Algorithms in Bioinformatics: A Practical Introduction

Phylogenetic Tree comparison and Consensus Trees

Phylogenetic Tree comparison

Why tree comparison?

- Different phylogenies are resulted using different
  - Kind of data (different segments of the genomes)
  - Kind of model (CF model, Jukes-Cantor Model)
  - Kind of reconstruction algorithm

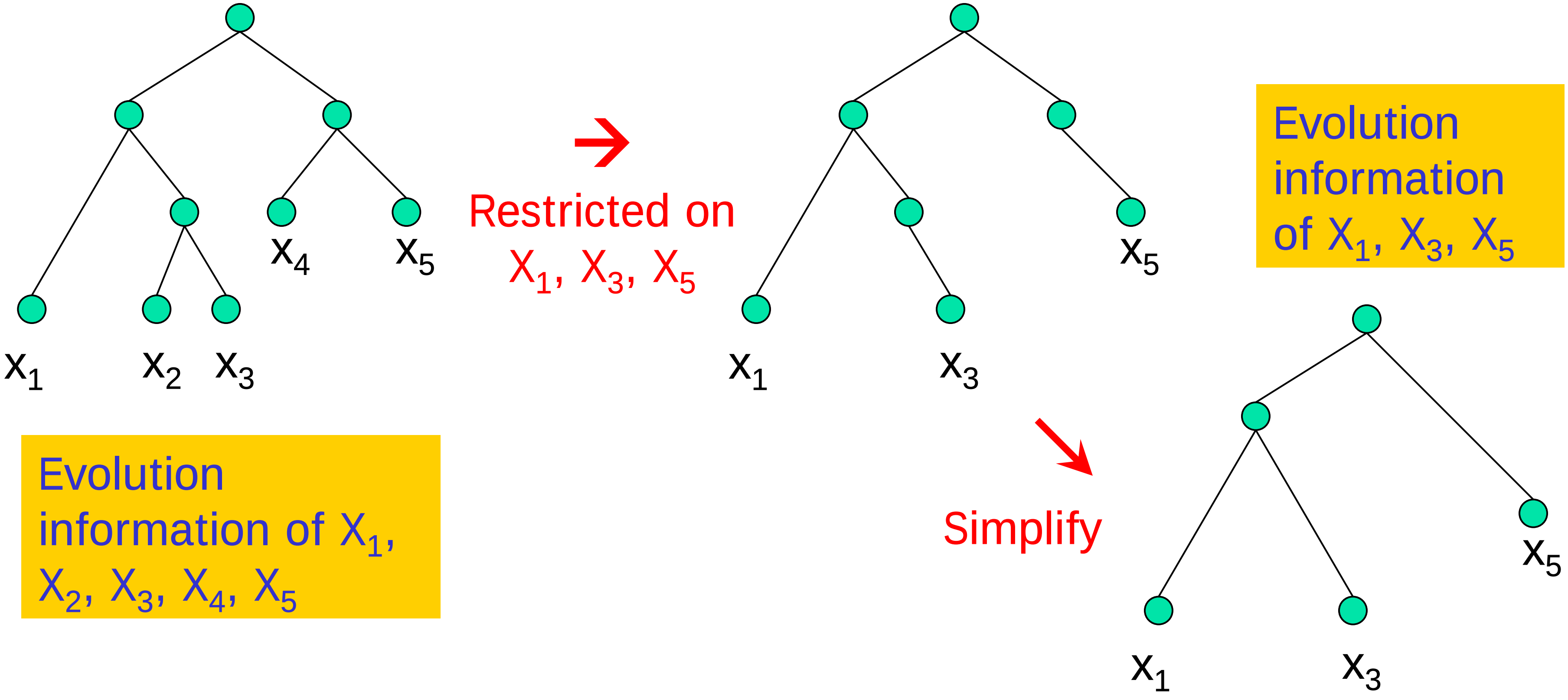
- Tree comparison helps us to gain information from multiple trees.

Two types of comparsions

- Similarity measurement
  - Find the common structure among the given trees
    - Maximum Agreement Subtree
- Dissimilarity measurement
  - Determine the differences among the given trees
    - Robinson-Foulds distance
    - Nearest neighbor interchange
    - Subtree Transfer Distance
    - Quartet Distance

Restricted subtree

- Consider a trees T



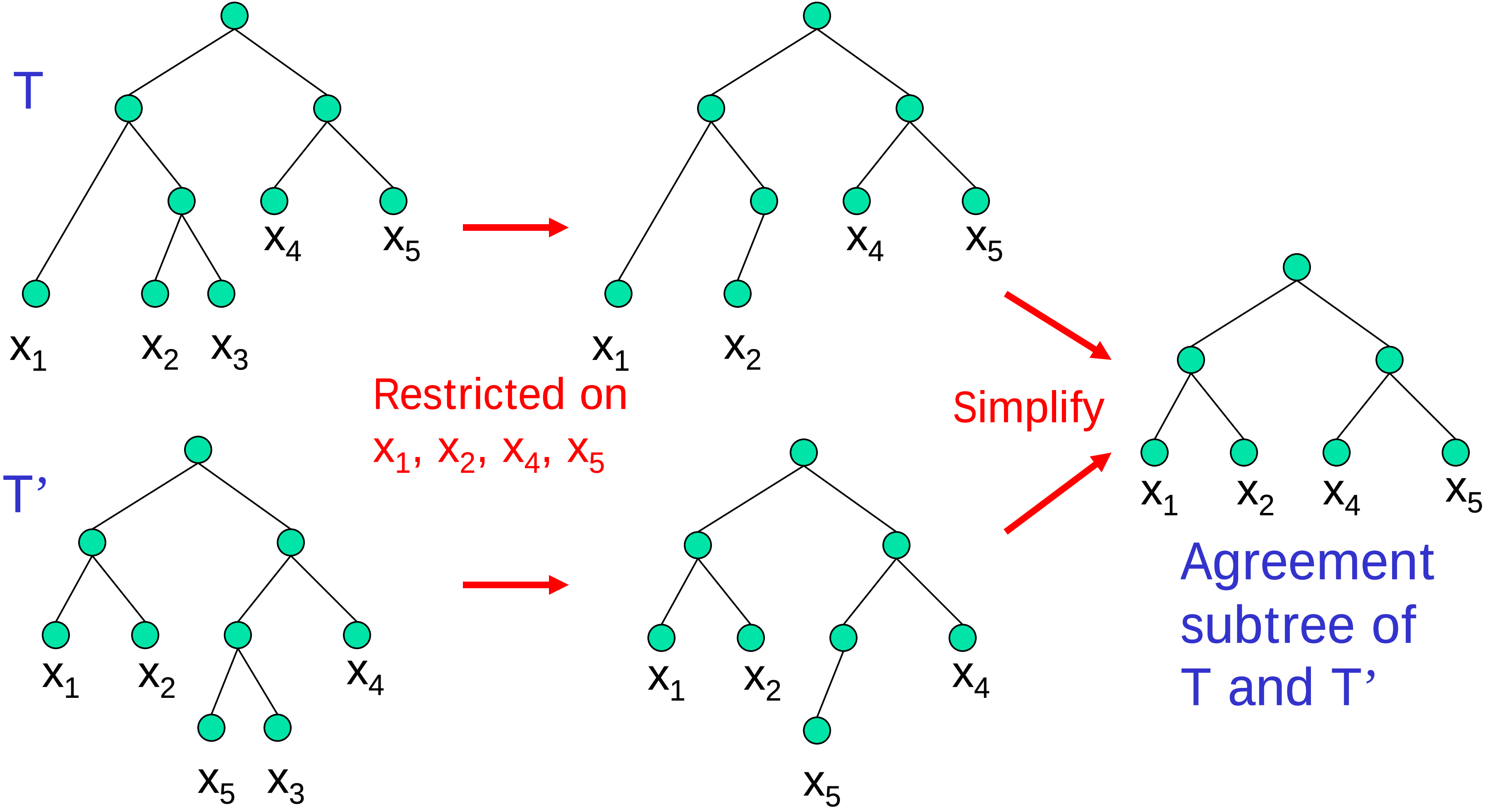
Restricted on X_1, X_3, X_5

Evolution information of X_1, X_3, X_5

Simplify

Evolution information of X_1, X_2, X_3, X_4, X_5

Agreement subtree



Restricted on x_1, x_2, x_4, x_5

Simplify

Agreement subtree of T and T’

Maximum agreement subtree (MAST)

- Given two trees T_1 and T_2
- Agreement subtree of T_1 and T_2 is the common information agreed by both trees.
  - Since it is agreed by both trees, the evolution of the agreement subtree is more reliable!
- Maximum agreement subtree problem
  - Find the agreement subtree with the largest possible number of leaves.
  - Such agreement subtree is called the maximum agreement subtree

MAST for rooted trees

- MAST of two degree-d rooted trees T_1 and T_2 with n leaves can be computed in
    - $O(\sqrt{d}\,n \log(\frac{n}{d}))$ time (Journal of Algorithm 2001)

- This lecture considers an $O(n^2)$-time algorithm which compute the maximum agreement subtree of two binary trees with n leaves.

Computing MAST by dynamic programming

- For any two binary rooted trees T_1 and T_2, denote MAST(T_1, T_2) be the number of leaves in the maximum agreement subtree
- Some definition:
  - For a tree T and a node u, T^u is the subtree of T rooted at u

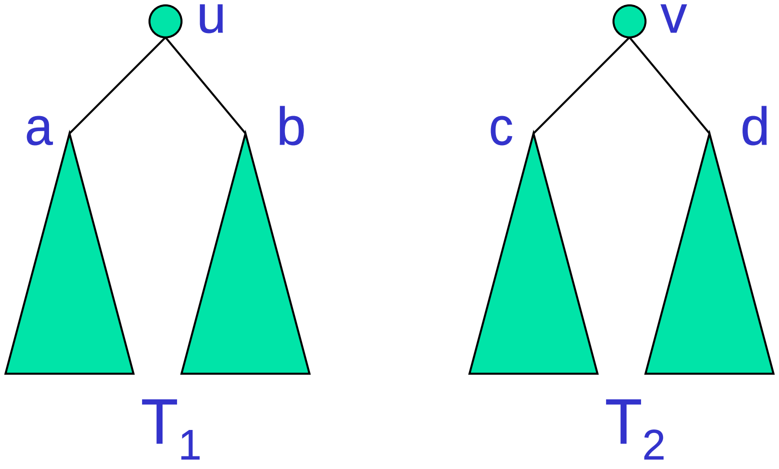
Not complete!

