CSCI1950-Z
Computational Methods for Biology
Lecture 2

Ben Raphael
January 26, 2009
http://cs.brown.edu/courses/csci1950-z/

Outline

• Review of trees. Counting features.
• Character-based phylogeny
  – Maximum parsimony
  – Maximum likelihood
Tree Definitions

tree: A **connected acyclic** graph \( G = (V, E) \).

graph: A set \( V \) of **vertices (nodes)** and a set \( E \) of **edges**, where each edge \((v_i, v_j)\) connects a pair of vertices.

A path in \( G \) is a sequence \((v_1, v_2, \ldots, v_n)\) of vertices in \( V \) such that \((v_i, v_{i+1})\) are edges in \( E \).

A graph is **connected** provided for every pair \( v_i, v_j \) of vertices, there is a path between \( v_i \) and \( v_j \).

A cycle is a path with the same starting and ending vertices.

A graph is **acyclic** provided it has no cycles.

---

degree of vertex \( v \) is the number of edges incident to \( v \).

A **phylogenetic tree** is a tree with a label for each **leaf** (vertex of degree one).

A **binary phylogenetic tree** is a phylogenetic tree where every **interior** (non-leaf) vertex has degree 3; (one parent and two children).

A **rooted** (*binary) phylogenetic tree is phylogenetic tree with a single designated vertex \( r \) (* of degree 2).

\( w \) is a **parent (ancestor)** of \( v \) provided \((v,w)\) is on path to root. In this case \( v \) is a **child (descendant)** of \( w \).
Tree Definitions

*tree*: A **connected acyclic** graph $G = (V, E)$.

*degree* of vertex $v$ is the number of edges incident to $v$.

A **phylogenetic** tree is a tree with a label for each *leaf* (vertex of degree one).

- Leaves represent existing species
- Other vertices represent most recent common ancestor.
- Length of branches represent evolutionary time.
- Root (if present) represents the oldest evolutionary ancestor.

Counting and Trees

- A tree with $n$ vertices has $n-1$ edges. (Proof?)
- A rooted binary phylogenetic tree with $n$ leaves has $n-1$ internal vertices; and thus $2n -1$ total vertices.
- How many rooted binary phylogenetic trees with $n$ leaves?
1. What is character data?
2. What is the criteria for evaluating a tree?
3. How do we optimize this criteria:
   1. Over all possible trees?
   2. Over a restricted class of trees?

Character-Based Tree Reconstruction

- Characters may be morphological features
  # of eyes or legs or the shape of a beak or a fin.

- Characters may be nucleotides of DNA (A, G, C, T) or amino acids (20 letter alphabet).

- Values are called states of character.

Gorilla: CCTTGACGTACAAACGA
Chimpanzee: CCTTGACGTGCACAAACGA
Human: CCTTGACGTGCACAAACGA

Non-informative character
Character-Based Tree Reconstruction

**GOAL**: determine what character strings at internal nodes would *best explain* the character strings for the *n* observed species

---

**An Example**

<table>
<thead>
<tr>
<th></th>
<th>Value1</th>
<th>Value2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mouth</td>
<td>Smile</td>
<td>Frown</td>
</tr>
<tr>
<td>Eyebrows</td>
<td>Normal</td>
<td>Pointed</td>
</tr>
</tbody>
</table>
Character-Based Tree Reconstruction

Which tree is better?

Character-Based Tree Reconstruction

Count changes on tree
Character-Based Tree Reconstruction

Maximum Parsimony: minimize number of changes on edges of tree

Maximum Parsimony

- Ockham’s razor: “simplest” explanation for the data
- Assumes that observed character differences resulted from the fewest possible mutations
- Seeks tree with the lowest possible parsimony score, defined sum of cost of all mutations found in the tree
Character Matrix

Given $n$ species, each labeled by $m$ characters. Each character has $k$ possible states.

Gorilla: CCTGTGAGCTAACAAACGA
Chimpanzee: CCTGTGAGCTAGCAAACGA
Human: CCTGTGAGCTAGCAAACGA

$n \times m$ character matrix

Assume that characters in character string are independent.

Parsimony Score

Gorilla: CCTGTGAGCTAACAAACGA
Chimpanzee: CCTGTGAGCTAGCAAACGA
Human: CCTGTGAGCTAGCAAACGA

Assume that characters in character string are independent.

