

# An Experimental Analysis of Robinson-Foulds Distance Matrix Algorithms

Seung-Jin Sul and Tiffani L. Williams

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

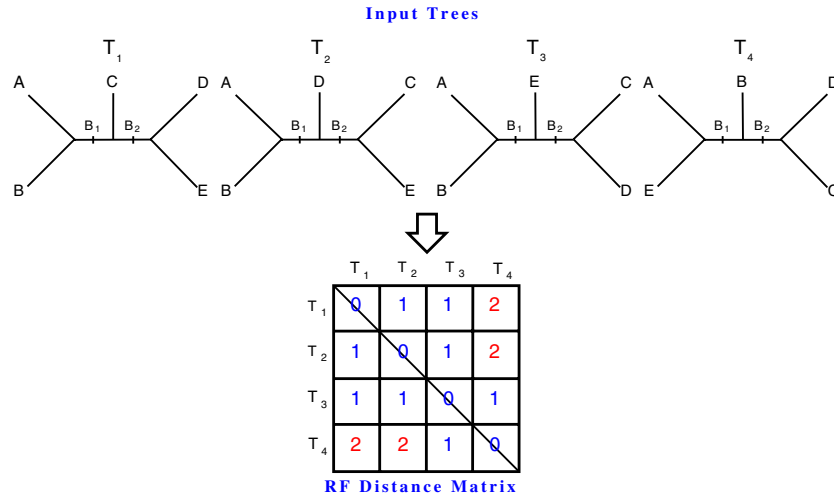
**Abstract.** In this paper, we study two fast algorithms—HashRF and PGM-Hashed—for computing the Robinson-Foulds (RF) distance matrix between a collection of evolutionary trees. The RF distance matrix represents a tremendous data-mining opportunity for helping biologists understand the evolutionary relationships depicted among their trees. The novelty of our work results from using a variety of different architecture- and implementation-independent measures (i.e., percentage of bipartition sharing, number of bipartition comparisons, and memory usage) in addition to CPU time to explore practical algorithmic performance. Overall, our study concludes that HashRF performs better across the various performance measures than its competitor, PGM-Hashed. Thus, the HashRF algorithm provides scientists with a fast approach for understanding the evolutionary relationships among a set of trees.

**Keywords:** phylogenetic trees, RF distance, performance analysis.

## 1 Introduction

Given a collection of organisms (or taxa), the objective of a phylogenetic analysis is to produce an evolutionary tree describing the genealogical relationships between the taxa. Since the true evolutionary history for a set of taxa is unknown, many phylogenetic techniques use stochastic search algorithms to solve NP-hard optimization criteria such as maximum likelihood and maximum parsimony. During a phylogenetic search, thousands of candidate trees can be found, each representing a hypothesis of the true tree. The collection of candidate trees represent a tremendous data-mining for understanding the evolutionary relationships depicted among the trees. For example, trees could be clustered based on the topological distances between every pair of trees [1]. However, such clustering strategies require fast algorithms for computing the distance between every pair of trees in the collection of interest.

In this paper, we study two of the fastest algorithms—HashRF [2], [3] and PGM-Hashed [4]—to compute a  $t \times t$  Robinson-Foulds (RF) distance [5] matrix in  $O(nt^2)$  time. Figure 1 presents an overview of the RF matrix problem. Here,  $t$  is the number of trees in the collection and  $n$  is the number of taxa (or leaves) in



**Fig. 1.** Overview of computing the RF distance matrix. 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. For example, according to the RF matrix, the number of bipartition that are different between trees  $T_1$  and  $T_2$  is 1.

each tree. Day [6] provided an optimal linear time algorithm for computing the RF distance between two trees by utilizing a special cluster representation of the trees. By using repeated applications of Day's algorithm, the RF distance matrix can be computed in  $O(nt^2)$  time. Although the PGM-Hashed algorithm by Pattengale, Gottlieb, and Moret has the same theoretical complexity as repeated applications of Day's approach, PGM-Hashed is much faster in practice [4],[3]. We compare the PGM-Hashed approach to an algorithm we developed called HashRF, which also has the same theoretical complexity of  $O(nt^2)$ .

