Algorithmic Approaches for Biological Data, Lecture #24

Katherine St. John

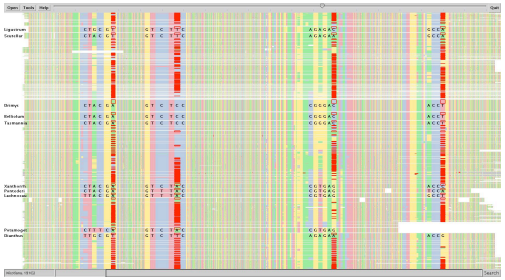
City University of New York
American Museum of Natural History

4 May 2016

Outline

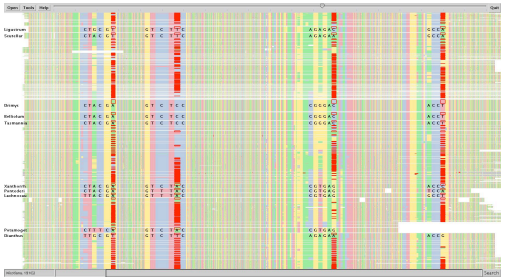
- Approximation Algorithms: UPGMA & Neighbor Joining
- Searching for Optimal Trees
- Early Project Presentations

Approximation Algorithms Using Distance Matrices



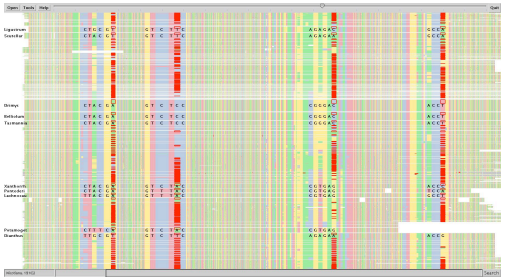
- Idea:
  - Compute Hamming distance between sequences.

Approximation Algorithms Using Distance Matrices



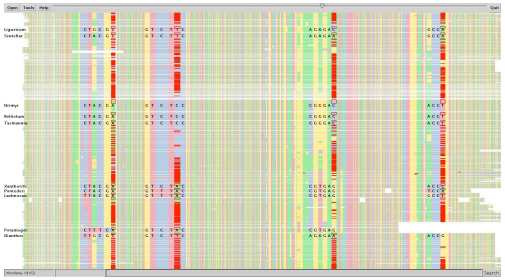
- Idea:
  - Compute Hamming distance between sequences.
  - Base a tree only on the distances.

Approximation Algorithms Using Distance Matrices



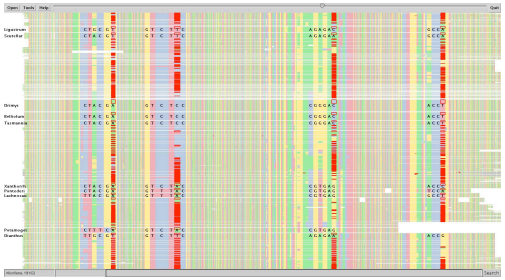
- Idea:
  - Compute Hamming distance between sequences.
  - Base a tree only on the distances.
- Common Algorithms:

Approximation Algorithms Using Distance Matrices



- Idea:
  - Compute Hamming distance between sequences.
  - Base a tree only on the distances.
- Common Algorithms:
  - Unweighted Pair Group Method with Arithmetic Mean (UPGMA) (Sokal & Michener 1958)

Approximation Algorithms Using Distance Matrices



- Idea:
  - Compute Hamming distance between sequences.
  - Base a tree only on the distances.
- Common Algorithms:
  - Unweighted Pair Group Method with Arithmetic Mean (UPGMA) (Sokal & Michener 1958)
  - Neighbor Joining (Saitou-Nei 1987)

UPGMA

Unweighted Pair Group Method with Arithmetic Mean (UPGMA) (Sokal & Michener 1958)

UPGMA

Unweighted Pair Group Method with Arithmetic Mean (UPGMA) (Sokal & Michener 1958)

- Compute the distance between every pair of leaves.

UPGMA

Unweighted Pair Group Method with Arithmetic Mean (UPGMA) (Sokal & Michener 1958)

- Compute the distance between every pair of leaves.
- Make a cluster for each leaf.
- Join the two clusters with lowest distance.

UPGMA

Unweighted Pair Group Method with Arithmetic Mean (UPGMA) (Sokal & Michener 1958)

- Compute the distance between every pair of leaves.
- Make a cluster for each leaf.
- Join the two clusters with lowest distance.
- Average the weights to get the new distances to other clusters.

UPGMA

Unweighted Pair Group Method with Arithmetic Mean (UPGMA) (Sokal & Michener 1958)

- Compute the distance between every pair of leaves.
- Make a cluster for each leaf.
- Join the two clusters with lowest distance.
- Average the weights to get the new distances to other clusters.

$$d(A \cup B, X) = \frac{\sum_{x \in X} \sum_{a \in A} d(x, a) + \sum_{x \in X} \sum_{b \in B} d(x, b)}{|X|(|A| + |B|)}$$

UPGMA

Unweighted Pair Group Method with Arithmetic Mean (UPGMA) (Sokal & Michener 1958)

- Compute the distance between every pair of leaves.
- Make a cluster for each leaf.
- Join the two clusters with lowest distance.
- Average the weights to get the new distances to other clusters.

$$d(A \cup B, X) = \frac{\sum_{x \in X} \sum_{a \in A} d(x, a) + \sum_{x \in X} \sum_{b \in B} d(x, b)}{|X|(|A| + |B|)}$$

- Repeat until only one cluster.

