran FastTree and other methods on the ungapped simulated alignments (i.e., before introducing the gaps). We found similar results as in the gapped simulations: FastTree was slightly more accurate than BIONJ for large alignments and was slightly more accurate than FastME for small aligments (Supplementary Table 1). Thus, the slightly higher accuracy of FastTree as compared to other minimum-evolution methods was not due to the presence of gaps in our simulations.

## Quality of FastTree's Support Values

To compare FastTree's local bootstrap to the traditional bootstrap, we used the simulations with 250 sequences, and we ran the traditional bootstrap with FastTree as the underlying method for inferring trees. For both kinds of bootstrap, we used 1,000 replicates. As shown in Figure 2, both methods were effective in identifying correct splits. For example, if we define strongly supported as having a local bootstrap of 95% of above, then 64% of correct splits were strongly supported, and 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 0.870 for the local bootstrap. Although the local bootstrap is not quite as effective as the global bootstrap, we argue that it is adequate, and it is orders of magnitude faster (see below).

## Quality of Trees for Genuine Protein Families

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 averiage difference in log-likelihood between that method's trees and FastTree's trees. The methods are sorted by the average difference. All of the other methods tested gave significantly worse average likelihoods than FastTree (paired $t$ test, all $P < 10^{-20}$). FastTree's average log-likelihood was 165 higher than for the next-best method, FastME, and FastTree's topology had a higher likelihood than FastME's for 262 out of the 310 trees. FastTree also outperformed BIONJ (with several different distance metrics), Clearcut, and traditional neighbor-joining as implemented in QuickTree. Furthermore, as in the simulations, FastTree's approximations and heuristics did not reduce the quality of the trees (Supplementary Table 2).

## Computational Performance

To test FastTree's speed and memory requirements, we ran 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 8,362, 39,092, and 158,022 distinct sequences, respectively. More statistics on these alignments are given in Table 4, and the CPU time and memory usage of various methods are given in Table 5. Given the running times for the smallest alignment (COG2814), most of the methods will not scale to larger alignments. So, we extrapolated the CPU times and memory usages to the larger alignments by using the theoretical complexity of each method (see Methods for details).

The maximum-likelihood methods we tested, PhyML (Guindon and Gascuel, 2003) and RAxML (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.) Indeed, PhyML 3 with a single rate category typically takes over a week to complete on genuine alignments of just 1,250 protein sequences (data not shown). Thus, current implementations of the maximum-likelihood approach to inferring topologies do not scale.

The other methods that we tested require a distance matrix as input, so we also compared the time to compute a distance matrix to the time for FastTree. As shown in Table 5, FastTree is over 1,000 times faster than computing maximum-likelihood protein distances, and for the 16S rRNA alignment, FastTree is faster than computing Jukes-Cantor distances.

If estimates of the tree's reliability is required, then even for the smallest of these alignments, FastTree is 1,000 times faster than traditional neighbor joining (QuickTree) with 100 bootstraps and the simplest possible distance measure (%different). Although we expected QuickTree with bootstrap to be 100 times slower than QuickTree without bootstrap, it is actually almost 300 times slower, and requires several times more memory as well. We believe that this is because QuickTree's algorithm for comparing the bootstrapped trees to each other requires $O(N^3)$ time and $O(N^2)$ space. Other popular tools for comparing tree topologies, such as treedist and consense from the phylip package, also appear to require $O(N^3)$ time (data not shown). In principle, tree comparison can be implemented in $O(N^2)$ time and $O(N)$ space by hashing the splits in the trees.

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. Unlike FastTree, Clearcut does not produce support values – producing those would take at least 100 times longer, or over half a year. Furthermore, Clearcut seems to be less accurate than FastTree (Tables 1 & 2). Overall, we found that FastTree scales to the largest sequence families, while the maximum-likelihood methods are far too slow, and the distance-matrix methods have prohibitive CPU and memory requirements.

