CSCE 471/871 Lecture 5: Building Phylogenetic Trees

Stephen Scott

sscott@cse.unl.edu

Outline

Phylogenetic Trees

- Assumption: all organisms on Earth have a common ancestor
  - $\Rightarrow$ all species are related in some way
- Relationships represented by *phyogenetic trees*
- Trees can represent relationships between orthologs or paralogs
  - Othorlogs: Genes in different species that evolved from a common ancestral gene by **speciation** (evolution of one species out of another)
    - Normally, orthologs retain the same function in the course of evolution
  - Paralogs: genes related by duplication within a genome
    - In contrast to orthologs, paralogs evolve new functions

Phylogenetic Trees (2)

We'll use binary trees, both rooted and unrooted

- Rooted for when we know the direction of evolution (i.e., the common ancestor)
- Can sometimes find the root by adding a distantly related organism/sequence to an existing tree (Fig 7.1)

Phylogenetic Trees (3)

- A weighted tree, where each weight (*edge length*) is an estimate of evolutionary time between events
  - Based on distance measure (e.g., substitution scoring matrices) between sequences
  - Gives a reasonably accurate approximation of relative evolutionary times, despite the fact that sequences can evolve at different rates
- Number of possible binary trees on n nodes grows exponentially in n
  - E.g., $n = 20$ has about 2.2×10^{20} trees
  - We'll use hueristics, of course

Building Trees from Pairwise Distances
UPGMA

Start with some distance measure between sequences, e.g., Jukes-Cantor:

$$d_{ij} = -0.75 \log(1 - 4f_{ij}/3) \ ,$$

where f_{ij} is fraction of residues that differ between sequences x_i and x_j when pairwise aligned

UPGMA (unweighted pair group method average) algorithm

- One of a family of hierarchical clustering algorithms
- Basic idea of algorithmic family: Find minimum inter-cluster distance d_{ij} in current distance matrix, merge clusters i and j, then update distance matrix
- Differences among algorithms lie in matrix update
- For phylogenetic trees, also add edge lengths

Building Trees from Pairwise Distances
UPGMA (2)

1. $\forall\, i$, assign seq x_i to cluster C_i and give it its own leaf, with height 0
2. While there are more than two clusters
   1. Find minimum d_{ij} in distance matrix
   2. Add to the clustering cluster $C_k = C_i \cup C_j$ and delete C_i and C_j
   3. For each cluster $C_\ell \notin \{C_k, C_i, C_j\}$

$$d_{k\ell} = \frac{1}{|C_k|\,|C_\ell|} \sum_{p \in C_k, q \in C_\ell} d_{pq}$$

   [Shortcut: Eq. (7.2)]

   4. Add to the tree node k with children i and j, with height $d_{ij}/2$
3. When only C_i and C_j remain, place root at height $d_{ij}/2$

Example: Fig 7.4

Building Trees from Pairwise Distances
UPGMA (3)

- If the rate of evolution is the same at all points in original (target) phylogenetic tree, then UPGMA will recover the correct tree
  - This occurs iff length of all paths from root to leaves are equal in terms of evolutionary time
- If this is not the case, then UPGMA may find incorrect topology (Fig. 7.5, p. 170)
- Can avoid this if distances satisfy ultrametric condition: for any three sequences x_i, x_j, x_k, the distances d_{ij}, d_{jk}, d_{ik} are either all equal, or two are equal and one is smaller

Building Trees from Pairwise Distances
Neighbor Joining

If ultrametric property doesn't hold, can still recover original tree if additivity holds

- If, in original tree, distance between any pair of leaves = sum of lengths of edges of path connecting them

If additivity holds, neighbor joining finds the original tree

- First, find a pair of neighboring leaves i and j, assign them parent k, then replace i and j with k, where for all other leaves m, $d_{km} = (d_{im} + d_{jm} - d_{ij})/2$
- But it does NOT work to simply choose pair (i,j) with minimum d_{ij} (Fig. 7.7)
- Instead, choose (i,j) minimizing $D_{ij} = d_{ij} - (r_i + r_j)$, where L is current set of "leaves" and

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

Building Trees from Pairwise Distances
Neighbor Joining (2)

1. Initialize $L = T =$ set of leaves
2. While $|L| > 2$
   1. Choose i and j minimizing D_{ij}
   2. Define new node k and set $d_{km} = (d_{im} + d_{jm} - d_{ij})/2$ for all $m \in L$
   3. Add k to T with edges of lengths $d_{ik} = (d_{ij} + r_i - r_j)/2$ and $d_{jk} = d_{ij} - d_{ik}$
   4. Update $L = \{k\} \cup L \setminus \{i,j\}$
3. Add final, length-d_{ij} edge between final nodes i and j

