Distance based tree reconstruction

Hierarchical clustering (UPGMA)
Neighbor-Joining (NJ)
All organisms have evolved from a common ancestor.

Infer the **evolutionary tree** (tree topology and edge lengths) from molecular data.

A **gene tree** inferred from a homologous gene from different species might be different from the **species tree**.
Distance based reconstruction

**Goal:** Reconstruct an evolutionary tree from a distance matrix

**Input:** $n \times n$ distance matrix $d$

**Output:** weighted tree $T$ with $n$ leaves fitting $d$

If $d$ is additive, this problem has an efficient solution (which we will come back to), otherwise we should aim at finding a tree which is “as good as possible” …
Distance based reconstruction

The “ideal” case

The tree reflects the distance matrix, i.e. the distances induced by the tree are equal to the distances in the matrix.

If this is possible, then the distance matrix is called **additive**.
A reconstruction method

Least square methods

Given a tree topology $T$, select edge lengths such that

$$Q(T) = \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} (D_{ij} - d_{ij})^2$$

is minimized, where $d_{ij}$ is given by the matrix, $D_{ij}$ is the distance induced by the tree and the selected edge lengths, and $w_{ij}$ is the "confidence" we have in $d_{ij}$, e.g. $w_{ij}=1$.

Optimal edge lengths can be found by standard least square methods in time $O(n^3)$. The "large" version of the problem when the tree is unknown is NP-complete ...
## Counting rooted binary trees

<table>
<thead>
<tr>
<th>Species</th>
<th>Number of trees</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>15</td>
</tr>
<tr>
<td>5</td>
<td>105</td>
</tr>
<tr>
<td>6</td>
<td>945</td>
</tr>
<tr>
<td>7</td>
<td>10,395</td>
</tr>
<tr>
<td>8</td>
<td>135,135</td>
</tr>
<tr>
<td>9</td>
<td>2,027,025</td>
</tr>
<tr>
<td>10</td>
<td>34,459,425</td>
</tr>
<tr>
<td>11</td>
<td>654,729,075</td>
</tr>
<tr>
<td>12</td>
<td>13,749,310,575</td>
</tr>
<tr>
<td>13</td>
<td>316,234,143,225</td>
</tr>
<tr>
<td>14</td>
<td>7,905,853,580,625</td>
</tr>
<tr>
<td>15</td>
<td>213,458,046,676,875</td>
</tr>
<tr>
<td>16</td>
<td>6,190,283,353,629,375</td>
</tr>
<tr>
<td>17</td>
<td>191,898,783,962,510,625</td>
</tr>
<tr>
<td>18</td>
<td>6,332,659,870,762,850,625</td>
</tr>
<tr>
<td>19</td>
<td>221,643,095,476,699,771,875</td>
</tr>
<tr>
<td>20</td>
<td>8,200,794,532,637,891,559,375</td>
</tr>
<tr>
<td>30</td>
<td>$4.9518 \times 10^{38}$</td>
</tr>
<tr>
<td>40</td>
<td>$1.00985 \times 10^{57}$</td>
</tr>
<tr>
<td>50</td>
<td>$2.75292 \times 10^{76}$</td>
</tr>
</tbody>
</table>
Counting rooted binary trees
Counting rooted binary trees

A rooted binary tree with \( n \) leaves has \( 2n-2 \) edges. A new species can be added to any edge or “above” the root, i.e:

\[
R(n) = R(n-1) \left( 2(n-1) - 2 + 1 \right) = R(n-1) \left( 2n - 3 \right)
\]

What about unrooted trees?

What about non-binary trees?
Hierarchical clustering and UPGMA
Hierarchical clustering

**Input:** A set of $n$ species (1,2,...,n) and a distance matrix $d$ giving the pairwise distances.

**Output:** A rooted binary tree $T$ with edge lengths and leaves labelled 1, 2, ..., $n$ such that similar species (according to the distance matrix) are grouped in the same subtree, and the tree distances correspond to the distance matrix.
Example
Example

Group the closest two data points.
Example

In each iteration, we join the closest two clusters.
Example
Example
Hierarchical clustering

**Input:** A set of $n$ species $(1,2,\ldots,n)$ and a distance matrix $d$ giving the pairwise distances.

**Output:** A rooted binary tree $T$ with edge lengths and leaves labelled $1$, $2$, ..., $n$ such that similar species (according to the distance matrix) are grouped in the same subtree, and the tree distances correspond to the distance matrix.

**Algorithm:**

**Step 1:** Let the $n$ species $1,2,\ldots,n$ be $n$ clusters $C_1,\ldots,C_n$ of size 1. Let each cluster correspond to a leaf in a tree $T$. The distance $D(C_i, C_j)$ between cluster $C_i$ and $C_j$ is $d(i,j)$.

**Step 2:** Pick the pair of cluster $C_i$ and $C_j$ where $D(C_i, C_j)$ is minimized and form a new cluster $C_k$ by merging $C_i$ and $C_j$, i.e. $C_k = C_i \cup C_j$. Make a new internal node $C_k$ in tree $T$ with children corresponding to $C_i$ and $C_j$. Place internal node $C_k$ at height $D(C_i, C_j) / 2$.

**Step 3:** Update distance matrix $D$ such that distance between the new cluster $C_k$ and all remaining clusters are computed.