# Discussion

## Scaling to a Million Sequences

As we have shown, 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 ($10^9$ entries at 20 bytes per entry), which we hope will be acceptable. 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 a million sequences would take 20-40 times longer than the large 16S rRNA alignment, or 2-4 weeks, which is a bit slow. Further improvements in performance might be achievable by tuning the top-hits heuristic to be more aggressive and relying on the nearest-neighbor interchanges to correct any errors that result.

### Large Alignments

In this work, we rely on profile-based multiple sequence alignment as the most practical method for the largest 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 higher-quality alignments. For example, our web site includes interactive tools for browsing large trees, for selecting relevant sequences, and for building progressive alignments and trees with those sequences (http://www.microbesonline.org/).

One of the limiting steps in progressive alignment is the construction of the guide tree. The top-hit heuristic might be useful for this step. For example, PartTree (Katoh and Toh, 2007) uses a divide-and-conquer algorithm and k-mer distances to compute a (lower accuracy) guide tree on unaligned sequences in $O(N \log(N) L)$ time. This tree could be used to generate an initial alignment, and FastTree could be used to generate the guide tree for another iteration of progressive alignment. A faster UPGMA variant of FastTree is available at http://www.microbesonline.org/fasttree and might be useful for this purpose, both because of its speed and because UPGMA guide trees may lead to better alignments (Edgar, 2004).

In principle this approach could lead to accurate progressive alignments in less than $O(N^2 L)$ time. However, the number of gaps in an alignment grows with the number of sequences, because of independent insertions in the subfamilies. To achieve progressive alignment in less than $O(N^2 L')$ time, where $L'$ is the length of the longest input sequence, would require masking out subfamily-specific insertion positions while analyzing the other parts of the tree.

## Conclusion

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 reduces memory requirements by storing profiles of internal nodes instead of a distance matrix and reduces CPU time by using heuristic search during the neighbor joining phase. FastTree then uses nearest-neighbor interchanges to refine the topology, which increases its accuracy slightly above that of other minimum-evolution methods. Finally, FastTree quickly estimates the reliability of internal nodes in the

tree by using the local bootstrap. FastTree executables and source code are available at http://www.microbesonline.org/fasttree; FastTree trees for every prokaryotic gene family will be available in the next release of the MicrobesOnline tree-browser (http://www.microbesonline.org/); and a FastTree tree for all sequenced full-length 16S ribosomal RNAs will be available in the next release of greengenes (http://greengenes.lbl.gov).

## Acknowledgements

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.

Anisimova, M., and O. Gascuel. 2006. Approximate likelihood-ratio test for branches: A fast, accurate, and powerful alternative. Syst Biol. **55**:539–52.

Dehal, P. S., and J. L. Boore. 2006. A phylogenomic gene cluster resource: the Phylogenetically Inferred Groups (PhIGs) database. BMC Bioinformatics **7**:201.

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.

Edgar, R. C. 2004. MUSCLE: multiple sequence alignment with high accuracy and high throughput. Nucleic Acids Res. **32**:1792–1797.

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-Bckler, 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.

Katoh, K., and H. Toh. 2007. PartTree: an algorithm to build an approximate tree from a large number of unaligned sequences. Bioinformatics **23**:372–374.

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.

Li, W., L. Jaroszweski, and A. Godzik. 2002. Tolerating some redundancy significantly speeds up clustering of large protein databases. Bioinformatics **18**:77–82.

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.

Mailund, T., G. S. Brodal, R. Fagerberg, C. N. Pedersen, and D. Phillips. 2006. Recrafting the neighbor-joining method. BMC Bioinformatics .

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.

Zwickl, D. J., and D. M. Hillis. 2002. Increased taxon sampling greatly reduces phylogenetic error. Systematic Biology **51**:588–598.

