

Efficiently Computing Arbitrarily-Sized Robinson-Foulds Distance Matrices

Seung-Jin Sul, Grant Brammer, and Tiffani L. Williams

Department of Computer Science
Texas A&M University
College Station, TX 77843-3112 USA
{sulsj,grb9309,tlw}@cs.tamu.edu

Abstract. In this paper, we introduce the HashRF(p, q) algorithm for computing RF matrices of large binary, evolutionary tree collections. The novelty of our algorithm is that it can be used to compute arbitrarily-sized ($p \times q$) RF matrices without running into physical memory limitations. In this paper, we explore the performance of our HashRF(p, q) approach on 20,000 and 33,306 biological trees of 150 taxa and 567 taxa trees, respectively, collected from a Bayesian analysis. When computing the all-to-all RF matrix, HashRF(p, q) is up to 200 times faster than PAUP* and around 40% faster than HashRF, one of the fastest all-to-all RF algorithms. We show an application of our approach by clustering large RF matrices to improve the resolution rate of consensus trees, a popular approach used by biologists to summarize the results of their phylogenetic analysis. Thus, our HashRF(p, q) algorithm provides scientists with a fast and efficient alternative for understanding the evolutionary relationships among a set of trees.

Keywords: phylogenetic trees, Robinson-Foulds distance, clustering, performance analysis.

1 Introduction

Bayesian analysis [1] is one of the most common approaches for reconstructing an evolutionary history (or phylogeny) of a set of organisms (or taxa). Such analysis can easily produce tens of thousands of potential, binary trees that later have to be summarized in some way. Currently, scientists use consensus trees to summarize the results from these multitudes of trees into a single tree. Yet, much information is lost by summarizing the evolutionary relationships between the trees into a single consensus tree [2], [3]. In this paper, we explore an alternative approach, which compliments consensus trees, to help scientists better understand the results of their phylogenetic analysis.

We developed the HashRF(p, q) algorithm as the basis for an alternative approach for understanding the relationships among a collection of t trees. The *novelty* of this algorithm is that it can compute arbitrarily-sized ($p \times q$) RF matrices (see Figure 1). Moreover, it can be leveraged to minimize the memory requirements of computing extremely, large RF matrices and it can be used to take

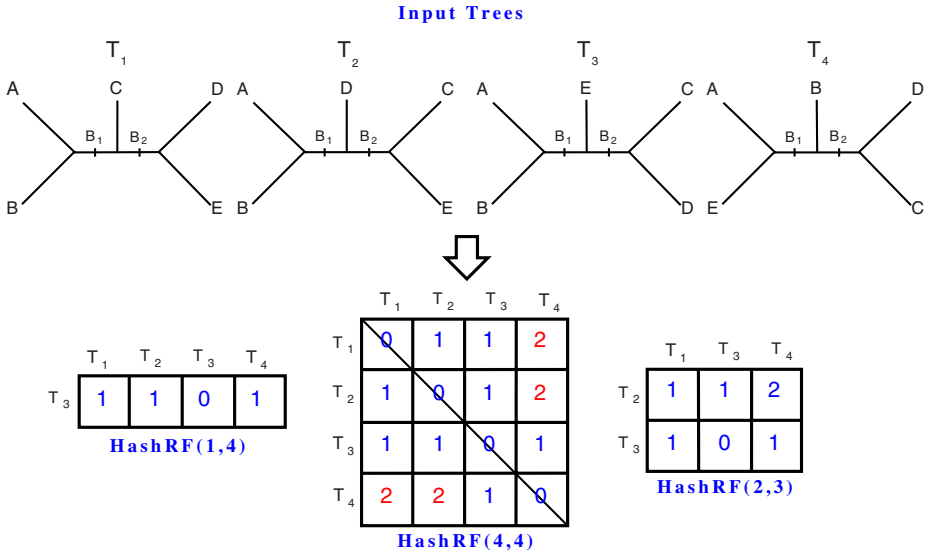


Fig. 1. Overview of computing the RF distance matrix using the HashRF(p, q) algorithm. The tree collection consists of four phylogenies: T_1, T_2, T_3 , and T_4 . Bipartitions (or internal edges) in a tree are labeled B_i , where i ranges from 1 to 2. Three different RF matrices are shown that can be produced by HashRF(p, q). HashRF(1,4) produces a *one-to-all* matrix, HashRF(4,4) computes an *all-to-all* matrix, and HashRF(2,3) computes a 2×3 matrix.

