Phylogenetic Reconstruction Using Competitive Neural Networks

by

George Matthew Fricke

B.A., Anthropology, Appalachian State University, 1996

THESIS

Submitted in Partial Fulfillment of the
Requirements for the Degree of

Master of Science
Computer Science

The University of New Mexico

Albuquerque, New Mexico

August 2003

George Matthew Fricke
Candidate

Computer Science
Department

This thesis is approved, and it is acceptable in quality and form for publication on microfilm:

Approved by the Thesis Committee:

, Chairperson

Accepted:

Dean, Graduate School

Date

ACKNOWLEDGEMENTS

Many have supported me over the past year of work which culminated in this thesis. Most importantly I would like to thank my committee members: my advisor Bernard Moret who has inspired me to do my best work, Laura Salter who has been my unfailing guide and source of encouragement in the field of phylogenetics, and Steven Poe and David Bader for being willing lend their expertise to me whenever I needed it.

I would also like to acknowledge Tom Caudell and George Luger for encouraging me in this thesis when it was still taking form in their Neural Networks and Artificial Intelligence courses.

To my wife, Suzanne, I owe an enormous debt of gratitude for putting up with a husband locked in his study much of the past three years doing homeworks and projects, and finally this thesis, and to my two-year old son Henry who frequently reminds me that everyone needs to pretend to be a triceratops once in a while, and to my as yet unborn son, Leo, who is proof that the evolutionary process is ongoing, vital, and very relevant. This work belongs to them as much as it does to me. Finally to my mother, Christine, I say thank you for instilling in me a sense of curiosity and for encouraging me to follow my interests.

Phylogenetic Reconstruction Using Competitive Neural Networks

by

George Matthew Fricke

ABSTRACT OF THESIS

Submitted in Partial Fulfillment of the
Requirements for the Degree of

Master of Science
Computer Science

The University of New Mexico

Albuquerque, New Mexico

August 2003

Phylogenetic Reconstruction Using Competitive Neural Networks

by

George Matthew Fricke

B.A., Appalachian State University. 1996

M.S., Computer Science, University of New Mexico, 2003

Abstract

Algorithms for reconstructing phylogenies from character sequences encounter enormous search spaces even for small numbers of taxa. Techniques are needed to speed up the exploration of these spaces so that larger problems can be approached. In other problem domains neural networks have provided a framework for tackling complex problems very quickly. This thesis describes a new architecture called KomPhy which makes use of neural networks, provides proofs for the upper (n^2) and lower bounds (nlogn) of running time as a function of the number of taxa and sequence length, as well as a discussion of the expected time complexity (nlogn). Empirically KomPhy is found to be faster and, for birth-death trees, more accurate than previous neural network solutions. Although KomPhy does improve the state-of-the-art for neural net approaches it does not perform as well as other well established algorithms, such as neighbor-joining [1]. A parallel implementation of KomPhy is also presented.

Contents

List of Figures

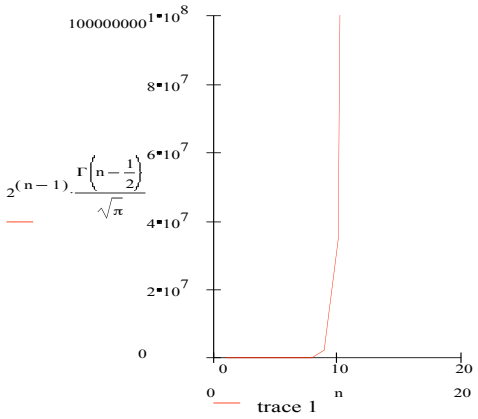
1 Introduction

1.1 Molecular Phylogeny Reconstruction

Phylogenetic analysis is the attempt to reconstruct the historical evolutionary relationships between biological organisms. Over the past few years the demand for phylogeny reconstruction has increased tremendously: epidemiologists need to be able to trace the rapid course of a new virus' evolution quickly enough for that knowledge to inform the creation of treatments, lawyers have used phylogenetics in court cases to establish family relationships, and the study of biological diversity is increasingly dependent on phylogenetic analysis to target conservation efforts [2].

Until genotypic data became available directly, endeavoring to reconstruct phylogenies had been the exclusive province of biologists who used comparative anatomy and the fossil record to reconstruct evolutionary trees. Technologies which exposed DNA and proteins to direct observation allowed more quantitative methods of reconstruction to be proposed. Rather than looking exclusively at the phenotype expressed by an organism it became possible to look directly at the molecular coding that determined that phenotype. The molecular sequences used to reconstruct phylogenies can consist of thousands of characters and the possible number of trees that can be inferred from those characters grows rapidly with the number of taxa being analyzed (figure 1). This shift from qualitative to quantitative analysis necessitated the use of automated computation. Over the past quarter century computer algorithms have therefore come to dominate the process of reconstructing the hereditary relationships between organisms.

n	N_R
2	1
3	3
4	15
5	105
…	
9	2,027,000
10	34,460,000
…	
25	1.193×10^{30}



n = Number of Taxa

$$N_R = \prod_{i=2}^{n} (2i - 3)$$

Figure 1: There are $2^n n!$ possible binary trees (O(n!) where n is the number of taxa being analyzed. If N_U is the number of unrooted trees then the number of rooted trees, N_R, is $N_U(n - 1)$.

The variable *n* above is the number of species we would like to place in an evolutionary tree while N_U is the number of possible way of arranging those species in an unrooted binary tree. If at the time of the big bang the fastest computers operating today had started a brute force reconstruction (examining each of the N_R tree topologies) of an evolutionary tree linking only 25 species they would still be far from finishing today. Many current projects require the reconstruction of phylogenetic trees consisting of hundreds of taxa. Tackling this enormous search space quickly and accurately is the primary challenge to phylogenetic reconstruction techniques today.

1.2 Neural Networks

A wide variety of approaches have been applied to the problem of phylogenetic reconstruction, [3]. However, there is a remarkable paucity of research into the application of unsupervised neural network clustering algorithms to phylogenetics. The primary strength of neural networks is their ability to produce solutions very quickly even in complex domains. A drawback of neural networks in many domains is that solutions are often approximate. A self organizing map [3] (SOM) algorithm called SOTA [2] is the one published effort to use an unsupervised network to reconstruct phylogenies. The algorithm described in this thesis provides an alternative neural network approach in order to place the SOTA algorithm in some perspective and to further analyze the applicability of this technology to phylogenetic analysis.

Neural networks are biologically inspired weighted graphs, in which nodes are analogous to neurons in the nervous system, edges perform the role of dendrites and axons, and weights mediate information flow in a functionally similar way to synapses. A weight is associated with every input coming into every neuron. These networks follow the ideas put forth by Ramón y Cajál in 1911 [4] who first described the brain as constituted of simple interconnected computational components acting in concert. Neuroscience has added a great deal to our knowledge of how the brain works since 1911, but the basic idea of massively parallel computational systems composed of very simple interconnected and adaptive information processing units has been so successful that the biological complexity discovered over the past few decades is largely ignored (though, identification of novel structural elements in neural computation in the brain has sometimes led to new artificial neural network designs).

At the simplest level a biological neuron is an information processing unit which works with other neurons to solve computational problems. Each artificial neuron in a neural network consists of scalar inputs and outputs and an arbitrary function, often a sigmoid, which modifies the inputs to create the output. These nodes, or neurons, are connected to one another via edges which map the output of one neuron onto the input of one or more adjacent neurons. Each input is associated with a weight which can be modified to increase or decrease the influence of that input on the computation performed by the associated neuron.

The weights associated with node inputs are analogous to the role of synapses in a biological system which can be dampened or excited by the neuron's environment. Neural networks adapt by modifying these weights such that the difference between the neural networks' actual output and the desired output is minimized. The process of measuring error and adapting weights can be supervised, in which case the desired output is explicitly provided by the designer, or unsupervised, in which case the measurement of error and resultant modification of input weights is directed by the network itself.

Almost all neural networks operate in two phases; first they identify neurons in the system which have a particular impact on the network's output as a whole (the impact can be either positive or negative depending on the architecture); secondly a learning rule is applied to those neurons to either enhance or subdue their influence on the result.

The earliest example of an unsupervised neural network is the competitive neural network. Competitive neural networks were inspired by the observation that in some clusters of neurons in the brain only a particular subset will fire given a particular input. These active neurons appear to dampen the firing of axons around them when presented

with the appropriate inputs. This idea of having neurons associate themselves with a particular input to the exclusion of surrounding neurons gives rise to the basic competitive neural network architecture.

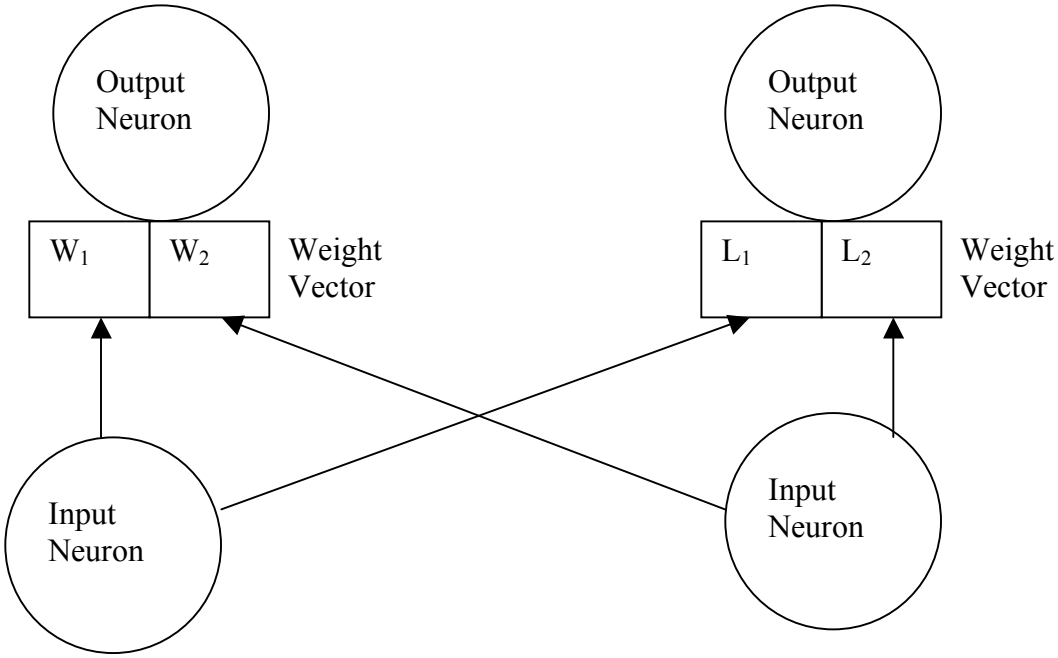


Figure 2: Schematic of a neural network. Notice that in the fully connected case the weight vectors will have as many components as there are input neurons [34].

It is important to understand that while neural networks are conceptually comprised of neurons connected by weighted directed edges they are almost never implemented that way. Often, a neural network is implemented such that each weight vector is represented as a row in a matrix, these rows are updated through matrix multiplication with the input vector. For the neural network architecture described in this thesis, the illusion of nodes and edges will be maintained even when the implementation consists entirely of vectors in a *weight space* -- a d dimensional vector space, where d is the number of input nodes and where each point in the weight space corresponds to a

potential input sequence or set of weights. Since for each neuron there are as many weights as input components both weights and inputs can be mapped into weight space.

Artificial competitive neural nets consist of two types of networks overlaid. First a Hemming net calculates the weighted sum of the inputs to the network, usually as a dot product $\sum wi = w \cdot i = \|w\|\|i\| \cos\theta$ at each non-input node, where w is a weight vector, i is the current input vector, and θ is the angle between i and w.

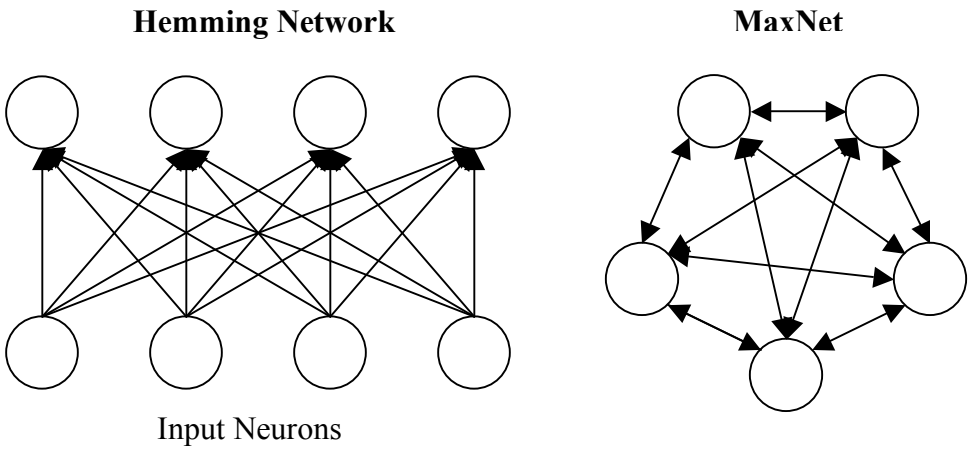


Figure 3: The Hemming network passes the input from every input node to every output node where weights and a processing function are applied resulting in values generated by the output nodes. The MaxNet allows only the neuron with the maximum fitness to display its result.

