Distance based tree reconstruction

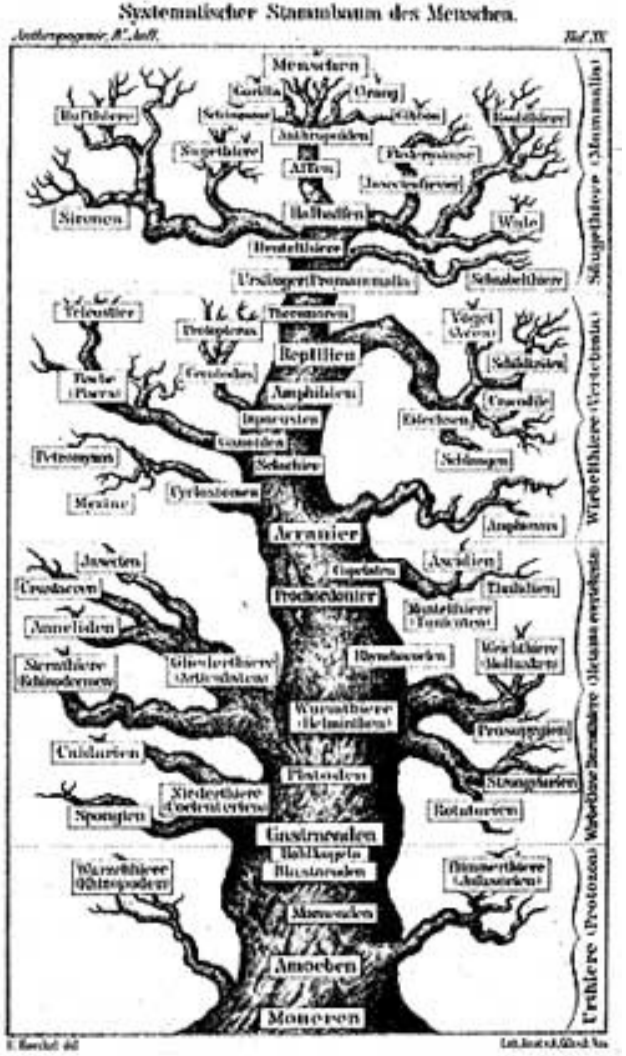
Hierarchical clustering (UPGMA)

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

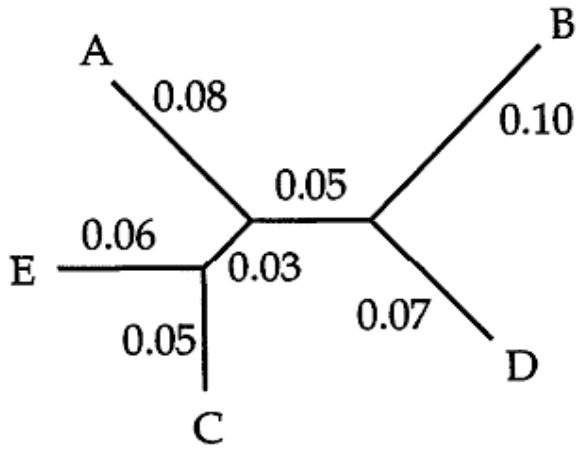
Goal: Reconstruct an evolutionary tree from a distance matrix

Input: n x n distance matrix d

Output: An additive tree T with n leaves, i.e. a tree with positive edge weights (but weight 0 is allowed on external edges).

If the matrix d is additive, this problem has an efficient solution (which we will come back to) that yields a tree T that is consistent with d, otherwise we should aim at finding a tree which is “as good as possible”, i.e. a tree as close as possible to the unknown “true tree”.

Distance based reconstruction



	A	B	C	D	E
A	0	0.23	0.16	0.20	0.17
B	0.23	0	0.23	0.17	0.24
C	0.16	0.23	0	0.15	0.11
D	0.20	0.17	0.15	0	0.21
E	0.17	0.24	0.11	0.21	0

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**

(**Theorem 10.4.18**: An additive matrix is consistent with an unique unrooted additive tree (that can be non-binary))

A reconstruction method

Least square methods

Given a tree topology T, select edge lengths such that

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

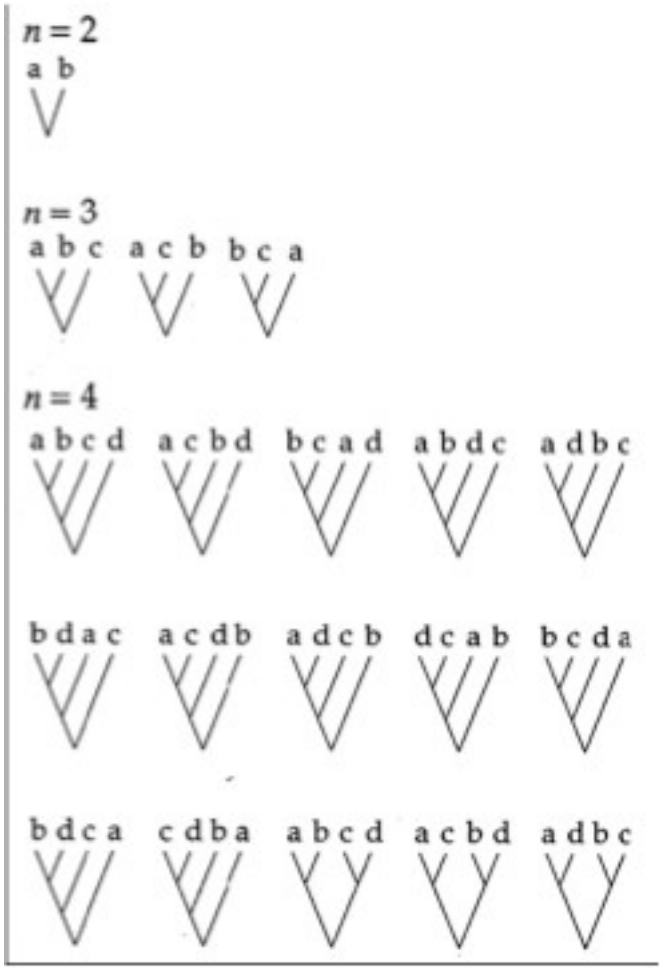
is minimized, where d_{ij} is given by the matrix, D_{ij} is the distance induced by the tree.

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 ...

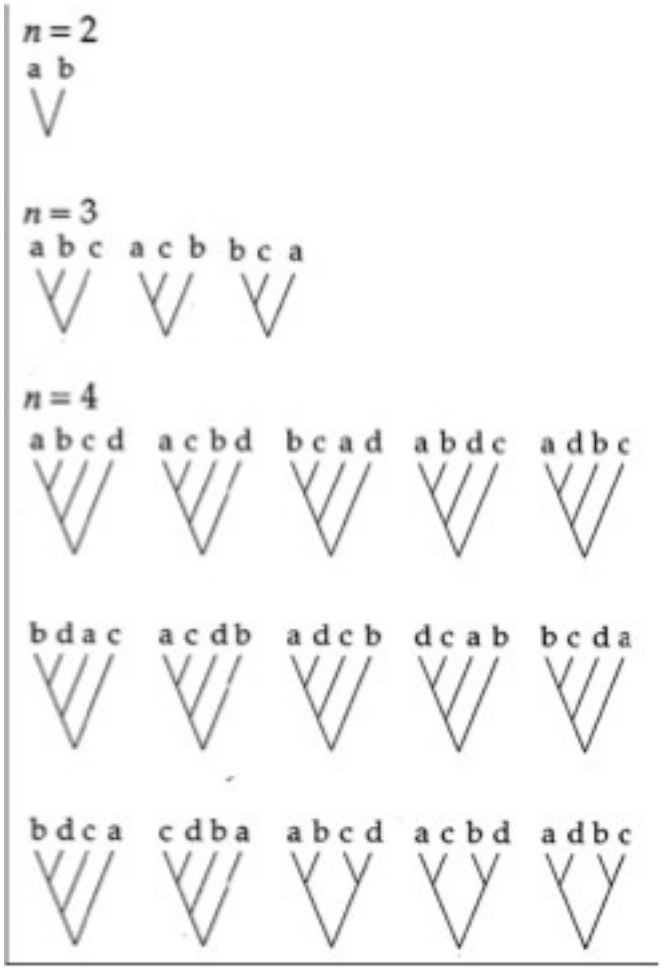
Counting rooted binary trees

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

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:

$$R(n) = R(n\text{-}1)\,(2(n\text{-}1)\text{-}2 + 1)$$
$$= R(n\text{-}1)\,(2n - 3)$$

What about unrooted trees?

What about non-binary trees?

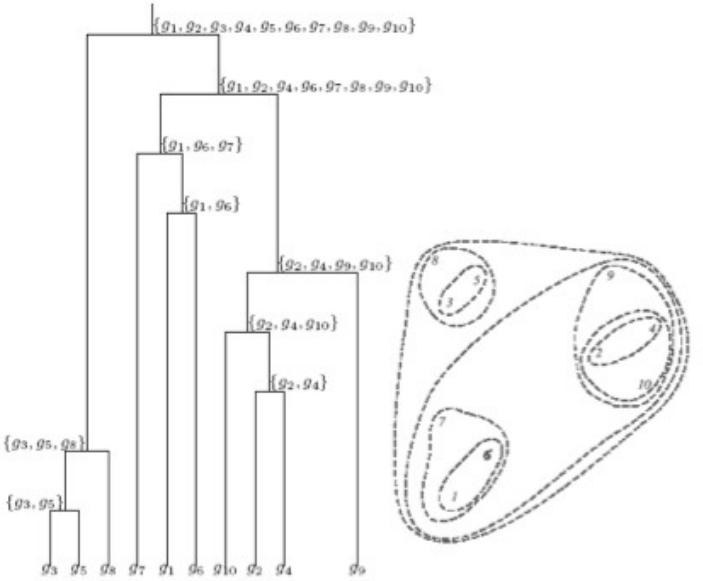
Hierarchical clustering and UPGMA

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 all root-leaf paths have equal length, similar species (cf. the distance matrix) are grouped in the same subtree, and the tree distances correspond to the distance matrix if possible.

	g_1	g_2	g_3	g_4	g_5	g_6	g_7	g_8	g_9	g_{10}
g_1	0.0	8.1	9.2	7.7	9.3	2.3	5.1	10.2	6.1	7.0
g_2	8.1	0.0	12.0	0.9	12.0	9.5	10.1	12.8	2.0	1.0
g_3	9.2	12.0	0.0	11.2	0.7	11.1	8.1	1.1	10.5	11.5
g_4	7.7	0.9	11.2	0.0	11.2	9.2	9.5	12.0	1.6	1.1
g_5	9.3	12.0	0.7	11.2	0.0	11.2	8.5	1.0	10.6	11.6
g_6	2.3	9.5	11.1	9.2	11.2	0.0	5.6	12.1	7.7	8.5
g_7	5.1	10.1	8.1	9.5	8.5	5.6	0.0	9.1	8.3	9.3
g_8	10.2	12.8	1.1	12.0	1.0	12.1	9.1	0.0	11.4	12.4
g_9	6.1	2.0	10.5	1.6	10.6	7.7	8.3	11.4	0.0	1.1
g_{10}	7.0	1.0	11.5	1.1	11.6	8.5	9.3	12.4	1.1	0.0



Hierarchical clustering

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

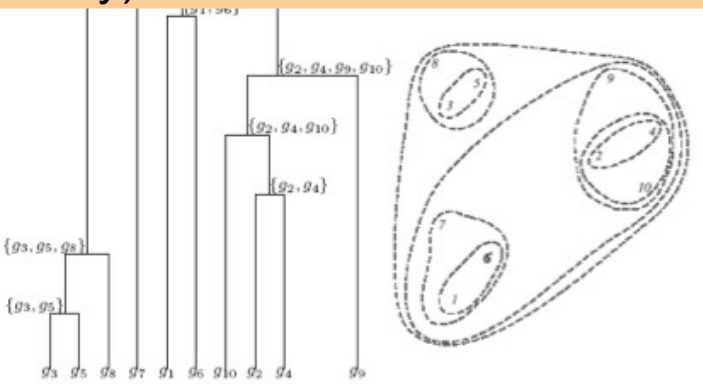
Output: A rooted binary tree *T* with edge lengths and leaves labelled 1, 2, ..., *n* such that all root-leaf paths have equal length, similar species (cf. the distance matrix) are grouped in the same subtree, and the tree distances correspond to the distance matrix if possible.

General approaches to hierarchical clustering

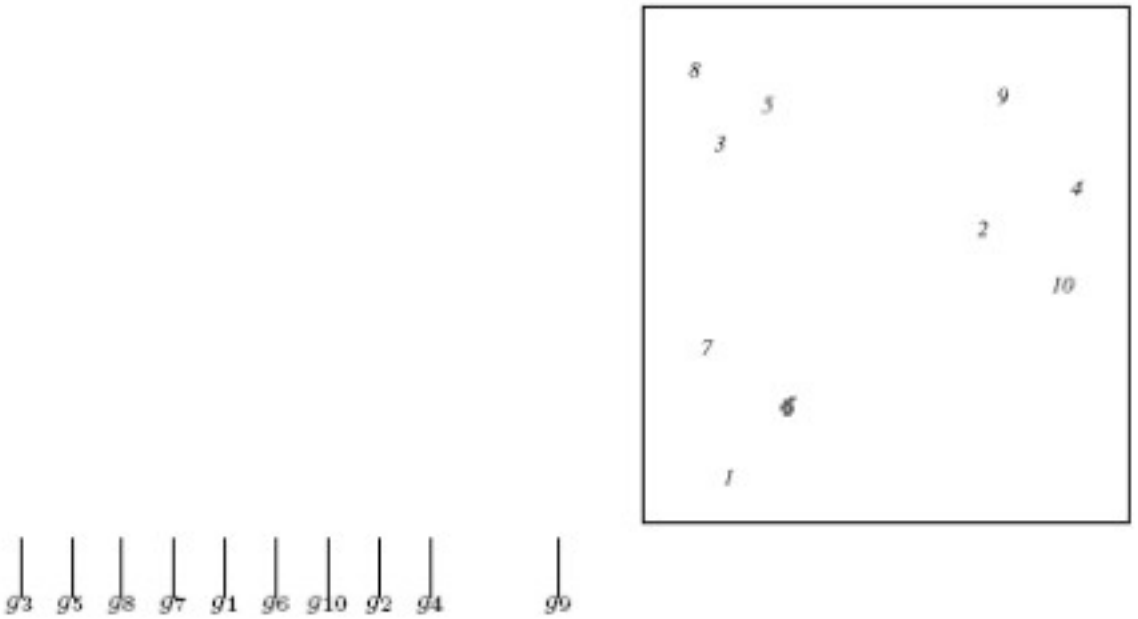
Agglomerative: Each species (observation) starts in its own cluster and at each step of the algorithm, two clusters (selected wisely) are combined into a single cluster.