# Figures

## Figure 1 - Overview of FastTree.

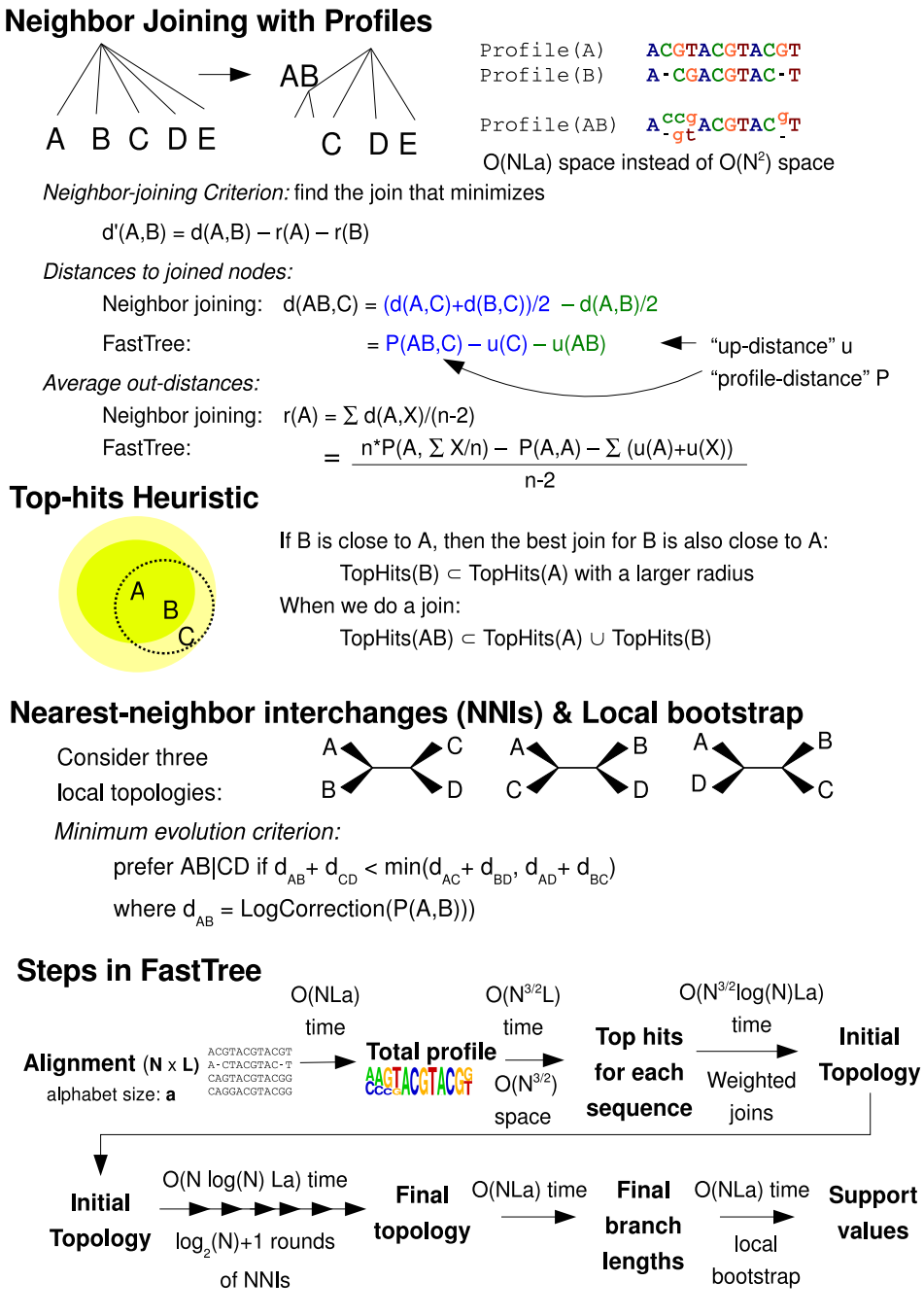
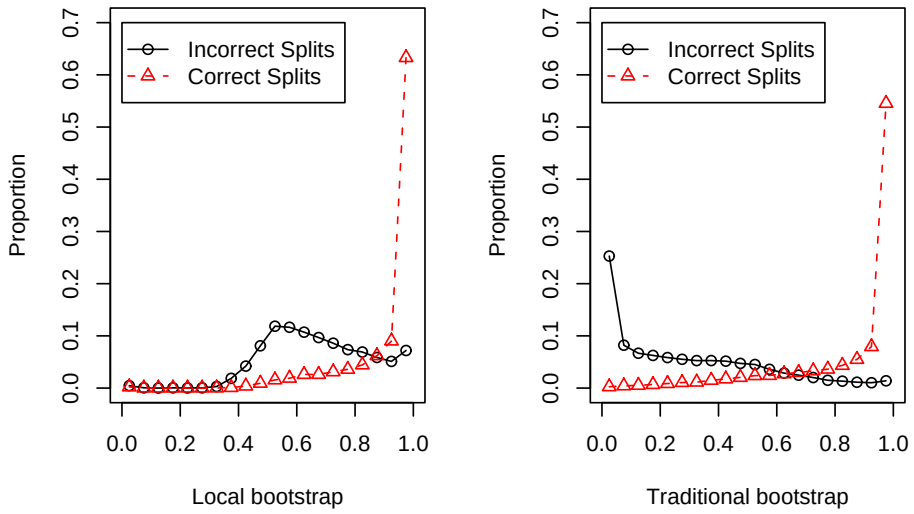


