A compressed format for collections of phylogenetic trees and improved consensus performance

Robert S. Boyer, Warren A. Hunt Jr, Serita M. Nelesen

Department of Computer Sciences, The University of Texas, Austin, TX 78712, USA ;
{boyer, hunt, serita}@cs.utexas.edu

Abstract. Phylogenetic tree searching algorithms often produce thousands of trees which biologists save in Newick format in order to perform further analysis. Unfortunately, Newick is neither space efficient, nor conducive to post-tree analysis such as consensus. We propose a new format for storing phylogenetic trees that significantly reduces storage requirements while continuing to allow the trees to be used as input to post-tree analysis. We implemented mechanisms to read and write such data from and to files, and also implemented a consensus algorithm that is faster by an order of magnitude than standard phylogenetic analysis tools. We demonstrate our results on a collection of data files produced from both maximum parsimony tree searches and Bayesian methods.

1 Introduction

Producing a phylogeny for a set of taxa involves four major steps. First, comparative data for the taxa must be collected. This data often takes the form of DNA sequences, other biomolecular information or matrices of morphological data. Second, this data is aligned to ensure that comparable information is considered as input to the tree producing step. The third step is to produce candidate trees. There are many techniques for doing this including optimizing maximum parsimony or maximum likelihood criteria, or, more recently, by using Bayesian methods [7] [10]. These techniques rarely result in a single optimal tree. Instead, there are often many trees that a phylogeneticist would like to save for further processing such as consensus analysis, which is used to summarize the collection of trees. These post-tree analyses are the final step.

We have developed methods for storing and retrieving phylogenetic tree data, and using these methods we have implemented a consensus algorithm. Our approach permits very large data sets to be compactly stored and retrieved without any loss of precision. Also, our implementation of our consensus algorithm provides greatly increased performance when performing strict and majority consensus computations as compared to PAUP [18] and TNT [8].

Our system is called the Texas Analysis of Symbolic Phylogenetic Information (TASPI), and it is an experimental system, written from scratch. It is a stand alone tool for a few kinds of phylogenetic data manipulation. TASPI is written in the ACL2 [12] formal logic, where all operations are represented as pure functions. Using ACL2's associated mechanical theorem prover, it is possible to prove assertions about the TASPI system.

In this paper, we explain our representation of phylogenetic trees, and how this format reduces the storage requirement for a collection of trees. We also give an algorithm for computing strict and majority consensus trees that exhibits improved performance as compared to currently available software. Finally, we include an empirical study confirming our results.

2 Representation

Newick format [6] is the standard way of storing a collection of phylogenetic trees. Adopted in 1986, Newick is a parenthetical notation that uses commas to separate sibling subtrees, parentheses to indicate children, and a semicolon to conclude a tree. Newick outlines each tree in its entirety whether storing one tree, or a collection of trees.

On the other hand, TASPI capitalizes on common structure within a collection of trees. TASPI stores a common subtree once, and then each further time the common subtree is mentioned, TASPI references the first occurrence. This saves considerable space since potentially large common subtrees are only stored once, and the references are much smaller (for empirical results see Section 5).

There are two layers to the TASPI representation of trees. At a high-level, trees are represented as Lisp lists, similar in appearance to Newick, but without commas and semicolons. This is the format presented to the user of TASPI and on which user functions operate. At a low-level, the data are instead represented in a form that uses hash-consing [9] to achieve decreased storage requirements and improved accessing speeds. For ease of reference in Section 5, we call this the Boyer-Hunt compression.

