Algorithms for Molecular Biology **Fall Semester, 2001**

Lecture 8: December 27, 2001

Lecturer: Ron Shamir *Scribe: Orly Stettiner and Ron Gabor*[1]

8.1 Preface: Phylogenetics and Phylogenetic Trees

Phylogeny - The ancestral relationship of a set of species.

8.1.1 What is Phylogenetics?

Phylogenetics is the area of research concerned with finding the genetic 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*[2] (see Figures 8.1 and 8.2).

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). The relationships between species are then deduced from well conserved blocks in the alignment of several sequences, one from each 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", see Figure 8.4).

When studying phylogeny using nuclear genes, we encounter several difficulties. 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.

---

[1] Based on a scribe of Yuval Inbar and Tzvika Hartman, December 18, 2000

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

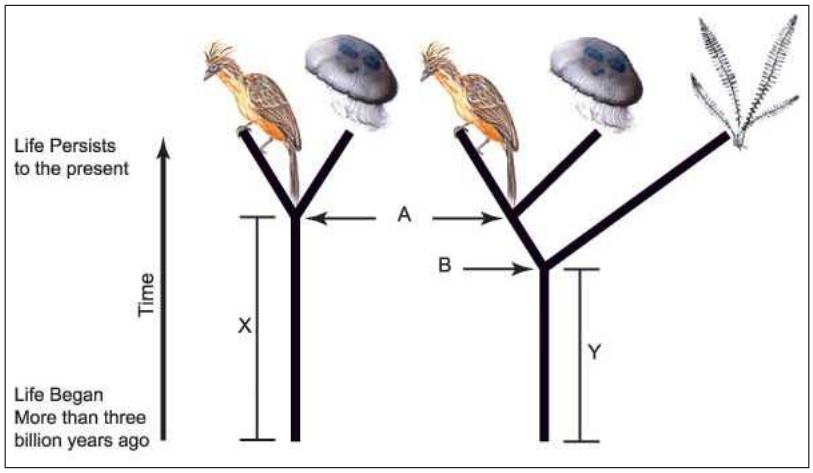

Figure 8.1: (Source [1]) Phylogeny in a nut shell. A: The most recent common ancestor of the bird and the jellyfish. At this point the two lineages diverged or split. X: The portion of history the bird and the jellyfish share. Their linages were one during that time. B: The most recent common ancestor of the bird, jellyfish and fern. Y: The portion of history the bird, jellyfish and fern share. The bird and jellyfish share a more recent common ancestor (A) then either does with fern (B). Therefore, they are more closely related to each other than either is to the fern.

When discussing matching genes in different species, several types of matches may exist:

Orthologous both genes are "the same" gene in the strong sense – they are connected directly, and not through a duplication. This usually means common ancestral species.

Paralogous matches of genes that are duplication of what used to be the same gene (in ancestral species). This kind of match is usually the result of some duplication along the evolutionary line.

Xenologs (horizontal transfers) matches of genes that were transferred between organisms in other ways, such as injection by a virus.

Example of phylogenic trees based on orthologous and paralogous are shown in Figure 8.3.

In order to simplify the analysis of ancestral relationships, we shall limit gene matches to ortholog matches only, as if paralogs or xenologs do not exist. Needless to point, wrong identification of orthologous will result in trees that have little to do with the real evolutionary species relationships (see Figure 8.5).

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

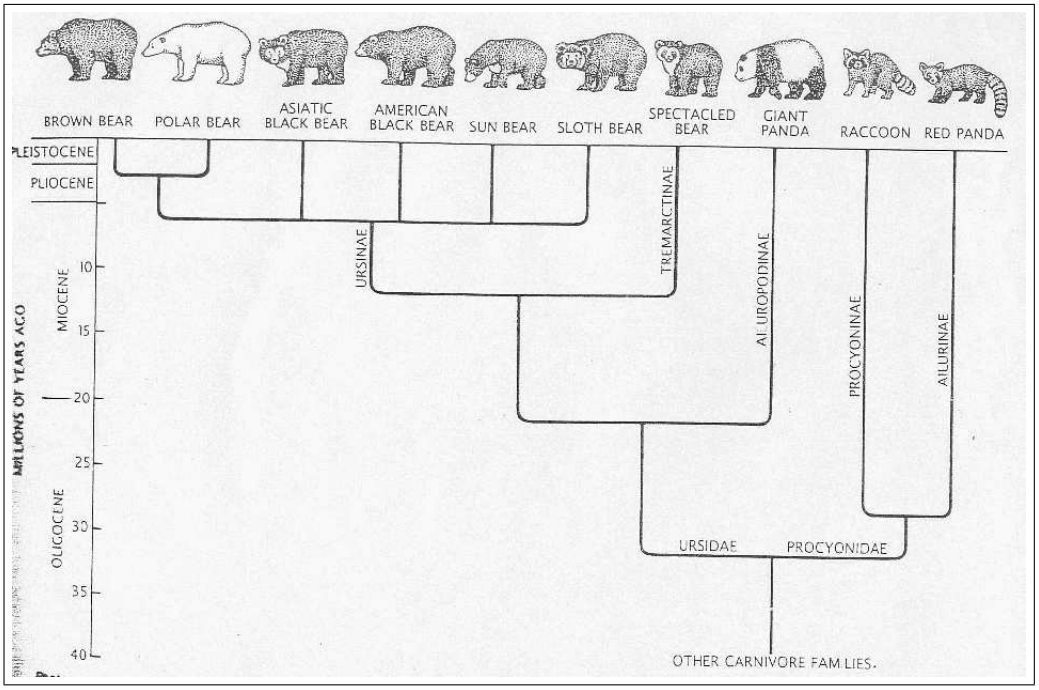

Figure 8.2: A phylogenic tree based on data obtained from modern molecular genetic methods places the giant panda in the Ursidae, bear family. The red panda is left in the Procyonidae, or racoon family. Molecular analysis of the chromosomes of these pandas suggests that the racoon and bear families diverged from a common carnivorous ancestor about 35 to 40 million years ago.

