# Algorithmic Approaches for Biological Data, Lecture #22

Katherine St. John

City University of New York
American Museum of Natural History

27 April 2016

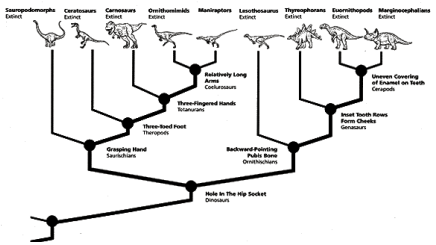
# Outline

Guest Lecturer: Dr. Eric Ford

- Fitch's Algorithm: scoring trees in linear time
- Building Trees: Wagner Builds, UPGMA, & Neighbor Joining

# Algorithm Design: Scoring Trees Under Parsimony

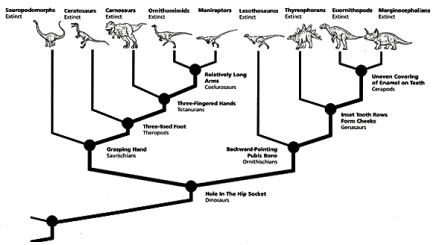
- How do you code this?




AMNH

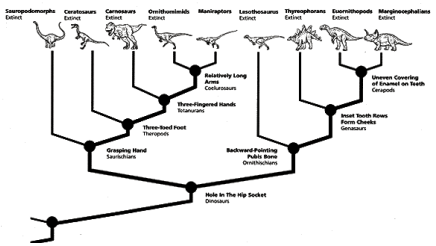
# Algorithm Design: Scoring Trees Under Parsimony

- How do you code this?
  - Input: A tree and sequences on the leaves.




AMNH

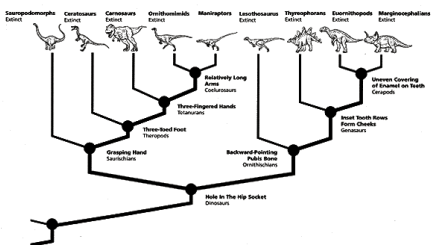
# Algorithm Design: Scoring Trees Under Parsimony




AMNH

- How do you code this?
  - Input: A tree and sequences on the leaves.
  - Output: The parsimony score of the tree (with respect to the leaf labels).

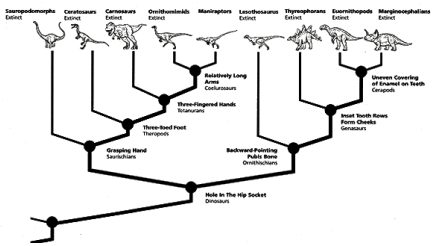
# Algorithm Design: Scoring Trees Under Parsimony




AMNH

- How do you code this?
  - Input: A tree and sequences on the leaves.
  - Output: The parsimony score of the tree (with respect to the leaf labels).
- What data structures do you need?

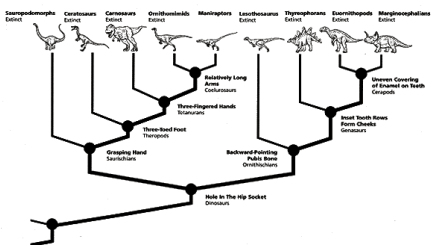
# Algorithm Design: Scoring Trees Under Parsimony




AMNH

- How do you code this?
  - Input: A tree and sequences on the leaves.
  - Output: The parsimony score of the tree (with respect to the leaf labels).
- What data structures do you need?
  - Tree structure

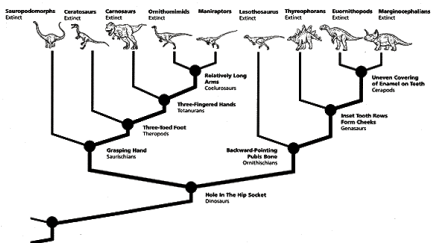
# Algorithm Design: Scoring Trees Under Parsimony




AMNH

- How do you code this?
  - Input: A tree and sequences on the leaves.
  - Output: The parsimony score of the tree (with respect to the leaf labels).
- What data structures do you need?
  - Tree structure
  - Count of the number changes

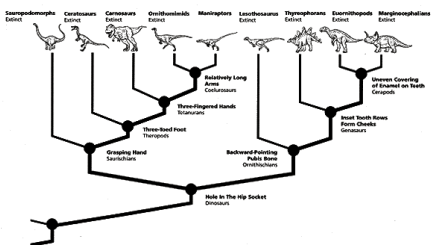
# Algorithm Design: Scoring Trees Under Parsimony




AMNH

- How do you code this?
  - Input: A tree and sequences on the leaves.
  - Output: The parsimony score of the tree (with respect to the leaf labels).
- What data structures do you need?
  - Tree structure
  - Count of the number changes
- Algorithm:
  - First pass: Starting at the leaves, label the internal leaves (with possible multiple labels).

