Fast Parallel Maximum Likelihood-based Protein Phylogeny

C. Blouin[1,2,3], D. Butt[1], G. Hickey[1], A. Rau-Chaplin[1].

1 – Faculty of Computer Science, Dalhousie University, Halifax, Canada, B3H 5W1
2 – Dept. of Biochemistry and Molecular Biology, Faculty of Medicine, Dalhousie University, Halifax, Canada, B3H 5X1
3 – Genome Atlantic

Keywords: Protein Phylogeny, Maximum Likelihood Methods, Parallel Computing.

Abstract

Inferring phylogenetic relationships between sequences is a difficult and interesting problem. Assuming that there is enough phylogenetic signal in biological sequence to resolve every tree bifurcation, the resulting tree is a representation of the vertical descent history of a gene. A popular method to evaluate a candidate phylogenetic tree uses the likelihood of the data, given an empirical model of character substitution. The computational cost of search for the maximum-likelihood tree is, however, very large. In this paper, we present an algorithm for protein phylogeny using a maximum likelihood framework. A key design goal, which differentiates our method from others, is that it determines a range (confidence set) of statistically equivalent trees, instead of only a single solution. We also present a number of sequential algorithmic enhancements and both sequential and parallel performance results.

Introduction

Proteins are genetically encoded polymeric molecules from linear chains of amino acids. Proteins adopt biologically active 3D structures through a process called protein folding. These structures offer the chemical environment to perform specific biological functions. Although genetics only indirectly encode 3D information, proteins have precise structures which are tuned through the evolution of their sequences. Over time, the sequence of a protein is perturbed by point mutations, insertions and deletion events. In general, phylogeny is the sequence of events involved in the development of the evolution of a gene, a species or a taxonomic group of organisms (Felsenstein, 2004). Molecular phylogeny reconstruction typically uses point mutations as a signal to infer the evolutionary history of homologous sequences. These reconstructed histories are represented by phylogenetic trees. Assuming that a gene has propagated via vertical descent (i.e. from a parent to subsequent generations), a phylogeny inferred from sequences can be extrapolated to the evolution of their respective host species.

While a protein structure remains constant, all *aligned residues* in one column of a multiple sequence alignment (a site) are assumed to be in positional homology. This assumption allows the evolution of a site to be modeled as a distance-dependent Markov process of substitution (Felsenstein, 1981). Given this model, the tree topology which maximizes the likelihood of an alignment is known as the maximum likelihood (ML) tree.

In terms of graph theory, a rooted phylogenetic tree is a binary tree. Each node represents a sequence while each edge, referred herein as branch length, represents the evolutionary distance between two sequences. This distance is measured in expected substitution per site. The main challenge in protein phylogeny is to traverse the search space to find the topology of the maximum likelihood tree. The search space for a tree with n terminal sequences is large, approximately equal to the term n_T defined in Equation 1 (Felsenstein, 2004).

$$n_T = \frac{(2n-3)!}{2^{(n-2)}(n-2)!} \qquad [1]$$

For very small numbers of sequences, it is possible to exhaustively search the solution space. However, this solution becomes intractable for datasets with 15 or more sequences. For the general case, it is thought that the problem of maximum likelihood phylogeny is NP-hard (Felsenstein, 2004). Global methods to traverse this search space have already been implemented, specifically for protein phylogeny in PROML (Felsenstein, 2002) and PAML (Yang, 1997). A selection of clever heuristics have been devised including Likelihood-Neighbor joining hybrid algorithms (Ota, 2001; Ota, 2000), Nearest Neighbors Interchange (Guindon, 2003) or the structural EM algorithm (Friedman, 2002). Although the performance of these heuristics is drastically better than global rearrangement search methods, their accuracy and robustness are affected (Vinh le, 2004). Another class of methods, Tree-Puzzle (Schmidt, 2002) are quartet based methods of reconstruction which are also popular in protein phylogeny.

In this paper, we present an algorithm with emphasis on a systematic tree traversal for protein phylogeny that is suitable for a parallel implementation. Our interest is in a practical algorithm that permits the solution of large protein phylogeny problems. This being the case, we explore both parallel and sequential efficiency enhancements. The

challenge is that the sequential enhancements often make the computation less regular, and therefore may reduce total time, yet still have a negative effect on speedup.