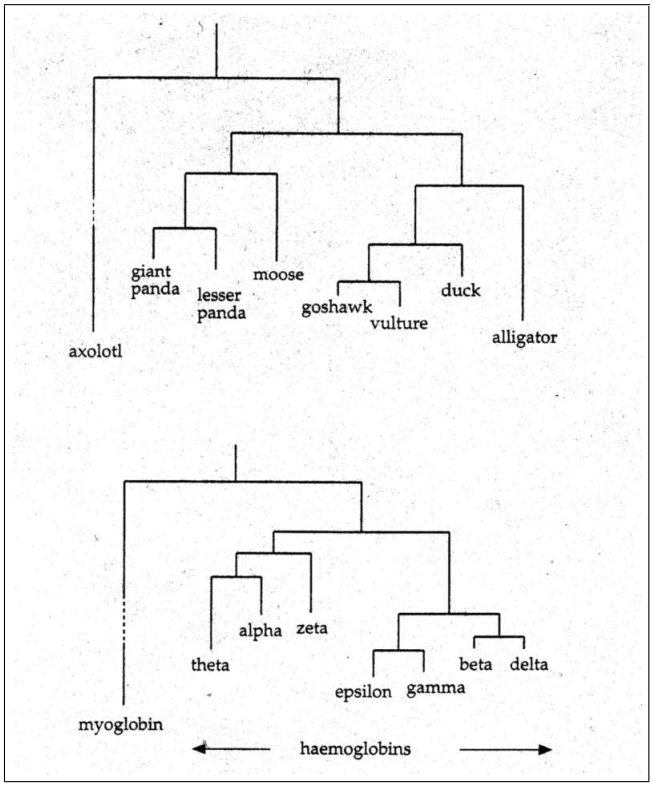

Figure 8.3: (Source [20]) Above: a tree of orthologues based on a set of alpha haemoglobins. Below: a tree of paralogues, the alpha, beta, gamma, delta, epsilon, zeta and theta chains of human haemoglobins, and human myoglobin.

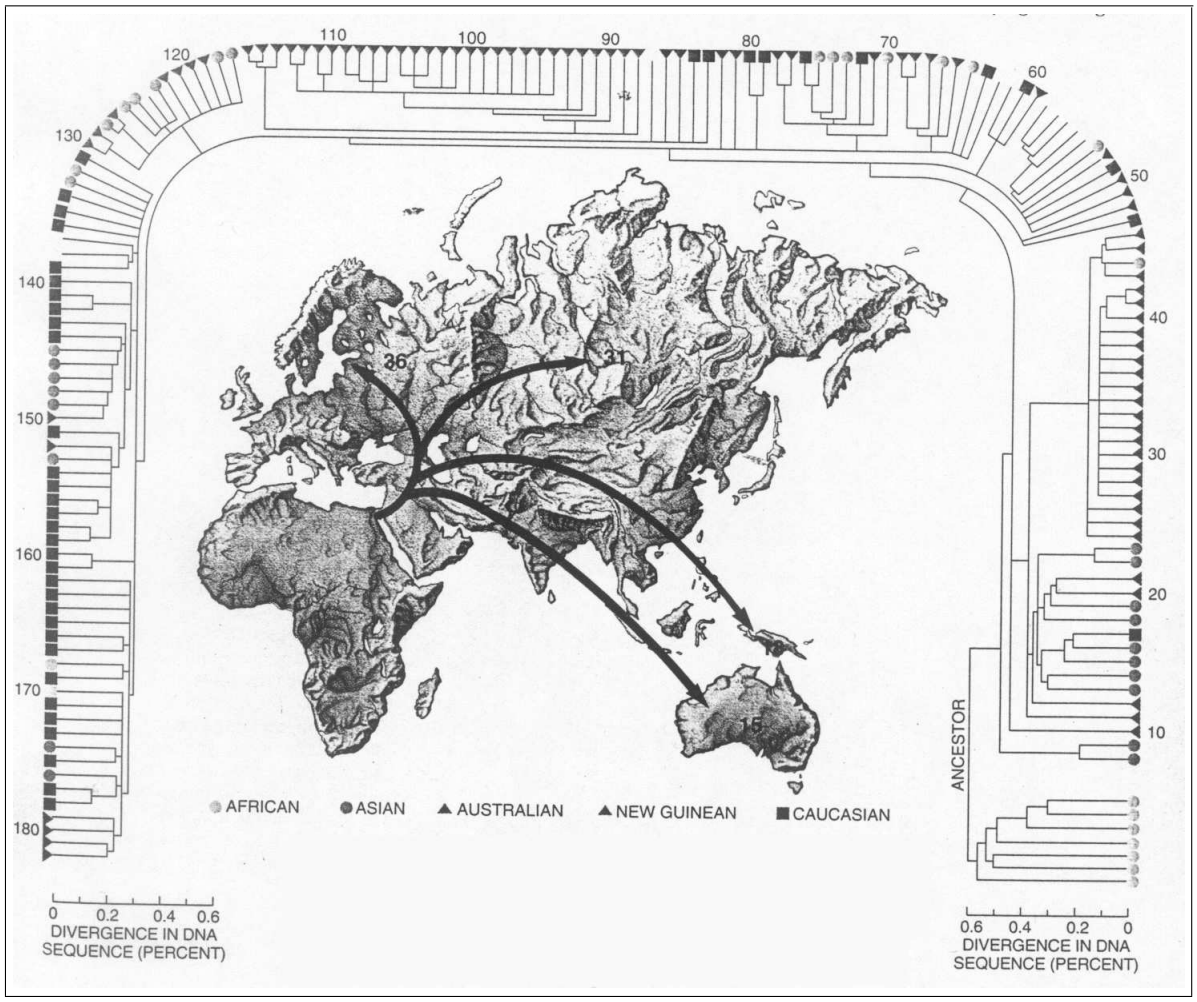

Figure 8.4: (Source: [24]) African origin for all modern human is indicated by the genetic evidence. The arrows on the maps (center) indicate the minimum number of unrelated females who colonized major geographic areas, as inferred from the branching pattern.

