Neighbor Joining for Hierarchical Clustering of Phylogenetic Trees

A Preprint

Yusuf Alnawakhtha
Department of Computer Science
University of Maryland, College Park
College Park, MD 20740

Marina Knittel
Department of Computer Science
University of Maryland, College Park
College Park, MD 20740

Hamed Saleh
Department of Computer Science
University of Maryland, College Park
College Park, MD 20740

May 13, 2019

1 Introduction

Phylogenetic trees are biological applications of clustering that provide evolutionary relations between sets of DNA or protein sequences. A rooted phylogenetic tree is a directed tree where the root is the least common ancestor of the leaves. In contrast, unrooted phylogenetic trees describe the relationship between the leaves without defining or assuming an ancestor. For this project, we will specifically address unrooted phylogenetic trees. Each node in the tree represents the most recent common ancestor of its children and the weights on the branches of the tree represent the difference between the sequences.

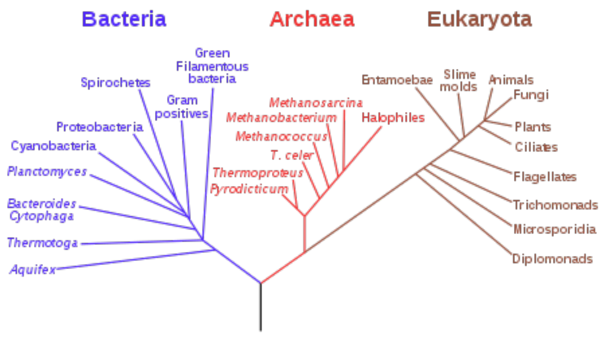
Phylogenetic trees provide insight and visualization of evolution and genetic mutation. This can be helpful for painting a better picture of how different sets of DNA or protein sequences are related, classifying species such as Apocynaceae [7], and assessing the targets of microbial warfare [2]. Furthermore, specific phylogenetic trees, such as the Y-Chromosomal phylogenetic tree, have applications in forensic science [3]. Phylogenetic trees can also be used to find the origin of pathogen outbreaks, like the avian influenza virus H7N3 outbreaks [4]. Moreover, they have applications in comparative linguistics to find how words are adopted from one language to another. Recently, a linguistic phylogenetic approach has been used for describing the evolution of color terms in languages [5].That is why it is desirable to have an efficient method of constructing phylogenetic trees from sets of DNA or protein sequences.

In this project, we will be utilizing the Neighbor Joining algorithm to create a phylogenetic trees given a distance matrix, which describes the difference between sequences, as input. The algorithm was purposed by Naruya Saitou and Masatoshi Nei in 1987 [1] and it has a time complexity of $\mathcal{O}(n^3)$ where n is the size of the set of sequences. We will also describe in this paper an attempt we had to provide a speedup to the algorithm.

2 The Neighbor Joining Algorithm

Neighbor joining is a particular method of hierarchical clustering for phylogentic tree construction. In this problem, we are given a bunch of species with "distances" between them, where $d(i, j)$ is the distance between species i and j, that represent how different they are from each other. It is specifically an agglomerative method, meaning that it starts with individual clusters for each species, and then iteratively merges the clusters into superclusters, eventually converging on a single cluster. We can depict this graphically as a binary tree, where each leaf is a species, and all the internal nodes represent the joining of two clustesr. This can be seen in Figure 1, and is more generally called a *dendrogram*.

Figure 1: A phylogenetic tree is an example of a dendrogram.



In neighbor joining, we start at the bottom of this tree and construct one internal node at a time working up the tree. Ideally, we want species that are more similar to be grouped closer in the tree, as in, they diverged more recently in evolutionary history. We do this by creating a sort of proxy for distance and store it in what we call a Q matrix. This matrix represnts the taxa we are currently looking to merge, so if we have t taxa, it will be a $t \times t$ matrix. The i, jth entry of Q corresponds to this distance proxy between taxa i and j. The proxy is as follows.

$$Q(i,j) = (n-2)d(i,j) - \sum_{k=1}^{n} d(i,k) - \sum_{k=1}^{n} d(j,k). \tag{1}$$

Intuitively, this is simply the distance between the two species minus all the distances between each of the two species and all others, where the distance between the two species is weighted much larger. Therefore, $Q(i,j)$ is small if $d(i,j)$ is small relative to the average distances between i and all other species, as well as j and all other species. We want this distance to be small, meaning these species are similar relative to other species.

In the actual algorithm, we start by calculating our Q matrix across all species. We then find the minimum value in Q, say $Q(i,j)$, and then merge species i and j. Now we will ignore species i and j because we have found the vertex directly above them in the dendrogram. They are replaced by a single new taxa that represents the cluster of i and j together.

