Lecture 8: December 18, 2000

Lecturer: Ron Shamir *Scribe: Yuval Inbar and Tzvika Hartman*

Based on a scribe of Gidon Shavit and Chaim Linhart, January 31, 1999.

8.1 Preface: Phylogenetics and Phylogenetic Trees

8.1.1 What is Phylogenetics?

Phylogenetics is the area of research concerned with finding the genetic connections and relationships between species. The basic idea is to compare specific *characters* (features) of the species, under the natural assumption that similar species (i.e., species with similar characters) are genetically close. The term *phylogeny* refers to these relationships, usually presented as a *phylogenetic tree*[1] (see below).

Classic phylogenetics dealt mainly with physical, or *morphological* features – size, color, number of legs, etc. Modern phylogeny uses information extracted from genetic material – mainly DNA and protein sequences. The characters used are usually the DNA or protein *sites* (a site means a single position in the sequence) after aligning several such sequences, and using only blocks which were conserved in all the examined species.

An interesting example is a research project that used phylogenetics in order to trace the origins of the human population on earth. Researchers investigated the mitochondrial DNA of 182 people all over earth (the mitochondrial DNA is especially good for phylogenetic research since it is copied completely from mother to son, without recombining with the father's DNA). The phylogenetic analysis provided evidence that all humans have a common female ancestor who lived in Africa ("African Eve").

When studying phylogeny using nuclear genes, we encounter a serious problem. During evolution, it is very common for a gene to be duplicated. The copies continue to evolve separately, resulting in two (or more) similar instances of the same gene along the genome of a species. Therefore, when discussing matching genes in different species, we differentiate between *orthologous* matches – which means both genes are "the same" gene in the strong sense – they are connected directly, and not through a duplication, *paralogous* matches – which are the result of some duplication along the evolutionary line, and *xenologs* (horizontal transfers) which are genes that are transfered between organisms in other ways (e.g., by virus). Therefore, if we base our analysis on paralogs or xenologs (rather than orthologs)

[1] The terms *phylogenetic tree* and *phylogeny* will be used synonymously throughout this lecture

we are in big trouble. However, throughout this lecture we will assume there were no gene duplications.

Note: In this lecture we shall refer to the objects whose phylogeny is in question as *species*. However, the discussion is valid not only to the phylogeny of different species, but also to other objects, e.g., duplicated genes of the same species. We shall also often refer to characters as sites, because this is the most common case.

8.1.2 Phylogenetic Trees

The most convenient way of presenting phylogenetic information is using a *phylogenetic tree*. In a phylogenetic tree, every node represents a species. Nodes are labeled, either with species names or the values (also referred to as *states*) of their characters, and the edges represent the genetic connections. It is important to note that there is usually a big difference between the leaf nodes, that represent real species, and the internal nodes, that in most cases represent the hypothetical evolutionary ancestors of the species in the data.

Phylogenetic trees take several forms: They can be *rooted* or *unrooted*, binary or general, and may show, or not show, edge lengths. A *rooted* tree is a tree in which one of the nodes is stipulated to be the root, and thus the direction of ancestral relationships is determined. An *unrooted* tree, as could be imagined, has no pre-determined root and therefore induces no hierarchy. Therefore, in this case, the distance between the nodes should be symmetric (since the tree edges are not directed). Rooting an unrooted tree involves inserting a new node, which will function as the root node. This can be done by introducing an *outgroup*, a species that is definitely distant from all the species of interest. The proposed root will be the direct predecessor of the outgroup. Figures 8.1 and 8.2 show a rooted tree and its unrooted counterpart, respectively.

A binary, or *bifurcating*, tree is of course a tree in which a node may have only 0 to 2 subnodes, that is, in an unrooted tree, up to three neighbors. It is sometimes useful to allow more than 2 subnodes (*multifurcation*), but the discussion in this lecture will be limited to binary trees.

A tree can show edge lengths, indicating the genetic distance between the connected nodes. We sometimes assume the existence of a *molecular clock*, a constant pace of the evolutionary processes. If this is the case, we could theoretically produce a phylogenetic distance-preserving tree which can be presented along a time-axis – assigning to each node the time in which it “occurred” in the history of evolution. In such a “perfect” tree, the length of each edge would be the difference in time between the parent node and the child node.

There are two types of data used for building phylogenetic trees:

1. *Distance-based:* The input is a matrix of distances between the species (e.g., the alignment score between them or the fraction of residues they agree on).

2. *Character-based:* Examine each character (e.g., a base in a specific position in the DNA) separately.

8.2 Character Based Methods

We will discuss the following problem:

Problem 8.1 Optimal Phylogenetic Tree.
INPUT:

- A set of n species
- A set of m characters pertaining to all of these species
- For each species, the values of each of the characters

QUESTION: What is the fully-labeled phylogenetic tree that best explains the data, i.e., maximizes some target function.

The process of solving this problem is called *inferring the phylogeny.* The input is usually given as an $n \times m$ matrix M, where $M_{i,j}$ represents the value of the jth character of the ith species. The state (value), of each character is taken from a known set A_j.

The input may also include other relevant parameters – e.g., the distribution of changes (mutations) in each character, weights representing relative importance of characters, etc. The goal will be to maximize some *score* over the possible phylogenetic trees and produce the best one.

We will make the following assumptions in attempting to infer phylogenies:

- Characters are mutually independent – that is, change in one character has no effect on the distribution of another character.
- After two species diverge in the tree, they continue to evolve independently.

None of these assumptions is necessarily (or even probably) correct, but they make our life much easier, simplifying the discussion considerably.

A Simple Solution?

The trivial solution to the phylogeny problem would be to enumerate over all possible trees and calculate the target function for each one. However, the number of non-isomorphic, labeled, binary, rooted trees containing n leaves, can be shown to be:

$$(2n-3)!! = \prod_{i=2}^{n}(2i-3) \tag{8.1}$$

(or $(2n-5)!!$ for unrooted trees) which is of course super-exponential – for $n = 20$, for instance, there are about 10^{21} such trees. This means that exhaustive enumeration is unfeasible even for a relatively small number of species.

The next sections will present several approaches towards defining a target function, and attempting to solve the problem for that target function.