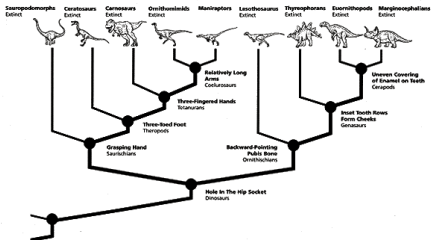
# Algorithm Design: Scoring Trees Under Parsimony




AMNH

- How do you code this?
  - Input: A tree and sequences on the leaves.
  - Output: The parsimony score of the tree (with respect to the leaf labels).
- What data structures do you need?
  - Tree structure
  - Count of the number changes
- Algorithm:
  - First pass: Starting at the leaves, label the internal leaves (with possible multiple labels).
  - Second pass: Starting at the root, choose a labeling, then work towards the leaves minimizing the conflicts.

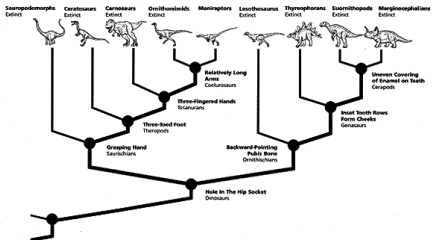
# Fitch's Algorithm: Pseudocode




AMNH

- First pass: Starting at the leaves, label the internal leaves (with possible multiple labels):

# Fitch's Algorithm: Pseudocode

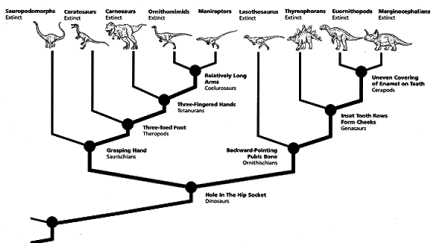



AMNH

- First pass: Starting at the leaves, label the internal leaves (with possible multiple labels):
  - Given labels for children, compute label for the parent:

    A T A T G

# Fitch's Algorithm: Pseudocode



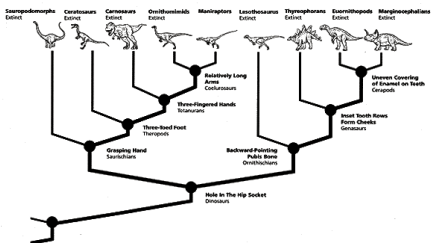

AMNH

- First pass: Starting at the leaves, label the internal leaves (with possible multiple labels):
  - Given labels for children, compute label for the parent:

    A T A T G

    A A T T G $\rightarrow$ A AT AT T G
  - Go position by position:

# Fitch's Algorithm: Pseudocode



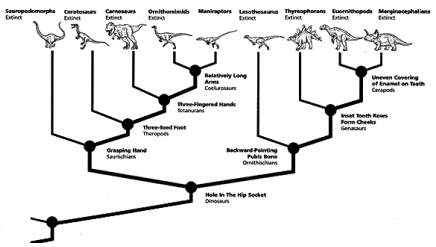

AMNH

- First pass: Starting at the leaves, label the internal leaves (with possible multiple labels):
  - Given labels for children, compute label for the parent:

    A T A T G

    A A T T G $\rightarrow$ A AT AT T G
  - Go position by position:
    - ★ If overlap, use that label.

# Fitch's Algorithm: Pseudocode



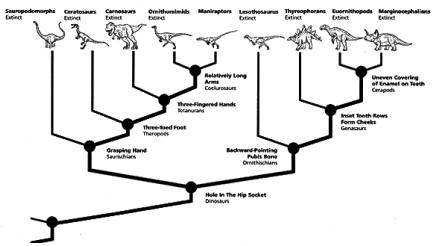

AMNH

- First pass: Starting at the leaves, label the internal leaves (with possible multiple labels):
  - Given labels for children, compute label for the parent:

    A T A T G

    A A T T G $\rightarrow$ A AT AT T G
  - Go position by position:
    - ★ If overlap, use that label.
    - ★ If no overlap, use the union.