UPGMA

Unweighted Pair Group Method with Arithmetic Mean (UPGMA) (Sokal & Michener 1958)

- Compute the distance between every pair of leaves.
- Make a cluster for each leaf.
- Join the two clusters with lowest distance.
- Average the weights to get the new distances to other clusters.

$$d(A \cup B, X) = \frac{\sum_{x \in X} \sum_{a \in A} d(x, a) + \sum_{x \in X} \sum_{b \in B} d(x, b)}{|X|(|A| + |B|)}$$

- Repeat until only one cluster.

	a	b	c	d	e
a	0	17	21	31	23
b	17	0	30	34	21
c	21	30	0	28	39
d	31	34	28	0	43
e	23	21	39	43	0

UPGMA

Unweighted Pair Group Method with Arithmetic Mean (UPGMA) (Sokal & Michener 1958)

- Compute the distance between every pair of leaves.
- Make a cluster for each leaf.
- Join the two clusters with lowest distance.
- Average the weights to get the new distances to other clusters.

$$d(A \cup B, X) = \frac{\sum_{x \in X} \sum_{a \in A} d(x, a) + \sum_{x \in X} \sum_{b \in B} d(x, b)}{|X|(|A| + |B|)}$$

- Repeat until only one cluster.

	a	b	c	d	e
a	0	17	21	31	23
b	17	0	30	34	21
c	21	30	0	28	39
d	31	34	28	0	43
e	23	21	39	43	0

	(a,b)	c	d	e
(a,b)	0	25.5	32.5	22
c	25.5	0	*28*	*39*
d	32.5	*28*	0	*43*
e	22	*39*	*43*	0

UPGMA

Unweighted Pair Group Method with Arithmetic Mean (UPGMA) (Sokal & Michener 1958)

- Compute the distance between every pair of leaves.
- Make a cluster for each leaf.
- Join the two clusters with lowest distance.
- Average the weights to get the new distances to other clusters.

$$d(A \cup B, X) = \frac{\sum_{x \in X} \sum_{a \in A} d(x, a) + \sum_{x \in X} \sum_{b \in B} d(x, b)}{|X|(|A| + |B|)}$$

- Repeat until only one cluster.

	a	b	c	d	e
a	0	17	21	31	23
b	17	0	30	34	21
c	21	30	0	28	39
d	31	34	28	0	43
e	23	21	39	43	0

	(a,b)	c	d	e
(a,b)	0	25.5	32.5	22
c	25.5	0	*28*	*39*
d	32.5	*28*	0	*43*
e	22	*39*	*43*	0

	((a,b),e)	c	d
((a,b),e)	0	30	36
c	30	0	*28*
d	36	*28*	0

UPGMA

Unweighted Pair Group Method with Arithmetic Mean (UPGMA) (Sokal & Michener 1958)

- Compute the distance between every pair of leaves.
- Make a cluster for each leaf.
- Join the two clusters with lowest distance.
- Average the weights to get the new distances to other clusters.

$$d(A \cup B, X) = \frac{\sum_{x \in X} \sum_{a \in A} d(x, a) + \sum_{x \in X} \sum_{b \in B} d(x, b)}{|X|(|A| + |B|)}$$

- Repeat until only one cluster.

	a	b	c	d	e
a	0	17	21	31	23
b	17	0	30	34	21
c	21	30	0	28	39
d	31	34	28	0	43
e	23	21	39	43	0

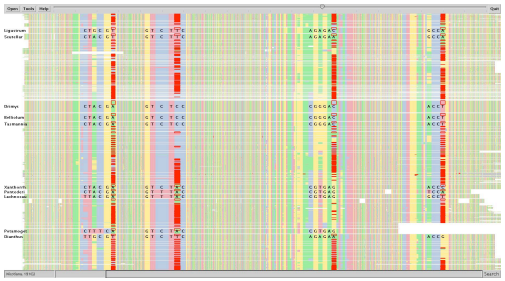
	(a,b)	c	d	e
(a,b)	0	25.5	32.5	22
c	25.5	0	*28*	*39*
d	32.5	*28*	0	*43*
e	22	*39*	*43*	0

	((a,b),e)	c	d
((a,b),e)	0	30	36
c	30	0	*28*
d	36	*28*	0

	((a,b),e)	(c,d)
((a,b),e)	0	33
(c,d)	33	0

wiki

Neighbor Joining

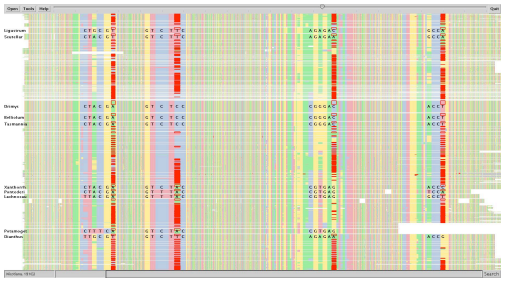


Neighbor Joining (Saitou & Nei 1987)

- A similar idea, but a different matrix is computed:

$$Q(i,j) = (n-2)d(i,j) - \sum_{k=1}^{n} d(i,k) - \sum_{k=1}^{n} d(j,k)$$

Neighbor Joining



Neighbor Joining (Saitou & Nei 1987)

