Heterogeneous Compression of Large Collections of Evolutionary Trees

Suzanne J. Matthews

Abstract—Compressing heterogeneous collections of trees is an open problem in computational phylogenetics. In a heterogeneous tree collection, each tree can contain a unique set of taxa. An ideal compression method would allow for the efficient archival of large tree collections and enable scientists to identify common evolutionary relationships over disparate analyses. In this paper, we extend TreeZip to compress heterogeneous collections of trees. TreeZip is the most efficient algorithm for compressing homogeneous tree collections. To the best of our knowledge, no other domain-based compression algorithm exists for large heterogeneous tree collections or enable their rapid analysis. Our experimental results indicate that TreeZip averages 89.03 percent (72.69 percent) space savings on unweighted (weighted) collections of trees when the level of heterogeneity in a collection is moderate. The organization of the TRZ file allows for efficient computations over heterogeneous data. For example, consensus trees can be computed in mere seconds. Lastly, combining the TreeZip compressed (TRZ) file with general-purpose compression yields average space savings of 97.34 percent (81.43 percent) on unweighted (weighted) collections of trees. Our results lead us to believe that TreeZip will prove invaluable in the efficient archival of tree collections, and enables scientists to develop novel methods for relating heterogeneous collections of trees.

Index Terms—Phylogeny, heterogeneity, heterogeneous, compression, trees, collections, TreeZip

1 Introduction

Phylogenetic data is growing at an incredible rate. Modern ("next-gen") sequencing techniques greatly reduce the cost of molecular sequencing, enabling scientists access to larger pools of source genomes. High performance phylogenetic search programs such as MrBayes [1], [2], RAxML [3] and BEAGLE [4] can run longer and more comprehensive searches, producing larger collections of trees. As phylogenetic data continues to grow, the question of how best to archive the resulting tree collections rises to prominence. In other biological communities, compression techniques [5], [6], [7] offer some of the best solutions.

Scientists currently respond to the challenge of phylogenetic data storage by discarding the bulk of the trees returned by phylogenetic search and storing only the consensus tree in repositories such as Dryad [8] and TreeBASE [9]. We have previously established the value of saving entire tree collections [10], [11], but will not belabor the point here. Despite this reduction of data and the increasing cheapness of hard disk space, archiving and organizing these disparate analyses is no easy task.

Furthermore, it is not easy for researchers to easily relate and compare their experimental results with the work done by others. Phylogenetic analysis by definition is heterogeneous, with each community of researchers focusing on their own genus or family of interest. When genera and families are small, it is easy to visually compare competing hypotheses or analyses. However, as families and genera grow large, it becomes very difficult to compare competing (and overlapping) hypotheses of evolutionary relationships. As we move toward building the Tree of Life, analyses must be combined and compared to analyses performed on orders, classes, phylums and kingdoms. Automated methods for detecting and efficiently storing relationships contained in heterogeneous collections of evolutionary trees will be crucial for the success of such endeavors.

Previously, we introduced TreeZip [12], [13], a lossless compression algorithm and software package for phylogenetic tree collections. Our goal in designing TreeZip was to provide an efficient alternative to the Newick format [14], the most popular way to represent phylogenetic trees. In the Newick format, matching pairs of parentheses symbolize internal nodes of the tree. A key limitation of the Newick format is that there are $O(2^{n-1})$ Newick representations for a single tree with n taxa. Consider tree T_1 in Fig. 1. The Newick strings `(((A,B),C),(D,E));` and `((C,(A,B)),(E,D));` equally represent the relationships contained in T_1. Lastly, a collection of t trees over n taxa contains $O(nt)$ total evolutionary relationships. These relationships must be rediscovered and related to each other every time the Newick file is analyzed.

While general-purpose data compression techniques (e.g. 7zip) can reduce the storage requirements of trees, they cannot recognize (or leverage) domain-specific information contained in the Newick-formatted file. Thus 100,000 different, but equivalent Newick string representations of a single tree is viewed as 100,000 unique strings by 7zip. Furthermore, the compression process further obscures the evolutionary relationships contained in a tree collection. The compressed Newick file must be decompressed for analysis, losing all space savings gained through compression. Analyzing the

• *The author is with the Department of Electrical Engineering and Computer Science, United States Military Academy, West Point, NY 10996. E-mail: suzanne.matthews@usma.edu.*

Manuscript received 30 Dec. 2013; revised 25 Aug. 2014; accepted 19 Oct. 2014. Date of publication 3 Nov. 2014; date of current version 4 Aug. 2015.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TCBB.2014.2366756

1545-5963 © 2014 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

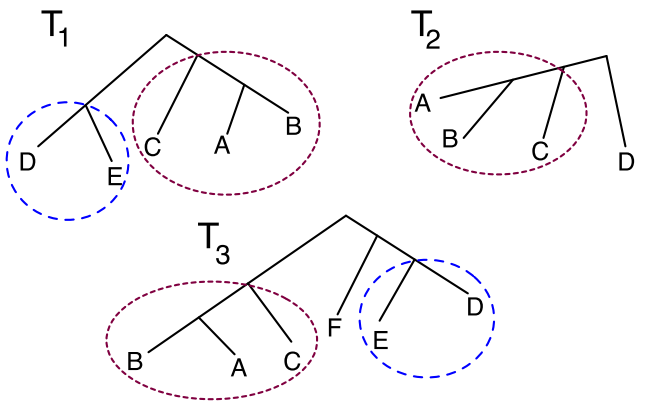


Fig. 1. Sample collection of three heterogeneous trees, consisting of common clades `(D,E)` and `((A,B),C)` (circled).

uncompressed Newick file requires the overhead cost of rediscovering the evolutionary relationships contained in it. Then, the file must be re-compressed to regain space savings. This whole process must repeat every time the file needs to be analyzed.