Given that the RF matrix algorithms under investigation have the same theoretical complexity, what is to be gained from studying their actual performance in practice? The *novelty* of our study is that we explore algorithmic performance by: (i) varying the amount of shared evolutionary relationships among the  $t$  trees, (ii) counting the actual number of bipartitions compared, and (iii) considering the memory usage of the algorithms. Theoretical complexity does not provide the algorithm designer with such insightful information as it relates to the above criteria.

In this study, we use artificial trees to assess the performance of the RF matrix approaches. Although biological tree collections are preferable, their availability for the parameters of interest in this study are limited (see Section 3.2). For our artificial tree collections, the number of taxa,  $n$ , and number of trees,  $t$ , ranged from 128 to 2,048. The experimental results show that HashRF's performance decreases as the number of shared relationships among the trees increases. PGM-Hashed, on the other hand, shows the opposite performance. That is, PGM-Hashed

executes faster as the number of shared bipartitions increases. Overall, regardless of the number of evolutionary relationships shared among the trees, HashRF is about 1.2 to 13 times faster than PGM-Hashed depending on the level of bipartition sharing in the collection of trees. Our results also demonstrate that HashRF requires a smaller number of bipartition comparisons among the  $t$  trees than PGM-Hashed. Finally, we show that HashRF uses one-third of the memory required by its competitor.

Given that the grand challenge in phylogenetics is to infer *The Tree of Life*, which is estimated to contain 10 to 100 million taxa, significant reductions in the running time and storage requirements of RF matrix algorithms is necessary to handle the increasing size of evolutionary trees and the collections that contain them. Our experimental study shows that the HashRF algorithm provides scientists with a fast approach for computing the all-pairs RF distance between their collection of trees, which could lead scientists to understanding the evolutionary relationships among their collection of trees in new and exciting ways.

## 2 Computing the Robinson-Foulds Matrix

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:  $AB|CDE$  and  $ABD|CE$ . An evolutionary tree is uniquely and completely defined by its set of  $O(n)$  bipartitions.

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:

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

Figure 1 depicts how the RF distance between two 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,  $d_{RF}(T_1, T_2) = 1$ .

In this paper, we are interested in computing the *all-to-all RF distance* or *RF distance matrix*. Given a set of  $t$  input trees, the output is a  $t \times t$  matrix of RF distances. Here, the matrix represents the topological distances between every pair of trees. Our work assumes that the input trees are binary. Hence, the largest possible RF distance between two binary trees is  $n - 3$ .

## 2.1 Bipartition Representations

For each tree in the collection of input trees, we find all of its bipartitions (internal edges) by performing a postorder traversal. In order to process the bipartitions, we need some way to store them in the computer's internal memory.

An intuitive bitstring representation requires  $n$  bits, one for each taxon. Consider Figure 1. The first bit is labeled by the first taxon name (e.g., taxon  $A$ ), the second bit is represented by the second taxon (e.g., taxon  $B$ ), etc. We can represent all of the taxa on one side of the tree with the bit '0' and the remaining taxa on the side of the tree with the bit '1'. Consider the bipartition  $ABE|CD$  from tree  $T_4$ . This bipartition would be represented as 11001, which means that taxa  $A$ ,  $B$ , and  $E$  are one side of the tree, and the remaining taxa are on the other side. Here, taxa on the same side of a bipartition as taxon  $A$  receive a '1'. For each tree in the collection of unrooted input trees, we arbitrarily root it. We find all of its bipartitions by performing a postorder traversal of each tree while performing an *OR* operation to the bitstrings of an internal node's (parent's) children.

The PGM-Hashed RF matrix algorithm [4] uses a compressed  $k$ -bitstring. Each input taxon is represented by a random  $k$ -bitstring, where  $k < n$ . Similarly to the  $n$ -bitstring case, all bipartitions are found by performing a depth-first search traversal of the tree. However, the bitstrings of an internal node's (parent's) children are *exclusive-OR*'ed together in this representation. One consequence of using a compressed bitstring is that there is a possibility that two different bipartitions may in fact be represented by the same compressed bitstring. If this happens, then the resulting RF matrix will be incorrect. Pattengale et al. show that the probability of colliding compressed bitstrings decreases exponentially with the number of bits chosen for representing the bitstrings [4].