To calculate the distance between u and both of its children i and j, we use the following method.

$$d(u,i) = \frac{1}{2}d(i,j) + \frac{1}{2(n-2)}\left[\sum_{k=1}^{n} d(i,k) + \sum_{k=1}^{n} d(k,j)\right].$$

We now need to find the distance between the new taxa and all the other taxa currently in question. Say i and j are the taxa we merged, and u is the new one, and we want to calculate the distance between u and some k. Because our graph is additive, we want the sum of the paths from i to k and j to k to be consistent on the graph we are constructing. The first path has length $d(i,k) = d(i,u) + d(u,k)$, and the second has length $d(j,k) = d(j,u) + d(u,k)$. Combining these together, we get the following.

$$d(u,k) = \frac{1}{2}[d(i,k) + d(j,k) - d(i,j)]. \tag{2}$$

Now we have all the new distances for the next iteration in the tree and are ready for the next iteration. We continue through this process, keeping track of the vertices we merge until we have constructed our entire tree.

An interesting property of the tree we construct comes from this notion of keeping the graph additive. In our resulting tree, note that we have each branch distance as represented by the d of adjacent vertices in the corresponding steps of the algorithm. It then holds true, by construction of the tree, that $d(i,j)$ for any species i and j is equal to the path length in the resulting tree. Because of this, if we initially derive our distances based off of some input tree T, then neighbor joining will output a tree that agrees with it.

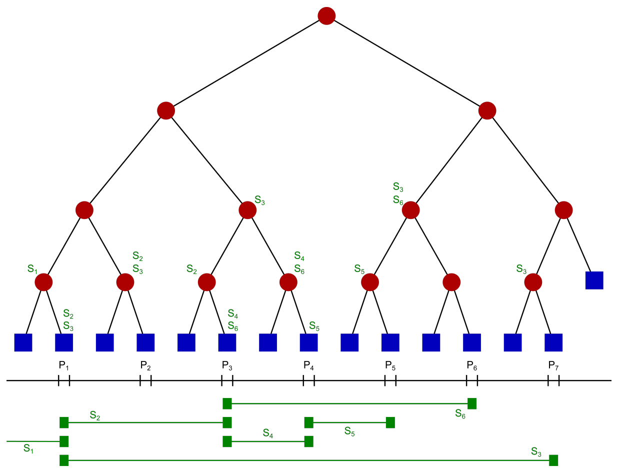
This algorithm is polynomial time, but it is unclear if the runtime can be improved while keeping the same structure. It requires $\mathcal{O}(n)$ iterations, and each iteration requries $\mathcal{O}(n^2)$ computations to compute Q and recompute distances. Therefore, it runs in $\mathcal{O}(n^3)$ time. We are interested in improving this.

3 Theoretical Work

Our approach to speedup the algorithm is to simulate the changes that occur to matrix Q during the algorithm in a data structure which handles the queries in polylogarithmic time. The desired data structure should capture changes to Q after each step while maintaining the minimum element in Q for the next step. We can use the minimum element in Q to merge two taxa, and make queries to perform the induced changes in Q. We breakdown the required updates to capture all changes in Q, and show all except one of them can be performed on a $(2t-1) \times (2t-1)$ two-dimensional segment tree in polylogarithmic time. Our goal is to handle all updates in each step with a total of $\widetilde{\mathcal{O}}(n)$ running time, while the number of updates in each step is $\mathcal{O}(n)$.

Segment Tree is a useful data structure to implement range queries and updates. It is a binary tree with n leaves, where each node represents an interval on a sequence of length n. The root of this tree represents interval $[1, n]$, and a node representing interval $[l, r]$, for $l < r$, has two children representing $[l, (r+l)/2]$ and $[(l+r)/2+1, r]$. This way, the i-th leaf represents interval $[i, i]$. In this data structure, we can perform range updates and range queries in logarithmic time. For example, a range update (l, r, v) adds a constant value v to elements $l, l+1, \ldots, r$, and a range query (l, r) returns the sum of elements $l, l+1, \ldots, r$. The main intuition for performing these updates and queries efficiently is that any given range $[l, r]$ can be decomposed into $\mathcal{O}(\log n)$ segment tree intervals. This process is illustrated in Figure 2.[1]

Figure 2: An example of segment tree with some range queries decomposed into segment tree intervals (green labels above the nodes show the decomposition)



One can extend the notion of segment tree into higher dimensions, say a d-dimensional segment tree, by simply creating a one-dimensional segment tree of $(d-1)$-dimensional segment trees. For the two-dimensional case, we create a one-dimensional segment tree by x-axis where each of its nodes contain a one-dimensional segment tree by y-axis. This results in $\mathcal{O}(n^{d-1}\log n)$ running time for submatrix queries and updates. Ibtehaz et. al [6] showed it is possible to implement a multi-dimensional segment tree capable of handling submatrix queries and updates in $\mathcal{O}(\log n^d)$ time. We can use this result to perform polylogarithmic submatrix updates in our two-dimensional segment tree.