**Figure 2 - Distribution of support values for simulated alignments of 250 protein sequences with gaps.**

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).



# Tables

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

| | | Topological Accuracy | | | | |
|---|---|---|---|---|---|---|
| **Method** | **Model** | **N=10** | **N=50** | **N=250** | **N=1,250** | **N=5,000** |
| PhyML | JTT | $0.744^{+}$ | $0.771^{+}$ | $0.817^{+}$ | – | – |
| FastTree | log-corrected | $0.724^{0}$ | $0.763^{0}$ | $0.797^{0}$ | $0.778^{0}$ | $0.780^{0}$ |
| FastME | log-corrected | $0.716^{-}$ | $0.754^{-}$ | $0.796^{0}$ | $0.777^{0}$ | $0.770^{0}$ |
| BIONJ | log-corrected | $0.725^{0}$ | $0.754^{-}$ | $0.766^{-}$ | $0.730^{-}$ | $0.741^{-}$ |
| BIONJ | JTT | $0.701^{-}$ | $0.758^{-}$ | $0.777^{-}$ | $0.737^{-}$ | $0.741^{-}$ |
| BIONJ | JTT+Γ | $0.567^{-}$ | $0.625^{-}$ | $0.737^{-}$ | $0.697^{-}$ | – |
| QuickTree | log-corrected | $0.716^{-}$ | $0.746^{-}$ | $0.760^{-}$ | $0.726^{-}$ | – |
| QuickTree | %different | $0.673^{-}$ | $0.678^{-}$ | $0.699^{-}$ | $0.672^{-}$ | – |
| Clearcut | log-corrected | $0.682^{-}$ | $0.733^{-}$ | $0.755^{-}$ | $0.723^{-}$ | $0.731^{-}$ |

$^{+}$ Significantly more accurate than FastTree ($P < 0.001$, paired $t$ test)
$^{0}$ Not significantly different from FastTree ($P > 0.01$, paired $t$ test)
$^{-}$ Significantly less accurate than FastTree ($P < 0.001$, 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** |
| Default settings | 0.797 | 0.778 | 0.780 |
| Extra NNI | 0.797 | 0.778 | 0.780 |
| Balanced joins | 0.797 | 0.778 | 0.777 |
| No NNI | 0.734 | 0.702 | 0.724 |
| Exhaustive, no NNI | 0.733 | 0.701 | – |
| BIONJ, uncorrected dist. | 0.731 | 0.699 | 0.722 |