Then a MaxNet ensures that only one output neuron will actually display its output. This is accomplished by having the neurons suppress adjacent neurons whose dot product output is lower than their own. In a fully connected MaxNet this results in a single *winning neuron*. Together these two networks comprise a competitive neural network.

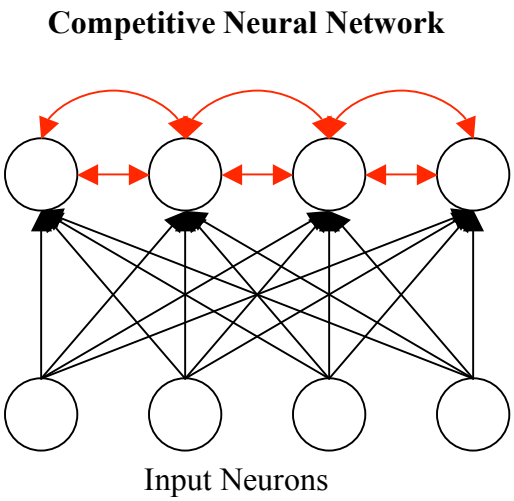


Figure 4: The Hemming network and the MaxNet together form a competitive neural network.

The MaxNet operating on the dot product results of the Henning network chooses the neuron with the largest dot product. The dot product of the weight and the input will be largest where the weight vector has the most pairwise similarities to the input vector.

Let the value of node i be X:

until X_i doesn't change **do**

foreach i **do**

$X_i = X_i + \Sigma_{j \neq i}(\omega X_j)$ where ω is negative

end

end

Figure 5: The MaxNet algorithm converges to X_i being 0 for all i but one. [5a]

This selection process results in a winning neuron that is closest to the input vector in weight space. Having identified the neuron which is most similar to the input sequence the network then applies a learning rule which further enhances the affinity of the winning neuron for the current input sequence. Various learning rules are available; all modify the weights associated with the winning neuron so that it and the input sequence have a higher dot product.

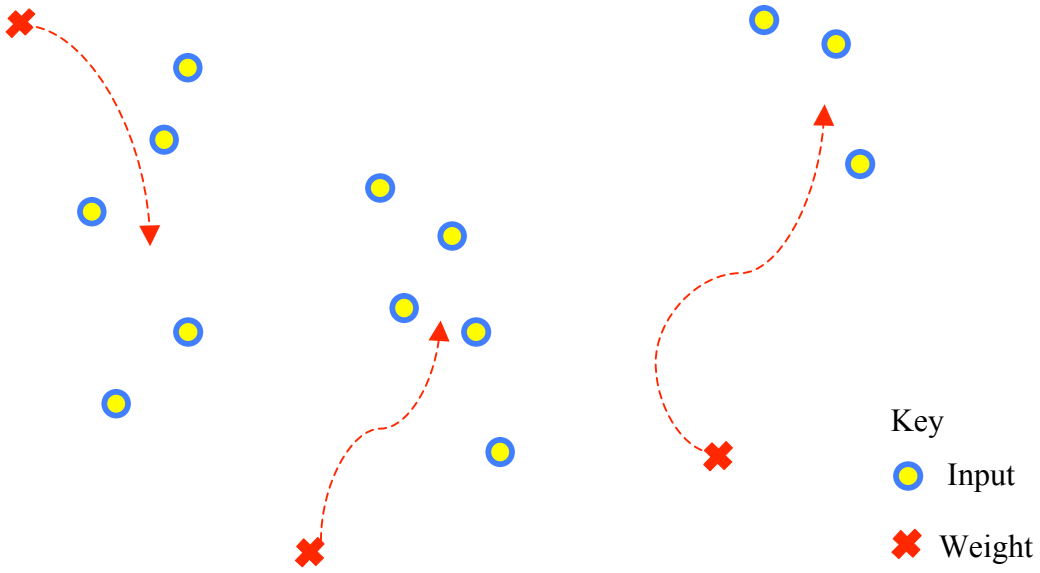


Figure 6: Weight (neuron) movement towards input clusters in a two dimensional weight space.

One common learning rule simply adds the difference between the winning neuron and the input sequence to the winning neuron. On its own this would result in the weight exactly equaling the input.

Learning rule: $weight_{new} = weight_{old} + \alpha(input - weight_{old})$

Having the weight simply equal the input vector would stop the network from being able to generalize. Each output neuron's weight vector should ultimately come to approximate

the center of the set of input sequences that it has become associated with. The weights of the winning neuron are therefore moved towards the input sequence by some fraction of the distance separating them. This fraction α is referred to as the *learning rate*. The optimal value for α is usually determined empirically.

Once a winning neuron has been selected and updated the next element in the set of sequences being analyzed is *presented* to the network as a series of values in the input nodes and the cycle is repeated with a new winning neuron being selected based on its similarity to the input and updated to enhance that similarity. From presentation to presentation the neuron selected to learn may be selected over and over again or a variety of different neurons could win. Each of these presentation-learning cycles comprises an *epoch*.

One particular problem that may be encountered is that of *dead neurons*. Dead neurons are those neurons whose initial state is such that they are further away from all the input sequences than any other neuron. Since the dead neuron will never be selected to learn it remains unchanged throughout the epoch. The final state of a dead neuron may communicate useful information about the distribution of input sequences if the initial states were chosen to contain some discernable bias. If neurons are initialized to random locations little information can be gained from the existence of dead neurons since they were almost certainly left out of the learning process due to being randomly assigned far from any of the input sequences. If however the initial locations of the neurons are all the same and in a neutral location then the existence of dead neurons indicates that the input sequence locations are so close together relative to the size of the weight space that the individual inputs appear equidistant. This results in the first neuron to be associated with

an input sequence winning when presented with all subsequent inputs and all other nodes becoming dead neurons.

The standard solution to the problem of dead neurons is the use of a *conscience* parameter [6]. The network conscience keeps a record of the ratio of wins to losses for each neuron and begins to artificially improve the chance of a neuron winning if it has lost during a high percentage of the input presentations. This ensures that every neuron will be moved towards the inputs so that the maximum number of potential partitions is not artificially reduced. Typically the conscience parameter is an exponential function that ensures a win if a neuron loses 100% of the time.

A particular application of competitive neural networks is in the partitioning of samples into discrete sets. Such networks are called *partitive neural network,s* and they have been applied to a number of diverse problem domains [7] Partitive networks effectively partition the input space into *n* partitions where *n* is the number of neurons in the system.

Competitive neural networks form the basis of the KomPhy architecture while SOTA is based on Self Organizing Maps [8]. Both approaches are based on neurons being mapped into the input space and then migrating through successive applications of a learning rule until the average distances between the neurons and the inputs are minimized. Competitive neural networks are the oldest and simplest form of unsupervised neural network, while self organizing maps are a much more sophisticated approach which builds on the competitive aspects of the competitive neural network while introducing a number of new concepts such as neighborhood. The SOM architecture encodes the relationships between large numbers of nodes in topologies such

that each neuron has a neighborhood of adjacent neurons whose output it can cause to be enhanced or inhibited at different stages in the presentation-learning cycle. SOMs are capable of mapping the distribution of input vectors onto its node graph effectively reducing the dimensionality of the data in much the same way Multi-Dimensional Scaling (MDS) does.

SOTA organizes the graph linking the competitive neurons in the standard SOM framework into a bifurcating tree rather than the more usual lattice. The neurons then migrate and map the probability distribution of the input character sequences while maintaining a tree structure through the neuron links. SOTA is based on Kohonen's unsupervised neural network of self-organizing maps and on Fritzke's growing cell structures algorithm to construct phylogenetic trees [9].

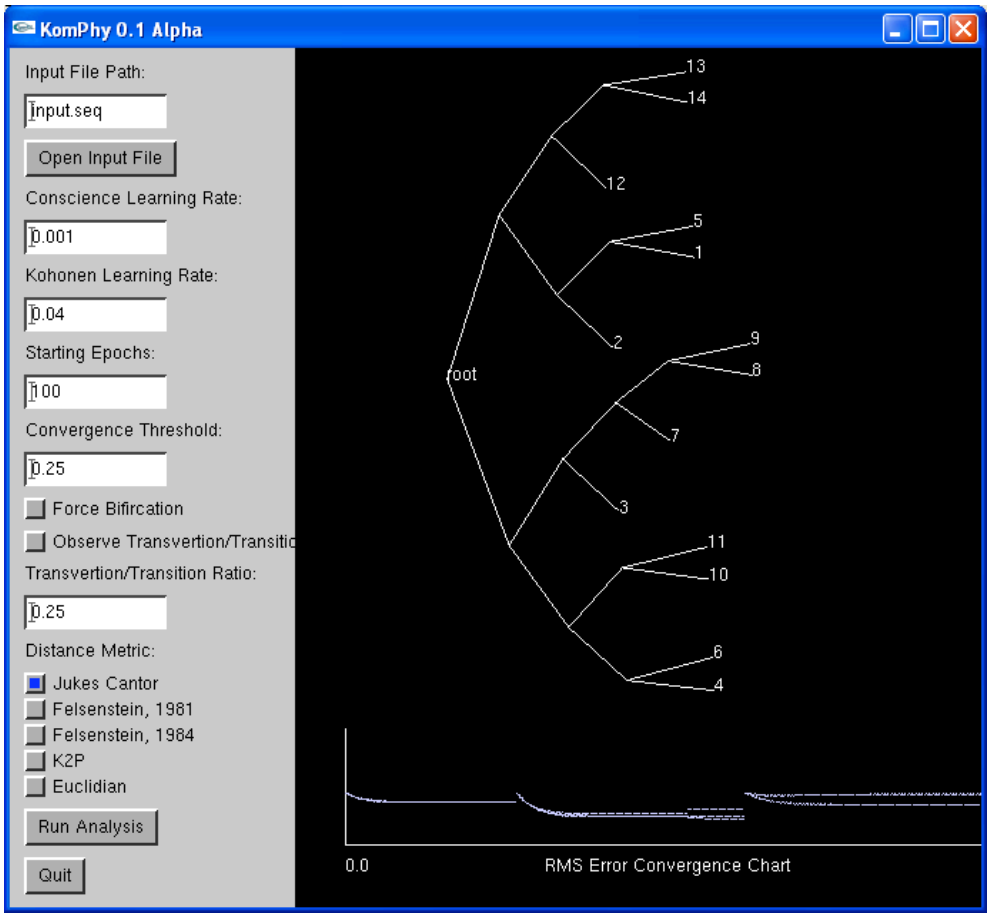
Supervised neural networks (typically back propagation architectures) and other self organizing maps have been applied to protein classification as a precursor to gene expression analysis but not to phylogenetic reconstruction per se ([10, 11, 12, 13, 14]). The SOM approaches discussed in these papers, with the possible exception of [14] are adaptations of SOTA.

Hierarchical neural networks [5], of which KomPhy could be considered an example, consist of multiple neural networks embedded into a graph. The networks within the graph can be single neurons or complex neural architectures. Tree-structured neural architectures are a special type of hierarchical neural network that have been used extensively in principal component analysis (PCA) which has itself been used in gene expression clustering [15, 16]. Decision trees, hierarchies of experts, classifiers and

especially scene graph classifiers are typical applications for hierarchical neural networks [17].

By developing another unsupervised neural architecture, which is simpler than SOTA, and comparing its performance to SOTA as well as to the standard fast phylogenetic algorithm Neighbor-Joining [1], the applicability of unsupervised neural networks to the domain of phylogenetics can be better understood.

