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

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

March 21, 2005

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 maximum parsimony searches.

1 Introduction

We have developed some improved methods for storing and retrieving phylogenetic data, and we have implemented a consensus algorithm with increased performance. Our approach permits very large data sets to be compactly stored and retrieved without any loss of precision. Our implementation of our consensus algorithm provides greatly increased performance when performing strict and majority consensus computations.

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 [8] [11]. 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.

In this paper, we propose a new format for storing trees which saves space, reduces time to read a collection of trees, and allows for better performance in computing strict and majority consensus trees. 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.

We begin with a survey of various consensus methods and introduce our representation. We then give an end-to-end example of a majority consensus analysis which we use to explain our algorithm for computing consensus. Finally, we explain our experiments and give their results.

2 Consensus Methods

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" [8]. There are many different kinds of consensus, each stressing a different commonality or difference between the input trees, and each deciding how to deal with conflicts between trees.

A conflict between two trees arises when a branch in one tree is not in the other. A branch separates a tree into two sections, creating a bipartition between the leaves in one part of the tree and the leaves in another. If the same bipartition is created by a branch in one tree and a branch in another, these trees are said to share that branch. If, however, there is no branch in the second tree that produces the same bipartition that is produced by a branch in the first tree, these two trees are said to be in conflict.

Consensus methods return a single tree, or an indication that no tree meeting the requirement of the method is possible. A consensus tree is created from the input trees based on some criteria. Two of the most common types of consensus trees are strict and majority. Both of these types of consensus 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 a majority of the input trees.

Edward N. Adams III was the first to propose and present a solution to the problem of "combining information from rival trees into one representative tree" [5] in 1972, and this form of consensus became known as Adams' consensus [1]. Since that time, many other versions have been presented and implemented.

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" [22].

Researchers continued to study the properties of consensus methods, as well as propose alternatives. In 1982 through 1985, McMorris and Neumann, in both joint and separate work, explored desirable properties of consensus methods. Unfortunately, they proved that it was impossible that a single method could simultaneously have all desirable properties explored. The properties they were interested in were whether a function was neutral, Pareto, dictatorial, and faithful [17]. They proved a theorem similar to a classical impossibility theorem of K. Arrow, namely that a consensus method can not be both neutral and Pareto without being dictatorial. McMorris and Neumann started by proving the impossibility for tree quasi-orders [16] and McMorris later proved it for undirected phylogenetic trees [14]. Neumann proposed a number of other consensus measures, such as the Durshnitt Rule and Cardinality Intersection Rule, which he believed were a better measure of similarity since they upheld faithfulness [17]. However, none of these faithful methods gained much popularity.

In 1983, McMorris, Meronk and Neumann parameterized the strictness of a tree [15] in an overview of consensus methods available at the time. This overview included Nelson consensus, which is based on replicated components. In this type of consensus, any clade (grouping of taxa) that is in at least two trees should be in the consensus, but notice that this process does not necessarily return a tree [15]. Page addressed this problem several years later (1990) by creating what is now known as Nelson-Page consensus, a version where a clade appearing more often is given greater weight, and the greatest weight tree is the consensus tree [5].

New forms of consensus continued be developed, including loose consensus

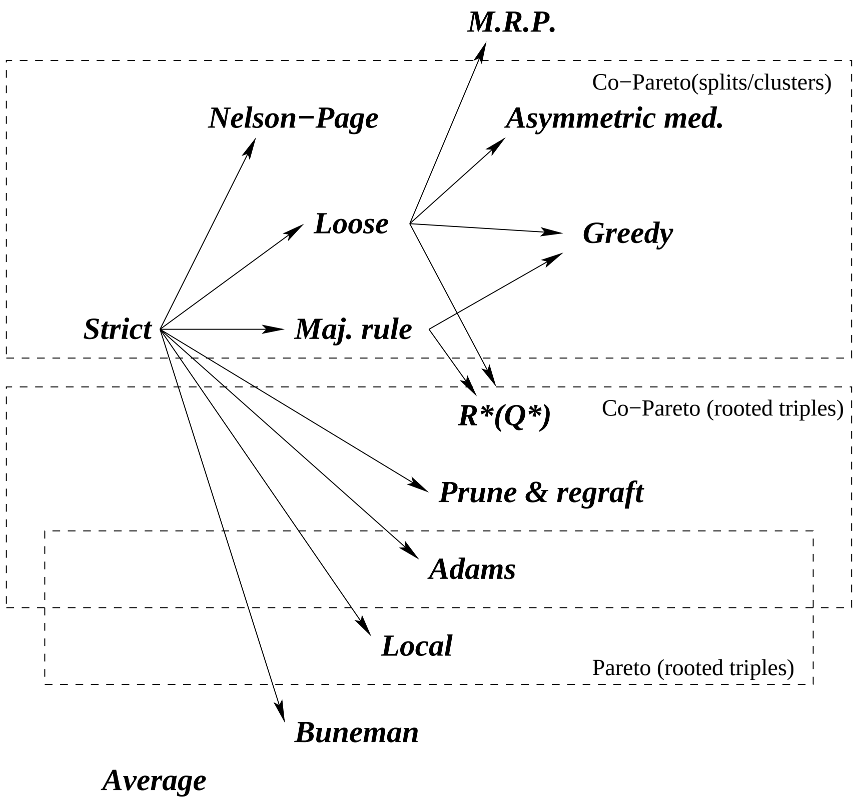


Figure 1: A classification of consensus methods from David Bryant [5]. There is an arrow from one method to another if every split in the consensus tree produced by the first method is contained in every consensus tree produced by the second method.

trees (also known as semi-strict, or combinable component) [18], local consensus [26] and asymmetric median trees [19]. Researchers continue to look for new methods, particularly for large and plentiful trees, since the accuracy of the consensus trees as compared to a known true tree appears to decrease with size and number of trees [4].

In terms of performance, in 1985 Day published a paper [6] showing how to compute a strict consensus tree in $O(kn)$ time where k is the number of trees in the input set, and n is the number of leaves in each of the trees. He outlined a process for rooted trees, and then showed how the unrooted case could be handled as well.

In 2003 a similar optimality constraint was developed for majority consensus. Amenta et al. were trying to generate majority trees more quickly for their interactive visualization system. In doing so, they achieved a linear time majority tree algorithm that makes use of randomization (that is, the expected running time is $O(kn)$) [2].

For many of the other forms of consensus, algorithm complexity results have also been achieved. While several of these have $O(kn^2)$ or other polynomial running times, many more are NP-hard for realistic situtations (for example, median consensus and asymmetric median trees) [26].

The latest development in consensus algorithms is in the area of online consensus. Berger-Wolf outlined online algorithms for computing strict and majority consensus trees which would allow an unchanging consensus to be used as a stopping criterion for heuristic tree searches [3].

It is important to note that all consensus methods, though they may return a tree, rarely return a tree that is most parsimonious, nor does the returned tree achieve the best likelihood score. Thus, there has been some debate about using consensus methods to infer phylogenies [5]. However, if only the appropriate information, namely that given by the criteria used to create the tree, is gleaned from the consensus tree, they are very useful.

