



QuickJoin—fast neighbour-joining tree reconstruction

Thomas Mailund* and Christian N. S. Pedersen

Bioinformatics Research Center (BIRC), University of Aarhus, Høegh-Guldbergs
Grade 10, Bldg. 090, DK-8000 Århus C, Denmark

Received on March 5, 2004; revised on May 19, 2004; accepted on June 1, 2004
Advance Access publication June 16, 2004

ABSTRACT

Summary: We have built a tool for fast construction of very large phylogenetic trees. The tool uses heuristics for speeding up the neighbour-joining algorithm—while still constructing the same tree as the original neighbour-joining algorithm—making it possible to construct trees for 8000 species in <10 min on a single desktop PC. In comparison, the same task takes more than 30 min using the QuickTree neighbour-joining implementation.

Availability: The source code for QuickJoin is available from <http://www.birc.dk/Software/QuickJoin> under the GNU Public License (GPL).

Contact: mailund@birc.dk

1 INTRODUCTION

The neighbour-joining method is a distance-based method for constructing evolutionary trees, introduced by Saitou and Nei (1987), and later improved by Studier and Keppler (1988). It has become a mainstay of phylogeny reconstruction, and is probably the most widely used distance-based algorithm in practice. Empirical work shows it to be quite accurate, at least for cases where the rate of evolution is not extremely high or low, and with a running time of $O(n^3)$ on n taxa (Studier and Keppler, 1988), it is applicable for small to medium taxa sets. For large taxa sets, $n > 1000$, the runtime of the algorithm becomes prohibitive. The need to analyse larger taxa sets is necessary and increasing (Howe *et al.*, 2002), e.g. the Pfam database contains several protein families of more than 8000 sequences (Bateman *et al.*, 2002).

2 ALGORITHM

The neighbour-joining method is a greedy algorithm that attempts to minimize the sum of all branch-lengths on the constructed tree. Conceptually, it starts out with a star-formed tree where each leaf corresponds to a species, and iteratively picks two nodes adjacent to the root and joins them, by inserting a new node between the root and the two selected nodes.

When joining nodes, the method selects the pair of nodes i, j that minimizes the branch-length sum of the resulting new tree. This pair can be found by minimizing the expression $Q_{ij} = (r - 2)d_{ij} - (R_i + R_j)$, where d_{ij} is the distance between node i and j (assumed symmetric, i.e. $d_{ij} = d_{ji}$), R_k is the row sum over row k of the distance matrix: $R_k = \sum_i d_{ik}$ (where i ranges over all nodes adjacent to the root node), and r is the remaining number of nodes adjacent to the root (Studier and Keppler, 1988). When nodes i and j are joined, they are replaced with a new node, A , with distance to the remaining nodes given by $d_{Ak} = (d_{ik} + d_{jk} - d_{ij})/2$. The method performs a search for $\min Q_{ij}$, using time $O(r^2)$, and joins i and j , using time $O(r)$ to update d . This search and join is continued until only three nodes are adjacent to the root (i.e. for $n - 3$ joins), giving a total time complexity of $O(n^3)$, and a space complexity of $O(n^2)$.

We have developed an algorithm that uses a number of heuristics to speed up the search for $\min Q_{ij}$ while still being exact, that is the search always finds the minimal value, but does not always examine all possible values of Q_{ij} when some values are known to be non-minimal. The basic idea in the algorithm is to build a quad-tree of a set of linear functions that acts as lower bounds for sets of Q_{ij} values, and prune the search for a minimal Q_{ij} when the lower bounds are higher than some known Q_{ij} value. A quad-tree is a four-ary-tree modelling a two-dimensional area, e.g. a matrix, recursively into quadrants (Finkel and Bentley, 1974). By building a quad-tree of height $\log r$ on top of the $r \times r$ matrix Q and storing in the nodes of the quad-tree the minimal values in the subtree rooted at that node, we can search for the $\min Q_{ij}$ in time $O(\log r)$, compared with time $O(r^2)$ by the naive approach. Unfortunately, since Q changes globally in each iteration, it would be necessary to rebuild the quad-tree, taking time $O(r^2)$ each time, leading to no overall improvement of the running time. To improve the total running time, we must avoid rebuilding the quad-tree in each iteration.

In Brodal *et al.* (2003), we show that we can rewrite Q_{ij} as a linear function f_{ij} over r plus some correction terms c_i and c_j , where f_{ij} only depends on d_{ij} . By building a quad-tree based on top of the matrix f —where each internal node stores

*To whom correspondence should be addressed.