Basic Parallel Method

Parallel computation has been successfully applied to DNA phylogeny in packages such as Parallel DNAml (Ceron, 1998), and a parallel implementation of fastDNAml (Olsen, 1994). Parallel protein phylogeny can also be performed using the quartet puzzling method (Strimmer, 1996) implemented in TREE-PUZZLE (Schmidt, 2002). In this section, we describe our basic parallel approach for protein phylogeny using a maximum likelihood framework. A key design goal, which differentiates our method from others, is that we are interested in determining a range (confidence set) of statistically equivalent trees, instead of only the best solution. The membership of the confidence set can be defined by a selection of ad hoc criterion or using confidence interval approximations. It is unclear as to what constitutes the real confidence intervals, so the confidence sets should be regarded as approximation at this point. The statistical properties of using confidence intervals approximation as a means to determine the memberships of candidate tree topologies is beyond the scope of this paper but is described in Pepke *et al.*.

Because we are ultimately interested in approximating confidence intervals in tree space, our basic algorithm attempts to explore a broad sample of topologies through the enumeration of all possible Subtree Pruning Regrafting (SPR) operations (Felsenstein, 2004) for each tree in the solution pool. At each iteration, the SPR enumeration is performed on all trees newly added to the solution pool. The likelihood of each of these enumerated trees is optimized with respect to branch lengths. All returned trees are compared to the current best tree using a specified statistical test. The computation terminates when both the reference tree is unchanged and the evaluation of the trees corresponding to all SPR enumerations in the solution pool have been completed (i.e. no new trees are returned from an iteration). As a practical matter it may also be terminated when a user specified maximum number of iterations, M, has been reached.

Our basic parallel method, as executed by the “Master” processor, is sketched in Algorithm 1; while Algorithm 2 outlines the task of the “Worker” processors. The key issues driving the design of these algorithms were as follows: 1) How to minimize communication overhead, 2) How to evenly distribute workload across the processors, 3) How to avoid re-evaluating trees which have been traversed in previous iterations, and lastly 4) How to efficiently perform branch length optimization, which is the most computationally intensive component of the method. In the remainder of this section we will discuss Issues 1 and 2, while Issues 3 and 4 are discussed in the following sections.

Our general approach to minimizing communication is to perform some redundant computations. Whenever possible, Algorithm 1 avoids communicating sets of trees by communicating singleton trees and allowing the worker processors to reconstruct the sets at the cost of some redundant computation. For example, the Master processor communicates single trees from the current pool at Line 9 forcing the redundant computation of unique SPR descendants by the Workers (Lines 6-11) but saving on significant communication. The one place where communication is not spared is in the repeated communication of the reference tree, T_r, by the Master (Line 7). Maintaining this global information as to the best tree found globally to date allows the Workers to prune

their local pools, LP, and reduces both local computational costs (Lines 13-17) and communication costs (Line 21).

Algorithm 1 covSEARCH Master

Input: *A* – the sequence alignment. *M* – the maximum number of iterations. FF – a filter function to determine trees in the confidence set. MLtolerance – a Maximum likelihood tuning parameter.

Output: *P* - The set of most likely trees.

1: From the alignment A, load an initial pool of trees, P_0, or generate one using neighbour joining.
2: $T_r \leftarrow T \in P$ with highest ML value
3: Perform FF on P removing any trees which are sufficiently unlikely compared to T_r.
4: $P_1 \leftarrow P_0$. $i \leftarrow 2$. done $\leftarrow$ FALSE.
5: WHILE NOT done
6: $P_i \leftarrow \varnothing$.
7: Broadcast reference tree T_r to all workers.
8: FOR ALL $T \in P_{i-1}$
9: Broadcast T to all workers.
10: * Workers perform their local computation and return resulting trees to the master*\
11: FROM EACH worker
12: Receive result set of trees J and set $P_i \leftarrow P_i \cup J$
13: IF there is a tree $T \in P_i$ with a higher ML value than T_r THEN
14: $T_r \leftarrow T$.
15: Perform FF on P_i removing trees which are sufficiently unlikely compared to T_r.
16: $P \leftarrow P \cup P_i$
17: IF P_i is empty or $i > M$ THEN
18: done $\leftarrow$ TRUE
19: Perform FF on P removing trees which are sufficiently unlikely compared to T_r.
20: ELSE
21: $i \leftarrow i + 1$
22: Return P, the final pool of trees, and, T_r, the most likely tree found.