Divisive: All species (observations) start in one cluster, and at each step of the algorithm, clusters are partitioned into a pair of clusters (selected wisely).

g_3	9.2	12.0	0.0	11.2	0.7	11.1	8.1	1.1	10.5	11.5
g_4	7.7	0.9	11.2	0.0	11.2	9.2	9.5	12.0	1.6	1.1
g_5	9.3	12.0	0.7	11.2	0.0	11.2	8.5	1.0	10.6	11.6
g_6	2.3	9.5	11.1	9.2	11.2	0.0	5.6	12.1	7.7	8.5
g_7	5.1	10.1	8.1	9.5	8.5	5.6	0.0	9.1	8.3	9.3
g_8	10.2	12.8	1.1	12.0	1.0	12.1	9.1	0.0	11.4	12.4
g_9	6.1	2.0	10.5	1.6	10.6	7.7	8.3	11.4	0.0	1.1
g_{10}	7.0	1.0	11.5	1.1	11.6	8.5	9.3	12.4	1.1	0.0

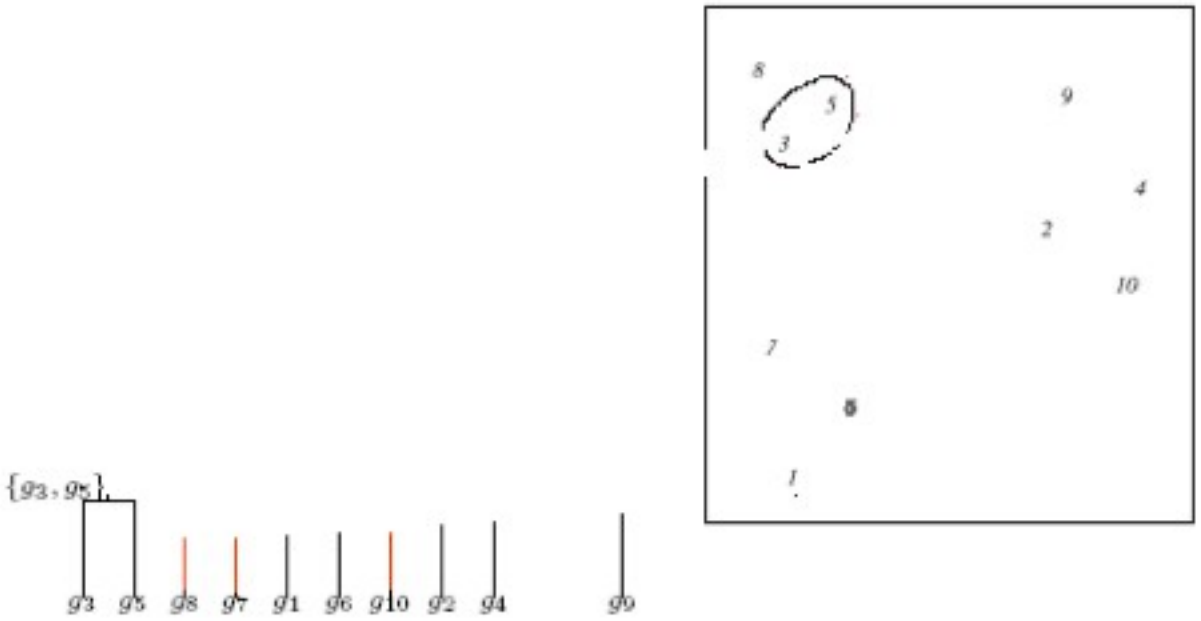


Example - Agglomerative



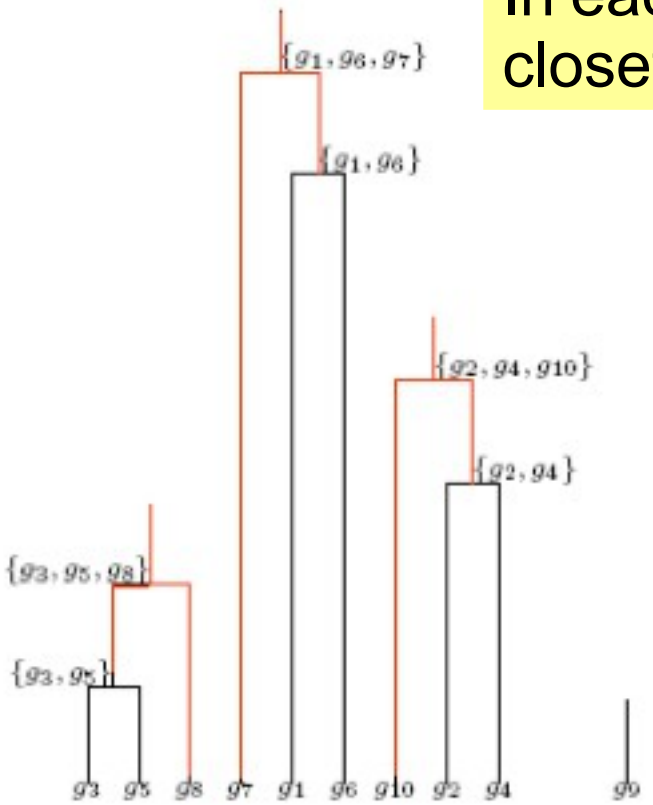
Example - Agglomerative

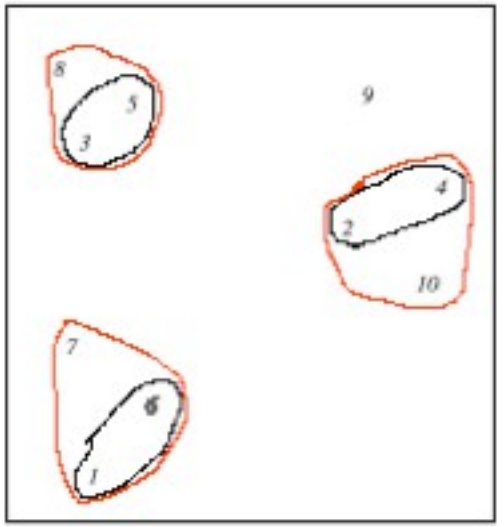
Group the closet two data points.



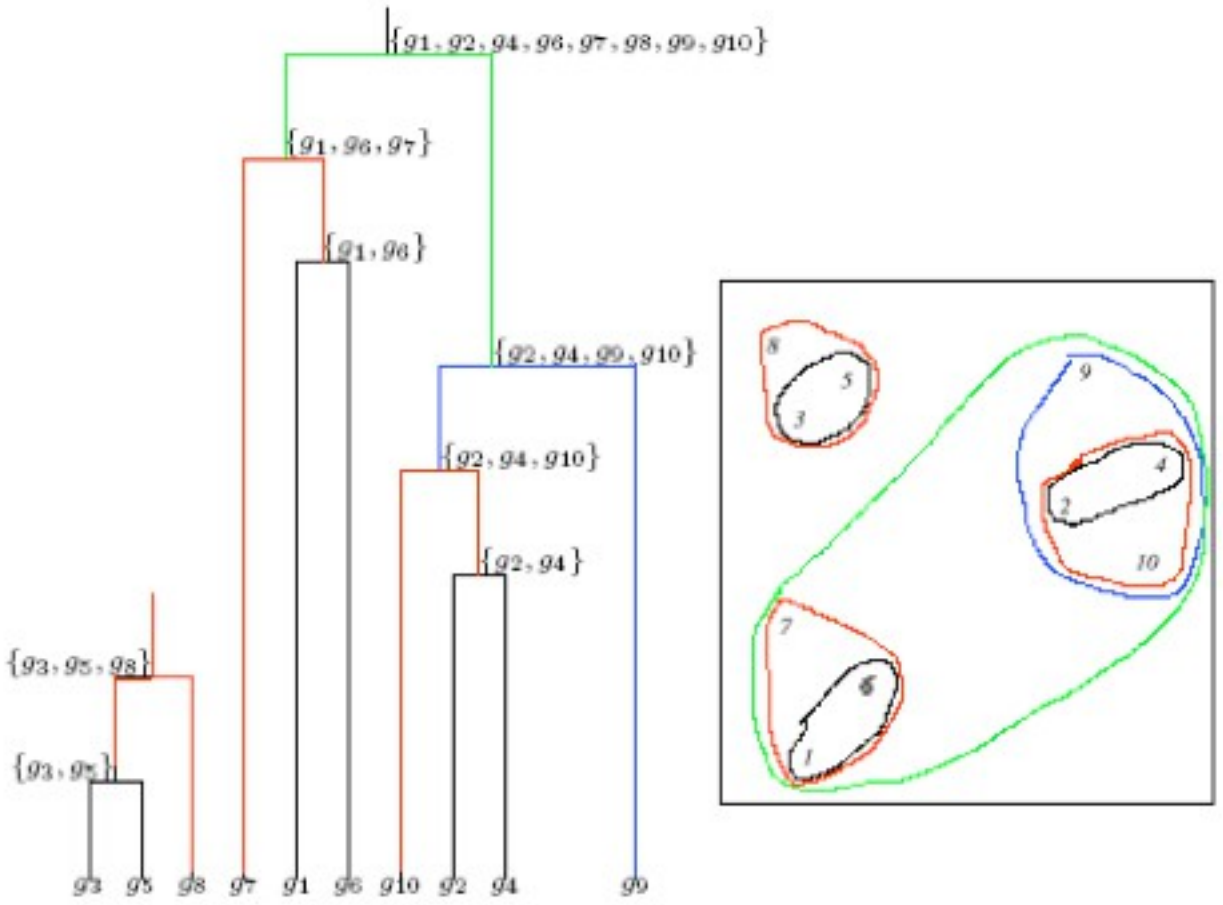
Example - Agglomerative

In each iteration, we join the closet two cluster.

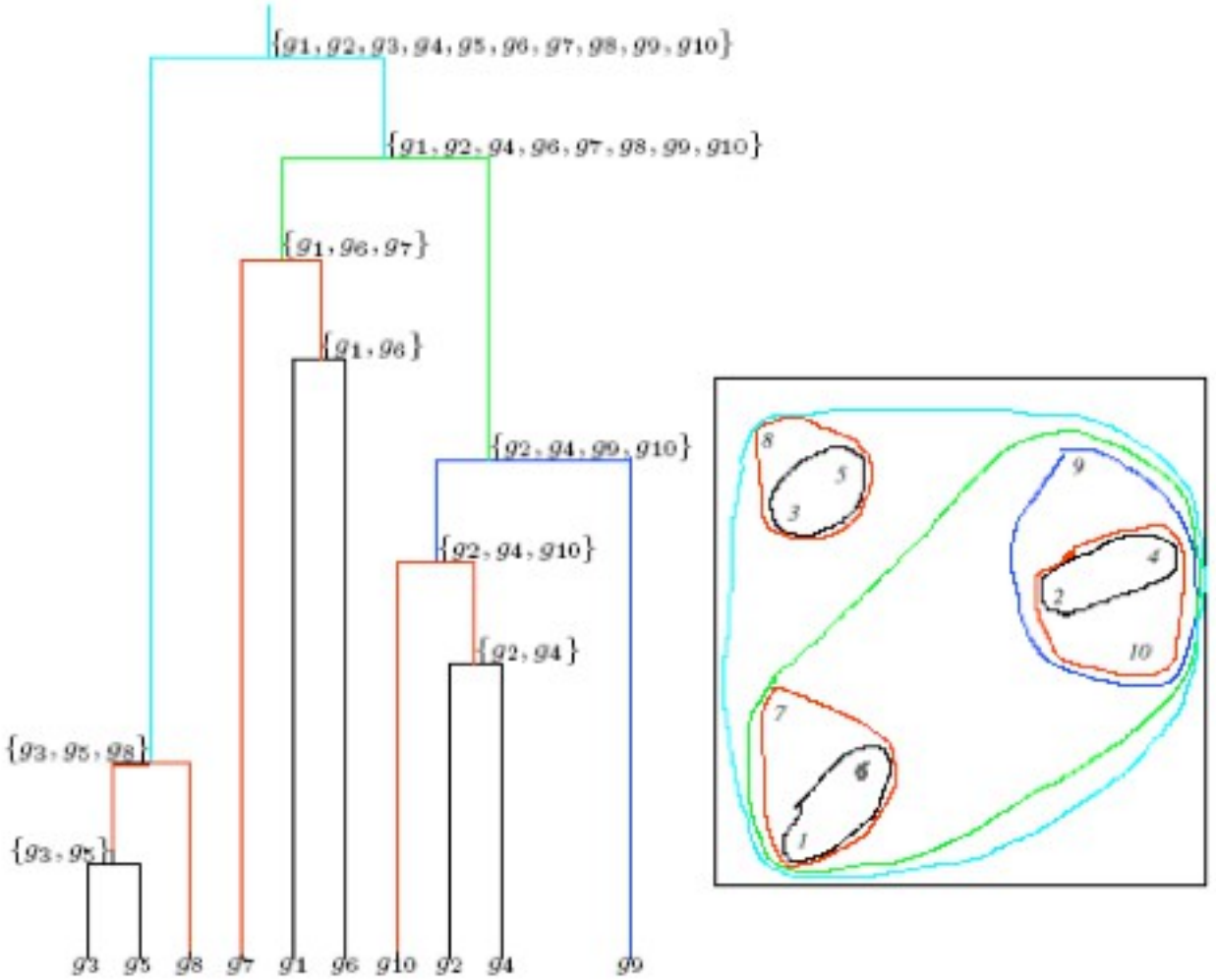




Example - Agglomerative



Example - Agglomerative



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 all root-leaf paths have equal length, similar species (cf. the distance matrix) are grouped in the same subtree, and the tree distances correspond to the distance matrix if possible.

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.

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 all root-leaf paths have equal length, similar species (cf. the distance matrix) are grouped in the same subtree, and the tree distances correspond to the distance matrix if possible.

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 $D(C_i, C_j) / 2$.

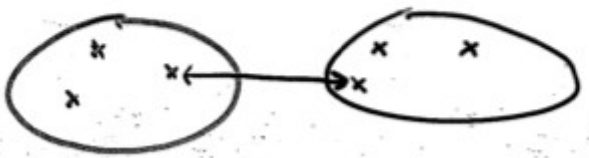
Step 3: Update distance matrix *D* such that distance between the new cluster C_k and all remaining clusters are computed.

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

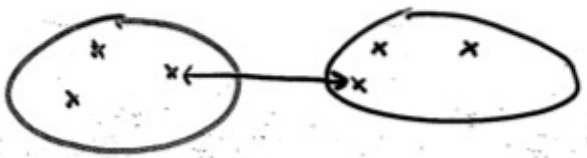
Single linkage



minimum dist between any two members ...

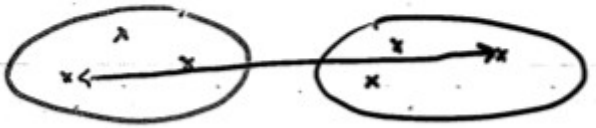
Distance measures between clusters

Single linkage



minimum dist between any two members ...

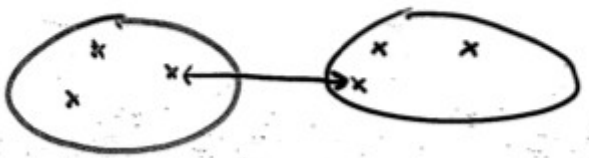
Complete linkage



maximum distance between any two members ...