**Step 4:** Repeat Step 2 and 3 until only one cluster remains.
Hierarchical clustering

**Input:** A set of $n$ species (1,2,...,$n$) and a distance matrix $d$ giving the pairwise distances.

**Output:** A rooted binary tree $T$ with edge lengths and leaves labelled 1, 2, ..., $n$ such that similar species (according to the distance matrix) are grouped in the same subtree, and the tree distances correspond to the distance matrix.

**Algorithm:**

**Step 1:** Let the $n$ species 1,2, ..., $n$ be $n$ clusters $C_1$,...,$C_n$ of size 1. Let each cluster correspond to a leaf in a tree $T$. The distance $D(C_i,C_j)$ between cluster $C_i$ and $C_j$ is $d(i,j)$.

**Step 2:** Pick the pair of cluster $C_i$ and $C_j$ where $D(C_i,C_j)$ is minimized and form a new cluster $C_k$ by merging $C_i$ and $C_j$, i.e. $C_k = C_i \cup C_j$. Make a new internal node $C_k$ in tree $T$ with children corresponding to $C_i$ and $C_j$. Place internal node $C_k$ at height $D(C_i,C_j)/2$.

**Step 3:** Update distance matrix $D$ such that distance between the new cluster $C_k$ and all remaining clusters are computed.

**Step 4:** Repeat Step 2 and 3 until only one cluster remains.

How to define and compute the distances between clusters?
Distance measures between clusters

Single linkage

minimum dist between any two members...
Distance measures between clusters

**Single linkage**

Minimum distance between any two members...

**Complete linkage**

Maximum distance between any two members...
Distance measures between clusters

**Single linkage**

minimum distance between any two members...

**Complete linkage**

maximum distance between any two members...

**Average linkage**

average of all pairwise distances between members of the two clusters

\[
D(C_i, C_j) = \frac{1}{|C_i| \cdot |C_j| - \sum_{p \in C_i, q \in C_j} D(p, q)}
\]
Distance measures between clusters

**Single linkage**

- Minimum dist between any two members...

**Complete linkage**

- Maximum distance between any two members...

**Average linkage**

- Average of all pairwise distances between members of the two clusters

\[
D(C_i, C_j) = \frac{1}{|C_i| \cdot |C_j|} \sum_{p \in C_i, q \in C_j} D(p, q)
\]

If \( C_k = C_i \cup C_j \) then

\[
D(C_k, C_k) = \frac{D(C_i, C_j) \cdot |C_i| + D(C_j, C_j) \cdot |C_j|}{|C_i| + |C_j|}
\]
Hierarchical clustering (UPGMA)

**Input:** A set of \( n \) species \( (1, 2, \ldots, n) \) and a distance matrix \( d \) giving the pairwise distances.

**Output:** A rooted binary tree \( T \) with edge lengths and leaves labelled 1, 2, ..., \( n \) such that similar species (according to the distance matrix) are grouped in the same subtree, and the tree distances correspond to the distance matrix.

**Algorithm:**

**Step 1:** Let the \( n \) species 1, 2, ..., \( n \) be \( n \) clusters \( C_1, \ldots, C_n \) of size 1. Let each cluster correspond to a leaf in a tree \( T \). The distance \( D(C_i, C_j) \) between cluster \( C_i \) and \( C_j \) is \( d(i, j) \).

**Step 2:** Pick the pair of cluster \( C_i \) and \( C_j \) where \( D(C_i, C_j) \) is minimized and form a new cluster \( C_k \) by merging \( C_i \) and \( C_j \), i.e. \( C_k = C_i \cup C_j \). Make a new internal node \( C_k \) in tree \( T \) with children corresponding to \( C_i \) and \( C_j \). Place internal node \( C_k \) at height \( D(C_i, C_j) / 2 \).

**Step 3:** Update distance matrix \( D \) such that distance between the new cluster \( C_k \) and all remaining clusters \( C_i \) are computed as:

\[
D(C_k, C_i) = \frac{D(C_i, C_j) \cdot l_{C_i} + D(C_j, C_k) \cdot l_{C_j}}{l_{C_i} + l_{C_j}}
\]

**Step 4:** Repeat Step 2 and 3 until only one cluster remains.

**Running time? Space consumption?**
Hierarchical clustering (UPGMA)

**Input:** A set of \( n \) species \((1,2,\ldots,n)\) and a distance matrix \( d \) giving the pairwise distances.

**Output:** A rooted binary tree \( T \) with edge lengths and leaves labelled \( 1, 2, \ldots, n \) such that similar species (according to the distance matrix) are grouped in the same subtree, and the tree distances correspond to the distance matrix.

**Algorithm:**

**Step 1:** Let the \( n \) species \( 1,2,\ldots,n \) be \( n \) clusters \( C_1,\ldots,C_n \) of size 1. Let each cluster correspond to a leaf in a tree \( T \). The distance \( D(C_i,C_j) \) between cluster \( C_i \) and \( C_j \) is \( d(i,j) \).

**Step 2:** Pick the pair of cluster \( C_i \) and \( C_j \) where \( D(C_i,C_j) \) is minimized and form a new cluster \( C_k \) by merging \( C_i \) and \( C_j \), i.e. \( C_k = C_i \cup C_j \). Make a new internal node \( C_k \) in tree \( T \) with children corresponding to \( C_i \) and \( C_j \). Place internal node \( C_k \) at height \( D(C_i,C_j) \)/2.

