

An $O(n^2t + t^2)$ algorithm for computing the symmetric distances between t trees of size n

Thomas Mailund

January 22, 2007

I describe an algorithm for computing all pairwise symmetric distances (or related distance measures such as Robinson-Foulds distance²) in time $O(n^2t + t^2)$ for t trees of size n . The t^2 term is from outputting the distance matrix and can be improved if different output is needed.

Keywords: Tree comparison; symmetric distance; all pairwise distances

Introduction

For a leaf-labelled tree, T , a split is a pair of sets, $A|B$, such that $A \cup B$ is the set of leaves of T , $A \cap B = \emptyset$ and such that T contains an edge with A being the set of leaves on one side of the edge and B being the set of leaves on the other side. Given two trees, T and T' , with the same leaf-set, let $S(T)$ be the set of splits of T and $S(T')$ be the set of splits in T' . The symmetric distance between trees T and T' is then defined to be $\text{symdiff}(T, T') := |S(T) \setminus S(T')| + |S(T') \setminus S(T)|$. Related measures are the Robinson-Foulds distance²—that is simply the symmetric distance divided by two—and the split similarity that is $\text{splitsim}(T, T') := |S(T) \cap S(T')|$. For binary trees, the number of edges is always $n - 3$ and $\text{symdiff}(T, T') = (n - 3) - \text{splitsim}(T, T')$.

By sorting the leaf-labels and fixing an arbitrary leaf-label as the reference, each split in a tree has a unique bit-vector representation. A simple way of calculating the symmetric distance or split similarity is then to build a table of bit-vectors for T and for each split in T' test whether the split is in the table or not. If the table is a hash-table, the expected time to insert an element and to test for the inclusion of an element is proportional to calculating the hash-key and comparing two values. For bit-vector representations of splits, these can be (and typically are) done in time $O(n)$ —since the length of the bit-vectors are exactly n —and with $O(n)$ such splits per tree, the total time for calculating the symmetric distance with this approach is $O(n^2)$. The theoretical lower bound for calculating the symmetric distance is, of course $\Omega(n)$ and this is achieved using Day's algorithm¹.

When computing all pairs of distances between t trees, the direct application of the bit-vector approach would give an $O(n^2t^2)$ algorithm. The RF-hashing approach in Sul and Williams³ also runs in worst case $O(n^2t^2)$ but is analysed to run in expected time $O(nt \log nt + t^2)$ (although I am not completely convinced that there is not a factor of n missing from this analysis as they seem to assume they can compare two bit-vectors in constant time). In comparison, the direct application of t^2 applications of Day's algorithm would give a worst case

running time of $O(nt^2)$. In the following I present an $O(n^2t + \text{output})$ algorithm for the same problem, where if the output is the similarity or distance matrix is of order $O(t^2)$ but can be smaller if we just report which trees share splits.

A suffix-tree based algorithm

The algorithm is based on representing the set of all splits as a string and then building a suffix-tree over this string, from which shared splits can be read.

Building this string is straightforward: We start with an empty string, where, in the actual implementation we pre-allocate sufficient memory that we do not need to use time on reallocating during the algorithm. To this string we then append, for each tree and each split, the bit-vector representation of the split followed by a character encoding a pair (i, n) where i is the tree id and n is a nonce, a unique symbol found nowhere else, that is used ensure that all splits end in leaves in the suffix tree. The use of the special terminator symbols makes the alphabet size of the suffix tree non-constant, but since these symbols are only used to ensure that certain strings are leaves, they can essentially be ignored in the suffix-tree construction algorithm and will not affect the linear time construction time.

With t trees of size n there are nt splits than combined results in a string of length $O(n^2t)$ constructable in time $O(n^2t)$. Using a linear time suffix-tree construction algorithm, we can thus obtain of suffix-tree representation of the set of all splits in time $O(n^2t)$. In this tree, shared splits are found as leaves having of the same inner node, starting with a special symbol. We can traverse the tree and read off these in time $O(n^2t)$.

If we wish to explicitly enumerate splits and the trees sharing them, we need to print the individual splits, in time $O(n)$ for each, followed by the tree ids. Since the total number of splits is $O(nt)$ we cannot report more than that, so explicitly reporting the shared splits can be done in $O(n^2t)$.

If we wish to report the symmetric distance matrix or the split similarity matrix for all pairs of trees, we can calculate both of these by traversing the tree and then output the matrix in time $O(t^2)$.

References

1. W. Day. Optimal algorithms for comparing trees with labeled leaves. *Journal of Classification*, 2:7–28, 1985.
2. D. F. Robinson and L. R. Foulds. Comparison of phylogenetic trees. *Mathematical Biosciences*, 53:131–147, 1981.
3. S.-J. Sul and T. L. Williams. A randomized algorithm for comparing sets of phylogenetic trees. In *Proceedings of the Asia-Pacific Bioinformatics (APBC) 2007*, 2007.