TreeZip employs domain-based compression to compactly store large tree collections, and is immune to the multiple representations of input Newick strings. The *novelty* of TreeZip is that it stores the evolutionary relationships contained in a tree collection uniquely in the compressed (TRZ) file, which itself is a unique 1:1 representation of the input tree collection. TreeZip pays for the $O(nt)$ cost required to collect evolutionary relationships exactly once and up-front through its compression algorithm. This enables subsequent operations to be performed on the TRZ file in close to real-time, without losing the space savings achieved though compression.

Our previous experimental results showed that TreeZip averages 96 percent space savings on unweighted collections, and 74 percent space savings on weighted collections of trees [12]. Compressing TRZ files with 7zip yields additional space savings, averaging around 99 percent on unweighted tree collections, and 95 percent on weighted tree collections [12]. A consequence of phylogenetic search is that the outputted trees share a high degree of evolutionary relationships. The more similar a set of trees, the better TreeZip's space savings. TreeZip is currently the most efficient method for compressing and storing large collections of phylogenetic trees.

However, our previous version of TreeZip was limited to homogeneous collections of trees. The trees outputted from phylogenetic search are homogeneous. That is, all trees in the collection are defined over a single, identical set of taxa. In heterogeneous collections of trees, every tree in the collection can have a unique set of taxa, or contain a subset of taxa present in other trees in the collection. For example, the trees in Fig. 1 are heterogeneous. Likewise, the analyses produced by separate groups of researchers may not be over the exact same group of organisms (despite significant overlap). The lossless compression of such datasets would therefore be beneficial to the scientific community.

In this paper, we discuss TreeZip's new heterogeneous algorithms and their performance. Our heterogeneous compression algorithm is centered around uniquely identifying common subtrees (instead of bipartitions) in a collection of trees. We simulated heterogeneous datasets of trees based on two existing biological datasets and two artificial collections of our creation. In our simulations, we varied the number of unique taxa sets contained in each tree collection.

Our results indicated that while performance does understandably degrade as the number of unique taxa sets in the collection approaches the number of trees, TreeZip's compression algorithm is capable of maintaining high space savings for smaller numbers of unique taxa sets. When the number of unique taxa sets is below 100, TreeZip is achieves average space savings of 72.69 percent and 89.03 percent on weighted and unweighted trees respectively. Furthermore, operations on heterogeneous collections of trees stored in TRZ files can be performed quickly. When combined with 7zip, TreeZip achieves average respective space savings of 81.43 percent and 97.34 percent on our weighted and unweighted datasets. This strongly indicates that TreeZip is effective at compressing and relating heterogeneous collections of trees.

The rest of the paper is organized as follows. Section 2 discusses the theory behind our heterogeneous compression algorithm. Section 3 discusses our experimental evaluation. We summarize our findings in Section 4.

2 Heterogeneous Compression

Consider the three trees in Fig. 1. The entire collection of trees is over the taxa set {A,B,C,D,E,F}. However, each tree contains a sub-collection of these six taxa. While each tree is distinct, there are clear relationships between them. For example, all three trees share clade `((A,B),C)`, while trees T_1 and T_3 share clade `(D,E)`. Our compression algorithm for heterogeneous trees uniquely identifies and stores such relationships, significantly reducing storage requirements and enabling fast analysis.

2.1 Representation of Heterogeneous Trees

Any phylogenetic tree can be decomposed into a set of bipartitions. A bipartition is a cut on an edge in the tree that partitions the taxa into two sets. Each tree with n taxa necessarily has $2(n-1)$ total bipartitions. While quartets [15] are another common building block for trees, bipartitions are preferable for compression. First, while a tree over n taxa has $O\binom{n}{4}$ quartets, it has $O(n)$ bipartitions. Second, while there are a greater number of total possible bipartitions than quartets ($O(2^n)$ vs $O\binom{n}{4}$), trees returned from phylogenetic search tend to be very similar to each other, and thus have high levels of bipartition sharing.

Our empirical analysis over homogeneous biological tree collections have confirmed that the set of total bipartitions over any particular *collection* of trees is very small, due to the high degree of similarity between the trees [12], [16]. The goal of heterogeneous compression of trees is to create a consensus of the work done by a group of scientists who operate over largely similar (but not identical) sets of taxa. We hypothesize that heterogeneous trees over a common taxa set returned from multiple phylogenetic searches should also contain high degrees of similarity, as all should have converged on similar solution spaces.

Bipartitions are not effective at representing heterogeneous collections of trees. For the rooted trees in Fig. 1, tree

T_1's non-trivial bipartitions are $ABC|DE$, $AB|CDE$, and $DE|ABC$. T_2 contains bipartitions $AB|CD$ and $ABC|D$. Lastly, T_3 contains bipartitions $ABC|DEF$, $AB|CDEF$, $DE|ABCF$ and $DEF|ABC$. Despite the high level of similarity between the underlying structure of the trees, there are no common bipartitions within our group of trees.