- For any node pair $(u,v) \in T_1 \times T_2$,
  - let a and b be two children of u
  - let c and d be two children of v
- Let R be the maximum agreement subtree of T_1 and T_2.
- We have the following cases:
  - R is an agreement subtree of T_1^a
  - R is an agreement subtree of T_1^b

Recurrence

$$MAST(T_1^u, T_2^v) =$$

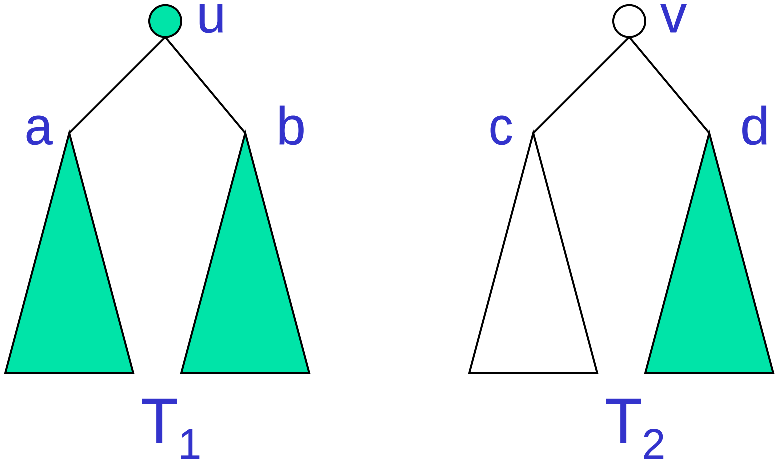
$$\max\begin{cases} MAST(T_1^a, T_2^c) + MAST(T_1^b, T_2^d) \\ MAST(T_1^a, T_2^d) + MAST(T_1^b, T_2^c) \\ MAST(T_1^a, T_2^v) \\ MAST(T_1^b, T_2^v) \\ MAST(T_1^u, T_2^c) \\ MAST(T_1^u, T_2^d) \end{cases}$$



Recurrence (II)

$$MAST(T_1^{\,u}, T_2^{\,v}) =$$

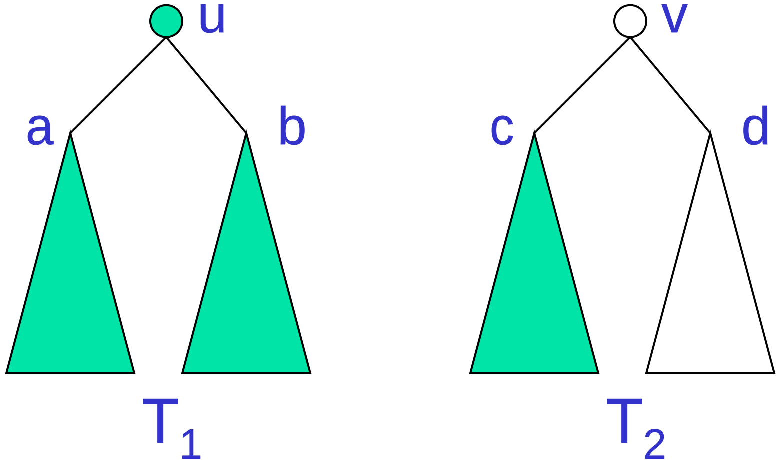
$$\max\begin{cases} MAST(T_1^{\,a}, T_2^{\,c}) + MAST(T_1^{\,b}, T_2^{\,d}) \\ MAST(T_1^{\,a}, T_2^{\,d}) + MAST(T_1^{\,b}, T_2^{\,c}) \\ MAST(T_1^{\,a}, T_2^{\,v}) \\ MAST(T_1^{\,b}, T_2^{\,v}) \\ MAST(T_1^{\,u}, T_2^{\,c}) \\ MAST(T_1^{\,u}, T_2^{\,d}) \quad \leftarrow \end{cases}$$



Recurrence (III)

$$MAST(T_1^u, T_2^v) =$$

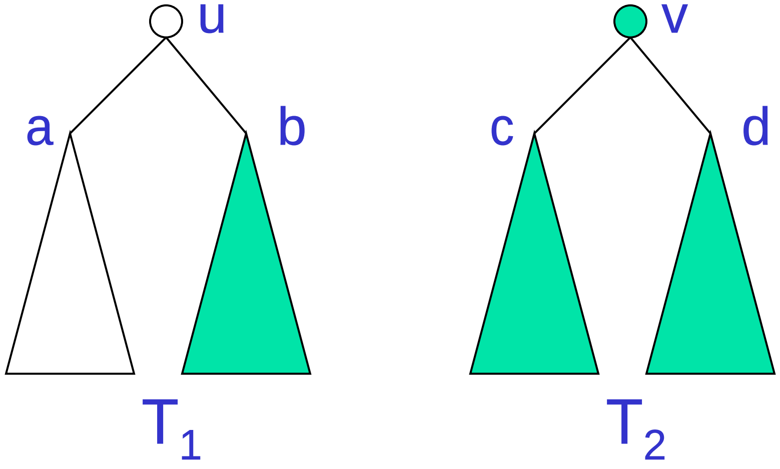
$$\max\begin{cases} MAST(T_1^a, T_2^c) + MAST(T_1^b, T_2^d) \\ MAST(T_1^a, T_2^d) + MAST(T_1^b, T_2^c) \\ MAST(T_1^a, T_2^v) \\ MAST(T_1^b, T_2^v) \\ MAST(T_1^u, T_2^c) \quad \leftarrow \\ MAST(T_1^u, T_2^d) \end{cases}$$



Recurrence (IV)

$$MAST(T_1^u, T_2^v) =$$

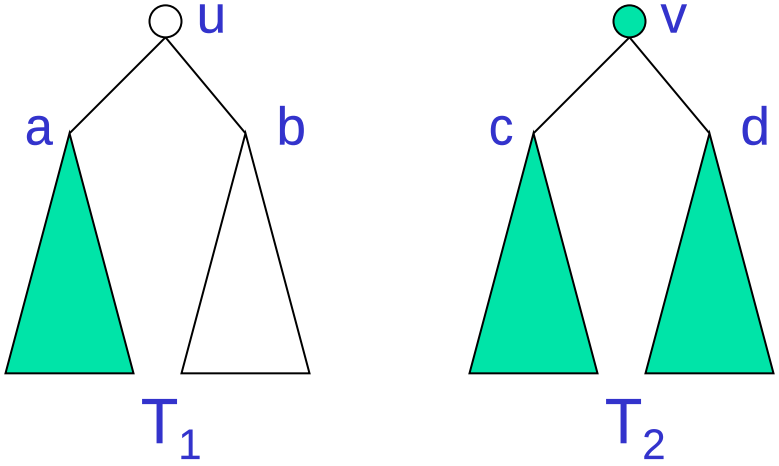
$$\max \begin{cases} MAST(T_1^a, T_2^c) + MAST(T_1^b, T_2^d) \\ MAST(T_1^a, T_2^d) + MAST(T_1^b, T_2^c) \\ MAST(T_1^a, T_2^v) \\ MAST(T_1^b, T_2^v) \quad \leftarrow \\ MAST(T_1^u, T_2^c) \\ MAST(T_1^u, T_2^d) \end{cases}$$



Recurrence (V)

$$MAST(T_1^{u}, T_2^{v}) =$$

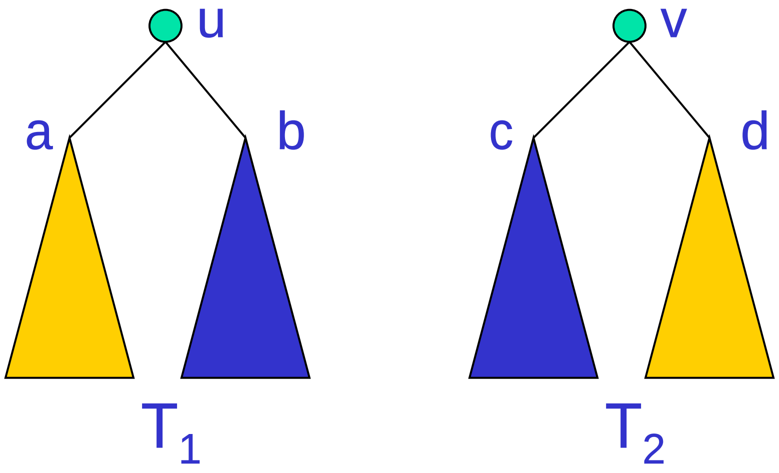
$$\max\begin{cases} MAST(T_1^{a}, T_2^{c}) + MAST(T_1^{b}, T_2^{d}) \\ MAST(T_1^{a}, T_2^{d}) + MAST(T_1^{b}, T_2^{c}) \\ MAST(T_1^{a}, T_2^{v}) \quad \leftarrow \\ MAST(T_1^{b}, T_2^{v}) \\ MAST(T_1^{u}, T_2^{c}) \\ MAST(T_1^{u}, T_2^{d}) \end{cases}$$



Recurrence (VI)

$$MAST(T_1^{\,u}, T_2^{\,v}) =$$

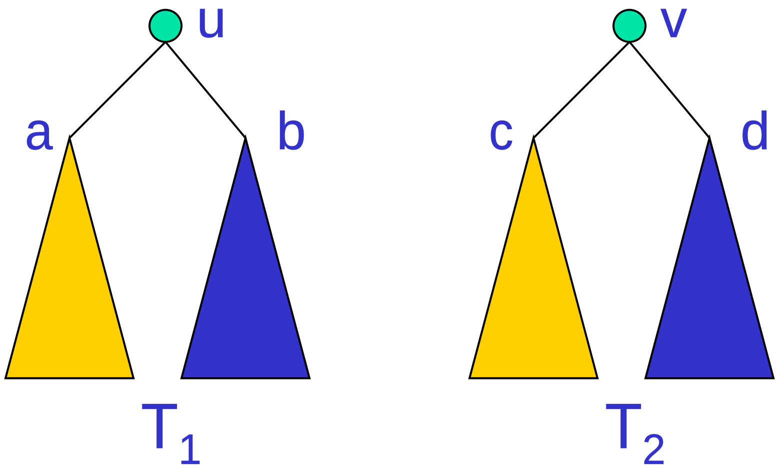
$$\max\begin{cases} MAST(T_1^{\,a}, T_2^{\,c}) + MAST(T_1^{\,b}, T_2^{\,d}) \\ MAST(T_1^{\,a}, T_2^{\,d}) + MAST(T_1^{\,b}, T_2^{\,c}) \quad \leftarrow \\ MAST(T_1^{\,a}, T_2^{\,v}) \\ MAST(T_1^{\,b}, T_2^{\,v}) \\ MAST(T_1^{\,u}, T_2^{\,c}) \\ MAST(T_1^{\,u}, T_2^{\,d}) \end{cases}$$