Consider the following set of rooted trees in Newick format:

```
(a,((b,(c,d)),e));
(a,((e,(c,d)),b));
(a,(b,(e,(c,d))));
((a,b),(e,(c,d)));
```

The format of these trees presented to the user of TASPI is straightforward:

```
(a ((b (c d)) e))
(a ((e (c d)) b))
(a (b (e (c d))))
((a b) (e (c d)))
```

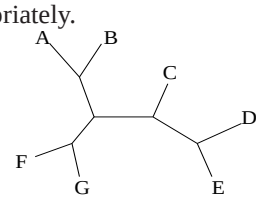
Notice that storing this set of trees involves restoring the subtree containing taxa `c` and `d` once for every tree. The Boyer-Hunt compression instead stores the `c-d` clade once, the first time it is encountered. If, subsequently, the `c-d` clade is encountered again, the first time is marked with "#n=" for the current value of a counter n that is incremented each time it is used. Then, instead of re-storing the `c-d` clade, a reference in the form "#n#" is stored in its place. This compression has parallels to the Lempel-Ziv data compression which is based only on characters seen so far [20]. The compressed version of the trees above is given below:

```
((A ((B #1=(C D )) E ))
(A (#2=(E #1#) B))
(A (B #2#))
((A B)#2#))
```

We use a technique sometimes called hash-consing, which ensures that no object is ever stored twice. In the context of phylogenetic trees, an object is a subtree, and consing is a tree constructor that joins subtrees. Hashing, put simply, is a technique that creates a table that allows for fast searches. In this case, hashing is used to quickly determine if a subtree was previously encountered. The format, using "#n=" and "#n#", is a standard read dispatch macro from Lisp programming [17].

Two subtleties remain to be addressed. First, though we will be presenting rooted trees in this paper, trees are not all rooted. In fact, most tree searching algorithms return unrooted trees since determining the root of a tree may itself be a computationally intensive problem [7]. Newick format does not distinguish between rooted and unrooted trees except through the use of auxiliary flags. By placing [&R] and [&U] just before the beginning of a tree, rooted and unrooted trees, respectively, are indicated. Without these flags, the onus is on the user to interpret the trees appropriately.

Second, Newick does not give a unique representation for a tree. Consider the tree on the right. There are many representations for this tree in both Newick and TASPI. Possible TASPI representations include:



`((F G) ((A B) (C (D E))))` and
`((C (E D)) ((B A) (G F)))`.

To ensure a unique answer in our computations, we order the output with respect to an ordering on the taxa. As far as we can tell, PAUP also does this. Thus, given an alphabetical ordering, we would order the tree above as `(A B ((C (D E)) (F G)))`.

3 Consensus Analysis

3.1 Background

Consensus trees are defined by Felsenstein as "trees that summarize, as nearly as possible, the information contained in a set of trees whose tips are all the same species" [7]. The idea of a consensus tree was first proposed by Day in 1972 [1], and quickly followed by other criteria for agreement between trees. In 1981, Margush and McMorris defined the majority rule trees as we know them today. They proposed this form of consensus as following best the "dictionary definition of consensus as 'general agreement' or 'majority of opinion"' [13]. It was also around this time that Sokal and Rohlf coined the term "strict consensus" [16].

Consensus methods return a single tree, or an indication that no tree meeting that method's requirements exists. The types of consensus include Adams, maximum agreement subtree, semi-strict, also called loose, combinable component or Bremer [7], greedy, local, and Nelson-Page. See Bryant [4] for an overview of various consensus methods and their interrelationships.

Two of the most common types of consensus trees are strict and majority. Both of these decide which branches in the input trees to keep, and then build a tree from the

resulting branches. Strict consensus requires that any branch in the consensus tree be a branch in every input tree, while a majority tree only requires that any branch in the consensus tree be a branch in at least some majority of the input trees. A threshold is a parameter to majority consensus that determines what percentage is to be used as a cutoff. Strict consensus is a special case of majority consensus; that is, it is a majority consensus with a threshold of 100%. Strict and majority consensus algorithms always return a tree, and have optimal O(kn) algorithms as described by Day [5] and Amenta et al. [2] (where k is the number of trees and n is the number of taxa).

3.2 Our Algorithm

We compute a consensus through a sequence of steps. We first read the source file containing the trees for which a consensus is to be computed. During the read process, we identify every subtree for which we have already read an identical subtree; thus, instead of creating a new data structure for the subtree just read, we reference the previously created subtree. We next create a mapping from all subtrees to every parent in which a subtree is referenced. Using this information, we compute the occurrence frequency of every subtree. Finally, after we have selected the subtrees that match our selection criteria, we construct the consensus answer. We give an example computation in Subsection 3.3.

