Design and Analysis of Algorithms CSCI 5454
Lecture 11 28 February 2011
Prof. Aaron Clauset

1 Computational Biology

Computational biology is the application of computational tools (algorithms) to automatically extract information and/or patterns directly from biological data. Today, that biological data is most commonly derived from DNA sequences. To understand the basics of the algorithms, we don't need to know very much biology. Of course, the more biology we learn, the more clearly we see that underlying data can be quite messy, that the assumptions we make to design our algorithms are often not biologically realistic, or that we simply don't know as much about either the data-generating or the evaluation process as we like to think.

Here's a quick sketch of the basics: a DNA sequence is a string of characters $x_1, x_2, x_3, \ldots x_\ell$, each of which is drawn from the alphabet $x_i \in \Sigma = \{A, T, G, C\}$. In an organism, DNA sequences are "double-stranded" meaning that each sequence x is paired with a complementary sequence x', with $x'_i \in \Sigma$ but where an A in x is paired with a T in x', a T with A, a G with C and a C with G. Thus, the string $x =$ `GATTACA` would be paired with $x' =$ `CTAATGA`.

The data-generating process for naturally derived DNA sequences is evolution, which has two main components. The first is "variation with descent," in which some set of processes generate differences in the DNA sequences of an organism and its offspring. Perhaps the most commonly known of these are point mutations, in which single characters change, e.g., `GATTACA`→`CATTACA`. Sequences can also experience deletions (`GATTAC`), duplications (`GATTATTACA`), insertions (`GACATTACA`) or reversals (`GATACAT`) of both single characters or entire subsequences. The second part of evolution is "selection," in which some set of processes prevent the replication of some sequences versus others. This aspect of evolution is sometimes (erroneously) referred to as "survival of the fittest." The real story is more complicated: there are many reasons that lead to some sequences being passed on to offspring despite the fact that they confer little, no or even negative "selective advantage". The good news is that evolution works so long as, on average, in large populations and over long timescales, the genome as a whole confer a net selective advantage. (This is a fairly weak condition, which is partly what makes evolution such a powerful strategy for finding relatively good solutions to relatively hard problems, and it is what underlies the way genetic algorithms work.)

With this background established, we can talk about one of the main areas of computational biology: inferring the history of genetic diversification, given a set of character vectors.[1]

[1]The two main areas in computational biology are sequence alignment and phylogenetic tree reconstruction. Lately, there has been additional emphasis on applying graph theory ideas and techniques to understanding biological

2 Phylogenetic Trees

If we could watch evolution progress forward in time, we would see it produce genuinely new species by taking a single existing species and splitting into two distinct genetic groups (this process is called "speciation", and can happen quickly or slowly, depending on the mechanism, where the timescales here are somewhere between hundreds of years to hundreds of thousands of years; that is, evolution can be very very slow). Even without active selection, these groups will "drift" apart genetically over time (a process called "neutral evolution") so long as they do not exchange genetic material sufficiently quickly.[2]

If we imagine evolution running backwards in time, however, we would see separate species "merge." The character vector of the merged species is interpreted as the potential character vector of the common ancestor of the two descendent species. A *phylogenetic tree* is the set of all such mergers on a set of n species.[3] That is, if we are given a set of n character vectors $\{X_i\}$, we place these on the leaves of a tree. A phylogenetic tree is a tree that connects these sequences. Internal nodes of this tree are inferred common ancestors. For instance, Figure 1 shows an inferred phylogenetic tree on the three major domains of life: bacteria, archaea and eucarya.