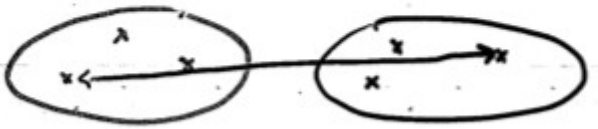
Distance measures between clusters

Single linkage



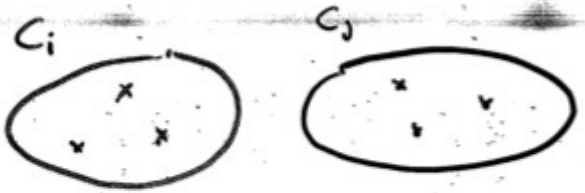
minimum dist between any two members ...

Complete linkage



maximum distance between any two members ...

Average linkage

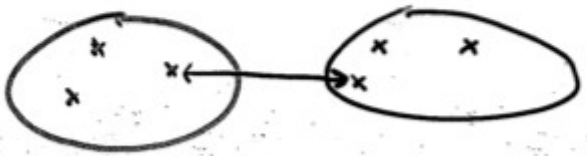


average of all pairwise distances between members of the two clusters

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

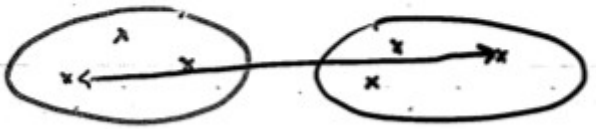
Distance measures between clusters

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

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

If $C_k = C_i \cup C_j$ then

$$D(C_k, C_\ell) = \frac{D(C_i, C_\ell) \cdot |C_i| + D(C_j, C_\ell) \cdot |C_j|}{|C_i| + |C_j|}$$

Hierarchical clustering (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 all root-leaf paths have equal length, similar species (cf. the distance matrix) are grouped in the same subtree, and the tree distances correspond to the distance matrix if possible.

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:

$$D(C_k, C_\ell) = \frac{D(C_i, C_\ell) \cdot |C_i| + D(C_j, C_\ell) \cdot |C_j|}{|C_i| + |C_j|}$$

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,...,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 all root-leaf paths have equal length, similar species (cf. the distance matrix) are grouped in the same subtree, and the tree distances correspond to the distance matrix if possible.

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:

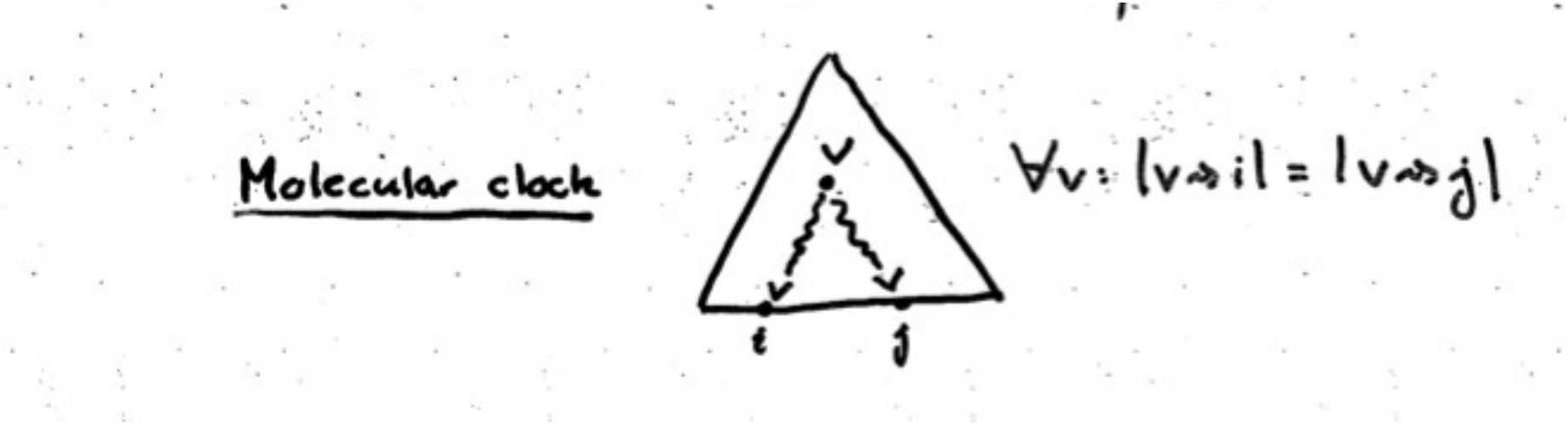
$$D(C_k, C_\ell) = \frac{D(C_i, C_\ell) \cdot |C_i| + D(C_j, C_\ell) \cdot |C_j|}{|C_i| + |C_j|}$$

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

Running time $O(n^3)$. Space consumption $O(n^2)$

Proporties of an UPGMA tree

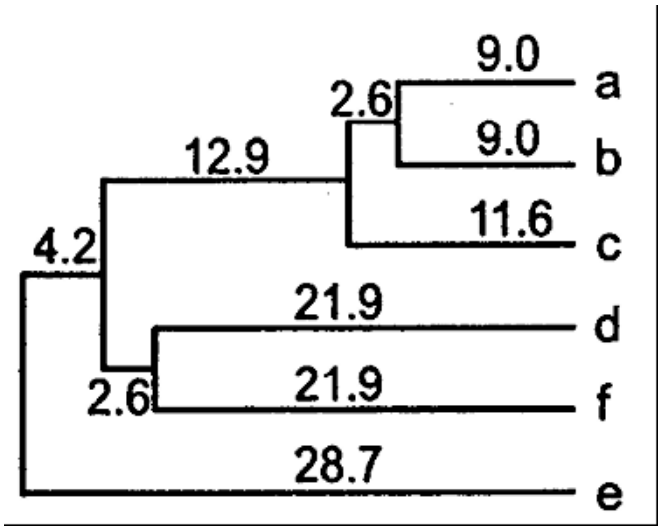
If the unknown “true tree” is a tree with a molecular clock, and our distance matrix reflects this without errors, i.e. corresponds to adding up the edge lengths in the “true tree”, then UPGMA will reconstruct the unknown “true tree” correctly.

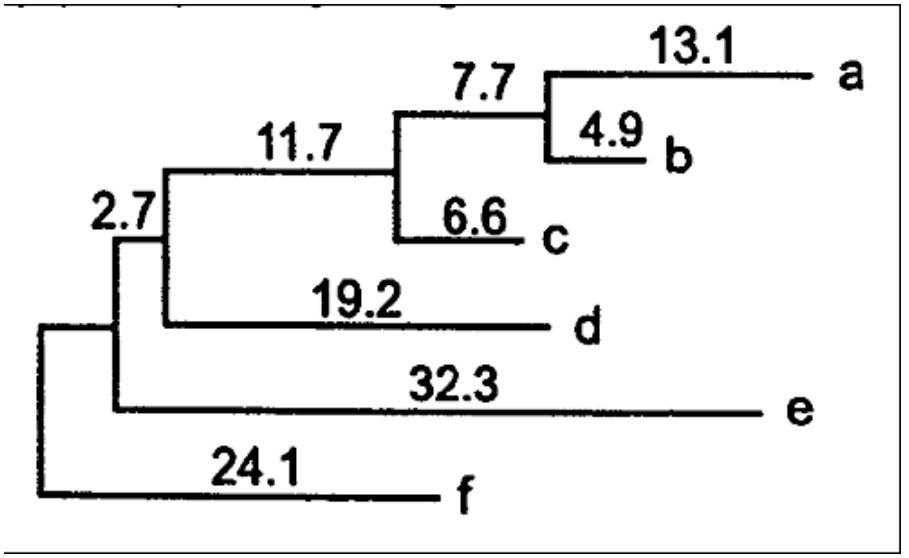


If the unknown “true tree” does not obey a molecular clock, then UPGMA does not reconstruct it correctly.

Trees reflecting a molecular clock

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

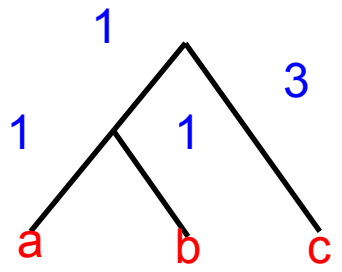
Formally: A matrix d_{ij} is ultrametric iff it is metric and any three species x, y, z satisfy that $d(x,y) \leq \max\{d(x,z), d(y,z)\}$, i.e there is a tie for the maximum of $d(x,y)$, $d(x,z)$, and $d(y,z)$

Theorem 10.3.4: A symmetric matrix D is consistent with an ultrametric tree iff D is an ultrametric matrix

Algorithm 10.1: If D is ultrametric then the unique ultrametric tree T that is consistent with D can be constructed in time $O(n^2)$

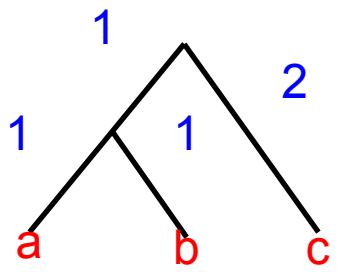
Two examples

Ultrametric doesn't imply molecular clock



	a	b	c
a	0	2	5
b		0	5
c			0

Molecular clock implies ultrametric



	a	b	c
a	0	2	4
b		0	4
c			0

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 all root-leaf paths have equal length, similar species (cf. the distance matrix) are grouped in the same subtree, and the tree distances correspond to the distance matrix if possible.

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:

$$D(C_k, C_\ell) = \frac{D(C_i, C_\ell) \cdot |C_i| + D(C_j, C_\ell) \cdot |C_j|}{|C_i| + |C_j|}$$

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 all root-leaf paths have equal length, similar species (cf. the distance matrix) are grouped in the same subtree, and the tree distances correspond to the distance matrix if possible.

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:

$$D(C_k, C_\ell) = \frac{D(C_i, C_\ell) \cdot |C_i| + D(C_j, C_\ell) \cdot |C_j|}{|C_i| + |C_j|}$$

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 all root-leaf paths have equal length, similar species (cf. the distance matrix) are grouped in the same subtree, and the tree distances correspond to the distance matrix if possible.

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 $D($

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 $D(C_i, C_j) / 2$.

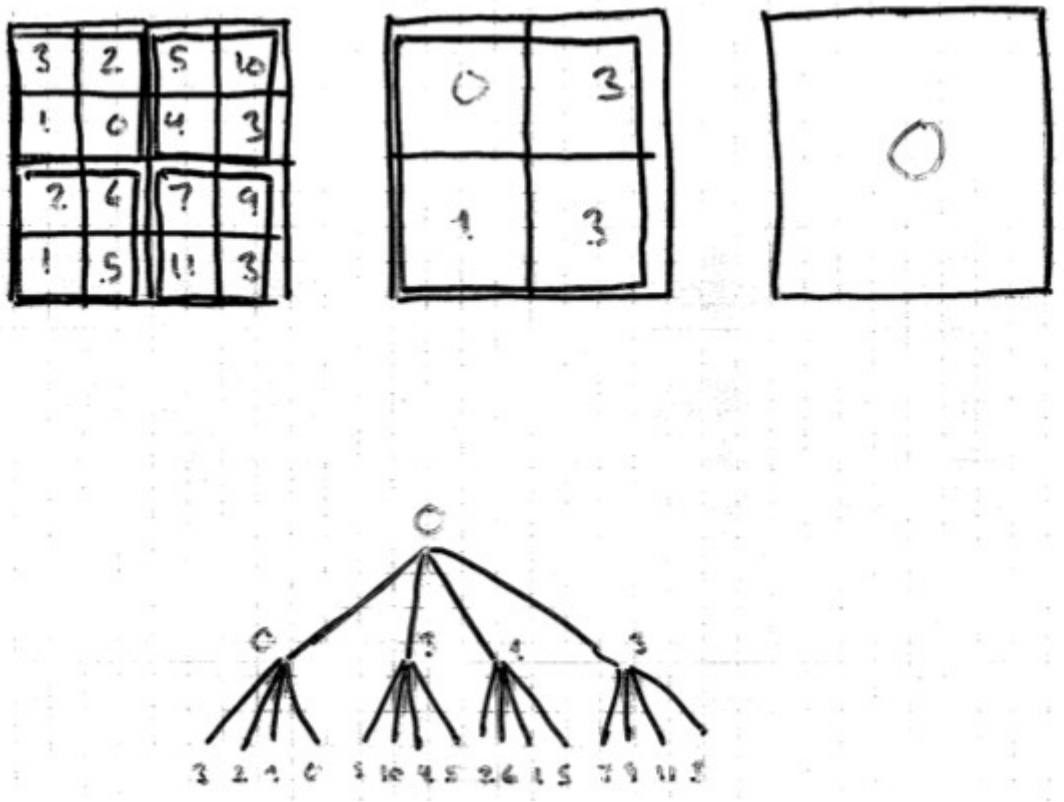
Step 3: Update distance matrix *D* such that distance between the new cluster C_k and all remaining clusters C_l are computed as:

$$D(C_k, C_\ell) = \frac{D(C_i, C_\ell) \cdot |C_i| + D(C_j, C_\ell) \cdot |C_j|}{|C_i| + |C_j|}$$

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

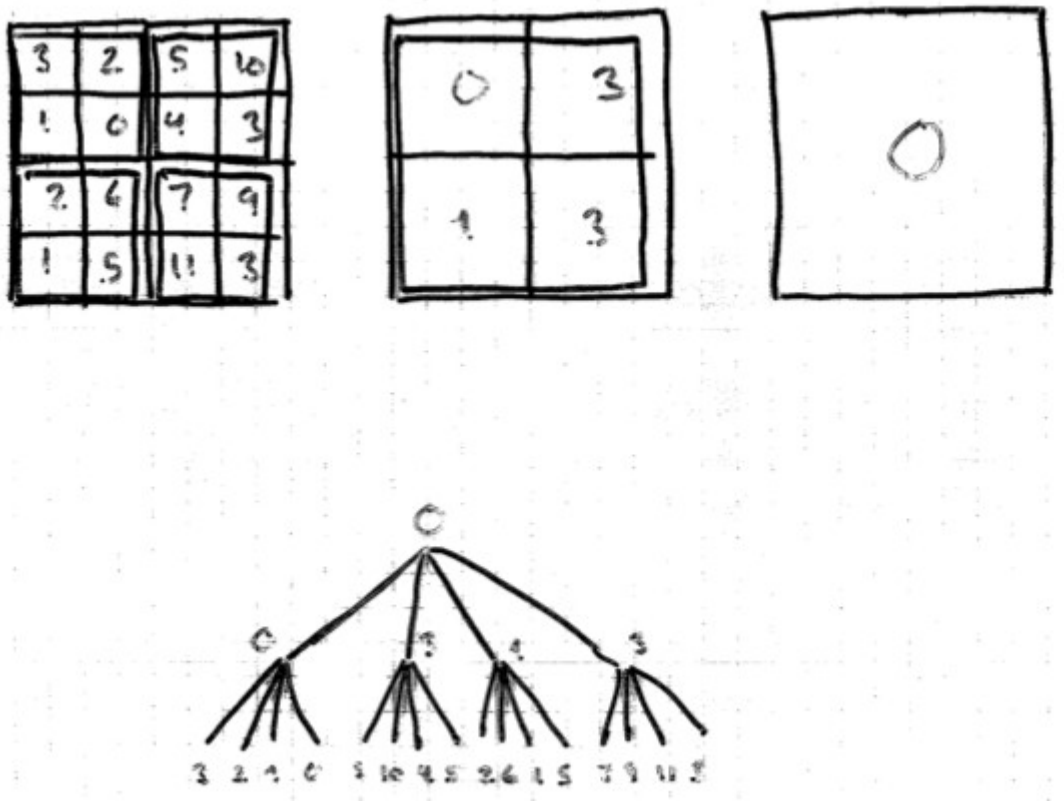
Running time $O(n^3)$. Space consumption $O(n^2)$