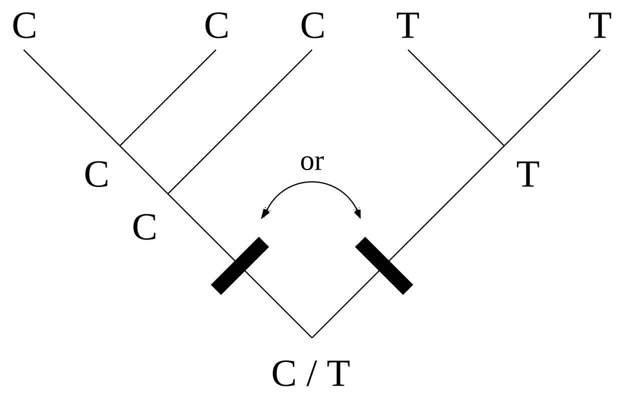

Figure 8.1: A most parsimonious 5-species phylogeny for a single DNA site. The bars mark the two possible edges along which a mutation might have occurred.

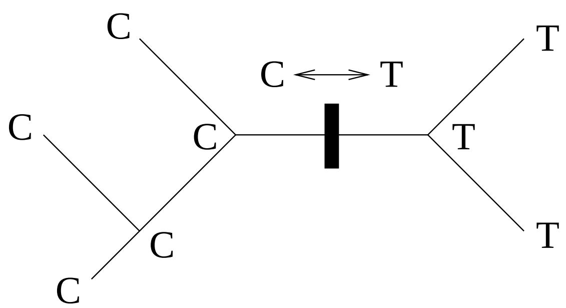

Figure 8.2: The unrooted counterpart of the phylogeny in figure 8.1. Notice that there is now no ambiguity about the placement of the mutation.

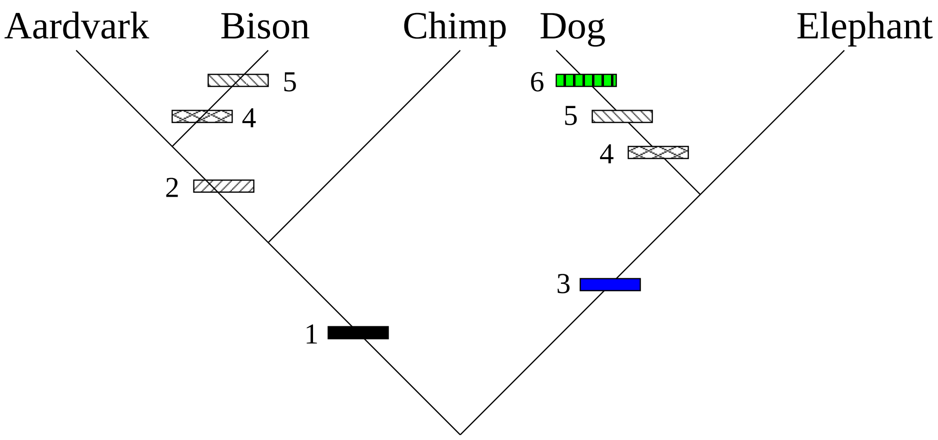

Figure 8.3: A most parsimonious 5-species phylogeny for 6 characters, reconstructed from the data in table 8.1. The numbers by the mutation bars indicate the changed character.

8.2.1 Parsimony

One intuitive score for a phylogenetic tree is the number of *changes along edges*. The approach of minimizing this score is called *parsimony*. The logic is the basic philosophy of Okham's razor – finding the simplest explanation that works. Let us mark the vertices of a tree by $V(T)$, and its edges by $E(T)$. Denote the value of the jth character of vertex $v \in V(T)$ by v_j.

Definition Given a phylogenetic tree T, its *parsimony scre* can be defined as:

$$S(T) \equiv \sum_{(v,u)\in E(T)} |\{j : v_j \neq u_j\}| \tag{8.2}$$

That is – the total number of times the value of some character changes along some edge.

Example Figure 8.1 shows a most parsimonious phylogeny for 5 species with a single character. The parsimony score of this tree is 1 – with the change being either from T to C or vice-versa. Note that this tree can be unrooted, yielding the tree in figure 8.2. The unrooted tree has the same parsimony score as the rooted one. In fact, no matter how we choose to root it, the score will remain the same. Figure 8.3 illustrates a more complex parsimony tree, in which the species have 6 characters each.

There are two levels of problems in parsimony, *small parsimony* and *large parsimony*.

Small Parsimony

Problem 8.2 Small Parsimony.
INPUT: The topology of a rooted phylogenetic tree with labeled leaves.

	1	2	3	4	5	6
Aardvark	C	A	G	G	T	A
Bison	C	A	G	A	C	A
Chimp	C	G	G	G	T	A
Dog	T	G	C	A	C	T
Elephant	T	G	C	G	T	A

Table 8.1: 6 DNA site values for 5 species. This data was used to infer the phylogeny in figure 8.3.

QUESTION:

1. What is the minimum number of changes for this topology?

2. What is the optimal labeling of the internal nodes?

This problem is relatively easy to solve. First of all, it is clear that we can solve it for each character separately, characters being mutually independent. For a single character, we will present the following algorithm:

Fitch's algorithm [6]:

Input: A phylogenetic tree T, with n nodes, and a single character c with a set A of k possible values. Denote the value of the character for node v by v_c.

Step 1: We will assign to each node v a set $S_v \subseteq A$, in the following fashion:

$$\text{For each leaf } v: \qquad S_v = \{v_c\}.$$

$$\text{For any internal node } v, \text{ with children } u, w: \qquad S_v = \begin{cases} S_u \cap S_w & S_u \cap S_w \neq \emptyset \\ S_u \cup S_w & \text{otherwise} \end{cases}$$

To compute S_v we will of course have to traverse the tree in *postorder* – starting with the leaves and working our way down to the root (this is actually a dynamic programming algorithm).

Step 2: Given the sets S_v, we will now determine the value v_c to assign to the character c in each internal node v. This time, we traverse the tree in *preorder*, i.e., from the root up. For each internal node v, if its parent u satisfies $u_c \in S_v$, set $v_c \leftarrow u_c$; Otherwise, (including for the root node), arbitrarily assign any $t \in S_v$ to v_c.

The result of this algorithm is a fully-labeled tree. The number of changes in this tree is equal to the number of times $S_u \cap S_w$ was empty, in step 1.

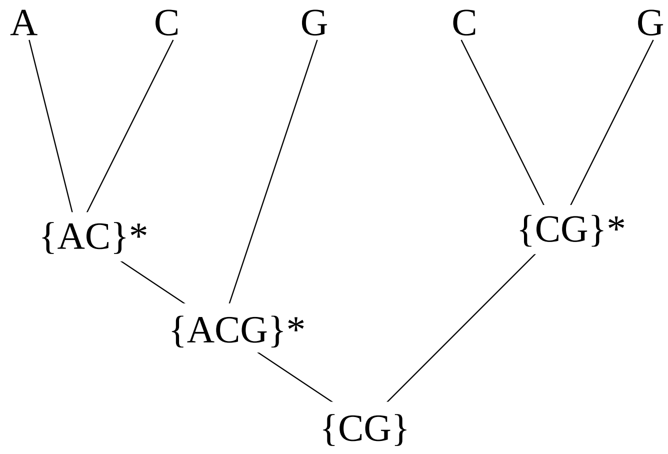

Figure 8.4: An example of step 1 of Fitch's algorithm for a 5-species phylogeny. Nodes marked by an asterisk (*) require a change along one of the edges to their children, adding 1 to the parsimony score.

