**ABSTRACT**

Summary: QDist is a program for computing the quartet distance between two unrooted trees, i.e. the number of quartet topology differences between the trees, where a quartet topology is the topological subtree induced by four species. The program is based on an algorithm with running time $O(n \log^2 n)$, which makes it practical to compare large trees. Available under GNU license.

Availability: [http://www.birc.dk/Software/QDist](http://www.birc.dk/Software/QDist)

Contact: mailund@birc.dk

**1 INTRODUCTION**

The evolutionary relationship for a set of species can be described by a rooted tree where leaves correspond to the species, and the internal nodes correspond to speciation events. The direction of the evolution is described by the location of the root, which corresponds to the most recent common ancestor for all the species, and the rate of evolution is described by assigning lengths to the edges. Estimating aspects of the evolutionary tree from obtainable information about the species, e.g. genomic data, is a widely studied problem see Gusfield (1997), Chapter 17. Many methods are only concerned with estimating the undirected tree topology induced by ignoring the location of the root and the length of the edges, usually under the further assumption that all internal nodes have degree 3. For the remainder of this paper, an evolutionary tree denotes such an unrooted evolutionary tree of degree 3.

Different methods usually yield different evolutionary trees for the same set of species, and the same method can yield different evolutionary trees for the same set of species when applied to different information about the species, e.g. different genes. To study such differences in a systematic manner, one must be able to quantify differences between trees by well-defined and efficient methods.

One approach is to compute the distance between two trees based on a well-defined distance measure. Several distance measures have been proposed, each with different properties that reflect different aspects of biology. Bryant et al. (2000) discuss different distance measures and conclude that the quartet distance (Estabrook et al., 1985) has several attractive properties. Compared with the split distance, one can say that the quartet distance penalizes edges individually, depending on the structure of the subtrees connected by the edge. Here, we present an implementation of the algorithm in Brodal et al. (2001) for computing the quartet distance between two trees of $n$ species in time $O(n \log^2 n)$ and space $O(n)$.

**2 ALGORITHM**

The algorithm counts the number of quartet topology differences between two trees $T_1$ and $T_2$ of $n$ species by counting the number of shared quartets in $T_1$ and $T_2$ and subtracting this from the total number of $\binom{n}{4}$ possible quartets. To count shared quartets, the algorithm associates a quartet $a b c d$ to the unique node in $T_1$ where the paths from $a$ and $b$ to $c$ join. In a recursive traversal of $T_1$, it counts for each internal node in $T_1$ the number of associated quartets that are also quartets in $T_2$. To implement the counting efficiently, the algorithm employs a data structure where the main part is a hierarchical decomposition of $T_2$. This is a rooted binary tree with height $O(\log n)$. Each node in the hierarchical decomposition stores a triplet of integers and a polynomial of at most nine variables with total degree at most 4. Counting is performed by manipulation of the polynomials along leaf-to-root paths [see Brodal et al. (2001) for details].

**3 IMPLEMENTATION**

QDist is an implementation of the above algorithm in C++ that 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 qdist takes as input two trees (over the same set of species) in Newick format and outputs the quartet distance between them in a format determined by the --verbose option. Use the --help option for more information.

Our initial implementation of QDist followed closely the description in Brodal et al. (2001). Profiling showed, as suspected, that most of the running time was spent manipulating...
polynomials. Further experiments showed that many identical manipulations were done several times. To remedy this situation, we extended the implementation by storing the result of each manipulation when performed, allowing us to look up the result of an already performed manipulation instead of redoing it.

Figure 1 shows the running time in seconds for comparing trees of increasing size with and without caching the manipulations performed. For time comparison of smaller trees, we have performed 1000 comparisons of randomly chosen trees of size 50. The total running time was 547.1 s with caching and 1635.3 s without caching. Even though the worst-case running time of a manipulation in both cases is constant, it follows that caching the manipulations performed improves the running time substantially in practice. The downside to caching is that the worst case space consumption increases to $O(n \log n)$, but this should not affect the applicability of the algorithm in practice. Caching can be disabled using the --no-cache option. Both implementations, with and without caching, have been tested successfully on a large set of trees for which the pairwise distances were known in advance.

REFERENCES