Storing matrix *D* in a quad tree



Building the quad tree for a *n*x*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

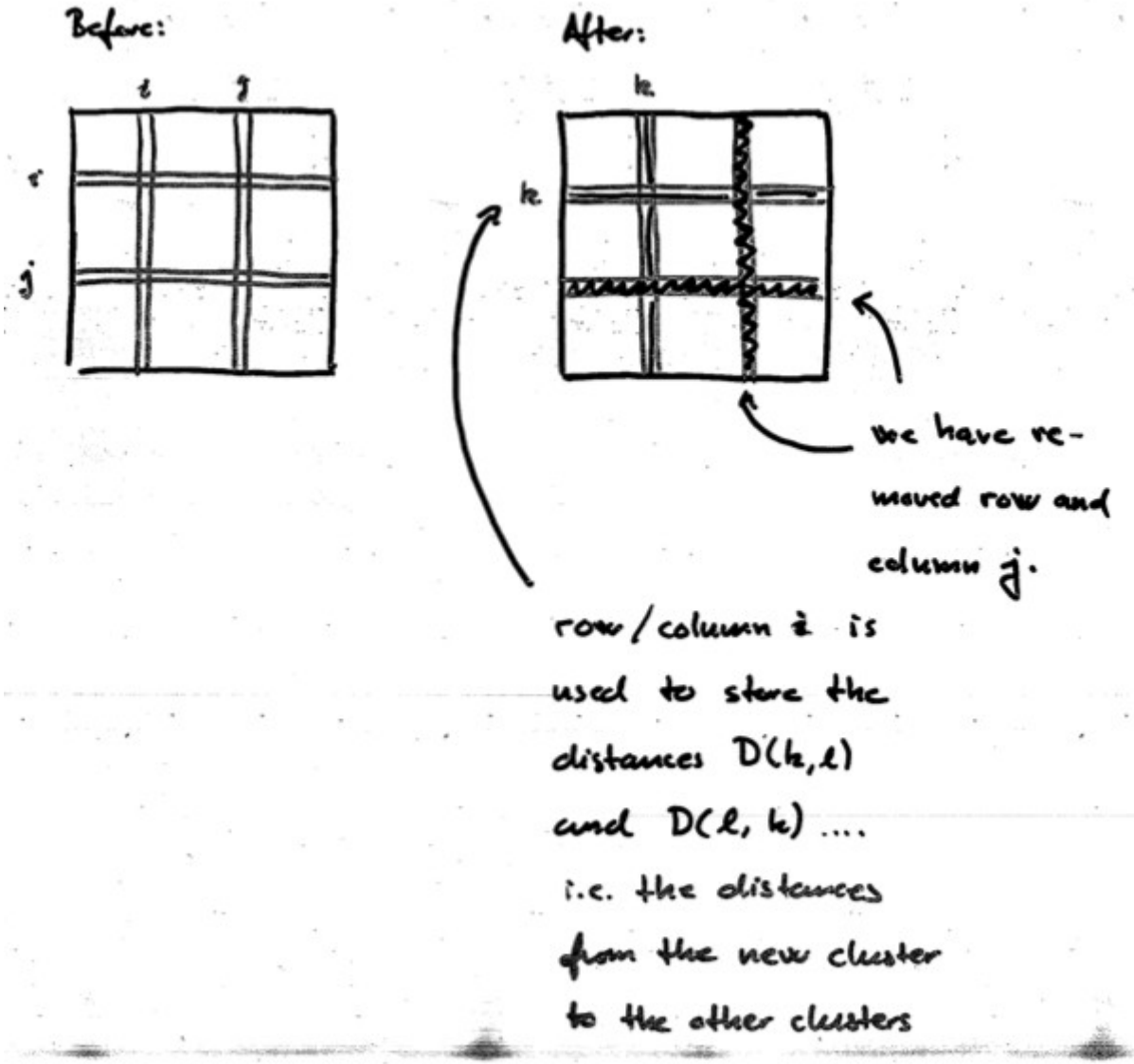


Building the quad tree for a *n*x*n* matrix takes time O(n^2), when built we can pick the minimum in constant time O(1). Not surprising.

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

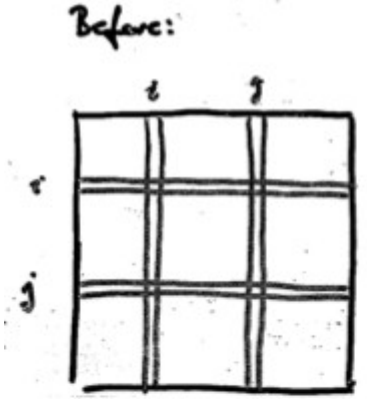
Updating the quad-tree

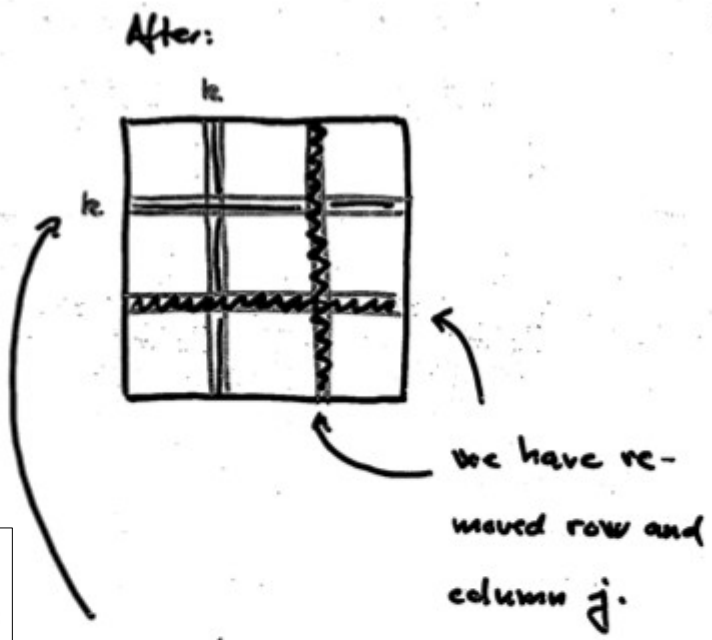
Question: How fast can we rebuilt the quad-tree to reflect the modified distance matrix we built in Step 3? In Step 3, we only change 2 row/columns in *D.*



Updating the quad-tree

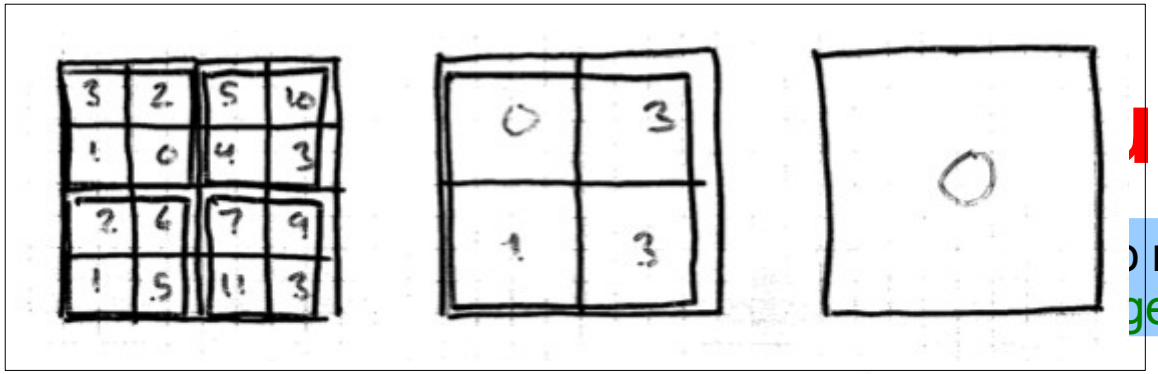
Question: How fast can we rebuilt the quad-tree to reflect the modified distance matrix we built in Step 3? In Step 3, we only change 2 row/columns in *D.*





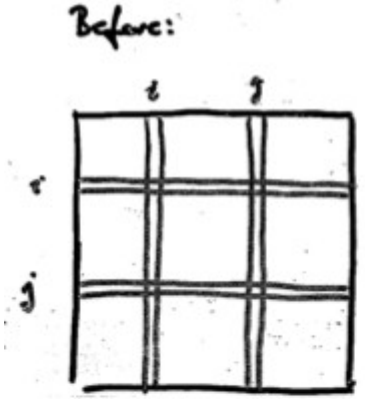
... we update two row/columns in each iterations, how long does it take to rebuilt the quad-tree to reflect this?

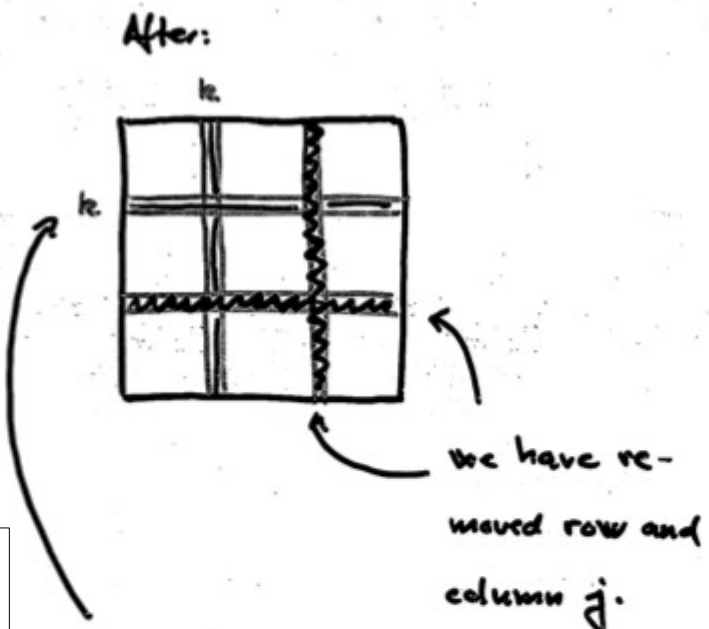
$$4 \cdot n + 4 \cdot \frac{n}{2} + 4 \cdot \frac{n}{4} + \ldots + 4 = 4\left(n + \frac{n}{2} + \ldots + 1\right) < 8n = O(n)$$



uad-tree

reflect the modified distance
ge 2 row/columns in *D*.





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

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

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 all root-leaf paths have equal length, similar species (cf. the distance matrix) are grouped in the same subtree, and the tree distances correspond to the distance matrix if possible.

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:

$$D(C_k, C_\ell) = \frac{D(C_i, C_\ell) \cdot |C_i| + D(C_j, C_\ell) \cdot |C_j|}{|C_i| + |C_j|}$$

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 all root-leaf paths have equal length, similar species (cf. the distance matrix) are grouped in the same subtree, and the tree distances correspond to the distance matrix if possible.

Algorithm:

O(n)

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 $d(i,j)$.

O(1) using quad-tree, $O(n^2)$ without quad-tree

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 $D(C_i, C_j) / 2$.

O(n) for computing new distance and rebuilding quad-tree

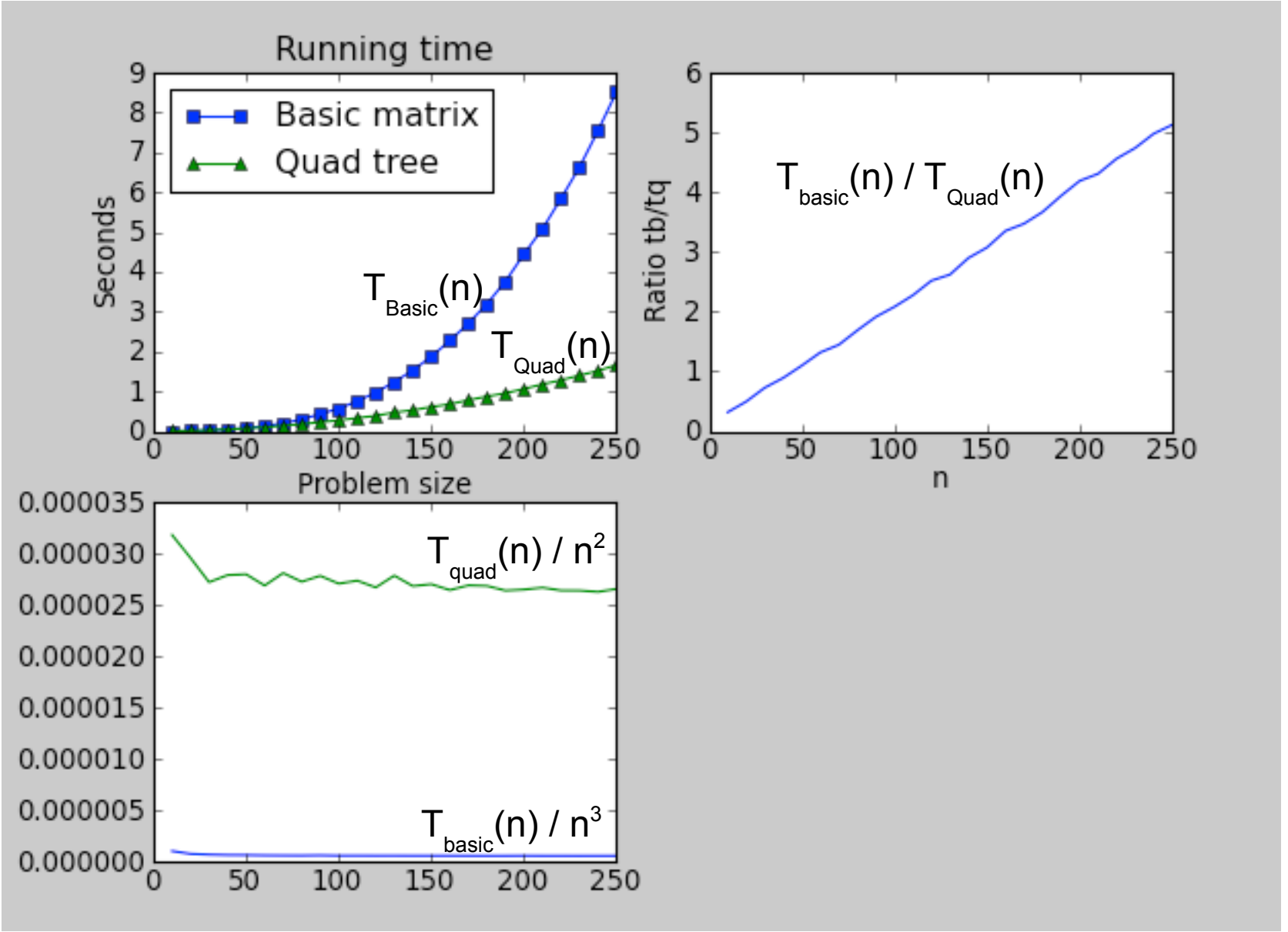
Step 3: Update distance matrix *D* such that distance between the new cluster C_k and all remaining clusters C_l are computed as:

$$D(C_k, C_\ell) = \frac{D(C_i, C_\ell) \cdot |C_i| + D(C_j, C_\ell) \cdot |C_j|}{|C_i| + |C_j|}$$

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 $O(n^2)$

Does it matter in practice?



