On the Importance of Starting Trees

Suzanne J. Matthews and Dr. Tiffani L. Williams

Department of Computer Science

Texas A&M University

College Station, TX 77843

Contents

Abstract—**Our premise is that starting tree structure affects the quality of a maximum parsimony search. While it is generally believed that starting tree structure is important to phylogenic search, no study has yet been performed to show the degree to which (if any) a starting tree's structure improves the quality of a phylogenic search. We hypothesize that better starting trees will yield better phylogenic searches; according to this hypothesis, random trees (a tree created by an arbitrary arrangement of taxa) would perform the worst. To rigorously test this hypothesis, starting trees created with the distance-based neighbor-joining (NJ) method, as well as random trees and random sequence addition (RSA) trees were fed into the parsimony-ratchet framework found within PAUP; maximum parsimony scores resulting from the search were then extracted and compared. This process was performed over multiple taxa sets, ranging from 51 to 1127 taxa in size. In addition to testing the performance of existing starting tree methods, we also introduce a novel method for starting tree construction called *The RAq Approach*, a DCM-based algorithm that is partially inspired by threshold graphs. Our results indicate the the *RAq* approach performs competitively well to the other starting tree construction methods. However, our results also indicate that starting tree structure is not important to phylogenetic search, as the random tree assisted search consistently performed the best.**

I. Introduction

THE study of phylogeny, or the organization of species in relation to one another, has increasingly captivated scientists. With a burgeoning wealth of biological datasets scattered all over the globe, the need to successfully organize and manipulate this data is growing ever stronger. The ability to classify different species assists in various applications, including forensics, disease and poison control, drug development, and the prediction of the evolutionary relationships between all life on this planet.

Mutations, and their relative fitness in a particular environmental context, drive the process of evolution. In the process of *speciation*, two or more new species originate from an ancestral species, due to a mix of environmental and genetic changes. This is the generally accepted model on how all life evolved on the planet. In order to study how different organisms are related to each other, it is therefore very important to take a look at the changes that take place at the genetic level, particularly at the protein and DNA levels. In the context of maximum parsimony, it is desirable to analyze sequence data at the RNA or DNA levels, as the "wobble effect" found in amino-acid chains can mask individual base mutations.

A. Maximum Parsimony

Maximum parsimony (MP) is a popular criterion that is applied to many phylogenetic construction algorithms. Its relative simplicity is derived from the principle of Occam's razor, in which the simplest explanation is often the most desirable. Known as the *lex parsimoniae* in Latin, it states that "entities are not to be multiplied without necessity". Applied to phylogeny, this roughly translates to "the path of evolution is the simplest, and the most direct", implying that evolution occurs with the least number of changes as possible. Using this criteria, organisms with less sequential differences are categorized as being more closely related that those with larger number of differences; with that said, it should be noted that DNA sequences are largely conserved over a large spectrum of species. In the scope of this paper, our phylogenetic searches and starting tree constructions are made under the assumption of maximum parsimony.

B. Methods for Phylogenetic Tree construction

Under the MP criterion, various methods for inferring phylogenies exist; of these, the most popular are distance based methods, and heuristic searches. Distance-based methods build phylogenetic trees off of an existing distance matrix, using some pre-defined distance metric. The most naive of these methods is the Hamming (or uncorrected) distance, in which the pairwise "distance" between two organisms is simply the number of differing bases between the two taxa. However, this is generally accepted as being an unreliable metric for distance. The more popular Jukes-Cantor model for distance assumes that each position in the sequence has an equal chance of mutating; therefore, there is only a 75% chance that the mutation of a certain base will be different. The Jukes-Cantor method thus "corrects" the uncorrected distances to overcome this issue. However, when the distance between two sequences is greater than 0.75, the distance becomes undefined. The Felsenstein 81 model can overcome this annoyance.

Despite the large popularity of the JC method, many dislike it, as it assigns an equal probability to all bases. To counter this, the closely related Kimura 2-parameter (K2P) model was created, having different probabilities assigned depending on whether or not the mutation is labeled a transition (A-G, C-T) or a transversion (everything else). Still others prefer methods such as Felsenstein 84 or HKY85 distance, since they tend to take other biological trends into context. After the distance matrix is created, a tree structure is obtained using some type of supertree method, the most popular of these being the *neighbor joining* method. Trees created with this method tend to be statistically consistent, and highly efficient; trees created with the neighbor joining can construct trees from large biological datasets in polynomial time. However, due to the basic "greedy" nature of the algorithm, it is possible for the tree to be not be a global approximation of the true tree; rather, it may be a good local approximation of some subset of the tree [2]. Furthermore, the neighbor joining method will always yield a single tree per matrix. This can be undesirable, as the one tree this method produces is the only one presented as the "true" tree approximation.