# Fitch's Algorithm: Pseudocode



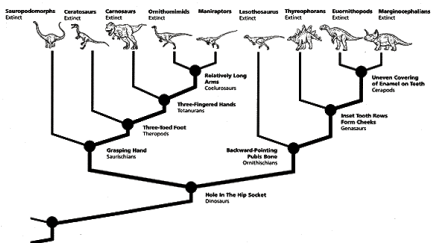

AMNH

- First pass: Starting at the leaves, label the internal leaves (with possible multiple labels):
  - Given labels for children, compute label for the parent:

    A T A T G

    A A T T G $\rightarrow$ A AT AT T G
  - Go position by position:
    - ★ If overlap, use that label.
    - ★ If no overlap, use the union.
  - Useful Python container type: `set`

# Fitch's Algorithm: Pseudocode



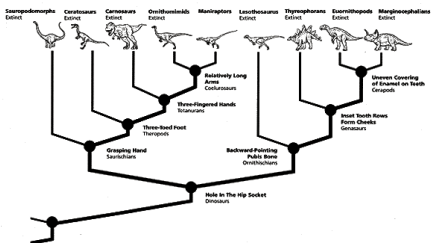

AMNH

- First pass: Starting at the leaves, label the internal leaves (with possible multiple labels):
  - Given labels for children, compute label for the parent:

    A T A T G

    A A T T G $\rightarrow$ A AT AT T G
  - Go position by position:
    - ★ If overlap, use that label.
    - ★ If no overlap, use the union.
  - Useful Python container type: `set`
    - ★ Has functions for union and intersection of sets.

# Fitch's Algorithm: Pseudocode



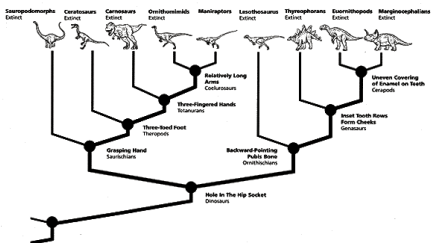

AMNH

- First pass: Starting at the leaves, label the internal leaves (with possible multiple labels):
  - Given labels for children, compute label for the parent:

    A T A T G

    A A T T G $\rightarrow$ A AT AT T G
  - Go position by position:
    - ★ If overlap, use that label.
    - ★ If no overlap, use the union.
  - Useful Python container type: `set`
    - ★ Has functions for union and intersection of sets.
    - ★
```
s1 = set(l1)
s2 = set(l2)
```

# Fitch's Algorithm: Pseudocode



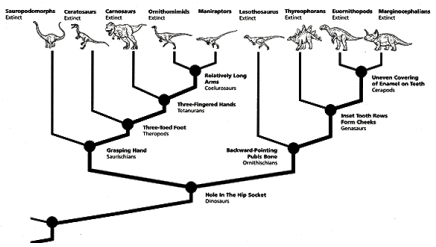

AMNH

- First pass: Starting at the leaves, label the internal leaves (with possible multiple labels):
  - Given labels for children, compute label for the parent:

    A T A T G

    A A T T G $\rightarrow$ A AT AT T G
  - Go position by position:
    - ★ If overlap, use that label.
    - ★ If no overlap, use the union.
  - Useful Python container type: `set`
    - ★ Has functions for union and intersection of sets.
    - ★
```
s1 = set(l1)
s2 = set(l2)
print s1.intersection(s2)
```

# Fitch's Algorithm: Pseudocode



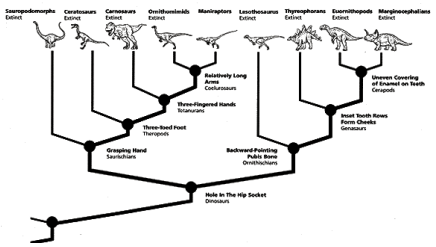

AMNH

- First pass: Starting at the leaves, label the internal leaves (with possible multiple labels):
  - Given labels for children, compute label for the parent:

    A T A T G

    A A T T G $\rightarrow$ A AT AT T G
  - Go position by position:
    - ★ If overlap, use that label.
    - ★ If no overlap, use the union.
  - Useful Python container type: `set`
    - ★ Has functions for union and intersection of sets.
    - ★
```
s1 = set(l1)
s2 = set(l2)
print s1.intersection(s2)
print s1.union(s2)
```

# Fitch's Algorithm: Pseudocode



AMNH

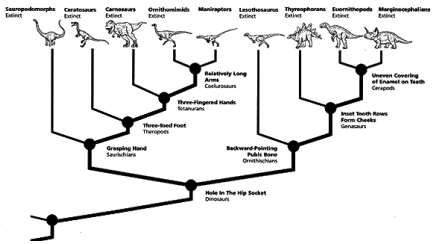
- First pass: Starting at the leaves, label the internal leaves (with possible multiple labels):
  - Given labels for children, compute label for the parent:

    A T A T G

    A A T T G $\rightarrow$ A AT AT T G
  - Go position by position: for-loop
    - ★ If overlap, use that label. if-statement
    - ★ If no overlap, use the union.
  - Useful Python container type: `set`
    - ★ Has functions for union and intersection of sets.
    - ★ `s1 = set(l1)` set operations

      ```
      s2 = set(l2)
      print s1.intersection(s2)
      print s1.union(s2)
      ```

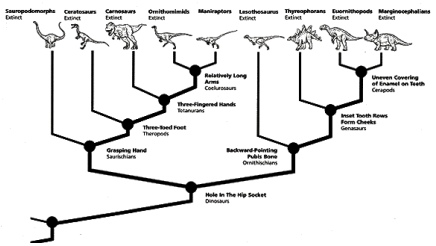
# Fitch's Algorithm: Pseudocode




AMNH

- Second pass: Starting at the root, choose a labeling, then work towards the leaves minimizing the conflicts.

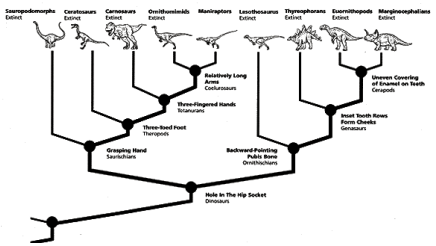
# Fitch's Algorithm: Pseudocode




AMNH

- Second pass: Starting at the root, choose a labeling, then work towards the leaves minimizing the conflicts.
  - At root, choose one labeling:

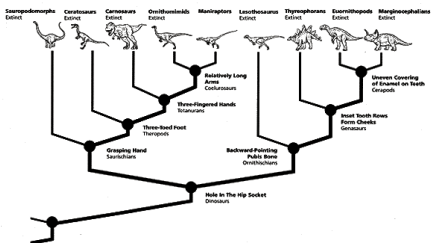
# Fitch's Algorithm: Pseudocode




AMNH

- Second pass: Starting at the root, choose a labeling, then work towards the leaves minimizing the conflicts.
  - At root, choose one labeling:
    `A AT AT T G` $\rightarrow$ `A A T T G`
  - For all other nodes, compare to the parent:

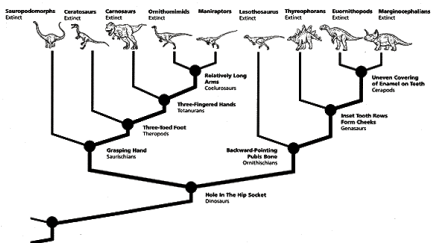
# Fitch's Algorithm: Pseudocode




AMNH

- Second pass: Starting at the root, choose a labeling, then work towards the leaves minimizing the conflicts.
  - At root, choose one labeling:
    A AT AT T G $\rightarrow$ A A T T G
  - For all other nodes, compare to the parent:
    A T A T G
    AT AT G ACT G

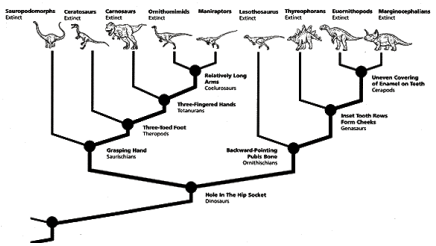
# Fitch's Algorithm: Pseudocode




AMNH

- Second pass: Starting at the root, choose a labeling, then work towards the leaves minimizing the conflicts.
  - At root, choose one labeling:
    A AT AT T G $\rightarrow$ A A T T G
  - For all other nodes, compare to the parent:
    A T A T G
    AT AT G ACT G $\rightarrow$ A T G T G