3 Representation

Newick format is the standard way of storing a collection of phylogenetic trees. Adopted in 1986, Newick [7] 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 6).

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 [10] to achieve decreased storage requirements and improved accessing speeds. For ease of reference in Section 6, 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:

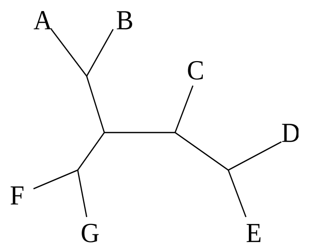
```
(a ((b (c d)) e))
(a ((e (c d)) b))
(a (b (e (c d))))
((a b) (e (c d)))
```

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 [27]. 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 [10], 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 [23].

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 [8]. 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))))`.

4 Our algorithm for consensus

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 4.1.

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 4.1.

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

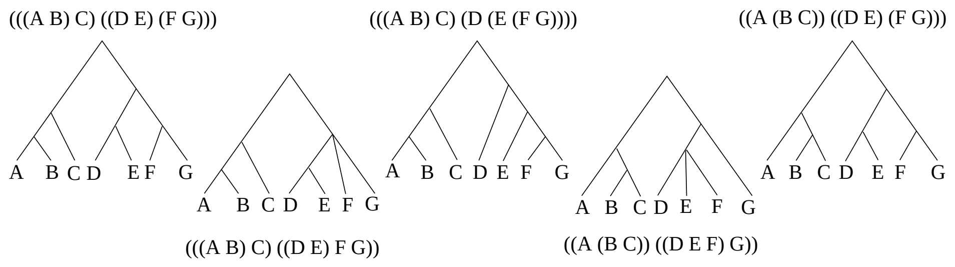


Figure 2: A collection of trees together with their TASPI representations

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.

4.1 Example

Consider the five trees in Figure 2. 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)
((A B C D E F G) . 5)               ((D E F G) . 5)
((D E F G) . 5)                     ((A B C) . 5)
((A B C) . 5)               and     ((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)))`.

5 Experimental Methodology

5.1 Data Set Collection

In order to see how well TASPI performs, as compared to currently available programs, we searched for collections of trees. We started with trees generated by Usman Roshan as part of his thesis work at the University of Texas [20]. Roshan obtained large data sets of biomolecular data from online databases as well from biologists at UT. He used the alignments provided, and then removed parsimony uninformative sites as well as low confidence sites. He then used this data to compare tree searching techniques such as those implemented in PAUP [24] and TNT [9] to his own (Recursive Iterative Disc Covering Methods). Doing these searches generated hundreds and sometimes thousands of trees for each data set. Roshan saved these trees and generously allowed us to analyze them.

Roshan provided these trees in over 650 different files. Each file was generated from a single data set and had the data set name as part of the file name. We created eleven files, one for each data set, that contained all of the trees that Roshan had provided for that data set. There are a few data sets for which Roshan provided trees on that we are not using in our benchmark because PAUP fails to generate accurate consensus trees. (We have communicated this error in PAUP to David Swofford). These now huge files are contained in a single directory and collectively referred to as Collection 1. We also kept several original files intact and created a directory of them which we refer to as Collection 2.

A third collection of trees was obtained from Tiffani Williams. William's collection consisted of trees generated from nine data sets, where 2505 trees were saved for each data set. We will refer to these as Collection 3.

An overview of the data sets used to generate these trees is given in Tables 2 and 3 which can be found in Appendix B. (For more information about the data sets used to generate these trees, see pages 55-59 of Usman Roshan's dissertation [20]. Note that Williams used data from Roshan, so some of the information on Williams trees is from [20].)

5.2 Transformation Process

The files of trees that were provided were not in a form that allowed them to be input directly to any of PAUP, TNT or TASPI. The files sometimes contained comments about how the trees were generated, parsimony scores, or other output from their production. Other times the files were just a line by line listing of trees in Newick format.

We created a suite of Perl scripts that take these input files and generate the input files for TNT and PAUP. For PAUP, these scripts collect the taxa list from the first tree in the file, and the trees themselves, and put them into a Nexus file that can then be executed by PAUP. The command to compute consensus (see Section 5.3) is also put into the Nexus file. Similarily for TNT, the taxa names are read and put into the file along with the trees themselves and the commands to do the desired computation. We also have wrapper scripts which allow us to compute consensus in PAUP or TNT for a directory of source files.

TASPI was designed to read the source files directly. It uses a variant of JCL (Job Control Language) to read the file and strip out unnecessary information, which presently includes branch lengths (since branch lengths are not used in computing consensus). While the trees are being read, they are also being transformed into TASPI's internal format.

5.3 Commands Used

In PAUP, the command to compute a consensus tree is *contree*. We used this command and set various options. We first computed the strict consensus tree by setting the option for strict to yes and the option for majority to no. We then computed a majority rule tree by setting strict to no and majority to yes with percent 50. We noticed much longer times for computing the strict tree than for majority, which, given our understanding of the problem made little sense. So, we also computed the strict majority tree by issuing a command that set strict to no, but majority to yes with percent 100. As expected, this resulted in the same trees as the strict command, but in much more reasonable times. The commands as they appeared in our input files are as follows:

```
contree all/strict=yes majrule=no treefile=<outputFile> replace;
contree all/strict=no majrule=yes percent=50 treefile=<outputFile> replace;
contree all/strict=no majrule=yes percent=100 treefile=<outputFile> replace;
```

We also ran PAUP with a paup block that computed first the majority tree (with percent 50), and then the strict majority by using the majrule option with percent 100 to test whether or not PAUP makes any use of the work done in the first *contree* command.

In TNT, there are two commands to compute (and not approximate) consensus trees. To compute a strict consensus, the command is *nelsen* (which, incidentally, has nothing to do with Nelson consensus except a play on words by the authors). To compute a majority consensus, the command is *majority*. The majority command allows the user to specify the cutoff to be used. This cutoff is the opposite of the percent in PAUP. In PAUP, "percent" is the percentage of input trees in which a branch must occur in order to be in the majority tree. In TNT, "cutoff" is the frequency at which a group is collapsed. Thus, it is not possible to compute a strict consensus tree using the majority command, since entering a cutoff of 100 tells TNT to collapse all groups with a frequency of 100 or less. We could attempt to compute a strict consensus using the majority command by entering a cutoff of 99 (cutoffs must be integers), but for greater than 100 trees this need not be a strict tree, and in our tests, we often have more than 100 trees. Thus, we do not use the majority command to compute a strict consensus tree in TNT.

One more point about TNT commands remains to be addressed. By default, TNT collapses branches with low support before doing a consensus computation. However, we were not dealing with any initial data other than the trees, so we turned this feature off by using the command *collapse notemp*. The other reason for doing this was to ensure that we were doing appropriate timing comparisons.