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

| **Method** | **Model** | **Average Log-Lik.** | **% Worse than FastTree** |
|---|---|---|---|
| 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% |

**Table 4 - 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 5 - CPU time and memory usage for computing distances and inferring trees for genuine alignments.**

| | | **COG2814** | | **PF00005** | | **16S rRNA** | |
|---|---|---|---|---|---|---|---|
| **Program** | **Support** | **hours** | **GB** | **hours** | **GB** | **hours** | **GB** |
| FastTree | none | 0.06 | 0.16 | 0.52 | 0.3 | 16.3 | 2.4 |
| FastTree | 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 | 49.9 | 46.5 |
| Max-lik. Distances[b] | | 138 | 0.72 | $> 10^3$ | – | – | – |
| Clearcut[c] | none | 0.06 | 0.22 | 1.44 | 5.2 | $\approx 37.0$ | $\approx 52$ |
| RapidNJ[c] | none | 0.05 | 2.2 | $\approx 0.9$ | $\approx 55$ | $\approx 30.1$ | $\approx 549$ |
| FastME[c] | none | 0.52 | 4.2 | $\approx 12.5$ | $\approx 105$ | $\approx 147$ | $\approx 10^3$ |
| QuickTree[c] | none | 0.24 | 0.16 | 22.7 | 2.9 | $\approx 10^3$ | $\approx 47$ |
| QuickTree[d] | boot (100) | 63.5 | 0.71 | $\approx 10^4$ | $\approx 15.5$ | $\approx 10^5$ | $\approx 254$ |
| BIONJ[c] | none | 32.9 | 0.44 | $\approx 10^3$ | $\approx 10.9$ | $\approx 10^5$ | $\approx 110$ |
| PhyML 3[e] | aLRT | >1,000 | 9.5 | – | – | – | – |
| RAxML VI[f] | none | >1,000 | 0.70 | – | – | – | – |

[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.

[b] Computed using phylip's protdist and default options (JTT model, no variation of rates across sites).

[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.

**Supplementary Table 1 - Topological accuracy of tree-building methods on simulated protein alignments with 10, 50, 250, or 1,250 sequences and no gaps.**

| Method | Model | N=10 | N=50 | N=250 | N=1,250 |
|---|---|---|---|---|---|
| PhyML | JTT | 0.768 | 0.797 | 0.837 | 0.827 |
| FastTree | log-corrected | 0.752 | 0.800 | 0.841 | 0.827 |
| FastME | log-corrected | 0.742 | 0.796 | 0.839 | 0.828 |
| BIONJ | log-corrected | 0.752 | 0.797 | 0.817 | 0.788 |
| BIONJ | JTT | 0.714 | 0.794 | 0.831 | 0.799 |
| BIONJ | JTT + $\Gamma$ | 0.564 | 0.624 | 0.782 | – |
| QuickTree | log-corrected | 0.746 | 0.788 | 0.810 | 0.781 |
| QuickTree | %different | 0.692 | 0.708 | 0.726 | 0.703 |
| Clearcut | log-corrected | 0.702 | 0.774 | 0.802 | 0.777 |
| FastTree | No NNI | – | – | 0.765 | 0.740 |
| FastTree | Exhaustive, no NNI | – | – | 0.764 | 0.738 |
| BIONJ | uncorrected | – | – | 0.764 | 0.738 |

**Supplementary Table 2 - Relative log-likelihoods of trees inferred by variants of FastTree for genuine protein alignments of 500 sequences.**

| Method | Average Log-Lik. | % Worse than FastTree |
|---|---|---|
| Default settings | 0.0 | – |
| Balanced joins | -106.4 | 77% |
| No NNI | -402.3 | 100% |
| Exhaustive, no NNI | -428.5 | 100% |
| BIONJ, uncorrected dist. | -514.1 | >99% |

# Supplementary Note 1: Formulas and Derivations for Neighbor Joining with Profiles

## Reduction Formulas

First we show that, for gap-free alignments and uncorrected distances, we can compute the distances between internal nodes, according to the reduction formula of neighbor-joining or BIONJ, from the profiles and the "up-distances" $u(i)$. We also give the formula for the up-distances.

