CSCI 4180

Phylogenetics & Beyond

Blouin

Alex Safatli

Tuesday, October 30, 2012

Contents

Genomics and Biological Computing

Interpreting Trees and Distance Methods

Where did we come from? Where are we going? Why are we doing this? Bioinformatics will not tackle the latter two questions, but the first question can be looked at from a phylogenetic perspective.

The "Out of Africa" hypothesis infers that human migration and dispersal came about from Africa around 100,000 years ago. Figuring out how this happened involves using mitochondrial DNA. If we were to use the human genome as our source of data, we have duplicate copies of our genome and chromosome crossovers and recombinations can complicate matters. A mixed history can result and a messy story will come out. So it is easier to use the mitochondrial genome.

Recall that the mitochondria is an endosymbiont, has a circular genome and no introns, is an energy power plant, and is inherited from the mother (in animals). A phylogeny of mitochondrial DNA, in principle, represents the maternal line history. See slide for an evolutionary tree of humans based on complete mitochondrial genome sequences.

Time and evolutionary distance are related if you assume the rate of evolution is constant over time. Longer branches can infer a great deal of time or extremely selective pressure on evolution (such as domesticated pets). Note that these trees *are inferred from genes and are purely based on sequence characters.* The place where the taxon Chimp attaches to the human tree suggests the location of the root of all humans. This is not yet known. If you choose something outside of the dataset, you can pick the most reasonable location of the root.

But why is the topology of the Eurasian (top-mostern) unresolved (no resolving into subgroups)? An explosion of population happened here where people split too fast. No time will occur here where evolution can happen to make any specific section of the tree unique and distinct from the other. These are called *stars* and typically infer mass migration in a short time frame.

The other hypothesis is the "Multiregional" hypothesis, but this is not supported by mtDNA sequence data. This is a hypothesis where Neanderthals actually went into Europe for a great deal of time, and when humans came in, they left.

To make a "tree of life", you require common genes. These may include basic cell metabolism or a small subunit of ribosomal RNA genes. A phylogeny from this can create the kind of topology seen on slide. Note this is biased towards eukaryotes. However, we cannot root the tree of life. There is really nothing outside of the data set. Viruses would not work as they came out of life and steal common genes as they are parasites.

Phylogeny relates to a clustering problem except that the clusters must be a representation of the process of differentiation over time. A non-biological example refers to manuscript evolution. There is a strong correspondences between this and genomic phylogeny. See the slides for the analogy.

Thanks to Gutenberg and his invention of the printing press, the rate at which manuscripts are evolving have decreased by many orders of magnitude (next to 0, actually).

Distance-Based Clustering

There are two variations on simple clustering: **UPGMA** and **neighbour joining**. Algorithmic methods are fast. UPGMA assumes a constant rate of evolution throughout the evolutionary span of the dataset. This allows for an assumption that internal nodes are at the midpoint between two leaves. In a larger dataset

Distance-Based Clustering continued on next page...

with more leaves, internal nodes will begin to merge with internal nodes – clusters of leaves are merged which minimizes the average distance between members.

$$1/(A \times B) \times \sum_{x \in A} \sum_{y \in B} d(x, y)$$

The shortcoming of UPGMA is that the "molecular clock" assumption can be rejected in most cases. In the example on the slides, un-equally evolving sequences are clustering according to their rate of evolution rather than according to the history of the genes.

NJ assumes that the distances used are an exact reflection of real evolutionary distances. The neighbour-joining algorithm is applied to create a **dendrogram**. It is a deterministic algorithm that may not be the best-fitting tree, but at least puts one in the right search space. This is very similar to UPGMA except that the method of choosing close nodes differs because of the definition of "neighbourhoods". A pair is chosen which minimizes the function on the slides – grouping outliers first and then the nearest possible pairs. This guarantees to recover the true tree if the distance matrix is an exact reflection of the tree. There is, however, an issue with distances. There is a point where distance estimates actually break down, making it difficult for things that are distantly related.