## 2.2 HashRF

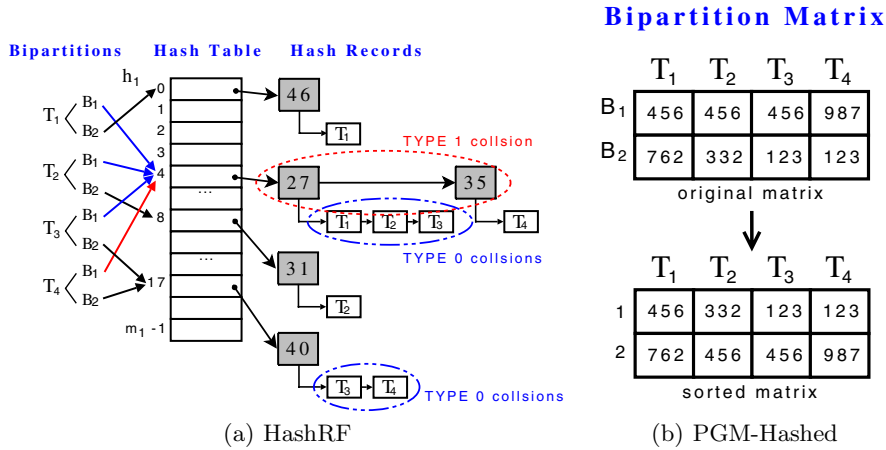
Figure 2(a) provides an overview of the HashRF algorithm, which runs in  $O(nt^2)$  time. Each input tree,  $T_i$ , is traversed in post-order, and its bipartitions are fed through two hashing functions,  $h_1$  and  $h_2$ . Hash function  $h_1$  is used to generate the location needed for storing a 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.

Similarly to Amenta et al. [7], Our  $h_1$  and  $h_2$  universal hash functions are defined as follows.

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

$$h_2(B) = \sum b_i s_i \bmod m_2 \quad (3)$$

$m_1$  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,



**Fig. 2.** Overview of the RF matrix algorithms under study. 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. (b) Each unique bipartition in the tree is represented by a unique integer value. Both the original matrix and its sorted version, which is then used to compute the RF distance matrix, are shown.

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. By using an implicit representation, we can avoid sending the  $n$ -bitstring representations to our hashing functions. 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,  $h_1(B_i) = h_1(B_j)$ . *Type 2* collisions occur when  $B_i$  and  $B_j$  hash to the same location in the hash table with the same bipartition IDs (BIDs). In other words,  $h_1(B_i) = h_1(B_j)$  and  $h_2(B_i) = h_2(B_j)$ . If a Type 2 occurs, the result output matrix will be incorrect. The probability of an incorrect answer is  $O(\frac{1}{c})$ , where  $c$  can be made arbitrarily large. Since  $c = 1,000$  in our experiments, there is a 0.001% chance that our HashRF algorithm will return an incorrect result.

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 index (BID) node. HashRF uses a  $t \times t$  dissimilarity matrix,  $D$ , to track the number of bipartitions that are different between all tree pairs. In the case of binary trees, the  $D_{i,j}$  entries are initialized to  $n - 3$ , the 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  $(ABC|DE$  from Figure 1). 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  $j$ .

### 2.3 PGM-Hashed

Pattengale, Gottlieb, and Moret [4] develop an  $O(nt^2)$  algorithm that uses  $k$ -length bitstrings to represent each tree's bipartitions. In their paper, Pattengale et al. describe a number of exact and approximate algorithms to compute the RF distance matrix. Since we focus strictly on exact approaches, we study Pattengale et al.'s Hashed algorithm, which we call PGM-Hashed.

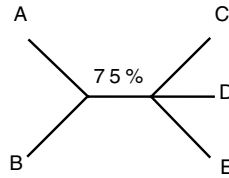
The algorithm starts by assigning a 64-bit integer random number to each taxon and using the XOR accumulator to combine the taxa numbers to represent the bipartition found during depth-first search traversal. Since each binary tree contains  $n - 3$  bipartitions, the entire set of bipartitions collected from the  $t$  trees are stored in a  $(n - 3) \times t$  two-dimensional array (or bipartition table). Entry  $(i, j)$  in the table represents the integer (converted from the 64-bitstring) representing bipartition  $i$  from tree  $j$  (see Figure 2(b)). Although the PGM-Hashed does not explicitly use a hash table, it is considered a hashing approach because different bipartitions may be represented with the same 64-bit integer.