2 Kohonen Competitive Phylogenetics (KomPhy)



2.1 Approach

The KomPhy architecture is a greedy clustering algorithm that uses the same framework as the quicksort sorting algorithm [19]. The partitioning function is a two neuron unsupervised competitive neural network. Sequences are mapped into real valued vectors which at each stage in the algorithm are segregated into two clusters. New neural networks are created as needed to partition each of these clusters into a further two partitions. The algorithm proceeds recursively until each partition contains only one or two sequences. Both of these cases are trivial since there is only one phylogenetic arrangement for one and two taxa. The resulting function call tree is the proposed

phylogenetic reconstruction. KomPhy could be described as a dynamic hierarchical network architecture though the component neural nets are typically heterogeneous and fixed in traditional hierarchical networks.

Encoding of input sequences such that they can be operated on by the neural networks is of central importance since it is that representation which will define the topology of the solution space. Here DNA sequences are mapped into the real-valued weight space by representing each character as a point within the volume of a tetrahedron. The apices of the tetrahedron represent the four bases. This representation allows neuron weights to represent intermediary values between the normally discrete bases and to move smoothly within the solution space. The difference between the current state of a character in a weight vector and the corresponding character in an input vector is easily determined by looking at the Euclidean distance between the two within the tetrahedron. Using the Euclidian distance ensures that the bases are equidistant from one another and from the center of the volume.

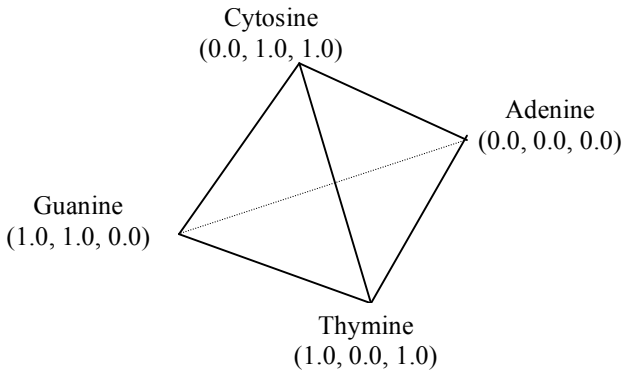


Figure 7: Representation of a character as a three component vector within the volume of a tetrahedron in which the apices are four nucleotide bases.

Given the Euclidean distance between a character in the weight vector of a node and the corresponding character in a DNA sequence the total distance between the weight and the sequence can determined. The traditional Kohonen learning rule [20], in which the distance measure is the dot product of the two sequence vectors, was used in an initial version of KomPhy but Euclidean distance and phylogenetic distance metrics such as Jukes-Cantor [21], have also been applied successfully. More complex distance metrics designed specifically for nucleotide sequences (F84 [22] and the Tamura-Nei [23] metrics) were also implemented but involve taking the log of a negative when the distance is too large, this make them difficult to apply when two sequences are very different from one another, as is the case with the initial starting condition of the neuron weights. Further work, perhaps involving the use of a simple metric initially followed by a more complex one as the weights converge, needs to be done before F84 and Tanamura-Nei can be used reliably with KomPhy.

All phylogenetic metrics were modified to operate on real-valued characters rather than the more usual discrete ones. The Euclidean distance is a close approximation to unweighted parsimony in a continuous solution space. The SOTA program infers distances between sequences by observing the ratios of character pairs at corresponding sites and encodes the input sequences as a vector of the resulting probabilities. Comparisons of Jukes-Cantor model correction and the Euclidean distance function as well as the results generated by SOTA are explored in the results section.

A previous version of the algorithm performed random restarts of the initial neuron weights and ran the network successively with twice as many epochs as the previous iteration until two consecutive runs yielded the same partitioning. This process

would help to avoid local minima (of which phylogenetic tree space can have many [24]) and provided a dynamic criterion for deciding when to stop iteratively presenting each network with samples. A related technique common to neural networks was also tried in which the initial weights are set to two of the input taxa. The initial weights determined in these two ways were found to completely determine the resultant topology. Empirically, using neutral initial values for the network weights generated the same topology as the best of the randomly initialized weights. The neutral points were found by setting every character in the weight's vectors to be the center point of the tetrahedron which defines base-pair space. This point is equidistant from all possible input vectors and so does not introduce an unnecessary bias on the initial relationship between weights and the sequences being analyzed.

Since the weights are initialized to neutral locations, KomPhy does not generate any dead neurons during its tree reconstructions. When random weights were used, dead neurons occurred in a small but significant portion of the reconstructions. In those experiments where a conscience parameter was used to alleviate this problem, accuracy was degraded by 50 percent or more, even when no dead neurons were present. The lack of dead neurons when using neutral initial weights indicates that very little structure in the data (as little as one character difference) is sufficient for KomPhy to partition the sequences.

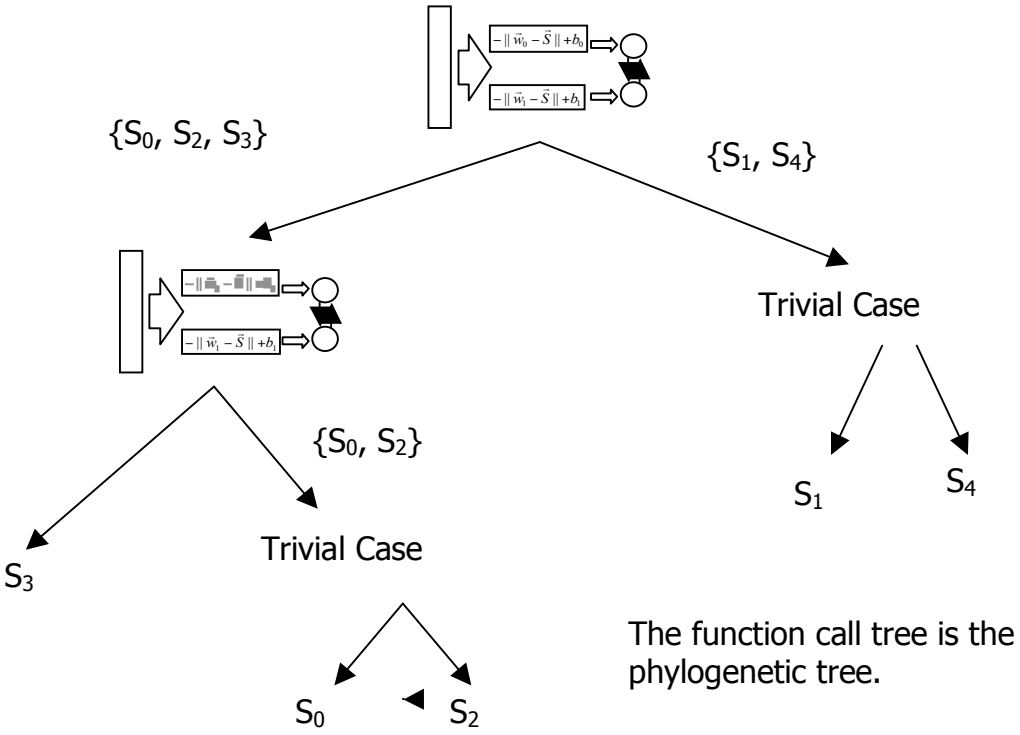


Figure 8: The KomPhy architecture. The set S_i in the graph above is the input sequences. The figure to the left shows how the algorithm partitions the input recursively until the base cases are reached. S0 = (acg...), S1 = (tac...), S2 = (cgc...), S3 = (ggt...), S4 = (act...).

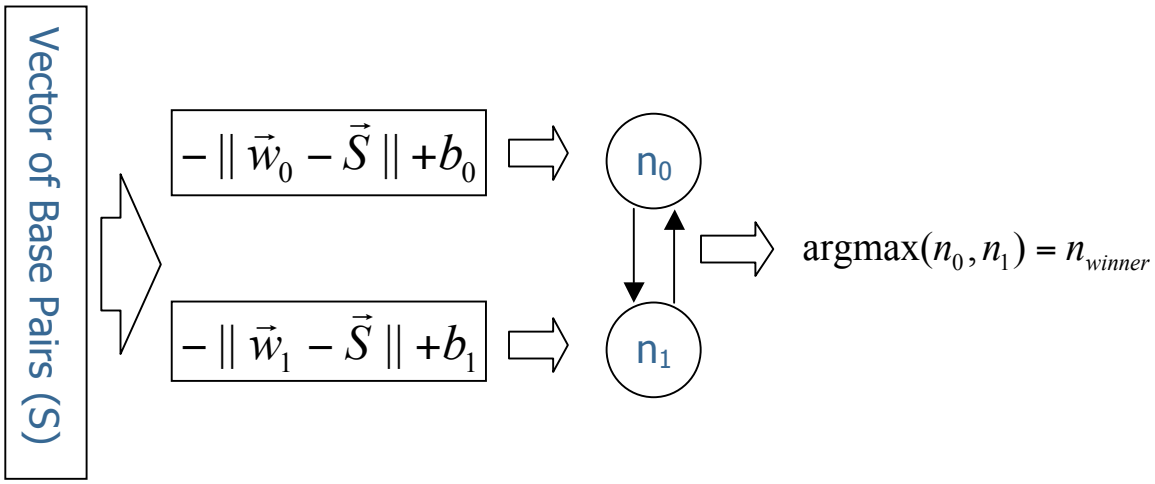


Figure 9: Each partitioning neural network shown in figure 3 is a competitive network which maps the input sequence into an $|S_i|$ dimensional space (where $|S_i|$ is the length of the sequences). The distance from each sequence in the sequence space to the weights in weight space is then calculated. The neuron with the weight vector closest to the sequence becomes identified with that sequence and

is moved closer to it in weight space by the learning rule (Equation. 1), b_i is the bias for the i^{th} weight. Biases are used to direct weight convergence, with a conscience parameter for example. `Argmax` returns the larger of its two arguments, i.e. the neuron which is most similar to the input.

$$\boldsymbol{w}_{winner}(t + 1) = \boldsymbol{w}_{winner}(t) + \alpha(\mathbf{S}(t) - \boldsymbol{w}_{winner}(t)) \quad (1)$$

Equation 1 shows the Kohonen learning rule [8] which is applied to the weights of the winning neuron and serves to move the neuron in weight space closer to the last input sequence. $\vec{w}_{winner}$ is the vector of weights associated with the winning neuron. t is the current time step. $\vec{S}$ is the input sequence and α is the learning parameter.

Input: a set of n sequences, S, with $|S_i| = k$,
a competitive neural network consisting of *w* weights, W,
a function **D**(W_i,S_j) that returns the distance between W_i and S_j using some metric.
a learning function **L**(W_i, S_j) as defined in *Equation 1* above.

Output: *w* partitions where **D**(W_i, S_j) for all S_j members of P_i is less than **D**(W_i, S_p) for
any S_p not a member of P_i.

```
begin Partition( S )

if |S| < 3 then
        base case
        return;
end

while counter < epochs do
        while j < k do
                winner = argmax( foreach weight Wi do D(Wi, Sj) end )
                push(Pj, Wwinner)
        end

        foreach weight Wi
                foreach  element v of Pi
                        L(W1, v)
                end
        end

        counter ← counter + 1;
end
```

Figure 10: Algorithm for the Partition function in KomPhy

2.2 Time Complexity

The Quicksort framework makes the running time analysis easy since the partitioning step is linear in the number of characters in the sequence.

In the following discussion n = |S|, m = $|S_i|$, where S is a set of sequences, each of which is a set of characters. The worst-case running time occurs when the partitions, P, are unbalanced such that only one sequence is in P_1 and |Si| - 1 are in P_2. This leads to the internal nodes doing *n* - *depth* work for each partition or $n + (n\text{-}1) + (n\text{-}2) + \ldots + (n - n + 1)$ total partitioning cost, which is $O(n^2)$. In the best case each partition costs $n/2^{depth}$ or $n\log(n)$ total cost. The running time is therefore dependent on the partition sizes at each step [25] If we assume the partition sizes are drawn from a uniform distribution, then the probability of a particular partition size is $p(|n_i| \leq 2) = 3/n$ for the trivial partition sizes (i.e. the recursion base cases) and $p(|n_i| = k) = 1/n$, $k = 3, 4, \ldots, n - 1$ for partitions which are large enough to be repartitioned. This observation yields the following recurrence relation:

$$T(n) = \frac{1}{n}\left(T(1) + T(n-1) + \sum_{d=1}^{n-1} (T(n-d) + T(n)) \right) + \Theta(n) \quad (2)$$

(T = running time, n = input size)

Using the earlier result that the worst case running time is $O(n^2)$, we can state that:

$\frac{1}{n}(T(1) + T(n-1)) = \frac{1}{n}(\Theta(1) + O(n^2)) = O(n)$ now the θ(n) term in Equation 2 can be used to absorb the $\frac{1}{n}(T(1) + T(n-1))$ term yielding $T(n) = \frac{1}{n}\left(\sum_{d=1}^{n-1} (T(n-d) + T(n)) \right) + \Theta(n)$.

