An Overview of Three Main Consensus Tree Algorithms for Phylogenetic Trees

Matthew Smith-Erb

December 1st, 2021

1 Introduction

In biology, a phylogenetic tree is a type of tree, rooted or unrooted, in which its leaves represent living species. An internal vertex represents an extinct species which is a common ancestor to its descendants. For my final, I researched various algorithms which construct *consensus trees*. A consensus tree is a phylogenetic tree which attempts to summarize the topological structure of several inputted phylogenetic trees, all of which have the same set of leaves. The set of species which labels the leaves are also known as the *taxa* of the tree.

One may wonder why there would exist phylogenetic trees with the same taxa but conflicting internal structure. Though there are different methods to construct phylogenetic trees, they all rely on collecting some type of data or metric on each taxon which is used to identify similarities amongst other taxa. Recent studies have shown that these differences in types of data are a strong explanation for the different topologies that phylogenetic trees can have. In fact, in a study done by Reddy et al[8], results suggested that the differences in types of experimental data had a greater effect on topologies than the amount of data collected on each taxon. Thus, different types of experiments can yield different trees, and so a consensus tree acts as a way to represent these conflicts and agreements.

2 Adams Consensus Tree

The idea behind a consensus tree was first proposed by Edward Adams in a paper he published in 1972 [1]. The time complexity of the first implementation of the Adams consensus tree was $O(tn^2)$, where t is the number of input trees and n is the number of taxa. However, the algorithm has since been improved to run in $O(tn\log(n))$ [6]. A limitation to the Adams consensus tree algorithm is that it only works on rooted phylogenetic trees. Stated broadly, the algorithm works recursively by combining smaller Adams trees which take inputs as restricted versions of the original input trees.

To further elaborate, we will introduce new terms and notation. First, we will use the notation T^u to be the subtree of T rooted at vertex u and containing only the descendants of u. Next, $\Lambda(T)$ is the set of leafs in tree T. An important term used in the Adams algorithm is the partition of tree T, denoted $\pi(T)$, which is defined by $\pi(T) = \{\Lambda(T^c) : \text{c is a child of the root of T}\}$. Thus, in a phylogenetic tree where the root has two children, the partition will contain two blocks, one block containing the taxa that are descendants of the root's left child and one block containing the taxa that are descendants of the root's right child. The product of the partitions of a set of trees is a partition where any two taxa a and b are in the same block if and only if they are in the same block in the partition of each input tree. Lastly, the restriction of tree T on $X \subseteq \Lambda(T)$, is the tree consisting only of the paths from the root of T to the taxa in X, discarding all unnecessary paths to taxa not in X.

The Adams algorithm first checks in a base case if the input trees contain one leaf, returning this simple tree. In the recursive case, the product of the partitions of the input trees is calculated. For each block in the product of partitions, each input tree is restricted to that block and an Adams tree is constructed for this set of restricted input trees. Lastly, it creates and returns a tree whose root points to the root of each of these smaller Adams trees.

The Adams algorithm is useful in summarizing input trees because it retains the nesting structure common to each input tree and it does not introduce new rooted triplets. A rooted triplet, $ab|c$, is when the lowest common ancestor (LCA) of a and b is a proper descendant of the LCA of a, b, and c. These two features come from two properties which always hold: (i) for any two subsets A, B of the leaf set L, if the LCA of A is a proper descendant of the LCA of B for every input tree, then the LCA of A is a proper descendant of the LCA of B in the Adams tree T, (ii) for any two

vertices u, v in the Adams tree T, if u is a proper descendant of v, then in every input tree, the LCA of $\Lambda(T^u)$ is a proper descendant of the LCA of $\Lambda(T^v)$ [6].