We can break down changes induced on Q after each step as different submatrix queries. Recall equation 1 for computing $Q(i, j)$:

$$Q(i,j) = (n-2)d(i,j) - \sum_{k=1}^{n} d(i,k) - \sum_{k=1}^{n} d(j,k).$$

[1]For more details, take a look at https://en.wikipedia.org/wiki/Segment_tree.

At each step, we choose two taxa i and j with minimum $Q(i,j)$ to join. Next, we replace vertices i and j with a dummy new vertex labeled u. We can keep track of $\sum_{k=1}^{n} d(v,k)$ for each vertex v. It is easy update them by subtracting $d(v,i)$ and $d(v,j)$, and adding new $d(v,u)$ we just computed for each vertex in linear time. We can also find $\sum_{k=1}^{n} d(u,k)$ for the new vertex, and therefore, finding $Q(u,k)$ for any k is possible in linear time, and inserting them in matrix Q is a collection of $2(n-1)$ 1×1 submatrix updates. We can similarly update $Q(i,k)$ and $Q(j,k)$ for any k with another collection of 1×1 submatrix updates with large enough values, we only need to prevent them to be considered the minimum element from now on. It only remains to update $Q(i',j')$ for any pair of $i',j' \notin \{i,j,u\}$. We breakdown the modifications to $Q(i',j')$ separately for each term in Equation 1:

- $\sum_{k=1}^{n} d(i',k)$: Since vertices i and j are removed and vertex u is added, according to Equation 2, we decrease this term by the following.

$$d(i',i) + d(i',j) - d(i',u) = d(i',u) + d(i,j)$$

  This value is the same for every $Q(i',k)$, which means we add a specific value to each row. Thus, we can translate the changes to this term as $1 \times n$ submatrix updates.
- $\sum_{k=1}^{n} d(i',k)$: Similar to the previous term, we can translate the changes as $n \times 1$ submatrix updates, because we add a specific value to each column.
- $(n-2)d(i',j')$: Since $d(i',j')$ is not changed, we only need to reduce $d(i',j')$ from each $Q(i',j')$. This is the part we are not able to capture with a two-dimensional segment tree.

4 Conclusion

We have described in this paper the workings of Saitou and Nei's Neighbor Joining algorithm for constructing phylogenetic trees and our attempt of improving the run time. We provide alongside this paper an implementation of their algorithm in Python, as well as a testing function for the algorithm.

The testing function works by specifying a number of leaves, assigning a parent to two randomly chosen nodes, assigning the branch weights randomly, and repeating the process until all leaves are connected. Once the tree is generated, we calculate the distance between the leaves by adding the branch weights leading to their most recent common ancestor. This provides us with a distance matrix that can be fed into the Neighbor Joining algorithm and we expect the output of the algorithm to be the randomly generated tree.

It is important to note that while phylogenetic trees provide various applications, they also have their shortcomings. Most notably, they do not always represent evolution and gene mutation accurately. This can be caused by occurrences such as species hybridization, which would cause clustering algorithms to assign different species closer on the phylogenetic tree than they should be. Another clear phenomenon that would cause an inaccurate phylogenetic tree is convergent evolution of different species. Despite this, phylogenetic trees remain a useful visualization of evolutionary history.

References

[1] N. Saitou and M. Nei, "The neighbor-joining method: a new method for reconstructing phylogenetic trees." *Molecular biology and evolution*, vol. 4, no. 4, pp. 406–425, 1987.

[2] M. A. Riley, C. Goldstone, J. Wertz, and D. Gordon, "A phylogenetic approach to assessing the targets of microbial warfare," *Journal of evolutionary biology*, vol. 16, no. 4, pp. 690–697, 2003.

[3] A. Van Geystelen, R. Decorte, and M. Larmuseau, "Updating the y-chromosomal phylogenetic tree for forensic applications based on whole genome snps," *Forensic Science International: Genetics*, vol. 7, no. 6, pp. 573–580, 2013.

[4] L. Lu, S. J. Lycett, and A. J. L. Brown, "Determining the phylogenetic and phylogeographic origin of highly pathogenic avian influenza (h7n3) in mexico," *PloS one*, vol. 9, no. 9, p. e107330, 2014.

[5] H. J. Haynie and C. Bowern, "Phylogenetic approach to the evolution of color term systems," *Proceedings of the National Academy of Sciences*, vol. 113, no. 48, pp. 13 666–13 671, 2016.

[6] N. Ibtehaz, M. Kaykobad, and M. S. Rahman, "Multidimensional segment trees can do range queries and updates in logarithmic time," *arXiv preprint arXiv:1811.01226*, 2018.

[7] B. Sennblad and B. Bremer, "Classification of apocynaceae s. 1. according to a new approach combining linnaean and phylogenetic taxonomy," *Systematic biology*, vol. 51, no. 3, pp. 389–409, 2002.