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 (cf. the distance matrix) are grouped in the same subtree, and the tree distances correspond to the distance matrix if possible

Popular distance based phylogenetic inference method with good accuracy.

Introduced by Saitou and Nei in 1987.

Reformulated by Studier and Keppler in 1988.

Criticized by many but still widely used.

Often used as a benchmark for other methods.

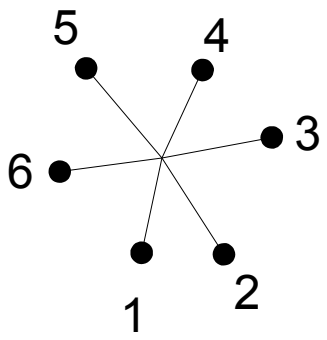
Neighbor Joining

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 (cf. the distance matrix) are grouped in the same subtree, and the tree distances correspond to the distance matrix if possible

A **divisive** method: All species (observations) start in one cluster, and at each step of the algorithm, clusters are partitioned into a pair of clusters (selected wisely).

The **neighbor joining** idea: If we (from the distance matrix) can infer which pair of observations (a leaf or subtree) are neighbors in the unknown true tree ,then we can reconstruct the tree iteratively by joining these neighbor in a new subtree.



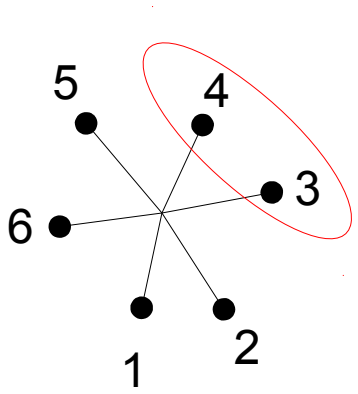
Neighbor Joining

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 (cf. the distance matrix) are grouped in the same subtree, and the tree distances correspond to the distance matrix if possible

A **divisive** method: All species (observations) start in one cluster, and at each step of the algorithm, clusters are partitioned into a pair of clusters (selected wisely).

The **neighbor joining** idea: If we (from the distance matrix) can infer which pair of observations (a leaf or subtree) are neighbors in the unknown true tree ,then we can reconstruct the tree iteratively by joining these neighbor in a new subtree.



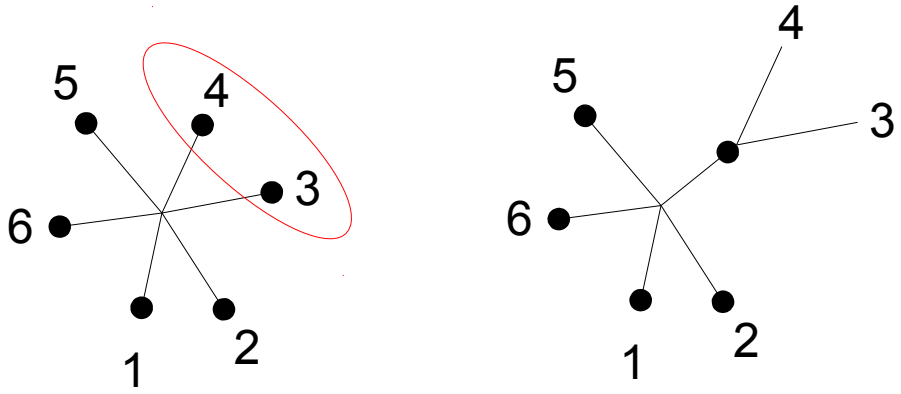
Neighbor Joining

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 (cf. the distance matrix) are grouped in the same subtree, and the tree distances correspond to the distance matrix if possible

A **divisive** method: All species (observations) start in one cluster, and at each step of the algorithm, clusters are partitioned into a pair of clusters (selected wisely).

The **neighbor joining** idea: If we (from the distance matrix) can infer which pair of observations (a leaf or subtree) are neighbors in the unknown true tree ,then we can reconstruct the tree iteratively by joining these neighbor in a new subtree.



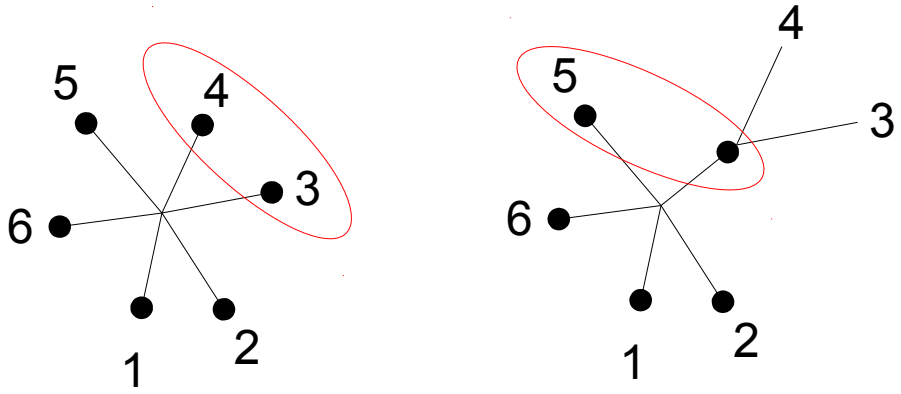
Neighbor Joining

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 (cf. the distance matrix) are grouped in the same subtree, and the tree distances correspond to the distance matrix if possible

A **divisive** method: All species (observations) start in one cluster, and at each step of the algorithm, clusters are partitioned into a pair of clusters (selected wisely).

The **neighbor joining** idea: If we (from the distance matrix) can infer which pair of observations (a leaf or subtree) are neighbors in the unknown true tree ,then we can reconstruct the tree iteratively by joining these neighbor in a new subtree.



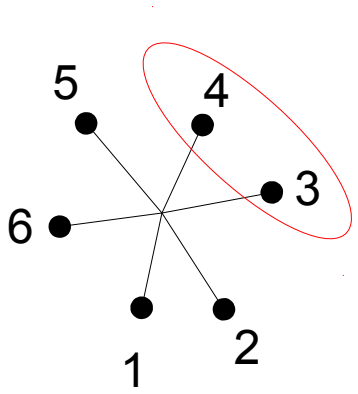
Neighbor Joining

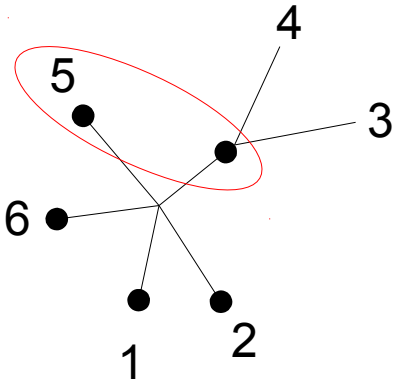
Input: A set of *n* species (1,2,...,n) and a distance matrix *d* giving the pairwise distances.

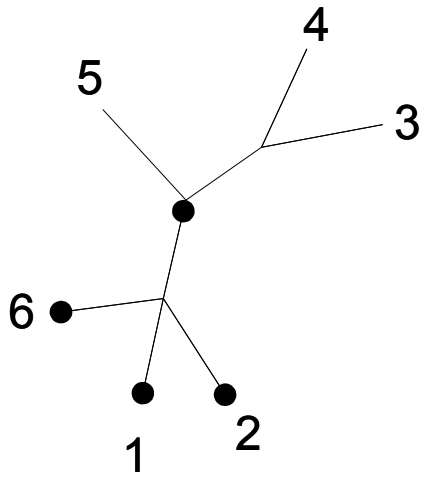
Output: An unrooted binary tree *T* with edge lengths and leaves labelled 1, 2, ..., *n* such that similar species (cf. the distance matrix) are grouped in the same subtree, and the tree distances correspond to the distance matrix if possible

A **divisive** method: All species (observations) start in one cluster, and at each step of the algorithm, clusters are partitioned into a pair of clusters (selected wisely).

The **neighbor joining** idea: If we (from the distance matrix) can infer which pair of observations (a leaf or subtree) are neighbors in the unknown true tree ,then we can reconstruct the tree iteratively by joining these neighbor in a new subtree.







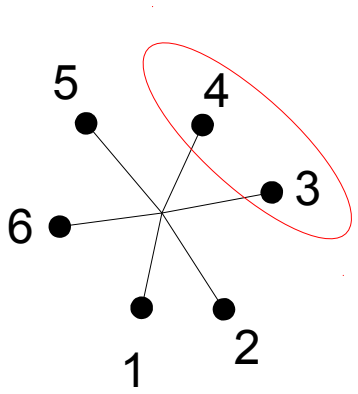
Neighbor Joining

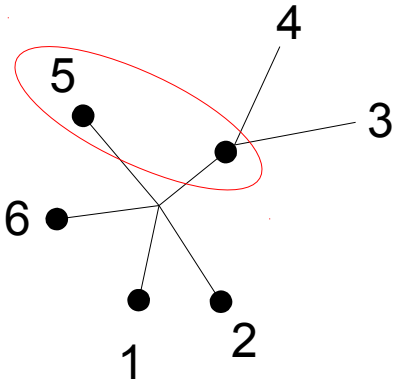
Input: A set of *n* species (1,2,...,n) and a distance matrix *d* giving the pairwise distances.

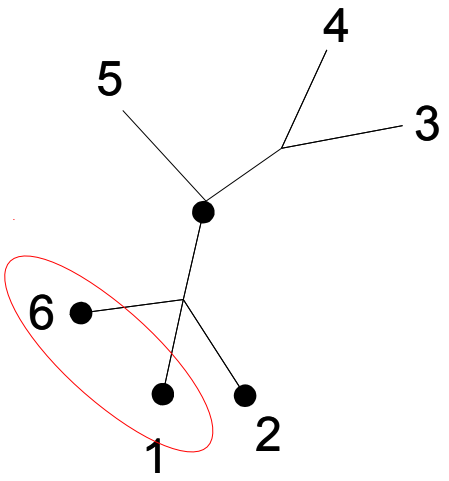
Output: An unrooted binary tree *T* with edge lengths and leaves labelled 1, 2, ..., *n* such that similar species (cf. the distance matrix) are grouped in the same subtree, and the tree distances correspond to the distance matrix if possible

A **divisive** method: All species (observations) start in one cluster, and at each step of the algorithm, clusters are partitioned into a pair of clusters (selected wisely).

The **neighbor joining** idea: If we (from the distance matrix) can infer which pair of observations (a leaf or subtree) are neighbors in the unknown true tree ,then we can reconstruct the tree iteratively by joining these neighbor in a new subtree.







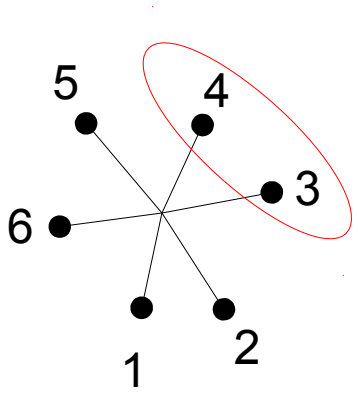
Neighbor Joining

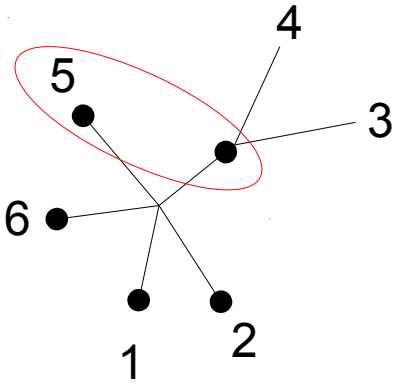
Input: A set of *n* species (1,2,...,n) and a distance matrix *d* giving the pairwise distances.

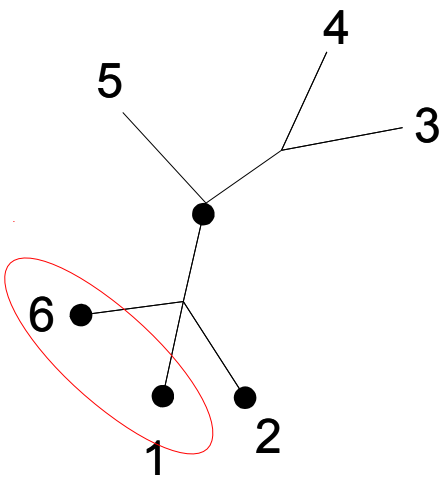
Output: An unrooted binary tree *T* with edge lengths and leaves labelled 1, 2, ..., *n* such that similar species (cf. the distance matrix) are grouped in the same subtree, and the tree distances correspond to the distance matrix if possible

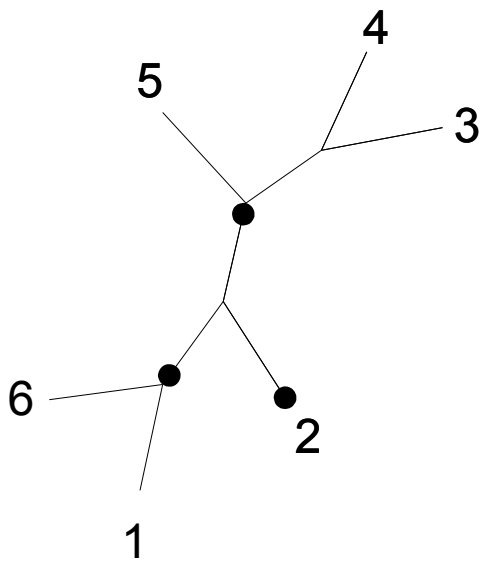
A **divisive** method: All species (observations) start in one cluster, and at each step of the algorithm, clusters are partitioned into a pair of clusters (selected wisely).

The **neighbor joining** idea: If we (from the distance matrix) can infer which pair of observations (a leaf or subtree) are neighbors in the unknown true tree ,then we can reconstruct the tree iteratively by joining these neighbor in a new subtree.









Neighbor Joining

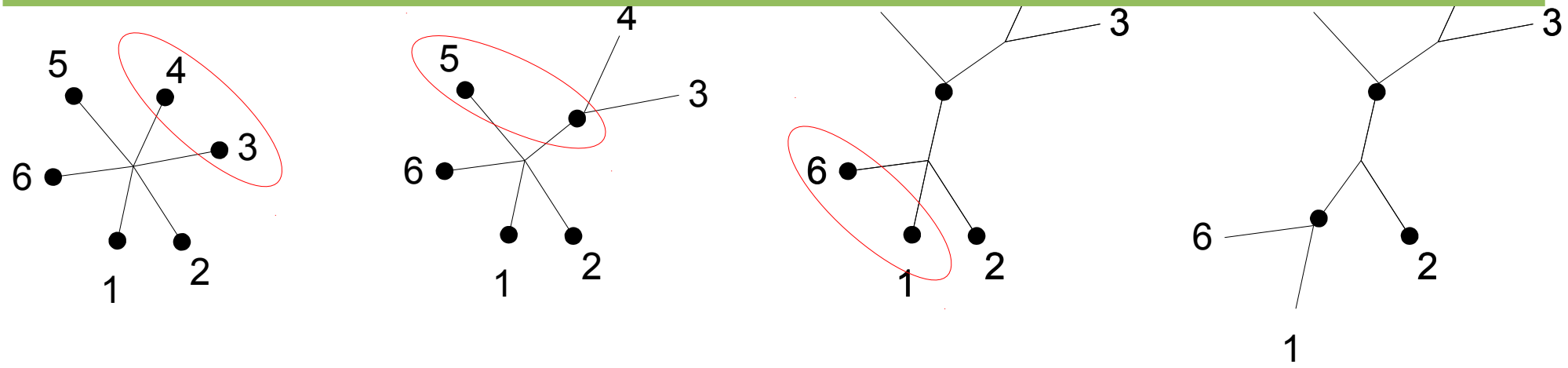
Input: A set of *n* species (1,2,...,n) and a distance matrix *d* giving the pairwise distances.