Parsimony

- Widely used approach for tree building
- Scores tree based on the cost of substitutions going from node to its child
  - ⇒ Will assign hypothetical ancestral sequences to internal nodes, e.g., Figure 7.9
- Generally consists of two components
  1. Computing cost of tree T over n aligned sequences
  2. Searching through the space of possible trees for min-cost one
- Treat each site independently of the others, so for a length-m alignment, run scoring algorithm on each of the m sites separately
- Let $S(a,b)$ be cost of substituting b for a
- Scoring site (tree) $u \in \{1, \ldots, m\}$, let $S_k(a)$ be the minimal cost for the assignment of symbol (residue) a to node k

Parsimony (2)

1. Initialize $k = 2n - 1$ (index of the root node)
2. Recursively compute $S_k(a)$ for all a in the alphabet:
   1. If k is a leaf, set $S_k(a) = 0$ for $a = x_u^k$ and $S_k(a) = \infty$ otherwise
      - ⇒ a must match uth symbol in sequence
   2. Else $S_k(a) = \min_b(S_i(b) + S(a,b)) + \min_b(S_j(b) + S(a,b))$, where i and j are k's children
3. Return $\min_a\{S_{2n-1}(a)\}$ as minimum cost of tree

Can recover ancestral residues by tracking where min comes from in recurisve step

Parsimony (3)
Searching for a Tree

- Not practical to enumerate the entire set of possible trees and score them all
- Will use branch and bound to speed it up (though no guarantee of an efficient algorithm)
  - When incrementally building a tree, adding edges will never decrease its cost
  - Thus if a tree's cost already exceeds the final cost of the best tree so far, we can discard it
- Algorithm: systematically grow existing tree by adding edges, stopping expansion if current tree's cost exceeds final cost of best tree so far

Hein's Algorithm

For simultaneously finding alignment and phylogeny

- Similar to parsimony in that, given a topology, it infers ancestral sequences
- But this algorithm uses an affine gap penalty model (separate penalties for opening and extending gaps)
- First, it ascends the tree from the leaves, determining the set of sequences that best align with leaf sequences
  - Represents such a set of sequences as a digraph
- Then it works its way up toward the root, at each step inferring the set of sequences that best align with the child graphs
- Finally, it descends from the root to the leaves, fixing the specific ancestral sequences

Hein's Algorithm
Finding Set of Sequences that Best Align with Leaves

- GOAL: Given sequences x and y, find set of sequences such that for each such sequence z, $S(x,z)+S(z,y)=S(x,y)$
- Use DP to handle affine gap penalties
  - $V^M(i,j)=$ min cost aligning $x_{1\ldots i}$ to $y_{1\ldots j}$; x_i aligned to y_j

$$V^M(i,j) = \min\{V^M(i-1,j-1),\ V^X(i-1,j-1),\ V^Y(i-1,j-1)\}+S(x_i,y_j)\}$$

  - $V^X(i,j)=$ min cost aligning $x_{1\ldots i}$ to $y_{1\ldots j}$; x_i aligned to gap

$$V^X(i,j)=\min\{V^M(i-1,j)+d,\ V^X(i-1,j)+e\}$$

  - $V^Y(i,j)=$ min cost aligning $x_{1\ldots i}$ to $y_{1\ldots j}$; y_j aligned to gap

$$V^Y(i,j)=\min\{V^M(i,j-1)+d,\ V^Y(i,j-1)+e\}$$

Hein's Algorithm
Finding Set of Sequences that Best Align with Leaves (2)

- Dynamic programming example in Fig. 7.13
  - j indexes rows, i indexes columns; seq. x is bottom/horizontal
  - E.g., row $j=0$, X entries are costs of opening + extending gaps aligned against x
- Result is a set of paths through the DP table, each corresponding to an optimal alignment between x and y:

```
CAC---        C--AC-
CTCACA        CTCACA
```

- Each alignment implies a set of valid ancestral sequences, where each such sequence z satisfies $S(x,z)+S(z,y)=S(x,y)$

Hein's Algorithm
Finding Set of Sequences that Best Align with Leaves (3)

```
CAC---        C--AC-
CTCACA        CTCACA
```

- Each alignment implies a set of valid ancestral sequences, where each such sequence z satisfies $S(x,z)+S(z,y)=S(x,y)$
  - If one position is a match between x_i and y_j, then a valid ancestral sequence z contains either x_i or y_j in that position
  - If a gap is involved, can take the gap or the residue
    - But since cost function is not linear, need to either take the entire gap or none of the gap
    - E.g., in Fig. 7.13, with leaves $y=$ CAC and $x=$ CTCACA, can use as ancestral sequence $z=$ CTC, CAC, CACACA, etc., but not CACAC (why?)