Solving this recurrence relation yields an expected running time of $O(nm\log(n))$ for a uniform distribution of partition sizes [26]. There is still a constant-factor difference

between the expected running time and the lower bound on the running time. The running time then depends heavily on the depth of the tree being formed.

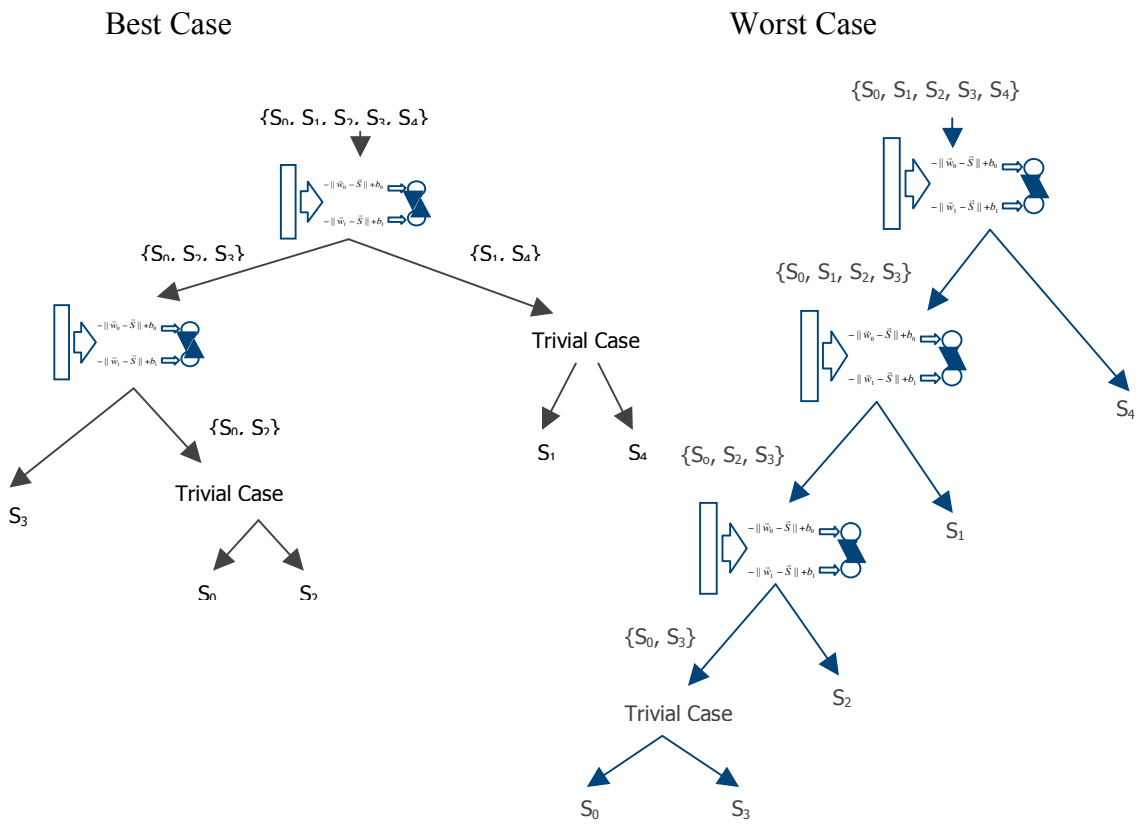


Figure 11: Best and worst case tree structures for time complexity. The worst case tree results in $O(n^2m)$ running time while the best case running time is $\Omega(nmlogn)$.

3 Experimental Results

3.1 Methodology

Tree Generation

KomPhy, SOTA, and Neighbor-Joining were run over numerous datasets, each of which consisted of 96 trees with the first tree having four taxa, the next five taxa, and so on until the 96th tree had 100 taxa. Two tree-generation models were used: Birth-Death and Uniform. Birth-Death trees are generally more symmetric (balanced) than uniform; this difference makes it possible to test the effect tree balance on KomPhy's accuracy and running-time. Ultrametric birth-death trees were generated with Phylogen 1.0 [27], uniform trees, in which all topologies are equally likely, were generated with Component 2.00a [28]. Phylogen was used with a simple birth-death process with a constant birth rate and death rate. The branch lengths of the birth-death trees ranged from 0.005 to 1.2. The branch lengths for uniform trees were drawn from a uniform distribution of values from 0.0 to 1.0. Here branch length is defined to be the expected number of transitions per site in a DNA sequence. A branch length of 0.05 would indicate that five characters would be expected to change value in a sequence of 100 characters.

Sequence Generation

Character sequences were generated from trees with Seq-Gen 1.2.5 [29]. During the sequence generation process, branch lengths were scaled by 1.4, 1.2, 1.0, 0.8, 0.6, 0.2, 0.1, and 0.05 to create differing sets of sequences. Sets of sequences were created with 1,000 characters each in most cases. Sequences with 500 and 2,000 characters were also generated for comparison. All sequences were generated using the Jukes-Cantor model.

Accuracy Measure

KomPhy, SOTA, and Neighbor-Joining were run over all data sets and the reconstructed trees compared to the model tree. Several difference measurements were considered, *agreement subtree*, the *Nearest-Neighbor interchange* metric, the partition metric, *quartet dissimilarity* measure, the *triplet dissimilarity* measure and the *Robinson-Foulds* metric [32]. None of these methods deviated substantially from the Robinson-Foulds distance given by the *treedist* program in Felsenstein's Phylip 3.6 alpha suite [30], which was ultimately used to score the success of the algorithms. Neither birth-death nor uniform trees can be considered to accurately represent the phylogenetic trees seen in nature, [31] however, they do provide some insight into the types of topologies these algorithms are able to recover.

Experiment Platforms

The serial implementation experiments were conducted on an Intel Pentium III running at 500 megahertz with 128 megabytes of RAM running Debian Linux using the GNU. Parallel experiments were performed on eight nodes of the Blackbear VA Linux cluster at the University of New Mexico's Center for High-Performance Computing (HPC@UNM). Blackbear has 32 Pentium III processors running at 550 megahertz each with 500 megabytes of RAM per node. Nodes in the Blackbear system are interconnected through Myrinet. The GNU g++ compiler was used with optimization flags.

Neighbor-joining Implementation

The neighbor-joining program used was that found in Felsenstein's Phylip 3.6 alpha package [30]. Neighbor-joining was run with Jukes-Cantor model correction.

3.2 Results

Model Correction

The accuracy and speed of KomPhy relative to SOTA and neighbor-joining was evaluated over various average branch lengths, generating tree topologies, and model corrections. KomPhy was run using a Euclidean distance metric and one based on the Jukes-Cantor model correction. Overall the Jukes-Cantor metric showed minimal improvement over the Euclidean metric and in many cases the Euclidean metric actually resulted in shorter Robinson-Foulds distances. Running time was not adversely changed by using the Jukes-Cantor model correction.

Comparison of KomPhy over Different Tree Topologies

Five pairs of trees with 64, 128, 256, 512, and 1,024 taxa each were designed such that one tree in the pair (tree A) has depth equal to the number of taxa (a caterpillar tree) and the second tree (tree B) has depth equal to $\log_2$(number of taxa) (complete balanced tree). Tree A in each pair is the deepest tree possible while tree B is the shallowest. Branch length was fixed at 6 changes per hundred characters. These two trees are at the extremes of topology and should provide some insight into the influence of topology on

running time and accuracy.

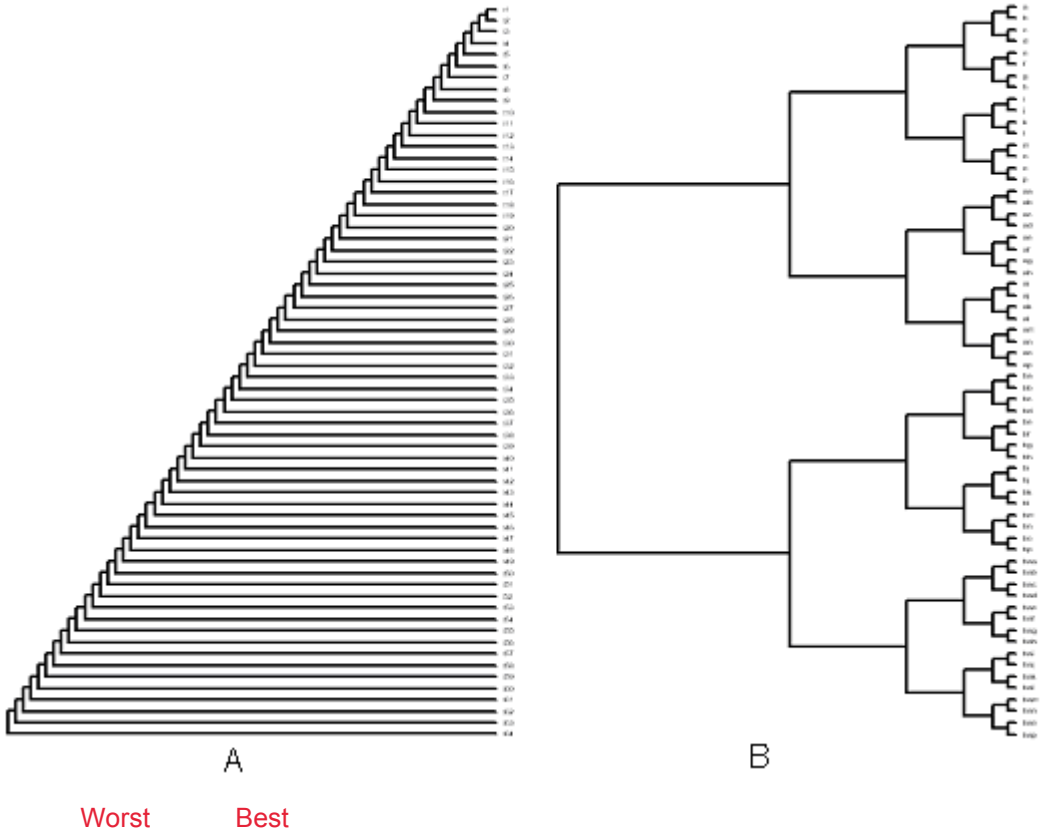


Figure 12: ~~Best~~ (A) and ~~Worst~~ Case (B) Trees for KomPhy running-time.

As suggested by the analysis in Section 2.2, running time was consistently larger for deeper trees in the idealized cases above. However, the absolute difference in running time was small: for 64 taxa the difference in running time was 3%, for 128 it was 4%. Figure 14 plots the running times for uniform and birth-death trees. By looking at the reconstructed tree produced by KomPhy from the worst model tree it can be seen that the fast running time was the result of KomPhy flattening the tree and making it more symmetric. This bias in KomPhy's reconstruction serves to ensure that the worst-case running time is unlikely to occur since KomPhy is biased towards reconstructing trees so that they are more balanced than they should be.

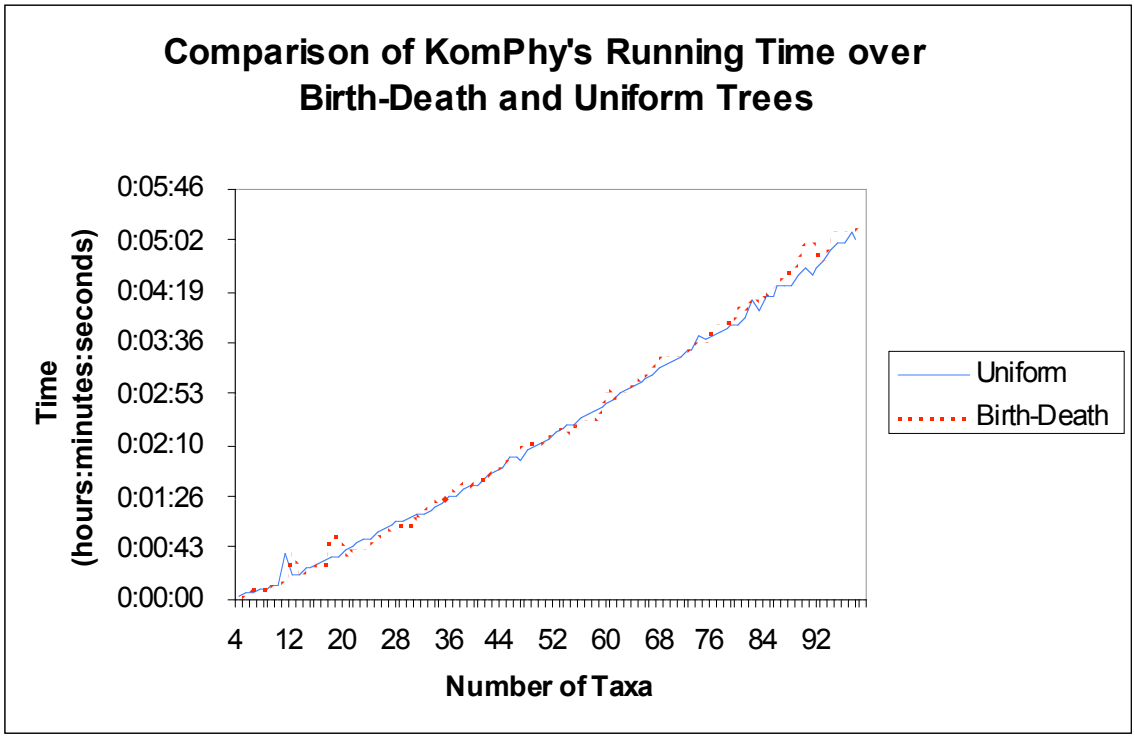


Figure 13: Little difference between the running time for birth-death trees and uniform trees is discernable.

