Using Parallel Computing for Maximum Likelihood Phylogenetic Analysis

Heiko A. Schmidt and Martin Vingron

Deutsches Krebsforschungszentrum, Heidelberg, Germany

Introduction

Phylogenetic reconstruction is used in several fields of genetic sequence analysis. It is applied not only for inferring relationships between species or within gene or protein families. It is also used for classifying organisms, e.g. viruses, and for reconstructing duplication events and their chronological order in genomes. Hence phylogenetic reconstruction plays an important role in understanding the development of sequences and genomes.

There are several methods for inferring phylogenetic trees from genetic sequence data. Approaches using maximum likelihood (ML) based techniques belong to the most respected. Since Felsenstein (1981) has first introduced this approach several further improvements and methods have been published (e.g. Olsen *et al.* 1994; Strimmer and von Haeseler 1996). Here we used the ML approach of Olsen *et al.* (1994) and present a way to use it for larger datasets.

1 The Algorithm

1.1 Computing the ML Value of a Tree

The algorithm introduced by Felsenstein (1981) for computing the maximum likelihood (ML) value for a 'fixed' binary tree topology models evolution as a reversible Markov process described by a transformation matrix. The ML value describes the probability, that this tree has given rise to the data (i.e. the multiple sequence alignment).

To compute the ML value one starts placing a root at an arbitrary location in the unrooted tree topology. From this root likelihood values are computed recursively for all subtrees, one for each alignment column. These likelihood values are summed up for the subtrees. In the end all columns are multiplied to a global likelihood value.

All edges, one after another, are now adjusted to maximize this value getting a maximum likelihood value.

For details see Felsenstein (1981) and Olsen *et al.* (1994).

1.2 Finding Optimal Trees: Brute Force

Given this algorithm, the best way would be to apply it to the whole set of possible binary tree topologies. After having all ML values one chooses the topology with the best ML value (Fig. 1).

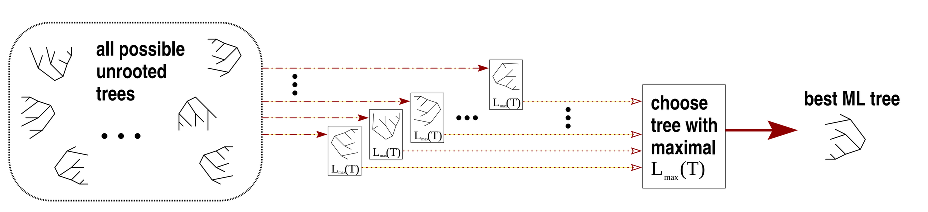


Figure 1: Exhaustive search for an optimal tree topology

This is a nice idea, but it is not feasible in practice, because the number of possible trees for n sequences is

$$\prod_{k=3}^{n}(2k-5) = \frac{(2n-5)!}{2^{n-3}(n-3)!}$$

(Fig. 4) and for multifurcating trees this number is even worse. Hence heuristic methods are applied.

1.3 Finding Optimal Trees: Stepwise Insertion

The method used here applies stepwise insertion to find a best tree. I.e. one starts building a tree from three sequences. Then one adds one sequence after another to the growing topologies using the edges giving the highest likelihood (Fig. 2).

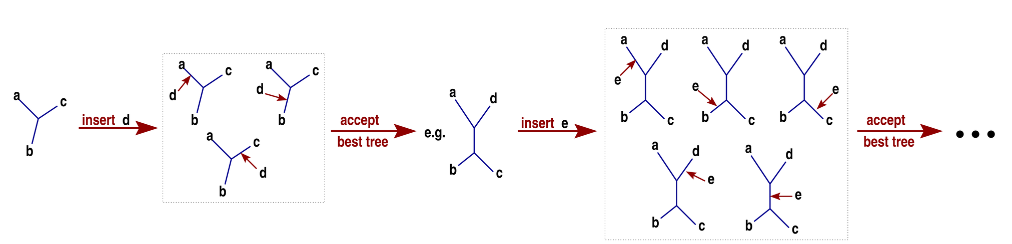


Figure 2: Stepwise insertion

Since this heuristic highly depends on the input order of the sequences, branch swapping is used to find better trees.