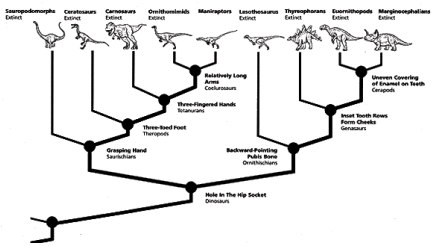
# Fitch's Algorithm: Pseudocode




AMNH

- Second pass: Starting at the root, choose a labeling, then work towards the leaves minimizing the conflicts.
  - At root, choose one labeling:
    A AT AT T G $\rightarrow$ A A T T G
  - For all other nodes, compare to the parent:
    A T A T G
    AT AT G ACT G $\rightarrow$ A T G T G
  - Go position by position:

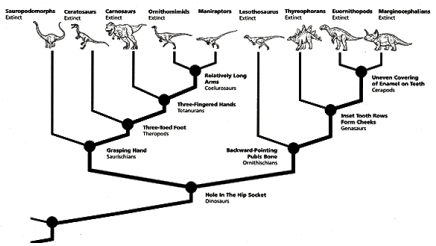
# Fitch's Algorithm: Pseudocode




AMNH

- Second pass: Starting at the root, choose a labeling, then work towards the leaves minimizing the conflicts.
  - At root, choose one labeling:
    A AT AT T G $\rightarrow$ A A T T G
  - For all other nodes, compare to the parent:
    A T A T G
    AT AT G ACT G $\rightarrow$ A T G T G
  - Go position by position:
    - ★ If overlap, use that label.
    - ★ If no overlap, choose label from child.

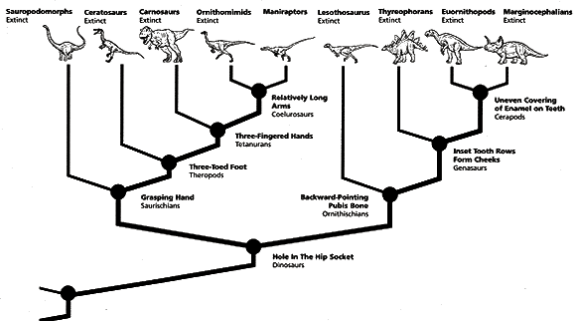
# Fitch's Algorithm: Pseudocode




AMNH

- Second pass: Starting at the root, choose a labeling, then work towards the leaves minimizing the conflicts.
  - At root, choose one labeling:
    A AT AT T G $\rightarrow$ A A T T G
  - For all other nodes, compare to the parent:
    A T A T G
    AT AT G ACT G $\rightarrow$ A T G T G
  - Go position by position: for-loop
    - ★ If overlap, use that label. if-statement
    - ★ If no overlap, choose label from child.

# In Pairs: Fitch's Algorithm: Pseudocode

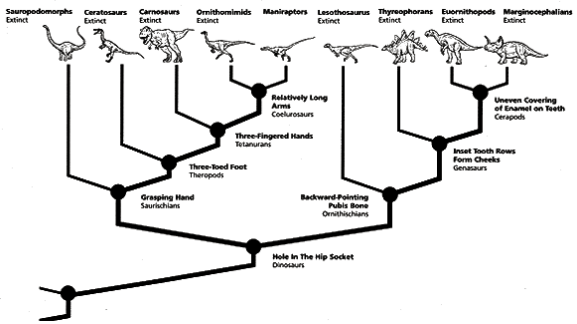



AMNH

Write out the pseudocode for:

1. taking two children labels and computing the parent label.

# In Pairs: Fitch's Algorithm: Pseudocode

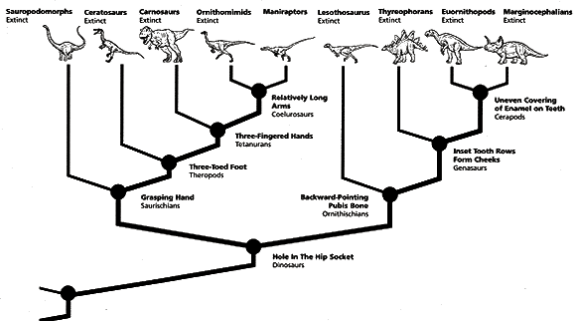



AMNH

Write out the pseudocode for:

1. taking two children labels and computing the parent label.
2. taking a root (possibly multiple at each site) and give a proper label.