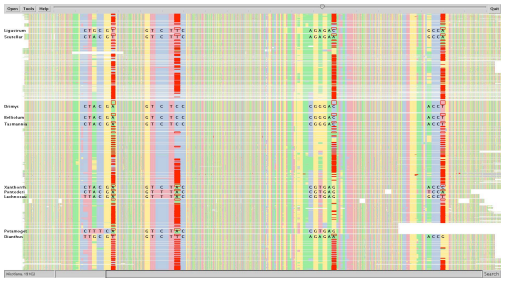
- A similar idea, but a different matrix is computed:

$$Q(i,j) = (n-2)d(i,j) - \sum_{k=1}^{n} d(i,k) - \sum_{k=1}^{n} d(j,k)$$

- If A and B are joined to new node u, then the distance from k to u

$$d(u,k) = \frac{1}{2}[d(A,k) + d(B,k) - d(A,B)]$$

Neighbor Joining



Neighbor Joining (Saitou & Nei 1987)

- A similar idea, but a different matrix is computed:

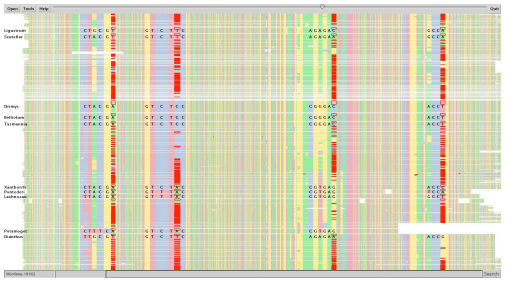
$$Q(i,j) = (n-2)d(i,j) - \sum_{k=1}^{n} d(i,k) - \sum_{k=1}^{n} d(j,k)$$

- If A and B are joined to new node u, then the distance from k to u

$$d(u,k) = \frac{1}{2}[d(A,k) + d(B,k) - d(A,B)]$$

- This relaxes the assumption of UPGMA that all lineages evolve at the same rate.

Neighbor Joining



Neighbor Joining (Saitou & Nei 1987)

- A similar idea, but a different matrix is computed:

$$Q(i,j) = (n-2)d(i,j) - \sum_{k=1}^{n} d(i,k) - \sum_{k=1}^{n} d(j,k)$$

- If A and B are joined to new node u, then the distance from k to u

$$d(u,k) = \frac{1}{2}[d(A,k) + d(B,k) - d(A,B)]$$

- This relaxes the assumption of UPGMA that all lineages evolve at the same rate.

In Pairs: Distance Methods

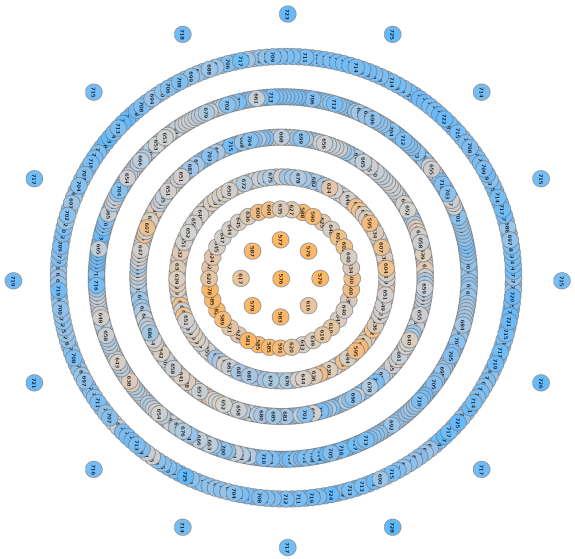
	a	b	c	d	e
a	0	5	9	9	8
b	5	0	10	10	9
c	9	10	0	8	7
d	9	10	8	0	3
e	8	9	7	3	0

wiki

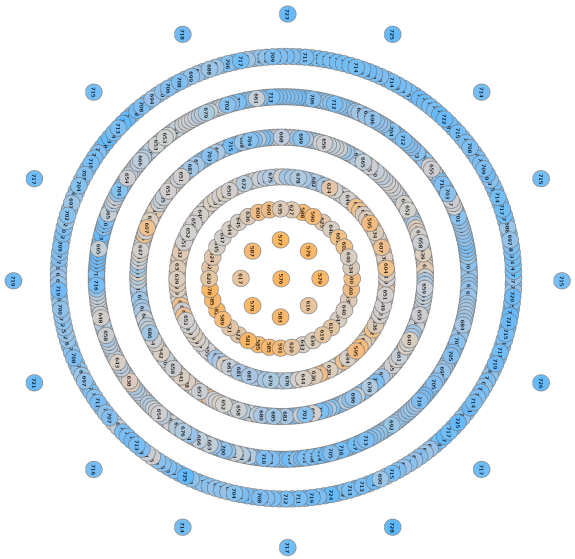
1. If you run the UPGMA algorithm on the matrix, what is the resulting tree?
2. What is an upper bound on the running time of UPGMA?
3. What is an upper bound on the running time of Neighbor Joining?