Recurrence (VII)

$$MAST(T_1^u, T_2^v) =$$

$$\max\begin{cases} MAST(T_1^a, T_2^c) + MAST(T_1^b, T_2^d) & \leftarrow \\ MAST(T_1^a, T_2^d) + MAST(T_1^b, T_2^c) \\ MAST(T_1^a, T_2^v) \\ MAST(T_1^b, T_2^v) \\ MAST(T_1^u, T_2^c) \\ MAST(T_1^u, T_2^d) \end{cases}$$



Time complexity

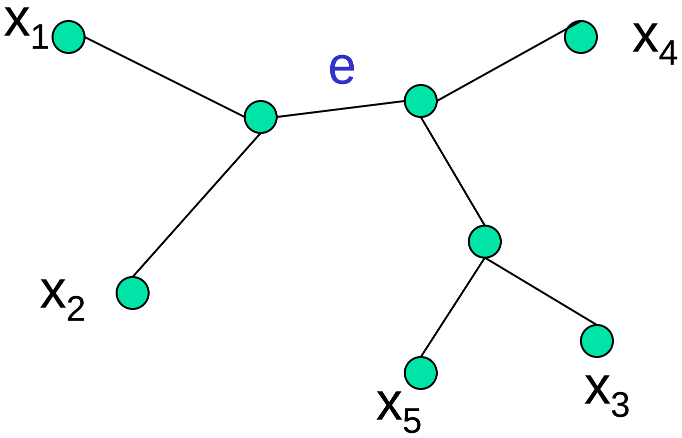
- Suppose T_1 and T_2 are rooted phylogenies for n species.
- We have to compute MAST(T_1^u, T_2^v) for every u in T_1 and v in T_2.
- Thus, we need to fill in n^2 entries. Each entry can be computed in O(1) time.
- In total, the time complexity is $O(n^2)$.

MAST for unrooted trees

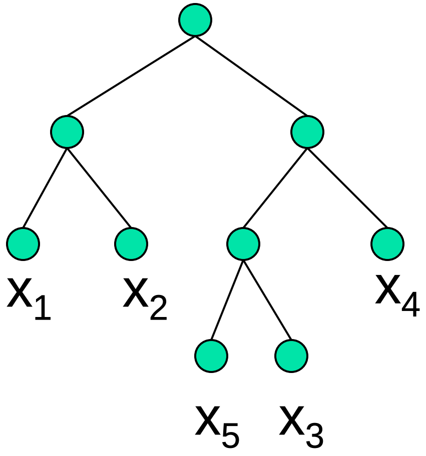
- In real life, we normally want to compute MAST for unrooted trees.
- For unrooted degree-3 trees U_1 and U_2, MAST(U_1, U_2) can be computed in O(n log n) time. (STOC 97)
- For general unrooted trees U_1 and U_2, MAST(U_1, U_2) can be computed in O($n^{1.5}$ log n) time. (SIAM J. of Comp 2000)
- This lecture shows the relationship between unrooted MAST and rooted MAST!

Relating rooted and unrooted trees (I)

- Definition:
  - For an unrooted tree U, for any edge e in U, U^e is the rooted tree rooted at the edge e.



→ rooted at edge e



Relating rooted and unrooted trees (II)

- Consider two unrooted trees U_1 and U_2
- Lemma: For any edge e of U_1,

$$MAST(U_1, U_2) = \max\{MAST(U_1^{e}, U_2^{f}) \mid f \text{ is an edge of } U_2\}$$

- Proof: Exercise!

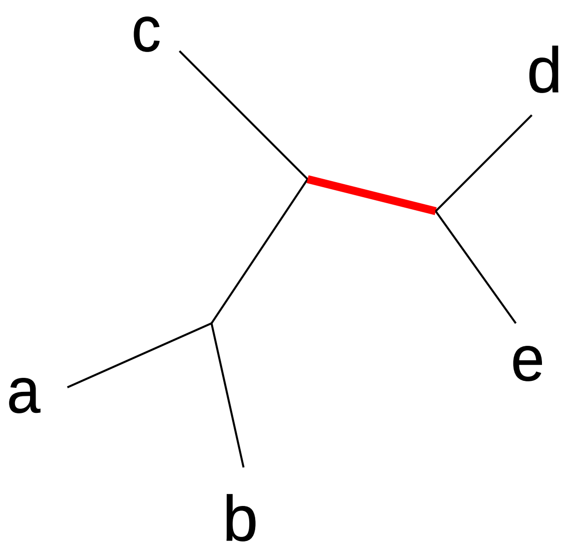
- Based on the above lemma, we can relate rooted MAST and unrooted MAST!

Robinson-Foulds distance

- Given two phylogenies T_1 and T_2,
- Intuitively, this method tries to count the number of edges which are not agreed by T_1 and T_2.
- First, we need to have some definitions!

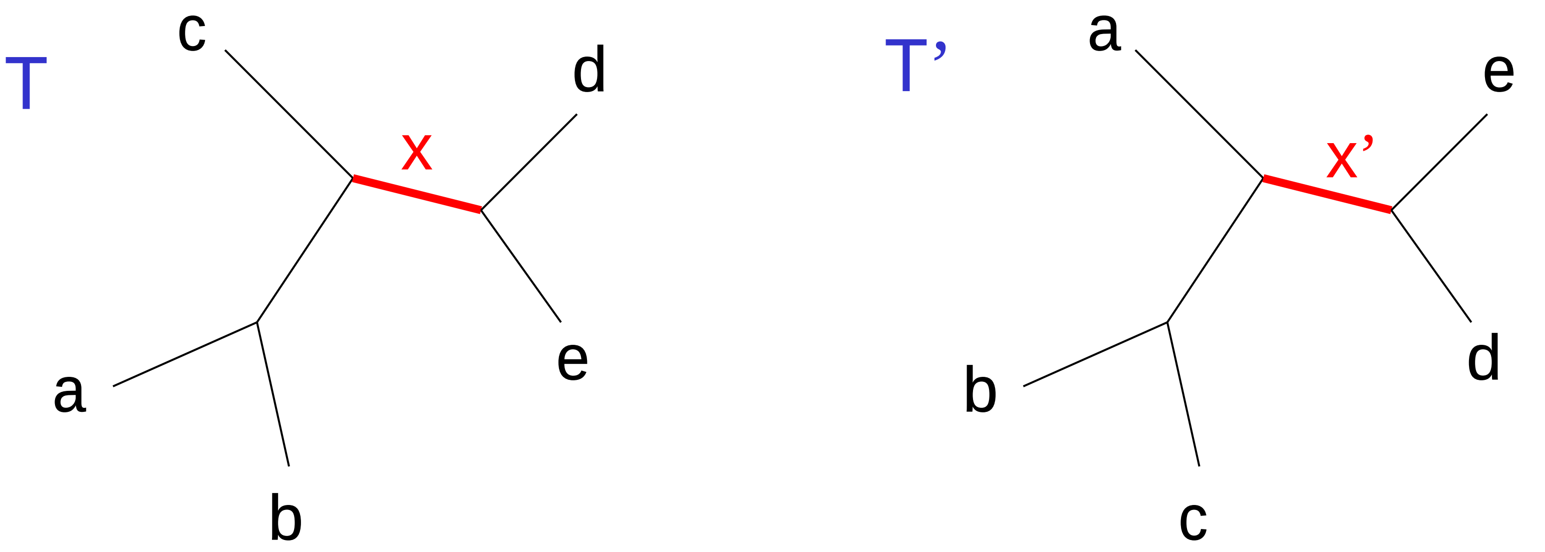
Partitioning of a tree

- Each edge can partition the set of species
- In the following tree, the red edge partition the species into {a, b, c} and {d, e}

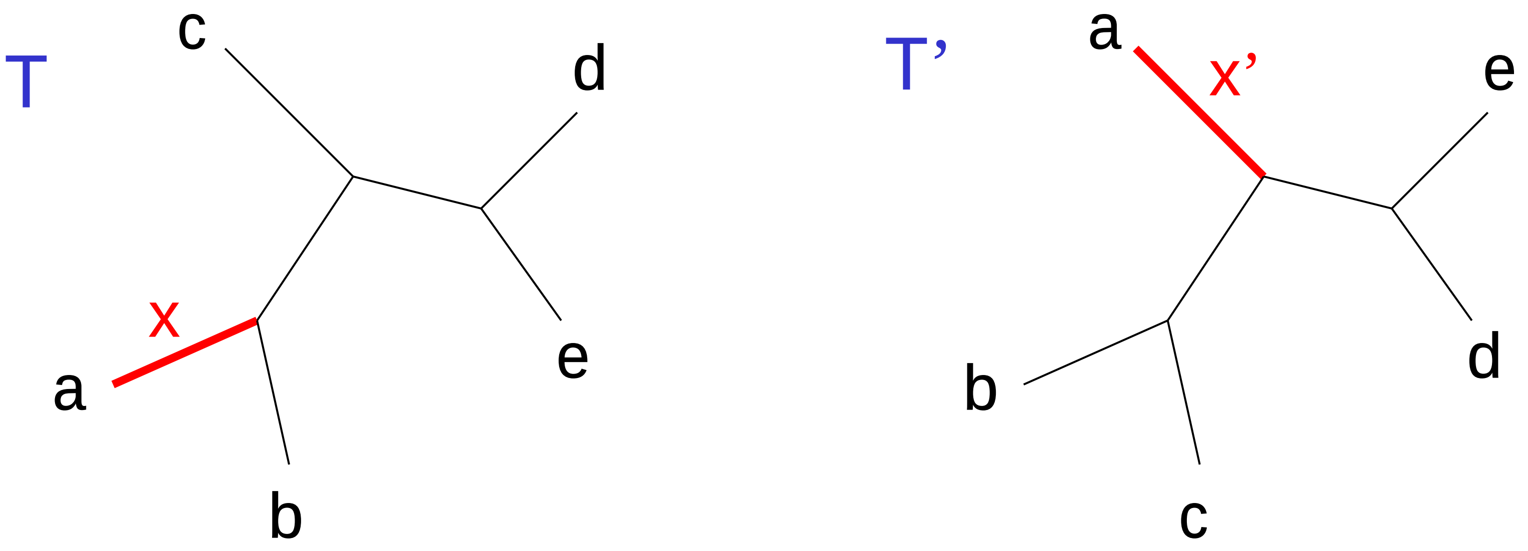


Good and bad edges

- Consider two unrooted trees T and T’, an edge x in T is called a good edge if there exists an edge x’ in T’ such that both of them form the same partitions! Similarly, x’ is also called a good edge.
- Otherwise, the edge is called a bad edge!