Multiple Sequence Alignments

NOTE The last assignment will be posted on Moodle shortly. Will not take a long time to do; assignment will focus on using a web portal. Will be due on November 22, 2012.

Multiple sequence alignments (MSA) are the optimal alignment of N sequences together, and not just as pairs. There is value to this as they may complement each other and provide greater alignment accuracy. A phylogeneticist would appreciate if all particular columns have a common ancestor within them.

Going back to what we know: the pairwise problem involves dynamic programming where a three-step algorithm solves for the best alignment without the requirements of manual intervention. The three-way sequence case would involve three sequences A, B, and C. A third dimension would be present in the accumulated score matrix and a path would be traced as normal. The problem here would be that the time complexity would increase with each added dimension (n^3 in algorithm and memory complexity). Larger alignments may include 1000 or more sequences. Global alignments using N-W cannot be used.

Heuristics involving MSA include near-optimal methods, progressive alignment methods, profile methods, Hidden Markov Model methods, genetic algorithm methods, and tree-based methods. The progressive method is the one most commonly used.

Progressive alignment involves building up an alignment from closely related sequences to distantly related ones, using a "guide tree" and "locking" the alignment as it goes. It is a heuristic method and is not guaranteed to find the "optimal" alignment. It requires "n choose 2" pairwise alignments as a starting point.

$$N_{pairs} = (n!/(2(n-2)!))$$

The workflow involves taking an input, leading to pairwise distances, to dendrogram clustering, taking the two nearest neighbours sequences and nodes (tree), pairwise alignment, iteration with these two steps, and finally outputting the result. Therefore, the progressive alignment approach includes:

Multiple Sequence Alignments continued on next page...

1. The UPGMA Dendrogram (Phylogenetic Tree).
2. The sum of pairs scoring scheme.
3. Weighing sequences (or why it should be done).
4. Modelling gap penalties.

Neighbours in the dendrogram are aligned in one of three cases of pairwise alignment: $SeqA$ vs. $SeqB$ (using gapped dynamic programming), $SeqC$ vs. $P(AB)$ where $SeqC$ is aligned against the profile of $P(AB)$, and $P(AB)$ vs. $P(CDEF)$ where both profiles of both nodes are aligned together.

A **profile** is a way to represent many sequences as one "character". Each character is a *vector* of n frequencies (n = 4 for DNA, n = 20 in proteins). Profiles can only be acquired from aligned sequences and minimizes of the effect of sampling.

Scoring profile alignments involves using the /textbfsum of pairs (SP) where, for each aligned site s in profile A and B you retrieve the sums of all pair's scores. Any gaps cannot be removed but only more gaps will be made.

$$SP(s) = \sum_{i \in A(s)} \sum_{j \in B(s)} similarity(i, j)$$

Otherwise, the similarity will depend on the frequency of the characters and their similarity.

This approach is essentially built using pairwise alignments of profiles. "Once a gap, always a gap" means that once committed to a gap, it cannot be removed. Consequently, an error in the insertion of a gap early forces the rest of the alignment to work around it. Similarly, once a mistake is made, the best can be done is to find an alignment corresponding to a **local maximum** score where it may have looked good early on but is holding the rest of the alignment back. If removed, this leads to sub-optimal solutions. This contrasts with the **global maximum** score alignment, which is the best possible alignment.

There is usually no way to tell what is the true alignment! There is also no theoretical ultimate method to score an alignment such that the highest scoring alignment would be the true alignment.

NOTE Gaps opening and extension is not a uniform process. There is a structural basis for gap regions. It makes sense to have larger gaps spanning regions rather than larger numbers of small gaps – this is the basis for $GOP > GEP$. Furthermore, there is an implied position-specific mapping of gap penalty. See slides for more information on modelling gaps.

Structural alignments are considered to be the closest solutions to the true alignment. Manual editing is often necessary to fix obvious errors and delete ambiguous regions.

