

Lecture #5

Now that we have at least a little bit of familiarity with programming Perl and R and working in a Unix environment, we are going to move a bit faster with bioinformatics content. In this lecture we will discuss the basics of phylogenetics and multiple sequence alignment. While we will not delve too deeply into any one of these areas, we will try to give you the quantitative flavor of the area. The goal here is to reacquaint you with some of the building blocks of genomics.

1 Phylogeny

1.1 Motivation

Phylogeny is the study of the evolutionary relationship *between* organisms. It is a macro-scale variant of Genealogy, which is the study of the evolutionary relationships *within* organisms, and which we will return to in the population genomics unit later in the quarter. You might think that something as fundamental as the family tree of life would have been settled long ago, and that further research in these fields would just be unimaginatively adding the next epsilon. After all, it is unlikely that someone is going to come up with a phylogenetic tree tomorrow which shows that humans are more closely related to *E. coli* than to chimpanzees.

This view, though understandable, is not entirely correct. Quantitative, molecular phylogenetics is actually a booming research area, for several reasons.

1. *Gene trees are not the same as species trees.* In any organism, just as any given individual may have a height which differs from the population average, so too any individual gene may have an uncharacteristic pattern of evolution which differs from that of the organism as a whole. Sampling variance occurs when the random variables are trees just as it does when the random variables are scalars. Individual gene trees may differ from the species tree because of strong selection, because the gene lies in a region of the genome which readily intercalates with environmental mutagens, or because of genetic drift¹.
2. *Horizontal gene transfer (HGT).* In microbes this picture is further complicated by the relatively recent discovery of large scale horizontal gene transfer, which was apparent after the boom in microbial genome sequencing in the 1990s (a boom that shows no signs of slowing). Microbes are known to exchange entire islands of modular genetic elements which confer functions on their host. For example, the difference between the harmless *E. coli* K12 and the diarrhea causing *E. coli* O157 boils down to a number of insertions of pathogenicity islands (Figure 1).

¹we will treat this in more detail later in the quarter.

have good digital encodings (like sequences and structures) can now be relatively easily propagated from organism to organism: we can characterize a protein in *Drosophila* and, if the sequence is mostly the same, get some clue as to what that protein does in other insects or possibly even in humans or yeast. However, many other objects (like tissues or behavioral patterns) are not yet easily “aligned” between organisms. In the future, it is possible that techniques like [network alignment](#) may partially address this problem by comparing network models of organs and higher order processes between species. Until then, though, we must simply resort to phylogenetic reasoning and reason that because (say) plant X is phylogenetically equidistant between plant Y and plant Z, that it will exhibit characteristics similar to those of both plants.

This is necessarily an incomplete list, but should give you some appreciation of why phylogenetics remains important and an active area of research.

1.2 Mathematical Basics

1.2.1 Distance Based Methods (UPGMA & Neighbor Joining)

Phylogenetics is a deep subject which could occupy us for an entire quarter, but we will be primarily considered with applications, which usually revolve around tree building. A basic concern in functional genomics is to produce reasonable (not necessarily optimal) trees to relate species, sequences, or other biological objects. Such trees are used as [guide trees](#) for multiple alignment or as tools in their own right for inferring interactions between protein families via [coevolution](#).

One of the most intuitive ways to build such trees is by calculating a distance between the sequences in a set and then clustering this distance matrix. Consider the example in [Figure 2](#). Here we have used the Jukes-Cantor distance² between nucleotide sequences and then applied the unweighted pair means group average (UPGMA) method to group sequences on the basis of their distance matrix. This method is conceptually simple, but a little bit tricky to execute manually as there are a few halves and subtractions to worry about. Given some distance measure³ $d(p, q)$ between two sequences p and q , we define the distance between clusters C_i and C_j to be:

$$d(C_i, C_j) = \frac{1}{|C_i||C_j|} \sum_{p \in C_i, q \in C_j} d(p, q)$$

This formula is slightly misleading, as we will see. It is this is just the average distance between pairs of sequences from each cluster, where $|C_i|$ and $|C_j|$ denote the number of sequences in each set. The clustering procedure occurs as follows:

1. Initialization

- (a) Assign each sequence i to its own cluster C_i
- (b) Define one leaf of the tree T for each sequence and set its height at zero

2. Iteration

²Just accept this on faith for now if you haven’t seen it before; we will motivate it later in the quarter.