Finally, in TASPI, the commands to compute a consensus are as follows:

```
(setq *taxa-list* '(<theTaxaList>))
(bmaj-nexus-file "<fileName>" *taxa-list*)
(build-term-top (strip-cars <*bio-majority*|*bio-common*>) *taxa-list*)
```

The first command sets a global variable `*taxa-list*` to a list representing the taxa list. This is analogous to the *taxalabels* command present in the taxa block of a Nexus file. The second command is what does the large part of the work of our algorithm: reads the input file, finds all partitions present, and creates two global variables, `*bio-majority*` and `*bio-common*` which have the partitions occuring in a majority of the input trees, and the partitions occuring in all input trees, respectively. The last command builds a TASPI representation of the appropriate tree, majority if `*bio-majority*` is passed, and strict if `*bio-common*` is passed as the argument. At present, only a 50% majority can be built by TASPI, but implementing other cutoffs is trivial.

6 Results

6.1 Storing trees

One of the major contributions of TASPI is the condensed format in which trees can be stored, while maintaining structural information. Figure 3 shows the four sizes for each of our benchmark data sets. The Newick line is the size of the trees as they were given to us, after removing non-topological information (that is, branch lengths and other comments within the file were first removed). The Newick.bz2 line is the size of the file after compression using the algorithm implemented in bzip2 [21]. The TASPI.bhz line is the size of the file after compression using the Boyer-Hunt method. Notice that this is a compression in that the size of the file is much smaller, but all the information present in the original files is still accessible, in fact, at reduced speeds. Finally, the TASPI.bhz.bz2 line is the size of the file if it is compressed using the Boyer-Hunt method and then bzip2 is applied.

It is readily apparent that bzip2 produces smaller files than the Boyer-Hunt compression. However, the compressed Boyer-Hunt files are ready to be used as input to analysis, such as consensus. Overall, this represents a potentially huge savings in space since the Boyer-Hunt files can be further compressed using bzip2 for sharing and transmission purposes. Notice that in all cases the smallest size for a file is that from TASPI.bhz.bz2.

6.2 Reading trees

An advantage of the TASPI system is its ability to quickly read large numbers of large trees. Figure 4 gives the read times in seconds for each of our benchmark collections of trees. The PAUP and TNT times are average times to read over a number of runs while the TASPI times are from a single run. (For complete reading timing data see Appendix D).

Notice that reading the compressed trees is always the fastest time for any collection, and not by a small margin. Even reading the source files is faster in TASPI than it is in either PAUP or TNT, again, usually by a large margin. (There is one possible exception to this in Collection 2, for t64.nwk. However, for TNT and PAUP we would require greater timing precision in order to accurately compare read times.)

PAUP and TNT are very comparable in their read times. On larger files such as those in Collections 1 and 3, TNT tends to do slightly better than PAUP, but the

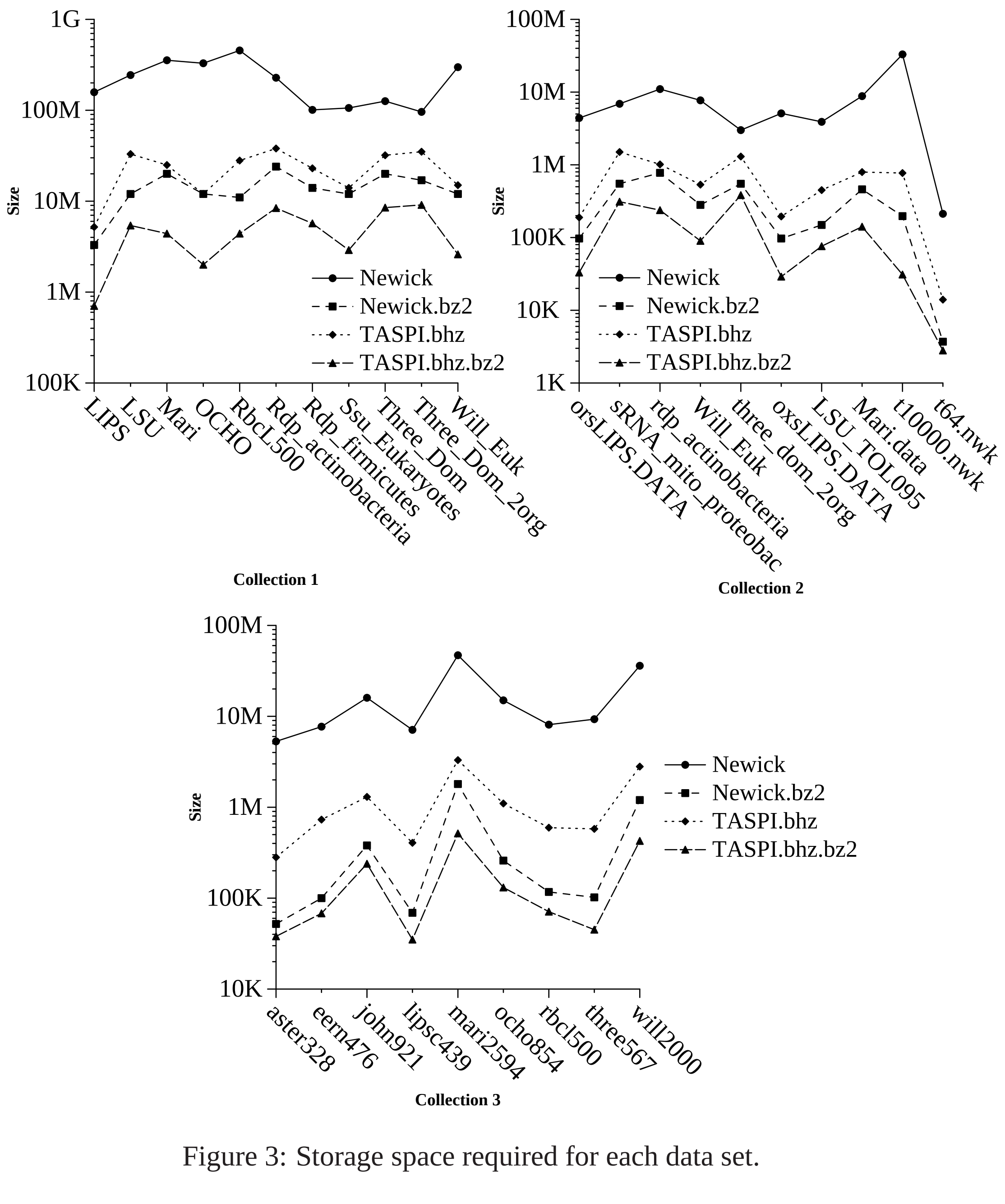


Figure 3: Storage space required for each data set.

difference is never much, and TNT doing better is not an absolute rule.

6.3 Computing Consensus

We ran consensus computations using PAUP, TNT, and TASPI, with the commands described in Section 5.3. Computations were on an Intel Pentium 4 CPU 3.4 GHz computer with 1024K cache and an Intel Itanium 2 CPU 900 MHz computer. (We were required to use two computers because of the memory required to do a consensus analysis in TASPI on Collection 1. Please see Section 6.4 for a further discussion of the memory requirement.)

Figure 5 shows the time to compute consensus with each of the programs. These times are all in seconds, and include the time to first read the input trees. In each case, the trees are read, and then both a strict tree and a majority 50 consensus tree are computed. Note that TASPI has better performance times compared to PAUP and TNT, and that TASPI does better the larger the files. Also note that TASPI achieves results identical to PAUP in all cases. This is also the same as TNT, but TNT uses a different taxon ordering, so we require some form of normalization to see that the trees are identical.