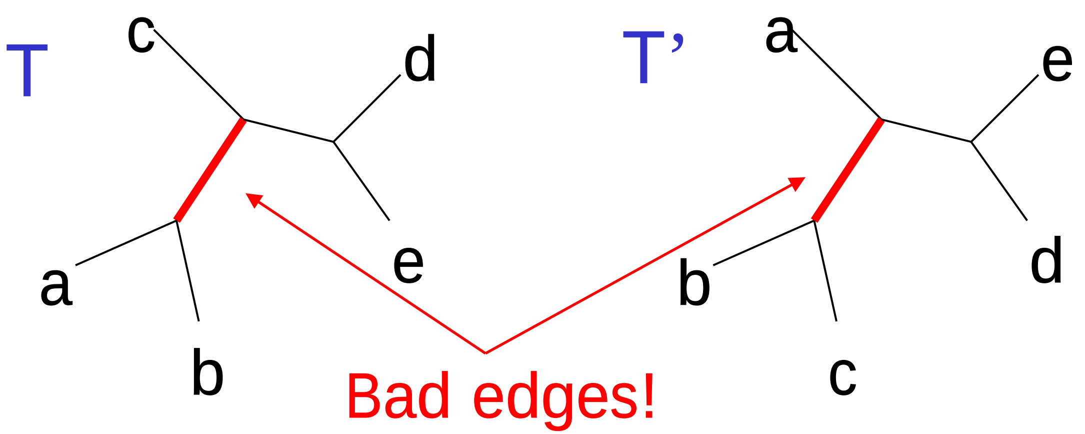


Leaf edges are always good



Robinson-Foulds (RF) distance

- Robinson-Foulds distance =
(number of bad edges in T w.r.t T' + number of bad edges in T' w.r.t. T)/2
- T and T' looks similar if RF-dist(T, T') is small.
- For example, the robinson-foulds distance of T and T' = (1+1)/2 = 1.



Degree-3 trees T and T’

- When both T and T’ are of degree-3, number of bad edges in T w.r.t. T’ = number of bad edges in T’ w.r.t. T
- Proof:
  - Since both T and T’ are of degree-3, T and T’ have the same number of edges
  - Number of good edges in T w.r.t. T’ = number of good edges in T’ w.r.t. T
  - Lemma follows.

How to find the set of good edges in T w.r.t. T'?

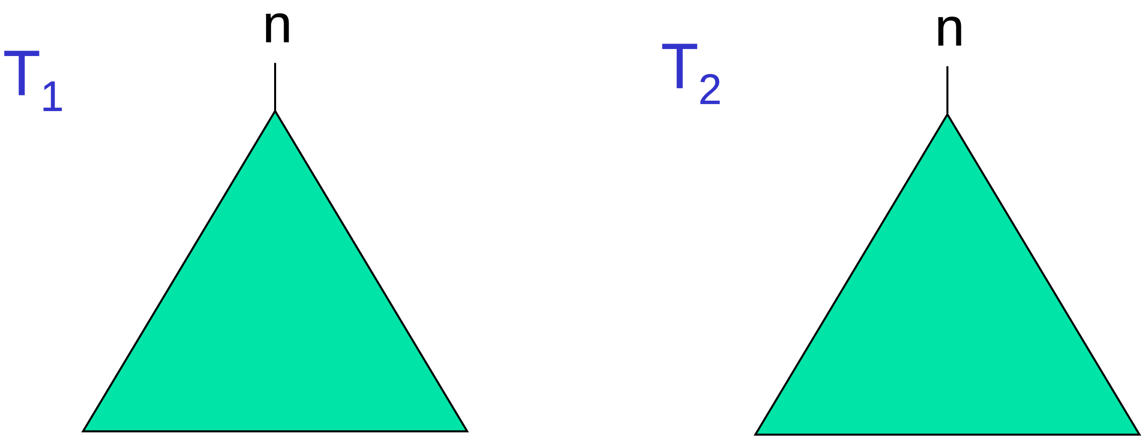
- Brute-force algorithm:
  - For every edge e in T,
    - If the partition formed by e is the same as the partition formed by some edge e' in T', e is a good edge!
- Time analysis:
  - For every edge e in T, the checking takes O(n) time.
  - In total, the time complexity is O(n^2)!
  - Can we do better?

Day’s algorithm

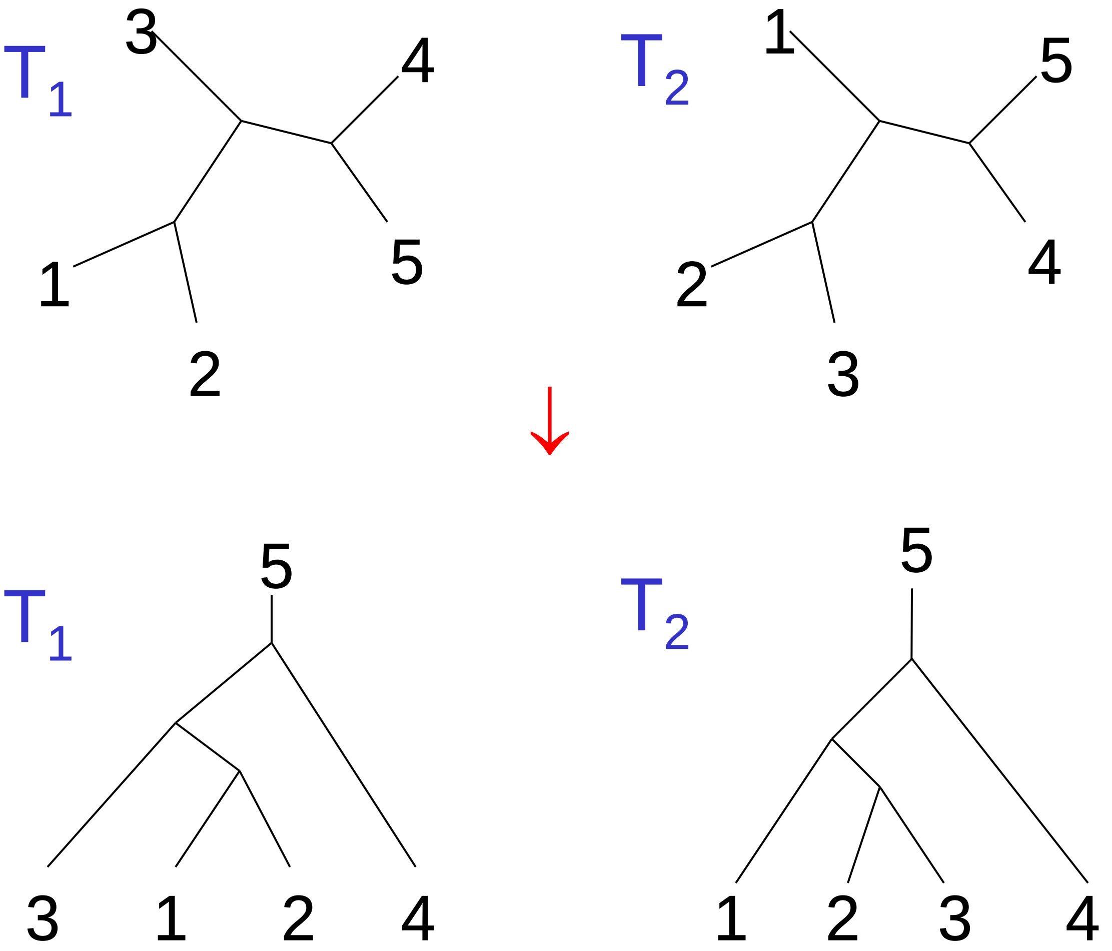
- Yes! The problem can be solved in O(n) time based on Day’s algorithm.
- Input: two unrooted phylogenies T_1 and T_2 for the same set of species
- Output: the set of good edges in T_1 w.r.t. T_2
- Idea:
  - Build data-structure which enables constant time checking whether a particular partition of leaves exists in T_1.

Step 1

- Root T_1 and T_2 at the leaves with label n.
- This step takes O(n) time.