Given character strings $S=s_1...s_m$ and $T=t_1...t_m$:

#changes ($S \rightarrow T$) = $\sum_i d_H(s_i, t_i)$

where $d_H = $ Hamming distance

$d_H(v, w) = 0$ if $v=w$
$d_H(v, w) = 1$ otherwise

parsimony score of the tree as the sum of the lengths (weights) of the edges
Parsimony and Tree Reconstruction

Maximum Parsimony

Two computational sub-problems:
1. Find the parsimony score for a fixed tree.
   – Small Parsimony Problem (easy)
2. Find the lowest parsimony score over all trees with $n$ leaves.
   – Large parsimony problem (hard)
Small Parsimony Problem

**Input:** Tree $T$ with each leaf labeled by an $m$-character string.

**Output:** Labeling of internal vertices of the tree $T$ minimizing the parsimony score.

Since characters are independent, every leaf is labeled by a single character.

---

Small Parsimony Problem

**Input:**
- $T$: tree with each leaf labeled by an $m$-character string.

**Output:**
- Labeling of internal vertices of the tree $T$ minimizing the parsimony score.

Large Parsimony Problem

**Input:**
- $M$: an $n \times m$ character matrix.

**Output:**
- A tree $T$ with:
  - $n$ leaves labeled by the $n$ rows of matrix $M$
  - labeling of the internal vertices of $T$ minimizing the parsimony score over all possible trees and all possible labelings of internal vertices.
Small Parsimony Problem

**Input:** Binary tree $T$ with each leaf labeled by an $m$-character string.

**Output:** Labeling of internal vertices of the tree $T$ minimizing the parsimony score.

Since characters are independent, every leaf is labeled by a single character.

Weighted Small Parsimony Problem

More general version of Small Parsimony Problem

- Input includes a $k \times k$ scoring matrix $\delta$ describing the cost of transforming each of $k$ states into another state.
- Small Parsimony Problem is special case:
  \[
  \delta_{ij} = \begin{cases} 
  0, & \text{if } i = j, \\
  1, & \text{otherwise.}
  \end{cases}
  \]
## Scoring Matrices

**Small Parsimony Problem**

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>T</th>
<th>G</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>T</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>G</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**Weighted Small Parsimony Problem**

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>T</th>
<th>G</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>3</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>T</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>G</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>C</td>
<td>9</td>
<td>4</td>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>

## Unweighted vs. Weighted

**Small Parsimony Scoring Matrix:**

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>T</th>
<th>G</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>T</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>G</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**Small Parsimony Score:** 5
Unweighted vs. Weighted

Weighted Parsimony Scoring Matrix:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>T</th>
<th>G</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>3</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>T</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>G</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>C</td>
<td>9</td>
<td>4</td>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>

Weighted Parsimony Score: 22

Weighted Small Parsimony Problem

**Input:**

* T*: tree with each leaf labeled by an *m*-character string from a *k*-letter alphabet.

* δ*: *k* x *k* scoring matrix

**Output:** Labeling of internal vertices of the tree *T* minimizing the weighted parsimony score.
Sankoff Algorithm

Calculate and keep track of a score for every possible label at each vertex:
\[ s_t(v) = \text{minimum parsimony score of the subtree rooted at vertex } v \text{ if } v \text{ has character } t \]

The score \( s_t(v) \) is based only on scores of its children:
\[
\begin{align*}
    s_t(\text{parent}) &= \min_i \{ s_i( \text{left child} ) + \delta_{i,t} \} + \\
                      &\quad \min_j \{ s_j( \text{right child} ) + \delta_{j,t} \}
\end{align*}
\]
Sankoff Algorithm (cont.)