advantage of parallel platforms. Several software packages, such as PAUP* [4] and Phylip [5], support computing the topological distance between phylogenetic trees. However, these tools are only suitable for small distance matrices. Other RF algorithms, such as Day’s algorithm [6], PGM-Hashed [7], and HashRF [8] cannot compute arbitrarily-sized matrices. They are limited to $t \times t$ matrices.

Our results are based on our large collection of biological trees (20,000 trees of 150 taxa and 33,306 trees of 567 taxa) obtained from a Bayesian analysis. Although the benefits of our HashRF(p, q) algorithm is that it can compute arbitrarily-sized matrices, we test its performance against other RF matrix algorithms (PAUP*, PGM-Hashed, and HashRF) that compute all-to-all distance matrices. Hence, in these experiments, the p and q parameters of the HashRF(p, q) algorithm are both set to t , the number of trees in the collection of interest. For large all-to-all matrices, the results clearly demonstrate that HashRF(p, q) is the best approach. It is over 200 times faster than PAUP*, and up to 40% faster than HashRF, one of the fastest algorithms for computing an all-to-all matrix.

Once a RF distance matrix is obtained for a tree collection, there are a number of data-mining techniques that can be used to help understand the relationship among the trees. In this paper, we cluster the RF distance matrices in order to improve the quality of the consensus trees, which are used to summarize large

numbers of trees into a single phylogeny. As stated earlier, consensus trees are traditionally used in this way. However, the disadvantage of such an approach is potentially losing vital evolutionary relationships that are in the tree collection, but not depicted in the consensus tree. For our tree collections, we found mixed results. For our 150 taxa trees, clustering the resulting RF distance matrices made a significant impact on improving the quality of the consensus tree. However, the improvement for our 567 taxa trees was minimal since the trees are already quite similar.

Overall, HashRF(p, q) provides researchers with a technique to produce arbitrarily-sized RF distance matrices. With our new algorithm, life scientists have a way to apply further clustering and post-processing methods to understand their large collections of phylogenetic trees.

2 Basics

2.1 Phylogenetic Trees

In a phylogenetic tree, modern organisms (or taxa) are placed at the leaves and ancestral organisms occupy internal nodes, with the edges of the tree denoting evolutionary relationships. Oftentimes, it is useful to represent phylogenies in terms of their *bipartitions*. Removing an internal edge e from a tree separates the taxa (or leaves) on one side from the taxa on the other. The division of the taxa into two subsets is the non-trivial bipartition B associated with internal edge e . (Note: all trees have trivial bipartitions denoted by external edges.) In Figure 1, T_2 has two bipartitions (or internal edges): $AB|CDE$ represented by B_1 and $ABD|CE$ represented by B_2 . An evolutionary tree is uniquely and completely defined by its set of $O(n)$ bipartitions.

2.2 Robinson-Foulds (RF) Distance

The Robinson-Foulds (RF) distance between two trees is the number of bipartitions that differ between them. Let $\Sigma(T)$ be the set of bipartitions defined by all edges in tree T . The RF distance between trees T_1 and T_2 is defined as:

$$d_{RF}(T_1, T_2) = \frac{|\Sigma(T_1) - \Sigma(T_2)| + |\Sigma(T_2) - \Sigma(T_1)|}{2} \quad (1)$$

In Figure 1, consider the RF distance between trees T_1 and T_2 . The set of bipartitions defined for tree T_1 is $\Sigma(T_1) = \{AB|CDE, ABC|DE\}$. $\Sigma(T_2) = \{AB|CDE, ABD|CE\}$. The number of bipartitions appearing in T_1 and not T_2 (i.e., $|\Sigma(T_1) - \Sigma(T_2)|$) is 1, since $\{ABC|DE\}$ does not appear in T_2 . Similarly, the number of bipartitions in T_2 but not in T_1 is 1. Hence, $d_{RF}(T_1, T_2) = 1$. The largest possible RF distance between two binary trees is $(n - 3)$, which results in a maximum RF value of 2 in the example shown in Figure 1.