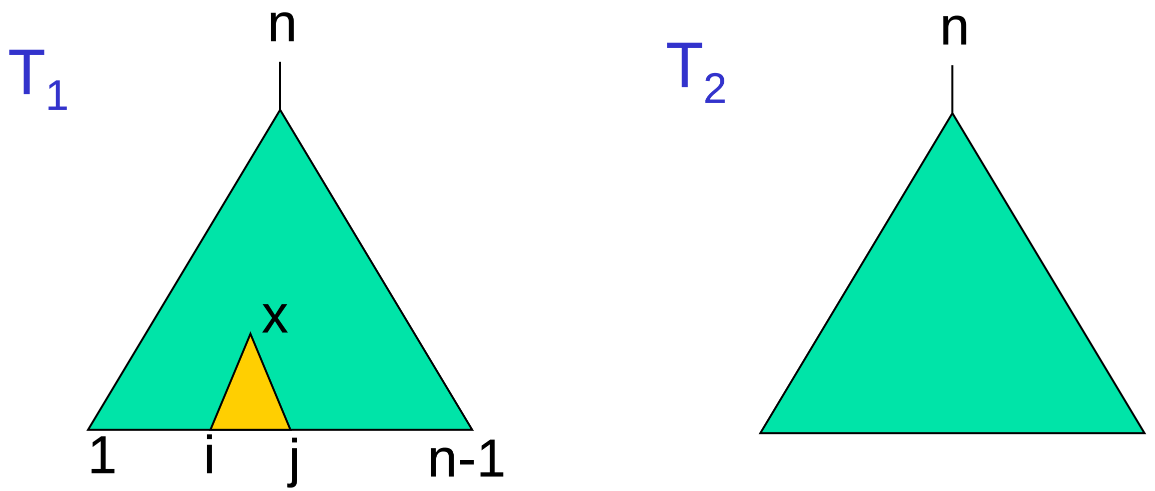


Example for step 1

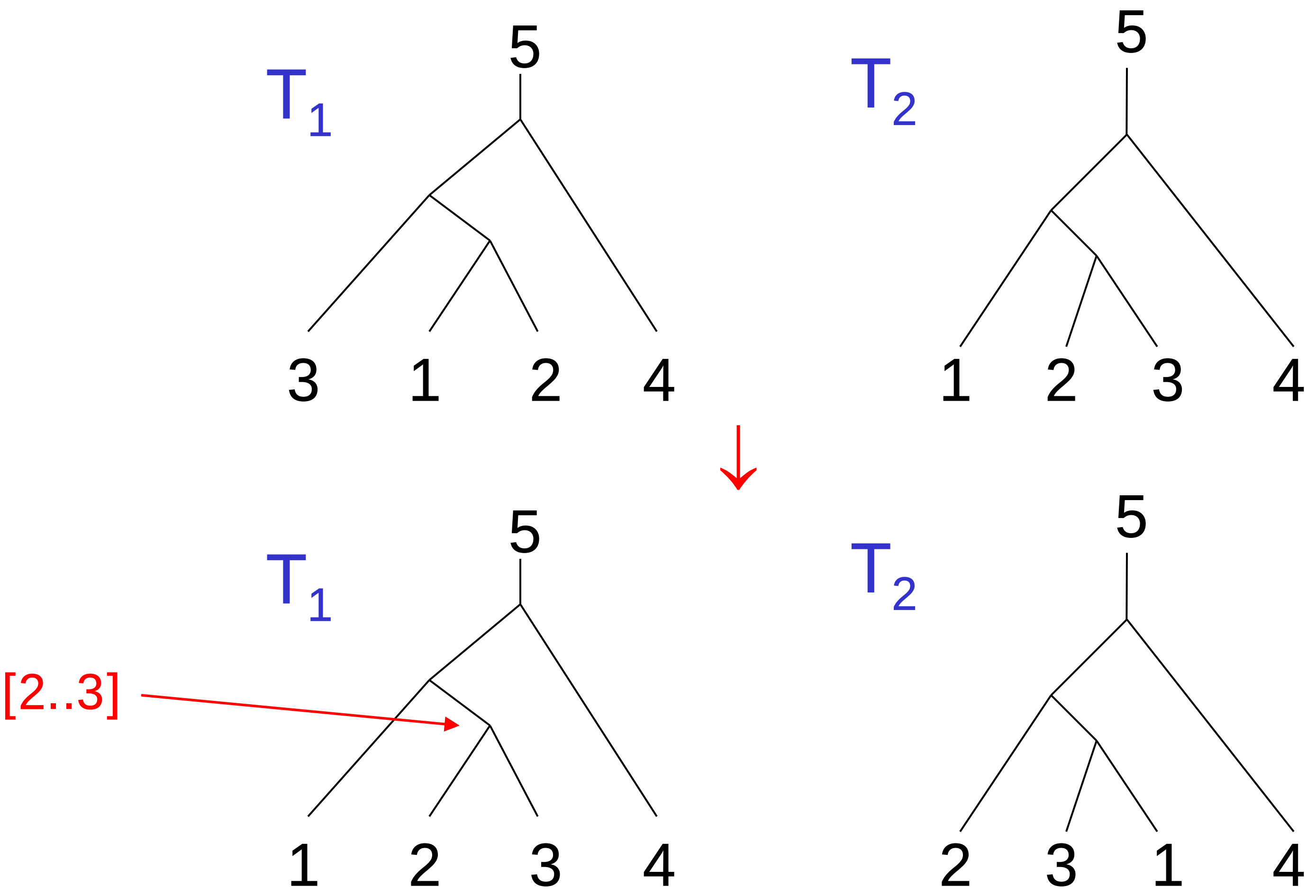


Step 2

- Relabel the leaves of T_1 in increasing order.
- Note: for every internal node x of T_1, the set of leaf labels in the subtree of x form an interval [i..j].
- This step takes O(n) time.



Example for step 2

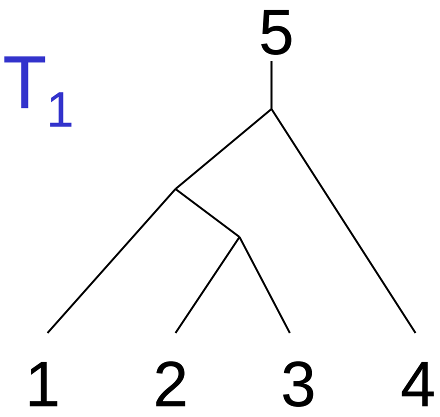


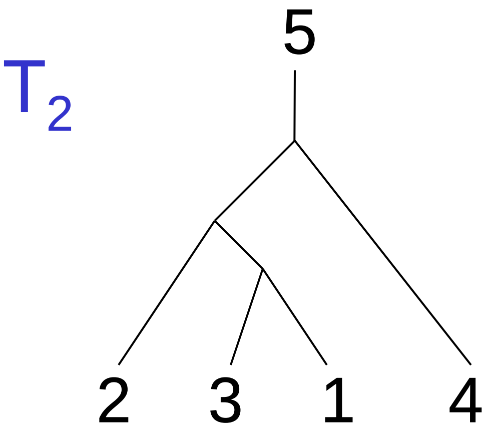
Step 3

- Create a hash table H[1..n]
- For every node x in T_1, we store the corresponding interval $[i_x..j_x]$ in either $H[i_x]$ or $H[j_x]$
  - Store $[i_x..j_x]$ in $H[j_x]$ if x is the leftmost child of its parent in T_1;
  - Otherwise, store the interval $[i_x..j_x]$ in the entry $H[i_x]$.
- This step takes O(n) time.
- Question: Will we store two intervals in the same entry in H?

Example for step 3

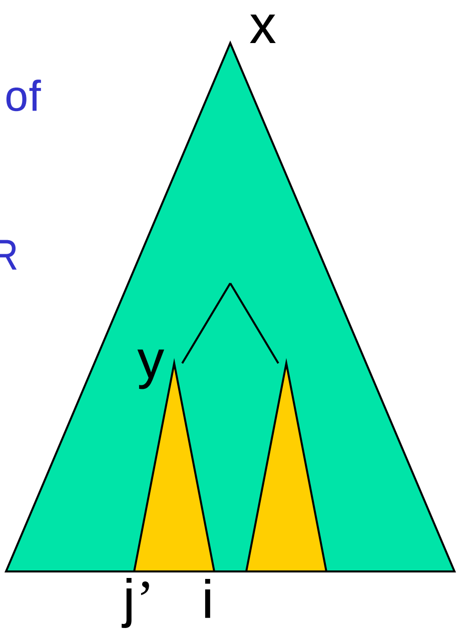
k	H(k)
1	
2	[2..3]
3	[1..3]
4	[1..4]





Observation

- Lemma: we store at most one interval in each entry in H.
- Proof:
  - By contrary, suppose H[i] contain two intervals which are represented by internal nodes x and y.
  - By definition, i should be the endpoints of the intervals represented by x and y. Thus, x and y should satisfy the ancestor-descendent relationship. WLOG, assume x is the ancestor of y. Then, y’s interval should be the subinterval of x’s interval
  - So, we can have either
    1. x’s interval = [j..i] and y’s interval = [j’..i] for j<j’; OR
       - This means that both x and y are the leftmost children of their parents.
       - The right endpoint of x’s interval should not be i!
       - Contradiction!
    2. x’s interval = [i..j] and y’s interval = [i..j’] for j>j’
       - Similar to the above case, we can arrive at contradiction!



More on step 3

- Given the hash table H, we can check whether an interval [i..j] exists in T_1 by checking if H[i] or H[j] equals [i..j]!

Step 4

- For T_2, by traversing the tree, for each internal node u, we compute
  - the minimum (min_u) and the maximum (max_u) leaf labels
  - the number of leaves ($size_u$)

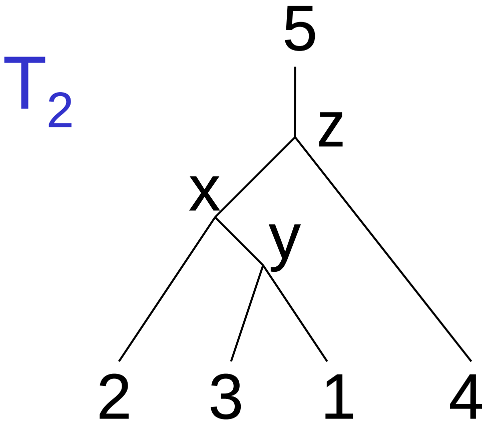
  in the subtree rooted at u
- If (max_u-min_u+1=$size_u$), then
  - the leaves labels in the subtree of node u form an interval [min_u..max_u].
  - Check whether H[min_u] or H[max_u] equals [min_u..max_u]. If yes, (u,v) is a good edge where v is the parent of u in T_2.

- This step takes O(n) time.

Example for step 4

	min_u	max_u	$size_u$	max_u-min_u+1
x	1	3	3	3
y	1	3	2	3

Note: $size_x$=max_x-min_x+1
Also, H[3]=[1..3]
Thus, (x, z) is a good edge!

