Evolutionary Trees.

- Unrooted / rooted evolutionary trees.
- Binary weighted trees where every internal vertex has degree equal to 3 and every edge has an assigned +ve wt.

Distance-based tree reconstruction:

Problem statement:

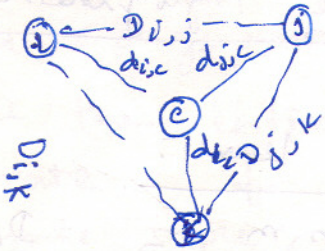
Reconstruct an evolutionary tree from a distance matrix.

Assuming D_{ij} satisfies metric condn

I/P: $\{D_{i,j}\}_{n \times n}$

O/P: A weighted unrooted binary tree T with n leaves fitting D i.e. $d_{i,j}(T) = D_{i,j}$, $1 \le i < j \le n$ if D_{ij} is additive (i.e. if $\exists$ a solution).

- T will have $2n-3$ edges. Hence $2n-2$ vertices, out of which n leaves & $n-2$ internal vertices.
- 3 - ~~vertex~~ leaves solution:



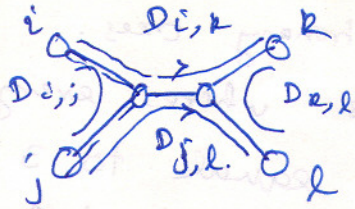
$$\therefore d_{i,c} = \frac{D_{i,j} + D_{i,k} - D_{j,k}}{2}$$

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

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

- For n leaves: no. of variables $(2n-3)$ (i.e. no. of edges) & no. of eqns. available $\binom{n}{2}$. [overconstrained]
  - $\rightarrow \exists$ may not be any solution.
  - $\rightarrow$ If $\exists$ a solution, $(D_{ij})_{n \times n}$ is called as additive matrix.

- Four point condition for checking additive matrix.



Compute 3 sums:

$D_{i,j} + D_{k,l}$

$D_{i,k} + D_{j,l}$ ⇒ Two of these would be same & one will be smaller than the other 2.

$D_{i,l} + D_{j,k}$

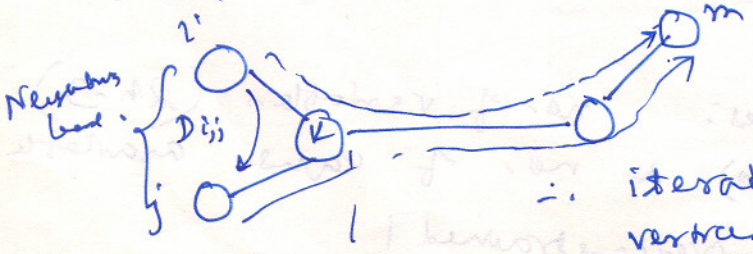
Th.: An $n \times n$ matrix D is additive iff the four point conditions hold for every 4 distinct elements $1 \le i, j, k, l \le n$.

- Least Square Distance based Phylogeny problem:

If the distance matrix is not additive, one might want instead to find a tree that approximates D using the sum of squared errors $\sum\sum (d_{i,j}(T) - D_{i,j})^2$ as a measure of the quality of approximation. This leads to the (NP-hard) least squares Distance based Phylogeny problem.

★ Algorithms for Additive Phylogenetic Tree Reconstruction:

- Collapsing neighboring leaves:

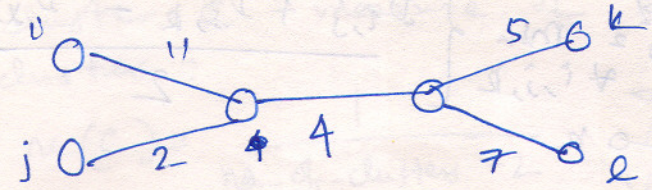


$$D_{k,m} = \frac{D_{i,m} + D_{j,m} - D_{i,j}}{2}.$$

T_n ⇒ Tree of n-vertex.