Searching Treespace

```
treeSearch(sequences):
```

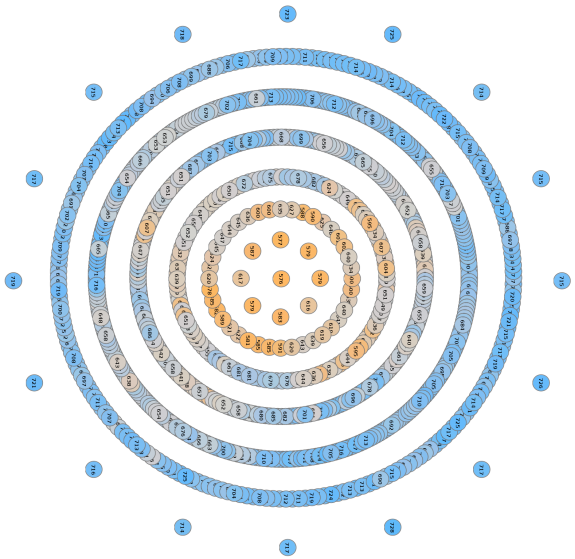


Searching Treespace



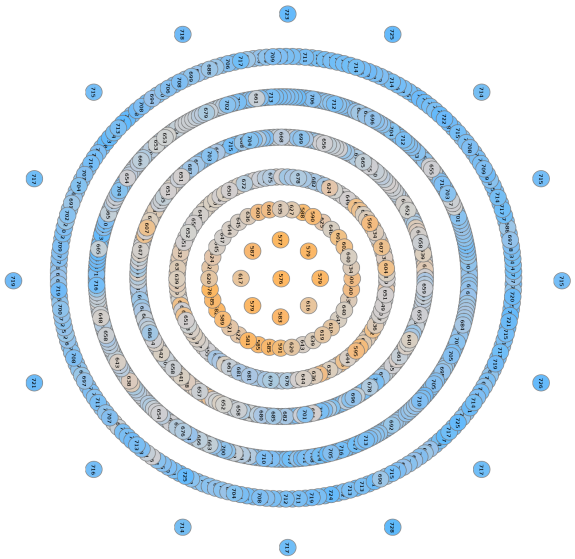
`treeSearch(sequences)`:
Input: A set of n sequences of length k

Searching Treespace



`treeSearch(sequences)`:
Input: A set of n sequences of length k
Output: The best scoring tree we could find after 1000 steps through the NNI search space.

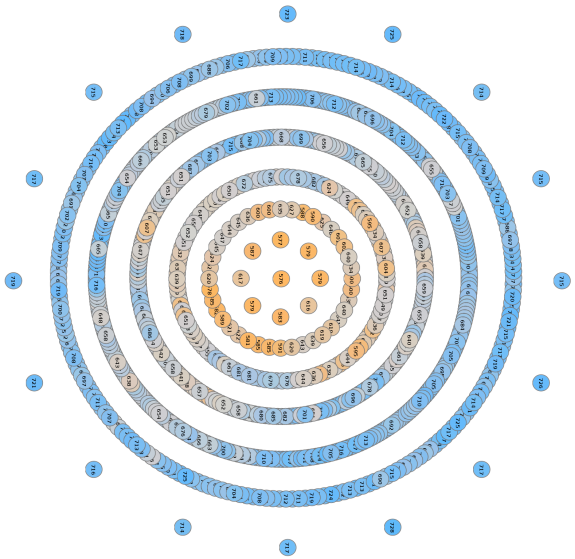
Searching Treespace



`treeSearch(sequences)`:
Input: A set of n sequences of length k
Output: The best scoring tree we could find after 1000 steps through the NNI search space.

1. Choose randomly a tree to be `bestSoFar`.

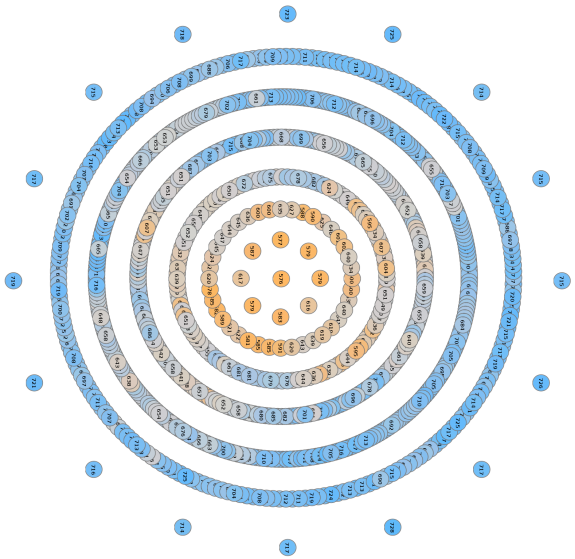
Searching Treespace



`treeSearch(sequences)`:
Input: A set of n sequences of length k
Output: The best scoring tree we could find after 1000 steps through the NNI search space.

1. Choose randomly a tree to be `bestSoFar`.
2. Score the tree, `bestSoFar`.

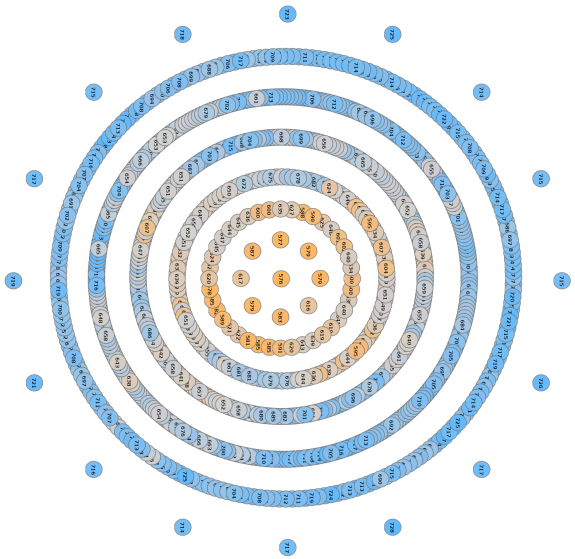
Searching Treespace



`treeSearch(sequences)`:
Input: A set of n sequences of length k
Output: The best scoring tree we could find after 1000 steps through the NNI search space.