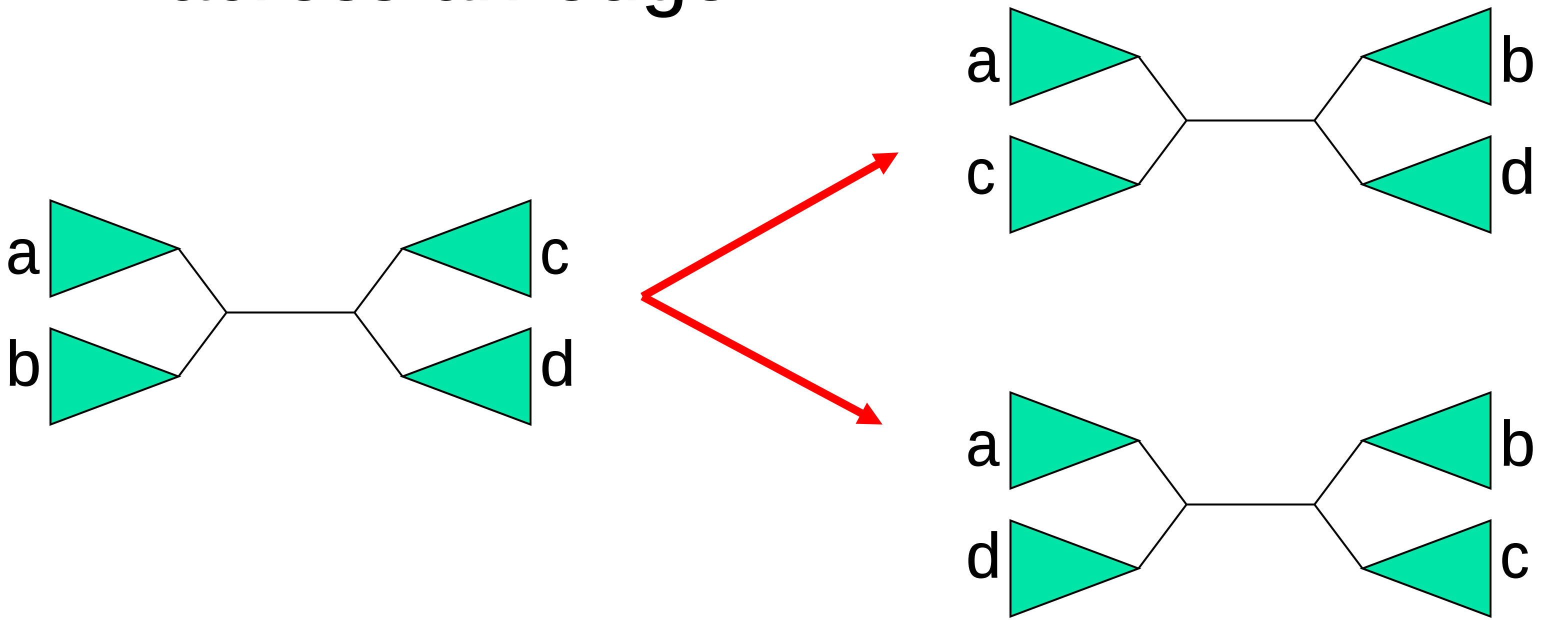


Time complexity

- All 4 steps can correctly recover the good edges.
- They can be computed in O(n) time.
- Thus, the total time complexity is O(n).

Nearest Neighbor Interchange (NNI)

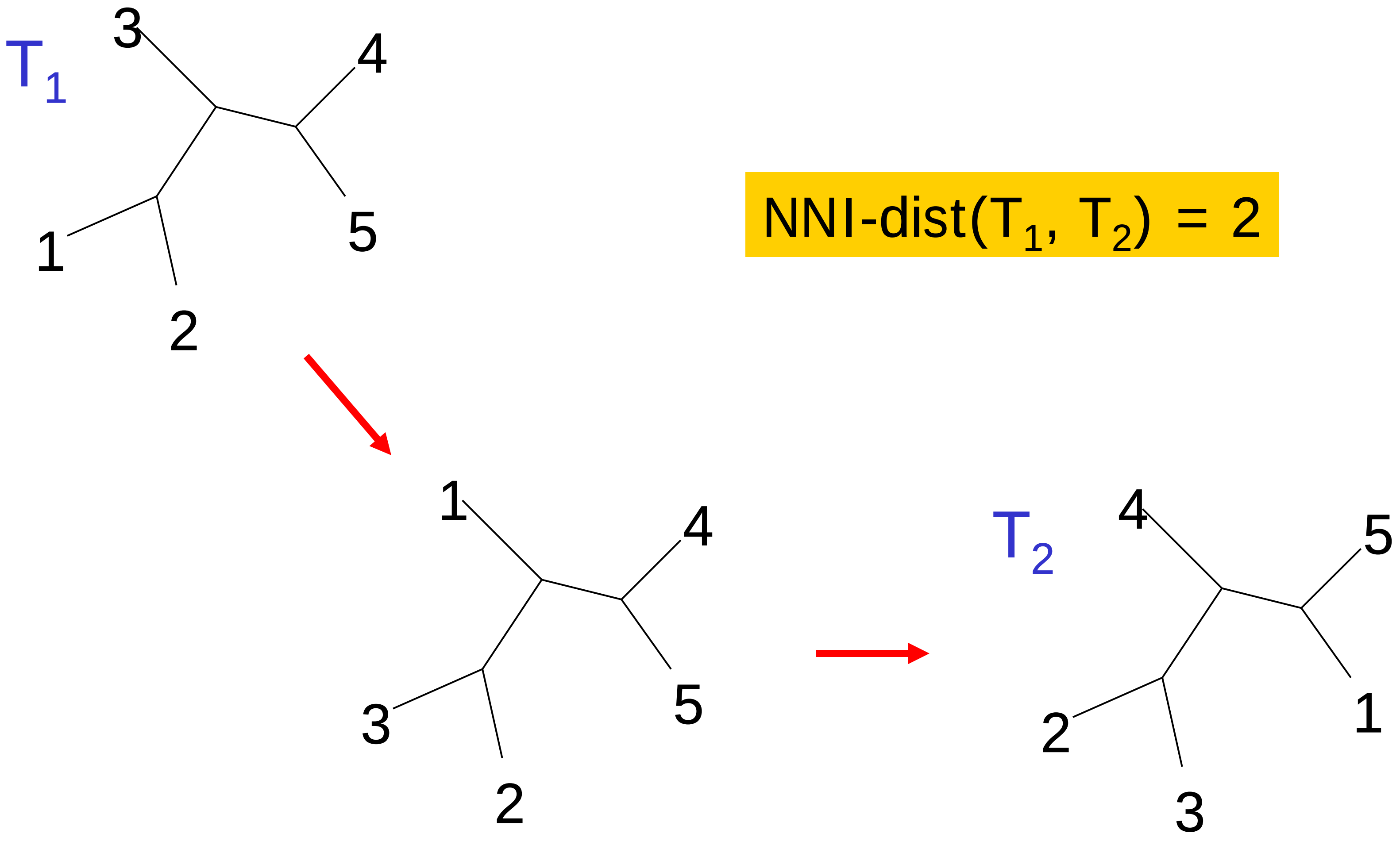
- Given an unrooted, degree-3 tree T,
- NNI operation exchanges two subtrees across an edge.



NNI-dist

- Given two unrooted, degree-3 trees T_1 and T_2,
- NNI-dist(T_1, T_2) is the minimum number of NNI-operations required to convert T_1 to T_2.
- T_1 and T_2 looks similar if NNI-dist(T_1, T_2) is small.
- Computing NNI-dist is NP-hard.

Example



NNI-dist(T_1, T_2) = 2

Properties of NNI-dist

- Property 1:

  $$\text{NNI-dist}(T_1, T_2) = \text{NNI-dist}(T_2, T_1)$$

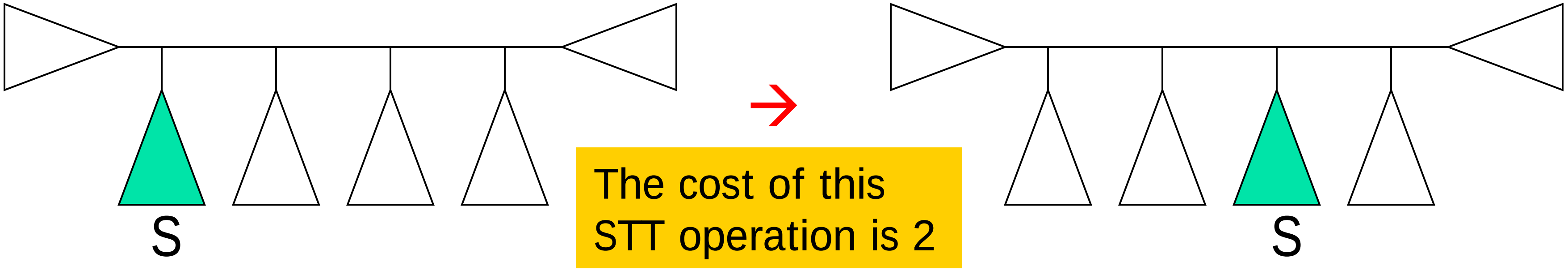
- Property 2: NNI-dist(T_1, T_2)≥number of bad edges in T_1 w.r.t. T_2.
- Proof:
  - To remove one bad edge, we require at least one NNI-operation

Approximation algorithm for NNI-dist

- There exists a polynomial time (log n)-approximated algorithm.

Subtree Transfer (STT)

- Consider a degree-3 unrooted tree T
- A subtree transfer operation is the operation of detaching a subtree and reattached it to the middle of another edge
- An STT operation is charged by the number of nodes the subtree is transferred.



The cost of this STT operation is 2

STT-dist

- Given two degree-3 unrooted trees T_1 and T_2,
- STT-dist(T_1, T_2) is the minimum cost series of STT operations which transform T_1 to T_2.
- T_1 and T_2 looks similar if STT-dist(T_1, T_2) is small.

Property of STT-dist

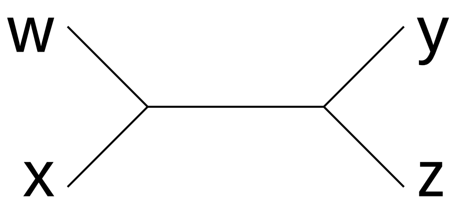
- STT-dist(T_1, T_2) = NNI-dist(T_1, T_2)
- Proof:
  - STT-dist(T1, T2) ≤ NNI-dist(T_1, T_2) because each NNI-operation is an STT-operation.
  - STT-dist(T1, T2) ≥ NNI-dist(T_1, T_2) because each STT-operation of cost k can be simulated by k NNI-operations.

More on STT-dist

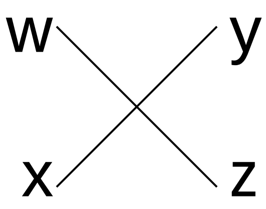
- Based on the result for NNI-operation, we have
  - STT-dist(T_1, T_2) is NP-hard to compute.
  - There exists a polynomial time (log n)-approximated algorithm to compute STT-dist(T_1, T_2)

Quartet

- A quartet is a phylogenetic tree with 4 species.