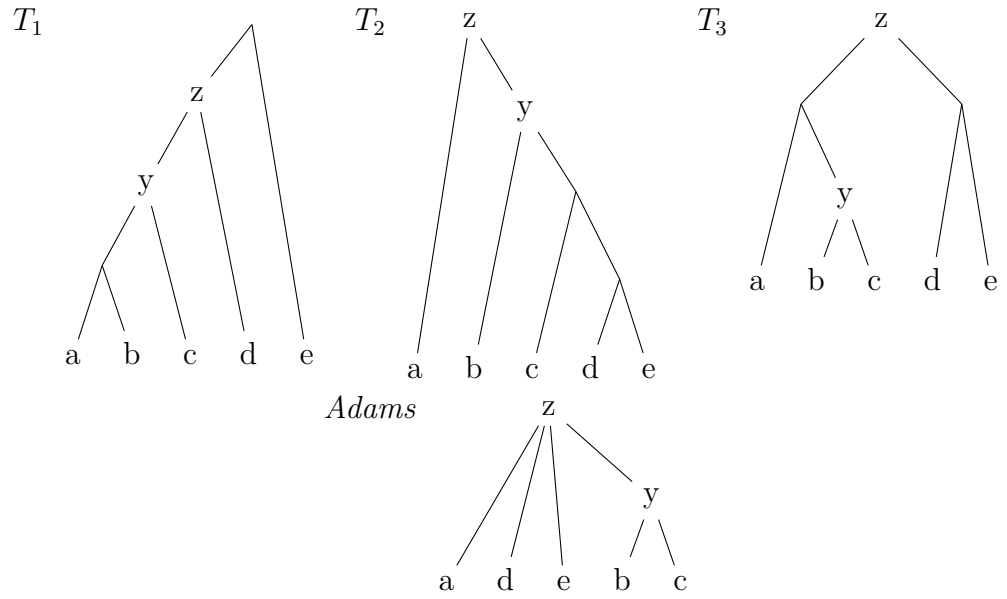


Figure 1: To demonstrate property (i) of Adams, consider the sets of taxa $Y = \{b, c\}$, $Z = \{a, d\}$. In each input tree, vertex y is the LCA of Y which is a proper descendent of vertex z, the LCA of Z. Thus, y is a proper descendent of Z is the Adams tree.

3 Strict Consensus Tree

Two drawbacks to the Adams consensus tree were its relatively poor runtime and its inability to work on unrooted trees. Published in 1985, William Days' strict consensus tree algorithm addressed both of these issues by working on rooted and unrooted trees in a time complexity of $O(tn)$ [5]. In a phylogenetic tree, a subset C of taxa is a *cluster* if and only if the descendant leaves of the LCA of C is equivalent to the set C. So, each vertex in a rooted tree can be identified by the unique cluster which contains leaves that are descendant leaves of the vertex. A strict consensus tree contains exactly those clusters common to every input tree [3].

Although a strict consensus tree is relatively straightforward to construct

and conceptualize, it does a worse job representing the agreements amongst the input trees. For example, if the cluster $\{a, b\}$ is in 9 out of 10 input trees, the cluster will not be present in the strict consensus tree.