1. Choose randomly a tree to be `bestSoFar`.
2. Score the tree, `bestSoFar`.
3. For 1000 steps:

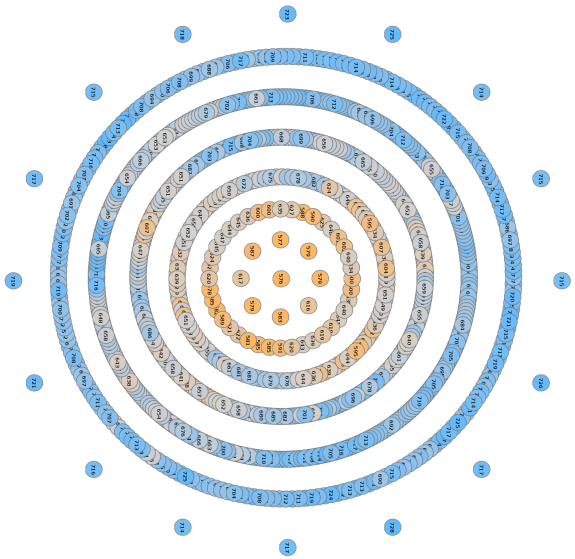
Searching Treespace



`treeSearch(sequences)`:
Input: A set of n sequences of length k
Output: The best scoring tree we could find after 1000 steps through the NNI search space.

1. Choose randomly a tree to be `bestSoFar`.
2. Score the tree, `bestSoFar`.
3. For 1000 steps:
4. if any of the NNI neighbors of `bestSoFar` have better score,

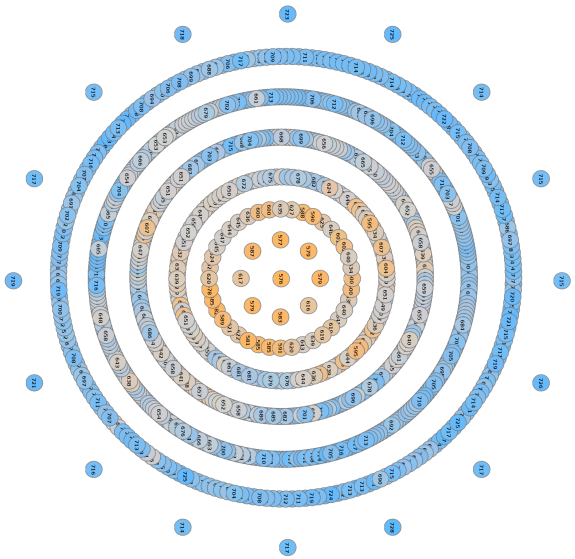
Searching Treespace



`treeSearch(sequences)`:
Input: A set of n sequences of length k
Output: The best scoring tree we could find after 1000 steps through the NNI search space.

1. Choose randomly a tree to be `bestSoFar`.
2. Score the tree, `bestSoFar`.
3. For 1000 steps:
4. if any of the NNI neighbors of `bestSoFar` have better score,
5. choose it to be the current tree

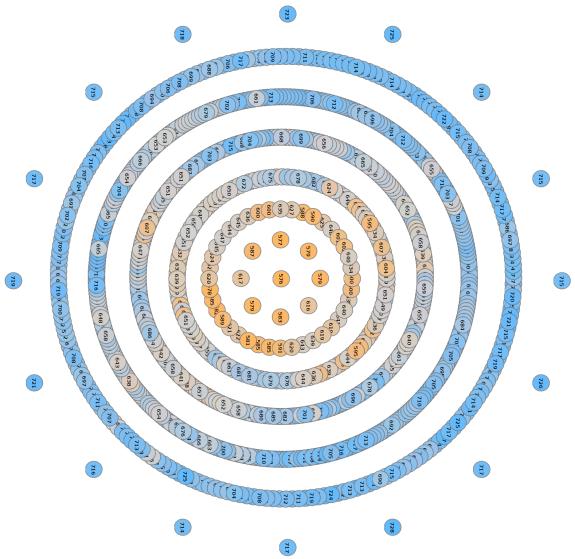
Searching Treespace



`treeSearch(sequences)`:
Input: A set of n sequences of length k
Output: The best scoring tree we could find after 1000 steps through the NNI search space.

1. Choose randomly a tree to be `bestSoFar`.
2. Score the tree, `bestSoFar`.
3. For 1000 steps:
4. if any of the NNI neighbors of `bestSoFar` have better score,
5. choose it to be the current tree
6. else:
7. break (stuck in local minima)

Searching Treespace

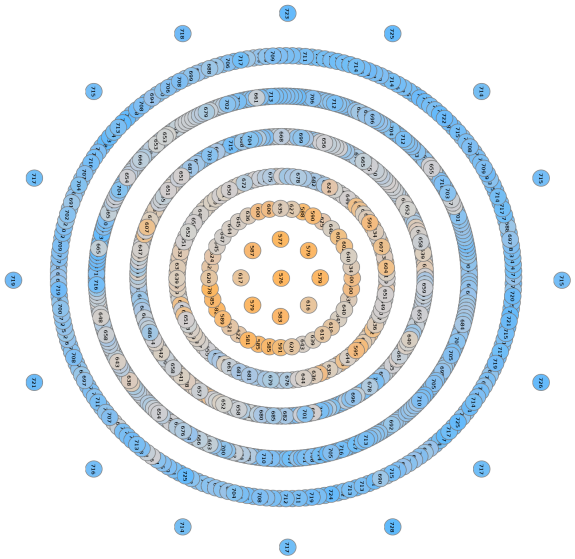


`treeSearch(sequences)`:
Input: A set of n sequences of length k
Output: The best scoring tree we could find after 1000 steps through the NNI search space.