Algorithm 2 covSEARCH Worker j

1: WHILE true DO
2: Receive a tree T from Master Processor.
3: If T is a new reference tree THEN
4: $T_r \leftarrow T \in P_{i-1}$
5: ELSE /* T is a tree from the pool*/
6: LP $\leftarrow$ all distinct SPR permutations of T
7: FOR ALL $T \in$ LP
8: IF $T \in$ TreeCache THEN
9: LP $\leftarrow$ LP – {T}
10: ELSE add T to TreeCache
11: Remove from LP all but every j^{th} tree
12: FOR ALL $T \in$ LP
13: Compute Maximum Likelihood of T using existing branch lengths
14: IF ML(T) + MLtolerance >= ML(T_r) THEN
15: Optimize the Branch Lengths of T
16: Recomputed the Maximum Likelihood of T
17: ELSE LP $\leftarrow$ LP – {T}
18: IF there exists a $T \in$ LP such that ML(T) > ML(T_r) THEN
19: $T_r \leftarrow T$
20: Perform FF on LP removing trees which are sufficiently unlikely compared to T_r.
21: Send trees in LP back to the Master Processor.

Avoiding Redundant Search

As the computation progresses, the probability of re-evaluating a tree which was traversed in a previous iteration increases. Especially when the search is about to converge, this duplication of the computation unnecessarily extends the tree search. To implement the Tree Cache used in Lines 8-10 of Algorithm 2, we have applied a string-based method for recording tree topologies. This method allows us to maintain a compact history of the search which is then used to avoid redundant tree evaluation.

The first challenge was to define an algorithm that generates a unique string representation for each topologically distinct tree. This task is performed by re-rooting each tree to the parent of an arbitrary terminal node: the sequence whose label is the first in the alphabetical order of all sequence labels. It thus generates a NEWICK string representation with the property that unique tree topologies result in unique string representations. The branch length values are then stripped to preserve only the topological information. An example of a string representation follows:

(A,((B,Z),C))

The Tree Cache itself was implemented as a PATRICIA tree (Morrison, 1968). In this compact Trie based data structure there is node for each common suffix in the stored strings. A Patricia Tree compacts this by merging single child nodes with their parents, and usually stores the strings in binary form. A new node in the tree is created when a difference in the string is encountered. Each internal node contains the character of the string that is different. Therefore, concatenating from a path from the root to an external node gives a string that is stored in the tree. For n keys stored in the tree, there are n external nodes. For keys of length k, insertion requires $O(\log(n+k))$ time assuming the keys are evenly distributed, while lookup requires $O(k)$ time. All string-based representations of phylogenetic trees with the same number of taxa are of the same length. Since only the differences in the strings need to be stored, this saves considerably on memory space. An obvious alternative approach would be a hash based scheme; however in practice we obtained better performance using the PATRICIA tree.

As the topological search expands from the current maximum likelihood tree, the candidate trees are one step of SPR away from the pool of the previous generation. Consequently, a fraction of the candidate in a given iteration will be overlapping with the candidate trees of the previous iteration(s). In fact, it appears that most trees that are generated through an enumerative SPR search strategy are overlapping with previously traversed topologies. Because the definition of confidence intervals requires a thorough traversal of the search space, this effect is marked for broad-search for which the membership to the solution pool is determined using a statistical criterion. Assuming that each candidate tree takes approximately the same amount of time to optimize its branch length using the maximum likelihood criterion, and that the time to look up a string in a PATRICIA is much smaller than optimizing an entire tree, the relative speedup of maintaining a history will be roughly proportional to the ratio of unique trees to total topology generated (Table 1). Note that an important feature of our Tree Cache is that it is global, that is to say the cache maintained on each processor is identical and records all trees visited previously by any processor. This is a second advantage of performing redundant SPR operations in Line 6 of Algorithm 2.