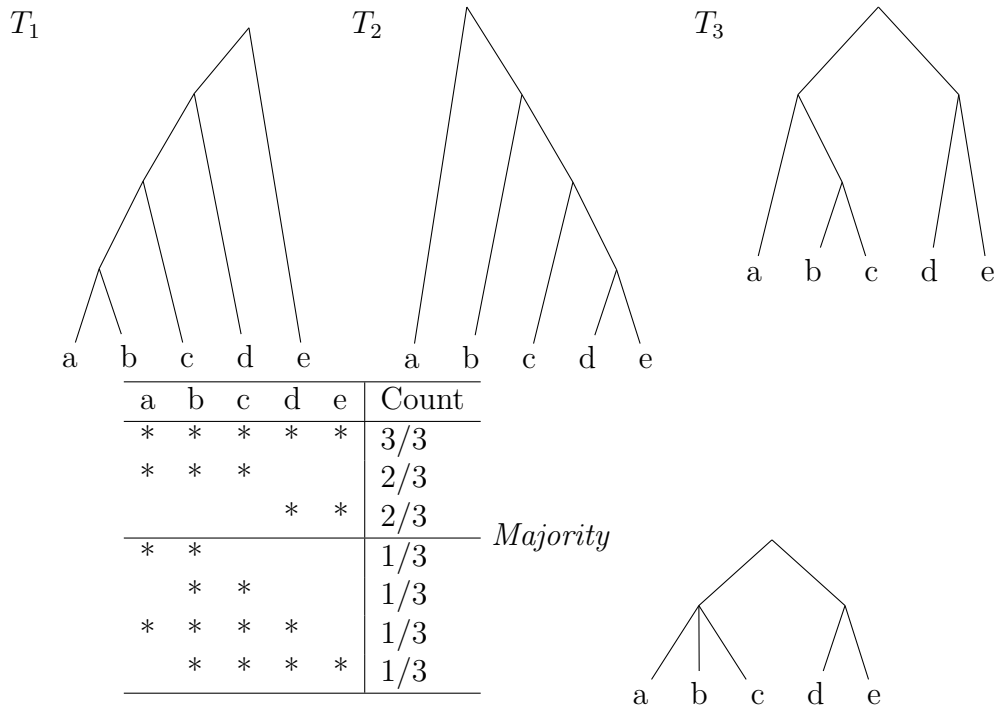
4 Majority Consensus Tree

A majority consensus tree, published by T. Margush and F.R. McMorris in 1981, contains the clusters present in over half of the input trees[7]. Therefore, given a set of input trees, every cluster in its strict consensus tree is also a cluster in the majority consensus tree. So, we can think of a majority tree as a refinement of the strict consensus tree.

Unlike Adams trees, a majority consensus tree is effected when there are multiple copies of a particular input tree. In this sense, it may better represent the agreements of clusters in the input trees. A majority consensus tree minimizes its symmetric differences with all of the input trees. The symmetric difference between two trees, notated $d(T_1, T_2)$, is the number of clusters present in one tree but not the other. For input trees $T_1, T_2, ...T_k$, the majority tree T minimizes $\sum_{i=1}^{k} d(T, T_i)$.

Although the majority tree was first proposed in 1981, a linear run time of $O(tn)$ was not discovered until 2003. This implementation, published by N. Amenta, F. Clarke, and K. St. John, utilizes universal hash functions to efficiently store and hash the many possible clusters that are present in the input trees[2].

In the algorithm, we first do a postorder traversal of each input tree. At a leaf, we calculate two universal hash codes, h_1 which will be its address in the hash table, and h_2, which will represent its unique ID to prevent double collisions. At an internal vertex, each of its children has disjoint clusters whose union is the cluster at this vertex. Because of this fact, we can simply calculate each hash code by summing its childrens' respective hash codes (and then modulo the prime used in the hash function). Now, we increment the value stored at h_1 with ID h_2. In subsequent traversals of input trees, if we arrive at a vertex with a cluster present in a past input tree, we will calculate the same hash codes and keep a running total in the hash table of the number of appearances the cluster has made.



a	b	c	d	e	Count
*	*	*	*	*	3/3
*	*	*			2/3
			*	*	2/3
*	*				1/3
	*	*			1/3
*	*	*	*		1/3
	*	*	*	*	1/3

Figure 2: Each row in the table represents a cluster present in the set of input trees. The columns with * indicate that the cluster contains that respective leaf. The clusters that appear in the majority of input trees are present in the majority tree.

Once we have completed counting the appearances of clusters in each input tree, we construct the majority tree by doing a preorder traversal of each input tree. Through the recursive preorder traversal calls, we pass down a variable c which points to the most recent vertex in the tree that had a majority cluster. For the traversal of each tree, it is initialized to be the tree's root.

Now, we use the hashes at the given vertex to check the hash table to determine if its cluster appears in a majority of trees. If the cluster is in a majority of trees and does not yet exist in the partially constructed majority tree, add the vertex to the majority tree with a parent of c. If the majority cluster does already exist in the majority tree, check if the number of appearances of c is less than the number of appearances of the parent of the majority cluster in the majority tree. If this is the case, switch the parent of the cluster to be c. Once the tree is constructed, we must do one final pass

through to make corrections in the event that two different clusters with the same number of occurrences randomly had the same hash codes.