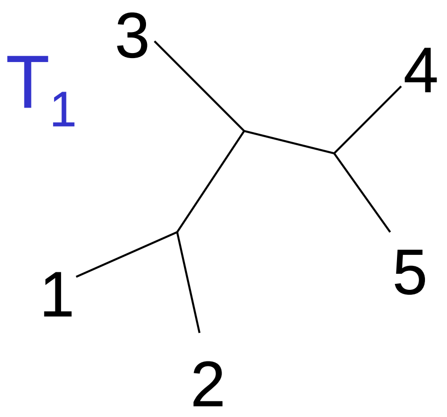
Butterfly quartet

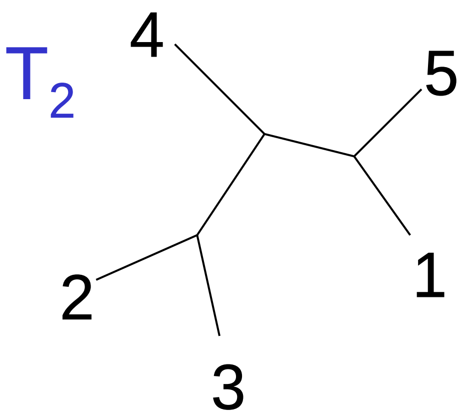


Star quartet

Quartet distance

- Given two unrooted trees T_1 and T_2,
  - The quartet distance is the number of set of 4 species {w,x,y,z} such that
    - $T_1|\{w,x,y,z\} \neq T_2|\{w,x,y,z\}$.





{1,2,3,4}: different
{1,2,3,5}: different
{1,2,4,5}: different
{1,3,4,5}: different
{2,3,4,5}: same

Quartet distance = 4

Previous works

- When T_1 and T_2 are of degree-3,
  - Steel and Penny (1993): $O(n^3)$ time.
  - Bryant et al. (2000): $O(n^2)$ time.
  - Brodal et al. (2003): $O(n \log n)$ time
- When T_1 and T_2 are of degree-d,
  - Christiansen et al. (2005): $O(n^3)$ time or $O(d^2n^2)$ time.

Property

- Number of different quartets + number of shared quartets = $\binom{n}{4}$.

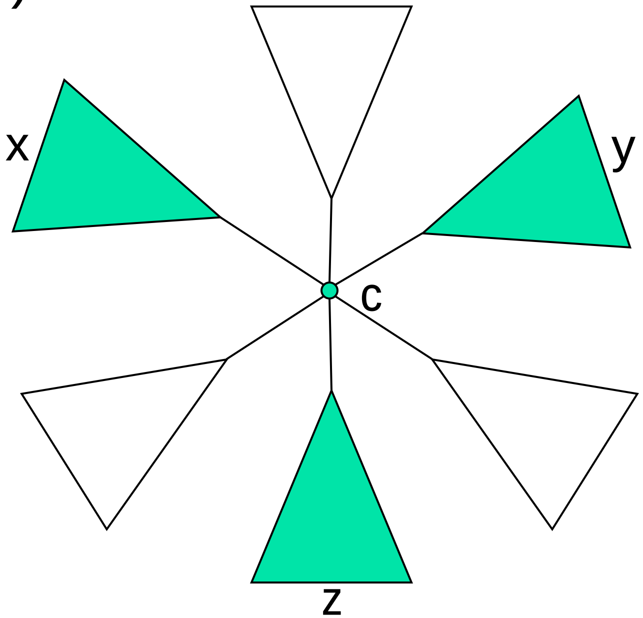
Brute-force method

- count = 0;
- for every $\{w,x,y,z\} \subseteq S$,
  - if $T_1|\{w,x,y,z\} = T_2|\{w,x,y,z\}$, count++;
- Report $\binom{n}{4}$ - count;

- The running time is at least $O(n^4)$.

Observation

- Consider a tree T which is leaf-labeled by S.
- For any $\{x,y,z\} \subseteq S$,
  - There exists a unique internal node c in T such that c appears in any paths from x to y, y to z, and x to z.
- We denote $T^{c,x}$ be a set of species which appear in the child subtree containing x. (Similarly, we define $T^{c,y}$ and $T^{c,z}$.)
- Let $T^{c,rest} = S - (T^{c,x} \cup T^{c,y} \cup T^{c,z})$.



- Note that, for all species $w \in T^{c,x}$, the quartet for $\{w,x,y,z\}$ in T is $wx|yz$.
- Similarly, for all species $w \in T^{c,y}$, the quartet for $\{w,x,y,z\}$ in T is $wy|xz$.
- Similarly, for all species $w \in T^{c,z}$, the quartet for $\{w,x,y,z\}$ in T is $wz|xy$.
- Similarly, for all species $w \in T^{c,rest}$, the quartet for $\{w,x,y,z\}$ in T is a star quartet.

- Consider two trees T_1 and T_2.
- The number of shared butterfly quartets involving x,y,z is $|T_1^{c,x} \cap T_2^{c',x}| + |T_1^{c,y} \cap T_2^{c',y}| + |T_1^{c,z} \cap T_2^{c',z}| - 3$.
- The number of shared star quartets involving x,y,z is $|T_1^{c,rest} \cap T_2^{c',rest}|$.

Algorithm

- count = 0;
- Compute $|R_1 \cap R_2|$ for any subtree R_1 of T_1 and any subtree R_2 of T_2.
- For every $\{x,y,z\} \subseteq S$,
  - Let c be the center of x,y, and z in T_1.
  - Let $T_1^{c,x}$, $T_1^{c,y}$, and $T_1^{c,z}$ be the subtrees attached to c containing x, y, z, respectively.
  - Set $T_1^{c,rest} = S - (T_1^{c,x} \cup T_1^{c,y} \cup T_1^{c,z})$.
  - Let c' be the center of x,y, and z in T_2.
  - Let $T_2^{c',x}$, $T_2^{c',y}$, and $T_2^{c',z}$ be the subtrees attached to c' containing x, y, z, respectively.
  - Set $T_2^{c',rest} = S - (T_2^{c',x} \cup T_2^{c',y} \cup T_2^{c',z})$.
  - $count = count + |T_1^{c,x} \cap T_2^{c',x}| + |T_1^{c,y} \cap T_2^{c',y}| + |T_1^{c,z} \cap T_2^{c',z}| + |T_1^{c,rest} \cap T_2^{c',rest}| - 3$
- Report $\binom{n}{4}$ - count/4;

Computing $|R_1 \cap R_2|$

- For any $e=(u,v)$ in T_1
  - e partitions T_1 into two subtrees with leaf sets Q_v and $Q_u = S-Q_v$.
  - For any $e'=(u',v')$ in T_2,
    - e' partitions T_2 into two subtrees with leaf sets $Q_{v'}$ and $Q_{u'}=S-Q_{v'}$.
    - $|T_1^{u,v} \cap T_2^{u',v'}|=|Q_v \cap Q_{v'}|$
- The running time is $O(n^3)$.
- The algorithm can be improved to $O(n^2)$ time.

Computing $|T_1^{c,rest} \cap T_2^{c',rest}|$ in O(1) time

- $|T_1^{c,rest} \cap T_2^{c',rest}| = |T_2^{c',rest}| - (|T_1^{c,x} \cap T_2^{c',rest}| + |T_1^{c,y} \cap T_2^{c',rest}| + |T_1^{c,z} \cap T_2^{c',rest}|)$

- $|T_2^{c',rest}| = |S| - |T_2^{c',x}| - |T_2^{c',y}| - |T_2^{c',z}|$

- $|T_1^{c,x} \cap T_2^{c',rest}| = |T_1^{c,x}| - (|T_1^{c,x} \cap T_2^{c',x}| + |T_1^{c,x} \cap T_2^{c',y}| + |T_1^{c,x} \cap T_2^{c',z}|).$
- $|T_1^{c,y} \cap T_2^{c',rest}| = |T_1^{c,y}| - (|T_1^{c,y} \cap T_2^{c',x}| + |T_1^{c,y} \cap T_2^{c',y}| + |T_1^{c,y} \cap T_2^{c',z}|).$
- $|T_1^{c,z} \cap T_2^{c',rest}| = |T_1^{c,z}| - (|T_1^{c,z} \cap T_2^{c',x}| + |T_1^{c,z} \cap T_2^{c',y}| + |T_1^{c,z} \cap T_2^{c',z}|).$

Time complexity

- $|R_1 \cap R_2|$ can be computed in $O(n^2)$ time.
- For every $\{x,y,z\} \subseteq S$,
  - $|T_1^{c,x} \cap T_2^{c',x}|$, $|T_1^{c,y} \cap T_2^{c',y}|$, $|T_1^{c,z} \cap T_2^{c',z}|$, and $|T_1^{c,rest} \cap T_2^{c',rest}|$ can be computed in $O(1)$ time.

- In total, the running time is $O(n^3)$.

Consensus Tree

Consensus tree problem

- Given a set of n species S
- Given a set of trees $\{T_1, T_2, \ldots, T_m\}$
  - where the leaves of every T_i are labeled by S

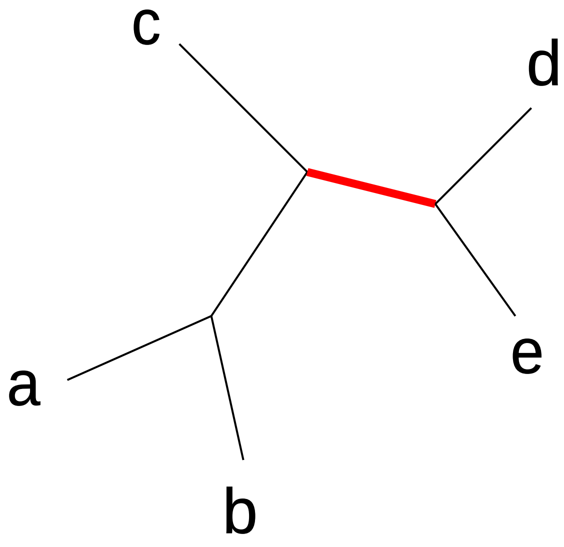
- Question: Find a tree which summarizes all the trees $T_1, T_2, \ldots, T_m$.

Applications

1. Find the bootstrapping tree.
2. Given a set of gene trees, infer the species tree.

Split of an edge

- Each edge can partition the set of species
- In the following tree, the red edge partition the species into {a, b, c} and {d, e}.
- So, the split of the red edge is {a,b,c}|{d,e}.
- Note that for any x∈S, {x}|S-{x} must be a valid split due to the leaf edge connecting the leaf x.