In order to support heterogeneous trees, we shifted from a bipartition model to a *subtree* model. A subtree is defined as the set of taxa residing directly *below* a cut on any internal edge or adjacent to the root. Each tree contains $2(n-1)$ subtrees. In Fig. 1, T_1 contains subtrees $\{ABC\}$, $\{AB\}$, and $\{DE\}$. T_2 contains subtrees $\{AB\}$, $\{ABC\}$, and $\{D\}$. Lastly, T_3 contains subtrees $\{ABC\}$, $\{AB\}$, $\{DE\}$, and $\{DEF\}$. Observe that all three trees in Fig. 1 contain subtrees $\{AB\}$ and $\{ABC\}$, and trees T_1 and T_3 also share the subtree $\{DE\}$. By viewing trees as a collection of subtrees rather than a collection of bipartitions, we are able to detect relationships within a heterogeneous collection of trees.

2.2 Compression Algorithm

For a given collection of trees, the set of unique taxa over the entire collection is first determined and ordered lexicographically. We refer to this global set of taxa as G, where $\mathcal{N} = |G|$. For each tree, we collect its subtrees via depth-first search. Each subtree is uniquely represented by a $\mathcal{N}$-length bitstring (B). Each taxon in our lexicographic ordering is assigned an index i and the corresponding bit in B (b_i) represents the presence or absence of that taxon in a particular subtree. If a taxon occurs in a particular subtree, the corresponding bit is set to 1. Otherwise, it is set to 0. For example, the subtree $\{AB\}$ in all three trees is represented by the bitstring `110000`, where the first bit denotes the presence of 'A' in the subtree and the second denotes the presence of 'B' in the subtree. We note that for any tree, the set of taxa $n \leq \mathcal{N}$ over which the tree is defined can be obtained by performing a bitwise OR operation over all the bitstrings derived from that tree. TreeZip's heterogeneous compression algorithm is $O(\mathcal{N}t)$, which is the time required to collect the $O(\mathcal{N})$ subtrees from each of the t trees in a collection.

2.2.1 Universal Hashing of Subtrees and Trees

We use universal hashing to quickly capture the set of unique subtrees within the tree. Let t represent the total number of trees in our collection of interest. For each taxon i from $1 \ldots \mathcal{N}$, two random numbers are generated, r_i and s_i. Each bitstring (B) representation of our subtrees is then fed into two hashing functions [16], [17] h_1 and h_2,

$$h_1(B) = \sum_{i=1}^{\mathcal{N}} b_i \times r_i \mod m_1,$$

$$h_2(B) = \sum_{i=1}^{\mathcal{N}} b_i \times s_i \mod m_2,$$

where m_1 is the first prime number greater than $\mathcal{N} \times t$ and m_2 is the first prime greater than $\mathcal{N} \times t \times c$, where c is a large constant number. Thus, each subtree is uniquely defined by $h_1(B).h_2(B)$ (where . represents concatenation). Our las-vegas randomized hashing strategy guarantees that $h_1(B_1).h_2(B_1) = h_1(B_2).h_2(B_2)$ iff $B_1 = B_2$. By storing an explicit bitstring representation of each subtree in our hash table, we can always determine if $B_1 = B_2$, guaranteeing us 0 percent probability of error. To eliminate the possibility of an error state, our code terminates and notifies the user to restart the program if such a collision is detected. This would occur due to repetition in our generated set of random numbers (r_i, s_i), and regenerating the random numbers would fix this problem. In practice, these collisions have never occurred. For each subtree, we store the set of ids of the trees that contain it. For weighted collections, we also store branch lengths.

Let k denote the number of unique subtrees in our set of t trees. We can represent each tree T as a k-bit bitstring (A), where each position $a_i \in A$ denotes a unique subtree in a collection. Ordering of subtrees do not matter, so long as the ordering is consistent for all trees. If T contains subtree i, then $a_i = 1$. Otherwise, $a_i = 0$. For each subtree i from $1 \ldots k$, two new random numbers are generated, o_i and p_i. We define two additional hashing functions [16] h_3 and h_4,

$$h_3(A) = \sum_{i=1}^{k} a_i \times o_i \mod m_3,$$

$$h_4(A) = \sum_{i=1}^{k} a_i \times p_i \mod m_4,$$

where m_3 is the first prime number greater than t and m_4 is the first prime number greater than $t \times c$. Thus, each unique tree in the collection is defined by $h_3(A).h_4(A)$. Our las-vegas randomized hashing strategy once again guarantees that $h_3(A_1).h_4(A_1) = h_3(A_2).h_4(A_2)$ iff $A_1 = A_2$ by storing the bitstring representation of each tree. Thus, we can always determine errors in our approach (see details above), and terminate the process if necessary.

2.2.2 The TRZ Encoding Scheme

Once the hash tables consisting of the unique subtrees and trees are collected, the last step is to write these contents to the TreeZip compressed (TRZ) file. One of the primary goals of the TRZ format is to be a compact representation that *highlights* the evolutionary relationships contained in a collection of trees. Storing encoded versions of these relationships is important, as it allows operations (e.g. computing the consensus) to be performed directly and quickly on the TRZ file, while reducing space requirements. As such, we eschewed binary compression schemes such as Huffman encoding [18] and LZW [19], which obscure the relationships in a collection of trees. Run-length encoding ensures reasonable space savings while enabling analysis to be performed directly on the TRZ file. We note that since the TRZ file is text, it can be further compressed with general-purpose text compression algorithms such as 7zip with little overhead.

The first three lines of the compressed TRZ file represent the global set of taxa names, the number of unique trees in the file, and the number of unique subtrees. If the file is heterogeneous, we add a 'H' flag on the line designating the unique set of trees. We first process the hash table that identified unique subtrees. Prior to encoding, subtrees are stored by the order of the number of 1s they contain, with ties

broken lexicographically. This ensures the 1:1 representation of a TRZ file to its collection of trees.

