A Randomized Linear-time Majority Tree Algorithm

Nina Amenta [*] Frederick Clarke [†] Katherine St. John [†]

Abstract

We give a randomized linear-time algorithm for computing the majority tree, a technique widely used for summarizing sets of phylogenetic trees. We are implementing the algorithm as part of an interactive visualization system for analyzing large sets of trees.

(**Keywords:** phylogeny, majority consensus, visualization)

With the recent explosion in the amount of genomic data available, and exponential increases in computing power, biologists are now able to consider larger scale problems in phylogeny, such as the construction of evolutionary trees on hundreds or thousands of taxa, and ultimately of the entire "Tree of Life" which would include millions of taxa. One difficulty is that most programs used for phylogeny reconstruction [6, 7, 11] are based upon heuristics for NP-hard optimization problems which generally output hundreds or thousands of likely candidates for the optimal tree, instead of producing a single optimal tree. This large volume of data is usually summarized with a consensus tree.

A consensus tree, for a set of input trees, is a single tree which includes features on which all or most of the input trees agree. The simplest is the *strict consensus tree*, which includes only subtrees that appear in all of the input trees. The *majority tree*, or M_l tree, includes nodes for exactly those subtrees which occur in more than half of the input trees, or more generally in more than some fraction l of the input trees (see Figure 1). Margush and McMorris [9] showed that this set of bipartitions does indeed constitute a tree for any $1/2 < l \leq 1$. McMorris, Meronk and Neumann [10] called this family of trees the M_l trees (e.g. the M_1 tree is the strict consensus tree); we shall call them all generically majority trees. While our example is for rooted binary trees, this can also be applied to non-binary and unrooted trees. The majority tree is interesting for a much broader range of inputs than the strict consensus tree (see [2], §6.2, for an excellent overview).

We have developed a randomized algorithm to compute the majority tree of a set of input trees, where the expected running time is linear both in the number t of trees *and* in the number n of taxa. Earlier algorithms were quadratic in n, which is problematic for larger phylogenies. Our $O(tn)$ expected running time is optimal, since just reading a set of t trees on n taxa requires $\Omega(tn)$ time. The expectation in the running time is over random choices made during the course of the algorithm, independent of the input; thus, on any input, the running time is linear with high probability.

Having an algorithm which is efficient in t is essential, and most earlier algorithms focus on this. Large sets of trees arise given any kind of input data on the taxa (e.g. gene sequence, gene order, character) and whatever optimization criterion is used to select the "best" tree. The heuristic searches used for maximizing parsimony often return large sets of trees with equal parsimony scores. Maximum likelihood estimation, also computationally hard, generally produces trees with unique scores. While technically one of these is the optimal tree, there are many others for which the likelihood is only negligibly sub-optimal. The output of the computation is again more accurately represented by a consensus tree.

As the number of taxa which can be handled increases into the thousands, having an algorithm which is linear in n is also becoming important, especially for interactive software. We were motivated to find an efficient algorithm for the majority tree because we wanted to compute it on-the-fly in an interactive visualization application [1]. Figure 2 shows a screen shot. The window on the left shows a representation of the distribution of trees, where each point corresponds to a tree. The user interactively selects subsets of trees and, in response, the consensus tree of the subset is computed on-the-fly and displayed. This package is built as a module within Mesquite [8], a framework for phylogenetic computation by Wayne and David Maddison. Our original version of the visualization system

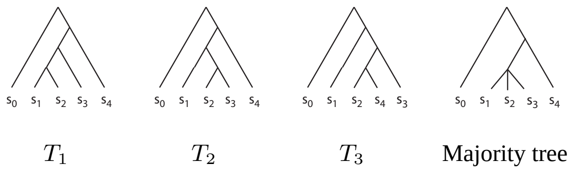
s_0 s_1 s_2 s_3 s_4
s_0 s_1 s_2 s_3 s_4
s_0 s_1 s_2 s_4 s_3
s_0 s_1 s_2 s_3 s_4
T_1
T_2
T_3
Majority tree

Figure 1: Three rooted input trees and their majority tree (for a 50 percent majority). The input trees need not be binary.

[*]`amenta@cs.ucdavis.edu`. Computer Science Department, University of California, Davis, CA 95616. Supported by an Alfred P. Sloan Foundation Research Fellowship and NSF ITR DEB-0121651.

[†]`fclarke72@aol.com`, `stjohn@lehman.cuny.edu`. Math & Computer Science, Lehman College– City University of New York, Bronx, NY 10468. Supported by NSF ITR DEB-0121682.

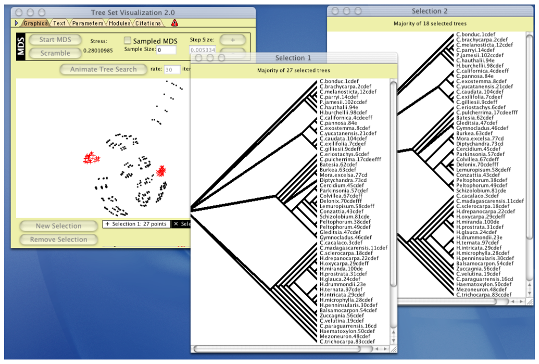

Figure 2: The tree visualization module in Mesquite. The window on the left shows a projection of the distribution of trees. The user interactively selects subsets of trees with the mouse, and, in response, the consensus tree of the subset is computed on-the-fly and displayed in the window on the right. Two selected subsets and their majority trees are shown.

computed only strict consensus trees. We found in our prototype implementation that a simple $O(tn^2)$ algorithm for the strict consensus tree was unacceptably slow, and we implemented instead the $O(tn)$ strict consensus algorithm of Day [5]. This inspired our search for a linear-time majority tree algorithm.