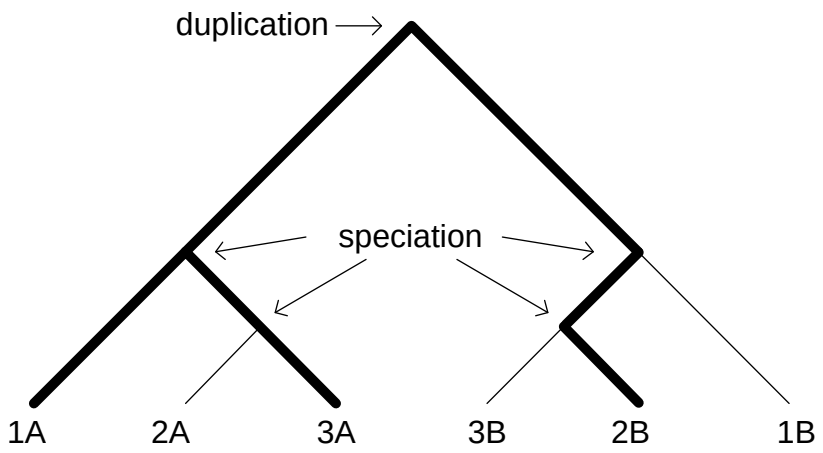

Figure 8.5: A phylogenic tree based on wrong identification of ortholog matches. An ancestral species, had a gene duplication event, which resulted in two copies of the same gene (copies are marked as A and B), which continued to evolve independently. A speciaition event caused three species to evolve (denoted as 1, 2 and 3). Using genes $1A$, $2B$ and $3C$ as a basis for constructing a phylogenic tree would most probably result in the tree denoted by bold edges, which marks species 2 as distant from 1 and 3, which are close to each other (which is clearly not correct).

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, shows how close (or distant) the species are, and 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 8.6 and 8.7 show a rooted tree and its unrooted counterpart, respectively.

A binary, or *bifurcating*, tree is a tree in which a node may have only 0 to 2 children, that is, in an unrooted tree, up to three neighbors. It is sometimes useful to allow more than 2 children (*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:

Character Based Examine each character (e.g., a base in a specific position in a DNA sequence or amino acid in a protein) separately.

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 disagree on).

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.
- Optional parameters which are problem dependent.

GOAL: Find a 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 j-th character of the i-th species. The state (value) of each character j 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, a 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.

Assume that in evolution, each split is a two-way split, giving rise to bifurcating trees. We can either consider the space consisting of rooted or unrooted tree because it can be shown that the number of rooted trees with n leaves is equal to the number of unrooted trees with $n + 1$ leaves. Consider a rooted tree with 2 leaves. There is only one such tree. There are 3 possible branches where we can add an extra leaf, giving a rooted tree with 3

leaves. Given a rooted tree with 3 leaves, there are 5 branches where we can add the fourth leaf. Proceeding in this way, it can be shown that the number of non-isomorphic, labeled, binary, rooted trees containing n leaves is:

$$1 * 3 * 5 * ... * (2n-3) = \prod_{i=2}^{n}(2i-3) = (2n-3)!! \tag{8.1}$$

The number of rooted trees is of course super-exponential – for $n = 20$, for instance, there are about 10^{21} such trees. This means that exhaustive enumeration is infeasible even for a relatively small number of species. The same holds for the number of unrooted trees, which is $(2n-5)!!$[3] (equals to the number of rooted trees with $n-1$ leaves).

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

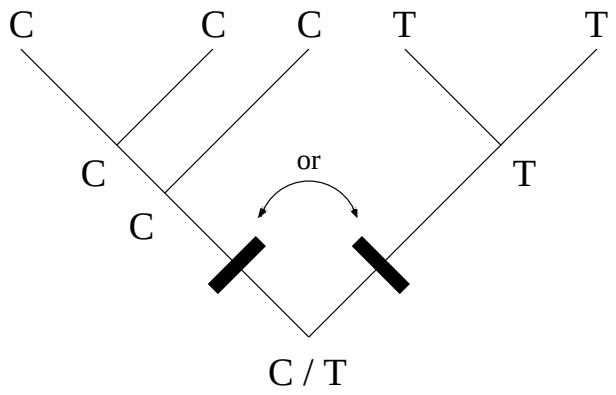

Figure 8.6: A most parsimonious 5-species phylogeny for a single DNA site (3 species have C character and two have T). The bars mark the two possible edges along which a mutation might have occurred.

8.2.1 Parsimony

One intuitive score for a phylogenetic tree is the number of *changes (mutations/mismatches) 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.

[3]Proving this is simply by observing that a new node (leaf) can be added to every edge, and will result in the addition of two edges and one node (junction).

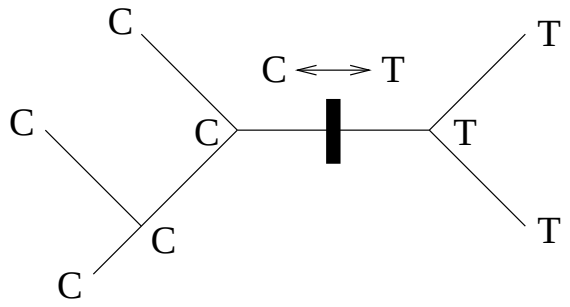

Figure 8.7: The unrooted counterpart of the phylogeny in Figure 8.6. Notice that there is now no ambiguity about the placement of the mutation.

The algorithm assigns a cost to a given tree, and searches through selected topologies to identify the "best" tree, which can explain the observed sequences with a minimal number of substitutes.

Let us mark the vertices of a tree by $V(T)$, and its edges by $E(T)$. Denote the value of the j-th character of vertex $v \in V(T)$ by v_j.