**Step 3:** Update distance matrix \( D \) such that distance between the new cluster \( C_k \) and all remaining clusters \( C_l \) are computed as:

\[
D(C_k,C_l) = \frac{D(C_i,C_4) \cdot l_{i1} + D(C_j,C_4) \cdot l_{j1}}{l_{i1} + l_{j1}}
\]

**Step 4:** Repeat Step 2 and 3 until only one cluster remains.

Running time \( O(n^3) \). Space consumption \( O(n^2) \)
```python
def upgma(d):
    """
    Returns a UPGMA tree based on the distance matrix d which is a QuadTree or BasicMatrix
    """
    n = d.dim
    cluster = [[x] for x in xrange(n)]
    size = n * [1]

    while n > 1:
        # find the two clusters i and j with minimum distance
        i, j = d.argmin()

        # build list of distances from the new cluster to all other clusters
        new_cluster = [cluster[i], cluster[j]]
        new_size = size[i] + size[j]

        new_dist = []
        for k in xrange(d.dim):
            if cluster[k] == None:
                new_dist.append(sys.maxint)
            else:
                new_dist.append((d.get_elm(i,k)*size[i] + d.get_elm(j,k)*size[j]) / new_size)

        # insert new row/col in d
        new_dist[i] = sys.maxint
        d.insert_row(i, new_dist)
        d.insert_col(i, new_dist)
        d.insert_row(j, d.dim * [sys.maxint])
        d.insert_col(j, d.dim * [sys.maxint])

        cluster[i] = new_cluster
        cluster[j] = None
        size[i] = new_size
        size[j] = 0

        n = n - 1

    return new_cluster
```
Properties of an UPGMA tree

... if our distance data are derived by adding up edge lengths in a tree with a molecular clock, then UPGMA will reconstruct T correctly...

\[ \forall v: |v_i| = |v_j| \]

Molecular clock

If the "true" tree does not satisfy this then UPGMA does not necessarily reconstruct it correctly...
A clocklike tree is a rooted tree, in which the total edge length from the root to any leaf is equal, i.e. there is a molecular clock that ticks in a constant pace (the mutation rate is identical for all species), and all the observed species are at an equal number of ticks from the root ...

A clocklike tree induces an ultrametric distance matrix ...
Ultrametric matrices and trees

Formally: A matrix $d_{ij}$ is ultrametric iff it is metric and any three species $x, y, z$ satisfy that $d(x,y) \leq \max\{d(x,z), d(y,z)\}$, i.e. there is a tie for the maximum of $d(x,y)$, $d(x,z)$, and $d(y,z)$.

Theorem (Gusfield): A symmetric matrix $D$ has an ultra-metric tree iff $D$ is an ultrametric matrix.

Theorem (Gusfield): If $D$ is ultrametric then the unique ultra-metric tree $T$ for $D$ can be constructed in time $O(n^2)$. 
Two examples

Ultrametric doesn’t imply molecular clock

```
   1
  / \
1  3
 /   \
 a   b
```

```
<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>b</td>
<td>0</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Molecular clock implies ultrametric

```
   1
  / \
1  2
 /   \
 a   b
```

```
<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>b</td>
<td>0</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Improving the running time of UPGMA

**Input:** A set of $n$ species (1,2,...,n) and a distance matrix $d$ giving the pairwise distances.

**Output:** A rooted binary tree $T$ with edge lengths and leaves labelled 1, 2, ..., $n$ such that similar species (according to the distance matrix) are grouped in the same subtree, and the tree distances correspond to the distance matrix.

**Algorithm:**

**Step 1:** Let the $n$ species 1,2, ..., $n$ be $n$ clusters $C_1$,..,$C_n$ of size 1. Let each cluster correspond to a leaf in a tree $T$. The distance $D(C_i, C_j)$ between cluster $C_i$ and $C_j$ is $d(i,j)$.

**Step 2:** Pick the pair of cluster $C_i$ and $C_j$ where $D(C_i, C_j)$ is minimized and form a new cluster $C_k$ by merging $C_i$ and $C_j$, i.e. $C_k = C_i U C_j$. Make a new internal node $C_k$ in tree $T$ with children corresponding to $C_i$ and $C_j$. Place internal node $C_k$ at height $D(C_i, C_j) / 2$.

**Step 3:** Update distance matrix $D$ such that distance between the new cluster $C_k$ and all remaining clusters $C_i$ are computed as:

$$D(C_k, C_i) = \frac{D(C_i, C_4) \cdot 1_{i \neq 1} + D(C_j, C_4) \cdot 1_{i = j}}{1_{i \neq 1} + 1_{i = j}}$$

**Step 4:** Repeat Step 2 and 3 until only one cluster remains.

Running time $O(n^3)$. Space consumption $O(n^2)$.
Improving the running time of UPGMA

**Input:** A set of \( n \) species (1,2,\,...,\,n) and a distance matrix \( d \) giving the pairwise distances.

**Output:** A rooted binary tree \( T \) with edge lengths and leaves labelled 1, 2, \,...,\, n such that similar species (according to the distance matrix) are grouped in the same subtree, and the tree distances correspond to the distance matrix.