In this paper, we are interested in computing the $p \times q$ matrix, where $1 \leq p, q \leq t$. Given two sets, S_1 and S_2 , of input trees (where $|S_1| = p$ and $|S_2| = q$),

the output is a $p \times q$ matrix of RF distances. If p and q are equal to t (the number of trees of interest), then the matrix represents the *all-to-all RF distance* (or $t \times t$ RF matrix) between every pair of t trees. When $p = 1$ and $q = t$, then the matrix represents the *one-to-all distance* between the trees. The all-to-all RF distance is quite useful if one is interested in feeding the resulting $t \times t$ matrix to a clustering algorithm. We show such an application for our collection of biological trees in Section 6.2. One-to-all RF distances are useful for comparing a single tree (such as the best tree found by a heuristic or a published tree) to a set of trees of interest.

2.3 Consensus Trees

Consensus trees are used to summarize the information from the set of trees usually resulting from phylogenetic search heuristics employed by popular software such as PAUP* [4] and MrBayes [9]. Among many different consensus methods, the strict consensus and majority consensus trees are widely used in phylogenetics. The strict consensus approach returns a tree such that the bipartitions of the tree are only those bipartitions that occur in every input tree. On the other hand, the majority consensus algorithm makes a consensus tree such that the bipartitions are only those that occur in over 50% of the input trees.

Oftentimes, the resulting consensus tree is not a binary tree. If this is the case, the consensus tree is considered to be multifurcating. We use *resolution rate* to measure the percentage of binary internal edges in a phylogenetic tree. The resolution rate is defined as the number of binary internal nodes (nodes of degree 3) divided by $(n - 3)$, which represents the maximum number of internal nodes in a binary tree of n taxa. Hence, a binary tree is 100% resolved. For the four trees in Figure 1, the majority consensus tree simply consists of a tree with the bipartition $AB|CDE$, which results in a resolution rate of 75%. On the other hand, the strict consensus trees is a star (i.e., it's completely unresolved) and results in a resolution rate of 0% resolved.

3 HashRF: Computing an All-to-All RF Matrix

Figure 2 provides an overview of the HashRF algorithm, which has a running time complexity of $O(nt^2)$, where n is the number of taxa and t is the number of trees. Each input tree, T_i , is traversed in post-order, and its bipartitions are fed through two hash functions, h_1 and h_2 . Hash function h_1 is used to generate the location needed for storing a bipartition in the hash table. h_2 is responsible for creating bipartition identifiers (BIDs). For each bipartition, its associated hash table record contains its BID along with the tree index (TID) where the bipartition originated.

3.1 Organizing Bipartitions in the Hash Table

Our h_1 and h_2 universal hash functions are defined as follows.

$$h_1(B) = \sum b_i r_i \bmod m_1 \quad (2)$$

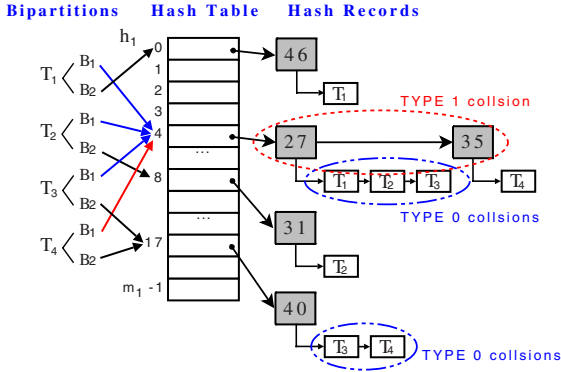


Fig. 2. Overview of the HashRF algorithm. Bipartitions are from Figure 1. (a) The implicit representation of each bipartition, B_i , is fed to the hash functions h_1 and h_2 . The shaded value in each hash record contains the bipartition ID (or h_2 value). Each bipartition ID has a linked list of tree indexes that share that particular bipartition.