Once the  $(n - 3) \times t$  bipartition table is constructed, the RF distance matrix is computed. Each of tree  $j$ 's bipartitions (i.e., column  $j$  in the bipartition table) are sorted since they are stored as integer values. After the sort, the RF distance between the trees is computed. For each pair of trees  $T_i$  and  $T_j$ , two pointers  $p$  and  $q$  are used to compare the bipartitions of  $T_i$  and  $T_j$ , respectively. If the bipartition pointed to by  $p$  is equal to the one referred to by  $q$ , then both pointers are incremented which means they have the same bipartition. However, if the bipartitions are different, a difference counter is incremented, and either the  $p$  or  $q$  pointer is incremented appropriately. To get the RF distance, the value of the difference counter is subtracted from  $n - 3$  since we are assuming binary input trees.

## 3 Our Collection of Evolutionary Trees

### 3.1 Overview

We test the performance of the HashRF and PGM-Hashed algorithms by using tree collections that share varying number of bipartitions between them. We



**Fig. 3.** Majority consensus tree for the input trees shown in Figure 1. The bipartition weight implies that 75% of the  $t$  trees must have the bipartition  $AB|CDE$ .

measure the amount of sharing among the  $t$  trees in a collection by the resolution rate,  $r$ , of the resulting majority consensus tree, which is one of the most popular consensus tree techniques used in a phylogenetic analysis. Majority trees contain bipartitions that appear in more than half of the  $t$  input trees. In Figure 3, the majority tree has a resolution rate of 50%. That is, for five taxa there are  $n-3$  (or 2) possible bipartitions in the resulting phylogenetic tree. However, the majority tree only has 1 of the 2 possible bipartitions. A 0% resolved tree represents a star whereas a 100% resolution rate denotes a binary tree. Larger resolution rates denote more shared bipartitions among the input trees.

### 3.2 Motivation for Using Artificial Trees

Our objective is to assess the RF matrix algorithms on large tree collections as a function of the number of taxa,  $n$ , the number of trees,  $t$ , and the resolution rate,  $r$ . In this paper, we create artificial tree collections to provide the diverse input trees we require to evaluate the algorithms. (Although their tree generation approach differs from ours, we note that Pattengale et al. also use artificial trees in their work.) There are a few large biological tree collections available, but they are often limited in one or more of the input parameters of interest. For example, we evaluate the performance of RF matrix algorithms on several biological tree collections provided to us by life scientists, where we found that HashRF is the best overall algorithm followed by PGM-Hashed [3]. The biological tree collections used contained trees with less than 600 taxa and the trees were quite similar since majority tree resolution rates were above 85%. Unfortunately, the biological tree collections are too limited (e.g., we are interested in bigger collections with more diverse bipartition sharing among the trees) for the objectives of interest in this paper. Our experiments are designed to understand how the algorithms perform under very diverse conditions of  $n$ ,  $t$ , and  $r$ . These results will then allow us to predict how the algorithms will respond as more biological tree collections become available for post-processing analysis.

### 3.3 Creating Artificial Tree Collections

To create our artificial tree collections, we used `apTreeshape` [8]—a R package for the simulation and analysis of phylogenetic tree topologies—to create a random,

Yule model tree consisting of  $n$  taxa. Next, we transform this tree into a  $r\%$  resolved multifurcating tree by randomly removing bipartitions. We take our  $r\%$  resolved tree and use it to generate  $t$  input trees. The resolved tree represents the majority consensus tree,  $T_{r\%}$ , of interest. Each bipartition in tree  $T_{r\%}$  is given a weight in the interval  $(50\%, 100\%]$ , which represents the percentage of the  $t$  trees that have that bipartition (see Figure 3).