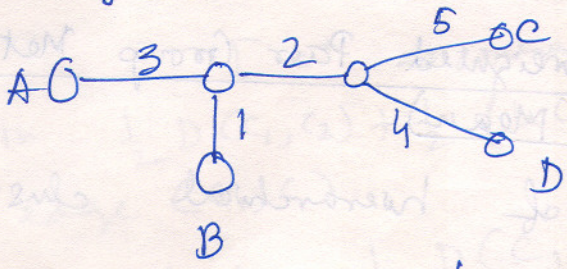
∴ iteratively collapse neighboring vertices & update distances reduce tree T_n to T_{n-1} & continue till arrived at T_1.

→ Problem of finding neighboring vertices, which is non-trivial. e.g.

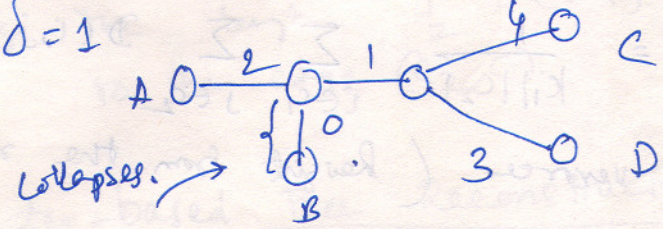


Though $D_{j,k} < D_{i,j}$, j is not neighbour of k but of i.

- Collapsing isolated vertices (from hanging edge) i.e. leaf by iterative pruning of all the edges (to leaves) by the same amount (δ e.g. δ) at any stage.



→ $\delta = 1$



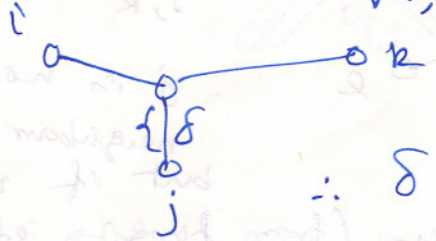
→ A triple of distinct elements $1 \le i, j, k \le n$ is called degenerate, if $D_{i,j} + D_{j,k} = D_{i,k}$.

We call the entire matrix D is degenerate if it has at least one degenerate triple. Convert $[D]_{n \times n}$ to $[D]_{(n-1)\times(n-1)}$ by removing j^{th} column & j^{th} row by making tree of $(n-1)$ vertices.

→ Iteratively collapse every vertex to get intermediate nodes & connections (dist. from the nodes).

→ δ : trimming parameter.

Min. δ ? $\geq \frac{1}{2} \min_{\forall i,j,k} \left\{ \frac{D_{i,j} + D_{j,k} - D_{i,k}}{2} \right\}$ [J, N].



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

★ Evolutionary trees and hierarchical clustering:

• UPGMA (Unweighted Pair Group Method with Arithmatic Mean)

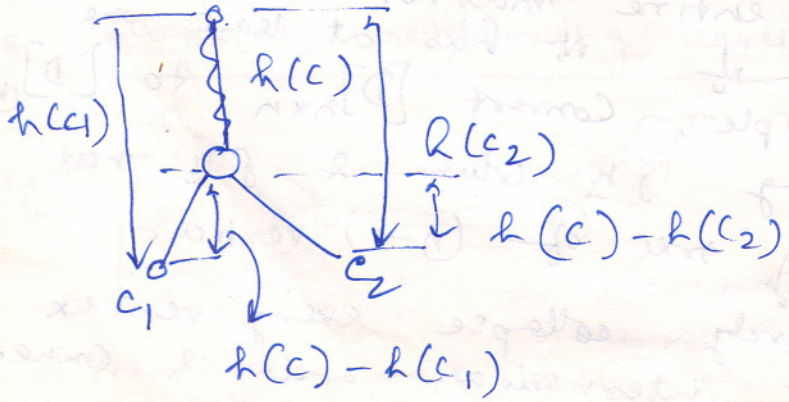
→ A variant of hierarchical clustering

→ $D(C_1, C_2) = \frac{1}{|C_1||C_2|} \sum_{i \in C_1} \sum_{j \in C_2} D(i,j)$

→ "dating" vertices (height from the root).

→ $h(C) = D(C_1, C_2)$.