- Begin at leaves:
  - If leaf has the character in question, score is 0
  - Else, score is $\infty$

Sankoff Algorithm (cont.)

$$s_i(v) = \min_i \{s_i(u) + \delta_{i,t}\} + \min_j \{s_j(w) + \delta_{j,t}\}$$

<table>
<thead>
<tr>
<th>$s(u)$</th>
<th>$\delta_{i,A}$</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>$\infty$</td>
<td>3</td>
</tr>
<tr>
<td>G</td>
<td>$\infty$</td>
<td>4</td>
</tr>
<tr>
<td>C</td>
<td>$\infty$</td>
<td>9</td>
</tr>
</tbody>
</table>
Sankoff Algorithm (cont.)

\[ s_i(v) = \min_i \{ s_i(u) + \delta_{i,t} \} + \min_j \{ s_j(w) + \delta_{j,t} \} \]

\[ s_j(u) \]

\[ \delta_j \]

\[ \sum \]

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>T</th>
<th>G</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>s(u)</td>
<td>\infty</td>
<td>0</td>
<td>0</td>
<td>9</td>
</tr>
<tr>
<td>\delta_{j,A}</td>
<td>3</td>
<td>\infty</td>
<td>\infty</td>
<td>9</td>
</tr>
<tr>
<td>sum</td>
<td>\infty</td>
<td>\infty</td>
<td>\infty</td>
<td>9</td>
</tr>
</tbody>
</table>

\[ s_i(v) = 0 \]

+ 9 = 9

Repeat for T, G, and C
Sankoff Algorithm (cont.)

Repeat for right subtree

Sankoff Algorithm (cont.)

Repeat for root
Sankoff Algorithm (cont.)

Smallest score at root is minimum weighted parsimony score

In this case, 9 – so label with T

Sankoff Algorithm: Traveling down the Tree

- The scores at the root vertex have been computed by going up the tree
- After the scores at root vertex are computed the Sankoff algorithm moves down the tree and assign each vertex with optimal character.
Sankoff Algorithm (cont.)

9 is derived from $7 + 2$
So left child is T,
And right child is T

Sankoff Algorithm (cont.)

And the tree is thus labeled…
Analysis of Sankoff’s Algorithm

A dynamic programming problem algorithm:

**Optimal substructure:** solution obtained by solving smaller problem of same type.

\[ s_i(\text{parent}) = \min_j \{ s_j(\text{left child}) + \delta_{i,j} \} + \min_j \{ s_j(\text{right child}) + \delta_{i,j} \} \]

Recurrence terminates at leaves, where solution is known.

How many computations do we perform for \( n \) species, \( m \) characters, and \( k \) states per character?

Forward step:
- At each internal node of tree:
  \[ s_i(\text{parent}) = \min_j \{ s_j(\text{left child}) + \delta_{i,j} \} + \min_j \{ s_j(\text{right child}) + \delta_{i,j} \} \]
- 2k sums and 2 \((k-1)\) comparisons = 4k - 2
- \( n-1 \) internal nodes.
- \((4k - 2)(n-1)\) sums.

Traceback: one “lookup” per internal node. \((n-1)\) operations

For each character \((4k - 2)(n-1) + (n-1)\) operations \(\leq C \ n \ k\)
- Above calculation performed once for each character:
  \(\leq C \ m \ n \ k\) operations
- \(O( m \ n \ k)\) time. [“big-O”]
- Increases linearly w/ \# of species or \# of characters.
Analysis of Sankoff’s Algorithm

*How many computations do we perform for n species, m characters, and k states per character?*

Traceback: 2k sums
- Above calculation performed once for each character
- O( m n k) time. [“big-O”]
- Increases linearly w/ # of species or # of characters.

Fitch’s Algorithm

- Solves Small Parsimony problem
  - Published 4 years before Sankoff (1971)
- Makes two passes through tree:
  - Leaves → root.
  - Root → leaves.
Fitch Algorithm: Step 1

Assign a set $S(v)$ of letters to every vertex $v$ in the tree, traversing the tree from leaves to root:

- $S(l)$ = observed character for each leaf $l$
- For vertex $v$ with children $u$ and $w$:
  
  $S(v) = \begin{cases} 
  S(u) \cap S(w) & \text{if non-empty intersection} \\
  S(u) \cup S(w) & \text{otherwise}
  \end{cases}$

- E.g. if the node we are looking at has a left child labeled $\{A, C\}$ and a right child labeled $\{A, T\}$, the node will be given the set $\{A, C, T\}$

Fitch’s Algorithm: Example

[Diagram of a tree with labeled nodes demonstrating the algorithm’s application]
Fitch Algorithm: Step 2

Assign labels to each vertex, traversing the tree from root to leaves.