6.4 Analysis

Storing trees in compressed TASPI format saves considerable memory space. For the Collections 2 and 3, using compressed TASPI format is over 10 times smaller than Newick, and compressed TASPI together with bzip2 is 76 times better than Newick. Using bzip2 alone is better than compressed TASPI alone, but compressed TASPI together with bzip2 is 2 times better than bzip2 alone. In other words, compressed TASPI together with bzip2 creates files that require less than 2% of the original Newick space requirement.

Notice that as simply a compressed format, compressed TASPI is not as good as bzip2, but bzip2 data is not ASCII. Compressed TASPI is ASCII, and can therefore be read as input to later algorithms. This is its strength. If the data is not required as input, compressed TASPI is still useful since by using bzip2 on the compressed TASPI files we can achieve even smaller data files.

Reading the compressed TASPI format saves considerable time. Again, for Collections 2 and 3, reading the compressed TASPI representation of trees is quicker than reading the Newick. TNT and PAUP are comparable in their read times, but reading Newick files with TASPI is 5 to 7 times faster than with either

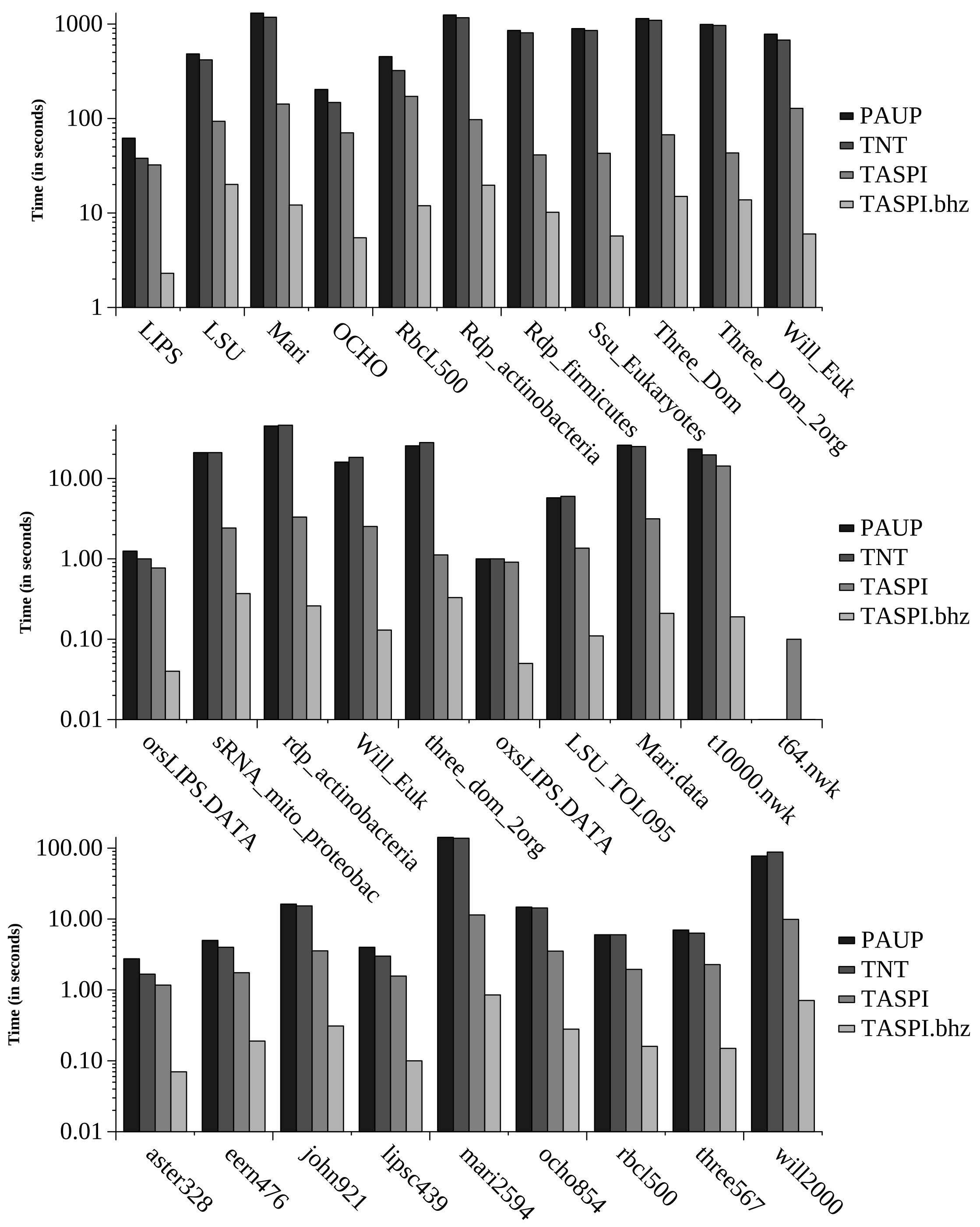


Figure 4: Time to read a collection of trees (Top-Collection 1, Middle-Collection 2, Bottom-Collection 3).