Afterward, we process each row in the hash table containing unique subtrees. Each row represents a separate line in the compressed file. There are three components (bitstrings, tree ids, branch lengths) to a TRZ line. The full details of TreeZip's encoding scheme can be found in [12]. Each bitstring is run-length encoded. Duplicate trees are removed from each row, and are encoded using TreeZip's encoding procedure. After the set of trees are encoded, the set of branch lengths for each tree is encoded and appended to the end of the row. At the end of the file, we encode the hash-table that identified unique trees in the collection. This table ensures that we will be able to rebuild the collection fully and correctly if decompression is required.

2.3 Computing Consensus and Other Extensible Operations

TreeZip stores trees more efficiently than Newick files. Furthermore, since the TRZ file stores only the unique set of subtrees in a tree collection, we can perform analysis on a TRZ file much quicker than than we can on a Newick file. Previously, we established TreeZip's equivalence to Newick files on homogeneous collections of trees and showed that operations on TRZ files are much faster than equivalent operations on Newick files [12]. By extending TreeZip's functionality to encompass heterogeneous collections, we've also extended TreeZip's ability to perform extensible operations on these collections.

Of greatest significance is TreeZip's ability to compute the strict or majority consensus tree of an encoded heterogeneous collection of trees, without any need for decompression. Unlike the consensus tree of a homogenous collection, the heterogeneous consensus tree may not contain the total set of $\mathcal{N}$ taxa in the collection. Consider the three trees in Fig. 1. TreeZip outputs `((A,B),C);` as the strict consensus for this collection, as these underlying subtrees appear in every collection. Likewise, TreeZip reports `(((A,B),C),(D,E));` as the majority consensus, as these underlying subtrees appear in at least two of the three trees. To the best of our knowledge, no other program exists that can compute the strict or majority consensus of a collection of trees over a heterogeneous taxa set.

Furthermore, TreeZip can directly compare collections of heterogeneous trees. Since the TRZ format is a 1:1 representation of a collection of trees, TreeZip can compute the total set of unique trees over two given collections (*union*), the set of unique trees that are contained in both collections (*intersection*), and the set of unique trees that are contained in one collection, but not the other (*set difference*). All these operations can be performed close to real-time without needing to decompress the TRZ file.

2.4 Decompression Algorithm

Since the TRZ file is designed to be an archive format for large collections of trees, it is not meant to be decompressed frequently. That said, TreeZip has a decompression routine that returns the set of trees in Newick format. The procedure is largely identical to that described in [12]. The TRZ file is decoded, and for each unique tree, the set of subtree bitstrings contained within them is stored. Each tree is then rebuilt sequentially, using the second table (which tracks duplicate trees) as reference.

To properly rebuild each tree, a "star" tree bitstring is used as the base. With the processing of each additional subtree, internal nodes are added, until the entire tree is rebuilt. In the case of homogeneous collections, this "star" tree is an n-length bitstring, where each bit is set to 1. For heterogeneous datasets, each tree can have some set of n taxa, where $n \leq \mathcal{N}$. Thus for each tree, it is necessary to calculate the corresponding "star tree". To do this, we initially populate an $\mathcal{N}$-length bitstring of all 0s for each tree. By performing an OR operation over all the subtrees contained in each tree, one gets appropriate star bitstring associated with each tree. One pass over the entire hash table is sufficient to generate the star bitstrings of all the trees in a collection.

3 Evaluation

Our implementation of TreeZip used in the following experiments is free to download at https://github.com/suzannejmatthews/treezip/. Experiments were conducted on a 64-bit Intel machine with two quad-core processors and 32 GB of RAM running Ubuntu Linux 12.04. TreeZip is written in C++ and compiled with g++ 4.6.3 with the -03 compiler option.

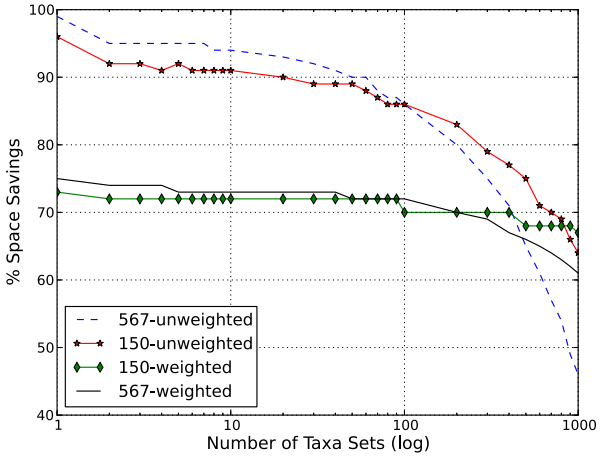
3.1 Description of Simulated Datasets

In order to build the Tree of Life, researchers must use a variety of different methods to collect and combine their analyses, which will have varying levels of heterogeneity. By studying how heterogeneity affects our approach, we gain clues on how best to combine and store such collections. Simulated datasets allow us to test different scenarios for how phylogenetic researchers may combine disparate tree collections.