# In Pairs: Fitch's Algorithm: Pseudocode

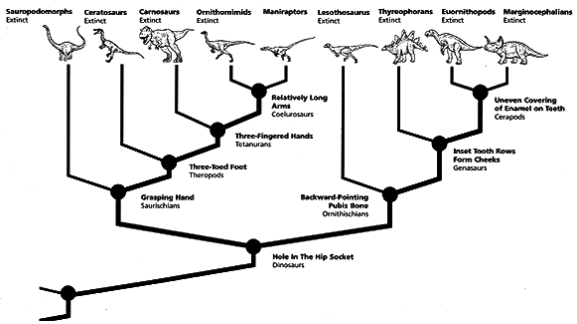



AMNH

Write out the pseudocode for:

1. taking two children labels and computing the parent label.
2. taking a root (possibly multiple at each site) and give a proper label.
3. taking a node and its parent, and give a proper label for child.

# In Pairs: Fitch's Algorithm: Pseudocode



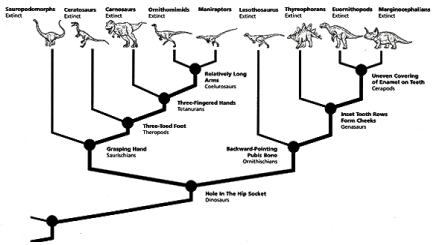

AMNH

Write out the pseudocode for:

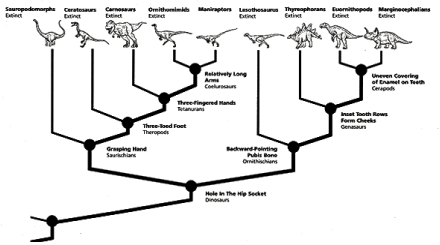
1. taking two children labels and computing the parent label.
2. taking a root (possibly multiple at each site) and give a proper label.
3. taking a node and its parent, and give a proper label for child.
4. pulling all these pieces together.

# Ways to Build Trees

- Often need a tree to start.

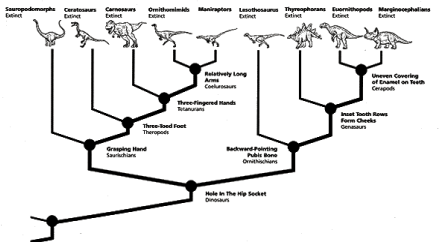


# Ways to Build Trees



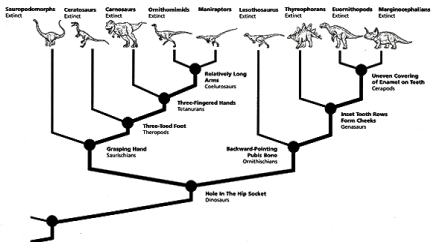

- Often need a tree to start.
- Might not be optimal, but a reasonable approximation quickly:

# Ways to Build Trees



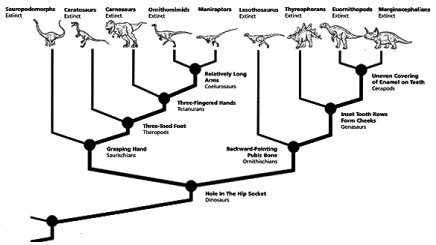

- Often need a tree to start.
- Might not be optimal, but a reasonable approximation quickly:
  - Wagner Build: Greedily grow a tree by adding leaves that minimize the current score.

# Ways to Build Trees




- Often need a tree to start.
- Might not be optimal, but a reasonable approximation quickly:
  - Wagner Build: Greedily grow a tree by adding leaves that minimize the current score.
  - UPGMA: Builds clusters based on Hamming distances between sequences. Assumes an 'ultrametric' clock.