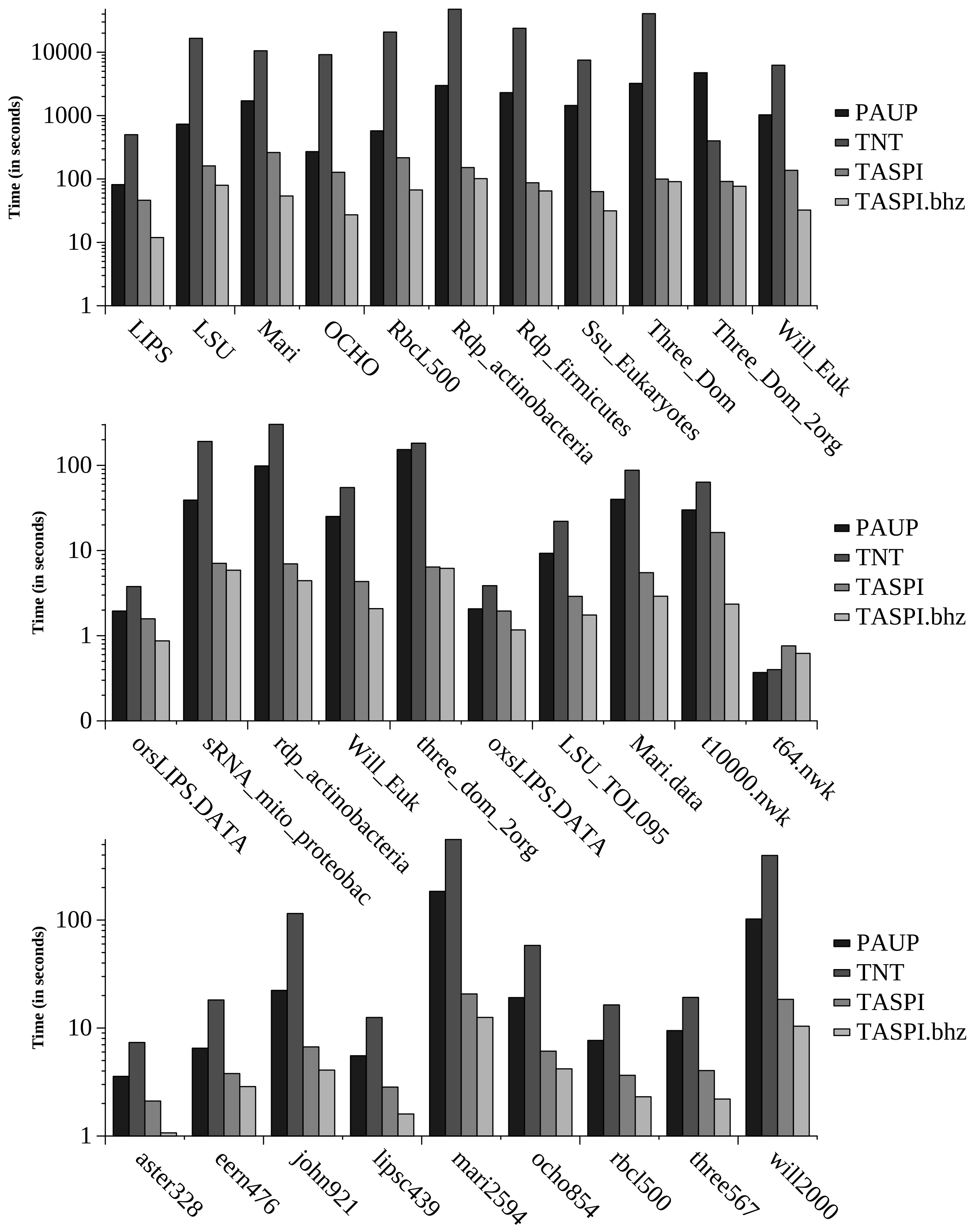


Figure 5: Time to read a collection of trees and compute strict and majority consensus trees (Top-Collection 1, Middle-Collection 2, Bottom-Collection 3).

TNT or PAUP. Reading the compressed TASPI format on the other hand is 97 times faster than reading with PAUP or TNT.

These fast read times translate into a fast consensus analysis. The total time to read a collection of trees and then compute a strict and majority consensus in TASPI is 14 times faster when reading compressed trees, and 7 times faster when reading Newick for Collection 2. Similarly, for Collection 3, TASPI is almost 9 times faster when reading compressed trees, and 5 times faster when reading Newick trees. It is important to note that in every case, TASPI and PAUP return the exact same results.

TASPI's total time is 29 times shorter than TNT on Collection 3 and 33 times shorter on Collection 2. We were expecting TNT to exceed PAUP in its ability to compute consensus trees since it is known for its ability to quickly generate maximum parsimony trees. Our results show that this speed did not translate to its consensus algorithms.

Now, the greatest drawback to TASPI is its memory requirements. In running our experiments we used both a 32 bit computer and a 64 bit computer. The 64 bit machine was required for Collection 1 of trees due to a lack of memory on the 32 bit machine. PAUP and TNT were both able to complete on the 32 bit machines, but did so with terrible performance.

TASPI is able to read all data files on the 32 bit machine (with the exception of RbcL500, which would be possible if we allocated memory differently). However, it is not capable of running a consensus analysis on the large files in Collection 1. Currently we are maintaining a charade of fully functional programming which is in large part where our memory issues arise. Early results with Collection 1 indicate that we may be able to achieve running times over 10 times faster than PAUP.

7 Conclusion

In phylogenetics, computing consensus trees is not where most computation time is spent. However, having fast consensus methods is becoming increasingly important. Researchers are proposing the use of the rate of change of a consensus tree as a stopping criterion for heuristic MP searches which would require recomputing a consensus tree multiple times over the course of an analysis [25].

We have given a new format for collections of phylogenetic trees that would make this possible. The format allows trees to be stored in considerably less space, while allowing for improved performance in computing consensus. Considering all of our data sets, TASPI format allows collections of trees to be stored in 1/47th the amount of space required by Newick. Using TASPI to compute consensus for all collections in our benchmarks (and using scaled numbers) is 25 times faster than using PAUP and over 300 times faster than using TNT.

8 Acknowledgments

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

References

[1] E. N. Adams. Consensus techniques and the comparison of taxonomic trees. *Systematic Zoology*, 21(4):390–397, 1972.

[2] Nina Amenta, Frederick Clarke, and Katherine St. John. A linear-time majority tree algorithm. In Gary Benson and Roderic D. M. Page, editors, *Algorithms in Bioinformatics, Third International Workshop, WABI 2003, Budapest, Hungary, September 15-20, 2003, Proceedings*, volume 2812 of *Lecture Notes in Computer Science*, pages 216–227. Springer-Berlin / Heidelberg, 2003.

[3] Tanya Y. Berger-Wolf. Online Consensus and Agreement of Phylogenetic Trees. In Inge Jonassen and Junhyong Kim, editors, *Algorithms in Bioinformatics, 4th International Workshop, WABI 2004, Bergen, Norway, September 14-17, 2004, Proceedings*, volume 3240 of *Lecture Notes in Computer Science*. Springer-Berlin / Heidelberg, 2004.

[4] Tanya Y. Berger-Wolf, Tiffani L. Williams, Bernard E. Moret, and Tandy J. Warnow. An experimental evaluation of phylogenetic consensus methods. Technical Report 19, University of New Mexico, 2003.

[5] 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, pages 163–184. DIMACS.AMS., 2003.

[6] William H. E. Day. Optimal Algorithms for Comparing Trees with Labeled Leaves. *Journal of Classification*, 2(1):7–28, 1985.

[7] J. Felsenstein. The Newick tree format. `http://evolution.genetics.washington.edu/phylip/newicktree.html`, 1986.

[8] Joseph Felsenstein. *Inferring Phylogenies*. Sinauer Associates, Inc., 2004.

[9] 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.

[10] 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.

[11] David M. Hillis, Craig Moritz, and Barbara K. Mable, editors. *Molecular Sytematics*. Sinauer Associates, Inc., Sunderland, Massachusetts, 2nd edition, 1996.

[12] Matt Kaufmann, Pete Manolios, and J. S. Moore. *Computer-Aided Reasoning: An Approach*. Kluwer Academic Publishers, July 2000.

[13] T. Margush and F.R. McMorris. Consensus n-Trees. *Bulletin of Mathematical Biology*, 43(2):239–244, 1981.

[14] F. R. McMorris. Axioms for Consensus Functions on Undirected Phylogenetic Trees. *Mathematical Biosciences*, 74(1):17–21, 1985.

[15] F. R. McMorris, D. B. Meronk, and D. A. Neumann. A view of some consensus methods for trees. In J. Felsenstein, editor, *Numerical Taxomony*, pages 122–125. Springer-Verlag, 1983.

[16] F.R. McMorris and D. Neumann. Consensus functions defined on trees. *Mathmatical Social Sciences*, 4(2):131–136, April 1983.

[17] D. A. Neumann. Faithful Consensus Methods for n-Trees. *Mathematical Biosciences*, 63:271–287, 1983.

[18] Rod Page. COMPONENT User's Guide. `http://taxonomy.zoology.gla.ac.uk/rod/cplite/ch4.pdf`, 1997.