Complexity: For each node v we work $O(k)$ time to compute S_v, and again $O(k)$ to compute v_c. Total – $O(n \cdot k)$ time (step 2 can be performed in only $O(n)$ total time in the average case).

The above algorithm works with a single character. To obtain the optimal score and labeling for the entire data, simply apply the algorithm once for each character. This leads to an overall complexity of $O(m \cdot n \cdot k)$.

Example In figure 8.4 we have the result of performing step 1 of Fitch's algorithm on a 5-species phylogeny showing a single character. The asterisks mark the nodes where $S_u \cap S_w$ was empty, which means that the minimum total cost of the tree is 3.

It is not very clear at first sight why this algorithm works. We will next present a generalization of the Fitch algorithm, that is perhaps easier to understand.

Weighted Parsimony

In this version of the problem the price of a change is not constant. Instead, denote by C^c_{ij} the cost of the character c changing from state i to state j. The problem is still to minimize the total cost of the tree given the topology and the leaf labels.

Problem 8.3 Weighted Small Parsimony.
INPUT:

- The topology of a rooted phylogenetic tree with leaves having labels in A_c.
- The costs C^c_{ij} for $i, j \in A_c$.

QUESTION:

1. What is the minimum possible cost for this topology?
2. What is the optimal labeling of the internal nodes?

We will present an algorithm by Sankoff [15] which is a generalization[2] of the Fitch algorithm.
Sankoff's algorithm:

Step 1: We will compute, for each node v and each state t a quantity $S_t(v)$ which is the minimum cost of the subtree whose root is v given $v_c = t$. The order of computation will be, as in step 1 of Fitch, postorder: For each leaf v:

$$S^c_t(v) = \begin{cases} 0 & v_c = t \\ \infty & \text{otherwise} \end{cases} \tag{8.3}$$

For an internal node v, with subnodes u and w, it is easy to see that:

$$S^c_t(v) = \min_i \{C^c_{ti} + S^c_i(u)\} + \min_j \left\{C^c_{tj} + S^c_j(w)\right\} \tag{8.4}$$

The minimum total cost of a tree with root r is:

$$S(T) = \sum_{c=1}^{m} \min_t S^c_t(r) \tag{8.5}$$

Step 2: Based on the numbers $S^c_t(v)$ calculated in step 1, we will now determine the optimal values for each character c in the internal nodes. We will traverse the tree in preorder this time:
For the root node r, we will choose $r_c = \arg\min_t S^c_t(r)$.
For any other node v, with parent node u, set:

$$v_c = \arg\min_t (C^c_{u_c t} + S^c_t(v))$$

Complexity: For every node we do $O(k)$ work in each step, meaning $O(n \cdot k)$ per character. The algorithm should be applied once for each character, with a total complexity of $O(m \cdot n \cdot k)$.

[2]Fitch's algorithm is really no more than a discrete version of this one – using costs of 1 for a change and 0 for no change.

Weighted Characters

It is possible to assign weights not only to state changes, but also to the characters themselves. Technically, this means assigning a number W_c to each character, and rewriting equation 8.5 to read:

$$S(T) = \sum_{c=1}^{m} W_c \cdot \min_i S_i(r) \tag{8.6}$$

Where do we get the weights W_c? For instance, if we are working with a DNA sequence, and we know the reading frame, we can make use of the fact that changes in the third codon position are more frequent, since in many cases they don't change the amino acid coded.

In section 8.2.2 we will see another possible source for weights – compatible characters. In short, we will give more weight to characters which seem to fit the tree well than to characters which fit it poorly.

Large Parsimony

Even after having solved the problem of small parsimony, we still have a long way to go, because the final goal is to find the optimal phylogeny, not just the optimal internal labeling of a given phylogeny:

Problem 8.4 Large Parsimony.
INPUT: A matrix M describing m characters of a set of n species,
QUESTION: What is the optimal phylogeny for these species, i.e., the one minimizing the parsimony score?

Again, this problem has a weighted and a non-weighted (discrete) version, but the difference is not essential. It can be shown that this problem is NP-hard, even in the discrete version. However, we will present several algorithms attempting to solve the problem of large parsimony.

Branch and Bound

The general paradigm of *Branch-and-Bound* (B&B) deals with optimization problems over a search space that can be presented as the leaves of a search tree. It was first used for parsimony by Hendy and Penny [8] in 1982. It works when the search tree is *monotonous* – the score of each node in the search tree is at least as bad as that of any of its ancestors. B&B is guaranteed to find the optimal solution, but its complexity in the worst case is as high as that of exhaustive search.

In the simplest form of the algorithm, the search tree is traversed in some order, and the score of the best leaf found so far is kept as a bound B. Whenever a node is reached whose score is worse than B, the search tree is pruned at that node, i.e., its subtree will not be searched, since it is guaranteed not to contain a leaf with a score better than B.

The algorithm can be improved by using some other method to come up with a relatively good candidate, which will yield a good bound before the search has even begun. It is also possible to heuristically improve the traversal order.

Let us, therefore, present our search-space as a search tree. This is not difficult: in the kth level of the search tree, we will have nodes representing all possible phylogenetic trees with k leaves for the first k species (the order doesn't matter, as long as it is pre-determined). The subnodes of a node in that level will be all the phylogenetic trees created by adding the $(k+1)$-th species to the phylogenetic tree, at each possible place. There are exactly $2k-1$ such places, and therefore this is the number of such subnodes.

This search tree clearly upholds the requirement of monotony, since adding a node to a given phylogenetic tree can never reduce its parsimony score. Therefore, the B&B algorithm can be used to help us prune the search tree. The enumeration and pruning can be done in $O(1)$ time per visited node in the search tree. Although it is better than simple brute-force exhaustive search, it does not lower the worst-case time complexity, and it is difficult to predict its exact effect in the average case. However, in real-life test cases it proved to speed up the search considerably.

Local Search

Another general class of optimization methods is local search. The basic idea is to define a neighborhood relation for the search-space, and given such a relation, traverse points in the search space by moving from one point to its neighbors. One can thus use several well-known heuristic algorithms – such as the greedy algorithm, simulated annealing, etc. – to find a local optimum, which will hopefully be also the global optimum.