**Algorithm:**

**Step 1:** Let the \( n \) species 1,2, \,...,\,n be \( n \) clusters \( C_1,\,...,C_n \) of size 1. Let each cluster correspond to a leaf in a tree \( T \). The distance \( D(C_i, C_j) \) between cluster \( C_i \) and \( C_j \) is \( d(i,j) \).

**Step 2:** Pick the pair of cluster \( C_i \) and \( C_j \) where \( D(C_i, C_j) \) is minimized and form a new cluster \( C_k \) by merging \( C_i \) and \( C_j \), i.e. \( C_k = C_i \cup C_j \). Make a new internal node \( C_k \) in tree \( T \) with children corresponding to \( C_i \) and \( C_j \). Place internal node \( C_k \) at height \( D(C_i, C_j) / 2 \).

**Step 3:** Update distance matrix \( D \) such that distance between the new cluster \( C_k \) and all remaining clusters \( C_i \) are computed as:

\[
D(C_k, C_i) = \frac{D(C_i, C_1) \cdot 1_{i=1} + D(C_j, C_4) \cdot 1_{i=4}}{1_{i=1} + 1_{i=4}}
\]

**Step 4:** Repeat Step 2 and 3 until only one cluster remains.

Running time \( O(n^3) \). Space consumption \( O(n^2) \)
Improving the running time of UPGMA

**Input:** A set of $n$ species (1,2,...,$n$) and a distance matrix $d$ giving the pairwise distances.

**Output:** A rooted binary tree $T$ with edge lengths and leaves labelled 1, 2, ..., $n$ such that similar species (according to the distance matrix) are grouped in the same subtree, and the tree distances correspond to the distance matrix.

**Algorithm:**

**Step 1:** Let the $n$ species 1,2, ..., $n$ be $n$ clusters $C_1$,...,$C_n$ of size 1. Let each cluster correspond to a leaf in a tree $T$. The distance $D(i)$ takes $O(n^2)$ time. Can be improved to $O(n)$. Yields a total running time of $O(n^2)$.

**Step 2:** Pick the pair of cluster $C_i$ and $C_j$ where $D(C_i, C_j)$ is minimized and form a new cluster $C_k$ by merging $C_i$ and $C_j$, i.e. $C_k = C_i \cup C_j$. Make a new internal node $C_k$ in tree $T$ with children corresponding to $C_i$ and $C_j$. Place internal node $C_k$ at height $D(C_i, C_j) / 2$.

**Step 3:** Update distance matrix $D$ such that distance between the new cluster $C_k$ and all remaining clusters $C_i$ are computed as:

$$D(C_k, C_i) = \frac{D(C_i, C_j) \cdot 1_{i \neq j} + D(C_j, C_k) \cdot 1_{i \neq j}}{1_{i \neq j} + 1_{j \neq i}}$$

**Step 4:** Repeat Step 2 and 3 until only one cluster remains.

Running time $O(n^3)$. Space consumption $O(n^2)$.
Storing matrix $D$ in a **quad tree**

Building the quad tree for a $n \times n$ matrix takes time $O(n^2)$, when built we can pick the minimum in constant time $O(1)$. Not surprising.
Storing matrix $D$ in a quad tree

Building the quad tree for a $n \times n$ matrix takes time $O(n^2)$, when built we can pick the minimum in constant time $O(1)$. Not surprising.

**Question:** How fast can we rebuilt the quad-tree to reflect the modified distance matrix we built in Step 3?
**Question:** How fast can we rebuild the quad-tree to reflect the modified distance matrix we built in Step 3? In Step 3, we only change 2 row/columns in $D$. 

The diagram shows a before and after comparison of the quad-tree. The before section has a grid structure labeled with indices, while the after section has the same grid with additional labels indicating the removal of a row and column. The text below the diagram explains that row/column $i$ is used to store the distances $D(k,i)$ and $D(k, k)$, and $D(k, k)$, i.e., the distances from the new cluster to the other clusters.
**Updating the quad-tree**

**Question:** How fast can we rebuild the quad-tree to reflect the modified distance matrix we built in Step 3? In Step 3, we only change 2 row/columns in $D$.

... we update two row/columns in each iteration, how long does it take to rebuild the quad-tree to reflect this?

$4 \cdot n + 4 \cdot \frac{n}{2} + 4 \cdot \frac{n}{4} + ... + 4 = 4(n + \frac{n}{2} + ... + 1) < 8n = O(n)$
Updating the quad-tree

Question:
How fast can be rebuilt the quad-tree to reflect the modified distance matrix we built in Step 3?

In Step 3, we only change 2 row/columns in $D$.

... we update two row/columns in each iteration. How long does it take to rebuild the quad-tree to reflect this?