As expected, the accuracy of the KomPhy algorithm is highly dependent on the degree to which the generating trees topology deviated from the symmetric. A *full* tree (completely balanced) was reconstructed with a RF error of 0.09, the tree of maximal depth was reconstructed with an RF error of 0.42. This relationship is further borne out by comparing accuracy of KomPhy when it reconstructs birth-death trees vs. uniform trees.

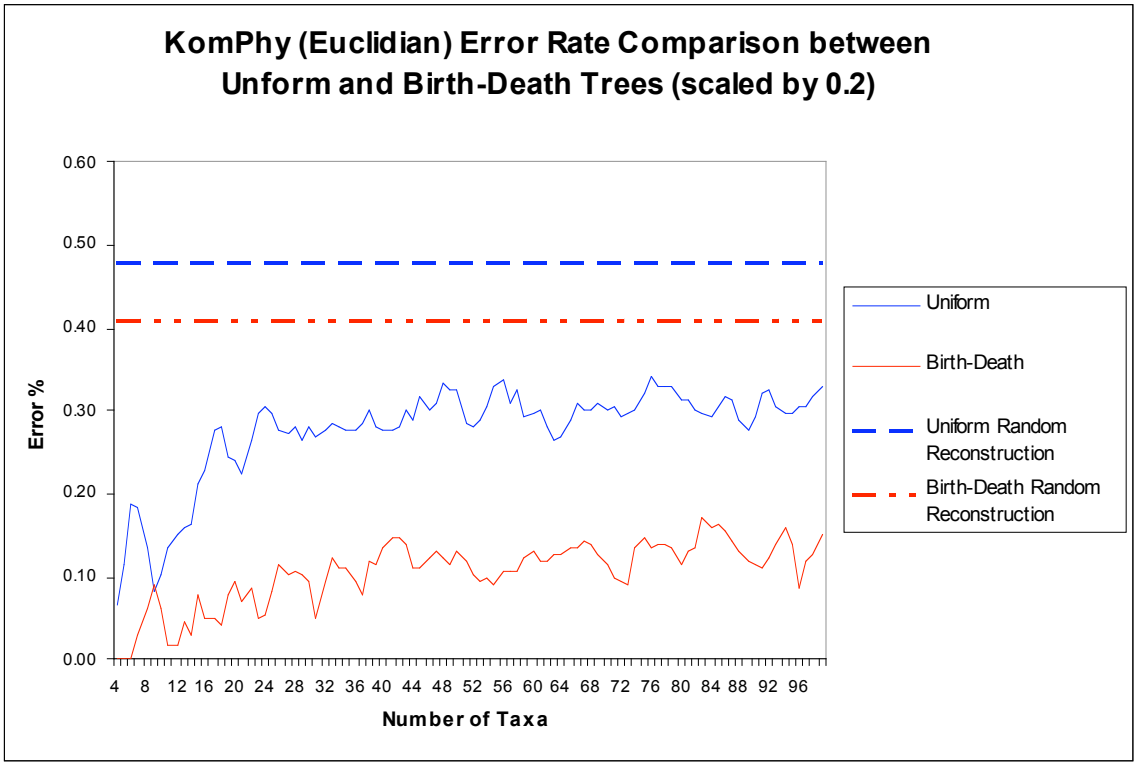


Figure 14: Disparity between accuracy of uniform and birth-death tree reconstruction. Error % is the running average of three adjacent data points. The random reconstruction error rate was determined by taking pairs of uniform trees and birth-death trees and finding the RF distance between them then averaging the result. Birth-death random reconstruction has a lower RF distance because birth-death trees are more confined in their topology than uniform trees are.

KomPhy is much more accurate when applied to Birth-Death trees than when applied to Uniform trees. The dashed lines indicate the RF error when two random uniform or birth-death trees are compared. This provides some baseline for comparing KomPhy's reconstruction against that which chance alone could have produced. Since Birth-Death trees are more confined and less variable than uniform trees random reconstruction tends to be more successful -- i.e. birth-death trees are easier to reconstruct. Even when taking that into account KomPhy does significantly better on birth-death trees than on uniform trees.

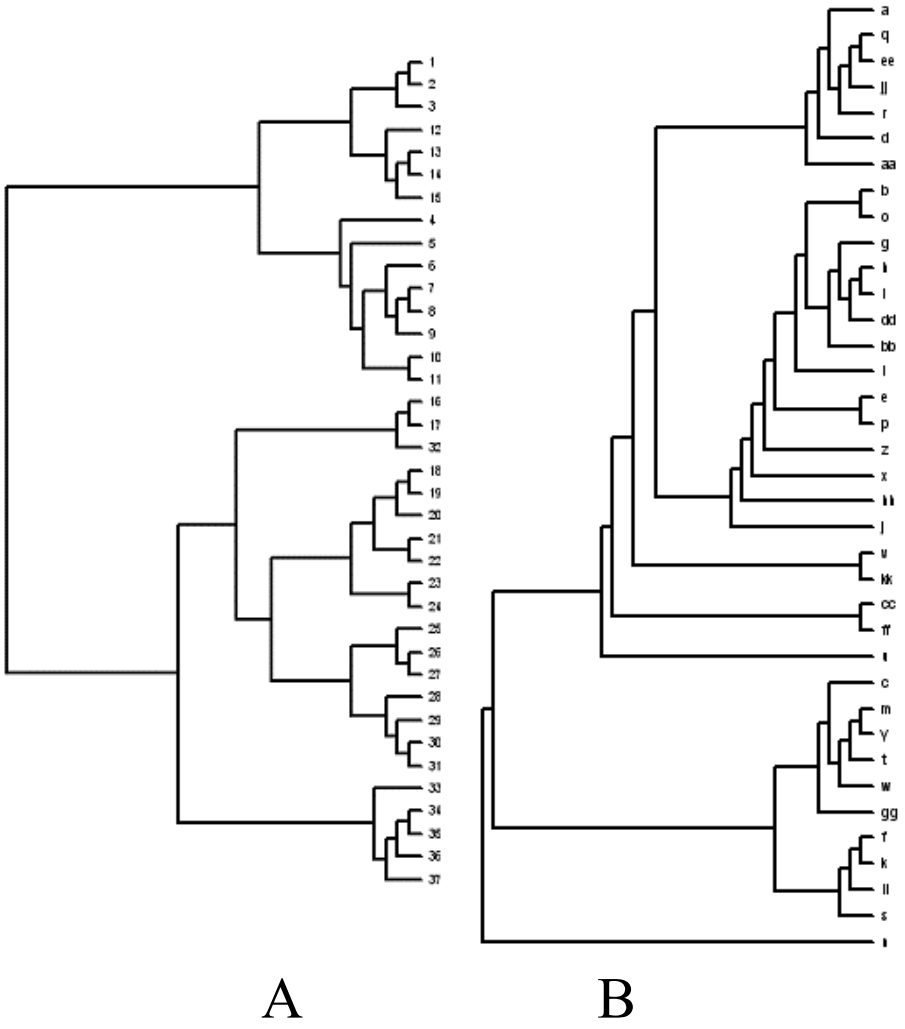


Figure 15: Tree A is a typical birth-death tree, tree B is a typical uniform tree. Tree A was recovered by KomPhy with an RF error of 7%, tree B was recovered with an RF error of 30%.

Comparison with other Reconstruction Algorithms

Two other reconstruction algorithms were chosen for comparison. SOTA is based on Self Organizing Maps which, as was discussed in Section 1.1, is the only other neural network approach to phylogenetic reconstruction. Neighbor-joining was selected for comparison because it is probably the most popular fast reconstruction algorithm in use. Sequence Length Requirements for Phylogenetic Methods,].

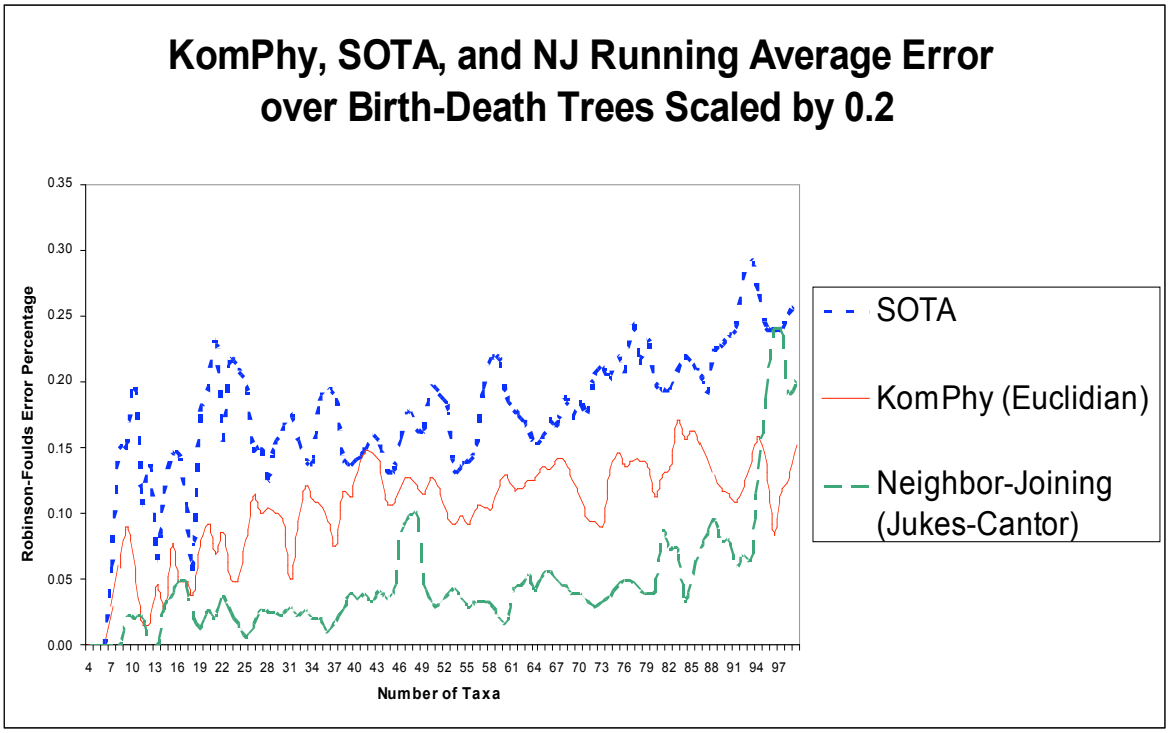


Figure 16: Comparison of birth-death tree reconstruction accuracy for Neighbor-Joining, SOTA, and KomPhy. Error % is the running average of three adjacent data points. For this particular run SOTA's reconstruction of the trees with 13, 14, 46, 81, 95, 96, 97, and 99 taxa did not converge on a topology within 3,000 cycles (SOTA typically converges in less than 200 cycles). Those reconstruction attempts were left out of the data set and the average RF for the nearest three reconstruction attempts were used to fill in the gap.

Neighbor-joining clearly produces the most accurate reconstructions, not only in this case, but also for uniform trees, KomPhy's mean RF distance over these 100 trees is about 5% less than NJ's mean RF distance. SOTA's mean RF distance is the largest with an error rate of around 20%.