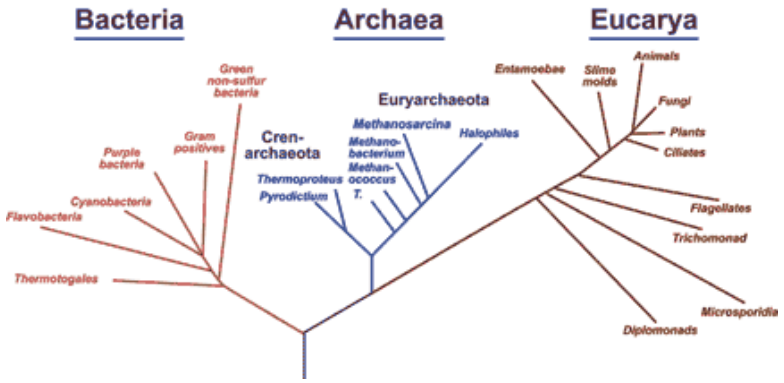


Figure 1: A phylogenetic tree on bacteria, archaea and eucarya. (From Carl Woese's website.)

networks.

[2]Sexual reproduction is one process by which a population of organisms can maintain a kind of "coherence" in the "gene space."

[3]A tree structure assumes that there are no mechanisms for genetic exchange between "species," which is not exactly accurate. In fact, there are many such mechanisms: viruses can transport DNA between species, the "horizontal gene transfer" mechanism allows bacteria and archaea to exchange DNA, and "hybridization" is a method by which species like plants and animals can exchange DNA by producing a descendant organism with genes drawn from two different parent species. (And, there are a few others, still.) Within a population, sexual reproduction serves the same function. Thus, the tree structure is simply a rough approximation of the large-scale genetic history of a set of species. A more general approach would use directed acyclic graphs.

You can see that many of the species most familiar to us (plants, animals, fungi, etc.) are in the eucarya group. This picture, reproduced from Carl Woese's lab at UIUC, shows one of the better inferred phylogenies for the base of the "tree of life." Here's a slightly more artistic (but still data-derived) phylogeny from the Tree of Life project, which shows more structure in the eucarya.

Figure 2: An artistic "tree of life," showing the eucarya in particular. (From `http://tolweb.org/`.)

The goal of phylogenetic tree reconstruction is to build accurate trees that capture the true biological relatedness of organisms. The input data can be any set of "characters", e.g., a DNA sequence (common for many molecular biology studies) or a set of morphological features (common in paleontology, where DNA is not typically available).[4] Given these inputs, the goal is to produce the "best" tree that connects these leaves, where "best" is determined by some kind of score function that encodes our scientific beliefs and methodological approach. There are many different approaches; here, we'll delve into the *maximum parsimony* and distance-minimization approaches. We won't cover *maximum likelihood* or probabilistic approaches, which are increasingly popular.

[4]Paleontologists identify sets of body structures or patterns or measurements for each species, and code these as a morphometric vector $\vec{\theta}$. The algorithms then work precisely the same, although typically, because the model of feature variation is not known, paleontologists work with maximum parsimony or distance-based approaches instead of the probabilistic methods favored for DNA evolution.

2.1 The maximum parsimony principle

For simplicity, let's consider sequences drawn from a binary alphabet $\Sigma_{01} = \{0, 1\}$. We are given n such binary sequences, each of length ℓ. The maximum parsimony tree on these n sequences is the one that minimizes the number of character mismatches between ancestors and descendants, which can be counted using the pairwise Hamming distance[5] of the ancestor and descendent strings:

$$d_{\mathrm{H}}(X, Y) = \sum_{i=1}^{\ell} \mathrm{XOR}(x_i, y_i) \ .$$

2.1.1 Big parsimony

The general problem of finding a maximum parsimony tree is then reduced to solving this equation:

$$T_{\mathrm{MP}} = \underset{T \in \mathcal{T}}{\operatorname{argmin}} \left(\sum_{X \in T} d(X, p(X)) \right) \ .$$

That is, we want the tree T out of all trees $\mathcal{T}$ (with n leaves) that minimizes the interior sum; the interior sum ranges over all pairs of descendants X and ancestors $p(X)$ within a given tree T, and requires that each interior node be labeled with the inferred ancestral sequence. The tree that minimizes this sum is the maximum parsimony tree T_{MP}. Note that T_{MP} is not necessarily unique. (What criteria would be necessary to make it unique?) Finding the maximum parsimony tree is sometimes called the "big parsimony" problem and is known to be HP-hard in general (via reduction to vertex cover). Existing techniques largely focus on sampling or searching the entire space of $\mathcal{T}$ trees, which contains $(2n - 3)!!$ candidate phylogenies,[6] for good trees.