→ $\begin{cases} h(C) - h(C_1) \text{ to the edge } (C_1, C) \\ h(C) - h(C_2) \text{ to the edge } (C_2, C). \end{cases}$



The 'ultrametric' tree i.e. the distances from root to leaves are same (for all leaves).

- Neighbor joining Alg. (Saitou & Nei (1987)).

→ For a cluster c measure of separation from other clusters?

$$u(c) = \frac{1}{\text{no. of clusters} - 2} \sum_{\forall c'} D(c, c')$$

→ For merging two clusters c_1 & c_2

minimize → $D(c_1, c_2) - u(c_1) - u(c_2)$

→ use same hierarchical clustering.

→ $D(c^*, c) = \frac{D(c_1, c) + D(c_2, c)}{2}$

[c^* → merged cluster from c_1 & c_2.

→ Assign length $\frac{1}{2} D(c_1, c_2) + \frac{1}{2} (u(c_1) - u(c_2))$ to the edge (c_1, c)

→ Assign the length $\frac{1}{2} D(c_1, c_2) + \frac{1}{2} (u(c_2) - u(c_1))$ to the edge (c_2, c).

★ Character-based Tree Reconstruction:

→ character ↔ characteristic represented by 'm' length DNA seq. (or Protein seq.)

→ For n species ~~each~~ each characterized by 'm' ~~'DNA'~~ length DNA seq.

→ Each leaf node represent a species.

→ Internal node ~~should differ~~ characterized by hypothetical 'm' length seq.

→ Minimize $\sum d_H(v_1, v_2)$ ← Parsimony Score

edge $\Rightarrow (v_1, v_2) \in$ Tree.

Small Parsimony Problem:

I/P: Tree T with each leaf labeled by an m-character string.

O/P: Labelling of internal vertices of the tree T minimizing the parsimony score.

Weighted small parsimony problem:

Instead of $d_H(\vec{v_1}, \vec{v_2})$ use $\sum_{i=1}^{m} \delta(x_{i_1}, x_{i_2})$ as a distance measure between two m-length strings., where, x_i's $\in$ Alphabet Set e.g. {A, C, T, G}

I/P: Tree T with each leaf node labeled by elements of k-letter alphabet (of length m strings) and a scoring matrix (δ_{ij}).

O/P: Labelling of internal vertices of the tree Minimizing the parsimony score.

Large Parsimony Problem → NP-complete Problem

Tree is not given. Only, m-character strings for n-species are given & problem is to find 'Tree' with Parsimonious Labelling of vertices'.

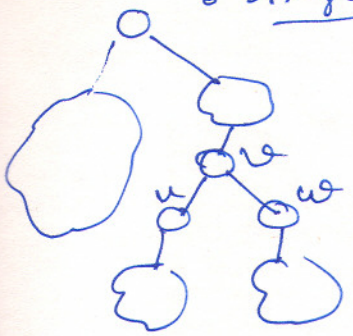
I/P: An $n \times m$ matrix M describing n-species.

O/P: A tree T with n-leaves labeled by n-rows of M & a labelling of internal vertices so that Mut. score is obtained.

- Solutions of Small parsimony Problem

→ wt. Small parsimony tree (Sankoff (1975))

- Single ~~char~~ alphabet string:



Let $S_t(v)$ = min. parsimony score of the subtree under 'v', given label at 'v'.

$\therefore S_t(v) = \min_i \{ (S_i(u) + \delta_{i,t}) \}$ + $\min_j \{ S_j(w) + \delta_{j,t} \}$

- At leaves, $S_t(v) = 0$, if label is 't' else ∞.
- Solve single alphabet assignment at every position of the string independently.
- Reconstruct the vertices by backtracking.
- At every node, computation is $O(k)$.
- Total runtime: $O(nk)$.
  as No. of vertex: $O(n)$.

→ unweighted small parsimony prob. (Fitch (1971)).

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

[From bottom to top]

2. From top to bottom:
   - Root(r) → any label x from S_r.
   - Label(v) → ~~any~~ label from S_v if ~~exists~~ is ~~labelled~~ assigned to its parents
     else any label from S_v.

→ $O(nk)$.