When we join two nodes $i$ and $j$, we store a profile or a frequency vector at each position $l$ for the new parent node $ij$ as the (weighted) average

$$\vec{P}_l(ij) = \lambda \vec{P}_l(i) + (1-\lambda)\vec{P}_l(j)$$

where $\lambda$ is the weight (as in BIONJ) or $\lambda = 1/2$. Because the profile distances $P(i,j)$ are linear,

$$P(ij,k) = \lambda P(i,k) + (1-\lambda)P(j,k)$$

Now, if we first consider unweighted joins, the reduction formula for neighbor joining is

$$d(ij,k) = \frac{d(i,k) + d(j,k) - d(i,j)}{2}$$

FastTree uses profiles and up-distances

$$d(ij,k) = P(ij,k) - u(ij) - u(k)$$

$$u(ij) \equiv \frac{P(i,j)}{2}$$

and $u(l) \equiv 0$ for leaves. For two leaves $i$ and $j$, $P(i,j) = d(i,j)$, so this gives the correct distance between leaves. Assume the distances are correct for all nodes so far and consider the next join $ij$:

$$\begin{aligned} d(ij,k) &= \frac{d(i,k) + d(j,k) - d(i,j)}{2} \\ &= \frac{P(i,k) - u(i) - u(k) + P(j,k) - u(j) - u(k) - P(i,j) + u(i) + u(j)}{2} \\ &= \frac{P(i,k) + P(j,k)}{2} - \frac{P(i,j)}{2} - u(k) \\ &= P(ij,k) - u(ij) - u(k) \end{aligned}$$

which shows that our distances are correct for $d(ij,k)$ and, by induction, for all nodes. For weighted joins, a similar argument shows that

$$u(ij) \equiv \lambda(u(i) + d(i,ij)) + (1-\lambda)(u(j) + d(j,ij))$$

gives the same result as the distance reduction formula for BIONJ

$$d(ij,k) = \lambda(d(i,k) - d(i,ij)) + (1-\lambda)(d(j,k) - d(j,ij))$$

For BIONJ, we also need to reduce the "variance" matrix. The variance values for pairs of leaves are the same as the distance values, and the BIONJ reduction formula for variances is

$$v(ij,k) = \lambda v(i,k) + (1-\lambda)v(j,k) - \lambda(1-\lambda)v(i,j)$$

which can be computed from profile-distances by using a "variance correction" $\nu(i)$ analogous to the up-distances, where $\nu(i) \equiv 0$ for leaves and

$$v(i,j) = P(i,j) - \nu(i) - \nu(j)$$

$$\nu(ij) \equiv \lambda\nu(i) + (1-\lambda)\nu(j) + \lambda(1-\lambda)v(i,j)$$

Given these variances, BIONJ weights the join of $i, j$ so as to minimize the variance of the distance estimates for the new node $ij$, using the formula

$$\lambda = \frac{1}{2} + \frac{\sum_{k \neq i,j} (v(j,k) - v(i,k))}{2(n-2)v(i,j)}$$

$$\begin{aligned}\sum_{k \neq i,j} (v(j,k) - v(i,k)) &= \sum_{k \neq i,j} (P(j,k) - P(i,k) - \nu(j) + \nu(i) + \nu(k) - \nu(k)) \\ &= (n-2)(\nu(i) - \nu(j)) + \sum_{k \neq i,j} P(j,k) - \sum_{k \neq i,j} P(i,k)\end{aligned}$$

where $n$ is the number of active nodes before the join takes place. FastTree computes this in $\mathrm{O}(La)$ time by using the total profile to compute sums of $P(i,k)$, as will be explained below.