$4 \cdot n + 4 \cdot \frac{n}{2} + 4 \cdot \frac{n}{4} + \ldots + 4 = 4(n + \frac{n}{2} + \ldots + 1) < 8n = O(n)$

we have removed row and column $j$. 

row/column $i$ is used to store the distances $D(k, i)$ and $D(k, j)$...

i.e. the distances from the new cluster to the other clusters
```python
class QuadTree:
    """
    A basic Quad Tree
    """
    def __init__(self, dim):
        """
        Constructs a quad tree of dimension dim containing 0's"
        """
        self.undef = sys.maxint
        self.dim = dim
        n = self.dim + self.dim % 2
        self.level = []
        while n > 1:
            n = n + (n % 2)
            self.level.append(Matrix(n))
            n = (n + 1) / 2

    def __str__(self):
        """
        returns a string containing a simple output of each level in the quad tree"
        """
        s = ""
        if len(self.level) > 0:
            s = str(self.level[0])
            for l in self.level[1:]:
                s = s + '\n' + str(l)
        return s

def get_elm(self, i, j):
    """
    Returns the element stored in entry (i,j) in the quad tree"
    """
    return self.level[0][i][j]

def insert_elm(self, i, j, elm):
    """
    Inserts elm in entry (i,j) in the quad tree"
    """
    for l in self.level:
        l[i][j] = elm
        elm = self.quad_min(i, j, l)
        i = i / 2
        j = j / 2

def insert_row(self, i, row):
    """
    Inserts row (of dim elements) as row i in the quad tree"
    """
    curr_row = row
    for l in self.level:
        if len(curr_row) % 2 == 1:
            curr_row = append(self.undef)
        next_row = []
        for j in xrange(len(curr_row)):
            l[i][j] = curr_row[j]
            if j % 2 == 1:
                next_row.append(self.quad_min(i, j, l))
        i = i / 2
        curr_row = next_row

class Matrix:
    """
    A very basic square matrix organized as a list of lists.
    """
    def __init__(self, dim, elm=sys.maxint):
        """
        Builds an empty nxn-matrix"
        """
        self.undef = sys.maxint
        self.dim = dim
        self.data = []
        for i in xrange(self.dim):
            self.data.append(self.undef * [elm])

    def __getitem__(self, i):
        """
        Returns row i"
        """
        return self.data[i]

    def __str__(self):
        s = ""
        for i in xrange(self.dim):
            for j in xrange(self.dim):
                s = s + str(self.data[i][j]) + " "
        s = s + \n"
        return string.strip(s)
```
class QuadTree:
    ""
    A basic Quad Tree
    ""
    def __init__(self, dim):
        """Constructs a quad tree of dimension dim."""
        self.undef = sys.maxint
        self.dim = dim
        n = self.dim + self.dim % 2
        self.level = []
        while n > 1:
            n = n + (n % 2)
            self.level.append(Matrix(n))
            n = (n + 1) / 2

    def __str__(self):
        """returns a string containing a size
        s = ""
        if len(self.level) > 0:
            s = str(self.level[0])
            for l in self.level[1::]:
                s = s + '\n' + str(l)
        return s

    def get_elm(self, i, j):
        """returns the element stored in entry (i,j) in the tree
        return self.level[0][i][j]
        ""
    def insert_elm(self, i, j, elm):
        """inserts elm in entry (i,j) in the tree
        for l in self.level:
            l[i][j] = elm
            i = i / 2
            j = j / 2
        ""
    def insert_row(self, i, row):
        """inserts row (of dim elements) as row
        for l in self.level:
            if len(l) % 2 == 1:
                l.append(self.undef)
            l[i] = row[j]
            if j % 2 == 1:
                j = j / 2
        ""
    def insert_col(self, j, col):
        """inserts col (of dim elements) as col j in the quad tree"
        curr_col = col
        for l in self.level:
            if len(curr_col) % 2 == 1:
                curr_col.append(self.undef)
            next_col = []
            for i in xrange(len(curr_col)):
                l[i][j] = curr_col[i]
                if i % 2 == 1:
                    next_col.append(self.quad_min(i, j, l))
            j = j / 2
            curr_col = next_col

    def min(self):
        """returns the minimum element stored in the quad tree"
        return self.quad_min(0, 0, self.level[-1])

    def argmin(self):
        """returns the coordinates of the minimum element stored in the quad tree"
        i = 0
        for l in reversed(self.level[1::]):
            i = l.quad_argmin(i, j, l)
            i = i * 2
            j = j * 2
        return self.quad_argmin(i, j, self.level[0])

    def quad_min_all(self, i, j, l):
        """returns the minimum element stored in the quad containing (i,j) and its coordinates"
        i = (i / 2) * 2
        j = (j / 2) * 2
        return min((l[i][j], i, j), (l[i+1][j], i+1, j), (l[i][j+1], i, j+1), (l[i+1][j+1], i+1, j+1))

    def quad_min(self, i, j, l):
        """returns the minimum element stored in the quad containing (i,j)"
        return self.quad_min_all(i, j, l)[0]

    def quad_argmin(self, i, j, l):
        """returns the coordinates of minimum element stored in the quad containing (i,j)"
        return self.quad_min_all(i, j, l)[1:]
Improving the running time of UPGMA

**Input:** A set of $n$ species (1,2,...,n) and a distance matrix $d$ giving the pairwise distances.

**Output:** A rooted binary tree $T$ with edge lengths and leaves labelled 1, 2, ..., $n$ such that similar species (according to the distance matrix) are grouped in the same subtree, and the tree distances correspond to the distance matrix.

**Algorithm:**

**Step 1:** Let the $n$ species 1,2, ..., $n$ be $n$ clusters $C_1,...,C_n$ of size 1. Let each cluster correspond to a leaf in a tree $T$. The distance $D(C_i, C_j)$ between cluster $C_i$ and $C_j$ is $d(i,j)$.