1.4 Finding Optimal Trees: Branch Swapping

Each subtree is cut off the current tree and reinserted into the remaining tree (Fig. 3) at every edge in a neighborhood of g edges (default $g = 1$). To every resulting rearrangement the ML evaluation is applied again until a better tree cannot be found any more.

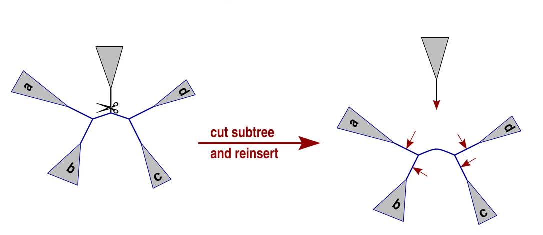


Figure 3: Branch swapping to rearrange the tree

After having added all sequences, a global rearrangement (g=*all possible*) is done by default.

2 Parallelization of the Algorithm

The amount of evaluated trees drops dramatically by using these heuristics (Fig. 4). But still the computation time is immense for larger data sets. Hence we applied parallel computation methods to decrease computation time.

2.1 Approach

Our parallelization approaches the number of trees possible to be evaluated concurrently in each step of the heuristic. The amount of trees in each of the steps grows linear with the number of sequences (i) involved (insertion: $2i-5$, local rearrangement with $g = 1$: $2i-6$), while in the global rearrangement it grows quadratically by approximately $4n^2 - 26n + 38$ (Fig. 4). We found this worth trying to parallelize.

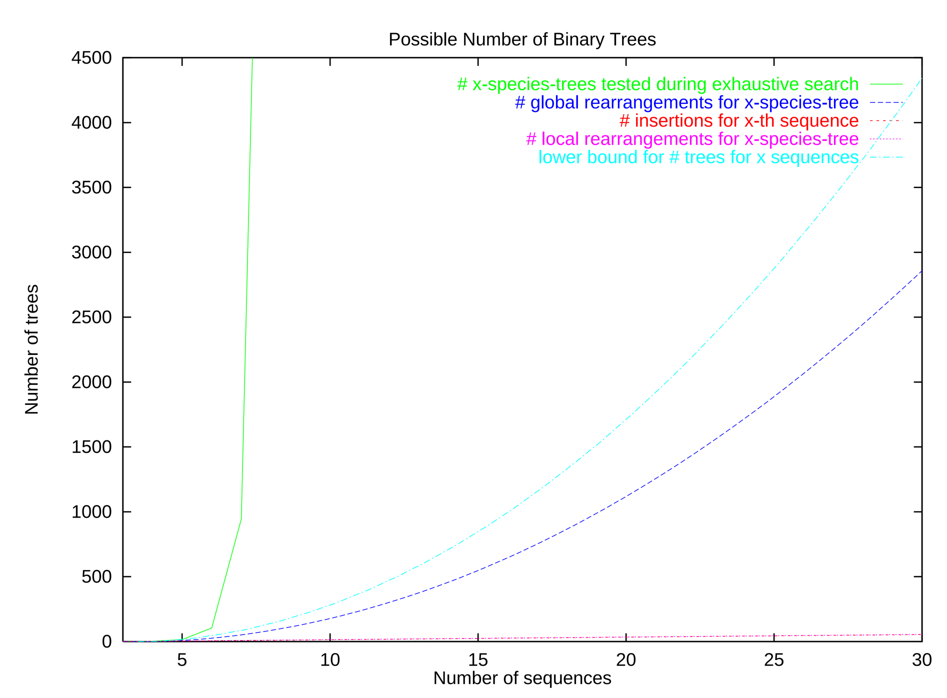


Figure 4: Numbers of possible binary trees in the analyses: for insertion, local and global rearrangement, as well as the lower bound, the sum of these. The green line gives the number of trees to be evaluated in an exhaustive search for the optimal tree.

2.2 Implementation

We first ported `fastDNAml` (Olsen *et al.*, 1994) to ANSI-C for portability reasons and recoded parts of the algorithm. Then we applied a master-slave topology (Fig. 5) that we implemented using MPI (Message Passing Interface, Snir *et al.* 1996), the *de facto* standard in message passing.

In our implementation a master process generates and rearranges the trees, while the slave processes carry out the ML evaluations (Fig. 5).