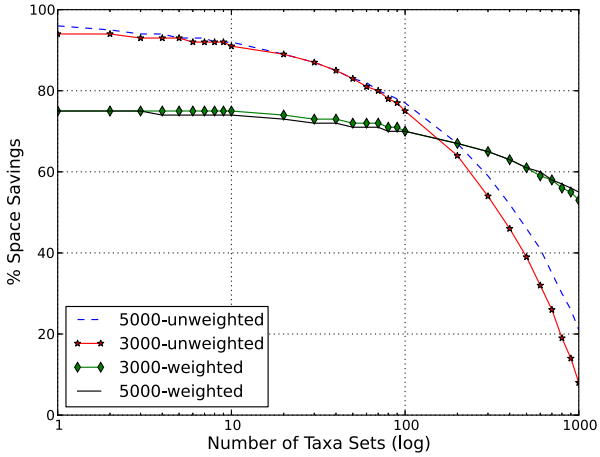
The compression quality of our algorithm is dependent on the number of unique subtrees in the collection. As the heterogeneity of a dataset increases, space savings is sharply reduced due to the increased size of the underlying set of unique subtrees. Simulated datasets allow us to precisely model and study the rate of this degradation.

We generate heterogeneous datasets by adding heterogeneity to homogeneous datasets. For each dataset, we randomly partition the total set of trees into u equally sized partitions, randomly remove a percentage of taxa from each partition, and recombine the partitions together. This yields a heterogeneous dataset with u taxa sets, $\mathcal{N}$ total taxa, and t trees. A *taxa set* is a unique set of taxa over which a collection of trees is defined. Consider Fig. 1 where each tree has a unique set of taxa, yielding three unique taxa sets in the collection. When $u = 1$, the collection is identical to the homogeneous source dataset. As u increases, the number of unique taxa sets and subtrees in a dataset also increases, resulting in greater levels of heterogeneity.

Our source homogeneous datasets include two existing biological tree collections and two artificial collections of our creation. In our experimentation, we considered trees with weighted and unweighted branches. Our unweighted collections are identical to our weighted ones, except that all branch lengths were removed.



(a) Biological heterogeneous collections of trees.



(b) Artificial heterogeneous collections of trees.

Fig. 2. TreeZip performance on heterogeneous trees.

We obtained biological datasets from two previously published Bayesian phylogenetic analyses that utilized MrBayes [20] . The first biological dataset consists of 20,000 unique trees obtained from an analysis of 150 taxa [21]. The weighted (unweighted) version of the Newick file for this dataset is 67 MB (16 MB). There are 1,318 (1,168) unique subtrees out of 5.9 million possible in the weighted (unweighted) collections. The second biological dataset consists of 33,306 trees obtained from an analysis of 567 angiosperm taxa [22]. Of the 33,306 trees, 33,169 are unique. The size of the weighted (unweighted) Newick file for this collection is 428 MB (105 MB). There are 3,011 (2,444) unique subtrees out of ≈ 37.7 million possible in the weighted (unweighted) sets.

To test TreeZip's performance on massive tree collections, we simulated two large artificial datasets that have similar majority consensus rates and proportion of subtrees to our previously described biological collections. The ratio of subtrees to taxa in the 150-taxa and 567-taxa datasets were 7.8 and 4.3 respectively. Using HashCS [23], the majority consensus rates for the 150-taxa and 567-taxa datasets were determined to be 85.7 and 92.5 percent respectively.

Our first artificial collection contains 3,000 taxa and 20,000 unique trees, and has a majority consensus rate of 85.7 percent and a subtree ratio of 7.6. The weighted (unweighted) version of the Newick file for this dataset is 1.4 GB (380 MB). There are 25,949 (22,949) unique subtrees out of ≈ 199 million possible in our weighted (unweighted) sets. Our second artificial collection contains 5,000 taxa and 20,000 unique trees and has a majority consensus rate of 92.5 percent and a subtree ratio of 4.4. The size of the weighted (unweighted) Newick file for this collection is 2.4 GB (647 MB). There are 27,004 (22,004) unique subtrees out of ≈ 332 million possible in our weighted (unweighted) sets.

3.2 Measuring Performance

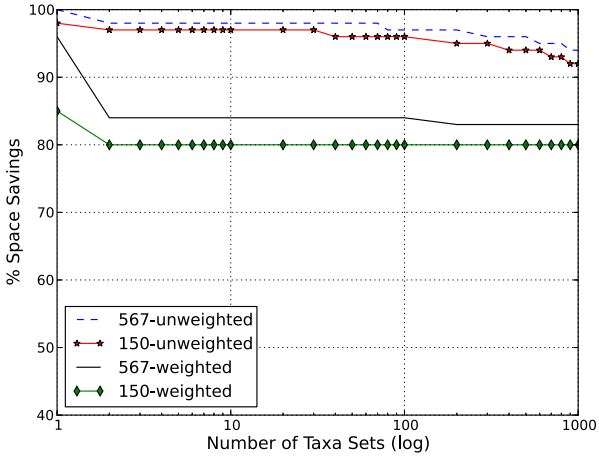
For each homogeneous dataset, we generated heterogeneous collections of t trees, and vary u from $1 \ldots 1,000$. For each heterogeneous dataset produced, we record the amount of space savings the TRZ file has over the corresponding Newick file. The space savings S is measured as $S = (1 - \frac{|TRZ|}{|Newick|}) \times 100$. Space savings of 0 percent means there is no reduction of file size. The goal for any compression algorithm is achieve space savings close to 100 percent. We also measure the running time of our compression, set operations, and decompression methods.