1. Choose randomly a tree to be `bestSoFar`.
2. Score the tree, `bestSoFar`.
3. For 1000 steps:
4. if any of the NNI neighbors of `bestSoFar` have better score,
5. choose it to be the current tree
6. else:
7. break (stuck in local minima)
8. Print number of steps, tree, and score.

In Pairs: Translating "To Do" into Pseudocode

Translate into pseudocode:

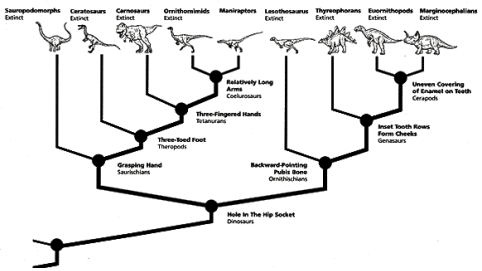


`treeSearch(sequences)`:
Input: A set of n sequences of length k
Output: The best scoring tree we could find after 1000 steps through the NNI search space.

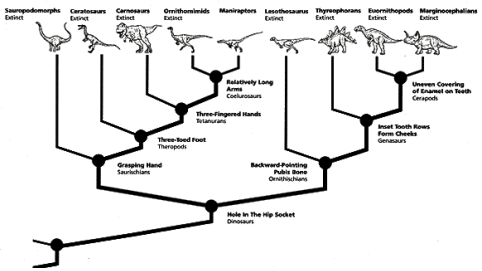
1. Choose randomly a tree to be `bestSoFar`.
2. Score the tree, `bestSoFar`.
3. For 1000 steps:
4. if any of the NNI neighbors of `bestSoFar` have better score,
5. choose it to be the current tree
6. else:
7. break (stuck in local minima)
8. Print number of steps, tree, and score.

Randomly Building A Tree

1. Put all leaves into a dictionary.

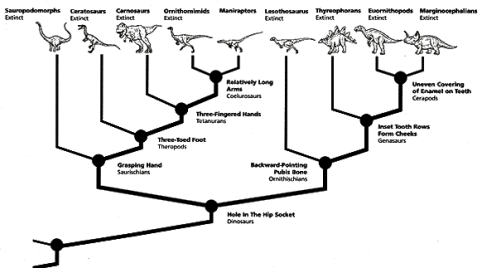


Randomly Building A Tree



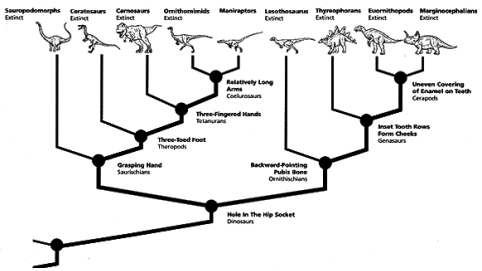
1. Put all leaves into a dictionary.
2. Also create a "To Do" list, `parentless`, of nodes without parents.

Randomly Building A Tree



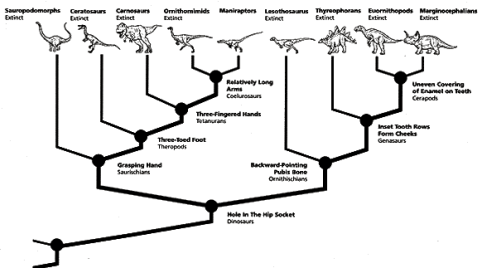
1. Put all leaves into a dictionary.
2. Also create a "To Do" list, `parentless`, of nodes without parents.
3. While `len(parentless) > 1`:

Randomly Building A Tree



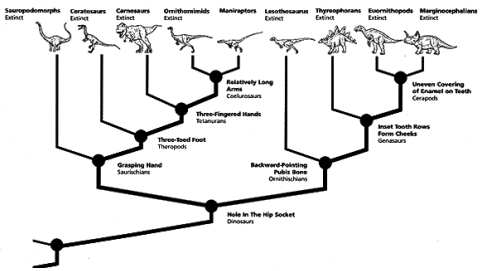
1. Put all leaves into a dictionary.
2. Also create a "To Do" list, `parentless`, of nodes without parents.
3. While `len(parentless) > 1`:
4. Make a new internal node and choose two from `parentless` to be its children.

Randomly Building A Tree



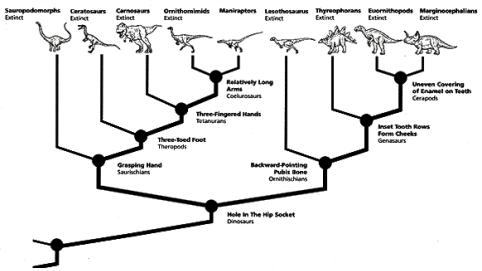
1. Put all leaves into a dictionary.
2. Also create a "To Do" list, `parentless`, of nodes without parents.
3. While `len(parentless) > 1`:
4. Make a new internal node and choose two from `parentless` to be its children.
5. Remove children from `parentless`.