There are many ways to define neighborhood among trees. For instance – we can use the notion of quartets -trees with four leaves. We can say that two unrooted [3] trees are neighbors, if they can be presented as quartets joining the same four subtrees, only with a different pairing within the quartet (see figure 8.5). This relation is called Nearest Neighbor Interchange. For trees of 5 species, its neighborhhod graph is known as Peterson's graph.

Another possibility to define neighborhood relation states the neighbors of a tree to be those trees created by detaching any subtree and attaching it in some place other than its original location.

8.2.2 Compatibility

Compatibility is another attempt to define a target function for the phylogeny problem: the number of characters which are *compatible* with the given tree. We will try finding the phylogeny which is compatible with as many characters as possible.

[3] In this section we will deal only with unrooted trees

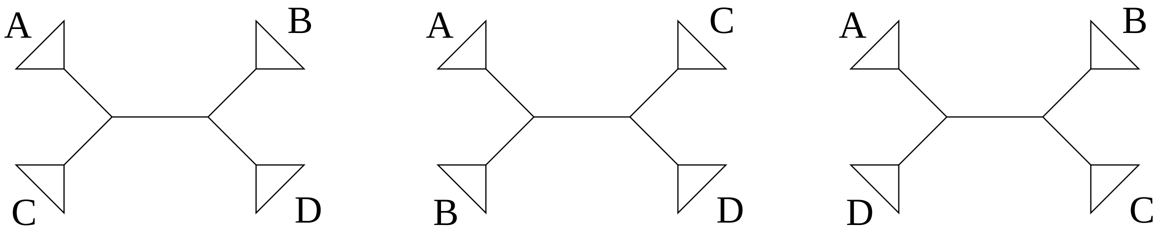

Figure 8.5: The left tree is an unrooted tree with 4 subtrees and the two other trees are its neighbors. The subtrees A, B, C, and D are common to all three trees.

Definition A character c with k possible states is said to be compatible with a tree T with labeled leaves, iff there is a labeling of the internal nodes such that the total number of changes of c is exactly $k-1$.

Note that when we say that a character c has k states, that means that there are k different values of c *in the input*. For instance, if for a given DNA site all the species in the input have either 'T' or 'C', this site has only 2 states, and not 4. It is clear then, that a k-state character can have no less than $k-1$ changes.

In the following discussion, we will assume that all characters are *binary*, i.e., have exactly two possible states, 0 and 1.

Example The binary character presented in the tree in figure 8.1 is compatible with that tree.

Compatibility and Parsimony

Compatibility, in the binary case at least, is easily shown to be a special case of parsimony. To do that, we will use a slightly different, *thresholded* version of parsimony, where for each character that changes at least twice we "charge" exactly 2 (instead of the real total number of changes). This problem is not more difficult than usual parsimony.

To prove this equivalence between thresholded parsimony and compatibility, let us now define n_i to be the number of characters that need i changes. The total score in that case is clearly $S(T) = n_1 + 2(n - n_0 - n_1) = 2n - 2n_0 - n_1$. The numbers n and n_0 are fixed for the given data, and so minimizing the score S means maximizing n_1, which is exactly the number of compatible characters.

Hence we can use the methods described for solving large parsimony, to solve the problem of compatibility. However, this is not of much help, since parsimony is an NP-hard problem.

Pairwise Compatibility

The first step in working with compatibility, is parallel to the small parsimony problem (see 8.2.1): Given a tree T with labeled leaves, find the best compatibility score that can be achieved for that tree, i.e., the maximum number, over all possible labelings of internal nodes, of characters compatible with the fully-labeled tree. This can be done easily using Fitch's algorithm (see 8.2.1).

The more interesting problem here is of course that of "large compatibility" – finding the best phylogeny given only the data matrix M. We shall tackle this problem through the notions of *pairwise compatibility* and *joint compatibility.*

Definition Two characters c_1 and c_2 are said to be *pairwise compatible* (written $PC(c_1, c_2)$), if there exists a tree T such that both c_1 and c_2 are compatible with T.

Definition Characters $c_1, \ldots, c_k$ are said to be *jointly compatible* if there exists a tree T such that $\forall i : c_i$ is compatible with T.

We will present two theorems. The first, by Wilson [16], identifies pairwise compatible characters:

Theorem 8.1 *Pairwise Compatibility Test:*
For characters i, j define the set S_{ij} to be:

$$\{(x, y) : \exists species\ k\ such\ that\ M_{ki} = x\ and\ M_{kj} = y\}$$

where M is the input matrix described in problem 8.1; then $PC(c, c')$ iff $S_{cc'} \neq \{0, 1\}^2$.

Proof Assume $S_{cc'} \neq \{0, 1\}^2$. Then the the set $S_{cc'}$ has at most 3 members. First of all, if $S_{cc'}$ has only a single member, then c and c' each have a single possible state, which is impossible – since they are both binary characters. If $S_{cc'}$ has only 2 members, then we can in fact treat the two characters as a single binary character. Let's assume then that $\{0, 1\}^2 \setminus S_{cc'} = \{(x, y)\}$. Figure 8.6 illustrates the basic structure of a tree that is compatible with two characters, having 3 combined values – $(\sim x, \sim y)$, $(\sim x, y)$, and $(x, \sim y)$. Each triangle represents a subtree in which the values of both characters remain constant. The only mutations are along the two edges marked with bars, proving this part of the theorem. The other direction is simple, and is left as an exercise to the reader. ■

The next, somewhat surprising, theorem by Estabrook et al. [7] identifies jointly compatible sets of binary characters:

Theorem 8.2 *Pairwise Compatibility Theorem:*
All binary characters in a set S are jointly compatible iff $\forall c, c' \in S, PC(c, c')$.

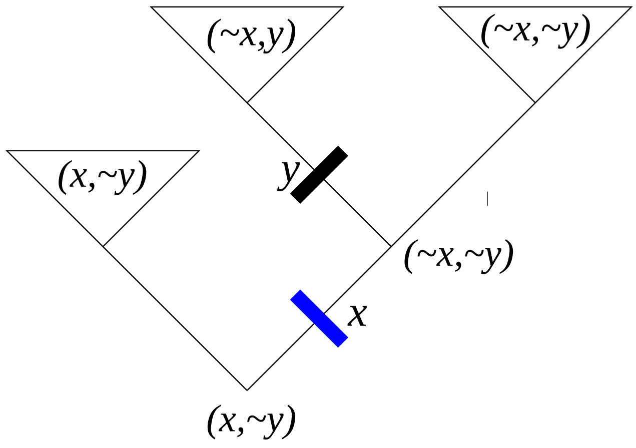

Figure 8.6: A schematic description of a tree that is compatible with two characters, having 3 combined values (see proof of theorem 8.1).