$$h_2(B) = \sum b_i s_i \text{ mod } m_2 \tag{3}$$

m_1 represents the number of entries (or locations) in the hash table. m_2 represent the largest bipartition ID (BID) that we can be given to a bipartition. That is, instead of storing the n -bitstring, a shortened version of it (represented by the BID) will be stored in the hash table instead. $R = (r_1, \dots, r_n)$ is a list of random integers in $(0, \dots, m_1 - 1)$, $S = (s_1, \dots, s_n)$ is a list of random integers in $(0, \dots, m_2 - 1)$, and $B = (b_1, \dots, b_n)$ is a bipartition represented by an n -bitstring. We can avoid sending the n -bitstring bipartition representations to our hash functions h_1 and h_2 . Instead, we use an implicit bipartition representation to compute the hash functions quickly. An implicit bipartition is simply the integer value (instead of the n -bitstring) that provides the representation of the bipartition.

A consequence of using hash functions is that bipartitions may end up residing in the same location in the hash table. Such an event is considered a collision. There are three types of collisions that our hashing algorithms must resolve. *Type 0* collisions occur when the same bipartition (i.e. $B_i = B_j$), is shared across the input trees. Such collisions are not serious and are a function of the set of input trees. *Type 1* collisions result from two different bipartitions B_i and B_j (i.e., $B_i \neq B_j$) residing in the same location in the hash table. That is, $h_1(B_i) = h_1(B_j)$. (We note that this is the standard definition of collisions in hash table implementations.) *Type 2* collisions are serious and require a restart of the algorithm if such an event occurs. Otherwise, the resulting output will be incorrect. Suppose that $B_i \neq B_j$. A Type 2 collision occurs when B_i and B_j hash to the same location in the hash table and the bipartition IDs (BIDs) associated with them are also the same. In other words, $h_1(B_i) = h_1(B_j)$ and $h_2(B_i) = h_2(B_j)$. The probability of our hash-based approach having to restart because of a double collision among any pair of the bipartitions is $O(\frac{1}{c})$. Since c

can be made arbitrarily large, the probability of a restart can be made infinitely small. In our experiments, $c = 1,000$.

3.2 Computing the RF Matrix

Once all the bipartitions are organized in the hash table, then the RF distance matrix can be calculated. For each non-empty hash table location i , we have a list of tree index (TID) nodes for each unique bipartition ID (BID) node. Consider the linked list of bipartitions in location 4 of the hash table in Figure 2 for BID 27 which contains TIDs $\{T_1\}$, $\{T_2\}$, $\{T_3\}$. The hash table shows that trees T_1, T_2 , and T_3 share the same bipartition ($ABC|DE$ from Figure 1). HashRF uses a $t \times t$ dissimilarity matrix, D , to track the number of bipartitions that are different between all tree pairs. For each tree i , its row entries are initialized to b_i , the number of bipartitions present in tree i . Hence, $D_{i,j} = b_i$ for $0 \leq j < t$ and $i \neq j$. $D_{i,i} = 0$. In the case of binary trees, the $D_{i,j}$ entries are initialized to $n - 3$, the maximum number of internal edges in a binary tree.

For each BID node at location l , every pair of TID nodes in the linked list are compared to each other. Then, the counts of $D_{i,j}$ and $D_{j,i}$ are decremented by one. That is, we have found a common bipartition between T_i and T_j and decrement the difference counter by one. For example, trees with BID 27 at location 4 in the hash table shows that the pairs (T_1, T_2) , (T_1, T_3) , and (T_2, T_3) share a bipartition. Thus, entries $D_{1,2}$, $D_{1,3}$, and $D_{2,3}$ are decremented by one. Once we have computed D , we can compute the RF matrix quite easily. Thus, $RF_{i,j} = \frac{D_{i,j} + D_{j,i}}{2}$, for every tree pair i and j .

4 Our New Approach: The HashRF(p, q) Algorithm

4.1 Motivation

Our previous work with HashRF [8] focused on designing a fast algorithm to compute the all-to-all (or $t \times t$) RF matrix between every pair of t trees. Consider a collection of t binary trees. There are two distinct advantages to using our new HashRF(p, q) approach, which extends our HashRF algorithm.