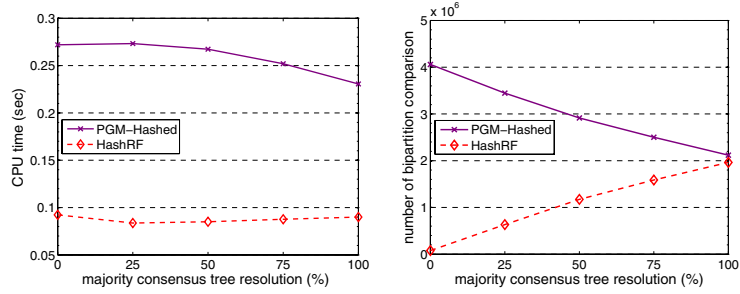
Once all of the bipartitions from the majority tree  $T_{r\%}$  have been distributed, each of the  $t$  trees is constructed. For each tree  $T_i$ , where  $1 \leq i \leq t$ , we construct tree  $T_i$  with the bipartitions that have been distributed to it. After the construction, any remaining multifurcating nodes are randomly resolved into binary nodes. These randomly resolved bipartitions (non-majority bipartitions) are then distributed to  $\lfloor p \rfloor$  trees, where  $1 < p \leq 0.50t$  and  $i < p \leq t$ . We distribute the non-majority bipartitions to the remaining trees in order to increase the amount of sharing among them in the tree collection. The above process is repeated five times for each  $n, t$ , and  $r$ . Thus, our plots show the average performance on our artificial datasets.

## 4 Experimental Results

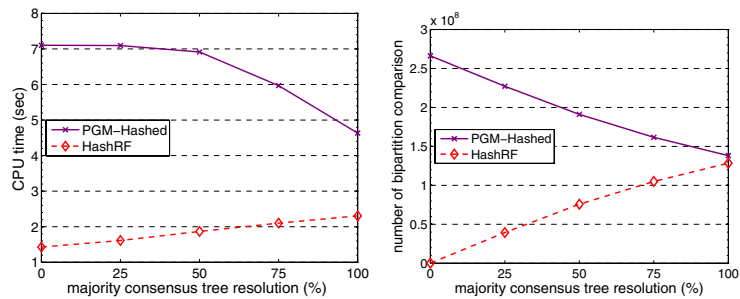
We ran a series of experiments to study the performance of HashRF and PGM-Hashed for computing the all-to-all RF distance problem across varying number of taxa ( $n$ ), trees ( $t$ ), and bipartition sharing ( $r$ ). We obtained the source code for PGM-Hashed from the authors. For HashRF, we pick a prime number to represent the size of our hash table,  $m_1$ , which is the smallest prime number bigger than  $O(tn)$ , the total number of bipartitions in the collection of trees. For  $m_2$ , we choose the smallest prime number larger than  $c \cdot tn$ , where  $c$  is chosen to be 1,000. 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.

Figure 4 shows the actual CPU time performance and bipartition comparison counts of the HashRF and PGM-Hashed algorithms on four of our artificial tree collection datasets as a function of the consensus tree resolution rate. CPU time includes the time to traverse the input trees, insert each tree's bipartitions into the hash table (HashRF) or bipartition table (PGM-Hashed), and compute the resulting RF distance matrix. Counting the number of bipartition comparisons to compute the RF distance matrix comes into play once all bipartitions have been collected and organized into the appropriate data structure used by the RF matrix algorithm.

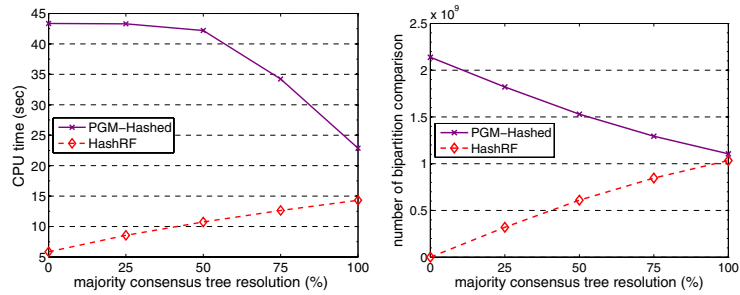
HashRF and PGM-Hashed show contrasting results in how they perform under different levels of bipartition sharing. Interestingly, the plots show that counting the number of bipartition comparisons is an effective measure for obtaining insights about algorithmic behavior since the trends shown by CPU time and bipartition comparison counts match very well. HashRF is the best overall performer both in