In contrast, heuristic approaches to tree building, such as Maximum Likelihood(ML) and MP, tend to produce several, statistically good solutions, allowing for a greater chance for a good tree to be found. In the scope of this paper, only maximum parsimony methods will be discussed. Popular examples of maximum parsimony searches include the Rec-I-DCM3 framework, developed by Tandy Warnow, and the Parsimony Ratchet method, supported by PAUP. Both these phylogenetic search utilities are dependent on an initial tree structure, known commonly as a starting tree.

C. Disk Covering Methods

Disk Covering Methods (DCMs) are a class of algorithms developed by Tandy Warnow. All disk covering methods are composed of three main parts: some decomposition strategy, some merging stategy, and resolution of the tree if needed. During the decomposition stage, the biological dataset is first decomposed into a series of overlapping subsets. Since the subsets overlap, the leafsets of the trees will also overlap[9]. These overlapping leafsets are then merged together by a supertree method; in the context of the DCMs discussed in Warnow's paper, the supertree construction method of choice is the *Strict Consensus Merger*, which expoits the great overlap of subsets in order to properly construct a phylogenetic tree. After the supertree is constructed, it is usually refined to maximize a certain criterion, such as maximum parsimony. Therefore, starting tree construction via DCMs can be considered just the set including decomposition and supertree construction.

All current DCMs start by creating a triangulated graph; this triangulated graph can be derived from a variety of methods. For example, threshold graphs (TGs) use a static q value (acting as the "threshold") to derive a subset of the taxa; the resulting subset is then triangulated. This is important to note for our later discussion of design decisions for the RAq algorithm.

D. Parsimony Ratchet

The parsimony ratchet is a particular kind of phylogenetic search performed with alternating cycles of reweighting and TBR (Tree Bisection Recombination). The approach works as follows; starting with an initial tree, a few of the characters (between 5 - 25%) are sampled, and reweighted. It suffices to say here that reweighting of characters involves using the duplicating the characters so that each shows up twice (or more) in the resulting dataset[2]. Then, using these reweighted characters, TBR search is performed until a new starting tree is reached using this subset of data. This new starting tree is then used with the original data set to repeat the phylogenetic search. Parsimony ratchet tries to refine the search by generating a tree from a small subset of the data and using it as a new starting point. The approach so far has worked well so far; in the famous Chase experiments, the Parsimony Ratchet found a new best tree score for the 500 zilla dataset in a fraction of the time than a previous study done by Rice et. al; speedup was estimated as above 1000 percent [5].

E. Starting Tree Construction

As mentioned before, phylogenetic searches like the parsimony ratchet require an initial starting tree from which to direct their searches. Currently, there are several ways to create a starting tree. The simples of these is the Random method, in which a set of n taxa are randomly place together into a tree structure, irregardless of any particular criterion. Some scientists, however, use the RSA (Random Sequence Addition) method to create a starting tree. RSA works as follows: Given a random sequence of taxa $\{a_1, a_2, ..., a_n\}$, where a represents a random, distinct taxon, construct a tree by adding one taxon at a time, at the most topologically favorable location. While there is more overhead using this method than using the Random method, it is assumed that since topology is the criterion being selected for, we will have a better approximation of the true tree than if we used the Random method. Other forms of starting tree construction include using distance-based trees, such as the aforementioned neighbor joining method.

If starting tree structure does affect the quality of a phylogenetic search, then it would be beneficial to see which starting tree are better, and under what conditions. This will allow scientists to increase the speed and efficacy with which they traverse tree space.

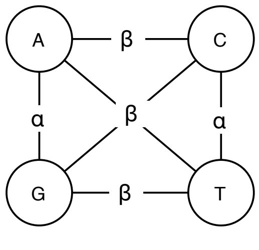
II. The RAq Approach

The RAq approach is a DCM-based method that was created to test how well a DCM with a dynamic threshold would perform in relation to previous methods, which only had a static threshold. In [9], Warnow discusses the problem with choosing an accurate threshold; if the threshold is too small, the resulting subgraph would not be connected. If the threshold is too large, then the subproblems would be almost as difficult as the original ones[9]. By using a dynamic threshold, it was hoped that this problem of choosing a correct threshold would be elimated, since a range of thresholds are used. Therefore, taxa that are closer together will fall together in one cluster, and those that are a bit less apart with fall into the next and so on. Furthermore, the taxa chosen in the first few intial clusters will determine what gets elimated in the creation of later clusters; in this manner, taxa won't be repeated a large number of times.

Like other DCM based methods, the *RAq* approach is composed of a decomposition method (*RAq* strategy), and a supertree construction method (we used neighbor joining). Furthermore, since *RAq* is a distance based DCM method, an initial distance matrix is also created.

A. Distance Matrix Creation

As discussed above, several metrics for phylogenic construction exist, the most popular being the Jukes-Cantor class of metrics. However, this class of distance metrics have a very serious flaw: all mutations, regardless of type, are considered as having equal probability. This is biologically inconsistent, as biological transitions are more likely than biological transversions. For this reason, a scoring model based on the K2P model (shown below) was used, where the values of α and β were parametized. We call this scoring model the DS_{dist} metric, for *Dynamic Score distance*. It is used as our primary distance metric in our phylogenetic tree constructions.