Inferring Evolution as Optimization

So far, we have gone through a very complex process. Starting with sequences, these sequences can be aligned, in a pairwise fashion, and compiled in order to build a matrix of pairwise distances. Finally, a tree can be formed; assuming perfect distance measures, neighbour joining will provide a perfect tree (this is not as true for UPGMA). However, distances are usually underestimated.

Deterministic methods of building trees from distance matrices are mostly considered as a quick heuristic for the real state-of-the-art method for inferring these solutions.

Using the trees that we infer, we can use these trees, as dendrograms, in order to perform a multiple sequence alignment.

There is a sort of function that, given an MSA and a tree, we can determine how much this tree fits the dataset. Inferring evolution would therefore become an optimization problem using this objective function where the tree is being optimized (its topology). Example methods include **parsimony** and **maximum likelihood**.

NOTE Phylogenetic trees are typically structured in a **Newick string format**. The principle of this string is fairly simple and one can convert between this and the data structure in a very straight-forward manner. A root is necessary in order to calculate an objective score, but is not necessary for the string format. Opening a parenthesis indicates getting into an internal node. Commas indicate switching to another child. Branch lengths are indicated by colons and the value. Only external nodes are labelled. Example: ((A:0.1, B:0.2):0.5, (C:0.3, D:0.4):0.0); if rooted to E.

Strategies of phylogeny are therefore comprised of:

- **Discrete Character Approaches**
  - Parsimonious Criterion
  - Model Likelihood Criterion
  - Bayesian Statistics
- **Distance-based Clustering**
  - Least-square
  - Neighbour-joining / UPGMA (implicit topology)

In parsimony, the **Fitch algorithm** involves going through a column of an MSA and infer ancestral state for each internal node. The parent of children are determined by looking at the intersection of children; if this is empty, it would instead be the union but the score is incremented by 1. The score counts the number of evolutionary events (substitutions).

Parsimony is an intuitive method that can be run manually. It assumes that everything observed in the data is connected by the most straightforward relationships. The problems with it are that it involves assuming the most parsimonious explanation is the correct one. It also assumes that all changes have a similar "cost". Therefore, the parsimony method does not seem to be designed to deal with saturation.

The *maximum likelihood criterion* is an abstraction of a collection items (sequences) where we know that all the instances in the collection are stochastically derived from a unique parent to the hierarchy. We also have a model for this stochastic process represented as a Markov process. This means we are looking for a tree (including its topology and distances) that will maximize the likelihood of the data given the Markov process. This is an optimization problem within an optimization problem. Recall the simple model for calculating likelihoods for distances – π are frequencies estimated from the data. The structure of the equation follows a postorder traversal of the tree. Because of this, we can actually calculate this in $O(n)$. Each branch must be optimized iteratively (through a line search), however.

Generally, maximum likelihood gives better solutions than parsimony.

Unlike UPGMA and NJ, the problem with this previous method is that you have to provide a topology prior to the calculation. The number of possible trees grows very quickly with number of leaves.

$$T(n) = 3 \times 5 \times 7 \times 9 \times 11 \times ... \times (2n - 3)$$

$$T(n) = ((2n-3)!/2^{n-1}(n-1)!)$$

Exploring the Topology Space

One way to explore this space is **step-wise addition** – breadth-first searching. Introduce one sequence at a time to the branch which gives the best likelihood. The disadvantage with this is that if your starting set of leaves is a poor choice. However, this heuristic does decrease the **tree space** (a discrete space) to:

$$T(n) = 3 + 5 + 7 + 9 + 11 + ... + (2n-3)$$

This method of building a topology is intrinsically greedy. It is typically used as a starting point for further searching.