³The following discussion is adapted from Durbin *et al.*, Biological Sequence Analysis.

- (a) Determine the two clusters i, j for which d_{ij} is minimal (choosing randomly if there is no unique min)
- (b) Define a new cluster $C_k = C_i \cup C_j$, and define d_{kl} by the equation above.
- (c) Define a node k with daughter nodes i and j and place it at height $d_{ij}/2$ (**this is important!**)
- (d) Add k to the current clusters and remove i and j

3. Termination

- (a) When only two clusters remain, place the root at height $d_{ij}/2$

In the example, we start by computing the distance matrix between the sequences. We then randomly choose $i = 1, j = 3$ to collapse; we could also have started with 2, 4. Now, the tricky bit is in updating the distance matrix. We need to take into account the fact that the 1, 3 node has moved up towards the common ancestor by $d_{13}/2 = .05$. Hence the distance to the 2 node and the 4 node will be reduced by .05, as can also be seen from the arrow pointing to the position of the 13 node on the right hand side of the figure. So we have:

$$d(13, 2) = \frac{d(1, 2) + d(3, 2) - d(1, 3)}{2} = \frac{.3 + .2 - .1}{2} = .2$$

This is the formula above for $d(C_i, C_j)$ is only really accurate after you move up leaves which you have just grouped by $d_{ij}/2$ and compute all quantities relative to this move. After this point the remaining steps are quite simple as illustrated in the figure.

The UPGMA method generates a so-called *ultrametric tree* in which the evolutionary distance between every leaf node (species) and the root (common ancestor) is the same. This is only true if each branch experiences the same rate of evolution. However, rate variation is usually observed with real data as species and sequences are often subject to different selection pressures. So instead of UPGMA we use a different distance-based algorithm, the neighbor joining method, which makes fewer assumptions in practice. Rather than assuming a molecular clock, we simply seek to collapse nodes which have the same parent node and build up the tree in this manner.

So, suppose we have the same setup with a distance matrix d . Suppose we can find a pair of nodes i, j which neighbor (have the same parent node). Then we can collapse them together into a new node k and update the distance matrix with

$$d_{km} = \frac{d_{im} + d_{jm} - d_{ij}}{2}$$

This is the same formula which we used to update distances with UPGMA. The key difference with neighbor joining is that we do not simply pick the smallest element in the distance matrix each time. A straightforward greedy look at the distance matrix can sometimes artificially collapse leaves which do not have the same parent (Figure 3). The trick is to define an auxiliary matrix D on which the greedy approach works. We have:

$$D_{ij} = d_{ij} - (r_i + r_j)$$

where

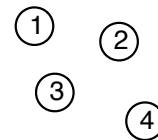
$$r_i = \frac{1}{|L| - 2} \sum_{k \in L} d_{ik}$$

a Given an (aligned) set of sequences...

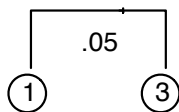
1 GATTACAGAT
 2 CACCACAGAT
 3 GACTACAGAT
 4 CACGACAGAT

b Calculate a distance matrix (e.g. frac. diff.)