## Out-distances

For either neighbor joining with unweighted joins or BIONJ, we need to compute the out-distances $r(i)$ so that we can compute the neighbor-joining criterion

$$d'(i,j) \equiv d(i,j) - r(i) - r(j)$$

$$r(i) \equiv \frac{\sum_{j \neq i} d(i,j)}{n-2}$$

In the absence of gaps, the average profile distance between a node and all other nodes can be inferred from the total profile $T$:

$$\begin{aligned}\sum_{j \neq i} d(i,j) &= \sum_{j \neq i} (P(i,j) - u(i) - u(j)) \\ &= \sum_{j} P(i,j) - P(i,i) - (n-1)u(i) - \sum_{j \neq i} u(j) \\ &= nP(i,T) - P(i,i) - (n-1)u(i) - (\sum_{j} u(j) - u(i))\end{aligned}$$

which can be computed in $\mathrm{O}(La)$ time if we store the total profile $T$ and the total of all the up-distances.

## Handling Gaps

Now, consider what happens if there are gaps. The distance between two profiles becomes

$$P(i,j) \equiv \frac{\sum_{l=1}^{L} D(\vec{P}_l(i), \vec{P}_l(j)) w_l(i) w_l(j)}{\sum_{l=1}^{L} w_l(i) w_l(j)}$$

where $w_l(i)$ is the proportion of non-gaps for $i$ at position $l$. The proportion of non-gaps for an internal node is just the weighted average of the values for its children. (The profiles' frequency vectors do not include gaps.)

In the presence of gaps, the above formula for the out-distance is not a good approximation because highly gapped sequences contribute little to the total profile. Instead, we need to take the weights of the comparisons into account. Let $T - i$ be the total profile with the contribution from $i$ removed. Then

$$\sum_{i \neq j} P(i,j) \approx (n-1) P(i, T-i)$$

$$P(i, T-i) = \frac{\sum_{j \neq i} \sum_{l=1}^{L} P_l(i,j) w_l(i) w_l(j)}{\sum_{j \neq i} \sum_{l=1}^{L} w_l(i) w_l(j)}$$

$$P(i,T) = \frac{\sum_{j} \sum_{l=1}^{L} P_l(i,j) w_l(i) w_l(j)}{\sum_{j} \sum_{l=1}^{L} w_l(i) w_l(j)}$$

which leads to a formula for $P(i, T-i)$ in terms of $P(i,T)$ and $P(i,i)$ and the total weights of those comparisons. In practice, this gives a good approximation for the out-distances in the presence of gaps (data not shown).

Gaps also complicate the interpretation of the "variances" used for weighted joins. In principle, the variances should be divided by the number of non-gap positions in the comparison, as distances that are computed from more positions are more reliable. However, if we do that, the reduction formula for variances given in the previous section becomes unreliable (data not shown). Instead, we implicitly weight less-gapped sequences more highly because the less-gapped member of a join contributes more strongly to the profile.

To compute the weight for each join in the presence of gaps, we need to estimate $\sum_{k \neq i,j} P(i,k)$. To do this, split the profile-distance into its numerator and its weight or denominator

$$P(i,j) = \frac{N(i,j)}{w(i,j)}$$

Then

$$\begin{aligned} \sum_{k \neq i,j} P(i,k) &\approx \sum_{k \neq i,j} P(i,k) \frac{(n-2) \cdot w(i,k)}{\sum_{k \neq i,j} w(i,k)} \\ &= \frac{(n-2) \cdot N(i, T-i-j)}{w(i, T-i-j)} \end{aligned}$$

where $T - i - j$ represents the total profile with i and j removed, and the terms can be computed with

$$N(i, T-i-j) = n \cdot N(i,T) - N(i,i) - N(i,j)$$

$$w(i, T-i-j) = n \cdot w(i,T) - w(i,i) - w(i,j)$$