Definition Given a phylogenetic tree T, its *parsimony score* 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 8.6 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 8.7. 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 8.8 illustrates a more complex parsimony tree, in which the species have 6 characters each.

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

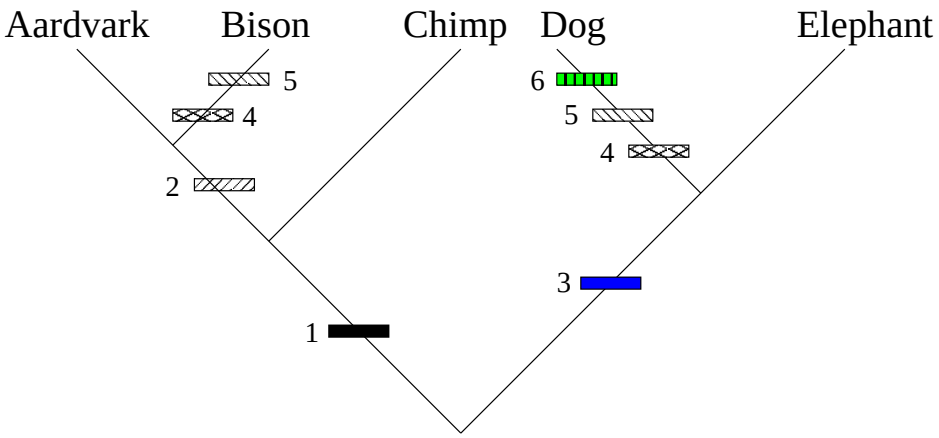

Figure 8.8: A most parsimonious 5-species phylogeny for 6 characters, reconstructed from the data in table 8.1. The numbers by the mutation bars indicate the changed character.

	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 8.8.

8.2.2 Small Parsimony and Fitch's Algorithm

Problem 8.2 Small Parsimony
INPUT: The topology of a rooted phylogenetic tree T with labeled leaves (i.e., all leaves have known sequences).
GOAL: Find a labeling of ancestral sequences implying minimum number of changes (most parsimonous) along the tree edges.

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 present Fitch's algorithm [8] in the next section.

Fitch's algorithm

The algorithm considers each site in a given DNA sequence separately. It computes the minimum number of changes required to explain the evolution of a given phylogeny representing a site in a DNA sequence. It is a dynamic programming style algorithm.

The algorithm constructs a set of possible states (possible nucleotides) for each internal node, which is computed from the states of its children. We start at the leaves of the phylogeny. Each leaf is labeled with the singleton set containing the nucleotide at that particular site. Then the tree is traversed in a postorder manner (such that all of the children of the current node have been visited before the current node). Following is a formal description.

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: Assign to each node v a set $S_v \subseteq A$ in the following manner:

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

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

To compute S_v we 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 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 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.

Complexity: For each node v we work $O(k)$ time to compute S_v, and again $O(k)$ to compute v_c. In total we spend $O(n \cdot k)$ 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 8.9 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.

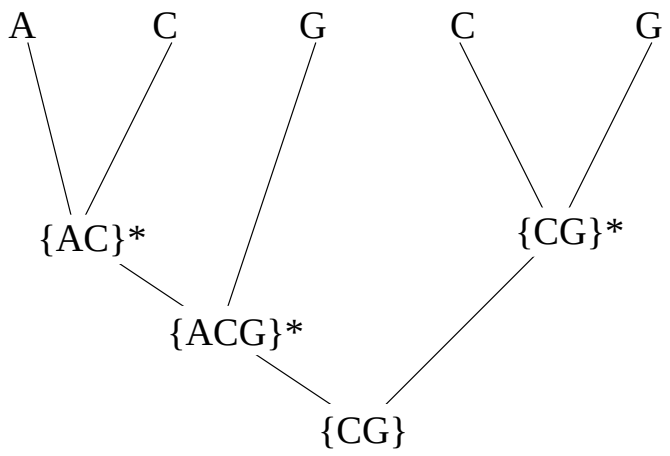

Figure 8.9: An example of step 1 of Fitch's algorithm for a 5-species phylogeny. Nodes marked by an asterisk (*) require a change along one of the edges to their children, adding 1 to the parsimony score.

8.2.3 Weighted Parsimony and Sankoff's Algorithm

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

Problem 8.3 Weighted Small Parsimony.
INPUT:

- K states, $S_1...S_k$.
- The topology of a rooted phylogenetic tree with labeled leaves.
- The costs C_{ij}^c for changing from state i to state j.

GOAL: Find ancestral sequences and alignments of them to the leaf sequences, that together minimize a tree-based parsimonious type cost.

Following is an algorithm by Sankoff [22] which generalizes[4] Fitch's algorithm:
Step 1: Compute for each node v and each state t a quantity $S_t(v)$ which is the minimum

[4]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.

cost of the subtree whose root is v, given $v_c = t$ (state of character c at v is t).
For each leaf v:

$$S_t^c(x) = \begin{cases} 0 & v_c = t, \\ \infty & \text{otherwise.} \end{cases} \tag{8.3}$$

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

$$S_t^c(v) = \min_i \{C_{ti}^c + S_i^c(u)\} + \min_j \left\{C_{tj}^c + S_j^c(w)\right\} \tag{8.4}$$

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

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

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

$$v_c = \arg\min_t (C_{u_c t}^c + S_t^c(v))$$

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 $O(m \cdot n \cdot k)$.

An illustration of Sankoff's algorithm can be seen in Figure 8.10.

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 8.2.7 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.

