New phylogenetic venues opened by a novel implementation of the DNAml algorithm

O. Trelles¹, C. Ceron¹, H.C. Wang², J. Dopazo³ and J.M Carazo¹,²

¹Computer Architecture Department, University of Malaga, 29071 Malaga, ²Biocomputing Unit, National Center for Biotechnology, 28049 Madrid and ³Bioinformatics, Glaxo Wellcome, 28760 Madrid, Spain

Received on February 9, 1998; revised and accepted on April 23, 1998

Abstract

Summary: This note describes the parallelization of a key biological application, such as the construction of phylogenetic trees, using the DNAml program. A careful run-time analysis allowed us to propose an efficient parallelization that scaled very well to large problem sizes.

Availability: By anonymous ftp at ftp.ac.uma.es (/pub/ots/pDNAm) or from our web site http://www.cnb.uam.es/www/software/software_index.html

Contact: ots@ac.uma.es

The computational demands of the DNAml program (Felsenstein, 1988) can quickly stress the computer resource of most research laboratories. As an example, it could be mentioned that this was the research subject that consumed most of the CPU time of the first IBM SP1 system in the Argonne National Laboratory (1993). Earlier approaches to applying maximum likelihood methods to very large sets of sequences have been centered on the development of new, simpler, algorithms, such as fastDNAml (Olsen et al., 1994). However, in this work, we have centered our attention on the original Felsenstein version of the method. The specific maximum likelihood algorithm used in this work corresponds to the current version of the DNAml program, obtained by anonymous ftp from evolution.genetics.washington.edu.

The DNAml program estimates phylogenies using the maximum likelihood criterion, one of the most widely accepted methods of statistical estimation. The DNAml program basically consists of two main coupled tasks: (i) insertion of a new sequence in the current topology and (ii) (local) optimization of the resulting topology. An additional final step of global optimization is optionally performed when all sequences have been incorporated in the topology. This last step is very similar to the second one from the algorithmic point of view, but its computational cost is extremely high because of the large number of topologies that must be evaluated.

The insertion step starts with a current-tree and a new sequence that must be inserted in the tree in that place which maximizes the likelihood value. To perform this, the program evaluates each of the $2k - 3$ possibilities ($k$ being the number of sequences in the current-tree), computing its likelihood. Then, the tree with the greatest likelihood value is selected to be the best-tree. Because the number of tree-evaluation tasks is high for a sufficiently large number of sequences, the proposed scheme of parallelization is one of the master–server type. The master process works as a by-demand tasks dispatcher for tasks that are computationally solved by the servers. When the servers send the results to the master, it selects that tree with the greatest likelihood (best-tree). The master is also prepared to send tasks in advance using a buffering scheme already successfully tested in Trelles et al. (1994).

In the local optimization step, minor rearrangements of the best-tree are performed, looking for an increase in likelihood. Thus, given a current-tree $T_k$ with likelihood $L_k$, one of its $k$ nodes is removed and rearranged in its two neighbor nodes. This process results in two new trees, $T_k$ and $\hat{T}_k$, with likelihood $L_k$ and $\hat{L}_k$, respectively. The tree with the greater likelihood value (including $L_k$) is chosen as the new best-tree, and it replaces the current-tree. This procedure is performed until the set of nodes to be rearranged is exhausted.

Strictly speaking, the reorganization of the tasks of one node is dependent on the reorganization of the previous nodes, due to the replacement of the current best-tree. From the point of view of an efficient parallelization of this step, this is a very serious drawback that has its origin in the fact that each new optimization task must be performed over the last best-tree found, and not over the initial topology. In practical terms, this situation makes it impossible to solve all the optimization tasks concurrently because there are only two tasks that can be solved in parallel and the maximum theoretical speed-up of this step will be limited to this value (2).

To find a solution to this task-dependency problem, we have studied the run-time behavior of the algorithm, observing that the number of times that a tree with a likelihood better than the current likelihood is obtained was extremely low (see Figure 1), which is equivalent to considering that in most of the cases there is a high probability that a best-tree will not be produced. The most important implication of this run-time observation is that, having evaluated the probability of rearranging a node, the next likelihood evaluation can be started using the same tree as was used to evaluate the previous one, with a high probability that this tree will really be the current best-tree. In this way, the task dependencies in the local optimization step are avoided.
The maximum likelihood algorithm itself was not modified from a functional point of view, but we have incorporated two additional options that seems to be useful when analysing large datasets: (i) saving the best-tree in each iteration and (ii) a checkpoint which allows the program to be stopped at a given step and then the execution re-started again.

Extensive tests were performed using either different sequence sizes or different numbers of sequences (for a detailed revision, see Ceron et al., 1998). The parallel code, written in C with PVM calls to program all interprocess communications, is very portable. The results obtained show that Felsenstein’s DNAml program parallelizes and scales very well when the number of processors rises in a wide range of multiprocessor environments. In fact, five specific parallel platforms were tested: SGI-Origin-2000, SGI-Power Challenge R10000, Cray T3D, Transtech Paramid and a LAN-based cluster of workstations.

Additionally, as a representative case of the computational ability of the proposed strategy to deal with large data sets, we present here a test using a complete set of aligned sequences of small subunit rRNA (SSU rRNA) obtained from the database RDP (http://rdpwww.life.uiuc.edu). This set is formed by 437 sequences, each having 4037 bases. The following test, using 70, 100, 120 and 147 sequences, has been performed and the CPU time needed to complete the run for each group was 121, 357, 601 and 1302 min, respectively, in a SGI-Power Challenge R10000 with up to eight nodes. As should be noticed, these CPU time requirements allow the routine use of the DNAml parallel version for large sets of sequences. Results from the biological side are perfectly consistent with the RDP classification (see the ftp site provided above for details).

**Acknowledgements**

This work has been partly supported by grants TIC96.1125.C03.01 and BIO95-0768 from the CICYT, together with a special grant from the Spanish DIGICYT for the promotion of University-Research Council Associated Units.

**References**


---

![Fig. 1. Run-time behavior of the Felsenstein DNAml algorithm in the local optimization step using a set of 50 sequences (30 residues length). The number of circles in each horizontal line represents the number of tasks performed as a function of the number of sequences (indicated in the left-hand column) already incorporated into the topology. Filled circles represent those points in which a new maximum value was detected, incurring a parallel-processing related penalty. On the right-hand side, a percentual representation of the number of penalties over the overall computations is shown. Indeed, it is worthwhile to highlight that, over 5732 evaluations, only 105 (1.83% of the total) have ended in producing a new best-tree event. In the bottom part of the figure, we present the same type of analysis using several groups of sequences (indicated in the first column). The estimated value (second column) means the number of tasks for the case in which a best-tree event does not occur. The next column represents the number of evaluations performed in the real case (when best-tree events appear). The fourth column shows how many times the best-tree event occurs and in the last column, it can be observed that the percentage of this event occurring decreases when the number of sequences increases, which allows the prediction that a strategy based on the assumption that the best-tree event occurs is minimum will be especially suitable for large groups of sequences.](image_url)