Pfam set	Unique topologies[1] [ml/SH][2]	Total topologies[3] [ml/SH]	Duplication [ml/SH]	Number of solution in final pool (SH test)
Cytochrom_B_C[4]	215/22,980	340/95,490	1.6/4.1	584
Cytochrom_B_N[5]	79/1671	114/7326	1.4/4.4	63
MAP[6]	1243/-	1614/-	1.3/-	>6000
clade 1 (7 seqs.)	97/821	172/9366	1.8/11.4	113
Clade 2 (6 seqs.)	43/105	110/2738	2.6/26.1	52

Table 1: Overlap caused by the enumerative SPR tree search strategy.

Efficient Computation of Branch Lengths

The likelihood of substitution is derived from matrices of probabilities of character substitution. Typical substitution matrices are developed using empirical data and different derivation methods. For amino acid data, the most commonly used matrices are PAM (Dayhoff, 1978), JTT (Jones, 1992) and WAG (Whelan, 2001). These matrices contain instant rate of substitution. From these instant rate matrices, it is possible to compute distance dependent matrices of probabilities of substitution. To optimize the branch lengths, the distance dependent probability matrices have to be continuously computed. For an evolutionary distance *t*, and given an instant rate matrix Q, the matrix of probability of substitution P is computed according to Equation 2.

$$P(t) = e^{Qt} \qquad [2]$$

The likelihood of observing a substitution from state *i* to *j* can thus be expressed as a function of *t* as $P_{ij}(t)$. The calculation of *P* is computationally costly. One method to compute the P-matrix proceeds through finding the *Q* matrix eigenvectors and corresponding eigenvalues by a divide-and-conquer approach. This algorithm requires on the order of $O(20^3)$ floating point calculations. As a first step, $Pi \rightarrow j$ values can be approximated by interpolation using Chebyshev polynomials. It has been shown that Chebyshev polynomials can significantly decrease the time to calculate *P* without noticeable error in likelihood values (Pupko, 2002).

In this paper our key approach to efficiently performing branch length optimizations is to cache the most commonly occurring values of P(*t*). Likelihoods are computed using matrices of probability of substitution P(*t*). As the number of internal nodes grows large, generating these matrices becomes a significant computational bottleneck. For a given model of substitution, there is a single P-matrix for each evolutionary distance *t*. Although the variable *t* is a continuous variable, no attempts are made to optimize this parameter beyond the precision threshold of 10E-5, some 90,000,000 matrices are necessary to cover the reasonable range to *t* from 0.00001 to 900. Since each matrix is 1600 bytes, 144 Gb of RAM would be required to store all possible P-matrices. This is

[1] Number of unique tree topology traversed during the calculation.
[2] ml: tree search with solution pool limited to one tree. sh : tree search with solution pool membership determined by the SH test.
[3] Number of tree topology generated through SPR enumeration.
[4] Pfam::Cytochrom_B_C seed alignment : 9 sequences X 79 sites, 84% average sequence identity.
[5] Pfam::Cytochrom_B_N seed alignment: 8 sequences X 190 sites (edited), 69% average sequence identity.
[6] Pfam::MAP seed alignment : 13 sequences X 87 sites, 51% average sequence identity

more memory than is typically available; however it turns out that it is not necessary to store all 90,000,000 P-matrices.

In general, the distribution of branch lengths is not uniform over the 0.00001 to 900 range. Caching only the most commonly used matrices appears thus to be appropriate. However, there is no reason to believe that there is a universal distribution of value *t*. The collection of cached matrices should thus be determined as the calculation progresses. Each input alignment will result in different tree topologies and associated branch lengths. The cache should therefore keep track of which branch lengths are being used most often, and save the corresponding P-matrix.