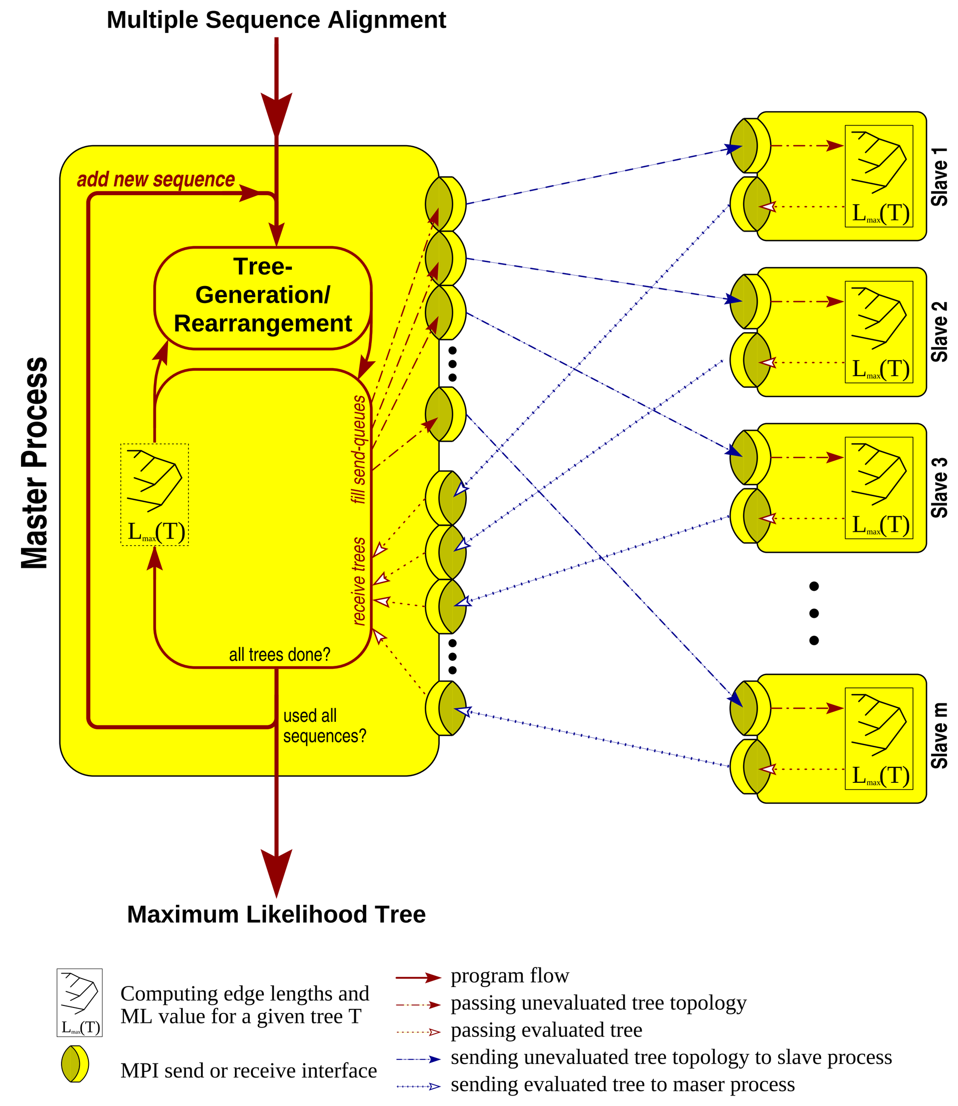


Figure 5: Structure of the parallel implementation `pfastDNAml`. The yellow boxes are separate processes in the parallel run.

3 The Program

The requirements for running `pfastDNAml` are:

- an ANSI C compiler (e.g. `gcc`, freely available)
- a Message Passing Interface (MPI) library 1.1 (e.g. MPICH or LAM, freely available)
- the `pfastDNAml` source code (please contact the authors)
- ... and you should have parallel computing platform available: e.g. a COW (Cluster of Workstations) or a parallel machine (SMP or MPP).