[19] Cynthia Phillips and Tandy J. Warnow. The asymmetric median tree - A new model for building consensus trees. *Discrete Applied Mathematics*, 71:311–335, 1996.

[20] Usman Roshan. *Algorithmic techniques for improving the speed and accuracy of phylogenetic methods*. PhD thesis, University of Texas at Austin, 2004.

[21] Julien Seward. bzip2. `http://sources.redhat.com/bzip2/`, 2002.

[22] Robert R. Sokal and F. James Rohlf. Taxonomic Congruence in the Leptopodomorpha Re-Examined. *Systematic Zoology*, 30(3):309–325, Sep 1981.

[23] Guy L. Steele. *Common Lisp the Language*, chapter 22.1.4. Digital Press, 2nd edition, 1990.

[24] D. L. Swofford. *PAUP*: Phylogenetic Analysis Using Parsimony (and Other Methods) 4.0 Beta*. Sinauer Associates, Sunderland, Massachusetts, 2002.

[25] Tiffani Williams, Tanya Berger-Wolf, Bernard Moret, Usman Roshan, and Tandy Warnow. The Relationship Between Maximum Parsimony Score and Phylogenetic Tree Topologies. Technical Report TR-CS-2004-04, University of New Mexico, 2004.

[26] Shibu Yooseph. *Phylogeny Construction and Consensus Methods*. PhD thesis, University of Pennsylvania, 1997.

[27] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions in Information Theory*, IT-24:337–343, 1977.

Data Set	Pentium	Itanuim	Factor (I/P)
orsLIPS.DATA_fixed	1.58	21.16	13.39
sRNA_mito_proteobac	7.08	31.83	4.49
rdp_actinobacteria	6.98	39.64	5.68
Will_Euk	4.33	24.81	5.73
three_dom_2org	6.41	32.73	5.11
oxsLIPS.DATA_fixed	1.95	12.70	6.51
LSU_TOLO95	2.90	15.81	5.45
Mari.data	5.50	27.13	4.93
t10000.nwk	16.29	61.41	3.77
t64.nwk	0.76	6.65	8.75
		Average	5.60

Table 1: Comparison of computation times

A Comparison of machines

Due to the memory requirements as discussed in Section 6.4, we were required to use two different machines in our benchmarks. The first was Dell computer with an Intel 3.4 GHz Pentium 4 processor while the second was an HP machine based on a 900 MHz Itanium 2 processor. In order to get comparable numbers, we ran consensus using TASPI on Collection 2 on both of the machines (see Table 1). We then found the average factor by which the Itanium machine was slower than the Pentium (we removed the largest factor to avoid biasing our results) and scaled our timing data from the Itanium (for Collection 1) by this factor (5.6).

B Data Set Overview

The trees in our collections were generated by outside sources, namely Usman Roshan and Tiffani Williams. Tables 2 and 3 summarize the data sets that they used to generate the trees used in our benchmarks.

Data Set	Source	Type	Seqs	Sites	Best Score	Number of Trees
Collection 1 – Trees from Usman Roshan						
LIPS	Goloboff	rDNA	439	2461	41290	28947
LSU	database	RNA	1322	1078	52053	26938
Mari	Kallersjo	rbcL DNA	2954	1232	59858	18938
OCHO	Ochoterena	rbcL DNA	854	937	23005	31085
RcbL500	Rice web site	rbcL DNA	500	759	16218	141065
Rdp_actinobacteria	RDP	16s ribosomal RNA	4583	1263	60892	6706
Rdp_firmicutes	RDP	16s ribosomal RNA	7233	1352	156219	1866
Ssu_Eukaryotes	ERD	Eukaryotes RNA	6590	1661	232630	2143
Three_Dom	Gutell	three-domain sRNA	7180	1122	91894	2346
Three_Dom_2org	Gutell	three-domain sRNA	8506	851	99839	1503
Will	Gutell	sRNA	2000	1251	74536	20926

Table 2: Data set overview

Data Set	Source	Type	Seqs	Sites	Best Score	Number of Trees
Collection 2 – Trees from Usman Roshan						
orsLIPS.DATA	Goloboff	rDNA	439	2461	41290	803
sRNA_mito_proteobac	unknown	unknown	2587	unknown	unknown	369
rdp_actinobateria	unknown	unknown	4583	unknown	unknown	301
Will_Euk_	Gutell	sRNA	2000	1251	74536	537
three_dom_2org	Gutell	3-dom sRNA	8506	851	99839	47
oxsLIPS.DATA	Goloboff	rDNA	439	2461	41290	926
LSU_TOL095	database	RNA	1322	1078	52053	423
Mari.data	Kallersjo	rbcL DNA	2954	1232	59858	465
t10000.nwk	Rice web site	rbcL DNA	500	759	16218	10000
t64.nwk	Rice web site	rbcL DNA	500	759	16218	64
Collection 3 – Trees from Tiffani Williams						
aster328	Gutell	RNA	328	946		2505
eern476	Eernisse	Metazoan DNA	476	1008	17765	2505
john921	Johnson	AvianCytochrome*b*DNA	921	713	42759	2505
lipsc439	Goloboff	rDNA	439	2461	41290	2505
mari2594	Kallersjo	rbcL DNA	2954	1232	59858	2505
ocho854	Ochoterena	rbcL DNA	854	937	23005	2505
rbcl500	Rice web site	rbcL DNA	500	759	16218	2505
three567	Soltis	rbcL, atpB, 18s DNA	567	2153	74536	2505
will2000	Gutell	sRNA	2000	1251	74536	2505

Table 3: Data set overview continued

C Storage

The raw data numbers for storing the collections of trees in our benchmarks are given in Table 4.

D Read times

This appendix shows additional read times for each of our collections of trees. All times are in seconds, and the type of computer used was an Intel(R) Pentium(R) 4 CPU 3.40GHz with 1024 KB. Table 5 gives the read times using PAUP and Table 6 gives the read times using TNT.

Table 7 gives comparative numbers. The times for TNT and PAUP are average times across a number of runs, while the TASPI numbers are from a single run.

E Computing consensus times

This appendix gives further details about computing consensus in PAUP and TNT. In PAUP, we computed a strict consensus using the "strict" option, a majority consensus with percent 50, a majority consensus with percent 100, and then both a majority with percent 50 and a majority with percent 100. Table 8 shows the total computation time for each of these possibilities. In each case the time shown is the number of seconds to both read in the trees and compute the consensus tree indicated. Notice that to compute a strict majority tree takes considerably more time than to compute the same tree using the majority option. Having no access to the source code of PAUP we have no ideas why this is, but it does seem to be an issue.

In TNT, we computed a strict consensus tree using the command *nelsen* and a majority tree with cutoff 50 using the *majority* command, and then also computed both strict and majority trees. As in the PAUP table, the times shown in Table 9 are total computation times and therefore include the time to read in the collection of trees.

Table 10 gives comparative numbers. Each entry is the time to read and compute both a strict and majority tree in the program given in the column heading. (The times for Collection 1 are scaled from the 64-bit computer.) Notice again that using the compressed TASPI format always results in by far the shortest time for any collection of trees.