**Step 2:** Pick the pair of cluster $C_i$ and $C_j$ where $D(C_i, C_j)$ is minimized and form a new cluster $C_k$ by merging $C_i$ and $C_j$, i.e. $C_k = C_i \cup C_j$. Make a new internal node $C_k$ in tree $T$ with children corresponding to $C_i$ and $C_j$. Place internal node $C_k$ at height $D(C_i, C_j) / 2$.

**Step 3:** Update distance matrix $D$ such that distance between the new cluster $C_k$ and all remaining clusters $C_l$ are computed as:

$$D(C_k, C_l) = \frac{D(C_i, C_j) \cdot l_{i1} + D(C_j, C_l) \cdot l_{j1}}{l_{i1} + l_{j1}}$$

**Step 4:** Repeat Step 2 and 3 until only one cluster remains.

Running time $O(n^2)$ using a quad-tree to store $D$. Space consumption $O(n^2)$
Improving the running time of UPGMA

**Input:** A set of \( n \) species \( (1,2,...,n) \) and a distance matrix \( d \) giving the pairwise distances.

**Output:** A rooted binary tree \( T \) with edge lengths and leaves labelled 1, 2, ..., \( n \) such that similar species (according to the distance matrix) are grouped in the same subtree, and the tree distances correspond to the distance matrix.

**Algorithm:**

1. Let the \( n \) species \( 1,2,...,n \) be \( n \) clusters \( C_1,...,C_n \) of size 1. Let each cluster correspond to a leaf in a tree \( T \). The distance \( D(C_i,C_j) \) between cluster \( C_i \) and \( C_j \) is \( d(i,j) \).

2. Pick the pair of cluster \( C_i \) and \( C_j \) where \( D(C_i,C_j) \) is minimized and form a new cluster \( C_k \) by merging \( C_i \) and \( C_j \), i.e. \( C_k = C_i \cup C_j \). Make a new internal node \( C_k \) in tree \( T \) with children corresponding to \( C_i \) and \( C_j \). Place internal node \( C_k \) at height \( D(C_i,C_j) / 2 \).

3. Update distance matrix \( D \) such that distance between the new cluster \( C_k \) and all remaining clusters \( C_i \) are computed as:

\[
D(C_k,C_i) = \frac{D(C_i,C_4) \cdot 1_{i=1} + D(C_j,C_4) \cdot 1_{i=j}}{1_{i=1} + 1_{i=j}}
\]

4. Repeat Step 2 and 3 until only one cluster remains.

Running time \( O(n^2) \) using a quad-tree to store \( D \). Space consumption \( O(n^2) \)
Does it matter in practice?

T_{basic}(n) / T_{Quad}(n)

\( T_{\text{quad}}(n) / n^2 \)

\( T_{\text{basic}}(n) / n^3 \)
Neighbor Joining
**Neighbor Joining**

**Input:** A set of \( n \) species (1,2,...,\( n \)) and a distance matrix \( d \) giving the pairwise distances.

**Output:** An unrooted binary tree \( T \) with edge lengths and leaves labelled 1, 2, ..., \( n \) such that similar species (according to the distance matrix) are grouped in the same subtree, and the tree distances correspond to the distance matrix.

Popular distance based phylogenetic inference method with good accuracy.

Introduced by Saitou and Nei in 1987.


Criticized by many but still widely used.

Often used as a benchmark for other methods.
**Neighbor Joining**

**Input:** A set of $n$ species (1,2,...,n) and a distance matrix $d$ giving the pairwise distances.

**Output:** An unrooted binary tree $T$ with edge lengths and leaves labelled 1, 2, ..., $n$ such that similar species (according to the distance matrix) are grouped in the same subtree, and the tree distances correspond to the distance matrix.

Let us assume that the distance matrix is additive, i.e. corresponds to tree distances from some tree with unknown topology and edge length, then this is the tree we want to infer …

If we from the distance matrix can infer which leaves (and sub trees) are neighbors in the unknown (true) tree, we can reconstruct the tree iteratively …
A “simple” way to solve the Distance-Based Phylogeny problem for additive trees\textsuperscript{16} is to find a pair of neighboring leaves, that is, leaves that have the same parent vertex.\textsuperscript{17} Figure 10.12 illustrates that for a pair of neighboring leaves $i$ and $j$ and their parent vertex $k$, the following equality holds for every other leaf $m$ in the tree:

$$D_{k,m} = \frac{D_{i,m} + D_{j,m} - D_{i,j}}{2}$$

Therefore, as soon as a pair of neighboring leaves $i$ and $j$ is found, one can remove the corresponding rows and columns $i$ and $j$ from the distance matrix and add a new row and column corresponding to their parent $k$. Since the distance matrix is additive, the distances from $k$ to other leaves are recomputed as $D_{k,m} = \frac{D_{i,m} + D_{j,m} - D_{i,j}}{2}$. 

\[\begin{array}{c}
\includegraphics[width=0.5\textwidth]{diagram.png}
\end{array}\]
A “simple” way to solve the Distance-Based Phylogeny problem for additive trees\textsuperscript{16} is to find a pair of neighboring leaves, that is, leaves that have the same parent vertex.\textsuperscript{17} Figure 10.12 illustrates that for a pair of neighboring leaves $i$ and $j$ and their parent vertex $k$, the following equality holds for every other leaf $m$ in the tree:

$$D_{k,m} = \frac{D_{i,m} + D_{j,m} - D_{i,j}}{2}$$

Therefore, as soon as a pair of neighboring leaves $i$ and $j$ is found, one can remove the corresponding rows and columns $i$ and $j$ from the distance matrix and add a new row and column corresponding to their parent $k$. Since the distance matrix is additive, the distances from $k$ to other leaves are recomputed as $D_{k,m} = \frac{D_{i,m} + D_{j,m} - D_{i,j}}{2}$.