In our implementation the P-matrix cache is implemented using a hash table with chaining. For a hash function, we simply use the mantissa bits of the floating point branch length. These lengths are rounded to increase the cache hit probability as described previously. One challenge is how to handle the situation when the size of the P-matrix cache reaches the redefined maximum size. Initially, we attempted to maintain access frequency counts for each entry to determine the least used so that it might be discarded. However, this approach has significant overhead which appears to make it unattractive in practice. Instead, we maintain an additional move-to-front data structure which contains an entry for each hashed matrix. Entries in this structure are moved to the front every time the associated matrix is referenced. When, to free up storage, a matrix must be discarded the "oldest" one, that is the one at the end of the move-to-front data structure, is selected. In this approach every operation requires expected O(1) time and the memory footprint is minimal.

Some branch length precision is lost when *t* is converted to obtain an integer value. Branch lengths, *t*, with very similar values, such as 1.245391 and 1.245398, get hashed to the same value floor(10000*t) = 124539. Observations indicate that the rounding the variable *t* is minimal (Table 2) and does not affect the likelihood ranking of the phylogenetic trees.

Range in *t*	**P-matrix entry (average difference)**	**P-matrix entry (max difference)**
0.123450 and 0.123459	8.65e-09	1.29e-07
1.345670 and 1.345679	8.52e-09	1.27e-07
10.342190 and 10.342199	7.64e-09	1.09e-07
50.781560 and 50.781569	4.76e-09	5.68e-08
100.975420 and 100.975429	2.77e-09	3.08e-08

Table 2: Sample rounding error on P-matrices due to hashing.

Furthermore, since substitutions are modeled as a reversible Markov-process, the probability of substitution of P(i->j) converges to the equilibrium frequency of j as t becomes large. In practice, P-matrices for t > 1.5 are essentially identical and the precision appears to be superfluous for the computation of likelihood and the ranking of trees according to the ML criterion.

Table 3 demonstrates that a caching between 32-64 thousand matrices is sufficient to store most of the matrices required to optimize trees with relatively similar topologies. The benefit of caching these matrices is a relative sequential speedup of between 15% and

20%. Storing more matrices continues to improve the hit percentage, but the improvement is likely not large enough to warrant the extra storage space.

A.a.fas 34x347				A.a.small.fas 17x347				Longal1.fas 34x1000			
Cache Size	Time	Hit %	Speed up %	Cache Size	Time	Hit %	Speed up %	Cache Size	Time	Hit %	Speed up %
0	1:36:28	-	-	0	3:52.85	-	-	0	1:07:34	-	-
8192	1:17:38	70	19.5	8192	3:13.61	52.6	16.8	8192	56:52.09	84.7	15.8
16384	1:17:32	70.8	19.6	16384	3:13.27	53.6	16.9	16384	57:07	85.3	15.4
32768	1:17:22	72	19.7	32768	3:12.66	55	17.2	32768	55:31	85.9	17.8
65536	1:17:12	73.9	19.9	65536	3:12.48	57.2	17.3	65536	55:29.6	86.8	17.8
131092	1:16:46	77	20.4	131092	3:11.01	60.9	17.9	131092	55:24.70	88.3	17.9

Table 3: Effects of P-matrix caching on sequential performance on three data sets.

Experimental Evaluation

We have implemented the parallel covSEARCH algorithm as presented in the previous sections. Our sequential code with the tree and P-matrix caching is written in C++ and makes extensive use of the phylogenetic library, libcov . Our parallel code is based on our sequential code and communication operations drawn from the MPI communication library.

The following performance evaluation uses 16 processor Beowulf style cluster. This shared nothing parallel machine consists of a collection of 1.8 GHz Intel Xeon processors each with 1 GB of RAM, a 40 GB 7200 RPM IDE disk and an onboard Inter Pro 1000 XT NIC. Each processor is running Linux Redhat 7.2 with gcc 2.95.3 and MPI/LAM 6.5.6. as part of a ROCKS cluster distribution. All processors are interconnected via a Cisco 6509 GigE switch.

In the following experiments, all sequential times are measured as wall clock times in seconds. All parallel times are measured as the wall clock time between the start of the first process and the termination of the last process. All times include the time taken to read the input from files and write the output into files. Furthermore, all wall clock times are measured with no other users on the machine.