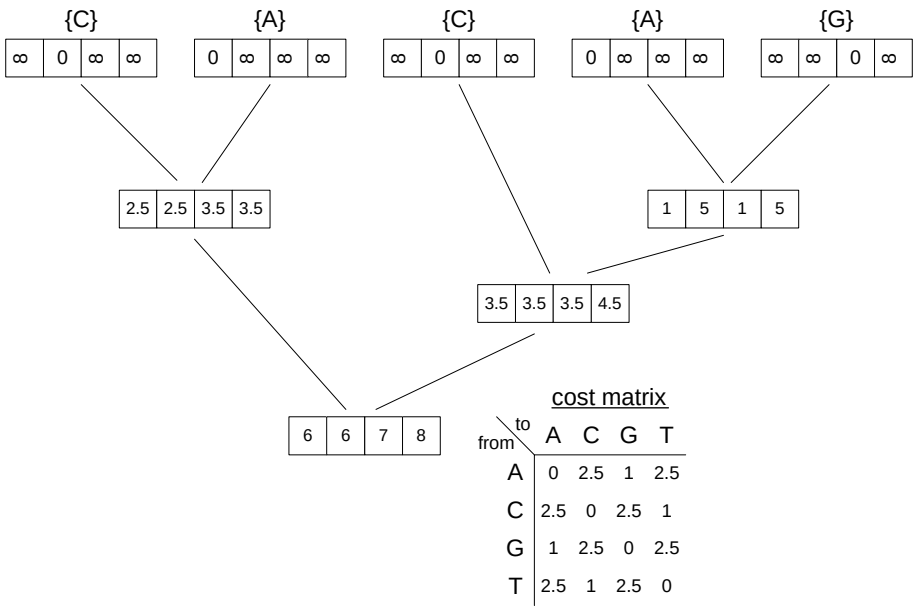

Figure 8.10: An example illustrating Sankoff's algorithm.

8.2.4 Large Parsimony Problem

Given that we have a way of determining the cost or score of a given tree, we now consider the combinatorial problem of finding the best tree that explains the given data. In the large parsimony problem, we have to search through the space of all possible trees to find the best tree.

Problem 8.4 Large Parsimony Problem.
INPUT: A matrix M describing m characters of a set of n species:
M_{ij} = state of the j-th character of the species i
M_i = label of i (all labels are distinct).
GOAL: Find a most parsimonious tree (topology, leaf labeling and internal sequences).

It can be shown (Gusfield [12]) that the Large Parsimony problem is equal to the unweighted Steiner tree problem on an m-dimenstional hypercube. It was established in the literature that this problem is NP-hard, both for the unweighted case (Foulds [9]) and for the weighted case (Gusfield [11]). However, we will present several efficient search strategies and algorithms attempting to solve the problem of large parsimony.

8.2.5 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 [13] in 1989. 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 k-th level of the search tree, we will have nodes representing all possible phylogenetic trees with k leaves for the first k species (the order does not matter, as long as it is pre-determined). The children 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 children.

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.

8.2.6 Nearest Neighbor Interchanges (NNI)

The idea of NNI is to rearrange a tree by dissolving the connections between subtrees to form different trees. It defines a neighborhood relation for the search-space, and given such a relation, traverses 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, hill-climbing, etc. – to find a local optimum, which will hopefully be also the global optimum.

There are many ways to define neighborhood relationship among trees. For instance, we can use the notion of quartets -trees with four leaves. We can say that two unrooted[5]

[5]In this section we deal with unrooted trees only.

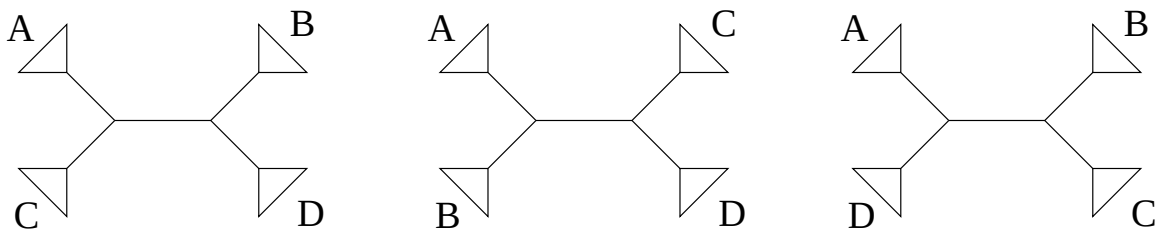

Figure 8.11: Neighbor interchange options referred to as (left to right) $AC|BD$, $AB|CD$, $AD|BC$, of an unrooted tree with 4 subtrees A, B, C, and D.

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 8.11). This relation is called Nearest Neighbor Interchange. For trees of 5 species, its neighborhod graph is known as Peterson's graph. This is also referred to as *Branch Swapping*.

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

8.2.7 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.

The compatibility criterion is a simplification of parsimony, which is most useful when all of the characters used to build a phylogeny are binary.

Definition A binary character c is *compatible* with a tree T iff there exists an assignment of the states of the character to nodes of T, such that a change occurs only along a single edge.

This definition clearly states that the assignment of a compatible character must group each state into a single connected graph component in the tree. The general case for character with any number of states is:

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

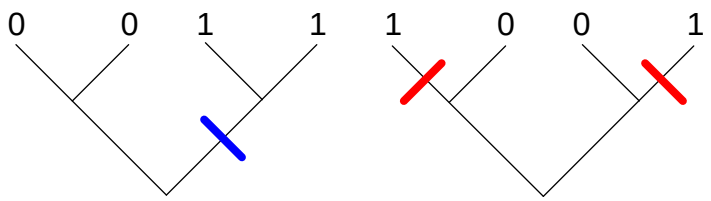

Figure 8.12: compatible and incompatible binary characters defined on a tree over four species. The character on the left is compatible with the tree because only a single edge change (indicated by the heavy line) is required to evolve a single, ancestral state at the root into the states at the leaves. In contrast, the character on the right requires two changes.

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. An example of compatibility is shown in Figure 8.12.

8.2.8 Compatibility and Parsimony

Compatibility, in the binary case at least, is easily shown to be a special case of parsimony. To solve the compatibility problem, we will use a slightly modified, *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 regular 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.

8.2.9 Pairwise Compatibility (compatibility of binary characters)

The first step in working with compatibility, is parallel to the small parsimony problem (see 8.2.2): 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 8.2.2).