Data Set	Newick	Newick.bz2	TASPI	TASPI.bz2
Collection 1 – Trees from Usman Roshan				
LIPS	158M	3.3M	5.2M	704K
LSU	244M	12M	33M	5.4M
Mari	355M	20M	25M	4.4M
OCHO	329M	12M	12M	2.0M
RbcL500	455M	11M	28M	4.4M
Rdp_actinobacteria	228M	24M	38M	8.4M
Rdp_firmicutes	101M	14M	23M	5.7M
Ssu_Eukaryotes	106M	12M	14M	2.9M
Three_Dom	126M	20M	32M	8.5M
Three_Dom_2org	96M	17M	35M	9.1M
Will_Euk	298M	12M	15M	2.6M
Collection 2 – Trees from Usman Roshan				
orsLIPS.DATA	4.4M	97K	189K	33K
sRNA_mito_proteobac	6.9M	550K	1.5M	311K
rdp_actinobacteria	11M	776K	1014K	238K
Will_Euk	7.7M	281K	534K	90K
three_dom_2org	3.0M	549K	1.3M	383K
oxsLIPS.DATA	5.1M	97K	195K	29K
LSU_TOLO95	3.9M	149K	449K	76K
Mari.data	8.8M	460K	795K	141K
t10000.nwk	33M	197K	771K	31K
t64.nwk	212K	3.7K	14K	2.8K
Collection 3 – Trees from Tiffani Williams				
aster328	5.3M	52K	281K	38K
eern476	7.7M	100K	731K	68K
john921	16M	380K	1.3M	239K
lipsc439	7.1M	69K	406K	35K
mari2594	47M	1.8M	3.3M	515K
ocho854	15M	259K	1.1M	131K
rbcl500	8.1M	117K	596K	71K
three567	9.3M	102K	578K	45K
will2000	36M	1.2M	2.8M	426K

Table 4: Data Set Sizes

Filename	Strict	Maj50	Maj100	Both	Average
Collection 1 – Trees from Usman Roshan					
LIPS.Data_All	51	67	66	64	62.00
LSU_tolo95_All	489	484	480	480	483.25
Mari_All	1285	1307	1315	1318	1306.25
OCHO.Data_All	198	204	206	205	203.25
RbcL500_All	451	452	452	456	452.75
Rdp_actinobateria_All	1246	1241	1254	1251	1248.00
Rdp_firmicutes_All	857	852	859	857	856.25
Ssu_Eukaryotes_All	895	892	896	897	895.00
Three_Dom_All	1143	1139	1148	1144	1143.50
Three_dom_2org_All	999	991	990	986	991.50
Will_Euk_All	779	785	782	784	782.50
Collection 2 – Trees from Usman Roshan					
orsLIPS.DATA	1	1	1	1	1.00
sRNA_mito_proteobac	21	22	22	21	21.50
rdp_actinobateria	46	46	55	45	48.00
Will_Euk	17	16	16	16	16.25
three_dom_2org	26	25	25	25	25.25
oxsLIPS.DATA	2	2	2	2	2.00
LSU_TOL095	6	6	6	7	6.25
Mari.data	27	33	41	26	31.75
t10000.nwk	31	27	28	28	28.50
t64.nwk	< 1	< 1	< 1	< 1	< 1
Collection 3 – Trees from Tiffani Williams					
aster328	2	3	2	2	2.25
eern476	4	5	4	5	4.50
john921	16	17	17	20	17.50
lipsc439	8	4	4	4	5.00
mari2594	137	137	145	205	156.00
ocho854	15	15	14	15	14.75
rbcl500	6	12	12	12	10.50
three567	15	7	7	7	9.00
will2000	74	73	73	73	73.25

Table 5: Time (in seconds) to read trees into PAUP when about to compute the type of consensus given in the column heading

Filename	Strict	Maj50	Both	Average
Collection 1 – Trees from Usman Roshan				
LIPS.Data_All	38.00	38.00	38.00	38.00
LSU_tolo95_All	425.00	429.00	400.00	418.00
Mari_All	1178.00	1185.00	1182.00	1181.67
OCHO.Data_All	147.00	149.00	148.00	148.00
RbcL500_All	328.00	330.00	309.00	322.33
Rdp_actinobateria_All	1165.00	1163.00	1175.00	1167.67
Rdp_firmicutes_All	809.00	811.00	805.00	808.33
Ssu_Eukaryotes_All	851.00	857.00	857.00	855.00
Three_Dom_All	1097.00	1096.00	1097.00	1096.67
Three_dom_2org_All	969.00	972.00	962.00	967.67
Will_Euk_All	679.00	679.00	673.00	677.00
Collection 2 – Trees from Usman Roshan				
orsLIPS.DATA	0.00	0.00	1.00	0.33
sRNA_mito_proteobac	21.00	22.00	20.00	21.00
rdp_actinobateria	47.00	45.00	45.00	45.67
Will_Euk	15.00	15.00	15.00	15.00
three_dom_2org	26.00	26.00	26.00	26.00
oxsLIPS.DATA	1.00	1.00	1.00	1.00
LSU_TOL095	5.00	6.00	6.00	5.67
Mari.data	25.00	26.00	25.00	25.33
t10000.nwk	20.00	20.00	19.00	19.67
t64.nwk	0.00	0.00	0.00	0.00
Collection 3 – Trees from Tiffani Williams				
aster328	1.00	2.00	2.00	1.67
eern476	4.00	4.00	4.00	4.00
john921	16.00	15.00	15.00	15.33
lipsc439	4.00	3.00	4.00	3.67
mari2594	136.00	139.00	143.00	139.33
ocho854	14.00	13.00	30.00	19.00
rbcl500	5.00	5.00	4.00	4.67
three567	13.00	13.00	6.00	10.67
will2000	71.00	72.00	71.00	71.33

Table 6: Time (in seconds) to read trees into TNT when about to compute the type of consensus given in the column heading

Data Set Name	PAUP	TNT	TASPI	TASPI.bhz
Collection 1 – Trees from Usman Roshan				
LIPS	62.00	38.00	32.28	2.30
LSU	483.25	418.00	93.50	20.07
Mari	1306.25	1181.67	142.31	12.15
OCHO	203.25	148.00	70.62	5.47
RbcL500	452.75	322.33	171.64	*11.94*
Rdp_actinobacteria	1248.00	1167.67	97.40	19.68
Rdp_firmicutes	856.25	808.33	41.21	10.18
Ssu_Eukaryotes	895.00	855.00	42.87	5.71
Three_Dom	1143.50	1096.67	67.31	14.99
Three_Dom_2org	991.50	967.67	43.25	13.78
Will_Euk	782.50	677.00	128.18	6.00
Collection 2 – Trees from Usman Roshan				
orsLIPS.DATA	1.25	1.00	0.77	0.04
sRNA_mito_proteobac	21.00	21.00	2.42	0.37
rdp_actinobacteria	45.00	46.00	3.31	0.26
Will_Euk	16.00	18.33	2.53	0.13
three_dom_2org	25.50	28.00	1.12	0.33
oxsLIPS.DATA	1.00	1.00	0.91	0.05
LSU_TOLO95	5.75	6.00	1.36	0.11
Mari.data	26.00	25.00	3.15	0.21
t10000.nwk	23.25	19.67	14.29	0.19
t64.nwk	< 1	0.00	0.10	0.00
Collection 3 – Trees from Tiffani Williams				
aster328	2.75	1.67	1.17	0.07
eern476	5.00	4.00	1.75	0.19
john921	16.25	15.33	3.56	0.31
lipsc439	4.00	3.00	1.57	0.10
mari2594	142.00	138.00	11.42	0.85
ocho854	14.75	14.33	3.53	0.28
rbcl500	6.00	6.00	1.95	0.16
three567	7.00	6.33	2.28	0.15
will2000	77.50	88.00	9.89	0.71

Table 7: Comparative read times (in seconds). Note that the time to read the compressed RbcL500 data is scaled from a 900MHz machine instead of the 3.4GHz machine in all other entries (see Section A).