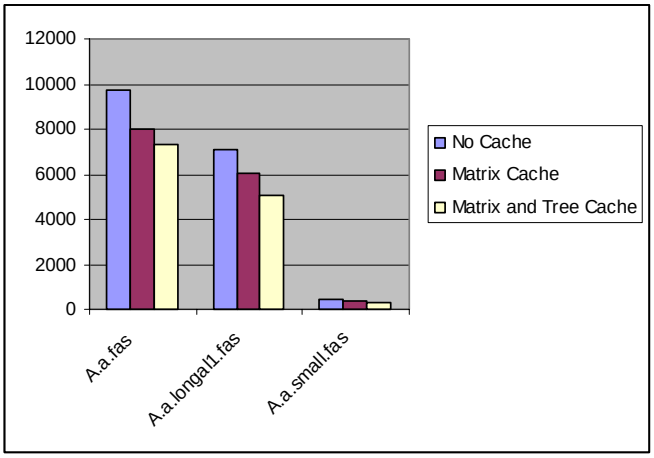

Figure 1: Cumulative effect of P-Matrix and Tree caches on wall clock time in seconds.

Figure 1 shows the cumulative impact of tree and P-matrices caching on the performance of covSEARCH method on a variety of datasets, and the corresponding relative speedup. We observe that these caching schemes improve sequential performance by between 25% and 30%.

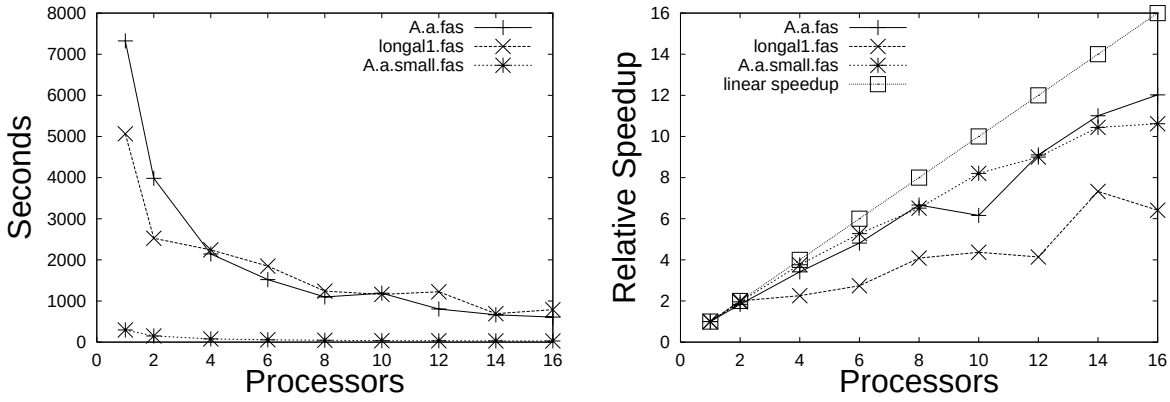

Figure 2: Parallel wall clock time in seconds as a function of the number of processors for three datasets and the corresponding speedup.

Figure 2 shows parallel wall clock time in seconds as a function of the number of processors for three datasets of varying size in terms of both number of sequences and number of sites, and the corresponding relative speedup. We observe that for two of these datasets covSEARCH exhibits near linear relative speedup on up to 8 processors and good speedup on up to 16 processors. For the third dataset we observe around 50% speedup. The problem here appears to be that we are not achieving a sufficiently even balance of workload in the branch optimization phase of the algorithm. We are currently exploring ways to predict which branch length optimization problems are likely to require significantly greater time so that we may ensure that they are distributed more evenly across the processors.

Conclusion

In this paper, we have presented a parallel algorithm for protein phylogeny using a maximum likelihood framework. A key feature of our covSEARCH approach is that it determines a range (confidence set) of statistically equivalent trees, instead of only a single solution. By treating the topologies of the phylogenetic tree as a variable with an intrinsic uncertainty, we have discovered that most multiple sequence alignments used to infer phylogeny do not contain enough information to be fully resolved into a single tree. Obtaining these types of results was made possible only by the enhanced throughput in computation provided by the parallel framework.