The K2P Model

Fig. 1. The K2P Model, on which the scoring metric is based.

To calculate the DS_{dist} of two taxa t_1 and t_2, it is required that t_1 and t_2 are properly aligned. The *Dynamic Score* is calculated as follows:

$$t_1, t_2[i] = \left\{ \begin{array}{cc} match & score+ = match\_score \\ gap & score+ = gap\_score \\ transition & score+ = \alpha \\ transversion & score+ = \beta \end{array} \right\}$$

Since pairwise alignments produce scores and not distances, and since higher scores often correspond to higher similarity, it was deemed necessary to normalize the scores in order to yield a suitable distance. Normalization gave us highly specific distances between 0 and 1. It also solved another problem: the quality of a score is often dependent on the size of the biological data set. For example, a score of 100 would be relatively poor when the sequences involved are > 1000 bps in length. Similarily, a score of 30 would be relatively good when the sequences are ≈ 10bps in length. For this reason, it was necessary to express the score as a function of length. It is hypothesized that this metric will yield distances of a greater accuracy than other metrics used. The DS_{dist} is caclulated as follows:

$$DS_{dist} = \frac{Score_{Max} - Score}{Score_{Max}} \quad (1)$$

Where $Score_{Max} = length_{seq} \times match\_score$.

As you will later see, the use of the DS_{dist} method yields distances of a higher degree of specificity than uncorrected distances. We will mention later why this is pertinent. For the distance matrix creation step, a total of $\binom{n}{2}$ scores are calculated and stored linearly in associative array container (STL map). The use of this data structure enforces that the matrix remain sorted during creation.

B. Decomposition

The sorted distance matrix then prepares us for the next phase of the RAq Approach: the decomposition step. Decomposition for the RAq approach is heavily dependent on the use of a dynamic q value, that is a modification from that found commonly in threshold graphs. Here, the base threshold is incremented by a constant amount, the q value, to yield several thresholds; elements that fall in those threshold ranges will then be clustered together. For example, if our q value is .1, we will ultimately have 10 clusters, in the range of $[0, .1)$, $[.1, .2)$, $[.2, .3)$ and so on. Increasing the q value will leave us with a smaller number of clusters with a larger diameter (number of elements), while decreasing the q value will leave us with several clusters, but with reduced diameter. The pseudo-code for this procedure is presented in figure 2.

```
void build_tree( new_tree, cluster, q) {
...
new_q = q/decay;
q_ = 0;
first_pass = 1;

while ( e_itr != cluster.end( )  ) {
   if ( (*e_itr).get_dist() > q_ ) {
      if ( !temp.empty() ) {
         if ( first_pass ) {
            newtree.insert(temp, 0); //insert on left
            newtree.left();
            first_pass = 0;
         }
         else {
            newtree.insert(temp,1); //insert on right
            newtree.right();
          }

         temp.clear();
       }
       q_ += new_q;
     }
     else {
        temp.push_back((*e_itr));
        ++e_itr;
     }
}
```

Fig. 2. Code for simple annealing step.

B.1 Problems with Annealing q method

Preliminary benchmarking, however, demonstrated a serious flaw in the Annealing q method. Confirming the results found by Warnow, cluster sizes still remain a little too large to be made into a tree. Increasing the "stringency" (or lowering the value) of our q does not help; large clusters still remain too large, and small clusters now become impossibly small (figure 7). The solution to this problem is to add a horizontal bound, or λ value that will restrict the diameter of the final clusters. If a particular cluster's diameter exceeds the λ bound, then it will be recursively broken down into smaller clusters until it is small enough to be processed. By creating such a horizontal bound, large clusters can be processed efficiently and independently, regardless of the size of other clusters. This prevents smaller clusters from becoming even smaller, or degenerating to the greedy solution. For every cluster that exceeds the λ bound, a new threshold value is calculated, using the following equation:

$$q_{new} = q_{old}/decay \tag{2}$$

From preliminary benchmarking, it was determined that a "good" decay value would be 2, since this prevents clusters from becoming small too rapidly. It should also be noted that a λ value of 2 has been hardcoded into the software implementation of *The RAq Approach*, and that the effect of no other λ and *decay* values have been tested [1]. For future studies, it would be interesting to see how adjusting these values will affect the RAq approach's efficacy as a starting tree method for MP search, or how the topology of the new starting tree will be affected.

With our new thresholding method in place, the annealing code from above can be altered to look like this, yielding the finalized version of our decomposition function (figure 3).