Another way to explore the space is through *local rearrangement* by **Nearest Neighbour Interchange (NNI)**. For example, for a tree with four lineages there is only three possible topologies, one of which is already computed. The problem with NNI is that is a very narrow-scope operator on trees. Local maxima are very common. Furthermore, it is sensitive to the initial tree which may not recover from some type of error early in the optimization. This has a $O(n)$ linear complexity. Note that a tree is required here.

Other operators that are used on trees (global methods) are ones that have the potential to sample the entire breadth of the search space in only a few consecutive iterations. These include **Subtree Pruning Regrafting (SPR)** and **Tree Bisection and Reconstruction (TBR)**.

SPR narrows the search space per step to $O(n^2)$. One internal edge is broken and you treat the internal node in one of the subtrees (A) as a notional root for that subtree. Next is a contraction of the other subtree (B) – going to an unrooted tree. Lastly, you add the first chosen subtree B to any edge in A. This opens up the possible of number of moves from a candidate tree to a larger set and amounts to the square of the number of leaves.

TBR is an algorithm that randomizes the site of reconnection in both subtrees. The search space here is narrowed down to $O(n^3)$. Breaking of the edges is done here as well, but then both subtrees are contracted and all rootings are systematically tested. However, you will most likely get a great deal of very poor trees.

There are several ways of exploring the tree space. However, given a tree, how do we evaluate it? How do we test for sampling error in trees? The burning question here is whether or not the topology of the optimal tree is very sensitive to the precise content of the input set? The solution is to use a permutation technique called **bootstrapping** to create a population of datasets. A tree is inferred for each replicate and the results are summarized. For instance, one can count the frequency of observing a given internal node. A high "bootstrap value" implies that a node is stable to sampling error. It does not mean that a given internal node is *real*.

After generating a number of bootstrap replicates, you can infer an equal number of trees. This collection of trees can be evaluated by looking at the bipartitions of these trees (a number of which are equal to the number of leaves in the tree) such that you can build a multi-set. Counting the most frequent bipartition in this set for the entire collection of trees, you can start reconstructing the tree iteratively.

When looking at a bootstrap tree, you can evaluate the presence and ratio of bipartitions in a given set of replicates. Therefore, low bootstrap values can be ignored.

Exploring the Topology Space continued on next page...

Note that bootstrapping is sensitive to systematic error since it considers only the best ree from n different replicate experiments. With an infinitely long MSA, all bootstrap support will be 100%. This does not rule out the presence of a systematic error, just that there is no sampling error.

Structural Bioinformatics

Protein Structure

Recall that from genes in DNA, and through transcription, we can get an mRNA molecule, which, by translation, becomes a chain of amino acids – this is a protein chain. Each amino acid has a slightly different molecule with different properties. When the energy of this chain is minimized, a unique folding will result.

Helices in a protein are a sort of **secondary structure**, as are strands (which snap together when they are side-by-side). In *6CRO*, its mode of action is to float and find a matching configuration on DNA where it will bind. This occurs at these helix substructures. This protein is an effector for transcription.

What are structures? they are a *hierarchical assembly of sequences of residues* in 3 dimensions. Primarily, there is a linear assembly of amino-acids. The second level of structures are spontaneous formations of locally stable geometrics. These include parallel and antiparallel β sheets, as well as right-handed α helices (which have 3.6 residues per turn). Furthermore, these can be classified through a taxonomy where structures can be defined and focused in on to a lower and lower level of classification (see slide). Tertiary structures are 3D assemblies of secondary structure elements into a molecular machine. The backbone of a protein is comprised of alpha-carbons and nitrogens.

Most proteins are made of a number of domains. **Domains** are evolutionary units which can be seen as modules. Studying the structure of modular assembly is useful in itself. Breaking a protein into its domain is common prior to comparing protein folds. Half of all known proteins are multi-domains and most of them are 2-domain assemblies.