In the following explanation, we use the notion of a "multiset", which is, intuitively speaking, a kind of set in which the number of occurrences count. More formally, one may regard a multiset as a function to the set of positive integers. If **A** and **B** are multisets, then **A** is a multisubset of **B** if and only if for each **x** in the domain of **A**, **x** is in the domain of **B** and **A(x)** <= **B(x)**.

For example, suppose **u**, **v**, and **w** are all distinct objects. Let **A** = <**u, 1**>, <**v, 2**> and let **B** = <**u, 2**>, <**v, 4**>, <**w, 5**>, then **A** is a multiset with one occurrence of **u** and two of **v**. Thus, **A(v)** = **2**. **A** is a multisubset of **B** because **A(u)** <= **B(u)** and **A(v)** <= **B(v)**.

One way to represent multisets is with lists in which the number of occurrences of an element in a list represents the number of times that the element is in the corresponding multiset. So for example, we may represent the example **A** above with the Lisp list **(u v v)**.

Several definitions will be useful.

- **tip**: a symbol or integer.
- **tree**: a tip or, recursively, a list of one or more trees.
- **fringe**: a list of all tips in a tree.
- **subtree**: If a and b are trees, then **a** is a *subtree* of **b** if and only if either (1) **a** is **b** or (2) **b** is a list and **a** is a subtree of a member of **b**.
- **proper subtree** If **a** and **b** are trees, **a** is a *proper subtree* of **b** iff **a** is a subtree of **b** and **a** is not **b**.
- **domain**: The *domain* of an association list (a list of key-value pairs) is the set of the keys of the members of the association list.
- **replete** An association list **db** is *replete* if and only if for all **t1** in the domain of **db**, (1) **t1** is a nontip tree and (2) if **t2** is a nontip proper subtree of **t1**, then **db(t2)**

is a list representing the multiset of all trees in the domain of **db** that have **t2** as a member, including **t1**. Note that the multiset **((a) (b) (a))** has the tree **(a)** as a member twice.

– **top level** A tree in the domain of a replete **db** is said to be *top level* if and only if it is a proper subtree of no member of the domain of **db**.

To compute the consensus, our algorithm proceeds by:

1. Producing a replete association list of all of the subtrees in the original input,
2. Counting the frequencies of the non-tip subtrees,
3. Collecting the subtrees that appear as often as the designated majority threshold, and finally,
4. Constructing the consensus tree.

Step one is accomplished by our function **replete-trees-list-top** which converts the original input list of trees into a replete association list (database). This replete database is a mapping from subtrees to every parent tree containing the subtree in question. Step two is performed by the function **fringe-frequencies** which counts the frequencies of every subtree fringe in the replete database by iterating through the replete database. Step three is collecting the subtrees that occur as often as the threshold. Finally, using this collection of subtrees, function **build-term-top** constructs the consensus answer.

Our function **replete-trees-list-top** takes a list **l** of non-tip trees no member of which is a proper subtree of another, such as a list of trees all with the same set of taxa. **replete-trees-list-top** returns a replete association list **db** such that (1) **x** is a member of the domain of **db** if and only if **x** is a member of **l** or is a non-tip proper subtree of a member of **l** and (2) if **x** is in the domain of **db**, then **db(x)** is an integer if and only if **x** is a member of **l** and **db(x)** is the number of times **x** occurs in **l**. For an example execution of **replete-trees-list-top**, see Subsection 3.3.

Function **fringe-frequencies** takes a list **l** of nontip trees such that no member of **l** is a proper subtree of any other member of **l** (such as that produced by **replete-trees-list-top**). **fringe-frequencies** returns a minimal length association list that pairs the fringe **fr** of each nontip subtree of each member of **l** with the number of occurrences in **l** of non-tip subtrees of members of **l** that have fringe **fr**.

By scanning through the resulting association list, we just pick out the subtrees that appear as often as the desired threshold. We have no need to store the actual number of times any specific subtree appears, we simply collect the desired subtrees (fringes) into a list.