1. We can now compute a $p \times q$ matrix, where $1 \leq p, q \leq t$.
2. There are no physical memory limitations regarding the size of a RF matrix we can compute.

The first advantage of our HashRF(p, q) algorithm results in being able to compute other types of RF matrices besides all-to-all matrices. Although all-to-all matrices are quite useful—we show their usefulness in Section 6.2—other matrix shapes are helpful. For example, one might be interested in a one-to-all RF matrix, where $p = 1$ and $q = t$, which shows how different a single tree (e.g., best-known or published tree) is from a collection of trees. The other important advantage of the HashRF(p, q) approach is that it can be used to compute extremely large matrices—especially those that are too large to fit entirely into

physical memory. By dividing such large $t \times t$ matrices into smaller $p \times q$ chunks, it becomes feasible to compute any RF matrix size desirable. Furthermore, if a multi-core machine is available, then these independent blocks could be computed in parallel to achieve a significant increase in performance.

4.2 HashRF(p, q) vs. HashRF

HashRF(p, q) works similarly to HashRF, which was described in Section 3. The first major difference between the two algorithms is that HashRF(p, q) requires as input two sets of trees, S_p and S_q , where $|S_p| = p$ and $|S_q| = q$. HashRF requires only one set of trees, S , as input and $|S| = t$. HashRF(p, q) requires the sets S_p and S_q of trees since the trees in S_p will only be compared to trees in S_q . Trees within a set will not be compared to each other.

In the HashRF(p, q) approach, we assume that $p \leq q$. As a result, we place all of bipartitions of the trees in S_q into the hash table. Once this is done, then we process each tree T_i in the set S_p . For each bipartition B of tree T_i , we apply our h_1 and h_2 hashing functions as described in Section 3. Once we determine where bipartition B would be located (using the h_1 function) in the hash table, we compare it's bipartition ID (using the h_2 function) to those nodes that are at that location. Suppose this location or index is l in the hash table. At location l , for each tree T_j at location l with the same BID as tree T_i , we increment the counter in the matrix at locations (T_i, T_j) and (T_j, T_i) by one—assuming the full matrix is of interest and not the lower or upper triangle. We repeat the above steps for each remaining tree in the set S_p . Afterwards, since we are interested in the RF distance (and not similarity), we subtract $n - 3$ from the values since that is the maximum RF distance for a binary tree consisting of n taxa.

In terms of running time, the HashRF approach requires $O(nt^2)$ time. For HashRF(p, q) since we assume that $p \leq q \leq t$, then the running time is $O(nq^2)$. Hence, if a smaller sub-matrix is of interest, it will be significantly faster to compute than an all-to-all RF distance matrix.

5 Experimental Methodology

5.1 Biological Tree Collections

The biological trees used in this study were obtained from two recent Bayesian analysis, which we describe below.

- 20,000 trees obtained from a Bayesian analysis of an alignment of 150 taxa (23 desert taxa and 127 others from freshwater, marine, and oil habitats) with 1,651 aligned sites [10].
- 33,306 trees obtained from an analysis of a three-gene, 567 taxa (560 angiosperms, seven outgroups) dataset with 4,621 aligned characters, which is one of the largest Bayesian analysis done to date [11].

In our experiments, for each number of taxa, n , we created different tree set sizes, t , to test the scalability of the algorithms. For $n = 150$ and 567, the entire