3.3 Space Savings Performance Results

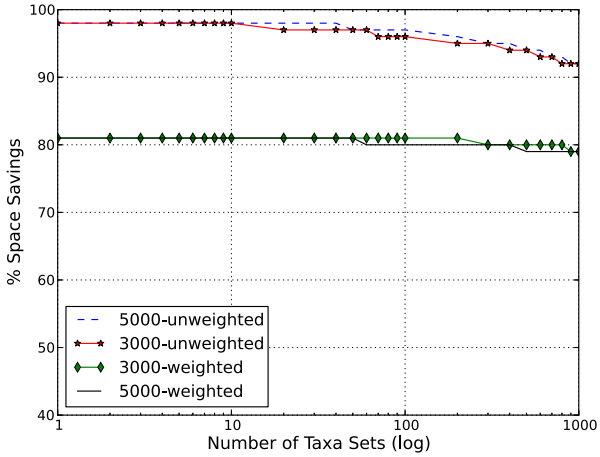
Fig. 2 shows the performance of TreeZip on our heterogeneous collections of trees. Figs. 2a and 2b depict the results on our biological and artificial datasets respectively. The x-axis denotes the number of unique taxa sets in the file. The y-axis denotes the amount of space savings the TRZ file achieves over the Newick file. Consistent with prior results [12], TreeZip achieves an average of 74.44 percent space savings on weighted datasets, and 96.43 percent space savings on unweighted data sets when $u = 1$, the homogeneous case. This discrepancy is caused by the presence of branch lengths.

As the number of unique taxa sets contained in a collection increases, the space savings degrades. As u increases to 100, average space savings of our unweighted biological datasets dips to 86 percent. In comparison, the average space savings of our unweighted artificial datasets reduces to 76 percent. However, when u is increased further to 1,000 the average space savings of our unweighted biological and artificial datasets drop to 55 and 14.5 percent respectively. This rapid decrease in space savings is due to the number of underlying subtrees exploding. Since our compression approach centers around identifying unique subtrees in a collection, performance degrades as the number of unique taxa sets approaches the number of trees in the collection.

The effect of subtree explosion is muted in our weighted datasets. As u increases to 100, our biological and artificial weighted datasets slightly decrease to 71 and 70 percent average space savings. However, as the number of unique taxa sets in the collection increases to 1,000, the average space savings for our weighted biological datasets dip to 64 percent. The average space savings for our artificial datasets decrease to 54 percent. These results are promising, as the majority of phylogenetic analyses are weighted. Even with high levels of heterogeneity ($u = 1{,}000$), the resulting TRZ file is at most half the size of the Newick file.



(a) Biological heterogeneous collections of trees.



(b) Artificial heterogeneous collections of trees.

Fig. 3. TreeZip+7zip performance on heterogeneous trees.

Despite the degradation in our larger datasets, we can still achieve additional space savings when we combine TreeZip with general purpose compression algorithms. Fig. 3 shows the additional space savings resulting from compressing the TRZ files with 7zip. Figs. 3a and 3b show the space savings on our biological and artificial collections respectively. TreeZip+7zip achieves average spaces savings of 96.29 and 96.3 percent on our unweighted biological and artificial datasets. For our weighted biological and artificial datasets, TreeZip+7zip averages 81.83 and 80.86 percent respectively.

3.4 Runtime Performance Results

Compression is a one-time cost TreeZip pays up-front to enable space savings and the high speed of subsequent operations. Due to the additional time needed to determine the global set of taxa (G), heterogeneous compression is on average 36.4 percent slower than its homogeneous counterpart. For our 150-taxa datasets, compression took on average 27.07 seconds (15.61 seconds) on our weighted (unweighted) collections. For our 567-taxa datasets, compression took on average 3.6 minutes (2.33 minutes) on our weighted (unweighted) collections. Processing our artificial datasets takes much longer, due to the sheer size of the $O(\mathcal{N}t)$ subtrees that are required to be collected from the trees. For our 3000-taxa dataset, compression took on average 23.78 minutes (21.86 minutes) on our weighted (unweighted) heterogeneous collections. Lastly for the 5000-taxa dataset, compression took on average 65.88 minutes (55.48 minutes) on our weighted (unweighted) heterogeneous collections.

TreeZip compression allows consensus trees and other operations to be cheaply computed. Fig. 4 shows the performance of the consensus operation on both our weighted biological and artificial heterogeneous collections as u is increased. We note that the consensus operation takes longer on weighted datasets. The x axis denotes the number of unique taxa sets in the file, while the y axis shows the average execution time in seconds (log scale). Regardless of the level of heterogeneity, a consensus tree can be computed in less than a second on our 150-taxa datasets and less than 5 seconds on our 567-taxa datasets. For our larger artificial datasets, the operation takes less than a minute to complete.

Set operations (union, intersection and set difference) compare two TRZ files and thus are dependent on the number of unique subtrees over both files. As the number of unique subtrees increase, the longer the compute time. For our 150-taxa and 567-taxa datasets, this does not affect the real-time nature of our set operations, which take less than a minute to compute. For our larger artificial datasets, set operations can take one to 3 minutes to complete.

TRZ files do not need to be decompressed in order to be analyzed (e.g. producing the consensus). However, decompression may be necessary if Newick files are required. For our 150-taxa datasets, decompression takes on average 38.65 seconds (32.08 seconds) on our weighted (unweighted) collections. For our 567-taxa datasets, decompression takes on average 5.74 minutes (4.81 minutes) on our weighted (unweighted) collections. For our 3000-taxa datasets, decompression took on average 33.64 minutes (24.49 minutes) on our weighted (unweighted) collections. Lastly for the 5000-taxa datasets, decompression took on average 84.47 minutes (57.73 minutes) on our weighted (unweighted) collections.