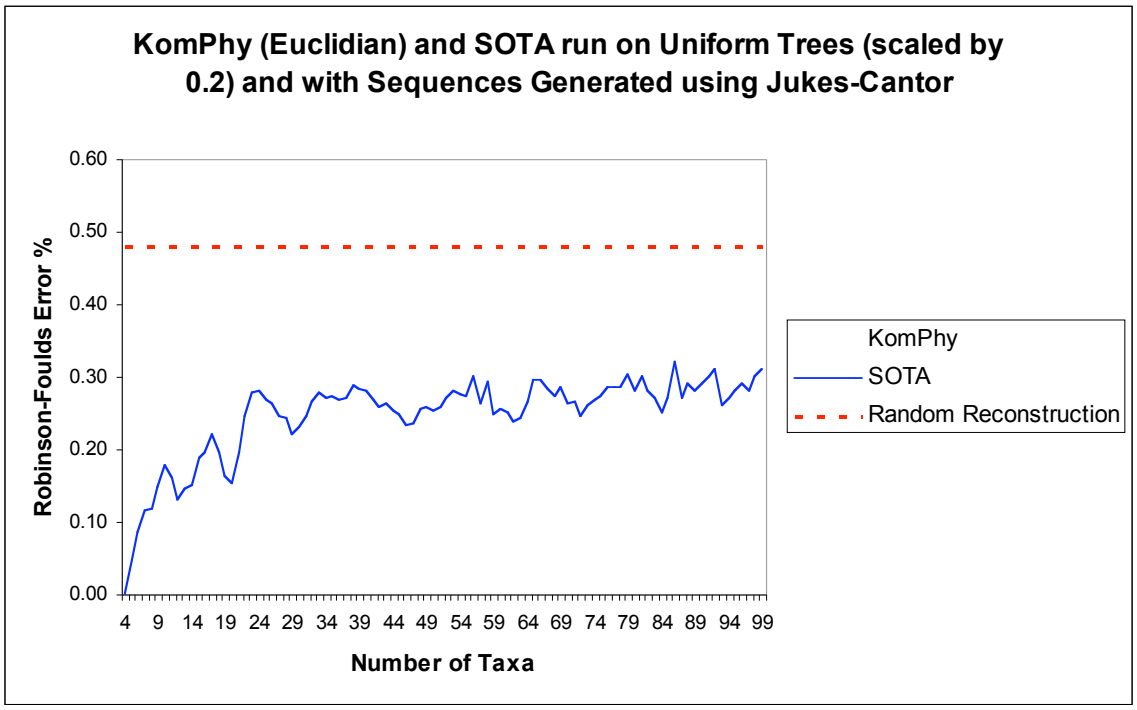


Figure 17: Comparison of uniform tree reconstruction accuracy for SOTA, and KomPhy. Error % is the running average of three adjacent data points.

Neighbor-joining was found to be very accurate on uniform trees: its RF distance never rises above 0.02. Both neural network approaches do very poorly when reconstructing uniform trees, with KomPhy being the less accurate of the two. Clearly symmetric trees match the clustering bias of KomPhy and SOM to a much greater degree than asymmetric trees do. This is not surprising since both approaches effectively reduce the dimensionality of their inputs and so are performing an averaging process.

Branch Lengths and Statistical Consistency

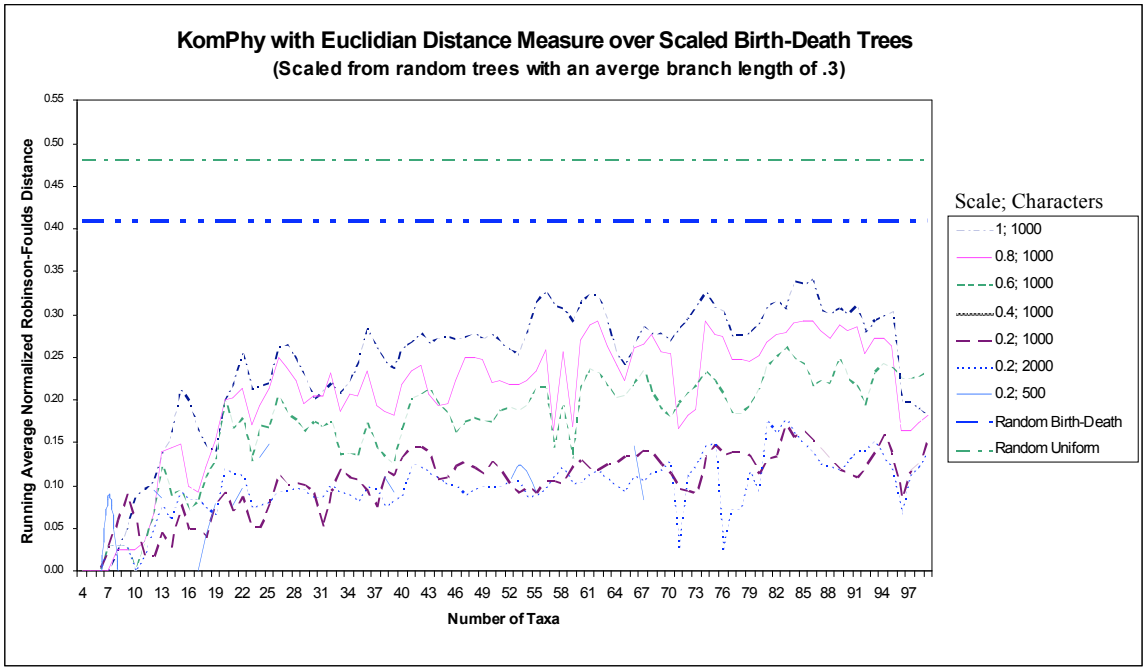


Figure 18: Reconstruction of trees by KomPhy with scaled branch lengths and varying with various sequence lengths. Average Robinson-Foulds error for 0.2 birth-death trees: KomPhy: 11%.

Scaling the branch lengths of Birth-Death trees resulted in marked improvement in KomPhy's accuracy. For trees in which 30% of the characters on average were expected to change per branch, a full third of the edges placed by KomPhy in the reconstructed tree were incorrect. When the expected number of changes per sequence per branch was reduced to 0.06, the number of misclassified edges was halved.

The normalized RF distance, over all 96 trees and using the Euclidean metric, was 0.11. Doubling the number of characters to 2000 resulted in an improved score of 0.10. Surprisingly, halving the number of characters also resulted in an improved score of 0.10. Nine-hundred and ninety-four sets of sequences with lengths from 4 to 1,000 were generated from and given to KomPhy to reconstruct. No Robinson-Foulds value for all these trees was more than 0.03 apart for all sequence lengths. This 0.03 variation could be

explained by variations in the sequences introduced by Seq-gen, since no trend in RF distances were observed as a function of sequence length. Some sequences with as few as 104 characters were observed to have better RF distances than reconstructions over the same tree with 2,000 characters. There is no indication therefore that KomPhy is statistically consistent.

Model Correction

Two distance functions were found to work well in the KomPhy framework: Euclidean distance and the Jukes-Cantor distance. The following charts show the relative error rates for the two distances when applied to reconstructing birth-death and uniform trees.

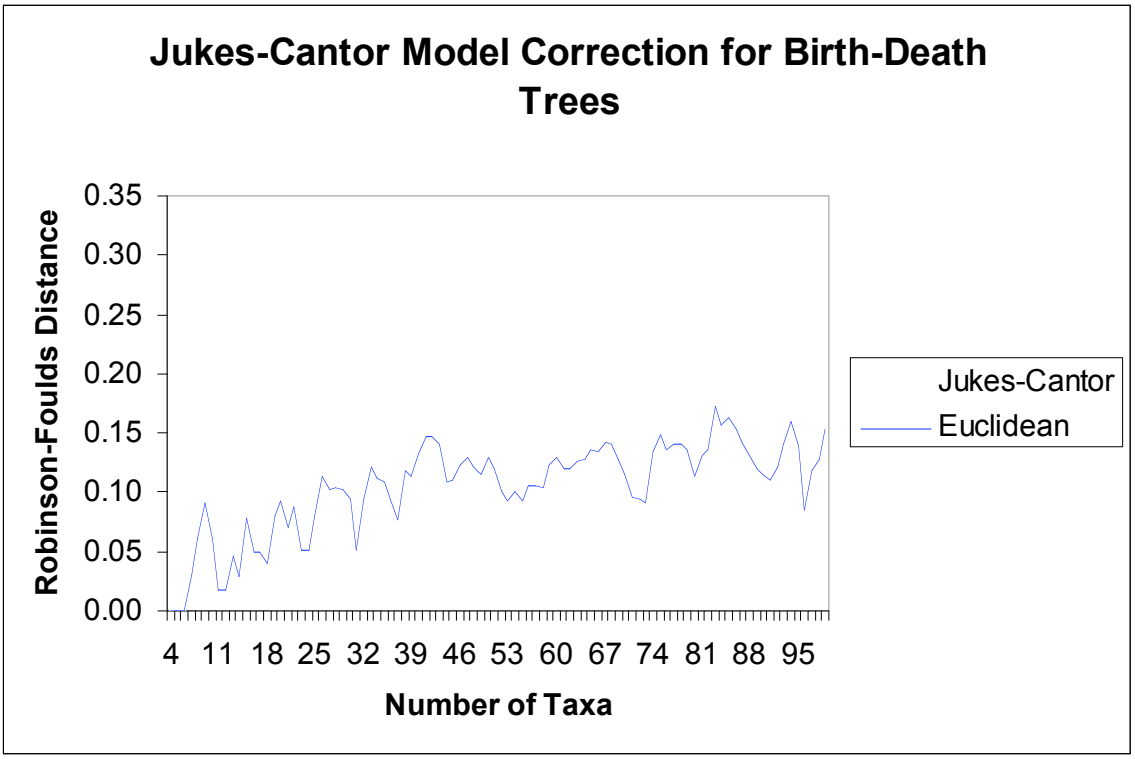


Figure 19: Comparison of the Euclidean and Jukes-Cantor distances when used by KomPhy to reconstruct birth-death trees.

Surprisingly the Euclidean distance outperforms the Jukes-Cantor distance when applied by KomPhy even though all the trees used in these experiments were generated with the Jukes-Cantor model. With uniform trees, on the other hand, using the Jukes-Cantor distance function generates slightly more accurate trees using the Euclidean distance.

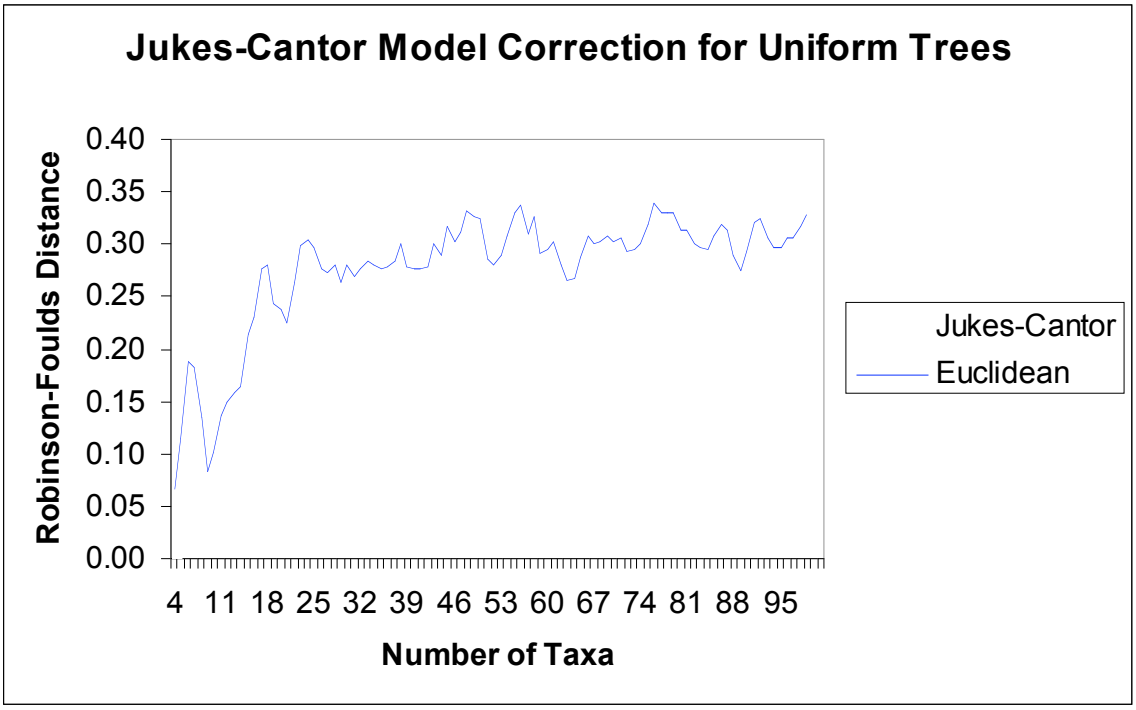


Figure 20: Comparison of the Euclidean and Jukes-Cantor distance when used by KomPhy to reconstruct birth-death trees.