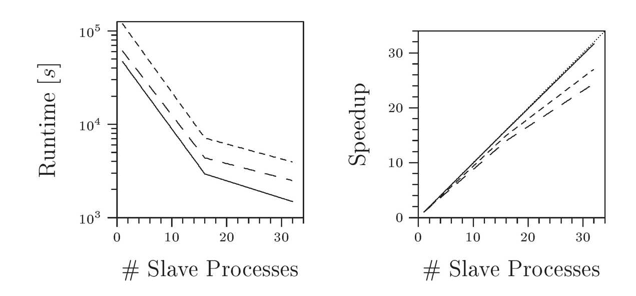
Because of using only standard software and libraries, which are also available for free the code should be portable to all kinds of platforms without changes.

4 Results

4.1 On Parallel Computers

In tests on several parallel computers like IBM SP2, NEC Cenju3, and Convex Exemplar S Class our implementation `pfastDNAml` scaled very well with increasing numbers of processors (Fig. 6) especially for growing datasets, because the global rearrangement part grows quadratically.

Figure 6: Runtimes and speedup of `pfastDNAml` on an IBM SP2 for 28r rRNA sequences (2 glob. rearr.). Linear speedup (dotted), 1st glob. rearrangements (solid), runtime with 1st (long dashed), and both (long dashed) rearrangements.



4.2 On Distributed Systems

Analyses of the Gantt charts and profiles, like Fig. 7, of parallel runs on a COW (cluster of workstations) of 12 SUN workstations show that computational resources remain unused, if the number of trees per step is small compared to the number of slave processes available. But this is on average leveled out in the later steps, where the number of dispatched trees and the computation time per tree increases.

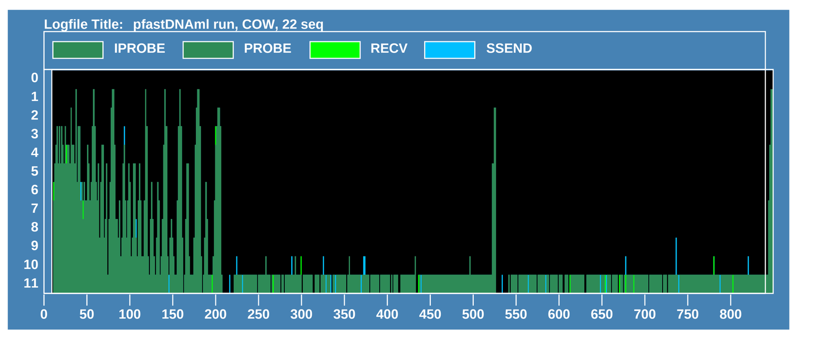


Figure 7: Profile of a parallel run of `pfastDNAml` for 22 sequences on a COW of 20 SUNs. Black means computation, light green and cyan are communication overhead, and dark green stands for waiting for communication.

4.3 Conclusion

Our tests have shown that our implementation is useful for parallel as well as for distributed platforms like COWs. We could reach tree reconstruction times of about 15 min for 28 species trees during daytime load of the COW, but this highly depends on the number of rearrangements required and the quality of the alignment.

Due to the reduction of the wall clock time needed for computation, this implementation gives us the possibility to reconstruct trees from larger sets of sequences.

5 Future Work

By using a more sophisticated dynamic scheduling scheme we want to decrease the amount of unused computational resources, to cope even better with the problems occurring on COWs with machines being very inhomogeneous in computational power.

References

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

Olsen, G. J., Matsuda, H., Hagstrom, R. and Overbeek, R. (1994) fastDNAml: A tool for construction of phylogenetic trees of DNA sequences using maximum likelihood. *Comput. Appl. Biosci.*, **10**, 41–48.

Snir, M., Otto, S. W., Huss-Lederman, S., Walker, D. W. and Dongarra, J. (1996) *MPI: The Complete Reference.* The MIT Press, Cambridge, Massachusetts.

Strimmer, K. and von Haeseler, A. (1996) Quartet puzzling: A quartet maximum–likelihood method for reconstructing tree topologies. *Mol. Biol. Evol.*, **13**, 964–969.

Acknowledgments

We thank the German National Research Center for Information Technology (GMD, Sankt Augustin) and NEC Europe Ltd. (C&C Research Laboratories, Sankt Augustin) for access to computing facilities.

Address

Deutsches Krebsforschungszentrum
Abt. Theoretische Bioinformatik (H0300)
Im Neuenheimer Feld 280
D-69120 Heidelberg, Germany
phone: ++49 +6221 42-2719, fax: -2849
email: `{H.Schmidt|M.Vingron}@DKFZ-Heidelberg.de`