- Assign root \( r \) arbitrarily from its set \( S(r) \).
- For all other vertices \( v \):
  - If its parent’s label is in its set \( S(v) \), assign it its parent’s label.
  - Else, choose an arbitrary letter from its set \( S(v) \) as its label.

Fitch’s Algorithm: Example
Fitch Algorithm (cont.)

Fitch vs. Sankoff

- Both have an O(nk) runtime
- Are they actually different?
- Let’s compare …
As seen previously:

**Comparison of Fitch and Sankoff**

- As seen earlier, the scoring matrix for the Fitch algorithm is merely:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>T</th>
<th>G</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>T</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>G</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

- So let’s do the same problem using Sankoff algorithm and this scoring matrix
Sankoff vs. Fitch

- The Sankoff algorithm gives the same set of optimal labels as the Fitch algorithm.
- For Sankoff algorithm, character $t$ is optimal for vertex $v$ if $s_t(v) = \min_{1 \leq i \leq k} s_i(v)$.
- Let $S_v$ = set of optimal letters for $v$.
- Then

$$S_v = \begin{cases} 
S_u \cap S_w & \text{if } S_u \cap S_v \neq \emptyset, \\
S_u \cup S_w, & \text{otherwise}. 
\end{cases}$$

- This is also the Fitch recurrence.
- The two algorithms are identical.
A Problem with Parsimony

Ignores branch lengths on trees

Same parsimony score.
Mutation “more likely” on longer branch.

Probabilistic Model

Given a tree $T$ with leaves labeled by present characters, what is the probability of a labeling of ancestral nodes?

Assume:
1. Characters evolve independently.
2. Constant rate of mutation on each branch.
3. State of a vertex depends only on parent and branch length:
   i.e. $\Pr[x \mid y, t]$ depends only on $y$ and $t$. (Markov process)
Probabilistic Model

Two species

Pr[x | y, t] = probability that y mutates to x in time t

Pr[x_1, x_2, a | T, t_1, t_2] = q_a Pr[x_1 | a, t_1] Pr[x_2 | a, t_2]

q_a = Pr[ancestor has character a]

Probabilistic Model

n species: x^1, x^2, ..., x^n

Let α(i) = ancestor of node i.

Let a^{n+1}, a^{n+2}, ..., a^{2n-1} = characters on internal nodes, where nodes are number from internal vertices up to root.

Pr[x^1, ..., x^n | T, t_1, ..., t_{2n-2}] =

\sum_{a^{n+1}, a^{n+2}, ..., a^{2n-1}} q_{a^{2n-1}} \prod_{i=n+1}^{2n-2} Pr[a^i | a^{α(i)}, t_i] \prod_{i=1}^{n} Pr[x^i | a^{α(i)}, t_i]

Follows from Law of Total Probability: P(X) = \Sigma P(X|Y_j) P(Y_j).
Felsenstein’s Algorithm

Let \( Pr[T_k \mid a] \) = probability of leaf nodes “below” node \( k \), given \( a^k = a \).

Compute via dynamic programming

\[
Pr[T_k \mid a] = \sum_b Pr[b \mid a, t_i] Pr[T_i \mid b] \sum_c Pr[c \mid a, t_j] Pr[T_j \mid c]
\]

Initial conditions. For \( k = 1, \ldots, n \) (leaf nodes)
\( Pr[T_k \mid a] = 1 \), if \( a = x^k \)
\( 0 \), otherwise.

Computing the Likelihood

Let \( Pr[T_k \mid a] \) = probability of leaf nodes “below” node \( k \), given \( a^k = a \).

\[
Pr[x^1, \ldots, x^n \mid T, t_*] = \sum_a Pr[T_{2n-1} \mid a] q_a
\]

Note: Root is node \( 2n-1 \)
Maximum Likelihood

Let $Pr[T_k | a] = \text{probability of leaf nodes “below” node } k, \text{ given } a^k = a.$

$$Pr[T_k | a] = \left( \max_b Pr[b | a, t_i] Pr[T_i | b] \right) \left( \max_c Pr[c | a, t_j] Pr[T_j | c] \right)$$

Traceback as before with Sankoff’s algorithm.

Max. Parsimony vs. Max. Likelihood

- Set $\delta_{ij} = -\log P(j | i)$ in weighted parsimony (Sankoff algorithm)
- Weighted parsimony produces “maximum probability” assignments, ignoring branch lengths