2.1.2 Little parsimony

We can, however, solve a more restricted problem, which is sometimes called the "little parsimony" problem, in polynomial time. This task is, given a candidate tree T on the n sequences, can we derive the ancestral sequences that maximize the parsimony of T.

Let's work a small example. Suppose we are given the strings $\{00011\}, \{11000\}, \{10001\}, \{00110\}$. Below are two candidate trees along with potential ancestral states. In each, the given sequences are given are shown in orange while the potential ancestral states are shown in green, and the ancestor-descendant distance shown as the edge label in the tree. The *weight* or *parsimony score* of a tree is simply the sum of the edge distances; the left tree has weight $d_{\mathrm{H}} = 8$ while the right tree has weight $d_{\mathrm{H}} = 6$, so the right-hand tree is the more parsimonious explanation.

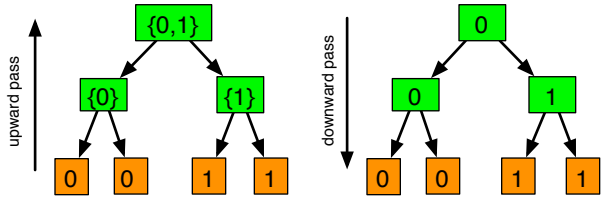
[5] In general, other measures of distance could be used, e.g., Euclidean, Mahalanobis, etc.

[6] The double factorial means this: $n!! = n(n-2)(n-4)...$; you can derive the $(2n-3)!!$ result for unrooted phylogenies, or $(2n-5)!!$ for rooted phylogenies, via induction.

To infer the maximum parsimony labeling of the ancestral states, we can use a dynamical programming algorithm (due to Fitch, 1971). Fitch's algorithm takes $O(n\ell)$ time in total. It treats each character in the sequence independently, each of which takes $O(n)$ time. By running the basic algorithm on each letter, it derives full ancestral sequences for each internal tree node.

We won't go into the algorithm in detail here (since we haven't covered dynamic programming), but here's the essence: starting at the leaves, it works up toward the root assigning each internal node v a set of letters: if the sets of v's children overlap, then we make v's set their intersection, otherwise, we make it their union. The algorithm then assigns a character to each tree node, starting at the root and progressing to the leaves: if v is the root, it chooses a character arbitrarily from v's set and then recurses on v's children; otherwise, if v's set contains the character assigned to its parent $p(v)$, then the algorithm assigns that same character to v, while if the parent character is not in v's set, the algorithm chooses arbitrarily from within v's set and recurses.

Here's what happens if we run Fitch's algorithm on the first character of our example: the left-hand tree shows the upward set-building pass and the right-hand tree shows the downward node-labeling pass. (Is the right-hand tree in the previous figure a maximum parsimony labeling on T? How many such labelings are there?)[7]



[7]Recall that a sequence of length ℓ can be considered a vector in a ℓ-dimensional hypercube. Choosing the ancestral states and the phylogeny can then be seen as the problem of finding a tree within the hypercube, with leaves located at the given n sequences, that has minimum length.

2.2 Distance methods

Suppose that our characters are not discrete but are instead drawn from some continuous space, for instance, instead of looking at DNA sequences, suppose we look at some set of morphological (or even linguistic) features of species.[8] This allows us to represent the state of a species as a vector. As a result, algorithms exist that can derive both the minimum weight tree and the ancestral states in polynomial time.

Most of these algorithms take a greedy approach by building the tree up by repeatedly merging the pair of states with minimum distance, much like we did in Lecture 9. Because evolution should generate only a small number of changes from ancestor to descendants, and because we assume these changes are largely independent, given two vectors x, y, these algorithms typically assume that the ancestral state splits the distance between the two descendent states, i.e., the inferred ancestral state is the mid-point of the descendants.