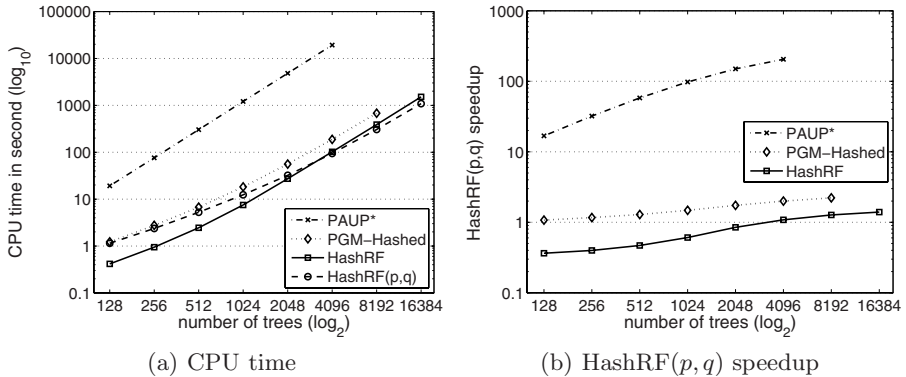


Fig. 3. The performance of the various RF matrix algorithms to compute a $t \times t$ matrix, where t is the number of trees, our 567 taxa dataset. For HashRF(p, q), p and q are both equal to t .

collection of trees is divided into smaller sets, where t is 128, 256, 512, \dots , 16,384 trees. Thus, for each (n, t) pair, t trees with n taxa were randomly sampled without replacement from the appropriate tree collection. We repeated the above randomly sampling procedure five times in order to plot the average performance of the RF algorithms.

5.2 RF Matrix Implementations

We obtained the source code for PGM-Hashed from the authors. We used version 4.0b10 of PAUP*. Experimental results were compared across the competing algorithms for experimental validation. All experiments were run on an Intel Pentium platform with 3.0GHz dual-core processors and a total of 2GB of memory. HashRF and PGM-Hashed were written in C++ and compiled with gcc 4.1.0 with the `-O2` compiler option. Each plot shows the average performance over five runs.

6 Experimental Results

6.1 HashRF(p, q) Performance

First, we consider the running time of our HashRF(p, q) algorithm against its competitors for computing an all-to-all (or $t \times t$) RF matrix. We also show the performance of HashRF in comparison with HashRF(p, q). Figure 3 shows the running time and speedup of HashRF(p, q) over its competitors on our 567 taxa tree collection. The results for our 150 taxa tree collections are similar. To compute the speedup values, the running times of PAUP*, PGM-Hashed, and HashRF are divided by the running time of HashRF(p, q), the algorithm of most interest to us in our experiments. Both HashRF and HashRF(p, q) clearly outperform the other algorithms. PAUP* is the slowest RF matrix algorithm requiring

5.35 hours to compute a $4,096 \times 4,096$ RF distance matrix. HashRF(p, q) only requires 93.9 seconds for the same data set, which results in a speedup of over 200 in comparison to PAUP*. As the number of taxa is increased, the speedup of HashRF(p, q) increases as well, where the closest competitor PGM-Hashed is up to two times slower than HashRF(p, q).

Figure 3 also depicts several other interesting points. Although HashRF(p, q) was designed to compute sub-matrices, it is the best choice for computing the all-to-all matrix when $t > 2,048$ trees. HashRF is already a fast approach, but HashRF(p, q) can improve performance by as much as 40% when $t = 16,384$. For computing smaller $t \times t$ matrices, HashRF is the preferred approach. Since PAUP* takes over 12 hours to compute $t \times t$ matrices, when $t \geq 8,192$, we choose to terminate the run. Finally, PGM-Hashed is unable to compute the $16,384 \times 16,384$ RF matrix.

Lastly, we consider computing the all-to-all matrix of our entire collection of trees, which is useful for the clustering of trees that is described in the next section. In other words, we are interested in computing the $20,000 \times 20,000$ matrix and the $33,306 \times 33,306$ matrix of our 150 and 567 taxa trees, respectively. Both the HashRF and HashRF(p, q) algorithms can compute the $20,000 \times 20,000$ matrix quite easily. However, both algorithms could not compute the all-to-all matrix for the 33,306 trees since the algorithms ran out of memory. In such a situation, we divided the larger matrix into 3 blocks, where each block was of size $11,102 \times 33,306$. We divided the input appropriately to feed to the HashRF(p, q) algorithm (i.e., one file contained 11,102 trees and the other had 33,306 trees). After each block was computed, we simply concatenated the results. Overall, 1 hour and 15 minutes was used to produce the $33,306 \times 33,306$ matrix.