Filename	Strict	Maj50	Maj100	Both
Collection 1 – Trees from Usman Roshan				
LIPS.Data_All	176.46	75.88	74.62	81.53
LSU_tolo95_All	1642.71	608.70	604.74	733.36
Mari_All	9487.74	1501.86	1507.31	1710.64
OCHO.Data_All	793.78	236.81	235.58	270.21
RbcL500_All	1046.88	508.22	507.95	574.82
Rdp_actinobateria_All	8870.73	2102.26	2114.41	2985.75
Rdp_firmicutes_All	12187.21	1592.72	1574.64	2304.54
Ssu_Eukaryotes_All	14053.96	1176.39	1156.38	1445.68
Three_Dom_All	11840.77	2251.45	2229.79	3325.03
Three_dom_2org_All	8145.78	2899.05	2853.41	4748.29
Will_Euk_All	3883.13	907.21	902.51	1027.35
Collection 2 – Trees from Usman Roshan				
orsLIPS.DATA	6.48	1.57	1.62	1.95
mito_proteobac	171.00	30.71	29.76	39.14
rdp_actinobateria	638.22	75.30	66.42	98.56
Will_Euk	189.59	20.62	20.22	25.14
three_dom_2org	418.39	106.76	76.72	153.36
oxsLIPS.DATA	7.17	1.79	1.78	2.07
LSU_TOL095	35.28	7.67	7.46	9.29
Mari.data	310.66	33.23	32.04	39.97
t10000.nwk	120.33	26.82	26.83	30.00
t64.nwk	0.95	0.27	0.26	0.37
Collection 3 – Trees from Tiffani Williams				
aster328	8.45	3.09	3.10	3.57
eern476	18.07	5.67	5.65	6.51
john921	71.12	19.26	19.19	22.32
lipsc439	18.00	4.86	4.82	5.54
mari2594	1303.51	157.28	164.94	184.52
ocho854	71.66	16.86	16.72	19.14
rbcl500	23.03	6.83	6.79	7.68
three567	36.08	8.40	8.37	9.47
will2000	498.92	88.63	86.84	102.29

Table 8: Time (in seconds) to read trees and compute consensus using PAUP.

Filename	Strict	Maj50	Both
Collection 1 – Trees from Usman Roshan			
LIPS.Data_All	91.32	451.57	500.27
LSU_tolo95_All	866.92	16372.49	16564.88
Mari_All	2522.30	9247.21	10536.53
OCHO.Data_All	359.18	9034.97	9154.71
RbcL500_All	669.54	20749.34	20792.32
Rdp_actinobateria_All	2614.17	46203.27	47675.47
Rdp_firmicutes_All	1879.99	22792.60	23878.79
Ssu_Eukaryotes_All	1920.54	6450.15	7514.41
Three_Dom_All	2488.50	39222.34	40600.86
Three_dom_2org_All	2231.63	38541.74	39964.79
Will_Euk_All	1499.25	5381.59	6236.13
Collection 2 – Trees from Usman Roshan			
orsLIPS.DATA	1.92	2.69	3.78
sRNA_mito_proteobac	41.25	169.74	191.09
rdp_actinobateria	93.59	255.65	303.13
Will_Euk	32.76	37.82	54.95
three_dom_2org	58.70	152.83	182.31
oxsLIPS.DATA	2.26	2.61	3.87
LSU_TOL095	11.76	19.81	22.06
Mari.data	51.90	61.54	87.75
t10000.nwk	40.63	42.25	63.52
t64.nwk	0.26	0.26	0.40
Collection 3 – Trees from Tiffani Williams			
aster328	4.23	4.95	7.35
eern476	8.78	13.13	18.20
john921	36.45	95.41	115.02
lipsc439	7.29	8.34	12.53
mari2594	309.08	396.75	557.13
ocho854	29.72	41.02	58.23
rbcl500	9.78	12.39	16.38
three567	11.12	14.38	19.23
will2000	171.59	296.47	396.44

Table 9: Time (in seconds) to read trees and compute consensus using TNT.

Data Set Name	PAUP	TNT	TASPI	TASPI.bhz
Collection 1 – Trees from Usman Roshan				
LIPS	81.53	500.27	*46.17*	*11.92*
LSU	733.36	16564.88	*160.95*	*79.60*
Mari	1710.64	10536.53	*261.61*	*53.95*
OCHO	270.21	9154.71	*127.59*	*27.21*
RbcL500	574.82	20792.32	*216.42*	*67.12*
Rdp_actinobacteria	2985.75	47675.47	*151.46*	*101.30*
Rdp_firmicutes	2304.54	23878.79	*87.19*	*64.79*
Ssu_Eukaryotes	1445.68	7514.41	*63.25*	*31.42*
Three_Dom	3325.03	40600.86	*99.68*	*90.76*
Three_Dom_2org	4748.29	39964.79	*91.47*	*76.78*
Will_Euk	1027.35	6236.13	*137.04*	*32.34*
Collection 2 – Trees from Usman Roshan				
orsLIPS.DATA	1.95	3.78	1.58	0.87
sRNA_mito_proteobac	39.14	191.09	7.08	5.88
rdp_actinobacteria	98.56	303.13	6.98	4.43
Will_Euk	25.14	54.95	4.33	2.08
three_dom_2org	153.36	182.31	6.41	6.18
oxsLIPS.DATA	2.07	3.87	1.95	1.17
LSU_TOLO95	9.29	22.06	2.90	1.75
Mari.data	39.97	87.75	5.50	2.91
t10000.nwk	30.00	63.52	16.29	2.35
t64.nwk	0.37	0.40	0.76	0.62
Collection 3 – Trees from Tiffani Williams				
aster328	3.57	7.35	2.11	1.07
eern476	6.51	18.20	3.79	2.87
john921	22.32	115.02	6.69	4.08
lipsc439	5.54	12.53	2.84	1.60
mari2594	184.52	557.13	20.69	12.54
ocho854	19.14	58.23	6.11	4.19
rbcl500	7.68	16.38	3.65	2.31
three567	9.47	19.23	4.04	2.20
will2000	102.29	396.44	18.43	10.4

Table 10: Comparative computation of consensus times (in seconds). Note that for Collection 1, the TASPI times are scaled from a 900Mhz machine instead of the 3.4GHz machine in all other entries (see Section A).