Maximum space savings can be achieved by combining TreeZip with general purpose compression algorithms such as 7zip. For our unweighted biological datasets, it takes less

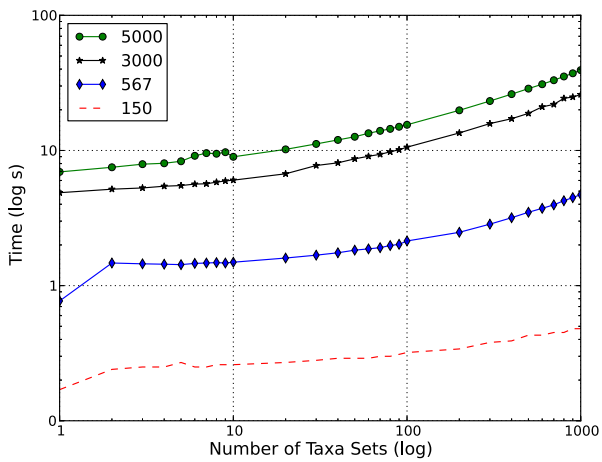


Fig. 4. Performance of consensus operations on weighted biological and artificial datasets.

than a second to compress a TRZ file with 7zip. For our weighted biological datasets, 11.28 seconds is required. On our unweighted and weighted artificial datasets, it takes 18.87 seconds and 2.12 minutes respectively to compress a TRZ file with 7zip. The TreeZip+7zip file (like all files produced by general-purpose compression algorithms) must be decompressed to a bare TRZ file prior to analysis.

4 Conclusions

Efficiently archiving and analyzing heterogeneous collections of trees is an open problem in computational phylogenetics. Research groups and tree repositories such Dryad and TreeBASE can greatly benefit from compressing and comparing related analyses over disparate taxa. General purpose compression methods, while capable of achieving space savings, are unable to leverage domain knowledge about tree collections. To analyze collections compressed with such programs, datasets will need to be decompressed, analyzed, and re-compressed to achieve original space savings, with this process repeating for subsequent analyses. The scientific community would greatly benefit from a compact, archival format for heterogeneous tree collections that enables fast analysis.

In this paper we discuss our implementation of heterogeneous compression in TreeZip, a lossless compression software package. To the best of our knowledge, no other domain-based compression algorithm exists for heterogeneous collections of trees. TreeZip achieves its space savings from storing all the evolutionary relationships and trees in a collection exactly once in the compressed (TRZ) file. For our datasets with moderate levels of heterogeneity (less than 100 unique taxa sets), TreeZip averages 89.03 percent space savings for our unweighted collections, and 72.69 percent space savings for our weighted collections.

Our experimental analysis on our larger artificial datasets indicate that TreeZip's compression algorithm takes upward of 20 minutes when the global set of taxa grows exceedingly large. This is only natural, as there are $O(\mathcal{N}t)$ relationships in a collection of t trees over $\mathcal{N}$ taxa. TreeZip pays this run-time cost up-front in order to enable the close to real-time performance of subsequent operations on the TRZ file, avoiding the need for decompression. Analysis operations (e.g. the consensus) on TRZ files leverage these pre-discovered relationships, and can be performed very quickly. In contrast, analysis operations on Newick files require that the total set of evolutionary relationships be rediscovered every time. TreeZip can be extended to include additional operations. We welcome suggestions from the scientific community and will gladly incorporate additional analytic features into TreeZip.

Lastly, the TRZ file can be further compressed with general purpose compression algorithms. Combining TreeZip with 7zip achieves average space savings of 97.34 percent (81.43 percent) on our unweighted (weighted) datasets. Our experimental analyses indicate that compressing TRZ files with 7zip is quick, making it ideal for long term storage where space savings is critical.

Future work will improve the performance of TreeZip's heterogeneous algorithms. As phylogenetic datasets get larger, TreeZip's running time becomes problematic. Other compression algorithms (such as 7zip) reduce their running time on large data by utilizing all the cores available on a system. Therefore, future work will concentrate efforts on developing new multi-core algorithms for TreeZip.

TreeZip can play a central role in the archival of phylogenetic tree collections. The goal in creating a heterogeneous tree collection is to generate a consensus of the work of various scientists. As scientists continue to piece together the Tree of Life, combining these disparate analyses into easy to use and compact formats becomes critical. TreeZip achieves space savings over raw Newick files, and enables faster analysis. These benefits will enable scientists to easily detect relationships in such collections, which may prove valuable in the creation of phylogenetic tree querying software and future supertree methods.

Acknowledgments

The author is grateful for the mentorship of T.L. Williams in the earlier stages of this work. This work was supported by the US National Science Foundation under Grants DEB-0629849, IIS-0713168, and IIS-1018785. This work was also supported in part by the Dissertation Fellowship program at Texas A&M University. The opinions in this paper are those of the authors and do not necessarily reflect the opinions of the US Department of Defense, or the U.S. Army.

References

[1] F. Ronquist and J. P. Huelsenbeck, "MrBayes 3: Bayesian phylogenetic inference under mixed models," *Bioinformatics*, vol. 19, no. 12, pp. 1572–1572, Aug. 2003.

[2] G. Altekar, S. Dwarkadas, J. P. Huelsenbeck, and F. Ronquist, "Parallel metropolis-coupled Markov chain Monte Carlo for Bayesian phylogenetic inference," *Bioinformatics*, vol. 20, pp. 407–415, 2004.