Comparison of Running Times for SOTA, KomPhy and Neighbor-Joining

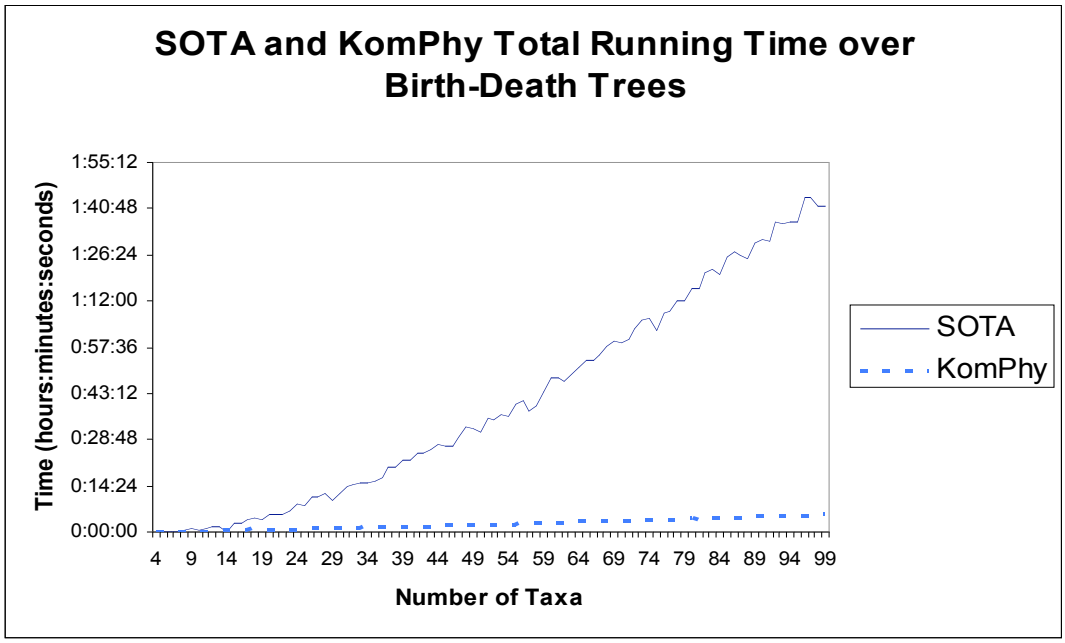


Figure 21: Comparison of SOTA and KomPhy total running times. Neighbor-joining reconstructed trees much more quickly than either KomPhy or SOTA. Neighbor-joining was faster than SOTA and KomPhy for all trees examined.

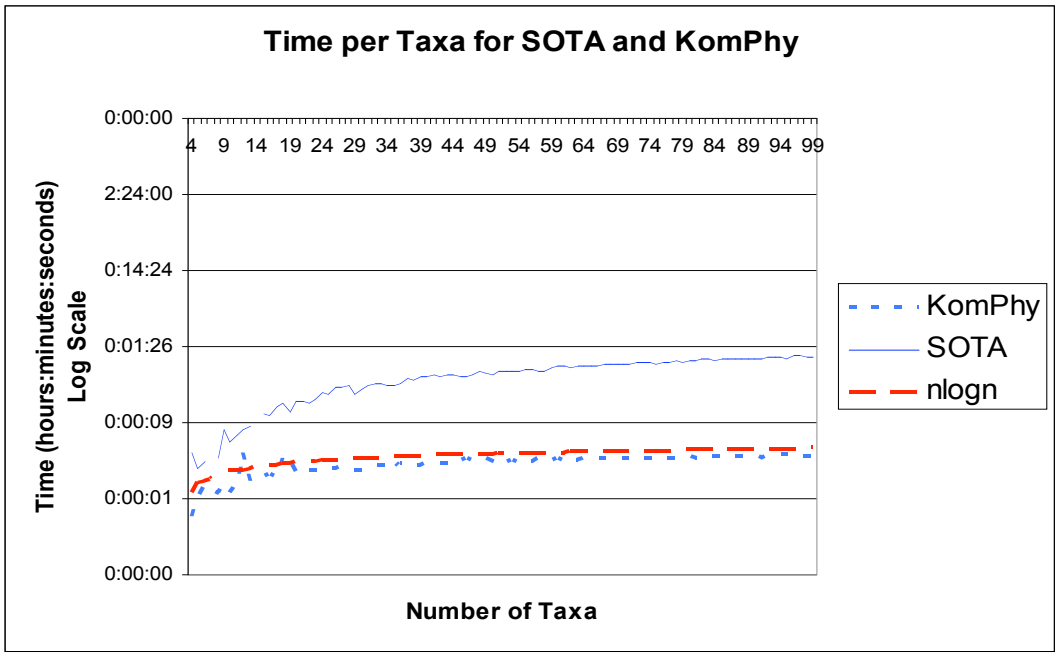


Figure 22: Comparison of per taxa running time between KomPhy and SOTA. A hypothetical *mn*log(*n*) algorithm with a cost for *m* of two seconds is plotted for comparison.

Experimental Results Summary

For birth-death trees KomPhy was found to produce reconstructions with an average error rate of 10% which is about 5% more accurate than SOTA and about 5% less accurate than neighbor-joining. For uniform trees SOTA was more accurate than KomPhy by 2-3%, and neighbor-joining was more accurate by about 30%. Neighbor-joining was found to perform very well on the uniform trees presented to it with error rates never rising above 2 or 3 percent. Both neural network architectures on the other hand produced very poor reconstructions with error rates of between 25 and 35%. KomPhy produced the most accurate reconstructions when branch lengths did not exceed 24 changes per 100 characters with the error rate rising to between 25 and 30% when maximum branch lengths were allowed to exceed 1.0 (Figure 19).

Empirically the difference between the running-time over worst case and best case trees with up to 1024 taxa was found to be only 10%. KomPhy's has an aversion to creating trees with imbalanced topologies that result in a degradation of its running time.

4 Parallel Implementation

4.1 Approach

The presentation step of the KomPhy algorithm is amenable to parallelization since the distance between sequences and weights is independent throughout if the learning step is fully decoupled by using *batch learning*. Batch learning is functionally equivalent to interlaced learning, but allows an entire epoch of presentations to occur

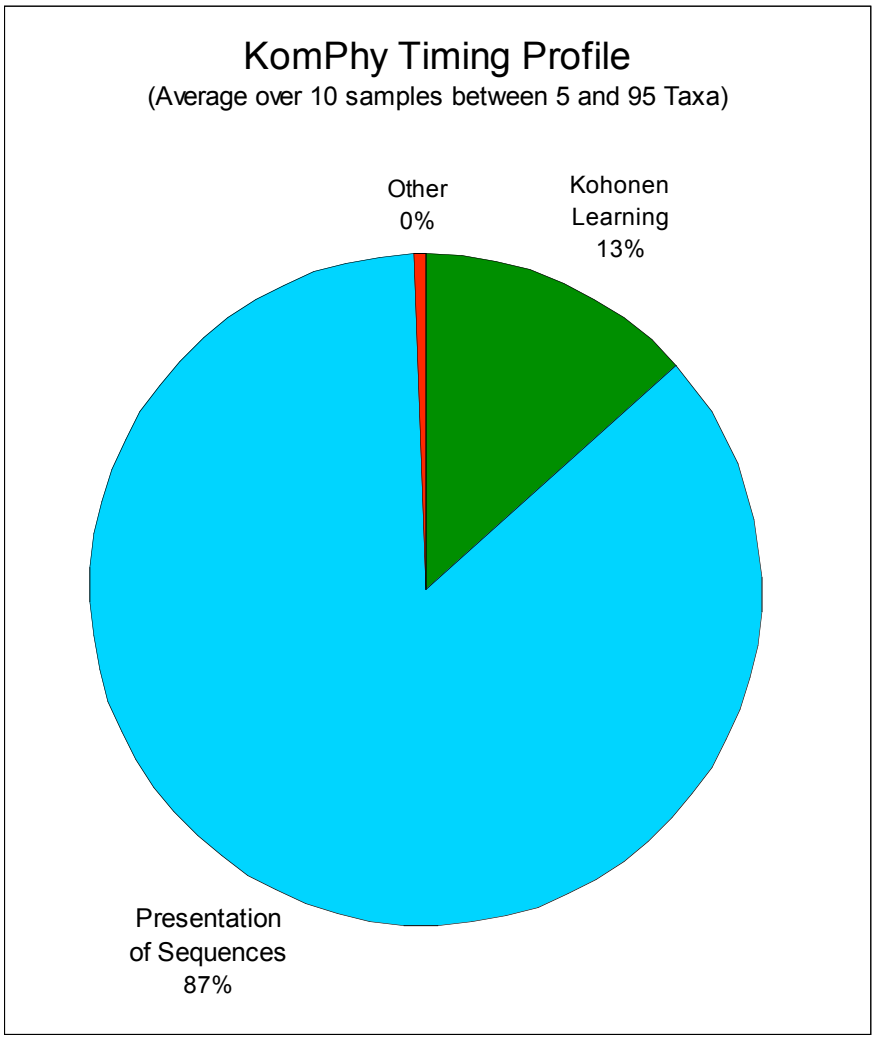


Figure 13: Time spent in calculating distances from weights to sequences (presentation of sequences), in applying the learning function (Kohonen learning), and in all other operations (overhead)

before the need to update the weight values (which must agree across all processors). Parallelizing the learning step was considered but every sequence in the system effects the change in weights. To synchronize the changes would cost more in communication than would be saved in local computation. KomPhy was profiled in order to discover whether parallelizing the presentation step would result in significant gains. Profiling showed that 87% of the time KomPhy was calculating distances between weights and sequences. (This percentage was the average over 96 runs on trees having between 4 and 100 taxa with KomPhy configured to iterate over 500 epochs.) Parallelization of sequence presentation would therefore have a significant impact on overall running time. Presentation time was also determined to grow faster as taxa were added than learning or overhead.

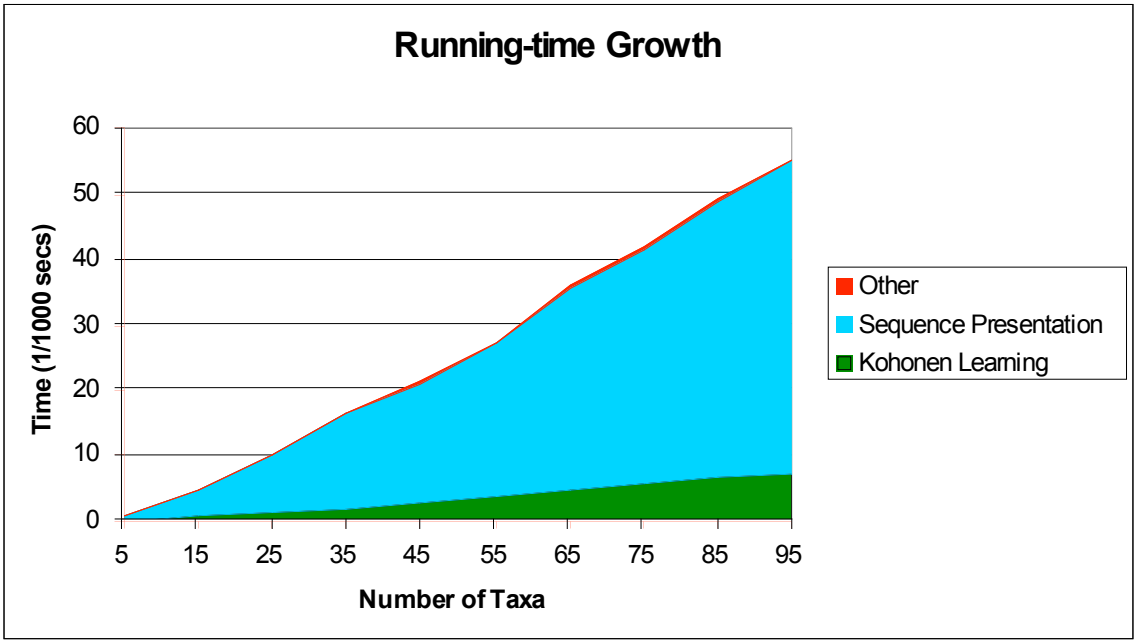


Figure 24: Time cost growth for presentation, learning, and overhead in the serial version as an incentive to parallelize the presentation step.