B.2 Taxon Recognition

In order for us to be able to merge overlapping subsets together in later steps of the algorithm, while staying loyal to the structure of our guide tree (which, consequentially, is our decomposition tree), we put a restricting factor on the number of times a certain taxon can appear in the decomposition tree: each taxon can appear at most twice. For example, a sequences of distances such as: $\{1.2,\ 2.3,\ 3.5,\ 4.6,\ 5.7\}$ is considered valid. However, the sequence $\{1.2,2.3,3.5,1.3\}$ is not valid, since three was already introduced twice. While this makes our merging step down the road a lot simpler, it drastically narrows down our search space; for future studies, it would be interesting to see how the removal of the recognition vector would affect the construction of the tree.

With these methods in place, a final decomposition may look like the tree in figure 4.

[1] Since we are dealing with distances, note that a λ bound of 2 indicates that each of the final clusters will contain either two or four taxa.

```
void build_tree( new_tree, cluster, q) {
...
new_q = q/decay;
q_ = 0;
first_pass = 1;

while ( e_itr != cluster.end( )  ) {
   if ( (*e_itr).get_dist() > q_ ) {
      if ( !temp.empty() ) {
         if ( first_pass ) {
            newtree.insert(temp, 0);
            newtree.left(); //insert on left
            first_pass = 0;
         }
         else {
            newtree.insert(temp,1);
            newtree.right();
          }

          if (temp.size() > lambda) {
             build_tree(newtree, temp, new_q);
          }

         temp.clear();
       }
       q_ += new_q;
     }
     else {
        temp.push_back((*e_itr));
        ++e_itr;
     }
}
```

Fig. 3. Code for recursive annealing decomposition.

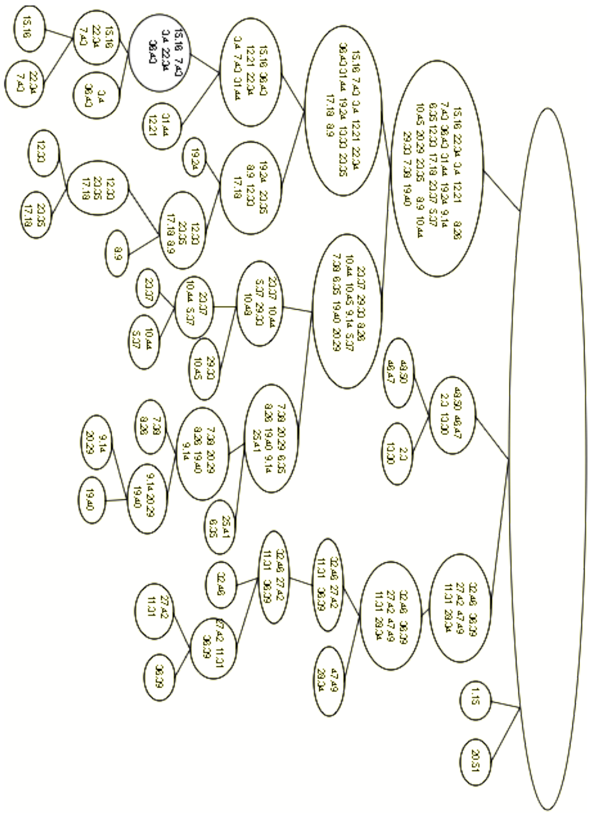


Fig. 4. Decomposition Example: 51 taxa

C. Merging and supertree creation

In order to better represent the decomposition step in code, the multifurcating tree was converted into a binary tree via the left child, right sibling method of tree construction. A multifurcating tree is converted to this as follows: Let T be a multifurcating tree, with subtree children $\{t_1, t_2, .. , t_n\}$, with root r. The binary representation of T, T^b, is represented as follows: t_1 is the left child of r. t_2 is the right child of t_1. t_3 is the right child of t_2, and t_4 is the right child of t_3 and so on. The diagram below indicates the conversion (figure 5).

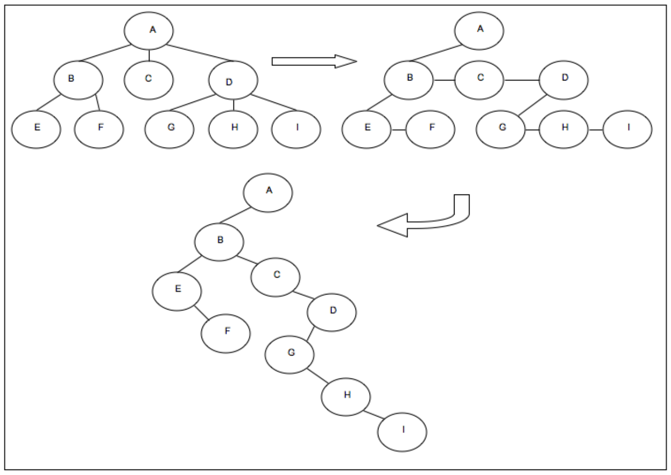


Fig. 5. conversion from multifurcation to bifurcation

As mentioned before, the decomposition tree will now act as a guide tree. However, repeated traversals through our guide tree during the merge stage would be tedious and time consuming; it would be more intuitive to grab the leaf nodes, via a post-order traversal and store them in a list to be merged. However, how does one do this, while keeping faithful to the structure of the guide tree? We refer to this as the numbering problem:

The Numbering Problem
Give a simple and intuitive method to number all the nodes in a tree, so that the structure of the tree would be obvious by analyzing the leaf nodes.