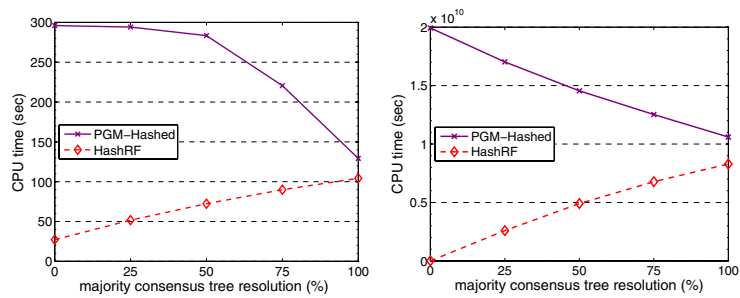
(a) 128 taxa, 128 trees



(b) 512 taxa, 512 trees

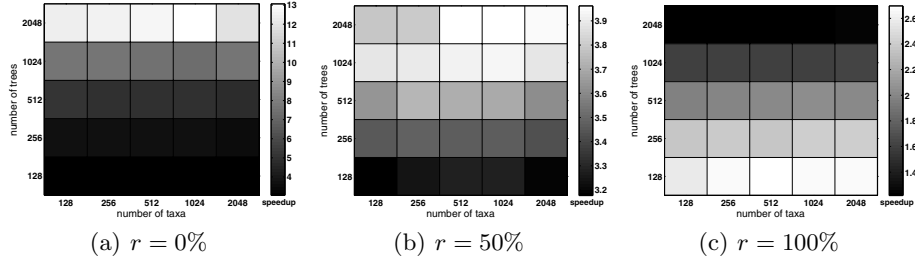


(c) 1024 taxa, 1024 trees



(d) 2048 taxa, 2048 trees

**Fig. 4.** RF matrix algorithms performance on four of our tree collections. The scale of the y-axis is different for all the plots.

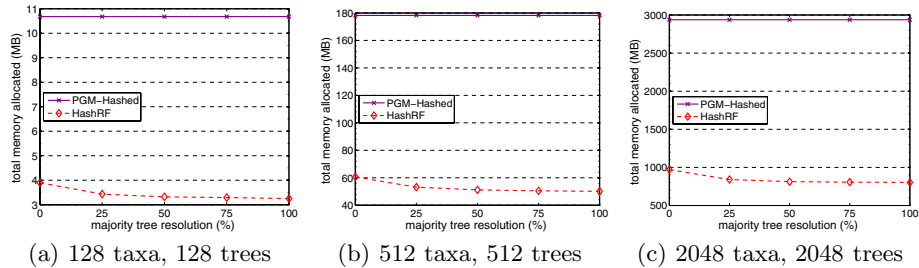


**Fig. 5.** Heatmaps depicting the speedup of HashRF over PGM-Hashed for various levels of bipartition sharing. The scale of the y-axis is different for all the plots. *For best results, please view electronically.*

terms of actual CPU time and number of bipartition comparisons performed. In Figure 4, the worst case for HashRF is when many bipartitions are shared, which is depicted with increasing consensus tree resolution rates. HashRF’s performance gets worse with increased bipartition sharing as a result of processing longer linked lists of trees in the hash table to compute the RF distance matrix. PGM-Hashed, on the other hand, gets faster as the amount of bipartition sharing increases. In PGM-Hashed, for each pair of trees  $T_i$  and  $T_j$ , two pointers  $p$  and  $q$  are used to compare the bipartitions, which are in sorted order based on their integer values. Two trees with identical bipartitions result in  $n - 3$  comparisons to compute the RF distance between them. Two trees that do not share any bipartitions require  $2(n - 3) - 1$  (or  $2n - 7$ ) bipartition comparisons. Thus, PGM-Hashed runs faster as the similarity among the trees increases.

In Figure 5, we use heatmaps to explore the speedup in terms of CPU time of HashRF over PGM-Hashed at specific majority resolution rates,  $r$ . In the heatmap representation, each speedup value (or cell) in the  $5 \times 5$  matrix is represented as a color. Darker (lighter) colors represent smaller (higher) speedup values in terms of how much faster HashRF is compared to PGM-Hashed. The speedup of HashRF over PGM-Hashed ranges from about 1.2 to 13. The plots clearly show that with increasing  $r$ , HashRF’s speedup over PGM-Hashed decreases significantly. Again, this is a result of HashRF having longer chains of linked lists to process. Another interesting observation from the plots is that with high resolution rates (e.g.,  $r = 100\%$ ), HashRF performs worse as the number of trees increases.