Note that the theorem does not hold for characters that are not binary. We will not present a proof for this theorem.

So the problem of "large compatibility" is reduced to the problem of finding the largest jointly compatible set of characters, which amounts to finding a maximum clique in the pairwise-compatibility graph, defined as:

$$G = (V, E);\ V = \{v_1, \ldots, v_m\}\ ;\ E = \{(v_i, v_j) : PC(c_i, c_j)\}$$

This seems to be of no great help, because as we know, finding a maximum clique in a graph is an NP-hard problem. However, there are algorithms, such as Bron and Kerbosch's [1] Branch-and-Bound clique-finding algorithm, which seem to work very well with biological data. All in all, compatibility methods usually run faster than parsimony methods for the same data.

Perfect Phylogeny

The problem of *perfect phylogeny* (also called *full compatibility*) is to decide if *all* the characters are jointly compatible. For binary characters the problem is easy: it is solved by checking if the pairwise-compatibility graph is a complete graph. For non-binary characters the problem is NP-hard (it relates to some properties of chordal graphs).

Finding the Tree

After finding the maximal clique in the compatibility graph, which means finding the largest set S^* of jointly compatible characters, we still have to construct the phylogeny itself. This is performed simply by successively splitting the set of species according to each of the characters in S^*:

Constructing the Phylogeny

In this algorithm we will iteratively construct an unrooted phylogeny, given the matrix M and the character set S^*. At each step we use one of the characters in S^* to extend the tree T. The set S holds the characters not yet used. Each node in T will be either unlabeled, or labeled with a set $L_v \subseteq \{1, \ldots, n\}$, representing species. Define, for each character and labeled node v: $L_v^i(c) \equiv L_v \cap \{k : k_c = i\}$.

- Initialization:

  1. $T \leftarrow$ a tree containing a single node r, labeled with the set $L_r = \{1 \ldots n\}$.
  2. $S \leftarrow S^*$.

- Iteration:

  1. Choose any character $c \in S$.
  2. Find a labeled node $v \in V(T)$ such that $|L_v^i(c)| \neq 0$ for $i = 0, 1$.
  3. Add two new vertices, v^0 and v^1 to T, labeling v^i with L_v^i.
  4. Add two edges connecting each v^i to v.
  5. Remove the label from v.
  6. $S \leftarrow S \setminus \{c\}$
  7. Go back to step 1 while $S \neq \emptyset$.

The resulting tree T is an unrooted phylogeny with labeled leaves. Finding the labeling of the internal nodes is simple and can be done using Fitch's algorithm described in section 8.2.1.

8.3 Distance Based Methods

8.3.1 Pairwise Distances

Given a measure of the distance between each pair of species, a simple approach to the phylogeny problem would be to find a tree that predicts the observed set of distances as closely as possible. This leaves out some of the information in the data matrix M, reducing

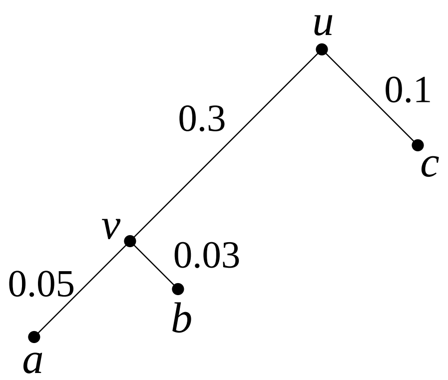

Figure 8.7: A small tree with 3 species – a, b, and c. The branch lengths correspond to the pairwise distances in table 8.2.

	a	b	c
a	0	0.08	0.45
b	0.08	0	0.43
c	0.45	0.43	0

Table 8.2: Distances $d_{i,j}$ predicted by the tree in figure 8.7.

it to a simple table of pairwise distances. However, it seems that in many cases most of the evolutionary information is conveyed in these distances.

For the analysis in this section, we shall first need to define an additive continuous distance function, so that the distance between two species would be expected to be proportional to the total branch lengths between the species. Thus if species a and b are connected via two edges in the tree, with lengths $d_{a,v}$ and $d_{b,v}$ (see figure 8.7), the distance between them would be $d_{a,v} + d_{b,v}$. Furthermore, given the distances between three species – $d_{a,b}$, $d_{a,c}$, and $d_{b,c}$, we could easily calculate the inner distances – $d_{a,v}$, $d_{b,v}$, and $d_{c,v}$, by solving a system of linear equations. Figure 8.7 illustrates a small tree, and table 8.2 contains the distances it predicts.

We will give some examples of how distances may be computed to make them comply with our requirements – one for proteins, and another for DNA sequences.

8.3.2 Distance between Proteins – PAM matrices

We have already defined the PAM matrices, when we discussed heuristics for sequence alignment (in lecture #3). The PAM_n matrix is designed to compare two amino-acid sequences which are n PAM units apart. Its calculation involves raising M, the mutation probabilities matrix for one PAM unit, to the power of n. For a continuous distance function, we need to define PAM matrices for non- integer units, as well.

Let $M = U^{-1}\lambda U$ be the diagonalization of M, where λ is a diagonal matrix, Whose entries are M's eigen-values, and U is an orthonormal matrix, which consists of the corresponding eigen-vectors. Given a real x, the PAM_x distance matrix is simply:

$$PAM_x(i,j) = \log \frac{M^x(i,j)}{f(i)}$$

where $f(i)$ is the frequency of the i-th amino-acid, and $M^x = U^{-1}\lambda^x U$.

8.3.3 Distance between DNA Sequences – Jukes-Cantor Model

According to the model of Jukes and Cantor [9] each base in the DNA sequence has an equal chance of mutating, and when it does, it is replaced by some other nucleotide uniformly. For a mutation probability of $3\alpha\Delta t$ during each infinitesimally small period of time Δt (frequency), the chance of a nucleotide x remaining unchanged over a period of T time units is (recall exercise #2):

$$P_{x\to x} = \frac{1}{4}(1 + 3e^{-4\alpha T})$$

Given a branch in the tree, the probability that the site is different at the two endpoints is therefore:

$$P_{u\neq v} = 1 - P_{x\to x} = \frac{3}{4}(1 - e^{-4\alpha T})$$

Other Related Models

- *Kimura's 2-parameters model [10]*
  Very similar to Jukes-Cantor model only there are two different rates
  - purine-purine(A,G) or pyrmidine-pyrmidine(C,T)
  - purine-pyrmidine or pyrmidine-purine
- Extensions of the Kimura's model to asymmetric base frequencies [4, 12].