The function **build-term-top** takes two arguments. The first argument is a sorted list **l** of the subtrees' fringes; **l** is sorted using a lexicographic (normalization) order that is based both on the internal tips and the size of the elements in each subtree. All the subtrees in **l** must appear in the consensus answer. The second argument is a normalization taxa list **tx**, that is used by our lexicographic ordering function so we can produce a unique representation of any subtree that itself includes more than one subtree. Remember, we represent each subtree as a list of subtrees, so to make the representation unique we sort members of each subtree. **build-term-top** constructs a consensus answer tree recursively by first building an answer of the first subtree of **l**. Once the first answer subtree is computed for the first element in **l**, any (sub-)subtrees

required to build the first subtree are "crossed out" from **l** that remain to be processed, and we continue with the next remaining element of **l** until no entries remain.

3.3 Example

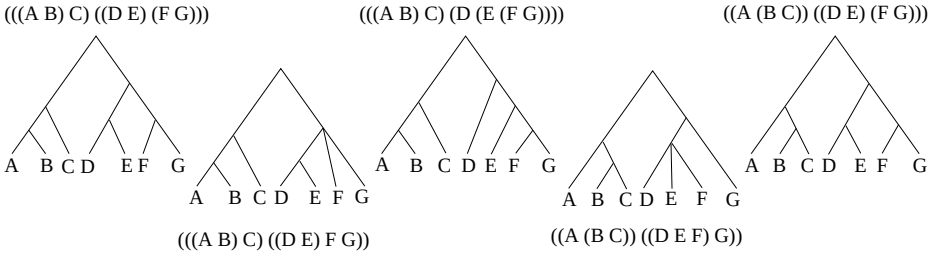


Fig. 1. A collection of trees together with their TASPI representations

Consider the five trees in Figure 1. The TASPI representation of these trees is the input to the function **replete-trees-list-top**. This function returns the following association list, where keys are in boldface:

```
((A B)((A B) C))
(((A B) C) ((D E) (F G))) . 1)
((D E)((D E) F G)
     ((D E) (F G)))
(((D E) F G) ((A B) C) ((D E) F G)))
(((A B) C) ((D E) F G)) . 1)
((A B) C)(((A B) C) (D (E (F G))))
        (((A B) C) ((D E) F G))
        (((A B) C) ((D E) (F G))))
((F G) (E (F G))
      ((D E) (F G)))
((E (F G)) (D (E (F G))))
((D (E (F G))) (((A B) C) (D (E (F G)))))
(((A B) C) (D (E (F G)))) . 1)
((B C) (A (B C)))
((D E F) ((D E F) G))
(((D E F) G) ((A (B C)) ((D E F) G)))
(((A (B C)) ((D E F) G)) . 1)
((A (B C))((A (B C)) ((D E) (F G)))
        ((A (B C)) ((D E F) G)))
(((D E) (F G))((A (B C)) ((D E) (F G)))
           (((A B) C) ((D E) (F G))))
(((A (B C)) ((D E) (F G))) . 1)
```

A subtree is the key for each element of the list, and the remainder of each entry (the values) is either (1) trees or subtrees in which the key appears, or (2) an integer representing the number of times this top level tree occurs in the input collection. Thus, this is a replete association list. This association list is now the input to the function **fringe-frequencies**, which produces this list:

```
((A B) . 3)     ((D E F). 1)
((D E) . 3)     ((A B C) . 5)
((F G) . 3)     ((D E F G) . 5)
((E F G) . 1)   ((A B C D E F G) . 5)
((B C) . 2)
```

This frequency list has each fringe from our replete association list, together with an integer. Remember, a fringe is simply a list of the tips in a tree, so we do not distinguish between the fringe from `(A (B C))` and the fringe from `((A B) C)`. The integer gives the number of trees that have a subtree with this fringe.