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

A **divisive** method: All species (observations) start in one cluster, and at each step of the algorithm, clusters are partitioned into a pair of clusters (selected wisely).

The **neighbor joining** idea: If we (from the distance matrix) can infer which pair of observations (a leaf or subtree) are neighbors in the unknown true tree ,then we can reconstruct the tree iteratively by joining these neighbor in a new subtree.

What about edge lengths? If we assume that the unknown true tree is additive and is consistent with the distance matrix, then there exists a simple method:

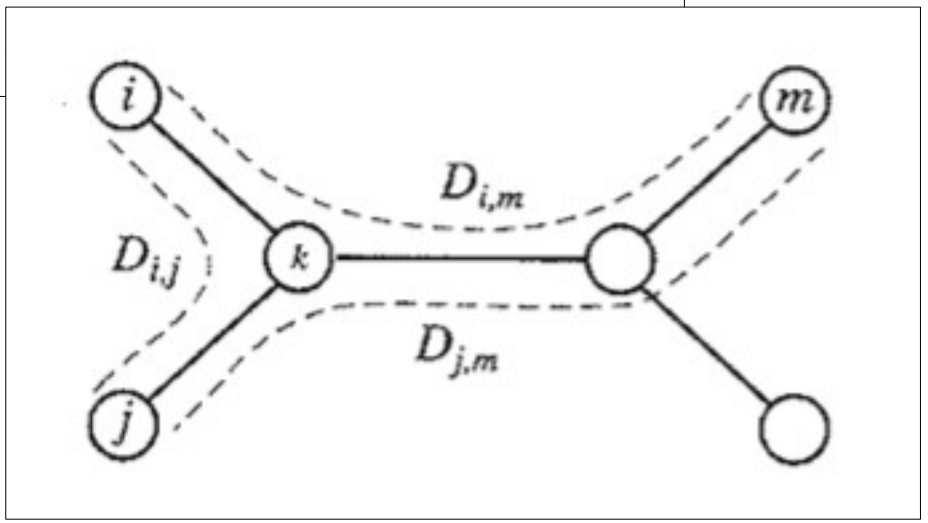


Neighbor Joining – Edge lenghts

A "simple" way to solve the Distance-Based Phylogeny problem for additive trees[16] is to find a pair of *neighboring* leaves, that is, leaves that have the same parent vertex.[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 $D_{k,m} = \frac{D_{i,m}+D_{j,m}-D_{i,j}}{2}$.



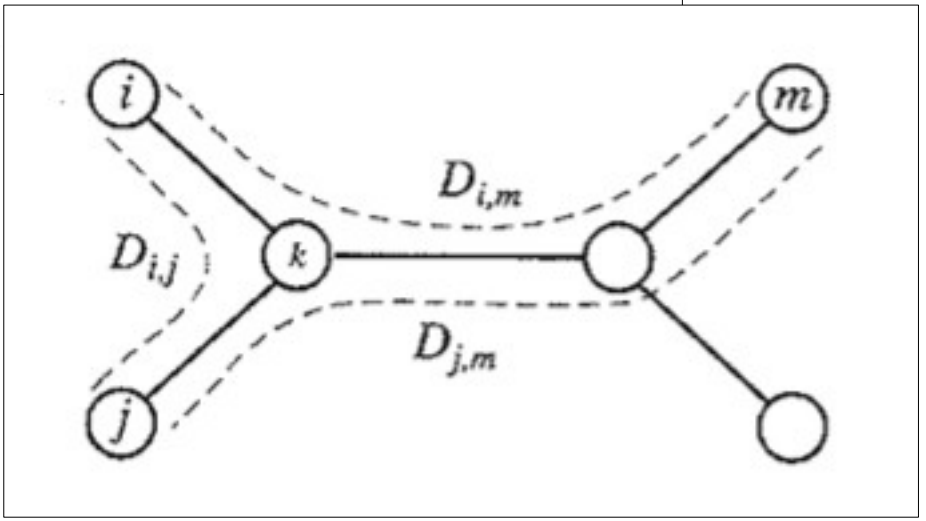
Neighbor Joining – Edge lenghts

A "simple" way to solve the Distance-Based Phylogeny problem for additive trees[16] is to find a pair of *neighboring* leaves, that is, leaves that have the same parent vertex.[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:

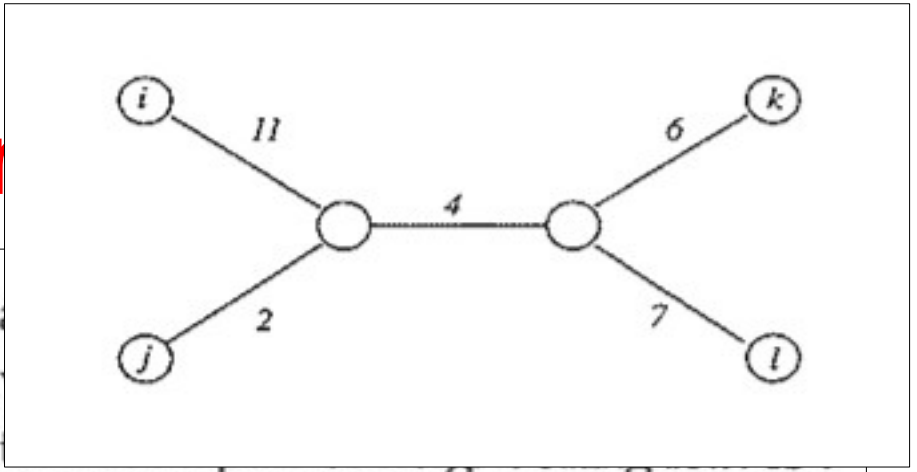
$$D_{k,m} = \frac{D_{i,m} + D_{j,m} - D_{i,j}}{2}$$

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

How do we infer neighboring leaves?



Neighbor Joini



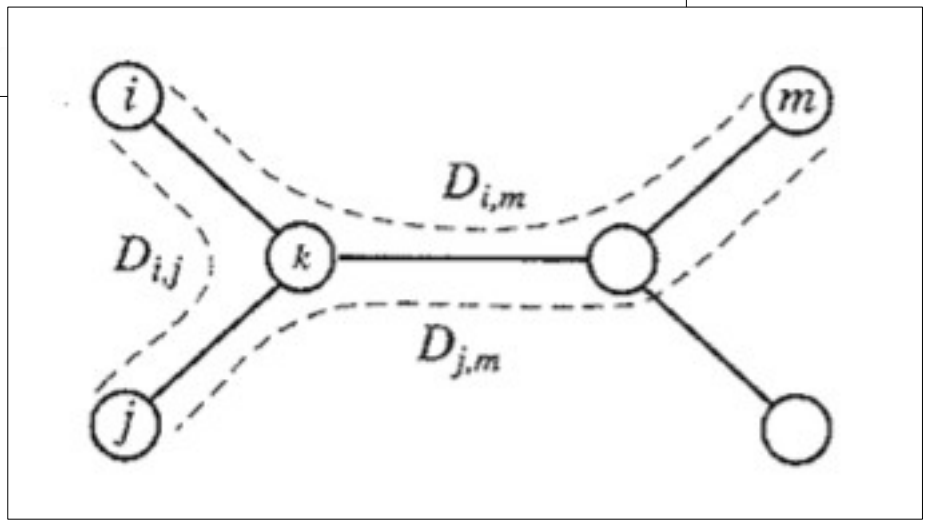
A "simple" way to solve the Distance-B
trees[16] is to find a pair of *neighboring* lea
parent vertex.[17] Figure 10.12 illustrates
and j and their pare
leaf m in the tree:

Therefore, as soon
remove the corresp
trix and add a new r
the distance matrix i
computed as $D_{k,m} = \frac{D_{i,m}+D_{j,m}-D_{i,j}}{2}$.

First idea: Pick leaves with minimum distance (like in UPGMA):

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.

How do we infer neighboring leaves?



Algorithm 10.4 Generic neighbor-joining algorithm.

Input: $n \times n$ dissimilarity matrix D, where $n \geq 3$.

Initialization:

1. Let $S = \{1, \ldots, n\}$ be the set of taxa.
2. Each taxon i is a leaf in the tree T.

while $|S| > 3$ **do**

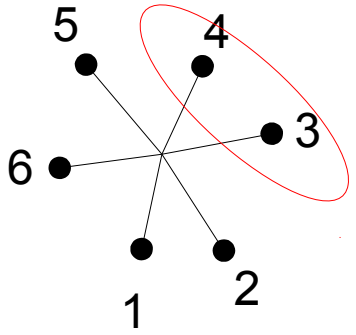
1. Using a specific neighbor selection criterion, select two taxa i and j that are leaf neighbors in the (yet unknown) tree T.
2. Add a new node k to the tree T.
3. Choose an $m \in S \setminus \{i, j\}$ and add edges (k, i) and (k, j) with weights $\gamma(k, i) = \frac{1}{2}(d_{im} - d_{jm} + d_{ij})$ and $\gamma(k, j) = d_{ij} - \gamma(k, i) = \frac{1}{2}(d_{jm} - d_{im} + d_{ij})$ to the tree T.
4. Update the dissimilarity matrix by deleting the rows and columns corresponding to i and j and adding a new row and column for the new taxon k with $d_{km} = \frac{1}{2}(d_{im} + d_{jm} - d_{ij})$ for all $m \in S \setminus \{i, j, k\}$.
5. Delete i and j from S and add the new (artificial) taxon k to S.

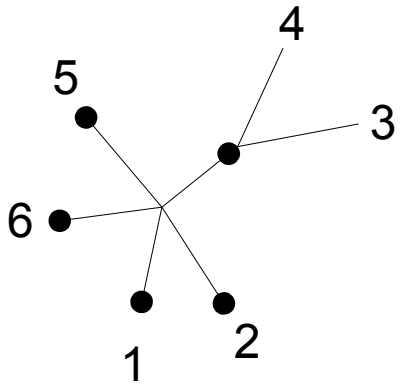
Termination:

Let i, j, m be the remaining three taxa. Add a new internal node v to the tree T, and add edges (v, i), (v, j), and (v, m) to the tree T with weights

$$\gamma(v, i) = \frac{d_{ij} + d_{im} - d_{jm}}{2}$$
$$\gamma(v, j) = \frac{d_{ij} + d_{jm} - d_{im}}{2}$$
$$\gamma(v, m) = \frac{d_{im} + d_{jm} - d_{ij}}{2}$$

Output: The tree T.





Algorithm 10.4 Generic neighbor-joining algorithm.

Input: $n \times n$ dissimilarity matrix D, where $n \geq 3$.

Initialization:

1. Let $S = \{1, \ldots, n\}$ be the set of taxa.
2. Each taxon i is a leaf in the tree T.

while $|S| > 3$ **do**

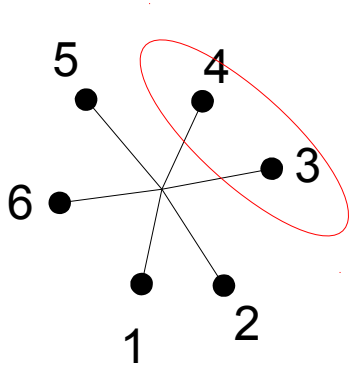
1. Using a specific neighbor selection criterion, select two taxa i and j that are leaf neighbors in the (yet unknown) tree T.
2. Add a new node k to the tree T.
3. Choose an $m \in S \setminus \{i, j\}$ and add edges (k, i) and (k, j) with weights $\gamma(k, i) = \frac{1}{2}(d_{im} - d_{jm} + d_{ij})$ and $\gamma(k, j) = d_{ij} - \gamma(k, i) = \frac{1}{2}(d_{jm} - d_{im} + d_{ij})$ to the tree T.
4. Update the dissimilarity matrix by deleting the rows and columns corresponding to i and j and adding a new row and column for the new taxon k with $d_{km} = \frac{1}{2}(d_{im} + d_{jm} - d_{ij})$ for all $m \in S \setminus \{i, j, k\}$.
5. Delete i and j from S and add the new (artificial) taxon k to S.

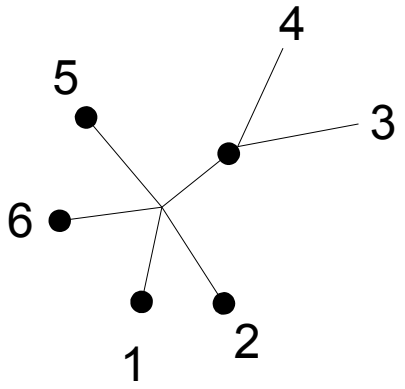
Termination:

Let i, j, m be the remaining three taxa. Add a new internal node v to the tree T, and add edges (v, i), (v, j), and (v, m) to the tree T with weights

$$\gamma(v, i) = \frac{d_{ij} + d_{im} - d_{jm}}{2}$$
$$\gamma(v, j) = \frac{d_{ij} + d_{jm} - d_{im}}{2}$$
$$\gamma(v, m) = \frac{d_{im} + d_{jm} - d_{ij}}{2}$$

Output: The tree T.





Algorithm 10.4 Generic neighbor-joining algorithm.

Input: $n \times n$ dissimilarity matrix D, where $n \geq 3$.

Initialization:

1. Let $S = \{1, \ldots, n\}$ be the set of taxa.
2. Each taxon i is a leaf in the tree T.

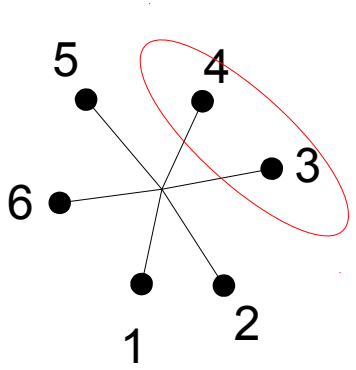
while $|S| > 3$ **do**

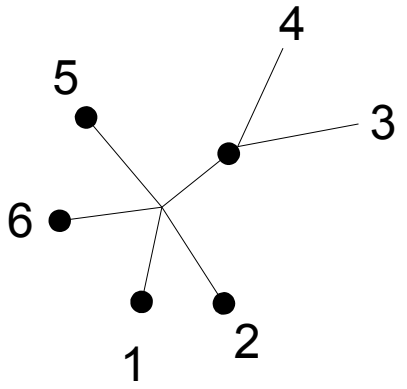
1. Using a specific neighbor selection criterion, select two taxa i and j that are leaf neighbors in the (yet unknown) tree T.
2. Add a new node k to the tree T.
3. Choose an $m \in S \setminus \{i, j\}$ and add edges (k, i) and (k, j) with weights $\gamma(k, i) = \frac{1}{2}(d_{im} - d_{jm} + d_{ij})$ and $\gamma(k, j) = d_{ij} - \gamma(k, i) = \frac{1}{2}(d_{jm} - d_{im} + d_{ij})$ to the tree T.
4. Update the dissimilarity matrix by deleting the rows and columns corresponding to i and j and adding a new row and column for the new taxon k with $d_{km} = \frac{1}{2}(d_{im} + d_{jm} - d_{ij})$ for all $m \in S \setminus \{i, j, k\}$.
5. Delete i and j from S and add the new (artificial) taxon k to S.