The parallel implementation of KomPhy makes two changes to the serial version. First, the for loop which cycles through the presentation of sequences to the neural network and calculates the winning neuron in each case is modified so that a sequence is only presented if the sequence number modulus the number of processors is equal to the local processor ID. This effectively divides the work such that each processor does roughly n/p work where n is the number of sequences and p is the number of processors. Once each processor has calculated which neuron is the winner of each sequence, a new function called `synch_partitions` is called. This function packages the association of winning weight to sequence with an integer paring function. This is accomplished with the MPI function mpi_`allgatherv` [33]. Once all processors have the global partitioning for the current epoch the Kohonen learning algorithm is run just as in the serial version. Batch learning is used in order to achieve the separation of sequence presentation and learning required to parallelize the presentation step.

This approach has the benefit that it does not disturb the serial implementation at all. The division of sequences to processors assigns all the work to processor 0 in the serial case and can simply skip the `synch_partitions` step if the program is not running in parallel.

Input: a local set, P, of partitions

a set, N, of processors

assume an integer pairing function **pair**(x, y)

assume an integer pair reversal function **unpair**(z)

Output: a global set, Q, of partitions

begin synch_partitions(P)

i = N_{this}

j = |P|

foreach sequence j in P **do**

send_array$_{ij}$ ← pair(P_j, S_k)

end

mpi_allgatherv(send_array, receive_array)

foreach element *m* of receive_array **do**

<Q_i, S_t> = unpair(receive_array$_m$)

push(Q_i, S_t)

end

return Q;

end

Figure 25: Algorithm for synchronizing local partitions, where the sequences in the local partition are those which have modulus of their index equal to the processor index.

4.2 Results

A significant reduction in running time was observed after parallelization. Running time over a single 97 taxa tree for the presentation step was reduced from 48 seconds on a single processor to 28 seconds on two processors, a 42% reduction in running time. Overall running time was reduced from 60 seconds to 40 seconds, a 30% reduction. Running KomPhy on 3 processors reduced the presentation time to 10 seconds for an 80% reduction over the serial case. Total running time was reduced to 20 seconds or a 66% reduction.

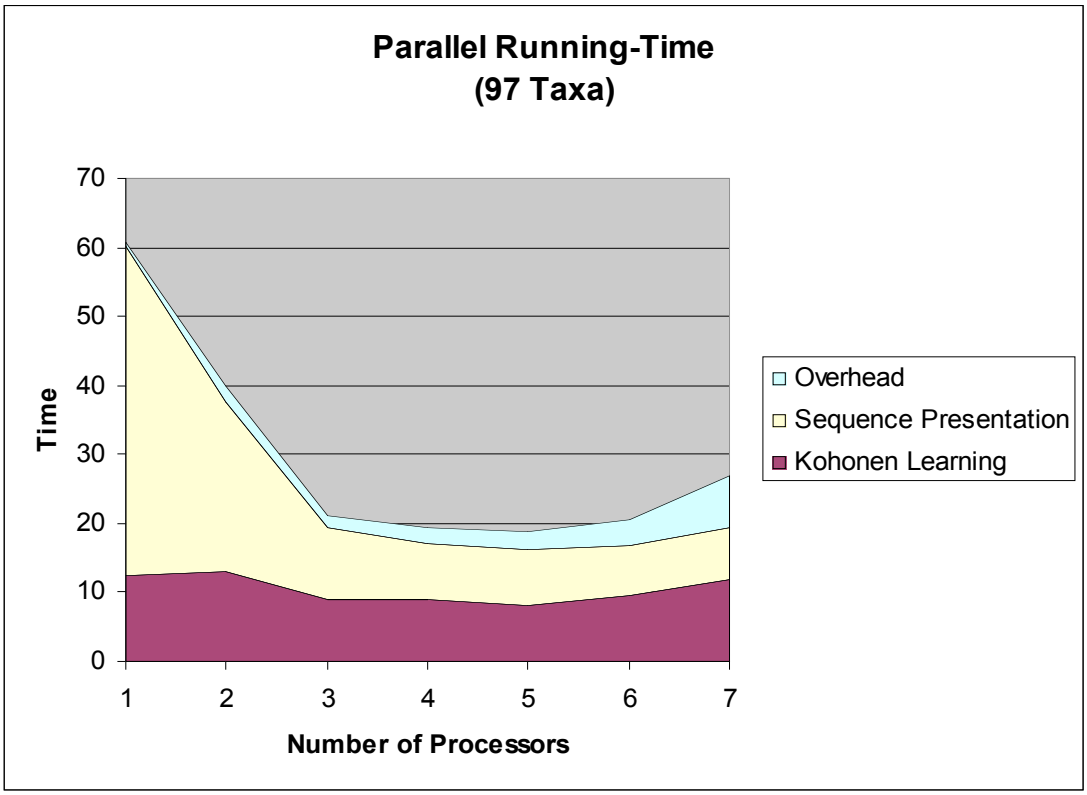


Figure 26: Parallel running-time of presentation step, overall reduction in running time, and eventual increase in communication overhead. The data used to create this chart came from a single run of the parallel implementation.

5 Summary and Conclusions

In this thesis, a new unsupervised neural network approach to phylogenetic reconstruction was described and analyzed. Comparisons were made between the new approach and existing methods including SOTA, which is the closest existing algorithm to KomPhy, and Neighbor-Joining, which is the most commonly used fast reconstruction algorithm. The running time of KomPhy was analyzed and found to be quadratic in the worst case but O(nlogn) in the expected case. SOTA is described as a linear algorithm but experimentation demonstrates KomPhy to be much faster (at least for trees with less than 100 taxa) and in the case of birth-death trees more accurate.

Komphy was not found to be accurate enough to compete with neighbor-joining ($O(n^3)$) when applied to uniform or birth-death trees. KomPhy's failure in this regard may indicate that competitive neural network approaches in general may not be as well suited to phylogenetic reconstruction as direct algorithms are. This is further supported by the relative similarity between the accuracy of SOTA and KomPhy despite their quite different representations of the search space and the differing frameworks within which the networks were embedded.

This is not to say that the continued study of competitive neural approaches may not bear fruit in the future but it does seem to suggest that the averaging effect of mapping nodes onto a solution space may be insufficient to guarantee acceptable reconstructions. The individual paths weights take through weight space are susceptible to local minima and to the relative movements of other neurons. The comparison to neighbor-joining may be instructive since in many ways neighbor-joining is the inverse strategy to KomPhy and SOTA. Whereas competitive neural networks and self

organizing maps are top-down algorithms that iteratively move from the more general solution to more particular ones, neighbor-joining starts by creating an initial partition of just two taxa and then builds the general structure from the successive addition of single taxa. This bottom up approach seems to do better than KomPhy because errors near the root may propagate throughout the all subtrees whereas errors made near the leaves are localized.

6 References

[1] Saitou, N. and M. Nei, "The neighbor-joining method: A new method for reconstructing phylogenetics trees," *Molecular Biological Evolution*, vol. 4, Pages 406-425, 1987

[2] May, Robert, Sean Nee, "Extinction and the Loss of Evolutionary History," *Science,* vol. 278, Pages 692-694, 1997

[3] Kohonen, T, "The self organizing map," Proceedings of the IEEE, vol. 78, Pages. 1464-1480, 1990

[4] Ramón y Cajál, S., "Histologie du Systéms Nerveux de l'himme et des vertebras," Paris: Maloine, 1911

[5] D. F. Robinson and L. R. Foulds, "Comparison of phylogenetic trees," *Mathematical Biosciences*, 53:131–147, 1981

[5a] Robert Eric, Course website, "The Intellectual Excitement of Computer Science," Stanford University, URL: http://www-cse.stanford.edu/classes/sophomore-college/projects-00/neural-networks/

[6] DeSieno D., "Adding a conscience to competitive learning," in: Proceedings. ICNN'88, International Conference on Neural Networks, IEEE Service Center, Piscataway, N J, Pages. 117-124, 1988

[7] J. Buhmann and H. Khnel, "Complexity optimized data clustering by competitive neural networks," Neural Computation., vol. 5, no. 3, Pages 75-88, May 1993

[8] Kohonen, T., *Self-Organization and Associative Memory*. Springer-Verlag, Berlin, 1989

[9] Dopazo J. and J. M. Carazo "Phylogenetic reconstruction using an unsupervised growing neural network that adopts the topology of a phylogenetic tree," *Journal of Molecular Evolution,* 44, 226-233, 1997

[10] Wu C, Shivakumar S "Back-propagation and counter-propagation neural networks for phylogenetic classification of ribosomal RNA sequences," *Nucleic Acids Research*. 22:4291-4299, 1994

[11] Ferran EA, Pflugfelder B, Ferrara P "Self-organized neural maps of human protein sequences," *Protein Science*. 3:507-521, 1994

[12] Andrade, M.A., G. Casari, C. Sander and A. Valencia. "Classification of protein families and detection of the determinant residues with an improved self-organising map," *Biological Cybernetics,* 76, 441-450, 1997

[13] Andrade, A. Valencia, E.L. Zapata and J.M. Carazo, "Computational space reduction and parallelization of a new clustering approach for large groups of sequences," *Bioinformatics*, 14, 439-451, 1998

[14] P. Tamayo, D. Slonim, J. Mesirov, Q. Zhu, S. Kitareewan, E. Dmitrovsky, E. Lander, T. Golub, "Interpreting patterns of gene expression with self-organizing maps:

methods and application to hematopoietic differentiation," Proceedings of the National. Academy of Science for 1999. Page 96, 1999

[15] Hartigan, J.A. Clustering Algorithms, John Wiley and Sons, New York, 1975

[16] S. Tavazoie, D. Hughes, M.J. Campbell, R.J. Cho, G.M. Church, "Systematic determination of genetic network architecture" *Nature Genet*. vol. 22, Pages 281-285, 1999

[17] Haykin S., *Neural Networks: a comprehensive foundation*, 2nd Ed., Prentice Hall, Upper Saddle, River, 1999

[19] Hoare, C. A. R. "Partition: Algorithm 63," "Quicksort: Algorithm 64," and "Find: Algorithm 65," *Communications of the ACM*, 4, 321-322, 1961

[20] Kohonen, T., "The Self-Organizing Map", Proceedings of the IEEE, Vol.8, No.9, Pages 1464-1480, September, 1990

[21] Jukes, T . K. and C. R. Cantor, Evolution of protein molecules, In *Mammalian protein metabolism* (H. N. Munroe, ed.), Academic, New York, 1969

[22] Felsenstein J., and G. A. Churchell, "A hidden markov model approach to variation among sites in rate evolution," *Molecular Biological Evolution*, vol. 13, Pages 93-104, 1996

[23] Tamura, K. and M. Nei., "Estimation of the number of nucleotide substitutions in the control region of mitochondrial DNA in humans and chimpanzees," *Molecular Biological Evolution*, vol. 10, Pages 512-526, 1993

[24] Salter, L, "Complexity of the Likelihood Surface for a Large DNA Data Set," *Systematic Biology*, 50(6): 970-978, 2001

[25] Moret, Bernard M. and Henry D. Shapiro, *Algorithms from P to NP: Design and Efficiency*, Benjamin/Cummings, Redwood City, 1991

[26] Corman, Thomas H, Charles E. Leiserson, and Ronald L. Rivest, *Introduction to Algorithms*, MIT Press, Cambridge, Pages 165-166, 1990

[27] Rambaut, A., "Phylogenetic Tree Simulator Package," Oxford, 2002

[28] Roderic D.M. Page, "Component," The Natural History Museum, London, 1993

[29] Rambaut, A. and Grassly, N. C. "Seq-Gen: An application for the Monte Carlo simulation of DNA sequence evolution along phylogenetic trees," *Compuational. Applied. Bioscience,* vol. 13, Pages 235-238, 1997

[30] Felsenstein, J., "PHYLIP - Phylogeny Inference Package (Version 3.2)," *Cladistics,* vol. 5: Pages 164-166, 1989

[31]Mooers,A.O. and Heard,S.B, "Inferring evolutionary process from phylogenetic tree shape," *Quarterly Review of Biology*, Vol. 72, Pages 31–54, 1997

[32] Robinson D. F.and L. R. Foulds, "The steiner problem in phylogeny is np-complete," *Proc. Natl. Academy Science*, 3:43 – 49, 1982

[33] Snir, Marc., *MPI: The Complete Reference*, MIT Press, Boston, 1996

[34] Rumelhart, D. E. and Zipser, "Feature discovery by competitive learning," *Cognitive Science*, vol. 9, Pages 79-112, 1985