We are now prepared to sweep through this list and record the fringes that occur at least as often as the threshold for both a strict and majority consensus. In this example, for the strict majority we collect those fringes that occur 5 times, and for the majority, we collect those that occur at least 3 times. This gives us:

```
((A B C D E F G) . 5)
((D E F G) . 5)
((A B C) . 5)
```

and

```
((A B C D E F G) . 5)
((D E F G) . 5)
((A B C) . 5)
((F G) . 3)
((D E) . 3)
((A B) . 3)
```

Finally, the function **build-term-top** uses either the strict or majority fringes together with a normalization list such as `(A B C D E F G)` to create the strict and majority consensus trees. In this case we create `((A B C) (D E F G))` and `(((A B) C) ((D E) (F G)))`.

4 Experiments

4.1 Data Sets

We first obtained collections of phylogenetic trees from Dr. Usman Roshan and Dr. Tiffani Williams. These trees were created by PAUP and TNT performing maximum parsimony searches using biomolecular data sets. We have analyzed hundreds of these collections though we only present the results from ten collections. The results presented are representative of the full set. We also generated sets of trees using MrBayes [11] that had more taxa than either PAUP or TNT can even read; these data sets were created by the third author.

Table 1 gives characteristic information for each collection we present, namely, the numbers of taxa per tree, the number of trees in the collection, and the source of the collection.

Table 1. Data set statistics

Data Set Number	Data Set Name	Number of Taxa	Number of Trees	Source
1	Dom_2org	8506	47	Roshan
2	sRNA_mito	2587	369	Roshan
3	Will_Euk	2000	537	Roshan
4	Three567	567	2505	Williams
5	Actino	4583	301	Roshan
6	Ocho854	854	2505	Williams
7	John921	921	2505	Williams
8	t10000	500	10000	Roshan
9	Will2000	2000	2505	Williams
10	Mari2594	2594	2505	Williams
11	20000seqs	20000	1001	Nelesen
12	50000seqs	50000	1001	Nelesen

4.2 Methods

The files we obtained often contained comments about how the trees were generated, parsimony scores, or other output from their production. TASPI does not store this information, so we began by creating files that contained only the topological tree information so that we could accurately assess our compression.

Next, we created a suite of Perl scripts that take these original files and generate appropriate input files for PAUP and TNT. In each case, the taxa list is created from the first tree in the file, and the trees themselves are collected. Then, for PAUP, a Nexus file is produced with the taxa list, the trees, and a PAUP block containing the commands to compute consensus. Similarly for TNT, an appropriate input file is created with the taxa list, trees, and commands to compute consensus.

TASPI reads the source files directly. As with PAUP and TNT, TASPI can be run both interactively, where we submit one command at a time, or using an input file containing all commands needed for the desired computation.

Using PAUP, TNT and TASPI, we measured the time it took the software to read each collection, and the time needed to compute both a strict and majority consensus tree. For PAUP, we produced a strict consensus tree using its majority consensus command with percent set to 100 since the strict consensus command took considerably longer to do the same calculation. Also, by default, TNT does not include branches that are not well supported by the data used to create trees. However, we were not including any initial data other than the trees themselves, so we turned this feature off using the command *collapse notemp*.

Our experiments, where we were able to compare PAUP, TNT and TASPI, were all performed on an Intel Pentium 4 CPU 3.4 Ghz computer. However, for the two largest data sets, we used an AMD Opteron CPU 2.4 Ghz computer, which has similar computational performance, but more physical memory. Either computer produces the same compressed files. The largest files are too large to be read by either PAUP or TNT due to internal limitations on the number of taxa allowed in a tree.