Termination:

Let i, j, m be the remaining three taxa. Add a new internal node v to the tree T, and add edges (v, i), (v, j), and (v, m) to the tree T with weights

Output: The tree T.





Theorem 10.5.1 *If a dissimilarity matrix D is consistent with an additive binary tree T and the neighbor selection criterion truly identifies leaf neighbors, then the generic neighbor-joining algorithm (Algorithm 10.4) applied to D constructs T.*

Neighbor Joining – Finding Neighbors

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 – Finding Neighbors

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 $u(C) = \frac{1}{\text{number of clusters}-2} \sum_{\text{all clusters } C'} D(C, C')$ as a measure of the separation of C from other clusters.[20] 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 – Finding Neighbors

10.5.2 Saitou and Nei's neighbor-joining algorithm

The most popular neighbor-joining algorithm is due to Saitou and Nei [120, 276, 303]. In the literature, it is most often referred to as "the neighbor-joining algorithm." Its neighbor selection criterion uses the matrix $N = (n_{ij})_{i,j \in S}$ defined by

$$n_{ij} = d_{ij} - (r_i + r_j),$$

where

$$r_i = \frac{1}{|S| - 2} \sum_{m \in S} d_{im}$$

and selects $i, j \in S$ so that n_{ij} is a minimum entry in N. Saitou and Nei's neighbor-joining algorithm is presented in Algorithm 10.7.

For a cluster C, define $u(C) = \frac{1}{\text{number of clusters} - 2} \sum_{\text{all clusters } C'} D(C, C')$ as a measure of the separation of C from other clusters.[20] To choose which two cluster
close t
simult
it is un

Theorem 10.5.6 *If a dissimilarity matrix D is consistent with an additive binary tree T and n_{ij} is a minimum entry in the corresponding N matrix, then i and j are leaf neighbors in T.*

$D(C_1, C_2)$ 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.

Saitou and Nei's NJ Algorithm

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

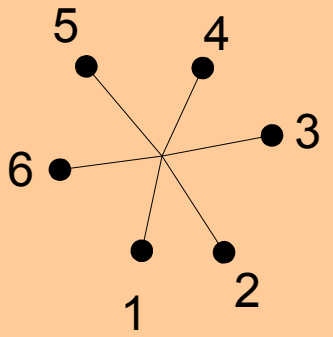
Output: An unrooted binary tree T with edge lengths and leaves labelled 1, 2, ..., n such that similar species (cf. the distance matrix) are grouped in the same subtree, and the tree distances correspond to the distance matrix if possible

Algorithm:

Input: $n \times n$ dissimilarity matrix D, where $n \geq 3$.

Initialization:

1. Let $S = \{1, \ldots, n\}$ be the set of taxa.
2. Each taxon i is a leaf in the tree T.



Saitou and Nei's NJ Algorithm

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 (cf. the distance matrix) are grouped in the same subtree, and the tree distances correspond to the distance matrix if possible

Algorithm:

while $|S| > 3$ **do**

1. a) Compute the matrix $N = (n_{ij})_{i,j \in S}$, where $n_{ij} = d_{ij} - (r_i + r_j)$ and $r_i = \frac{1}{|S|-2} \sum_{m \in S} d_{im}$.

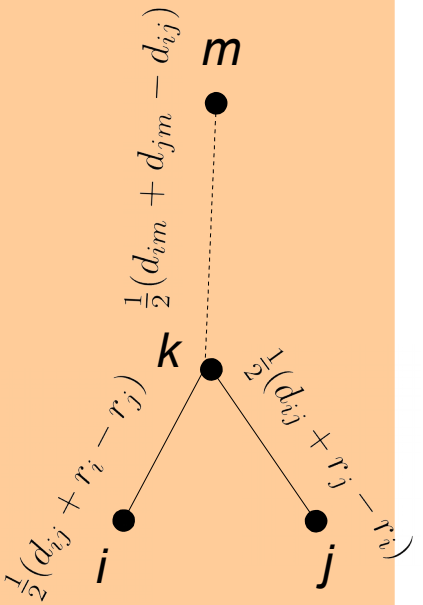
   b) Select $i, j \in S$ so that n_{ij} is a minimum entry in N.

2. Add a new node k to the tree T.

3. Add edges (k,i) and (k,j) with weights $\gamma(k,i) = \frac{1}{2}(d_{ij} + r_i - r_j)$ and $\gamma(k,j) = d_{ij} - \gamma(k,i) = \frac{1}{2}(d_{ij} + r_j - r_i)$ to the tree T.

4. Update the dissimilarity matrix by deleting the rows and columns corresponding to i and j and adding a new row and column for the new taxon k with $d_{km} = \frac{1}{2}(d_{im} + d_{jm} - d_{ij})$ for all $m \in S \setminus \{i,j,k\}$.

5. Delete i and j from S and add the new (artificial) taxon k to S.



Saitou and Nei's NJ Algorithm

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 (cf. the distance matrix) are grouped in the same subtree, and the tree distances correspond to the distance matrix if possible

Algorithm:

Termination:

Let i, j, m be the remaining three taxa. Add a new internal node v to the tree T, and add edges (v, i), (v, j), and (v, m) to the tree T with weights

$$\gamma(v,i) = \frac{d_{ij} + d_{im} - d_{jm}}{2}$$

$$\gamma(v,j) = \frac{d_{ij} + d_{jm} - d_{im}}{2}$$

$$\gamma(v,m) = \frac{d_{im} + d_{jm} - d_{ij}}{2}$$

Output: The tree T.

Input: A set of *r*

Output: An unr

that similar spec

and the tree dis

Algorithm:

Algorithm 10.7 Saitou and Nei's neighbor-joining algorithm.

Input: $n \times n$ dissimilarity matrix D, where $n \geq 3$.

Initialization:

1. Let $S = \{1, \ldots, n\}$ be the set of taxa.
2. Each taxon i is a leaf in the tree T.

while $|S| > 3$ **do**

1. a) Compute the matrix $N = (n_{ij})_{i,j \in S}$, where $n_{ij} = d_{ij} - (r_i + r_j)$ and $r_i = \frac{1}{|S|-2} \sum_{m \in S} d_{im}$.

   b) Select $i, j \in S$ so that n_{ij} is a minimum entry in N.
2. Add a new node k to the tree T.
3. Add edges (k, i) and (k, j) with weights $\gamma(k, i) = \frac{1}{2}(d_{ij} + r_i - r_j)$ and $\gamma(k, j) = d_{ij} - \gamma(k, i) = \frac{1}{2}(d_{ij} + r_j - r_i)$ to the tree T.
4. Update the dissimilarity matrix by deleting the rows and columns corresponding to i and j and adding a new row and column for the new taxon k with $d_{km} = \frac{1}{2}(d_{im} + d_{jm} - d_{ij})$ for all $m \in S \setminus \{i, j, k\}$.
5. Delete i and j from S and add the new (artificial) taxon k to S.

Termination:

Let i, j, m be the remaining three taxa. Add a new internal node v to the tree T, and add edges (v, i), (v, j), and (v, m) to the tree T with weights

$$\gamma(v, i) = \frac{d_{ij} + d_{im} - d_{jm}}{2}$$

$$\gamma(v, j) = \frac{d_{ij} + d_{jm} - d_{im}}{2}$$

$$\gamma(v, m) = \frac{d_{im} + d_{jm} - d_{ij}}{2}$$

Output: The tree T.

se distances.

2, ..., *n* such

ne subtree,

Running time and space consumption

Algorithm 10.7 Saitou and Nei's neighbor-joining algorithm.

Input: $n \times n$ dissimilarity matrix D, where $n \geq 3$.

Initialization:

1. Let $S = \{1, \ldots, n\}$ be the set of taxa.
2. Each taxon i is a leaf in the tree T.

while $|S| > 3$ **do**

1. a) Compute the matrix $N = (n_{ij})_{i,j \in S}$, where $n_{ij} = d_{ij} - (r_i + r_j)$ and $r_i = \frac{1}{|S|-2} \sum_{m \in S} d_{im}$.
   b) Select $i, j \in S$ so that n_{ij} is a minimum entry in N.
2. Add a new node k to the tree T.
3. Add edges (k, i) and (k, j) with weights $\gamma(k, i) = \frac{1}{2}(d_{ij} + r_i - r_j)$ and $\gamma(k, j) = d_{ij} - \gamma(k, i) = \frac{1}{2}(d_{ij} + r_j - r_i)$ to the tree T.
4. Update the dissimilarity matrix by deleting the rows and columns corresponding to i and j and adding a new row and column for the new taxon k with $d_{km} = \frac{1}{2}(d_{im} + d_{jm} - d_{ij})$ for all $m \in S \setminus \{i, j, k\}$.
5. Delete i and j from S and add the new (artificial) taxon k to S.

Termination:

Let i, j, m be the remaining three taxa. Add a new internal node v to the tree T, and add edges (v, i), (v, j), and (v, m) to the tree T with weights

$$\gamma(v, i) = \frac{d_{ij} + d_{im} - d_{jm}}{2}$$
$$\gamma(v, j) = \frac{d_{ij} + d_{jm} - d_{im}}{2}$$
$$\gamma(v, m) = \frac{d_{im} + d_{jm} - d_{ij}}{2}$$

Output: The tree T.

Initilization: O(n)

n iterations each taking:

Step 1:
Computing the row sum r_i for every *i* takes O(n^2) time.

Computing the matrix *N* takes O(n^2) time

Finding the minimum n_{ij} takes O(n^2) time

Step 2 :
Adding node *k* takes O(1) time

Step 3:
Adding two edges takes O(1) time

Step 4:
Updating the distance matrix takes O(n) time

Total running time: O(n^3)

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

Alg

Input: $n \times n$ dissimilarity matrix D, where $n \geq 3$.

Initialization:

1. Let $S = \{1, \ldots, n\}$ be the set of taxa.
2. Each taxon i is a leaf in the tree T.

while $|S| > 3$ **do**

1. a) Compute the matrix $N = (n_{ij})_{i,j \in S}$, where $n_{ij} = d_{ij} - (r_i + r_j)$ and $r_i = \frac{1}{|S|-2} \sum_{m \in S} d_{im}$.
   b) Select $i, j \in S$ so that n_{ij} is a minimum entry in N.
2. Add a new node k to the tree T.
3. Add edges (k,i) and (k,j) with weights $\gamma(k,i) = \frac{1}{2}(d_{ij} + r_i - r_j)$ and $\gamma(k,j) = d_{ij} - \gamma(k,i) = \frac{1}{2}(d_{ij} + r_j - r_i)$ to the tree T.
4. Update the dissimilarity matrix by deleting the rows and columns corresponding to i and j and adding a new row and column for the new taxon k with $d_{km} = \frac{1}{2}(d_{im} + d_{jm} - d_{ij})$ for all $m \in S \setminus \{i, j, k\}$.
5. Delete i and j from S and add the new (artificial) taxon k to S.

Termination:

Let i, j, m be the remaining three taxa. Add a new internal node v to the tree T, and add edges (v,i), (v,j), and (v,m) to the tree T with weights

$$\gamma(v,i) = \frac{d_{ij} + d_{im} - d_{jm}}{2}$$
$$\gamma(v,j) = \frac{d_{ij} + d_{jm} - d_{im}}{2}$$
$$\gamma(v,m) = \frac{d_{im} + d_{jm} - d_{ij}}{2}$$

Output: The tree T.

n iterations each taking:

Step 1:
Computing the row sum r_i for every *i* takes $O(n^2)$ time.

Computing the matrix *N* takes $O(n^2)$ time

Finding the minimum n_{ij} takes $O(n^2)$ time

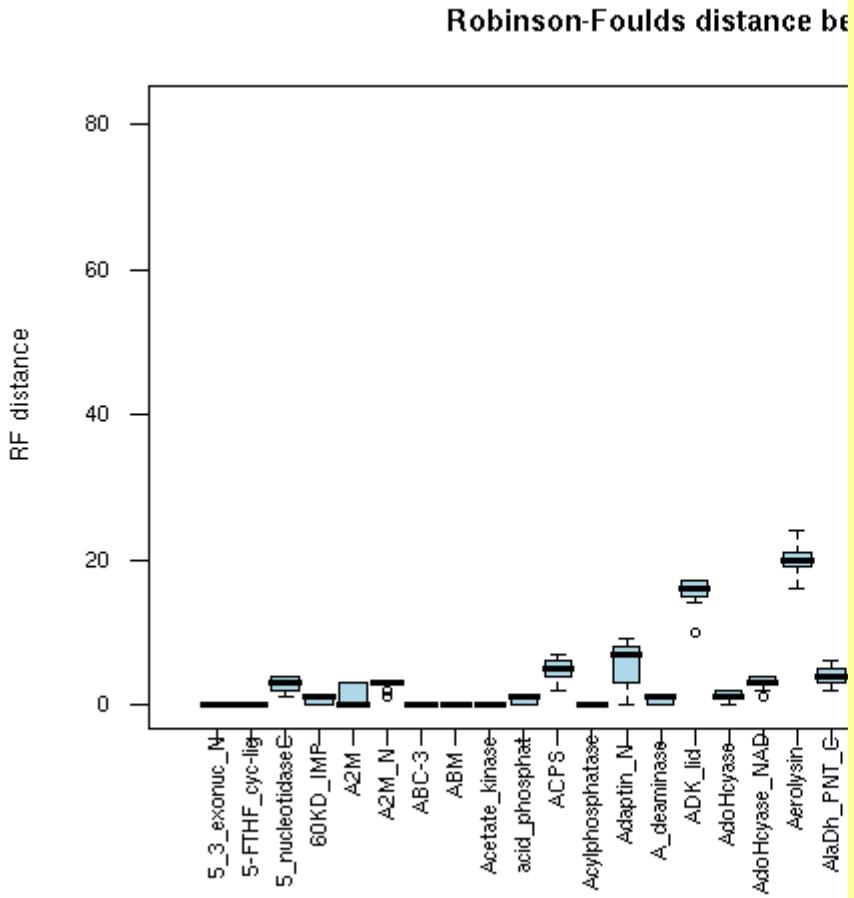
Step 2 :
Adding node *k* takes O(1) time

Step 3:
Adding two edges takes O(1) time

Step 4:
Updating the distance matrix takes O(*n*) time

Total running time: $O(n^3)$

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.

[...]