The solution to this is giving each node a unique id. The initial root node always has the id of 0. its children has the ids of $\{1..n\}$. Using this numbering scheme, we end up with a tree structure of ids that looks like figure 6.

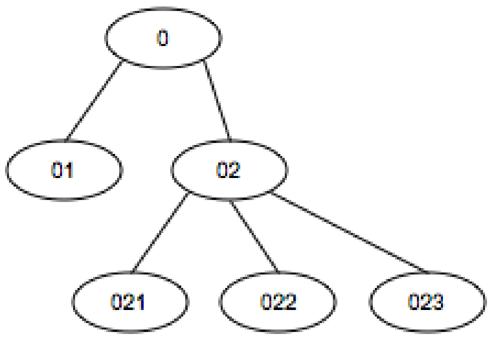


Fig. 6. numbering scheme for *RAq Approach*

By storing the id of its parent, each node remains faithful to the decomposition by which it was created. Therefore, when the leaves are grabbed via post-order traversal and stored linearly in a list, there is an associated ID with each leaf.

C.1 Subtree creation

As mentioned earlier, the λ-bound will cause clusters of either two or four taxa to be clustered together in each leaf node. As also previously mentioned, taxa were allowed to overlap, though minimally so. Therefore, the first step in subtree creation is merging together all the taxa that overlap. For example, looking at our guide tree, we can see that two of the clusters have elements 15.16 and 1.15 that overlap, though they are distantly related. With the assistance of a hash table, these are merged together, yielding 5.16.1. This newly formed merged lement then takes the id of the more-left leaf, which in this case would be [0111111].

The second step in subtree creation is taking all these merged taxa and forming a mini-subtree out of them. In the case where only two taxa are present, they are merged to form a simple tree structure (A,B). However, when more than two taxa are present, for example the list $\{t_1, t_2, t_3, .. , t_n\}$, we form a tree as follows; ((($(t_1, t_2), t_3), ..$) , t_n). These newly formed mini-subtrees each have the ids of the leaves that they belong to; in this manner, leaves that contained two sets of taxa can be merged consistently. The merging of these mini-subtrees into their proper subtrees follows the same methodology as overall supertree creation, which will be described next.

D. Supertree Creation

The now properly formed mini-subtrees are hashed, their ids acting as their keys. Like subtree creation, supertree creation is a two step process. In the initial state, our hash table is filled with the mini-subtrees ready to merged into their proper subtrees. Following with our current trend, and example of what the initial state of the hash table is shown below:

TABLE I
Hash Table

0111111	((15,16),1)
0111112	((22,34),28) - ((((7,43),38),36),39)
011112	((3,4),2)
01112	((((31,44),11),10),45) - (((((12,21),33),29),20),51)
01121	((19,24),40)
0112212	((((23,35),37),5),6) - (17,18)
011222	(((8,9),26),14)
01222	(25,41)
021	(48,50) - (((46,47),32),49)
022	(13,30)
0311121	(27,42)

Notice how the elements belonging to the same leaf are hashed to the same key. We also note the length of the longest ID in the creation of this initial hash table. During the first step, all the keys which have two or more trees associated with it have their list of trees recursively merged into one large subtree. This process is done as follows:

1. Grab first two trees from front of list, t_1, and t_2.
2. Create a a new tree T', and make t_1 and t_2 the left and right subtrees of T' respectively.
3. Push newly formed tree T' to front of list.
4. Repeat steps (1..3), until only one tree remains in the list

The next step involves rehashing. Finding the trees with the longest id, we pop off the last bit of the id, and then rehash the new id and tree into the table. In this manner, the subtrees are always merged together in a manner that is faithful to their decomposition; this is what makes the unique structure of the id so essential.

After rehashing, ids with more than one tree associated with it have their trees merged together, and the value of *longest* (containing length of longest id) is decremented by one. Then the rehashing process is performed again. This alternating cycling of merging and rehashing is repeated until one final tree remains, yielding a final tree represented in newick format.