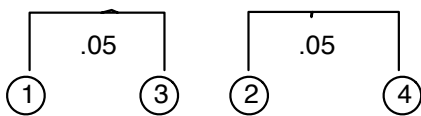
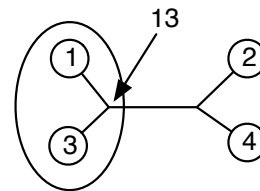
	1	2	3	4
1				
2	.3			
3	.1	.2		
4	.3	.1	.2	



c Successively apply UPGMA to objects



	13	2	4
13			
2	.2		
4	.2	.1	



	13	24
13		
24	.15	

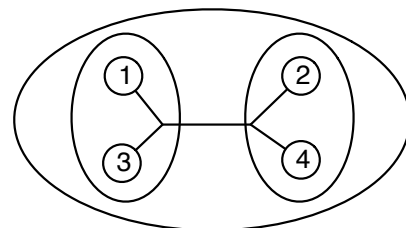
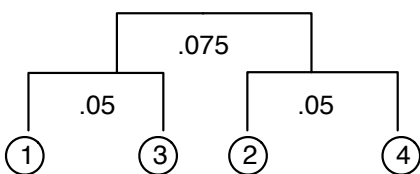
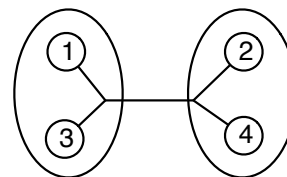


Figure 2 | An example of calculating a phylogenetic tree via UPGMA

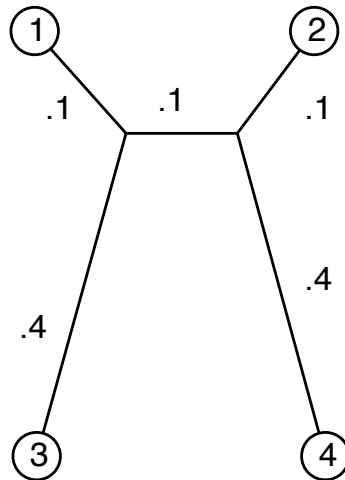


Figure 3 | An example of why the neighbor joining algorithm is somewhat trickier than picking the minimum element in the pairwise distance matrix. Note that 1 and 3 are neighbors (same parent node), but that 1 and 2 have minimum distance and different parent nodes. This problem was dubbed [long branch attraction](#) by Felsenstein and illustrates the problem with the UPGMA assumption of rate invariance in each lineage.

and $|L|$ is the total number of leaves. Thus $(r_i + r_j)$ gives the average distance of the nodes i and j to all $|L| - 2$ other leaves in the tree. This correction allows us to apply the greedy approach and join the leaves for which D_{ij} is minimal, which are guaranteed to be neighboring leaves.

Here is the NJ algorithm in full, from Durbin and Eddy's Biological Sequence Analysis:

1. Initialization
 - (a) Define T to be the set of leaf nodes, one for each given sequence, and put $L = T$
2. Iteration
 - (a) Pick a pair i, j in L for which D_{ij} is minimal
 - (b) Define a new node k and set $d_{km} = .5(d_{im} + d_{jm} - d_{ij})$ for all $m \in L$.
 - (c) Add k to T with edges of length $d_{ik} = .5(d_{ij} + r_i - r_j)$, $d_{jk} = d_{ij} - d_{ik}$, joining k to i and j respectively.
 - (d) Remove i and j from L and add k .
3. Termination
 - (a) When L consists of two leaves i and j , add the remaining edge between i and j , with length d_{ij} .

If you want to see another worked example, see this set of notes by [Terry Speed](#).

1.2.2 Character Based Methods (Minimum Parsimony and Maximum Likelihood)

There are two other major methods worthy of mention for building trees: the method of minimum parsimony (a discrete algorithm) and the method of maximum likelihood. These

algorithms are primarily useful to specialists and/or people who do not need to build a lot of trees, as they are currently far more computationally intensive. Maximum parsimony is probably the least desirable as it results in a combinatorial optimization problem and does not include any explicit biological modeling of evolution. In this respect it is similar to the use of edit distance for sequence alignment. Moreover, parsimony is not [statistically consistent](#) in the limit of infinite data.