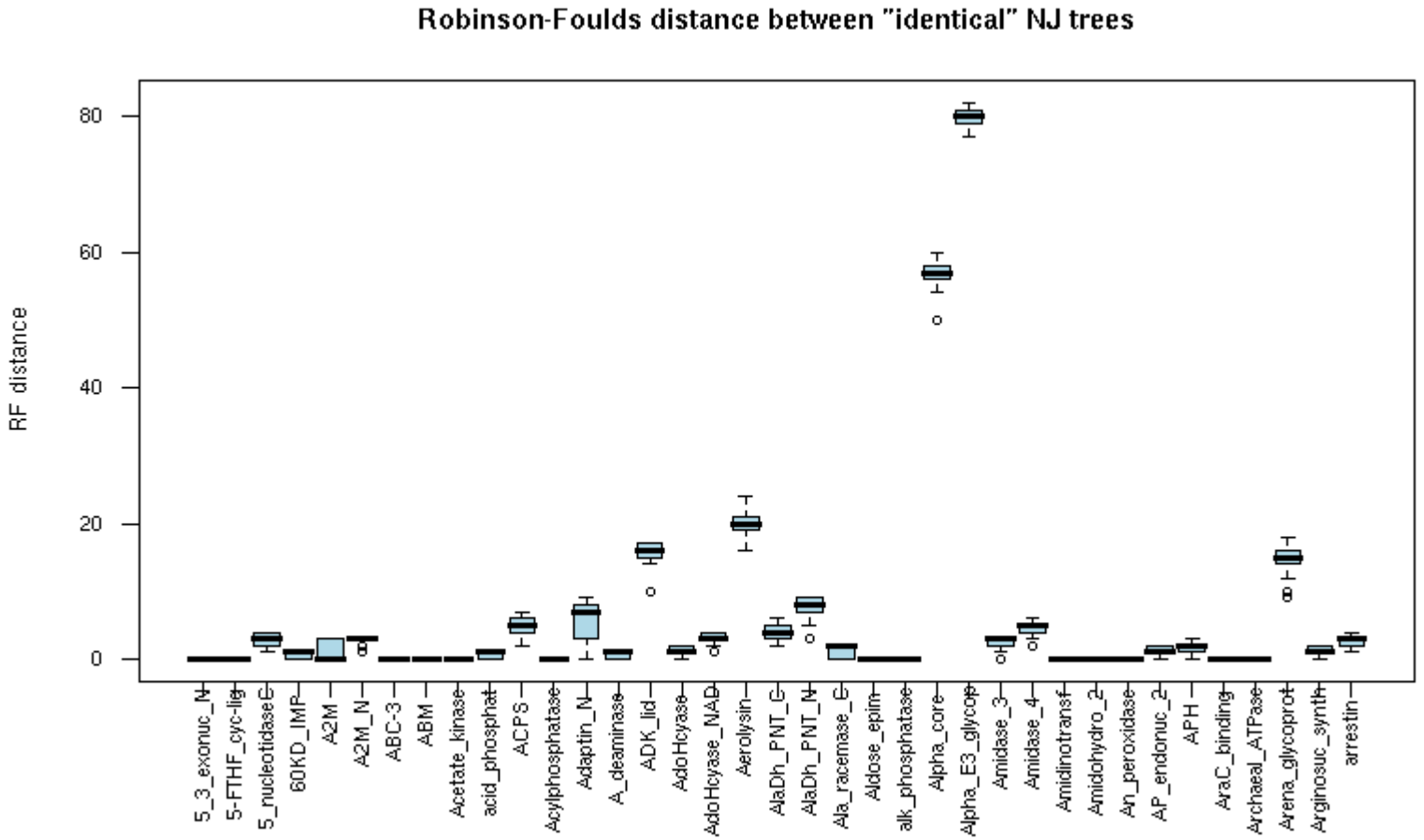
As you can see, you can get quite different trees out of equivalent distance matrices and on realistic data too.

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!

http://www.mailund.dk/index.php/2008/07/23/how-deterministic-is-neighbour-joining-anyway/

How 'deterministic' is NJ?



http://www.mailund.dk/index.php/2008/07/23/how-deterministic-is-neighbour-joining-anyway/

RapidNJ

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.

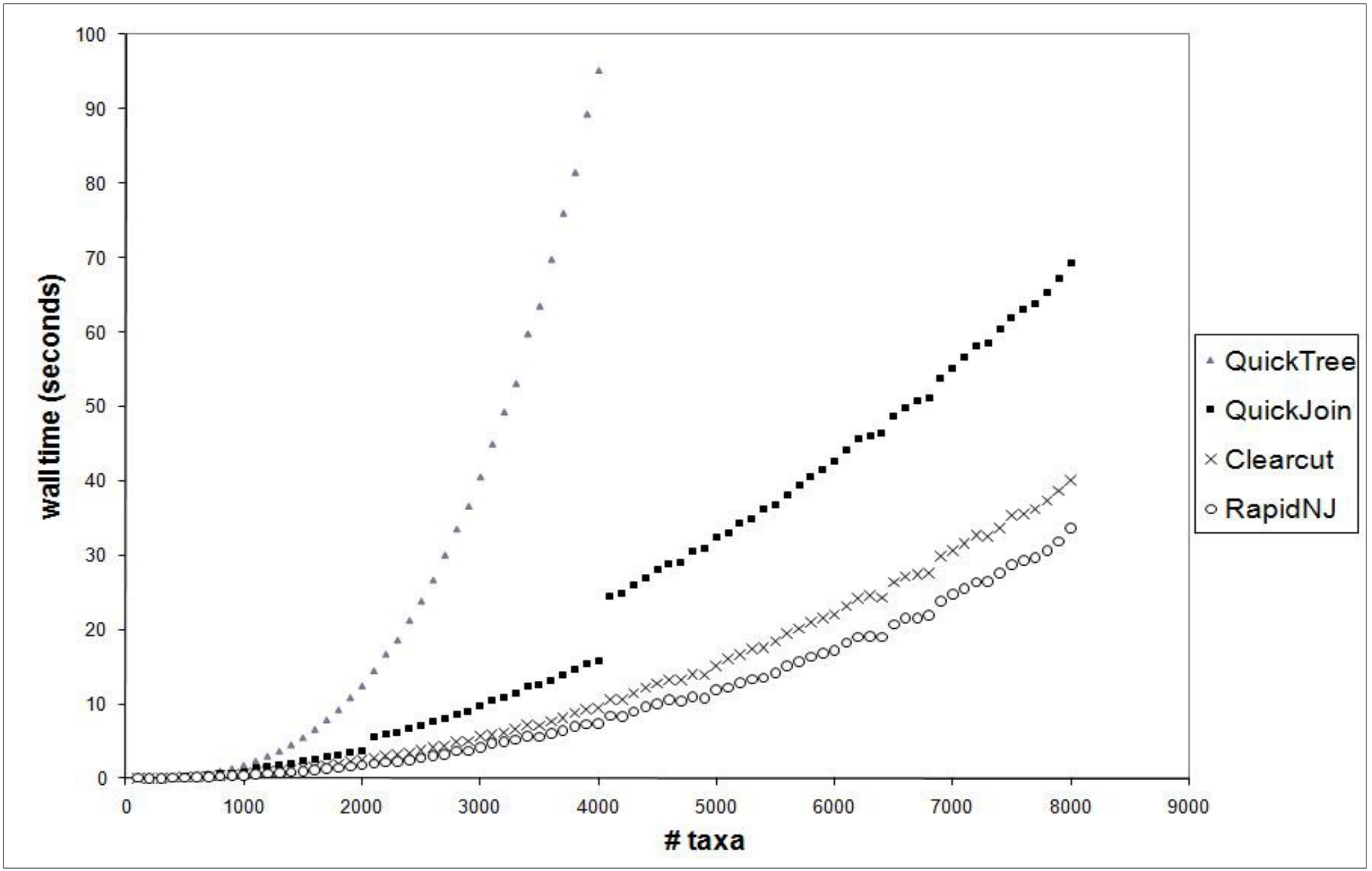
R8s was used to simulate phylogenetic trees from which distance matrices were constructed.

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

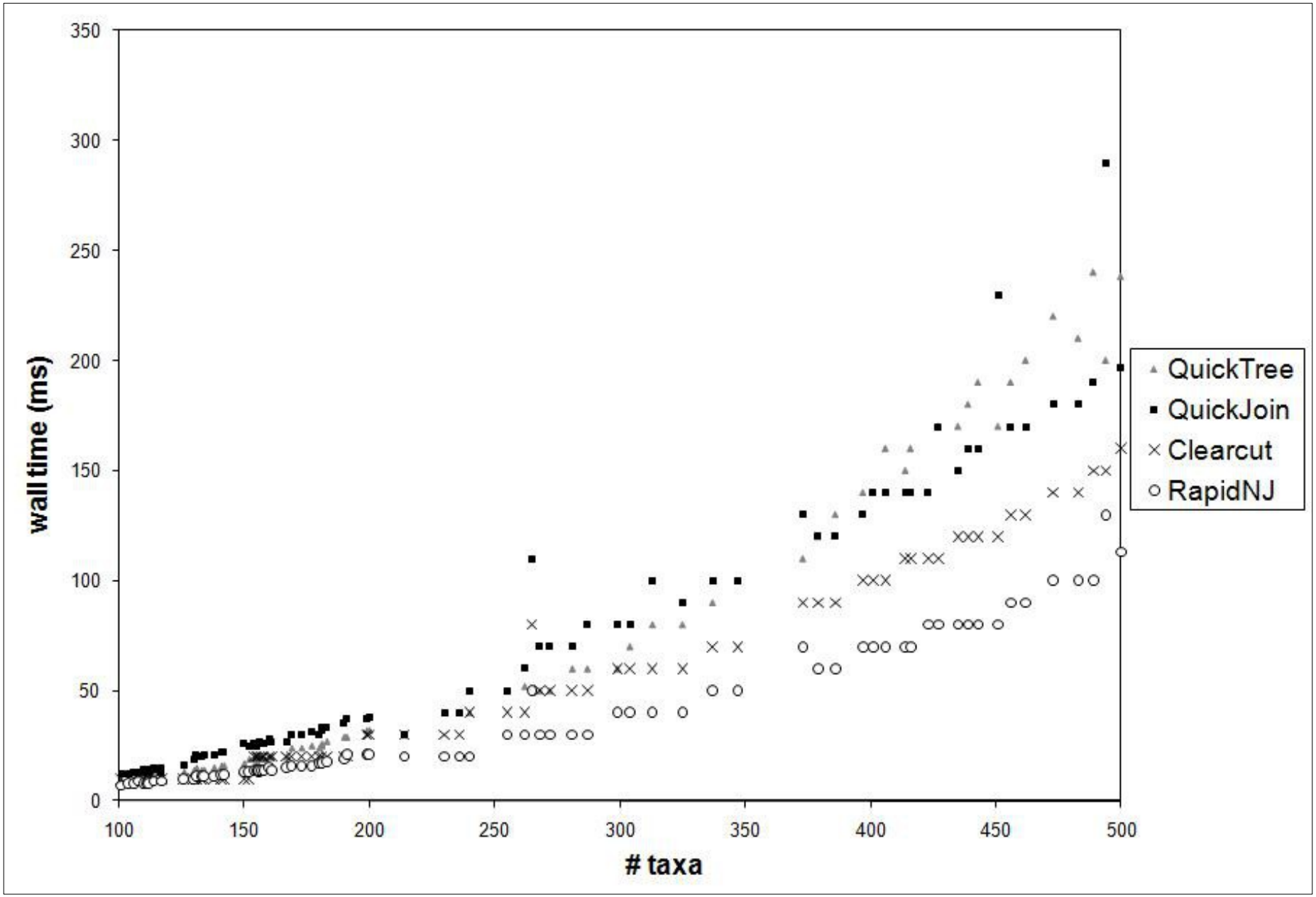
Also work on how to use external memory to allow construction of large trees (> 10.000 leaves).

http://birc.au.dk/Software/RapidNJ/

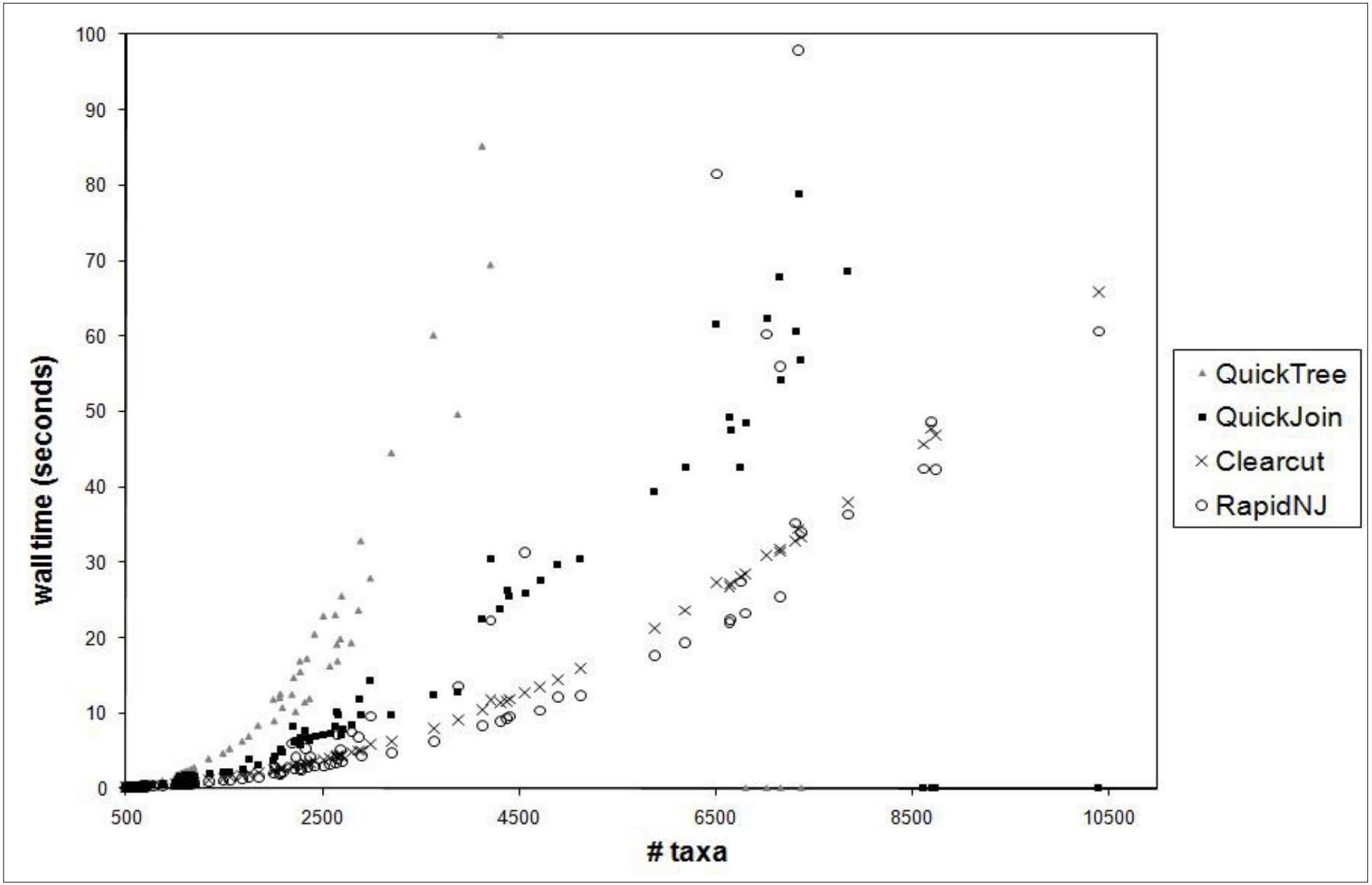
Results on Simulated Data



Results on small Pfam Data



Results on large Pfam Data



Running times on Pfam data