The structural data comprises the **PDB format** – an ASCII flat format. Each atom gets an (x, y, z) coordinate, each atom gets an atom name, a type, and a parent molecule; certainty is expressed as a temperature factor. Structures are typically found through x-ray diffraction, NMR, and crystallography. The PDB format is meant to be easily parsable by scripts and programs; it is in fact rather clunky to work with. The PDB is moving toward a new standard based on XML. Information is organized as a tree rather than to rely on the precise position in a line.

Interesting to note is that, despite the great number of possible gene sequences, there are only around 3000 possible superfamilies. Therefore, a great deal of sequences actually map to a very small subset of structures and shapes. There is, however, a historical bias for soluble structures – proteins found in membranes are, for the most part, very difficult to resolve.

Pairwise Alignments and Combinatorial Extension

Structural alignment is the ground truth for sequence alignments. In other words, two sequence positions are homologous only if they occupy the same position in three-dimensions.

Note that structures evolve at a much slower rate. The set of shapes is not very large and, therefore, any structure above 20% has a typically very good alignment score. Aligning structures gives insight into what is important in a structure, and how they differ between themselves. **Functional regions** are usually conserved.

Pairwise Alignments and Combinatorial Extension continued on next page...

The most simple way of aligning proteins is through **Rigid Body Superimposition (RBS)**. For two protein structures, a geometric transformation is applied on one of them which minimizes the overall distance between pairs of atoms. The problem with this approach is that there is no way to known which sites to pair together in the first place, or even whether a given site should be paired.

However, going back to sequence alignment, we could apply the same principle but change the scoring system. What we want is to define a coordinate system where we can fill the matrix in the first place. There are two ways to do this:

- **Intermolecular comparisons**: an absolute distance between two paired sites.
- **Intramolecular comparisons**: a difference between two sets of relative descriptions of a site's context. No transformation is involved.

For intermolecular comparisons, a proposal alignment is started with (which is usually done by aligning sequences where columns could be all of the same character). This is used as an anchor point where a matrix can be found to maximize the similarity between the proteins. Then, every residue vs. every residue can be scored. From there, a matrix can be accumulated and there will be a path through this matrix which can make a 1:1 correspondence. This can be used to find a better alignment in order to find an even better one. This is repeated until there is no longer any improvement.

The scoring scheme for the **STAMP** algorithm is as simple as:

$$S_{ij} = 1/(d_{ij})$$

Where d is the distance between a pair.

On the other hand, intramolecular comparisons involves defining a mean to evaluate the relationship of an element to the rest of a protein chain. The view would be a factor of beta-carbon to beta-carbon distances for each site. The coordinate referential for these vectors is based on the local geometry of the alpha-carbon (some origin). This is constructed for every residue.

Protein Folding

If we had the answer to the question, so many doors would be open into the potential of engineering. There exists a relationship between 1-dimensional data and the 3-dimensional effector. If we could create a model where, given a 1-dimensional sequence, the 3-dimensional structure could be predicted – an engineering process could be constructed.

The problem is we do not really have the ability to do that. What is the edge of our knowledge here in trying to engineer a protein to do exactly what we want? A great example of protein engineering is the engineering of an enzyme that would attach to a plastic and break it down into smaller molecules. You could make oil from plastic or bring it back into compost.

If a system was given with two charges, a positive and negative atom, attraction would occur. Once they get close enough, they will begin to bounce together and create a harmonic system. Everything we know about molecules here will be extended to a mechanical simulation. Note that a **model** is a hypothetical description of a complex entity or process.

To model a protein we need: (1) a description of the system – **atom coordinates**, (2) a model or abstraction of what is going on – **force field**, and (3) a process to find the best protein structure – **optimization**.

Force fields are a set of mathematical functions and parameters to model each relevant force (including VdW forces) and include an energy function where molecule interaction is modelled through elasticity. A sum of

terms that approximate the contributions of known theoretical molecular forces form the energy function. Note that electrostatic fields decay with $1/(distance)$ – they are the longest-ranged interactions and can be modelled by Coloumb's Law. Non-bond interactions create a computational bottleneck.