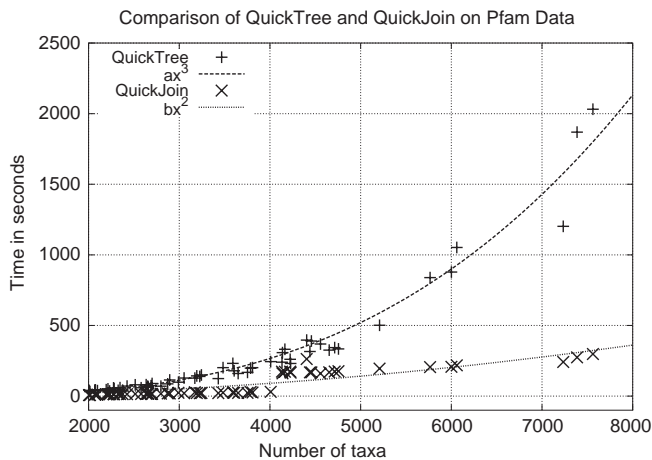


Fig. 1. The running time of QuickJoin on distance matrices obtained from the Pfam collection is compared with the running time of Quick Tree. An estimation of the constants yields $a = 4.16 \times 10^{-9}$ and $b = 5.64 \times 10^{-6}$.

a lower bound for all the linear functions stored at the leaves of the subtree rooted at the node—we can speed up the search for $\min Q_{ij}$ by skipping subtrees rooted at internal nodes where the lower bound is higher than some known Q_{ij} value. The stored lower bounds becomes less tight as the algorithm evolves which increase the search time. To remedy this, we can rebuild the quad-tree, implying a trade-off between search time and rebuild frequency. By choosing only to rebuild when we have examined at least r^2 nodes in the quad-tree during searching, we avoid using more time on rebuilding than on searching. The worst case time complexity remains $O(n^3)$, but experiments show that the developed algorithm on real data is significantly faster (for further details see Brodal *et al.*, 2003).

We have evaluated the performance of our algorithm empirically on distance matrices obtained from the Pfam collection of alignments (Bateman *et al.*, 2002), and compare the running time with that of the QuickTree tool (Howe *et al.*, 2002), a well-known and widely used implementation of the standard neighbour-joining method. The results in Figure 1 show

that the our algorithm yields a significant speed-up over the standard neighbour-joining method, already for moderately sized instances. Indeed, experiments indicate that the running time evolves as $\Theta(n^2)$ on the examined instance collection, as opposed to $\Theta(n^3)$ for QuickTree. The space complexity after adding the quad-tree is still $O(n^2)$ since a quad-tree with n^2 leaves can be represented in $O(n^2)$ space.

3 IMPLEMENTATION

QuickJoin is an implementation of the above algorithm in C++ which should compile on any platform supporting autoconf, make and gcc. It has been tested on various versions of Linux, Solaris, IRIX and OS X.

The program takes as input either a multiple alignment—in Stockholm format—or a distance matrix—in PHYLIP format—and outputs the constructed tree—in Newick format. If the input is provided as a multiple alignment, QuickJoin can perform bootstrap validation of the constructed tree, and the bootstrap values for the individual edges are outputted as annotations on the outputted tree. Use the `--help` option for more information.

REFERENCES

- Bateman,A., Birney,E., Cerruti,L., Durbin,R., Etwiller,L., Eddy,S., Griffiths-Jones,S., Howe,K., Marshall,M. and Sonnhammer,E. (2002) The Pfam protein families database. *Nucleic Acids Res.*, **30**, 276–280.
- Brodal,G., Fagerberg,R., Mailund,T., Pedersen,C. and Phillips,D. (2003) Speeding up neighbour-joining tree construction. *Technical Report ALCOMFT-TR-03-102*, ALCOM-FT.
- Finkel,R.A. and Bentley,J.L. (1974) Quad trees: a data structure for retrieval by composite key. *Acta Inform.*, **4**, 1–9.
- Howe,K., Bateman,A. and Durbin,R. (2002) QuickTree: building huge neighbour-joining trees of protein sequences. *Bioinformatics*, **18**, 1546–1547.
- Saitou,N. and Nei,M. (1987) The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Mol. Biol. Evol.*, **4**, 406–425.
- Studier,J.A. and Keppler,K.J. (1988) A note on the neighbor-joining method of Saitou and Nei. *Mol. Biol. Evol.*, **5**, 729–731.