In general, because such an algorithm reduces the number of entities by one each time we iterate, the algorithm will terminate after $n - 1$ steps. What distinguishes different distance-based algorithms is mainly their choice of distance measure, how it chooses which pair of entities to agglomerate, how it chooses the ancestral state, and how it manages all the data.

Most such algorithms maintain a pairwise distance matrix $S_{ij} = d(i, j)$. Given that we choose to replace some pair i, j with their merged state z, we then only need to update $4(n - 1)$ entries in the distance matrix S: we need to delete each of the entries in the ith row and column, and replace each entry in the jth row and column with $d(z, k)$ for all k. Here's pseudocode for a generic distance-based algorithm:

```
Generic-GreedyAgglomerative(X) {
// X = set of vectors
   S = {}
   for all pairs x,y in X {
      S.add(d(x,y),(x,y))
   }
   while S non-empty {
      (a,b) = S.returnElement()
      for each x in X {
         S.remove(d(a,x))
         S.remove(d(b,x))
```

[8] We can always apply a continuous-character algorithm to discrete-character data, but using a continuous-version of the `merge(a,b)` function below will produce continuous-character ancestors. A discrete version of `merge(a,b)` would fix this problem, allowing distance-based methods to be applied to discrete-character data.

```
      }
      remove a and b from X
      c = merge(a,b)
      for all x in X {
         S.add(d(c,x),(c,x))
      }
      add c to X
   }
}
```

If we choose which pair to merge in a greedy fashion, much like we did with the graph clustering algorithm in Lecture 9, then we can replace S with a min-heap data structure. It takes $O(\ell)$ time to compute the distance between some pair of sequences; thus, it takes $O(n^2\ell)$ time to build the initial heap (if we use something like a Fibonacci heap; if we use a binary heap, it takes $O(n^2\ell \log n)$). If we assume that the `merge(a,b)` step takes no more than $O(\ell)$ time, then each pass through the while loop takes $O(n\ell)$ time (do you see why?), and there are $n-1$ passes; so the greedy agglomerative algorithm runs in $O(n^2\ell)$ time overall.

The particular choice of `merge(a,b)` functions determines which agglomerative tree the algorithm constructs; if we choose the state that splits the difference between a, b, i.e., the equidistant point, then this algorithm is logically equivalent to a single-linkage hierarchical clustering algorithm, which is a popular (but not a very good[9]) technique for clustering spatial data. (The "split-the-difference" choice is not guaranteed to find the ancestral states that minimize tree weight; do you see why?)

2.2.1 UPGMA

A popular distance method is the Unweighted Pair Group Method Using Arithmetic Mean (UPGMA), named by Sokal and Michener (1958), which uses the arithmetic mean as the `merge(a,b)` function. Because of the averaging it does when merging entities, it implicitly assumes a uniformly constant molecular clock, which may not be desirable in some applications.

2.2.2 Neighbor joining

Another variation is the neighbor joining method, which yields a minimum-evolution tree (this is like a maximum parsimony tree, but on a general metric space) that does not assume a uniformly constant molecular clock. Instead of choosing a pair of entities based on their minimum distance,

[9]There are many other variations on such hierarchical clustering algorithms, most of which you would encounter in a data mining course. They should be treated with caution however as they are typically not based on any strong statistical or mathematical principles, as far as I can tell, and can perform badly under entirely reasonable conditions.

neighbor-joining instead uses the function

$$Q(x,y) = d(x,y) - \frac{1}{n-2}\left(\sum_{i \neq x,y} d(x,i) + \sum_{i \neq x,y} d(y,i)\right)$$

but still chooses the pair with minimum $Q(x,y)$. Intuitively, Q is the distance $d(x,y)$ minus the sum of the average distances from x (and y) to the rest of leaves. Neighbor-joining operates only on pairwise distances; once a pair x, y is chosen, the distance matrix is updated like so: let z denote the new "merged" entity, then for each neighbor i of either x or y, let

$$d(i,z) = \frac{1}{2}\left(d(x,i) + d(y,i) - d(x,y)\right) \quad .$$

3 For Next Time

1. More phylogenetic trees, the split-equivalence theorem and "consensus" trees