8.3.4 Least Squares Methods

One of the more statistically justified methods to approximate a distance matrix is the *least squares* approach. In this formulation we are giving, for each pair of species, the measurred distance $Di, jbetweenthem, andtheweight$wi,j that intuitively quuantifies the accuracy of this measure .Our goal is to find a tree T, whose leaves are the n given species, and that predicts distances d_{ij} between the species, so that the following expression is minimized:

$$SSQ(T) \equiv \sum_{i=1}^{n} \sum_{j \neq i} w_{ij}(D_{ij} - d_{ij})^2 \tag{8.7}$$

The SSQ is a measure of the discrepancy between the observed distances $D_{i,j}$ and the distances $d_{i,j}$ predicted by T. The weights $w_{i,j}$ are usually all 1, or $w_{i,j} = \frac{1}{D_{i,j}^2}$.

Problem 8.5 Least Squares Tree.
INPUT: The distance $D_{i,j}$ between species i and j, for each $1 \leq i, j \leq n$, and a corresponding set of weights $w_{i,j}$.
QUESTION: Find the phylogenetic tree T and its branches length, with the species as its leaves, that minimize $SSQ(T)$.

Again, a "small" version of this problam is formulated for a given tree, only trying to minimize SSQ by determining the branches length. In general, the "large" problam of finding the least squares tree is NP-complete [2]. We will discuss two polynomial heuristics – UPGMA and Neighbor-Joining. We have already studied these algorithms in lecture #5, where we used them to iteratively add one additional string to a growing multiple alignment, thus obtaining a *progressive alignment*.

8.3.5 UPGMA

Being able to assign branch lengths to a given tree, as we have demonstrated, we need to minimize $SSQ(T)$ over the possible tree topologies. The *UPGMA*, or Unweighted Pair Group Method with Arithmetic mean [13], is a heuristic algorithm that usually generates satisfactory results. Basically, the algorithm iteratively joins the two nearest clusters (or groups of species), until one cluster is left.

UPGMA algorithm:
Let d be the distance function between species, we define the distance Di, j between two clusters of species C_i and C_j the following:

$$D_{i,j} = \frac{1}{n_i + n_j} \sum_{p \in C_i} \sum_{q \in C_j} d(p, q)$$

where $n_i = |C_i|$ and $n_j = |C_j|$

- Initialization:
  1. Initialize n clusters with the given species, one species per cluster.
  2. Set the size of each cluster to 1: $n_i \leftarrow 1$.
  3. In the output tree T, assign a leaf for each species.

- Iteration:
  1. Find the i and j that have the smallest distance D_{ij}.
  2. Create a new cluster – (ij), which has $n_{(ij)} = n_i + n_j$ members.
  3. Connect i and j on the tree to a new node, which corresponds to the new cluster (ij), and give the two branches connecting i and j to (ij) length $\frac{D_{i,j}}{2}$ each.
  4. Compute the distance from the new cluster to all other clusters (except for i and j, which are no longer relevant) as a weighted average of the distances from its components:

$$D_{(ij),k} = (\frac{n_i}{n_i + n_j})D_{i,k} + (\frac{n_j}{n_i + n_j})D_{j,k}$$

  5. Delete the columns and rows in D that correspond to clusters i and j, and add a column and row for cluster (ij), with $D_{(ij),k}$ computed as above.
  6. Return to 1 until there is only one cluster left.

Complexity: The time and space complexity of UPGMA is $O(n^2)$, since there are $n - 1$ iterations, with $O(n)$ work in each one.

A *clocklike*, or *ultrametric*, tree is a rooted tree, in which the total branch length from the root to any leaf is equal. In other words, there is a "molecular clock" that ticks in a constant pace (i.e., the mutation rate is identical for all species), and all the observed species are at an equal number of ticks from the root (see also page 2). If the solution to the least squares problem is 0, and there is a molecular clock (i.e., the solution is a clocklike tree), then UPGMA is guaranteed to return the optimal solution. Actually, UPGMA implicitly assumes the existence of an ultrametric tree, which explains why the new node, (ij), is the mean of the two nodes that were joined to create it, as shown in figure 8.8. It is therefore not surprising that for substantially non-clocklike trees, the algorithm might give seriously misleading results.

Another assumption that $UPGMA$ does is *additivity*: In the "real" tree, distances between species are the sum of distances along the path between the corresponding leaves.

There are two corollaries of additivity that the next algorithm will use

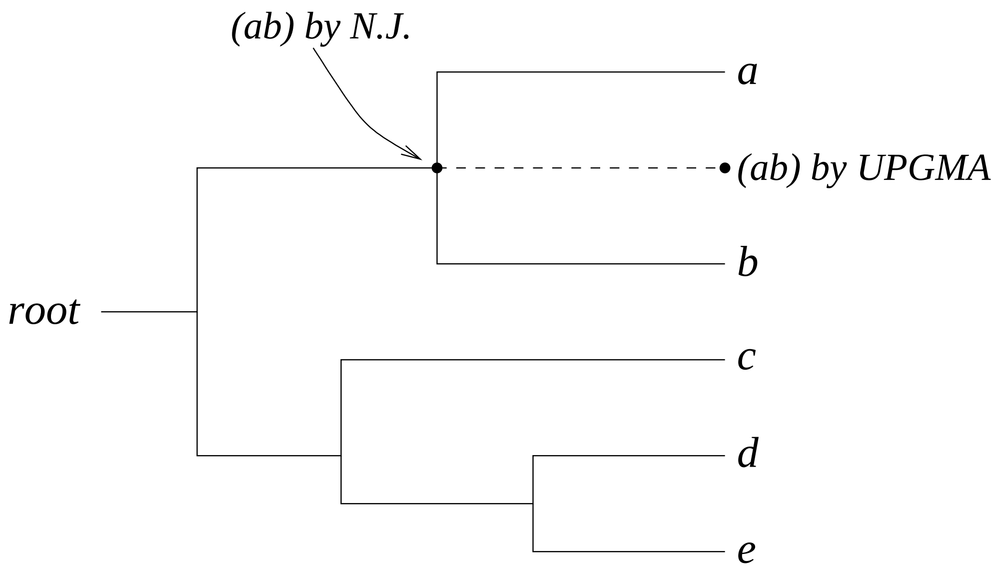

Figure 8.8: A clocklike tree, showing the clustering (ab) of the two nodes a and b by UPGMA and by the Neighbor-Joining algorithm.

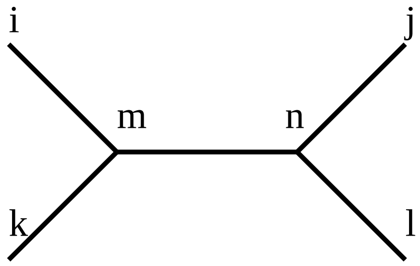

Figure 8.9: $di, j + dk, l <= di, j + dk, l = di, j + dk, l$.