III. Benchmarking and Results

Benchmarking was performed over four biological datasets, all originating from a plant or mammalian source. These included the famous 500-zilla taxa set from *Chase et. al* and the infamous experiments run by *Rice et. al.* Parsimony Ratchet was chosen as the phylogenetic search framework as choice, and 5 experimental runs(batches) of 200 iterations each were run on each data set. Some information about the data set appears below:

TABLE II
DATA SET INFORMATION

Id	Num. Taxa	Sequence Length	Best Score
1	51	2045	4458
2 (zilla)	500	759	16218
3	921	713	40494
4	1127	N	N

A. *Tree score graphs*

Best tree score found at each iteration was noted. Across all experimental runs, the best tree score for each iteration was averaged together. Best possible score was then subtracted form this average to yield the average deviation from the best, or *steps* from the best (see equation below). These steps were then plotted.

$$steps_i = \frac{\sum_{x=1}^{N} score\_x_i}{N} - best\_score \qquad (3)$$

where $steps_i$ represents the steps at the i^{th} iteration, which ranges from [1..200], N represents the total number of experimental runs, and $score\_x_i$ represents the score at iteration i for experimental run x.

.

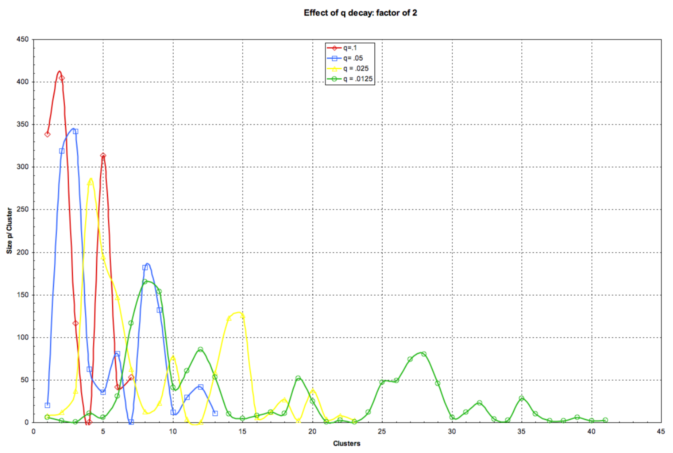


Fig. 7. Preliminary Benchmarking: Affect of q stringency on cluster size. Note how increasing the stringency of the q value causes cluster diameter to approach one very quickly; using a decay factor of 2 causes the cluster diameter to lessen at a slower rate.

IV. DISCUSSION AND FUTURE WORK

A. *Tree Score Analysis*

Tree scores and their steps were plotted over four datasets. For the 51 taxa data set, it can be observed that all four starting tree assisted searches found the best score fairly quickly, with neighbor joining finding the best score almost immediately. The RSA and RAq assisted searches performed a bit worse, and Random assisted search perfomed the worst. Over 200 iterations, however,

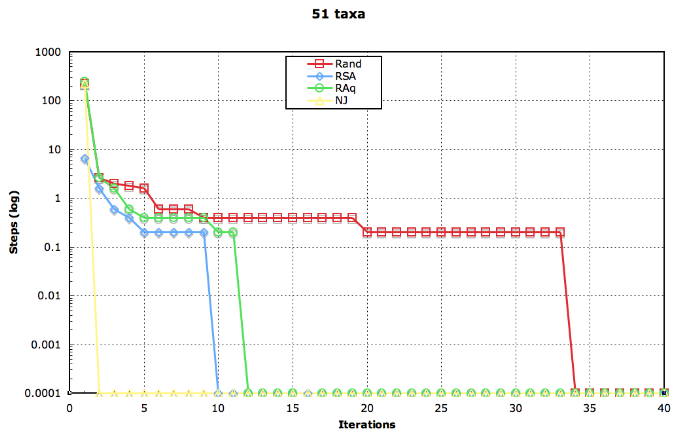


Fig. 8. Benchmark Analysis for 51 taxa - first 40 iterations. NJ finds the best tree score fairly quickly. RSA and RAq perform similarly to each other, followed by Random-assisted search, which performs the worst. However, all approaches find the best score within the first 40 iterations.

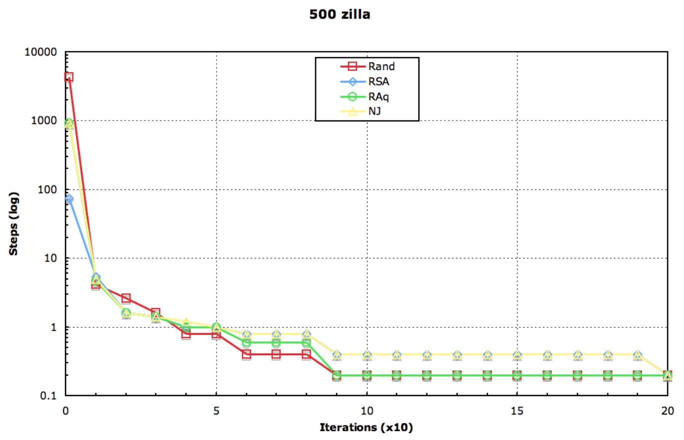


Fig. 9. Benchmark Analysis for 500 taxa (zilla) - overall. Random and Raq assisted approaches find the best score within the first 100 iterations, while NJ and RSA do not find the best score until iteration 199. This demonstrates the competitiveness of the RAq approach.