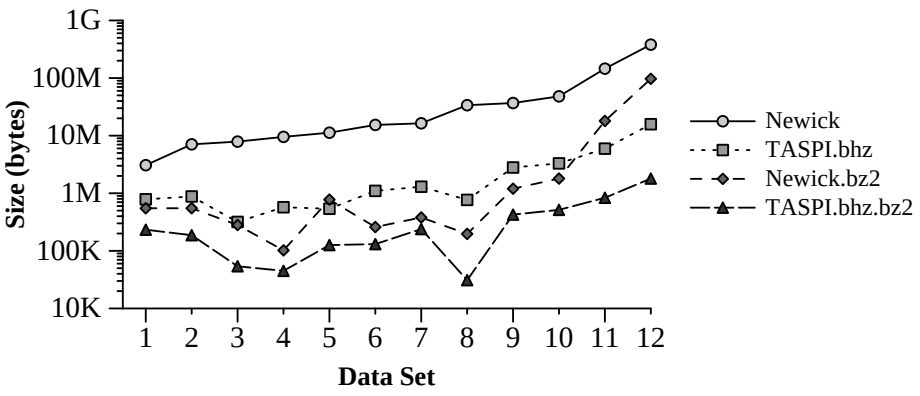


Fig. 2. Storage requirements

5 Results

The first major contribution of TASPI is the condensed format in which trees can be stored, while maintaining structural information. Figure 2 shows four sets of sizes for each of our benchmark data sets. The Newick data represents the size of the trees as they were given to us, after removing information that TASPI does not currently store (e.g. comments and branch lengths) and Newick.bz2 illustrates the size of the file after compression using the algorithm implemented in bzip2 [15]. TASPI.bhz displays the size of the file after compression using the Boyer-Hunt method. Notice that this file is still in ASCII, but with redundancies removed. Unlike most compression methods, all the information present in the original files is still immediately accessible, without a decompression step. Finally, TASPI.bhz.bz2 shows the size of the file if it is compressed using the Boyer-Hunt method and then bzip2 is applied.

Using the compressed TASPI format saves considerable memory space. For the data sets we present, the storage requirement for the TASPI format ranges from 2% of the storage requirement of Newick for the t10000 (data set 8) collection, up to 26% for the Dom_2org (data set 1) collection. Over all data sets, the compressed TASPI format uses just 5% of the storage requirement of the Newick format.

The amount of storage space saved is dependent on the amount of similarity between input trees. The more similarity between input trees (i.e. the greater the number of common subtrees) the more effective the compression. It is known in the phylogenetic community that trees derived from independent data sets are unlikely to have common structure [4]. However, it appears that collections of trees such as those we are presenting do have common structure since our compression was able to reduce the storage requirement for these collections of trees. Further, the greater the number of trees in the collection, the more likely there will be common structure.

It is readily apparent that bzip2 produces smaller files than the Boyer-Hunt compression on the smaller collections of trees, but for the very large data sets, the Boyer-Hunt compression produces smaller files than bzip2. Further, the Boyer-Hunt files are ASCII,