[3] A. Stamatakis, "RAxML-VI-HPC: Maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models," *Bioinformatics*, vol. 22, no. 21, pp. 2688–2690, 2006.

[4] D. L. Ayres, A. Darling, D. J. Zwickl, P. Beerli, M. T. Holder, P. O. Lewis, J. P. Huelsenbeck, F. Ronquist, D. L. Swofford, M. P. Cummings, A. Rambaut, and M. A. Suchard, "BEAGLE: An application programming interface and high-performance computing library for statistical phylogenetics," *Syst. Biol.*, vol. 61, no. 1, pp. 170–173, 2012.

[5] R. Giancarlo, D. Scaturro, and F. Utro, "Textual data compression in computational biology: A synopsis," *Bioinformatics*, vol. 25, no. 13, pp. 1575–1586, Feb. 2009.

[6] M. C. Brandon, D. C. Wallace, and P. Baldi, "Data structures and compression algorithms for genomic sequence data," *Bioinformatics*, vol. 25, no. 14, pp. 1731–1738, 2009.

[7] S. Deorowicz and S. Grabowski, "Compression of DNA sequence reads in FASTQ format," *Bioinformatics*, vol. 27, pp. 860–862, 2011.

[8] H. C. White, S. Carrier, A. Thompson, J. Greenberg, and R. Scherle, "The Dryad data repository: A Singapore framework metadata architecture in a DSpace environment," in *Proc. Int. Conf. Dublin Core Metadata Appl.*, 2008, pp. 157–162.

[9] W. Piel, M. Donoghue, and M. Sanderson, "TreeBASE: A database of phylogenetic information," in *Proc. 2nd Int. Workshop Species*, 2000, pp. 41–47.

[10] S.-J. Sul, S. J. Matthews, and T. L. Williams, "Using tree diversity to compare phylogenetic heuristics," *BMC Bioinformatics*, vol. 10, no. Suppl 4, p. S3, 2009.

[11] S. J. Matthews and T. L. Williams, "MrsRF: An efficient mapreduce algorithm for analyzing large collections of evolutionary trees," *BMC Bioinformatics*, vol. 11, no. Suppl 1, p. S15, 2010.

[12] S. J. Matthews and T. L. Williams, "An efficient and extensible approach for compressing phylogenetic trees," *BMC Bioinformatics*, vol. 12, no. Suppl 10, p. S16, 2011.

[13] S. J. Matthews, S.-J. Sul, and T. L. Williams, "A novel approach for compressing phylogenetic trees," in *Bioinformatics Research and Applications* (ser. Lecture Notes in Computer Science), vol. 6053. New York, NY, USA: Springer-Verlag, 2010, pp. 113–124.
[14] J. Felsenstein. (2009, Sep.). The Newick file format. [Online]. Available: http://evolution.genetics.washington.edu/phylip/newicktree.html
[15] J. Felsenstein, *Inferring Phylogenies*. Sunderland, MA, USA: Sinauer Assoc., 2003.
[16] S. J. Matthews, "Efficient algorithms for comparing, storing, and sharing large collections of evolutionary trees," Ph.D. dissertation, Dept. Comput. Sci. Engi., Texas A&M Univ., College Station, TX, USA, 2012.
[17] N. Amenta, F. Clarke, and K. S. John, "A linear-time majority tree algorithm," in *Proc. Workshop Algorithms Bioinformat.*, 2003, pp. 216–227.
[18] J. V. Leeuwen, "On the construction of Huffman-trees," in *Proc. 3rd Colloq. Automata, Lang. Programming*, 1976, pp. 382–410.
[19] M. R. Nelson, "LZW data compression," *Dr. Dobb's J.*, vol. 14, no. 10, pp. 29–36, 1989.
[20] J. P. Huelsenbeck and F. Ronquist, "MRBAYES: Bayesian inference of phylogenetic trees," *Bioinformatics*, vol. 17, no. 8, pp. 754–755, 2001.
[21] L. A. Lewis and P. O. Lewis, "Unearthing the molecular phylodiversity of desert soil green algae (chlorophyta)," *Syst. Biol.*, vol. 54, no. 6, pp. 936–947, 2005.
[22] D. E. Soltis, M. A. Gitzendanner, and P. S. Soltis, "A 567-taxon data set for angiosperms: The challenges posed by Bayesian analyses of large data sets," *Int. J. Plant Sci.*, vol. 168, no. 2, pp. 137–157, 2007.
[23] S.-J. Sul and T. L. Williams, "An experimental analysis of consensus tree algorithms for large-scale tree collections," in *Proc. 5th Int. Symp. Bioinformat. Res. Appl.*, 2009, pp. 100–111.

Suzanne J. Matthews is an Assistant Professor of Computer Science in the Department of Electrical Engineering & Computer Science at the United States Military Academy, West Point. She received her Ph.D. degree in Computer Science from Texas A&M University, and her M.S. and B.S. degrees in Computer Science from Rensselaer Polytechnic Institute. She was recognized as a Texas A&M University Dissertation Fellow during the final year of her Ph.D. studies, and served as a research assistant for three years prior. While completing her M.S. at Rensselaer, Dr. Matthews was recognized as a Master Teaching Fellow, and held research and teaching assistantships. As an undergraduate, she was awarded a summer research grant through the CRA-W Distributed Mentoring Program (now DREU). She is a member of the Upsilon Pi Epsilon and Phi Kappa Phi honor societies, and the Association of Computing Machinery. Her research interests lie in computational biology, high performance computing, data mining, experimental algorithmics, and version control systems.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/publications/dlib.