The more interesting problem here is of course that of "large compatibility" – finding

Species	Character X	Character Y
Species 1	0	1
Species 2	1	0
Species 3	0	0
Species 4	0	1
Species 5	1	0

Table 8.2: An example of 5 Species with two characters (X and Y). Character X and Y have only three of the four possible pairing ($\{1,1\}$ does not exist), and hence are compatible according to the pairwise compatibility test.

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 binary characters c_1 and c_2 in a set S of species, 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 *mutually compatible* if there exists a tree T such that for every i, c_i is compatible with T.

Theorem 8.1 *(Wilson [25]) Pairwise Compatibility Test:*
For characters i, j define the set $S_{ij} := \{(x, y) : \exists$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 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 us assume then that $\{0, 1\}^2 \setminus S_{cc'} = \{(x, y)\}$. Figure 8.13 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. identifies jointly compatible sets of binary characters:

Theorem 8.2 *(Estabrook et al. [10]) Pairwise Compatibility Theorem:*
All binary characters in a set S are jointly compatible iff $\forall c, c' \in S, PC(c, c')$.

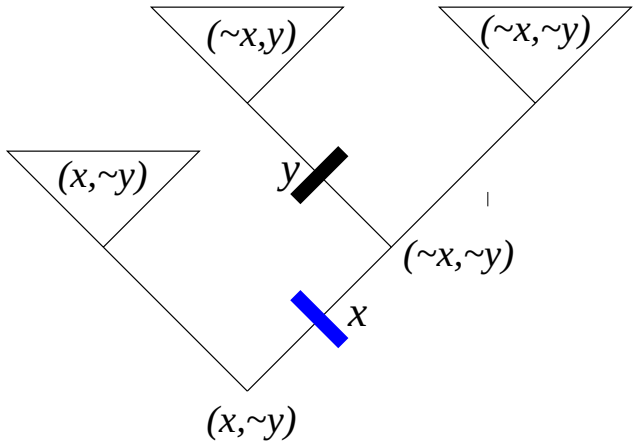

Figure 8.13: A schematic description of a tree that is compatible with two characters, having 3 combined values (see proof of theorem 8.1).

That is, a set of characters is mutually compatible iff they are pairwise compatible.
Note that the theorem does not hold for characters that are not binary. We will not present a proof for this theorem.

From the above discussion, 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:

$$G = (V, E);\ V = \{v_1, \ldots, v_m\};\ E = \{(v_i, v_j) : PC(c_i, c_j)\} \tag{8.7}$$

This seems to be of no great help, because as we know, finding a maximum clique in a graph is NP-hard. However, there are algorithms, such as Bron and Kerbosch's [3] 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.

8.2.10 Perfect Phylogeny - Large Compatibility Problem

Problem 8.5 Large Compatibility Problem.
INPUT: Matrix M of binary character states.
GOAL: Find a maximum number of mutually compatible characters, and the corresponding tree.

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, it was established (Kannan and Warnow [15]) that if the number of states that can be taken by each character equals 3 or 4, the problem has a polynomial-time solution in terms of n and m (the matrix dimensions). However, if the number of states is a variable, it can be shown (Bodlaender [2] , Steel [23]) that the generalized perfect phylogeny problem is NP-complete.

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^*. At each step of the algorithm, one of the characters in S^* is used 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 8.2.2.

8.3 Distance Based Methods

Phylogenies based on parsimony and compatibility both try to minimize the number of changes required to explain observed phenotype differences among a set of species. The "distance" between species, which is the number of character changes postulated to mutate one species into another, is unknown until a particular tree is chosen. An alternative approach to phylogeny is to define interspecies distances a priori, and then search for a tree which is consistent with these distances. This approach is at the core of all the distance-based phylogeny methods.

Formally, a distance-based method defines a distance function $d(i, j)$ on pairs of species (i, j).The distance function is inferred from the species according to some model of character change. The distance-based phylogeny problem is to find an edge-weighted tree in which the path joining species i and j has total weight $d(i, j)$ for every species pair (i, j).

Problem 8.6 Distance Based Phylogeny.
INPUT: Matrix d of distances between species.
GOAL: Find a tree, which is consistent with the distances given in d; the edges of the tree will have weights equal to the distances, and the species will be the leaves..

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 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 8.14), 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 8.14 illustrates a small tree, and table 8.3 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.

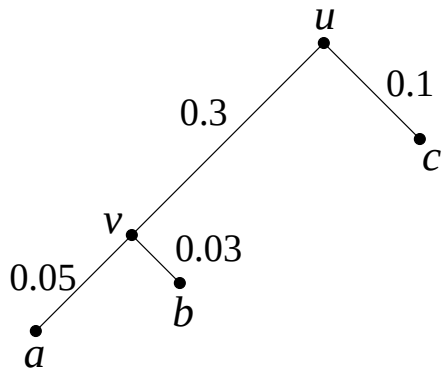

Figure 8.14: A small tree with 3 species – a, b, and c. The branch lengths correspond to the pairwise distances in table 8.3.

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

Table 8.3: Distances $d_{i,j}$ predicted by the tree in Figure 8.14.

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 [14] 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 (see Figure 8.15):

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

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 [16]*
  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 [6, 18].

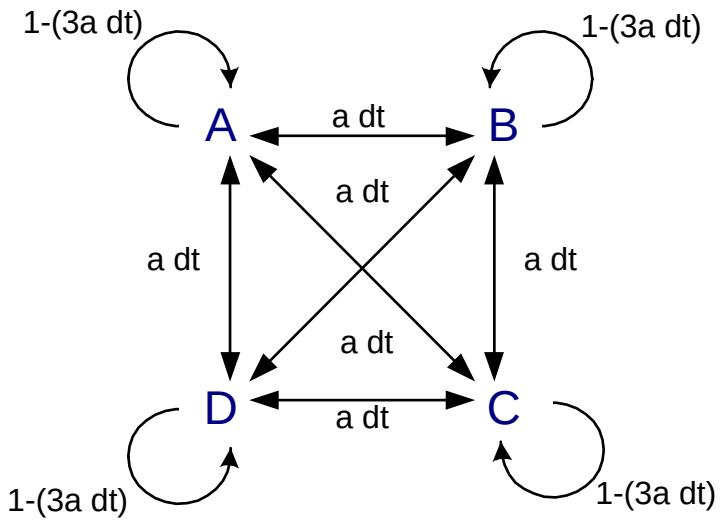

Figure 8.15: The Markov process version of the Jukes/Cantor model. a is the rate of mutation, and dt is the time interval per step of the process.