Randomly Building A Tree



1. Put all leaves into a dictionary.
2. Also create a "To Do" list, `parentless`, of nodes without parents.
3. While `len(parentless) > 1`:
4. Make a new internal node and choose two from `parentless` to be its children.
5. Remove children from `parentless`.
6. Add new node to `parentless`.

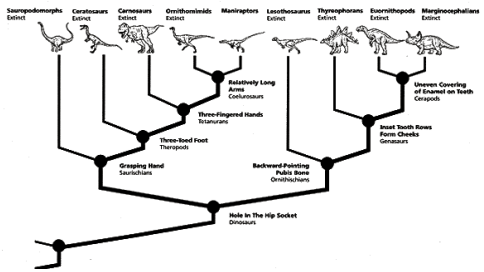
Randomly Building A Tree



1. Put all leaves into a dictionary.
2. Also create a "To Do" list, `parentless`, of nodes without parents.
3. While `len(parentless) > 1`:
4. Make a new internal node and choose two from `parentless` to be its children.
5. Remove children from `parentless`.
6. Add new node to `parentless`.
7. Rename remaining node in `parentless` to be `root`.

In Pairs: Randomly Building A Tree

Translate into pseudocode:



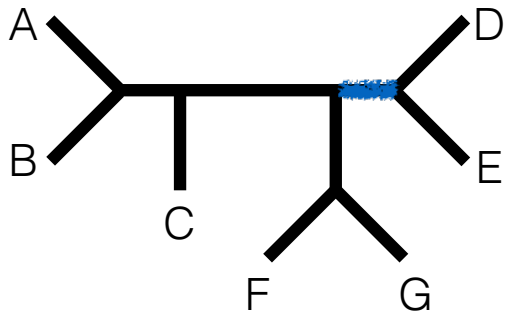

```
def randomBuild(sequenceList):
```

1. Put all leaves into a dictionary.
2. Also create a "To Do" list, `parentless`, of nodes without parents.
3. While `len(parentless) > 1`:
4. Make a new internal node and choose two from `parentless` to be its children.
5. Remove children from `parentless`.
6. Add new node to `parentless`.
7. Rename remaining node in `parentless` to be `root`.

Making NNI Move

Input: A tree, `t`, and an internal node, `node`.
Output: Two trees that result from switching the nodes' sibling with its children.

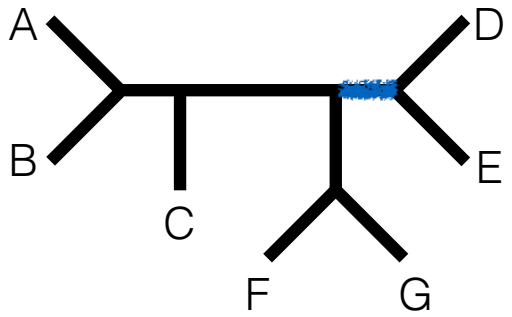
1. Let `sib` be the sibling of `node`.



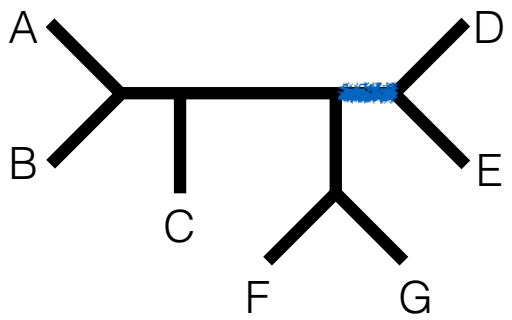
Making NNI Move

Input: A tree, `t`, and an internal node, `node`.
Output: Two trees that result from switching the nodes' sibling with its children.

1. Let `sib` be the sibling of `node`.
2. Let `kid1`, `kid2` be the children of `node`.



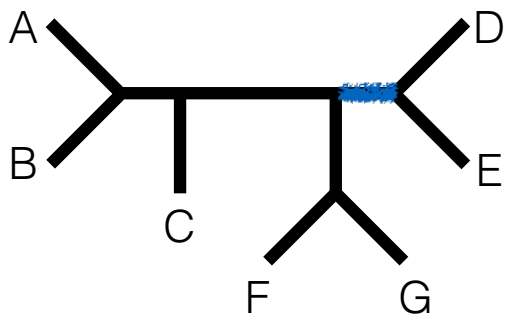
Making NNI Move



Input: A tree, `t`, and an internal node, `node`.
Output: Two trees that result from switching the nodes' sibling with its children.

1. Let `sib` be the sibling of `node`.
2. Let `kid1`, `kid2` be the children of `node`.
3. Make two (deep) copies of `t`: `tree1` and `tree2`

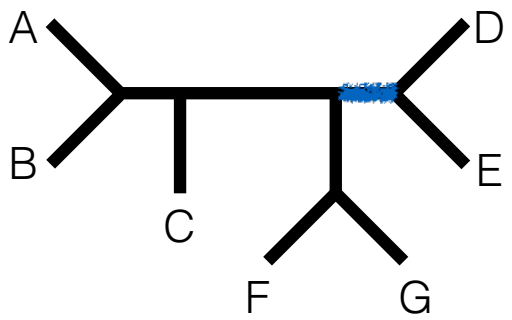
Making NNI Move



Input: A tree, `t`, and an internal node, `node`.
Output: Two trees that result from switching the nodes' sibling with its children.