# Ways to Build Trees



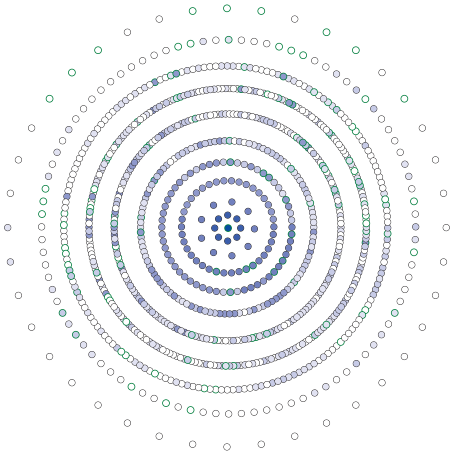

- Often need a tree to start.
- Might not be optimal, but a reasonable approximation quickly:
  - Wagner Build: Greedily grow a tree by adding leaves that minimize the current score.
  - UPGMA: Builds clusters based on Hamming distances between sequences. Assumes an 'ultrametric' clock.
  - Neighbor Joining: Also uses distances but uses 'corrected' distances to produce better trees.

# In Pairs: Wagner Build

AACCGTTA
AACGTTAA
ACCGTTTA
CACGTTAA
AACGGTAT

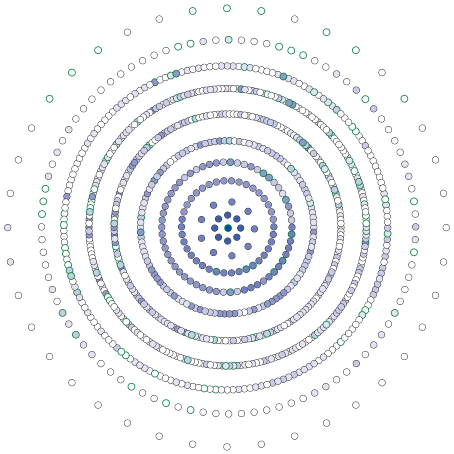
1. Start with a rooted tree on the first two leaves.
2. Score (& label) the tree.
3. For each of the remaining leaves:
   - Try adding it to each edge of the tree.
   - Take the tree with the best score

# Recap



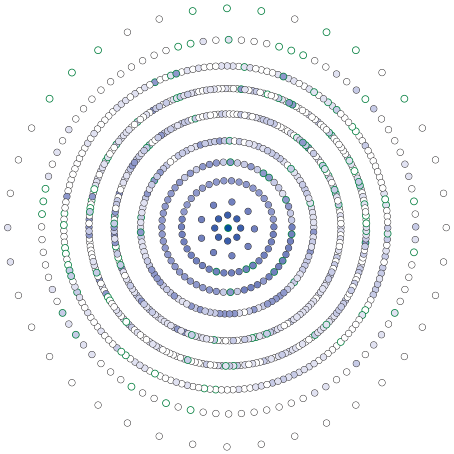
- "Small Parsimony" Problem: given a tree and leaf labels, can score the tree efficiently.

# Recap



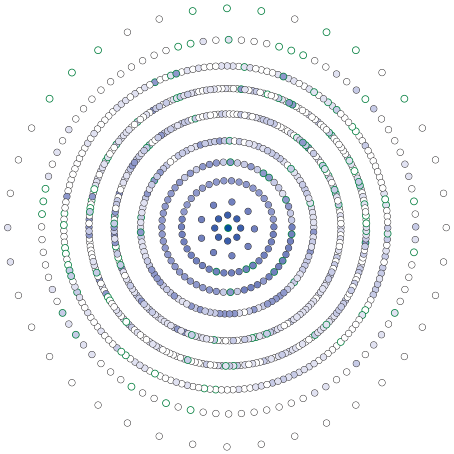
- "Small Parsimony" Problem: given a tree and leaf labels, can score the tree efficiently.
- Next week: "Large Parsimony" Problem: given leaf labels, find optimal tree.

# Recap



- "Small Parsimony" Problem: given a tree and leaf labels, can score the tree efficiently.
- Next week: "Large Parsimony" Problem: given leaf labels, find optimal tree.
- Email lab reports to `kstjohn@amnh.org`.

# Recap



- "Small Parsimony" Problem: given a tree and leaf labels, can score the tree efficiently.
- Next week: "Large Parsimony" Problem: given leaf labels, find optimal tree.
- Email lab reports to `kstjohn@amnh.org`.
- Challenges available at `rosalind.info`.