8.3.4 Computing Distance-Based Phylogencies: 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 measured distance Di, j between them, and the weight wi, j that intuitively quantifies 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:

$$SSQ(T) \equiv \sum_{i=1}^{n} \sum_{j \neq i} w_{ij}(D_{ij} - d_{ij})^2 \tag{8.8}$$

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 8.7 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 $w_{i,j}$.
GOAL: Find the phylogenetic tree T and its branch length, with the species as its leaves, that minimize $SSQ(T)$.

Again, a "small" version of this problem is formulated for a given tree, only trying to minimize SSQ by determining the branch lengths. In general, the "large" problem of

finding the least squares tree is NP-complete [4]. 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

UPGMA, or Unweighted Pair Group Method with Arithmetic mean [19], is a clustering procedure, which is simple and intuitive. It works by clustering the sequences, at each stage amalgamating two clusters, and at the same time creating a new node on a tree. The tree can be imagined as being assembled upwards, each node being added above the others and the edge lengths being determined by the difference in the heights of the nodes at the top and bottom of an edge.

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. UPGMA 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.[12pt]

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 as follows:

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

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: combine two clusters to form a new cluster.
  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.

Molecular Clocks and Ultrametric Property of distances

UPGMA produces a rooted tree of a special kind - a *clocklike*, or *ultrametric*, 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 7). 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 8.16. 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

- 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 $D(m,k) = 1/2(D(i,k) + D(j,k) - D(i,j))$ (see Figure 8.17).

- 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 the following holds: $D(i,k) + D(j,l) \leq D(i,j) + D(k,l) = D(i,l) + D(k,j)$ (see Figure 8.17).

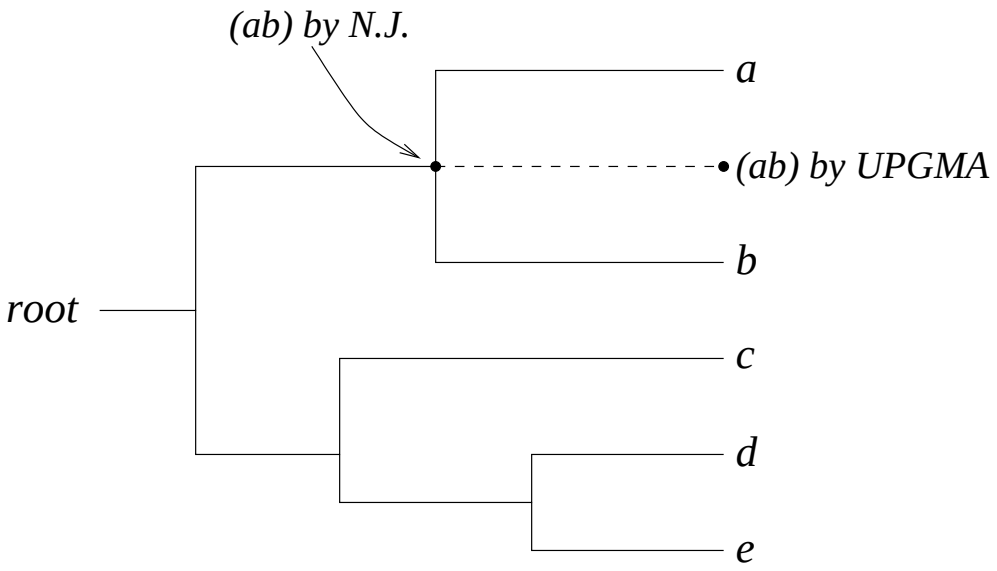

Figure 8.16: A clocklike tree, showing the clustering (ab) of the two nodes a and b by UPGMA and by the Neighbor-Joining algorithm.

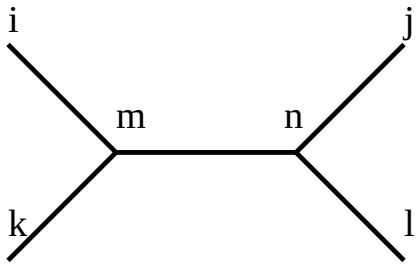

Figure 8.17: $D(i,j) + D(k,l) \leq D(i,j) + D(k,l) = D(i,j) + D(k,l)$.

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 be seen in Figure 8.18 (the reader is referred to [17] 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 8.9 and 8.10. Neighbor-Joining has a running time of $O(n^2)$, like UPGMA.

Neighbor-Joining algorithm [21]:

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

$$d_{i,(ij)} = \frac{1}{2} D_{i,j} + \frac{1}{2}(u_i - u_j) \quad , \quad d_{j,(ij)} = \frac{1}{2} D_{i,j} + \frac{1}{2}(u_j - u_i) \tag{8.9}$$

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

$$D_{(ij),k} = \frac{D_{i,k} + D_{j,k} - D_{i,j}}{2} \tag{8.10}$$

  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 $D_{i,j}$.

The time complexity of the algorithm is $O(n^2)$.

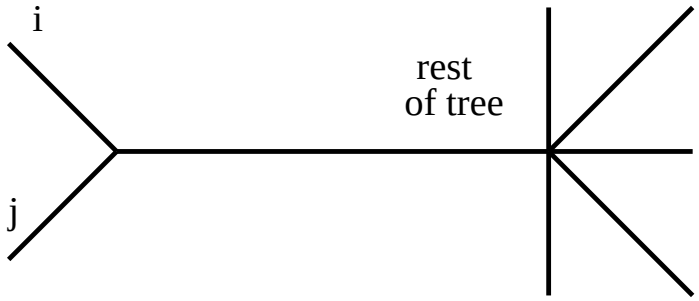

Figure 8.18: $D_{i,j} - u_i - u_j$ is the smallest, which means they are close to one another and far from the rest of the tree, therefore the neighbor-joining algorithm will cluster them together.