[Maximum likelihood](#) is better in that it provides an asymptotically consistent and offers the possibility of explicitly incorporating biology⁴ into the modeling process. Suppose we have an ancestral sequence y and a descendant sequence x which evolved in a time t . Then if we knew $P(x|y, t)$, we would have the conditional probability distribution over descendant sequences given the ancestor. Models of sequence evolution exist which can give us such distributions, at least for simple ungapped alignments, as we will see shortly. We can also, in theory, invert such models to maximize the likelihood of a vector of descendant sequences x_1, \dots, x_L observed at the present day for a tree topology T and a vector of branch lengths \vec{t} . However, a straightforward approach to this optimization problem requires a combinatorial search over tree topologies followed by a search over all possible valid branch lengths. Sampling methods can be used but the problem is extremely computationally intensive and still unsuitable for scaling up. In general, the neighbor joining algorithm produces decent albeit suboptimal trees in a time which scales as $O(L^2)$; as such it is usually the method of choice unless you are really concerned with an accurate tree.

1.3 Further Directions

Booming areas in modern phylogenetics include (1) creating genome trees rather than single gene or 16S rRNA trees, and (2) building phylogenetic webs rather than trees to accommodate the reality of HGT.

1. Papers [genome trees](#) and the [SHOT server](#) for whole genome phylogeny of microbes.
2. ISMB tutorial on [phylogenetic networks](#).

1.4 Software

1.4.1 PHYLIP

[PHYLIP](#), by Joe Felsenstein, is still probably the best single purpose piece of phylogeny software available. However, it has a *terrible* user interface for power users. The problem with PHYLIP's programs is that – unlike all standard GNU Unix commands – they do not accept command line arguments with one or two dash flags. This is an annoyance as one will frequently wish to run PHYLIP in a loop, e.g. to build up a number of trees from bootstrapped sequence alignments.

To run PHYLIP with command line arguments you can use my wrapper script, [phylip-wrapper.pl](#) After installing PHYLIP and making sure that the `protdist` function is in your path, you can look at the source code and download [CDD1008_N16.faa.out](#). Then run the

⁴With the NJ/distance based methods, the biology comes about in the choice of distance. I used fraction of differing characters above, but in practice one would use one of the several nucleotide distance measures like Jukes-Cantor and more sophisticated relatives, which we will look at later in the quarter.

command:

```
phylip_wrapper.pl protdist CDD1008_N16.faa.out CDD1008_N16.faa.out.dist P 2 Y
```

which will calculate protein distances between the rows of the multiple alignment using the [PMB protein distance](#) in phylip. After the program name, the input filename, and the output file are an arbitrary number of arguments which let you control the flow of the PHYLIP program. The inputs are P (toggles PMB distances), 2 (which turns off feedback), and Y (which ends option input). See [PHYLIP's documentation](#) (ctrl-f for “under control of a command file”) and the source of the perl script for more.

2 Multiple Alignment

2.1 Motivation

So far we have spoken about pairwise sequence comparison. However, we frequently wish to compare more than one sequence at a time. Here is a partial list of occasions in which this occurs:

1. Given N proteins which have all been found to be sequence similar to each other, and which are part of a protein family, calculate a multiple alignment which describes the protein family and which makes the domains and active sites which characterize the protein family “pop out” as columns of extreme conservation.
2. Given N upstream regions thought to contain a regulatory motif of interest, calculate a multiple alignment of the noncoding regions to make the motif “pop out”.
3. Given N genomes, calculate a multiple genome alignment to identify evolutionarily conserved (and hence likely functional) regions in the human genome.

2.2 Mathematical Basics

We first note that the dynamic programming algorithm described for pairwise sequence alignment is exponential in the number of sequences, requiring a score cube for three sequences and becoming essentially intractable after that point.

We then note that there are several ways in which a multiple alignment can be represented:

```

a  MAYPMQL-FQ
      MAH--QLGF-
      MAN-SQLGFQ

b  1234567089
      1230045670
      1230456789

c  0000000100
      0001200001
      0001000000

d  (7,1;10,0)
      (3,2;9,1)
      (3,1;10,0)

e  (7,-1,2)
      (3,-2,4,1)
      (3,-1,6)

```

Figure 4 | Five alternate representations of a multiple alignment. (a) Ordinary alignment matrix. (b) Index matrix, with 0's for gaps. (c) Gap state matrix. (d) Gap position and length. (e) Numbers of contiguous non-null (positive) and null characters (negative). Some of these representations are particularly useful for updating a multiple alignment without incurring significant memory costs. Adapted from Aluru, Handbook of Computational Molecular Biology.