1. Let `sib` be the sibling of `node`.
2. Let `kid1`, `kid2` be the children of `node`.
3. Make two (deep) copies of `t`: `tree1` and `tree2`
4. Switch `sib` and `kid1` in `tree1`.

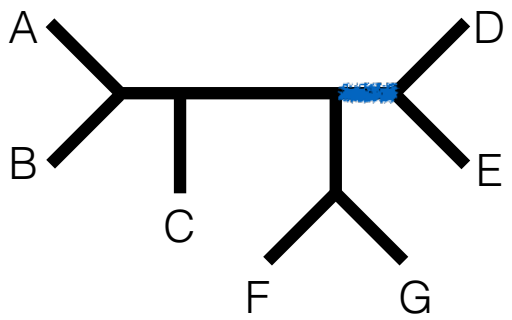
Making NNI Move



Input: A tree, `t`, and an internal node, `node`.
Output: Two trees that result from switching the nodes' sibling with its children.

1. Let `sib` be the sibling of `node`.
2. Let `kid1`, `kid2` be the children of `node`.
3. Make two (deep) copies of `t`: `tree1` and `tree2`
4. Switch `sib` and `kid1` in `tree1`.
5. Switch `sib` and `kid2` in `tree2`.

Making NNI Move

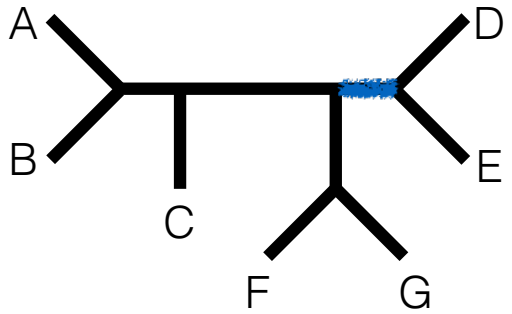


Input: A tree, `t`, and an internal node, `node`.
Output: Two trees that result from switching the nodes' sibling with its children.

1. Let `sib` be the sibling of `node`.
2. Let `kid1`, `kid2` be the children of `node`.
3. Make two (deep) copies of `t`: `tree1` and `tree2`
4. Switch `sib` and `kid1` in `tree1`.
5. Switch `sib` and `kid2` in `tree2`.
6. Return `tree1` and `tree2`.

In Pairs: Making NNI Move

Translate into pseudocode:

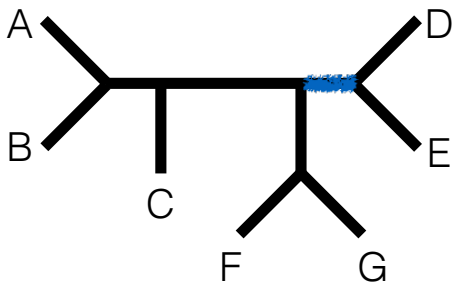


`def nniNeighbors(t, node):`
Input: A tree, `t`, and an internal node, `node`.
Output: Two trees that result from switching the nodes' sibling with its children.

1. Let `sib` be the sibling of `node`.
2. Let `kid1`, `kid2` be the children of `node`.
3. Make two (deep) copies of `t`: `tree1` and `tree2`
4. Switch `sib` and `kid1` in `tree1`.
5. Switch `sib` and `kid2` in `tree2`.
6. Return `tree1` and `tree2`.

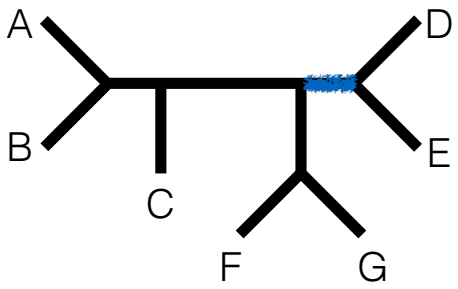
Early Project Presentaitons

Recap



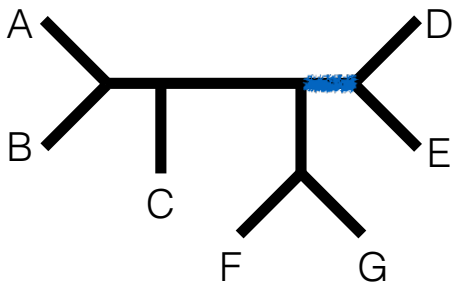
- Trees can be estimated used distance based methods.

Recap



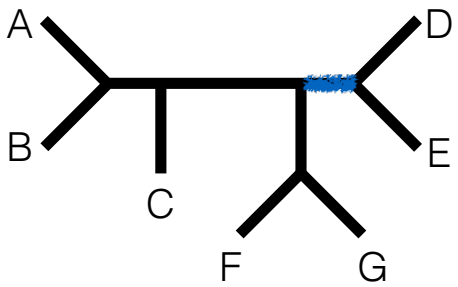
- Trees can be estimated used distance based methods.
- More on searching treespace in lab today.

Recap



- Trees can be estimated used distance based methods.
- More on searching treespace in lab today.
- Email lab reports to `kstjohn@amnh.org`.

Recap



- Trees can be estimated used distance based methods.
- More on searching treespace in lab today.
- Email lab reports to `kstjohn@amnh.org`.
- Challenges available at `rosalind.info`.