- For every three nodes i,j,k connected through an internal node m with the distances: $d(i, m) = a, d(j, m) = b, d(k, m) = c$ then $dm, k = 1/2(di, j + dj, k - di, j)$.
- For every four nodes i,j,k,l connected through an two internal nodes m,n where m is connected with i, k and n, and n is connected with j, l and m then $di, k + dj, l <= di, j + dk, l = di, l + dk, j$ (see figure 8.9).

8.3.6 Neighbor Joining

The *Neighbor-Joining* algorithm is another quick clustering technique, which attempts to approximate the least squares tree, this time relying strongly on the additivity (and its implied corollaries) but without resorting to the assumption of a molecular clock. The idea here is to join clusters that are not only close to one another, but are also far from the rest. In each iteration, the algorithm attempts to find the direct ancestor of two species in the tree. For node i, its distance u_i from the rest of the tree is estimated using the formula: $u_i = \sum_{k \neq i} \frac{D_{i,k}}{(n-2)}$. In order to minimize the sum of all branch lengths, also known as the *minimum-evolution* criterion, the nodes i and j that are clustered next are those for which $D_{i,j} - u_i - u_j$ is smallest as can see in figure 8.10 (the reader is referred to [11] for a more elaborate explanation on this issue). The lengths $d_{k,(ij)}$ of the new branches are calculated by solving the same system of linear equations mentioned earlier in section 8.3.1. The solutions are written below, in equations 8.8 and 8.9. Neighbor-Joining has a running time of $O(n^2)$, like UPGMA.

Neighbor-Joining algorithm [14]:

- Initialization: same as in UPGMA (see 8.3.5).
- Iteration:
  1. For each species, compute $u_i = \sum_{k \neq i} \frac{D_{i,k}}{(n-2)}$.

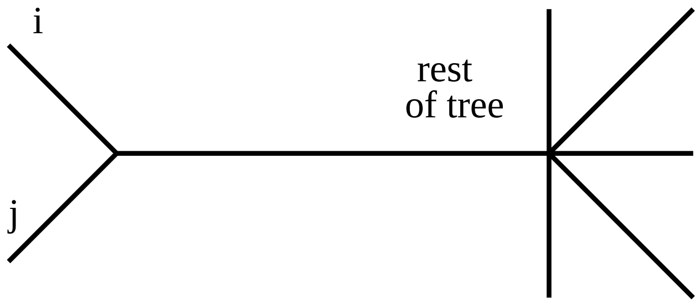

Figure 8.10: $D_{i,j} - u_i - u_j$ is the smallest, which means they are close to one anather and far from the rest of the tree, therefore the neighbor-joining algorithm will cluster them together.

2. Choose the i and j for which $D_{i,j} - u_i - u_j$ is smallest.
3. Join clusters i and j to a new cluster – (ij), with a corresponding node in T. Calculate the branch lengths from i and j to the new node as:

$$d_{i,(ij)} = \frac{1}{2}D_{i,j} + \frac{1}{2}(u_i - u_j) \quad , \quad d_{j,(ij)} = \frac{1}{2}D_{i,j} + \frac{1}{2}(u_j - u_i) \tag{8.8}$$

4. Compute the distances between the new cluster and each other cluster:

$$D_{(ij),k} = \frac{D_{i,k} + D_{j,k} - D_{i,j}}{2} \tag{8.9}$$

5. Delete clusters i and j from the tables, and replace them by (ij).
6. If more than two nodes (clusters) remain, go back to 1. Otherwise, connect the two remaining nodes by a branch of length $D_{i,j}$.

8.4 Maximum Likelihood

Given a tree, we often wish to have a statistical measure of how well it describes our data. As we have seen earlier in the course, we can use the likelihood score to estimate our hypothesis, which is in this case a phylogenetic tree T. For a set of species with observed values M, we would choose the tree that maximizes $P(M|T)$. In the following section, we shall assume that the tree topology is known, and show how to find the optimal branch lengths. To this end, we will first demonstrate how to calculate the likelihood of a tree efficiently. We will not discuss the issue of searching among tree topologies, which suffers from the same difficulties we mentioned in the previous sections, although is not proven to be NP-complete.

8.4.1 Likelihood of a Tree

For the analysis below, we shall use the following terms:

Definition *Labels*, or *states*, are the vectors of m character values associated with each species, or node in the tree (we will refer to a node and to its label interchangeably). A *reconstruction* is a full labeling of the tree's internal nodes. A *branch length* t_{vu} is the length of the edge between nodes v and u, and it measures the biological time, or genetic distance, between the species associated with these nodes.

As always, we assume that the characters are pairwise independent, and that the branching is a Markov process, that is, the probability of a node having a given label is a function only of the state of the parent node and the branch length, t, between them. Our model also includes a distance function to compute the latter probability, i.e.: $P_{x\to y}(t_{vu})$, the probability that state x will transform into state y within the time t_{vu}. We further assume that the character frequencies are fixed throughout the evolutionary history, and that they are given as $P(x)$.

Problem 8.6 Calculate the Likelihood of a Tree.
INPUT:

- A matrix M describing a set of m characters for each one of n given species.
- A tree T with the above species as its leaves and with known branch lengths t_{vu}.

QUESTION: Calculate the likelihood L of the tree: $L = P(M|T)$.

First, let us deal with a simple case, where there is only one character identifying each species. Since the labels of the internal nodes are unknown, we need to sum over all possible reconstructions. For example, for the tree illustrated in figure 8.11, we can immediately write down the following formula:

$$L = P(M|T) = \sum_r \sum_v P(r) \cdot P_{r\to s}(t_{rs}) \cdot P_{r\to v}(t_{rv}) \cdot P_{v\to u}(t_{vu}) \cdot P_{v\to w}(t_{vw}) \tag{8.10}$$

where r and v are possible labels (character values) for the corresponding nodes.