2.2.1 ClustalW and Progressive Alignment

Because exact dynamic programming is too slow, heuristics were developed for multiple alignment. One heuristic is to use a sum-of-pairs score as an optimality criterion for the multiple alignment. That is, given N sequences x_1, \dots, x_N and a set of pairwise alignment scores $S(x_i, x_j)$, simply define the multiple alignment score to be:

$$S = \sum_{i=1}^N \sum_{j=i+1}^N S(x_i, x_j)$$

This score function is problematic as we will see below. However, it is the basis of many aligners. To find a multiple alignment which approximately optimizes this function another set of heuristics is used. The first widely successful multiple aligner was ClustalW, which used reduced multiple alignment to a set of pairwise alignment problems by using two heuristics: progressive alignment of sequences which were close phylogenetic relatives and the concept of profile-profile alignment.

1. An approximate phylogenetic tree (a “guide tree”) is calculated from the data
2. Starting from the most closely related groups, sequences are combined going up the tree to yield alignments

One might ask how this guide tree is constructed in the absence of a multiple alignment. The answer is that it is a hack based upon (likely inconsistent) distances calculated from pairwise alignments. Here is the full pseudocode for [ClustalW](#):

1. Determine all pairwise alignments between sequences and the degree of similarity between them.
2. Construct a similarity tree.
3. Combine the alignments from 1 in the order specified in 2 using the rule “once a gap always a gap”

In more detail: In stage 1, clustalw uses a pairwise alignment to compute pairwise alignments. Using the alignments from 1.1 it computes a distance. This distance is commonly calculated by looking at the non-gapped positions and count the number of mismatches between the two sequences. Then divide this value by the number of non-gapped pairs to calculate the distance. Once all distances for all pairs are calculated they go into a matrix. This follows on in stage 2.

In stage 2, using the matrix from stage 1. and Neighbor-Joining, Clustalw constructs the similarity tree. The root is placed in the middle of the longest chain of consecutive edges. In stage 3, alignments are then combined alignments, starting from the closest related groups (going from the tips of the tree towards the root).

2.3 Further Directions and Software

While ClustalW was and continues to be widely used, it has largely been superseded by a new generation of [multiple alignment](#) algorithms. One of these, PROBCONS, is based on the concept of consistency:

A methodology that has been successfully used as an improvement of progressive alignment based on the SP formulation is consistency based scoring. Given three sequences, A, B and C, the pairwise alignments A-B and B-C imply an alignment of A and C that may be different from the directly computed A-C alignment. This motivates a search for an MSA that maximizes agreement (“consistency”) with a set of pairwise alignments computed for the input sequences.

Consistency based aligners like PROBCONS and aligners with different techniques for profile/profile alignment like MUSCLE have made a [huge impact](#) in recent years. In fact, even though the field is ancient in bioinformatics years, multiple alignment is still a topic of active research. For example, current alignment algorithms for proteins are not very good at handling situations in which domains are shuffled. New representations of alignments in terms of [graphs](#) provide richer representations which may help to handle this problem. In other areas the progress to be made is less incremental. Multiple sequence alignments of non-coding regions, of ncRNAs, and of genomes are still active areas of research.

The best program for general purpose use is probably [MUSCLE](#), programmed by Robert Edgar. NCBI uses MUSCLE to compute its multiple alignments and it is a serious piece of production code. For people who want a highly accurate alignment and don’t mind a lot of tuning and fiddling, then you can try [MAFFT](#); alternatively, if you want an accurate alignment without heuristics and don’t mind waiting a bit, [PROBCONS](#) by Tom Do is optimal. Finally, this [review of multiple alignment](#) by the author of MUSCLE is the best recent reference on the topic.