Acknowledgments

The authors acknowledge Dr. A. J. Roger, Dalhousie University for his involvement in the phylogeny aspect of this project. The author also thanks Ms. J. Murdoch for her implementation of the treespace module that is part of the libcov library. This work was

supported by the Genome Atlantic grant on Prokaryotic genome diversity and evolution and NSERC discovery grants 298397-04 (CB) and 170169-04 (ARC).

References

Butt, D. J., G. Hickey, A. J. Roger and C. Blouin. 2005. libcov: A C++ bioinformatic library to manipulate protein structures, sequence alignments and phylogeny, BMC Bioinformatics 2005, 6:138 (6 June 2005)

Ceron, C., J. Dopazo, E. L. Zapata, J. M. Carazo and O. Trelles. 1998. Parallel implementation of dnaml program on message-passing architectures. Parallel Computing. 24:710-716.

Dayhoff, M. O., R. M. Schwartz and B. C. Orcutt (1978). A model of evolutionary change in proteins. Atlas of protein sequence and structure. M. O. Dayhoff. Silver Spring, MA, National Biomedical Research Foundation. 5: 345-352.

Doolittle, W. F. 1999. Phylogenetic classification and the universal tree. Science. 284:2124-9.

_____. 2000. The nature of the universal ancestor and the evolution of the proteome. Curr Opin Struct Biol. 10:355-8.

Felsenstein, J. 1981. Evolutionary trees from DNA sequences: a maximum likelihood approach. J Mol Evol. 17:368-76.

_____. 2002. PHYLIP (Phylogeny Inference Package) version 3.6. Distributed by the author, Dept. of Genetics, U. of Washington, Seattle, Wa.

_____. 2004. Inferring Phylogenies. Sinauer Associates, Inc., Sunderland, MA.

Friedman, N., M. Ninio, I. Pe'er and T. Pupko. 2002. A structural EM algorithm for phylogenetic inference. J Comput Biol. 9:331-53.

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

Jones, D. T., W. R. Taylor and J. M. Thornton. 1992. The rapid generation of mutation data matrices from protein sequences. Comput Appl Biosci. 8:275-82.

Morrison, D. R. 1968. Patricia - practical algorithm to retrieve information coded in alphanumeric. Journal of ACM. 15:514-534.

Olsen, G. J., H. Matsuda, R. Hagstrom and R. Overbeek. 1994. fastDNAmL: a tool for construction of phylogenetic trees of DNA sequences using maximum likelihood. Comput Appl Biosci. 10:41-8.

Ota, S. and W. H. Li. 2000. NJML: a hybrid algorithm for the neighbor-joining and maximum-likelihood methods. Mol Biol Evol. 17:1401-9.

_____. 2001. NJML+: an extension of the NJML method to handle protein sequence data and computer software implementation. Mol Biol Evol. 18:1983-92.

Pupko, T. and D. Graur. 2002. Fast computation of maximum likelihood trees by numerical approximation of amino acid replacement probabilities. Computational Statistics & Data Analysis. 40:285-291.

Roberts, D. L. and A. R. Solow. 2003. Flightless birds: when did the dodo become extinct? Nature. 426:245.

Schmidt, H. A., K. Strimmer, M. Vingron and A. von Haeseler. 2002. TREE-PUZZLE: maximum likelihood phylogenetic analysis using quartets and parallel computing. Bioinformatics. 18:502-4.

Strimmer, K. and A. von Haeseler. 1996. Quartet puzzling: A quartet maximum likelihood method for reconstructing tree topologies. J Mol Biol. 13:964-969.

Vinh le, S. and A. Von Haeseler. 2004. IQPNNI: Moving Fast Through Tree Space and Stopping in Time. Mol Biol Evol. 21:1565-71.

Whelan, S. and N. Goldman. 2001. A general empirical model of protein evolution derived from multiple protein families using a maximum-likelihood approach. Mol Biol Evol. 18:691-9.

Yang, Z. 1997. PAML: a program package for phylogenetic analysis by maximum likelihood. Comput Appl Biosci. 13:555-6.