6.2 HashRF(p, q) Application: Improving Consensus Tree Resolution Rates

Now, consider an application of the HashRF(p, q) algorithm. Given a collection of trees returned from a Bayesian analysis, phylogenetic researchers often reduce this information into a single consensus tree. However, one disadvantage of such an approach is losing vital evolutionary relationships, which can be quantified by the resolution rate of the consensus tree. The higher the resolution rate, the more in agreement the consensus tree is with the overall collection of trees. We explore the impact of clustering based on using RF distance matrices to improve the resolution rates of majority and strict consensus trees.

Clustering Methodology. Using HashRF(p, q) we generated the all-to-all RF distance matrices for our two biological collections of t trees, where t is 20,000 and 33,306 for 150 and 567 taxa trees, respectively. Next, we converted the distance matrices into similarity matrices by subtracting the value in each cell from $n - 3$, the total possible number of bipartitions for n taxa. These similarity matrices are then used as the input to CLUTO [12], which is software designed for clustering high-dimensional datasets.

Unfortunately, we were unable to cluster the entire $t \times t$ matrices in CLUTO since its memory requirements were too large for our platform. Although we

have an approach that can create very large RF matrices that isn't limited by physical memory space, CLUTO required more memory space than we had available. Hence, for each of our tree collections, we created smaller, random samples of trees to cluster. These samples were created by selected 5,000 trees at random without replacement from our collection of 20,000 and 33,306 trees. We then created a $5,000 \times 5000$ similarity matrix, which was fed to CLUTO. The above process was repeated five times.

For our clusterings, we mostly used the default settings in CLUTO. For example, we used repeated bipartition clustering, where the graph is partitioned multiple times to produce the requested number of clusters. We also chose to use the default optimization criterion which is CLUTO's *i2* function. The *i2* function is meant to optimize the similarity between the data points in the clusters. Moreover, we felt that optimizing the internal similarity of the data suited our goals of increasing consensus resolution rates through clustering.

Clustering 150 and 567 Taxa Trees. The only default setting we changed was specifying the number of clusters, K , to partition the 5,000 trees in the sample. The default setting for K is 10. Since the best value for K is unknown, we set $K = 2$ for the 150 taxa dataset since the trees were obtained by two distinct runs, where each run consisted of 10,000 trees. Moreover, we also wanted to minimize the number of different clusterings produced. If we take the resolution rate of the entire collection 20,000 trees, the resulting majority and strict consensus resolution rates are 85.7% and 34.0%, respectively. But, by clustering the data into two partitions, we were able to increase the average consensus resolution rate to 89.2% and 38.2% for majority and strict trees, respectively.

However, the 567 taxa dataset tells a different story concerning its 33,306 total trees. We initially set $K = 3$ for this data because it was derived from three distinct runs, each consisting of 11,102 trees. However, setting $K = 3$ resulted in two clusters of around 2,500 trees and a third cluster of size two. Hence, we repeated the experiment with $K = 2$. The resolution rate of the majority and strict trees for the entire collection of 33,306 trees is 92.2% and 51.7%, respectively. By clustering, we were able to improve the average resolution rate slightly to 92.7% for the majority trees and 53.4% for the strict trees. In this case the clustering algorithm was unable to make significant improvements to the consensus resolution rates since the level of similarity among the trees is already quite high.

7 Conclusions and Future Work

Phylogenetic analysis can produce a large number of binary trees, which present a tremendous data-mining challenge for understanding the evolutionary relationships between them. Currently, life scientists often use consensus trees to summarize their trees. However, a lot of information regarding the evolutionary relationships contained in the trees can be lost. One of the advantages of computing a RF distance matrix to summarize the information is that it can be fed

to a clustering algorithm to explore the relationships among the trees, which we explore in our work.