8.4 Probabilistic Approaches

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 $P(x)$.

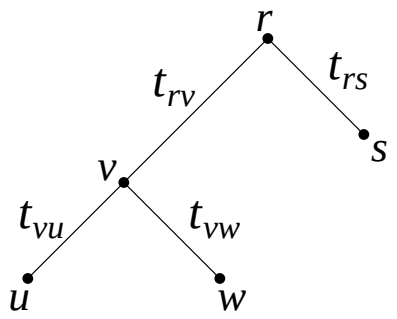

Figure 8.19: A simple tree with branch lengths. The likelihood of this tree is calculated in equation 8.11.

Problem 8.8 Calculating 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}.

GOAL: Maximize probability of $P(M|T)$ by finding an optimal reconstruction T, its internal nodes labeling and branch lengths.
Assumptions:

- Characters are independent of each other.
- Markov model - probability of a label is depends only on its parent.

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 8.19, we can immediately write down the following formula:

$$L = P(M|T) = \sum_r \sum_v P(r) \cdot P_{r\to s}(t_{rs}) \cdot P_{r\to v}(t_{rv}) \cdot P_{v\to u}(t_{vu}) \cdot P_{v\to w}(t_{vw}) \tag{8.11}$$

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:

$$
\begin{aligned}
L &= P(M|T) = \prod_{\text{character j}} P(M_j|T) \\
&= \prod_{\text{character j}} \left\{ \sum_{\text{reconstruction R}} P(M_j, R|T) \right\} \\
&= \prod_{\text{character j}} \{ \sum_{\text{reconstruction R}} \left[P(root) \cdot \prod_{\text{edge u}\to\text{v}} P_{u\to v}(t_{uv}) \right] \} \qquad (8.12)
\end{aligned}
$$

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 [5].

We can take this approach because of the tree likelihood properties in the Markov's model:

Additivity - $\prod_{x\to y}(t+s) = \sum_y P_{x\to y}(t)P_{y\to z}(s)$

Reversibility- $P(x)P_{x\to y}(t) = P(y)P_{y\to x}(t)$

8.4.2 Efficient Likelihood Calculation

Efficient likelihood calculating is done using dynamic programming. For a character j, denote:

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

$C_j(x,v)$ 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:

$$C_j(x,v) = \begin{cases} 1 & v_j = x \\ 0 & \text{otherwise} \end{cases} \qquad (8.13)$$

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

$$C_j(x,v) = \left[\sum_y C_j(y,u) \cdot P_{x\to y}(t_{vu}) \right] \cdot \left[\sum_y C_j(y,w) \cdot P_{x\to y}(t_{vw}) \right] \qquad (8.14)$$

- The final solution is:

$$L = \prod_{j=1}^{m} \left[\sum_{x} C_j(x, root) \cdot P(x) \right] \tag{8.15}$$

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 $O(n \cdot m \cdot k^2)$.

8.4.3 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 8.19), then we get:

$$\log L = \sum_{j=1}^{m} \log \left[\sum_{x,y} P(x) \cdot C_j^v(x, r) \cdot P_{x \to y}(t_{rv}) \cdot C_j^r(y, v) \right] \tag{8.16}$$

Here $C_j^u(x, y)$ 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. 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.

8.5 Phylogenetics related resources on the Internet

- List of phylogenetic resources –
  http://www.ucmp.berkeley.edu/subway/phylogen.html.

- List of phylogeny software packages available on the web –
  http://evolution.genetics.washington.edu/phylip/software.html.

- Detailed information on phylogentic algorithms – http://www.icp.ucl.ac.be/~opperd/private/phylogeny.html#anchor2824426.
  - UPGMA – http://www.icp.ucl.ac.be/~opperd/private/upgma.html.
  - Neighbor-joining – http://www.icp.ucl.ac.be/~opperd/private/neighbor.html.
  - Parsimony – http://www.icp.ucl.ac.be/~opperd/private/parsimony.html.
- Database of phylogenetic knowledge – http://herbaria.harvard.edu/treebase/
- Additional figures, scribes and information – http://www.cs.washington.edu/education/courses/590bi/98wi/

Bibliography

[1] Phylogeny in a nutshell. `http://paleobio.org/Ed/nutshell/nutshell.html`.

[2] H. L. Bodlaender, M. R. Fellows, and T. J. Warnow. Two strikes against perfect phylogeny. In *Proc. 19th.* Springer, 1992.

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

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

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

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

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

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

[9] L. R. Foulds and R. L. Graham. The Steiner problem in phylogeny is NP-complete. *Advances in Applied Mathematics*, 3:43–49, 1982.

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

[11] D. Gusfield. The steiner tree problem in phylogeny. Technical Report 334, Yale University, Computer Science Dep., 1984.

[12] Dan. Gusfield. *Algorithms on Strings, Trees and Sequences.* Cambridge University Press, 1997.

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

[14] T. H. Jukes and C. Cantor. *Mammalian Protein Metabolism*, pages 21–132. Academic Press, New York, 1969.

[15] S. Kannan and T. Warnow. Inferring evolutionary history from dna sequences. *SIAM J. Comput.*, 23:713–737, 1994.

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

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

[18] 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.

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

[20] A. Krogh G. Mitchison R. Durbin, S. Eddy. *Biological Sequence Analysis*, chapter 7, pages 160–191. Cambridge University Press, Cambridge, United Kingdom, 1998.

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

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

[23] M. Steel. The complexity of reconstructing trees from qualitative characters and subtrees. *J. of Classification*, 9:91–116, 1992.

[24] A. Wilson and R. Cann. The recent african genesis of humans. *Scientific American*, April, 1992.

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