The process to find the best structure involves optimization. Force fields provide all the functions and parameters to compute the energy of 1 structure. Finding the best structure comes down to finding the structure with lowest energy. The optimization is iterated until it is reasonable to believe that no better structure can be found.

Therefore, a simple process includes: starting with a protein of coordinates, evaluating the energy, finding the best structure, and repeat until no better structure can be found. This task is usually never straightforward unless the system is made of a very small number of atoms. Minimizing is a gradient method for each atom (see slide).

However, the problem here is there a very strong bias towards local minima. Optimization landscapes for proteins have a great deal of these and a minimization approach here has trouble finding the lowest energy structure. The odds of actually finding the global minimum is therefore very small using solely a minimization approach.

Molecular simulations involve time dependent methods (molecular dynamics). This involves Newtonian Physics. It makes use of the classical mechcanics equation:

$$F = m \times a$$

Each atom gets a random kinetic energy vector which is added to the resultant force vector. This simulates thermal motion. In the optimization landscape, more local minima can therefore be reached. The structure becomes much freer.

In molecular simulations, the **verlet algorithm** is made us of. It is a numerical solution to Newton's equations (see slides) – friction-less cart down an incline. This builds the **integrator**.

$$r_{i+1} = (2r_i - r_{i-1}) + a_i \times \Delta t^2$$

At this scale, quantum effects are neglected because for a 1000+ system, iterations would hardly be gone through. Empirically, however, this system works. No chemistry is typically involved.

What is a reasonable time step Δt? In molecular simulations, a reasonable one is femtoseconds: 10^{-15}. The scope of a protein folding simulation is ideally in the millisecond scale or practically in the microsecond scale – these are simulated in steps of femtoseconds.

Other optimization strategies involves **simulated annealing**. This is where the energy landscapes are scaled down, making the crossing of barriers much easier and more probable. If the landscape is cooled back down slowly, there is a guarantee the simulation will end in the right location at the global minimum. This, however, takes a great deal of time – 2 weeks can become 6 months because of this cooling down.

Folding polypeptides is not expected to work out as well because empirical models are parameterized with pre-folded proteins. The role of water in partially folded proteins is significant. The time scale for folding a protein is still a bit out of range for simulation. These are just a few of the problems arising (see slides).

So what is the solution here? You could always build a larger machine for protein folding such as where 65,000 processors were contributed for this cause (IBM Blue Gene). How many processors is enough, however? Parallel computing is used quite a bit in these simulations. There is a point where this no longer works

anymore as possible tasks that can be divided becomes smaller and smaller. Folding proteins from an extended conformation is a difficult problem because of the crossing of energy barriers.

Let's take the example of a thesis. One student takes about 1500 days. If the student is helped by someone, it may go two times as fast: 750 days. What if 1500 students are working on the same thesis? This will *not* work. There is overhead, communication that is necessary, and load balancing. There is a theoretical limitation to what you can do with parallel processing. Some work has to be executed in a sequence, communicating the task and results becomes increasingly important, each individual process has to wait for the slowest one to finish, and it does not make sense to have more CPUs than atoms in your system, as it is with the thesis.

NOTE Another way of parallelizing that makes the system more efficient is seen in the folding@home system. This uses unspent cycles from idle hardware including PS3s, Xboxes, or PCs where a parcel of a simulation is received, worked on, and sent back. On its own, it does not solve the problem. There is still an overhead problem. However, the approach in this system involves trying to crossing the energy barrier. The chance to witness this event during a 30 ns simulation is 0.3%. If the same simulation runs on 10,000 machines, one would expect to observe the event around 30 times over 30 ns. Every process is therefore *competing* against each other to be successful. In a way, this circumvents the limitation of parallel computation by becoming a system of competition rather than collaboration. This is known as **Ensemble Dynamics**. The communication overhead here is negligible if the crossings are rare events – this is the case here. Jumping between valleys occurs very quickly then.