Not easy to infer neighboring leaves!!
A “simple” way to solve the Distance-Based trees\(^\text{16}\) is to find a pair of neighboring leaves and their parent vertex.\(^\text{17}\) Figure 10.12 illustrates that, if \(i\) and \(j\) and their parent vertex \(k\), the following equality holds for every other leaf \(m\) in the tree:

\[
D_{k,m} = \frac{D_{i,m} + D_{j,m} - D_{i,j}}{2}
\]

Therefore, as soon as \(k\) is removed the corresponding rows and columns \(i\) and \(j\) from the distance matrix and add a new row and column corresponding to their parent \(k\). Since the distance matrix is additive, the distances from \(k\) to other leaves are recomputed as

Here \(D(i,j) = D(k,l) = D(j,l) = 13\), and \(D(j,k) = 12\), so the leaves with minimum distance in the distance matrix are not necessarily neighbors in the tree, i.e. UPGMA will not reconstruct the tree correctly.

Not easy to infer neighboring leaves!!
Neighbor Joining

and then abandoned in the previous section. In 1987 Naruya Saitou and Masatoshi Nei developed an ingenious neighbor joining algorithm for phylogenetic tree reconstruction. In the case of additive trees, the neighbor joining algorithm somehow magically finds pairs of neighboring leaves and proceeds by substituting such pairs with the leaves’ parent. However, neighbor joining works well not only for additive distance matrices but for many others as well: it does not assume the existence of a molecular clock and ensures that the clusters that are merged in the course of tree reconstruction are not only close to each other (as in UPGMA) but also are far apart from the rest.
Neighbor Joining

and then abandoned in the previous section. In 1987 Naruya Saitou and Masatoshi Nei developed an ingenious neighbor joining algorithm for phylogenetic tree reconstruction. In the case of additive trees, the neighbor joining algorithm somehow magically finds pairs of neighboring leaves and proceeds by substituting such pairs with the leaves’ parent. However, neighbor joining works well not only for additive distance matrices but for many others as well: it does not assume the existence of a molecular clock and ensures that the clusters that are merged in the course of tree reconstruction are not only close to each other (as in UPGMA) but also are far apart from the rest.

For a cluster $C$, define $u(C) = \frac{1}{\text{number of clusters} - 2} \sum_{\text{all clusters } C', C''} D(C, C')$ as a measure of the separation of $C$ from other clusters. To choose which two clusters to merge, we look for the clusters $C_1$ and $C_2$ that are simultaneously close to each other and far from others. One may try to merge clusters that simultaneously minimize $D(C_1, C_2)$ and maximize $u(C_1) + u(C_2)$. However, it is unlikely that a pair of clusters $C_1$ and $C_2$ that simultaneously minimize $D(C_1, C_2)$ and maximize $u(C_1) + u(C_2)$ exists. As an alternative, one opts to minimize $D(C_1, C_2) - u(C_1) - u(C_2)$. This approach is used in the NEIGHBORJOINING algorithm below.
Neighbor Joining

**Input:** A set of $n$ species (1,2,...,n) and a distance matrix $d$ giving the pairwise distances.

**Output:** An unrooted binary tree $T$ with edge lengths and leaves labelled 1, 2, ..., $n$ such that similar species (according to the distance matrix) are grouped in the same subtree, and the tree distances correspond to the distance matrix.

**Algorithm:**

1. Define $T$ to be the set of leaf nodes, one for each species.
2. Define set of active nodes $L$ to be the $n$ leaves.
Neighbor Joining

**Input:** A set of $n$ species (1,2,...,n) and a distance matrix $d$ giving the pairwise distances.

**Output:** An unrooted binary tree $T$ with edge lengths and leaves labelled 1, 2, ..., $n$ such that similar species (according to the distance matrix) are grouped in the same subtree, and the tree distances correspond to the distance matrix.

**Algorithm:**

1. Find $i,j$ in $L$ where $D_{ij}$ is minimal
   
   $$D_{ij} = d(i,j) - (r_i + r_j), \quad r_i = \frac{\sum d(i,k) / (n-2)}{K_{ij}}$$

2. Define new node $k$ with distance to other nodes as:
   
   $$d(k,m) = \frac{1}{2} (d(i,m) + d(j,m) - d(i,j))$$

3. Add $k$ with sons $i$ and $j$ to $T$

   $$\frac{1}{2} (d(i,j) + r_i + r_j)$$

4. Remove $i,j$ from $L$, and add $k$. 

\[ \frac{1}{2} \left( d(i,j) - r_i + r_j \right) \]
**Neighbor Joining**

**Input:** A set of $n$ species (1, 2, ..., $n$) and a distance matrix $d$ giving the pairwise distances.

**Output:** An unrooted binary tree $T$ with edge lengths and leaves labelled 1, 2, ..., $n$ such that similar species (according to the distance matrix) are grouped in the same subtree, and the tree distances correspond to the distance matrix.