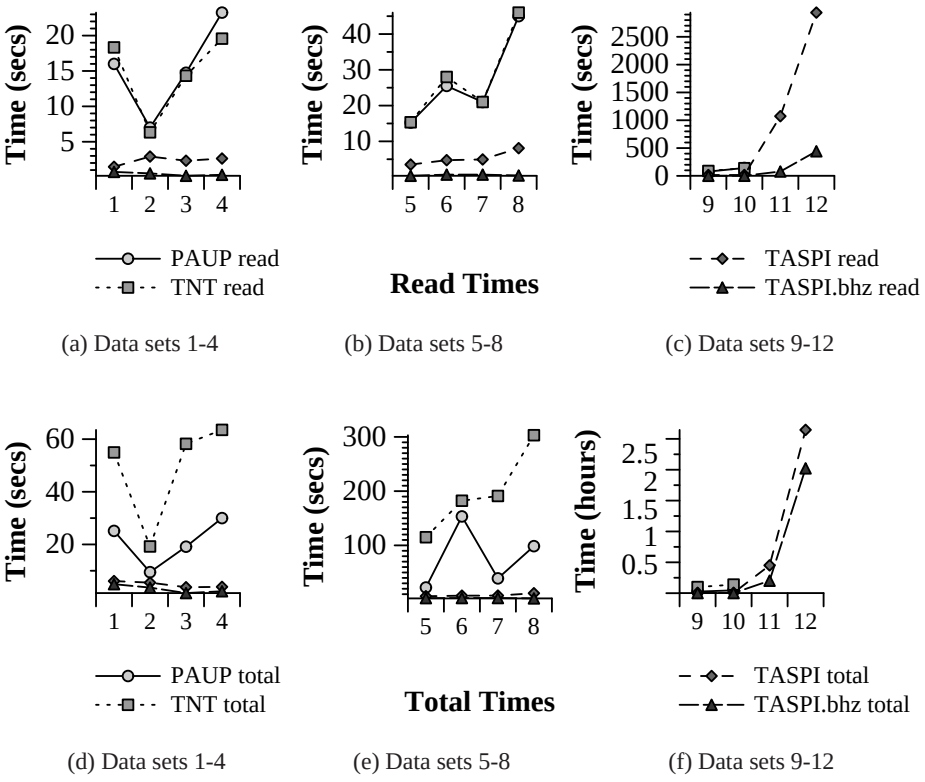


Fig. 3. Time to read a collection of trees (a-c) and compute strict and majority consensus trees with PAUP, TNT and TASPI (d-f)

and thus are ready to be used as input to analysis, such as consensus. If the data are not currently required as input to a post-tree analysis, compressed TASPI is even more useful. Boyer-Hunt files can be further compressed using bzip2 to produce even smaller files than those produced by using bzip2 on the original Newick files for sharing and transmission purposes. For our data sets, using the Boyer-Hunt compression together with bzip2 produces files that require 1% of the storage space of Newick.

The second major contribution of TASPI is its ability to read collections of trees quickly. Figure 3(a-c) shows average read times in seconds for each of our benchmark collections of trees. Notice that while reading trees with TNT or PAUP requires comparable times, reading the Boyer-Hunt compressed trees with TASPI is by far the fastest time for any collection. In fact, neither PAUP nor TNT is able to read the last two data sets. For the data sets which PAUP and TNT can read, reading the compressed TASPI format takes just 2% of the time to read the Newick files with PAUP. This means that loading these files takes more than 48 times longer when read with PAUP or TNT rather

than using TASPI to read their compressed counterpart. Even reading the source files is faster in TASPI than it is in either PAUP or TNT – using TASPI to read the Newick files takes just 16% of the time needed to read the same files with PAUP or TNT.

The third major contribution of TASPI is a consensus implementation with improved performance. Figure 3(d-f) shows the time to compute consensus with each of TASPI, TNT and PAUP. In each case, both a strict consensus tree and a majority consensus tree are computed. Notice that the time to compute consensus includes the time to read the collection of trees since the trees are the input to a consensus calculation. Thus, we show both the time to compute consensus when reading compressed trees and also the time when reading Newick trees.

In all cases, the result TASPI produces is identical to that produced by PAUP (when PAUP is able to read the input), but TASPI is faster. For the data sets PAUP and TNT can process that we present, using TASPI to compute consensus with input trees in compressed TASPI format requires 5% of the time it takes PAUP to compute consensus with input trees in Newick format. If we factor out the improved reading time, TASPI computes these consensus trees in about 10% of the time it takes PAUP to do the same computation.

6 Conclusion

In phylogenetics, the ability to store large numbers of trees is increasingly important. Bayesian methods, which use Monte Carlo Markov Chains, are visiting more trees than previous methods, and are growing in popularity. Biologists are also choosing to retain additional trees visited during a search. We have shown that our format provides decreased storage requirements, while maintaining data accessibility for further processing. Further, our format together with techniques like memoization allows for improved performance in post-tree analysis. We showed this using strict and majority consensus.

The use of post-tree analyses are also becoming more prevalent. Williams et al. propose using the rate of change of a consensus tree as a stopping criterion for heuristic maximum parsimony searches, which requires the computation of a consensus tree multiple times over the course of an analysis [19]. We have given a new format for collections of phylogenetic trees that would make this feasible. In addition, our replete database, the output of the first step in our consensus algorithm, provides a possible starting point for phylogenetic databases such as those proposed in [14].

In the future we hope to investigate the changes necessary to make our consensus algorithm incremental. This would allow online consensus analysis as proposed in [3]. We would also like to look at even larger collections of trees (larger both in number of trees and number of taxa) and consider application of our techniques to supertree methods.