Hein's Algorithm
Finding Set of Sequences that Best Align with Leaves (3)

- Can represent set of sequences as a digraph (e.g., Fig. 7.14(a); edges directed to the right), aka a sequence graph, where each path through the graph corresponds to a valid ancestral sequence
- Null ("dummy") edges (denoted by δ) allow gaps to be entirely skipped

Hein's Algorithm
Building Sequence Graphs for Higher-Level Nodes

- Now want to ascend the tree towards the root, building ancestral sequence graphs for internal nodes
- But SG construction previously described ran DP on individual sequences!
- Turns out we can also run DP on SGs
  - In DP equations, "$i-1$" means the set of previous nodes in the horizontal graph, "$j-1$" in the vertical graph
  - Now take minimum over entire set of previous nodes that have values defined (non-"–")
  - Scoring function S now defined on sets; it's 0 iff its set-type arguments have non-empty intersection
    - E.g., $S(\{\mathsf{A}\},\{\mathsf{A},\mathsf{T}\}) = 0$ due to overlap
- Once DP completed, do another traceback and build new SG
  - When labeling edges in new SG, use the intersection of the labels in the two defining edges, or the union if the intersection is empty

Hein's Algorithm
Filling in Ancestral Sequences

- Now choose a path in the root's SG, then go to child nodes and trace their SGs with its parent's ancestral sequence, choosing compatible symbols
- In final multiple alignment, need to fill in gaps

Hein's Algorithm
Building the Topology

- Still need to build the tree to align sequences to
- Hein's tree-building algorithm:
  1. Compute an informative subset of the inter-sequence distances
  2. Build a "distance tree" by adding sequences to it one by one
  3. Perform rearrangements on the tree to improve its fit to the distance data
  4. Align sequences to the tree (what we already covered)

Hein's Algorithm
Building the Topology (2): Computing Subset of Distances

- Assume that the distance measure and sequences form a metric space, implying:
  - $d(s_1,s_2) = 0 \Leftrightarrow s_1 = s_2$
  - $d(s_1,s_2) = d(s_2,s_1)$
  - $d(s_1,s) + d(s,s_2) \geq d(s_1,s_2)$
- Can use third eq. to upper- and lower-bound unknown distances
- I.e., if differences between upper and lower bounds is smaller than a paremeter, do not compute the exact value

Hein's Algorithm
Building the Topology (3): Computing Distance Tree

- Add sequences one at a time
- Choose to add to T_{k-1} the sequence s_k minimizing

$$d(s_k, T_{k-1}) = \min_{s_j \in leaves(T_{k-1})} \{d(s_k, s_j)\}$$

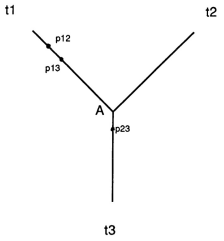
Hein's Algorithm
Building the Topology (4): Computing Distance Tree (cont'd)

Choose s_k's attachment point as follows:

- Let s_1 be sequence in tree most similar to s_k
- Let A be internal node closest to s_1, and $S = \{t_1, t_2, t_3\}$ be the set of subtrees leaving A
- For each $t_i, t_j \in S$, compute $d(t_i, s_k)$ and $d(t_i, t_j)$ by computing average distance among pairs of leaves

Hein's Algorithm

Building the Topology (5): Computing Distance Tree (cont'd)



Hypothetically, if we attached s_k on the path from t_i to t_j, then to preserve additivity, we'd place it at point p_{ij} such that

$$d(t_i, p_{ij}) = (d(t_i, s_k) + d(t_i, t_j) - d(t_j, s_k))/2$$

(I.e., if s_k is at p_{ij}, then $d(t_i, t_j) = d(t_i, s_k) + d(t_j, s_k)$)

Hein's Algorithm

Building the Topology (6): Computing Distance Tree (cont'd)

- Now let $v_1 =$ avg distance in direction of t_1 of p_{12} and p_{13} from A; similarly define v_2 and v_3
- Maximum of these 3 distances determines attachment point
  (Intuition: If t_i far from A and near s_k, this is s_k's home)
- If the max v_i takes us past the root of t_i, then t_i's root becomes A and the process repeats
- Once all nodes added, look at interchanging neighbors in tree to improve score