**Algorithm:**

- When $|I|=2$, join i, j with edge of length $d(i,j)$
**Input:** A set of \( n \) species (1, 2, ..., \( n \)) and a distance matrix giving the pairwise distances.

**Output:** An unrooted binary tree \( T \) with edge lengths and leaves labelled 1, 2, ..., \( n \) such that similar species are grouped in the same subtree, and the tree distances correspond to the distance matrix.

**Algorithm:**

**Initialization:**

1. Define \( T \) to be the set of leaf nodes, one for each species.
2. Define set of active nodes \( L \) to be the \( n \) leaves.

**Iteration:**

1. Find \( i,j \) in \( L \) where \( D_{ij} \) is minimal.

   \[
   D_{ij} = d(i,j) - (r_i + r_j), \quad r_i = \frac{\sum d(i,k)}{n-2}
   \]

2. Define new node \( k \) with distance to other nodes as:

   \[
   d(k,w) = \frac{1}{2} \cdot (d(i,w) + d(j,w) - d(i,j))
   \]

3. Add \( k \) with sons \( i \) and \( j \) to \( T \)

   \[
   \frac{1}{2} (d(i,j) + r_i - r_j) \quad \frac{1}{2} (d(i,j) - r_i + r_j)
   \]

4. Remove \( i,j \) from \( L \), and add \( k \).

**Termination:**

1. When \( |L| = 2 \), join \( i,j \) with edge of length \( d(i,j) \).
Running time and space consumption

Space consumption?

$O(n^2)$ for the distance matrix $d$

Running time?

$n$ iterations each taking time $O(n^2)$ ...

Step 1: compute row sums $\sum d(i,k)$ for each row in $d$ $O(n^2)$

- compute $D$ from $d$ and row sums $O(n^2)$

Step 2: add new node $k$, update $d$ $O(n)$

Step 3: extend $T$ $O(1)$

Step 4: remove rows from $d$ $O(cn)$

Total time $O(cn^2)$
Running time and space consumption

Space consumption?
$O(n^2)$ for the distance matrix $d$

Running time?
$n$ iterations each taking time $O(n^2)$ ...

**Step 1:** compute row sums $\sum_{b \in B} d(c,b)$ for each row in $L$ $O(n^2)$
- compute $D$ from $d$ and row sums $O(n^2)$

**Step 2:** add new node $h$, update $d$ $O(n)$

**Step 3:** extend $T$ $O(1)$

**Step 4:** remove rows from $L$ $O(n)$

Total time $O(n^2)$

Can we improve the running time of Step 1 by using a quad-tree? Why? Why not?
Properties of Neighbor Joining

... if our distance data are derived by adding up edge lengths in a tree, then NJ will reconstruct this tree correctly...

Such distance data is said to be additive.

If it holds that for any four species can label them 1, 2, 3, 4 such $d(i,j) + d(k,l) = d(i,k) + d(j,l) \geq d(i,l) + d(j,k)$ then $d$ is additive. [Buneman]
How 'deterministic' is NJ?

An experiment

To see how much this matters in practice, I made a small experiment. I took 37 distance matrices based on Pfam sequences, with between 100 and 200 taxa.

[...]

For each matrix I constructed ten equivalent distance matrices by randomly permuting the order of taxa. Then I used these ten matrices to construct ten neighbour-joining trees and finally computed the Robinson-Foulds between all pairs of these trees.

[...]

As you can see, you can get quite different trees out of equivalent distance matrices and on realistic data too.

I didn’t check if the differences between the trees are correlated with how much support the splits have in the data. Looking at bootstrap values for the trees might tell us something about that. Anyway, that is not the take home message I want to give here. That is that it doesn’t necessarily make sense to talk the neighbour-joining tree for a distance matrix.

Neighbour-joining is less deterministic than you might think!

How 'deterministic' is NJ?

Robinson-Foulds distance between "identical" NJ trees

RapidNJ is an implementation of NJ which uses heuristics to avoid searching large parts of the D matrix. No guarantees are given on how much of the matrix is avoided.

RapidNJ was compared with three fast Neighbour-Joining implementations:

**QuickTree** – A fast implementation of the canonical Neighbour-Joining method.

**QuickJoin** – Uses an advanced data structure to reduce the search space.

**Clearcut** – Relaxed Neighbour-Joining.

R8s was used to simulate phylogenetic trees from which distance matrices were constructed.

Alignments from Pfam were used to infer distance matrices using QuickTree.

Also work on how to use external memory to allow construction of large trees (> 10,000 leaves).

http://birc.au.dk/Software/RapidNJ/
Results on Simulated Data

The graph shows the relationship between walltime (seconds) and the number of taxa for different methods:
- QuickTree
- QuickJoin
- Clearcut
- RapidNJ

The graph indicates how each method scales with increasing number of taxa.
Results on small Pfam Data
Results on large Pfam Data
Running times on Pfam data