To expand the formula for multiple characters, we simply need to repeat the above calculation for each character separately, and then multiply the results (recall the assumption that the characters are pairwise independent). The general equation is now:

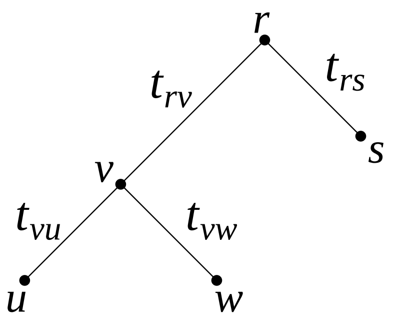

Figure 8.11: A simple tree with branch lengths. The likelihood of this tree is calculated in equation 8.10.

$$
\begin{aligned}
L &= P(M|T) = \prod_{\text{character j}} P(M_j|T) \\
&= \prod_{\text{character j}} \left\{ \sum_{\text{reconstruction R}} P(M_j, R|T) \right\} \\
&= \prod_{\text{character j}} \{ \sum_{\text{reconstruction R}} \left[P(root) \cdot \prod_{\text{edge u}\to\text{v}} P_{u\to v}(t_{uv}) \right] \}
\end{aligned} \tag{8.11}
$$

Note: The trees inferred by maximum likelihood appear from this description to be rooted trees. However, if the model of character substitution is reversible, i.e., $P(x)P_{x\to y}(t) = P(y)P_{y\to x}(t)$, then the tree is actually unrooted – the root can be chosen arbitrarily, without any change is the likelihood of the tree.

It now remains to show how this calculation can be performed efficiently. The following dynamic-programming "pruning" algorithm was introduced by Felsenstein [3].

We can take this approach because of the tree likelihood properties in the Markov's model:
Additivity - $\P_{x\to y}(t+s) = sum_y P_{x\to y}(t) P_{y\to z}(s)$
Reversibility- $P(x)P_{x\to y}(t) = P(y)P_{y\to x}(t)$

Calculating the likelihood of a tree using Dynamic Programming:

For a character j, denote:

$$C_j(x, v) = P(\text{ subtree whose root is } v \mid v_j = x \text{ })$$

$C_j(x, v)$ is the conditional likelihood of v's subtree, i.e., the probability of everything that is observed from node v on the tree down to the leaves, at character position j, given that v has the label x at this position.

- Initialization:
  For each leaf v and state x:

$$C_j(x,v) = \begin{cases} 1 & v_j = x \\ 0 & \text{otherwise} \end{cases} \tag{8.12}$$

- Recursion:
  Traverse the tree in postorder; for an internal node v with children u and w, compute for each possible state x:

$$C_j(x,v) = \left[\sum_y C_j(y,u) \cdot P_{x\to y}(t_{vu}) \right] \cdot \left[\sum_y C_j(y,w) \cdot P_{x\to y}(t_{vw}) \right] \tag{8.13}$$

- The final solution is:

$$L = \prod_{j=1}^{m} \left[\sum_x C_j(x, root) \cdot P(x) \right] \tag{8.14}$$

Complexity: For n species, m characters, and k possible states for each character, we perform $O(m \cdot k^2)$ work in $O(n)$ nodes, so the running time of the algorithm is $O(n \cdot m \cdot k^2)$.

8.4.2 Finding the Optimal Branch Lengths

We are now ready to tackle the more difficult task of finding the optimal branch lengths for a given tree topology. First, let us assume that all the lengths are known except for t_{rv}. If r is the root (as in figure 8.11), then we get:

$$\log L = \sum_{j=1}^{m} \log \left[\sum_{x,y} P(x) \cdot C_j^v(x,r) \cdot P_{x\to y}(t_{rv}) \cdot C_j^r(y,v) \right] \tag{8.15}$$

Notation: $C_j^u(x,y)$ means $C_j^u(x,y)$ in a tree where u is the root.
which is an elementary function of t_{rv} and some constants.

We now need to maximize $\log L$ with respect to t_{rv}. This can be done by many standard methods, e.g., Newton-Raphson, or EM algorithm. The same process we have just demonstrated can also be applied when r is not the original root. As explained earlier, assuming reversibility, for any x, y, and t, then the root can be set at any node, without affecting L. In other words, in order to find an optimal branch length between nodes r and v, we simply need to hang the tree from r, so that the previous analysis holds.

Our next step is to find optimal branch lengths, when none of them are known apriori. The main problem is that once one branch has changed length, there is no guarantee that the others are still at their optimal lengths. On the contrary, the branches are clearly not pairwise independent. In practice, however, locally improving the likelihood by optimizing the length of one branch at a time works quite well, as there are not very strong interactions between branch lengths. After a few sweeps through the tree, calculating the optimal length of each edge separately, the likelihood converges, and the result is a near-optimal phylogenetic tree.

Bibliography

[1] C. Bron and J. Kerbosch. Algorithm 457: Finding all cliques of an undirected graph. *Communications of the Association for Computing Machinery*, 16:575–577, 1973.

[2] W. H. E. Day. Computational complexity of inferring phylogenies from dissimilarity matrices. *Bulletin of Mathematical Biology*, 49:461–467, 1986.

[3] J. Felsenstein. Maximum likelihood and minimum-steps methods for estimating evolutionary trees from data on discrete characters. *Systematic Zoology*, 22:240–249, 1973.

[4] J. Felsenstein. Phylogenies from molecular sequences: inference and reliability. *Annuals Rev. Genetics*, 22:521–565, 1988.

[5] J. Felsenstein. *Inferring Phylogenies.* ASUW Publishing, Seattle, WA, 1998.

[6] W. M. Fitch. Toward defining the course of evolution: minimum change for a specified tree topology. *Systematic Zoology*, 20:406–416, 1971.

[7] Jr. G.F.Estabrook, C.S.Johnson and F.R.McMorris. An algebraic analysis of cladistic characters. *Dsicrete Mathematics*, 16:141–147, 1976.

[8] M. D. Hendy and D. Penny. Branch and bound algorithms to determine minimal evolutionary trees. *Mathematical Biosciences*, 60:133–142, 1982.

[9] T. H. Jukes and C. Cantor. *Mammalian Protein Metabolism*, chapter Evolution of protein molecules, pages 21–132. Academic Press, New York, 1969.

[10] M. Kimura. A simple method for estimating evolutionary rates of base substitutions through comparative studies of nucleotide sequences. *Molecular Evolution*, 16:111–120, 1980.

[11] W. H. Li. *Molecular Evolution*, chapter 5, pages 105–112. Sinauer Associates, Inc., Publishers, Sunderland, Massachusetts, 1997.

[12] T. Yano M. Hasegawa, H. Kishino. Dating of the human-ape splitting by a molecular clock of mitochondrial dna. *Molecular Evolution*, 22:160–174, 1985.

[13] C. D. Michener and R. R. Sokal. A quantitative approach to a problem in classification. *Evolution*, 11:130–162, 1957.

[14] N. Saitou and M. Nei. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, 4:406–425, 1987.

[15] D. D. Sankoff. Minimal mutation trees of sequences. *SIAM Journal on Applied Mathematics*, 28:35–42, 1975.

[16] E. O. Wilson. A consistency test for phylogenies base on contemporaneous species. *Systematic Zoology*, 14:214–220, 1965.