Acknowledgment

This work was funded in part by an ITR from the National Science Foundation (EF-0331453).

References

1. E. N. Adams. Consensus techniques and the comparison of taxonomic trees. *Systematic Zoology*, 21:390–397, 1972.
2. Nina Amenta, Katherine St. John, and Frederick Clarke. A linear-time majority tree algorithm. In Gary Benson and Roderic D. M. Page, editors, *Proc. of the 3rd International Workshop on Algorithms in Bioinformatics (WABI 2003)*, volume 2812 of *Lecture Notes in Computer Science*, pages 216–227. Springer-Verlag, 2003.
3. Tanya Y. Berger-Wolf. Online consensus and agreement of phylogenetic trees. In I. Jonassen and J. Kim, editors, *Proc. of the 4th International Workshop on Algorithms in Bioinformatics (WABI 2004)*, volume 3240 of *Lecture Notes in Computer Science*, pages 216–227. Springer-Verlag, 2004.
4. David Bryant. A classification of consensus methods for phylogenetics. In M. Janowitz, F.J. Lapointe, F. McMorris, B. Mirkin, and F. Roberts, editors, *Bioconsensus*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science. DIMACS-AMS, 2001.
5. William H. E. Day. Optimal algorithms for comparing trees with labeled leaves. *Journal of Classification*, 2(1):7–28, 1985.
6. J. Felsenstein. The newick tree format. `http://evolution.genetics.washington.edu/phylip/newicktree.html`, 1986.
7. Joseph Felsenstein. *Inferring Phylogenies*. Sinauer Associates, Inc., 2004.
8. P.A. Goloboff, J.S. Farris, and K.C. Nixon. TNT (Tree analysis using new technology) (BETA) ver. 1.0. Published by the authors, Tucumán, Argentina, 2000.
9. E. Goto, T. Soma, N. Inade, T. Ida, M. Idesawa, K. Hiraki, M. Suzuki, K. Shimizu, and B. Philpov. Design of a lisp machine - flats. In *LFP '82: Proceedings of the 1982 ACM Symposium on LISP and functional programming*, pages 208–215, 1982.
10. David M. Hillis, Craig Moritz, and Barbara K. Mable, editors. *Molecular Sytematics*. Sinauer Associates, Inc., Sunderland, Massachusetts, 2nd edition, 1996.
11. J. P Huelsenbeck and F. Ronquist. MRBAYES: Bayesian inference of phylogeny. *Bioinformatics*, 17:754–755, 2001.
12. Matt Kaufmann, Pete Manolios, and J. S. Moore. *Computer-Aided Reasoning: An Approach*. Kluwer Academic Publishers, 2000.
13. T. Margush and F.R. McMorris. Consensus n-trees. *Bulletin of Mathematical Biology*, 43(2):239–244, 1981.
14. L. Nakhleh, D. Miranker, F. Barbancon, W.H. Piel, and M.J. Donoghue. Requirements of phylogenetic databases. In *Proceedings of the Third IEEE Symposium on Bioinformatics and Bioengineering (BIBE 2003)*, pages 141–148. IEEE Press, 2003.
15. Julien Seward. bzip2. `http://sources.redhat.com/bzip2/`, 2002.
16. Robert R. Sokal and F. James Rohlf. Taxonomic Congruence in the Leptopodomorpha Re-Examined. *Systematic Zoology*, 30(3):309–325, 1981.
17. Guy L. Steele. *Common Lisp the Language*, chapter 22.1.4. Digital Press, 2nd edition, 1990.
18. D. L. Swofford. *PAUP*: Phylogenetic Analysis Using Parsimony (and Other Methods) 4.0 Beta*. Sinauer Associates, Sunderland, Massachusetts, 2002.
19. Tiffani Williams, Tanya Berger-Wolf, Bernard Moret, Usman Roshan, and Tandy Warnow. The relationship between maximum parsimony score and phylogenetic tree topologies. Personal Communication.
20. J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23:337–342, 1977.