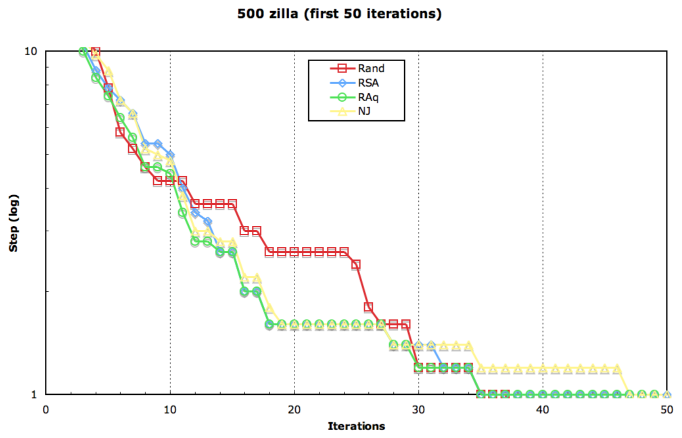


Fig. 10. Benchmark Analysis for 500 taxa (zilla) -first 50 iterations. Demonstrates a close-up of the behavior of RAq and Random assisted searches. RAq initially starts out as performing as the best, until Random passes it around iteration 35.

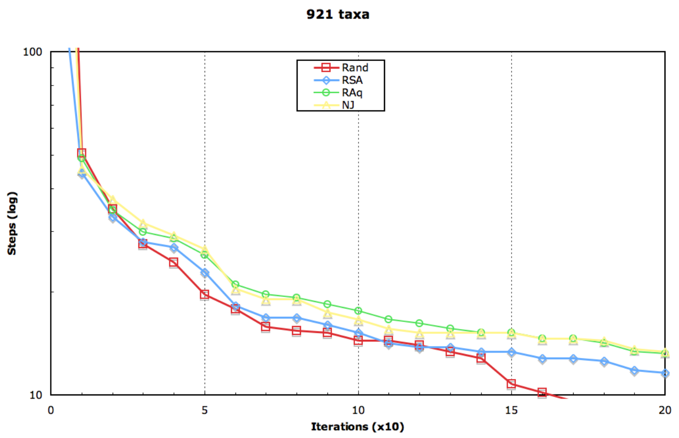


Fig. 11. Benchmark Analysis for 921 taxa. None of the starting tree assisted approaches find the best score in 200 iterations. However, Random by far gets the closet to the best score (9.6 steps away). The *RAq* approach, while outdoing RSA and NJ in the 500 zilla set, loses its lead over RSA in this dataset, and only slightly outperforms NJ.

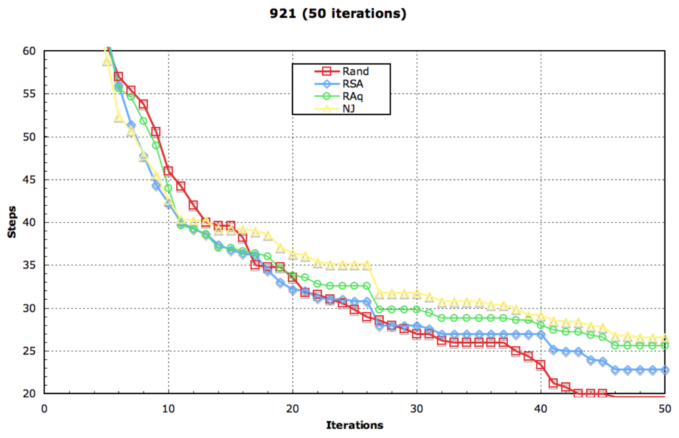


Fig. 12. Benchmark Analysis for 921 taxa - first 50 iterations. Demonstrates a close-up of the different starting tree approaches. Note how for the first 20 iterations, no starting tree approach demonstrates a clear advantage over the other. However, by iteration 25 and onward, the Random tree assisted search demonstrates a clear superiority over the other searches.

all starting tree assisted approaches found the best score.

A bit more separation was achieved for the 500 zilla dataset. Here, both RAq and Random approaches found the best scores far quicker than RSA or Neighbor Joining. If we take a closer look at the first 25 and 50 iterations of the experimental runs on this taxa set, we notice that the *RAq* approach does fairly well, in fact, outperforming the other methods; however, any advantage *RAq* had ends quickly, as the random-tree assisted search does just as well. By the end of 200 iterations, however, all starting tree approaches found the best score, in 4 out of the 5 experimental runs ran.

Our most important data set recorded thus far is the 921 taxa set. With this data set, none of the starting tree approaches reached the best score of 40494. However, the random tree assisted method came the closest, being only 9.6 steps away(table 4). This is significant, since this data set shows the Random-tree assisted search having a superior performance to the other three approaches, a first time in our analysis that we observe such a breakaway.