Prior Work: For the strict consensus tree, Day's deterministic algorithm uses a clever $O(1)$ representation for subtrees, and also achieves an optimal $O(tn)$ running time, which does not generalize easily to other M_l trees. Wareham [12], developed an $O(n^2 + t^2 n)$ algorithm for the M_l tree, which only uses $O(n)$ space. It uses Day's data structure to test each subtree encountered against all of the other input trees. Another algorithm for majority trees is implemented in PHYLIP [6] by Felsenstein *et al.*. The overall running time as implemented seems to be $O(tn^2/(\lg tn) + tn \lg(tn) + n^3/(\lg tn))$. Majority trees are also computed by PAUP [11], using an unknown (to us) algorithm.

Outline of Algorithm: A linear-time majority tree algorithm has been somewhat elusive. Our algorithm has two stages, similar to past algorithms. In the first stage, we read through the input trees and count the occurrences of each subtree, storing the counts in a hash table. Then, in the second stage, we create nodes for the subtrees that occur in a majority of input trees - the *majority subtrees* - and "hook them" together into a tree. We sketch the algorithm briefly below; more details and proofs will appear in the full paper.

We achieve the linear running time by reducing the space and time used for counting the common subtrees. The natural bit-string representation for a subtree has size $O(n/w)$, where w, the number of bits in a word, is usually taken to be $O(\lg tn)$. We introduce a representation of size $O((\lg tn)/w)$, which gives $O(1)$. The representation of a subtree is its constant-size hash code. We use a specific *universal hash function* for which we can compute the hash code for a node in constant time, given the hash codes for its children, so that we can compute hash codes bottom-up for all of the $O(tn)$ subtrees in $O(tn)$ time.

We also give an $O(tn)$ algorithm for hooking together the majority nodes. Hooking together the nodes is no longer trivial since we do not have explict representations of the majority subtrees that occur in more lt trees. We again traverse the set of input trees, keeping track of the ancestor-descendant relationships of the majority subtrees encountered. If p is the parent of a subtree s in the majority tree, the Pigeonhole Principal implies that there is at least one input tree in which p is an ancestor of s; we can recognize it because p will be the ancestor of s of minimum cardinality observed during the traversal.

The hash function is randomized, so there is a small possibility that collisions in the hash table may cause the algorithm to fail. We detect this event, and repeat the process with new random choices. The expected number of attempts at building the tree is less than two, so our expected running time is linear in both the number of trees and the number of taxa.

Implementation Our majority tree algorithm is being implemented within Mesquite [8], a framework for phylogenetic analysis written by Wayne and David Maddison. Mesquite is designed to be portable and extensible. It is written in Java and runs on a variety of operating systems (Linux, MacIntosh OS 9 and X, and Windows). Mesquite is made up of cooperating modules. The first version of the module for our visualization system, TreeSetVisualization, can be downloaded from our webpage [1]. The module (including only the strict consensus) was introduced this summer at Evolution 2002 and has since been downloaded by hundreds of researchers. In the communications we get from users, majority trees are frequently requested.

We are implementing the majority tree algorithm as part of the next version. Figure 2 shows our current prototype. The speed of the majority tree function seems comparable to our linear-time strict consensus tree implementation; our final paper will give comparisons.

Acknowledgments: We thank Jeff Klingner for the tree set visualization module and Wayne and David Maddison for Mesquite, and for encouraging us to consider the majority tree.

References

[1] Nina Amenta and Jeff Klingner. Case study: Visualizing sets of evolutionary trees. In *8th IEEE Symposium on Information Visualization (InfoVis 2002)*, pages 71–74, 2002. Software available at `www.cs.utexas.edu/users/phylo/`.

[2] David Bryant. *Hunting for trees, building trees and comparing trees: theory and method in phylogenetic analysis*. PhD thesis, Dept. of Mathematics, University of Canterbury, 1997.

[3] J. Lawrence Carter and Mark N. Wegman. Universal classes of hash functions. *Journal of Computer and Systems Sciences*, 18(2):143–154, 1979.

[4] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to algorithms*. MIT Press, Cambridge, MA, second edition, 2001.

[5] William H.E. Day. Optimal algorithms for comparing trees with labeled leaves. *J. Classification*, 2(1):7–28, 1985.

[6] J. Felsenstein. Phylip (phylogeny inference package) version 3.6, 2002. Distributed by the author. Department of Genetics, University of Washington, Seattle. The consensus tree code is in `consense.c` and is co-authored by Hisashi Horino, Akiko Fuseki, Sean Lamont and Andrew Keeffe.

[7] John P. Huelsenbeck and Fredrik Ronquist. Mrbayes: Bayesian inference of phylogeny, 2001.

[8] W.P. Maddison and D.R. Maddison. Mesquite: a modular system for evolutionary analysis. version 0.992, 2002. Available from `http://mesquiteproject.org`.

[9] T. Margush and F.R. McMorris. Consensus n-trees. *Bulletin of Mathematical Biology*, 43:239–244, 1981.

[10] F.R. McMorris, D.B. Meronk, and D.A. Neumann. A view of some consensus methods for trees. In *Numerical Taxonomy: Proceedings of the NATO Advanced Study Institute on Numerical Taxonomy*. Springer-Verlag, 1983.

[11] D.L. Swofford. *PAUP*. Phylogenetic Analysis Using Parsimony (*and Other Methods). Version 4*. Sinauer Associates, Sunderland, Massachusetts, 2002.

[12] H. Todd Wareham. An efficient algorithm for computing M_l consensus trees, 1985. BS honors thesis, CS, Memorial University Newfoundland.