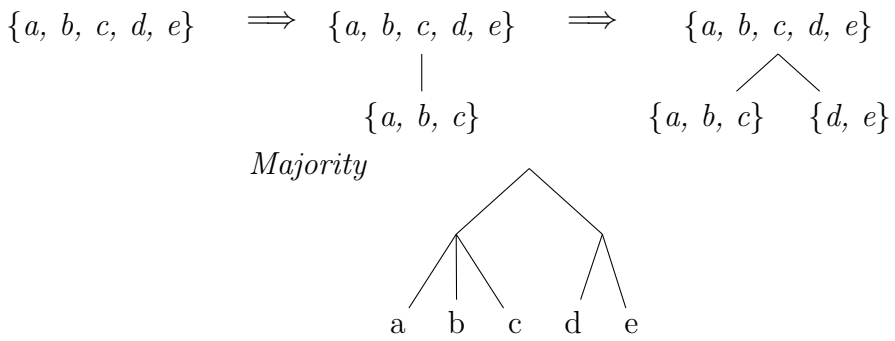


Figure 3: This shows the step-by-step process of adding majority clusters to the majority tree. Each cluster corresponds to a unique vertex so we can convert the tree with clusters as vertex labels to a typical phylogenetic tree.

When constructing the tree, we checked if a cluster was in over half of the input trees. This however, can easily be modified to see if more than some fraction l, between $\frac{1}{2}$ and 1, of the trees have the cluster. We call this family of consensus trees M_l trees. As such, we can use this algorithm to construct a strict consensus tree, which is an M_1 tree[2].

5 Discussion

The implementations of the Adams and M_l consensus tree algorithms done for this final project were both written in Python. The open source library Biopython[4] was used for its useful phylogenetic classes and functions. It should be noted that the theoretical run time of $O(tn^2)$ is not achieved in the Adams implementation because of a custom function which restricts the trees to a set of leaves. Also, instead of using hash codes, the M_l implementation uses Python's built-in dictionary to store the counts of clusters, where the key to the dictionary is the cluster at a given node. This eliminates the need to check for possible collisions in hashing functions.

We have explored the qualities of several different consensus tree algorithms, both in terms of their overall summary of inputs and also their complexities. In a time when there is new research coming out constantly, methods for consolidating results across studies is imperative.

References

[1] Adams, E. N. (1972). Consensus techniques and the comparison of taxonomic trees. *Systematic Biology, 21*(4), 390–397. https://doi.org/10.1093/sysbio/21.4.390

[2] Amenta, N., Clarke, F., & St. John, K. (2003). A Linear-time Majority Tree Algorithm. http://comet.lehman.cuny.edu/. Retrieved November 10, 2021, from http://comet.lehman.cuny.edu/treeviz/papers/majority.pdf.

[3] Bryant, D. (2003). A Classification of Consensus Methods for Phylogenetics. *Bioconsensus*, 163–183. https://doi.org/10.1090/dimacs/061/11

[4] Cock, P. J., Antao, T., Chang, J. T., Chapman, B. A., Cox, C. J., Dalke, A., ... others. (2009). Biopython: freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics, 25*(11), 1422–1423.

[5] Day, W. H. (1985). Optimal Algorithms for Comparing Trees with Labeled Leaves. *Journal of Classification, 2*(1), 7–28. https://doi.org/10.1007/bf01908061

[6] Jansson, J., Li, Z., & Sung, W.-K. (2017). On finding the Adams Consensus Tree. *Information and Computation, 256*, 334–347. https://doi.org/10.1016/j.ic.2017.08.002

[7] Margush, & McMorris, F. . (1981). Consensus n-trees. *Bulletin of Mathematical Biology, 43*(2), 239–244. https://doi.org/10.1016/S0092-8240(81)90019-7

[8] Reddy, S., Kimball, R. T., Pandey, A., Hosner, P. A., Braun, M. J., Hackett, S. J., Han, K.-L., Harshman, J., Huddleston, C. J., Kingston, S., Marks, B. D., Miglia, K. J., Moore, W. S., Sheldon, F. H., Witt, C. C., Yuri, T., & Braun, E. L. (2017). Why do phylogenomic data sets yield conflicting trees? data type influences the avian tree of life more than taxon sampling. *Systematic Biology, 66*(5), 857–879. https://doi.org/10.1093/sysbio/syx041

Downloadable Code

The implementations of the algorithms for this final project can be found here: https://github.com/matthewse19/cs362-final