We develop a new algorithm called HashRF(p, q) which is not limited to computing the all-to-all (or $t \times t$) matrices between a collection of t trees. HashRF(p, q) can compute arbitrarily-sized $p \times q$ matrices, where $1 \leq p, q \leq t$. Moreover, the HashRF(p, q) approach can be used to compute very large RF matrices, which are not bounded by the amount of physical memory available on a user's system. Our experimental study on large collections of Bayesian trees shows that HashRF(p, q) is the best performing algorithm for large $t \times t$ matrices, where the number of trees is greater than 2,048. Popular phylogenetic software, such as PAUP*, is up to 200 times slower than HashRF(p, q). Furthermore, HashRF(p, q) is around 40% faster than our HashRF approach for large all-to-all matrices. Such performance results suggests that we could combine the two hash-based approaches into one.

The main motivation for computing the RF distance matrix for such a large collection of trees is to provide researchers with input data for post-processing techniques. In this paper, we show clustering as a method for increasing the resolution rates of consensus trees. For the 150 taxa dataset, by creating two clusters of trees, we were able to improve the resolution rate of the majority and strict consensus trees by 3.5% and 4.2%, respectively. More specifically, clustering providing us with two majority (strict) trees that had an average resolution of 89.2% (38.2%). However, for the 567 taxa trees, clustering was not able to improve the quality of the consensus trees as the 33,306 trees are already quite similar.

Our work can be extended in many different directions. First, we plan to develop a parallel algorithm that uses HashRF(p, q) to compute an arbitrarily sized RF matrix faster. Now that we are able to obtain distance matrices of arbitrary size, we can use them for further exploration of clustering and other post-processing methods. Finally, we believe our approach is a good step toward handling larger sets of trees since the goal of a phylogenetics is to reconstruct the Tree of Life, which is estimated to contain up to 100 million taxa.

Acknowledgements

We would like to thank Matthew Gitzendanner, Bill Murphy, Paul Lewis, and David Soltis for providing us with the Bayesian tree collections used in this paper. Funding for this project was supported by the National Science Foundation under grants DEB-0629849 and IIS-0713618.

References

1. Huelsenbeck, J.P., Ronquist, F., Nielsen, R., Bollback, J.P.: Bayesian inference of phylogeny and its impact on evolutionary biology. *Science* 294, 2310–2314 (2001)
2. Hillis, D.M., Heath, T.A., John, K.S.: Analysis and visualization of tree space. *Syst. Biol.* 54(3), 471–482 (2005)

3. Stockham, C., Wang, L.S., Warnow, T.: Statistically based postprocessing of phylogenetic analysis by clustal. In: Proceedings of 10th Int'l Conf. on Intelligent Systems for Molecular Biology (ISMB 2002), pp. 285–293 (2002)
4. Swofford, D.L.: PAUP*: Phylogenetic analysis using parsimony (and other methods), Sinauer Associates, Sunderland, Massachusetts, Version 4.0 (2002)
5. Felsenstein, J.: *Inferring Phylogenies*. Sinauer Associates (2003)
6. Day, W.H.E.: Optimal algorithms for comparing trees with labeled leaves. *Journal Of Classification* 2, 7–28 (1985)
7. Pattengale, N., Gottlieb, E., Moret, B.: Efficiently computing the Robinson-Foulds metric. *Journal of Computational Biology* 14(6), 724–735 (2007)
8. Sul, S.J., Williams, T.L.: A randomized algorithm for comparing sets of phylogenetic trees. In: Proc. Fifth Asia Pacific Bioinformatics Conference (APBC 2007), pp. 121–130 (2007)
9. Huelsenbeck, J.P., Ronquist, F.: MRBAYES: Bayesian inference of phylogenetic trees. *Bioinformatics* 17(8), 754–755 (2001)
10. Lewis, L.A., Lewis, P.O.: Unearthing the molecular phylodiversity of desert soil green algae (chlorophyta). *Syst. Bio.* 54(6), 936–947 (2005)
11. Soltis, D.E., Gitzendanner, M.A., Soltis, P.S.: A 567-taxon data set for angiosperms: The challenges posed by bayesian analyses of large data sets. *Int. J. Plant Sci.* 168(2), 137–157 (2007)
12. Karypis, G.: CLUTO—software for clustering high-dimensional datasets. Internet Website (last accessed, June 2008), <http://glaros.dtc.umn.edu/gkhome/cluto/cluto/overview>