Finally, Figure 6 shows the memory usage of the RF matrix approaches. We used version 3.3.0 of the Valgrind software package (<http://www.valgrind.org>) to obtain our results. HashRF uses about three times less memory than PGM-Hashed. Interestingly, HashRF memory usage decreases as the number of shared bipartitions increases. When there are many unique bipartitions, a bipartition index (BID) and tree index (TID) have to be stored for each bipartition. However, for shared bipartitions, a single BID is stored for a linked list of TIDs. Thus, for HashRF this translates into less memory consumption. For PGM-Hashed, the memory usage is essentially constant.



**Fig. 6.** Memory usage of the HashRF and PGM-Hashed algorithms. For the case of 2,048 taxa and 2,048 trees additional swap space beyond the 2 GB of physical memory on our platform was required. The scale of the y-axis is different for all the plots.

## 5 Conclusions and Future Work

Phylogenetic searches usually produce a large number of candidate trees, which present a huge data-mining challenge for understanding the evolutionary relationships between them. In our study, we empirically compare the performance of HashRF and PGM-Hashed—which are the two fastest RF matrix algorithms available—for computing the  $t \times t$  RF distance matrix. Given that large collections of real biological trees are not readily available, we characterize the behavior of our algorithms on artificial tree instances. The *novelty* of our work concerns our methodology for assessing the practical performance of these  $O(nt^2)$  RF matrix algorithms. Thus, algorithmic performance of HashRF and PGM-Hashed is evaluated based on (i) the number of shared bipartitions among the  $t$  trees, (ii) the number of bipartition comparisons, and (iii) the amount of memory used.

Our experiments show that although the performance of the algorithms are impacted significantly by the number of shared bipartitions, HashRF shows better performance in all cases. Hence, the constants involved in the running time analysis are smaller for HashRF than for PGM-Hashed. For high levels of bipartition sharing ( $r \geq 50$ ), HashRF can execute from 1.2 to 3.9 times faster than PGM-Hashed. When there is very little sharing ( $r = 0\%$ ), HashRF is over 13 times faster. By using the number of bipartition comparisons as an architecture- and implementation-independent substitute for CPU time, HashRF's good performance is not based on programmer skill or taking advantage of specific architectural features of the underlying platform. Furthermore, our experiments show that HashRF also requires less memory than its counterpart. Given HashRF's good performance on the wide-range of bipartition sharing provided by our artificial tree collections, it should perform quite well when confronted with diverse collections of biologically-based trees.

Our work can be extended in many different directions. In this paper, we assumed that the input trees are binary. We plan to extend our study by including datasets that include multifurcating evolutionary trees. Additional experiments will study much larger sets of trees since the goal of a phylogenetic is to reconstruct the *Tree of Life*, which is estimated to contain between 10 to 100 million

taxa. Finally, we are currently obtaining large collections of biological trees from life scientists to compliment the results shown here.

## Acknowledgements

Funding for this project was supported by the National Science Foundation under grants DEB-0629849 and IIS-0713618. The authors wish to thank Eric Gottlieb, Bernard Moret, and Nick Pattengale for providing the PGM-Hashed code.

## References

1. Hillis, D.M., Heath, T.A., John, K.S.: Analysis and visualization of tree space. *Syst. Biol.* 54(3), 471–482 (2005)
2. 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)
3. Sul, S.J., Williams, T.L.: HashRF: a fast algorithm for computing the Robinson-Foulds distance matrix. Technical Report TR-CS-2008-6-1, Department of Computer Science, Texas A& M University (2008), <http://www.cs.tamu.edu/academics/tr/tamu-cs-tr-2008-6-1>
4. Pattengale, N., Gottlieb, E., Moret, B.: Efficiently computing the Robinson-Foulds metric. *Journal of Computational Biology* 14(6), 724–735 (2007)
5. Robinson, D.F., Foulds, L.R.: Comparison of phylogenetic trees. *Mathematical Biosciences* 53, 131–147 (1981)
6. Day, W.H.E.: Optimal algorithms for comparing trees with labeled leaves. *Journal Of Classification* 2, 7–28 (1985)
7. Amenta, N., Clarke, F., John, K.S.: A linear-time majority tree algorithm. In: *Workshop on Algorithms in Bioinformatics. LNCS*, vol. 2168, pp. 216–227 (2003)
8. Bortolussi, N., Durand, E., Blum, M., Franois, O.: apTreeshape: statistical analysis of phylogenetic tree shape. *Bioinformatics* 22(3), 363–364 (2006)