Upon inspecting the performance of the four starting tree approaches over the three datasets mentioned, we come to the conclusion that, at least from a tree score perspective, Random-tree assisted searches are the best for phylogenetic search in a parsimony ratchet framework. Even though it seems that the random tree assisted search and the *RAq* assisted search tie performance-wise for the 500 zill taxa set, closer inspection even shows the random-assisted search's superiority (table 3). As you can see, the random-assisted search still finds the best tree score faster than *RAq*, though not by much. This seems to suggest that starting tree structure is not relevant for phylogenetic search. However, we need to inspect the results of a tree topology analysis before making any solid conclusions.

TABLE III
500 ZILLA TAXA SET: A CLOSER LOOK. THIS TABLE SHOWS THE EARLIEST ITERATION AT WHICH AN APPROACH HAD X/5 RUNS FIND THE BEST SCORE OF 16218. IF YOU NOTICE, NONE OF THE STARTING TREE METHODS FOUND A BEST SCORE ALL FIVE RUNS; THEREFORE, IN THE SIXTH COLUMN, WE SHOW THE NEXT BEST SCORE FOUND.

	1/5	2/5	3/5	4/5	5/5 (n/a)
Random	31	53	58	87	16219
RAq	51	58	87	90	16219
RSA	58	87	90	199	16219
NJ	58	87	90	199	16219

TABLE IV
921 TAXA SET: A CLOSER LOOK. THIS TABLE SHOWS NUMBER OF STEPS EACH APPROACH WAS AWAY FROM THE BEST SCORE (40494) AT A SET NUMBER OF ITERATIONS. PLEASE NOTE THAT THIS TABLE IS SHOWING INHERENTLY DIFFERENT INFORMATION THAN THE PREVIOUS, AS NONE OF THE STARTING TREE APPROACHES FOUND THE BEST SCORE WITH THIS TAXA SET.

	50	100	150	200
Random	19.6	14.4	10.8	9.6
RAq	22.8	15.2	13.4	11.6
RSA	25.6	17.6	15.2	13.2
NJ	26.6	16.6	15.2	13.4

B. Future Work

For future studies, it would be interesting to study the tree score rates of all the different starting tree methods over more data sets, with ten experimental runs each. It would also be interesting to test the affect of these different approaches in different phylogenetic search frameworks, such as TNT.

It would also be interesting to see how changing the supertree method used by *RAq* would affect its performance; it was noted by Warnow that as the evolutionary dataset gets bigger, the error rate used by the neighbor joining appraoches also increases rapidly. It would therefore be useful to see how using a different supertree method would affect the performance of *RAq* in relation to other starting tree structures.

V. Acknowledgements

This work was supported by a grant through the Computing Research Association (CRA) and the Distributed Mentoring Project (DMP). Special thanks goes to Tiffani Williams for being my advisor and having faith in me and my crazy idea, and the following people for their support during this project: grad mentors SeungJin Sul and Hyun Jung, for their general advice and assistance with *PAUP**; D. Axelrod, E. Glasser-Camp, D. Harrison and Wm. Harris for their advice and help troubleshooting, you made my life fun and bearable. Also a special thanks to anyone else I failed to mention here for their continuous support for me throughout this summer; I sincerely thank you for believing in me.

References

[1] Chase et. al. *Phylogenetics of seed plants: an analysis of nucleotide sequences from the plastid gene rbcL.* Ann.Missouri Bot. Gard. 80, 528-580, 1993.
[2] Felsenstein, Joseph. *Inferring Phylogenies.* Sinauer Associates. 2004.
[3] Goloboff, Pablo A. *Analyzing large datasets in reasonable times: solutions for composite optima.* Cladistics 15, 415-418. 1999.
[4] Linder, Randal C. and Tandy Warnow. *An overview of phylogeny reconstruction.* CRC Press, LLC. 2001.
[5] Nixon, Kevin C. *The Parsimony Ratchet, a New Method for Rapid Parsimony Analysis.* Cladistics 15, 407-414. 1999.
[6] Rice et. al. *Analyzing large datasets: rbcL 500 revisited.* Syst. Biol. 46, 554-563, 1997.
[7] Roshan et. al. *Performance of supertree methods on various dataset decompositions.* Phylogenetic Supertrees: Combining information to reveal the Tree of Life, O.R.P. Bininda-Edmonds, ed., Kluwer Acad. Publ., Dordrecht, 301-328, 2004
[8] Saitou, Naruya and Masatoshi Nei. *The neighbor-joining method: a new method for reconstructing phylogenetic trees.* Mol. Evol. Biol 4(4):406-425, 1987.
[9] Warnow, Tandy. *Disk Covering Methods: improving the accuracy and scale phylogenetic analyses.* CRC Press, LLC. 2001.
[10] Williams, Tiffani L. and Mark L. Smith. *"Cooperative Rec-I-DCM3: A Population- Based Approach for Reconstructing Phylogenies.* 2005 IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB '05), 127-134, 2005.