Properties of split

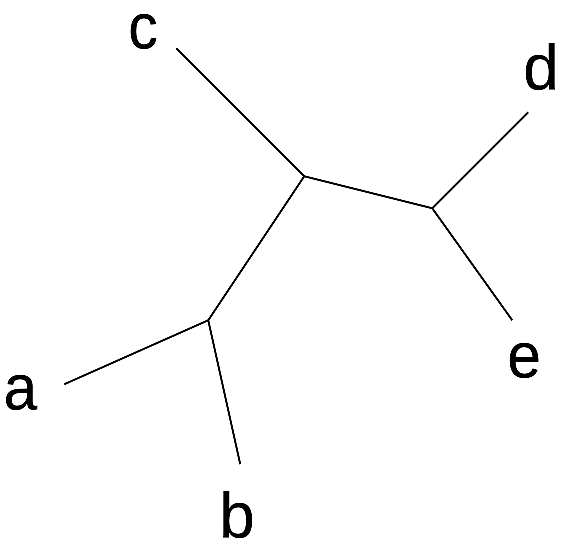
- Two splits A|S-A and B|S-B are compatible if $A \subseteq B$ or $A \subseteq S-B$ or $B \subseteq A$ or $B \subseteq S-A$.

- For any tree T, any two splits of T are compatible.

- Given a set of splits W which are pairwise compatible, there exists a tree T which contains all the splits in W.

Example

- There is a one-to-one correspond between the tree and the set of splits of all its edges.

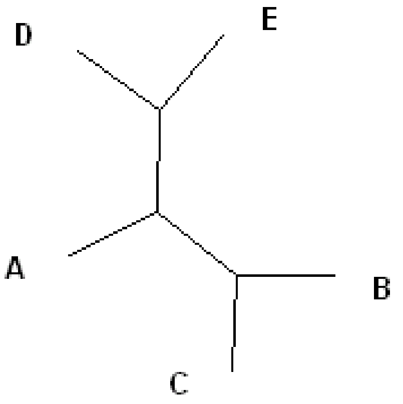


{a}|{b,c,d,e}
{b}|{a,c,d,e}
{c}|{a,b,d,e}
{d}|{a,b,c,e}
{e}|{a,b,c,d}
{a,b}|{c,d,e}
{a,b,c}|{d,e}

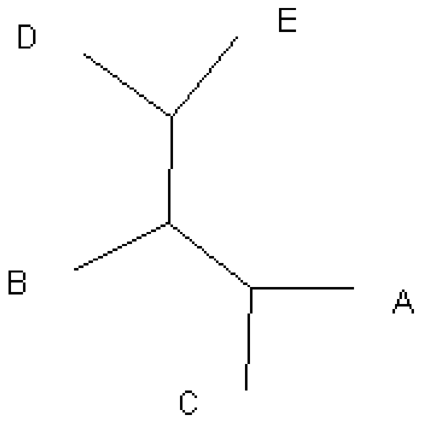
Strict consensus tree

- The strict consensus tree T of $\{T_1, T_2, \ldots, T_m\}$ contains exactly those splits which appear in all T_i.
- The strict consensus tree always exists.

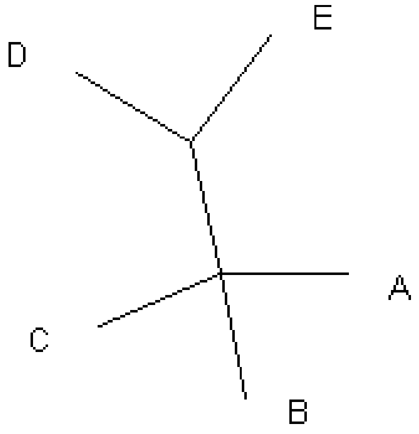
- Example: T is the strict consensus tree of T_1 and T_2.



T_1



T_2



T

The strict consensus tree always exists

- Let W_i be the set of splits of T_i, $i=1,2,\ldots,m$.
- The set of splits of the strict consensus tree is $W_1 \cap W_2 \cap \ldots \cap W_m$.

How to find strict consensus tree of two trees?

Input: Two trees T_1, T_2

Output: the strict consensus tree

- Run O(n) time Day's algorithm to find all the good edges.
- Generate the strict consensus tree.
  - Precisely, the strict consensus tree is formed by contracting all bad edges.
- Time complexity: O(n).

How to find strict consensus tree of m trees?

Input: m trees T_1, T_2, …, T_m.

Output: the strict consensus tree

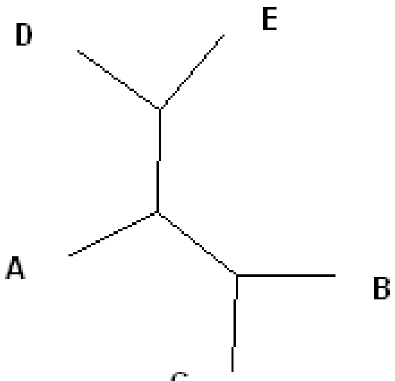
- Let $T=T_1$.
- For i = 2 to m
  - Set T be the strict consensus tree of T and T_i.
- Return T;

- Time complexity: O(mn)

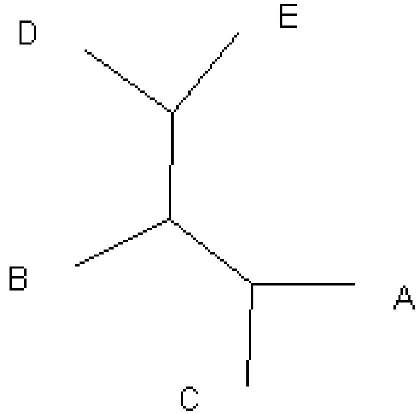
Majority rule tree

- The majority rule tree contains exactly those splits that appear in more than half of the input trees.
- The majority rule tree is unique (why?) and always exists.

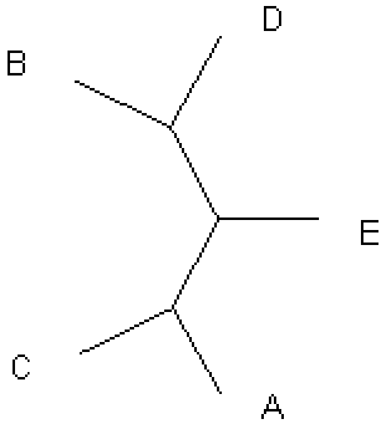
- Example: T is also the majority rule tree of T_1 , T_2, and T_3.



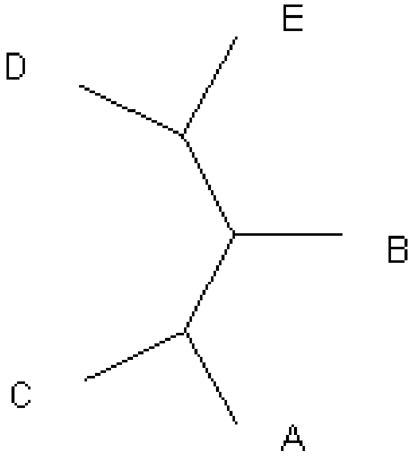
T_1



T_2



T_3



T

- Given two trees, the majority rule tree is the same as the strict consensus tree.

Algorithm

Input: m trees T_1, T_2, …, T_m.

Output: the majority tree

1. Count the occurrences of each split, storing the counts in a table.
2. Select those splits with occurrences > m/2.
3. Using the selected splits, create the majority tree.

Step 1

- For each T_i,
  - We run Day's algorithm for (T_i, T_j) for all j = i+1, …, m.
  - For every edge in T_i which are unmarked, we count the number of good edges in T_j for $j>i$.
  - Also, we mark those good edges in T_j as counted.

- Time complexity: Each T_i takes O(nm) time. Hence, Step 1 takes $O(m^2n)$ time.

A lemma for step 3

- Suppose we rooted the majority consensus tree at the leaf 1.

- Lemma: If p is a parent split of c in the majority tree, there exists a tree T_j which contains both splits p and c.
- Proof: Both p and c appears in more than m/2 trees. By pigeon-hole principle, there exists a tree which contains both p and c.

Step 3

- We root all tree T_i at the leaf 1.
- For each T_i, we get T'_i which is the tree formed by contracting all the non-majority splits.
- Let T' be T'_1.
- For each i=2, …, m,
  - We traverse T'_i in depth first search order.
  - For any split c in T'_i, let p be its parent split in T'_i.
  - If c does not exists in T', we introduce c as the child split of p in T'. (Note: p must exists in T' since we traverse the tree in depth first search order.)

- Time complexity: O(nm) time.

Time complexity for constructing majority consensus tree

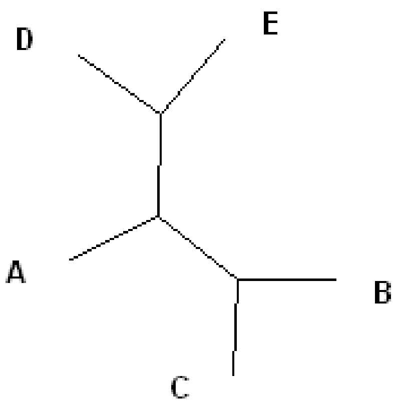
- In summary, the majority consensus tree can be constructed in O(nm^2) time.

- Note: Majority consensus tree can be built in O(nm) expected time.
  - Nina Amenta, Frederick Clarke and Katherine St. John. A Linear-time Majority Tree Algorithm, 216-227, WABI, 2003.

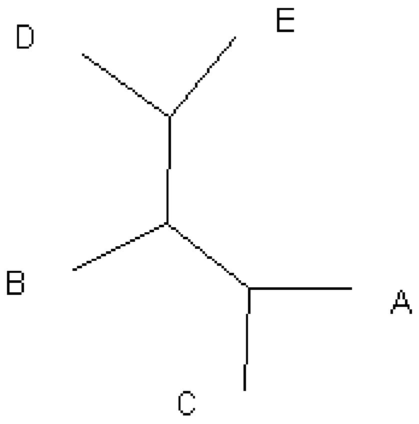
Symmetric difference distance

- Denote d(T_1, T_2) be the symmetric difference between T_1 and T_2.
  - The number of splits appearing in one tree but not the other.

- Example: For T_1 and T_2, {A,D,E}|{B,C} only appears in T_1 and {A,C}|{B,D,E} only appears in T_2. Hence, d(T_1, T_2) = 2.



T_1



T_2

Median tree

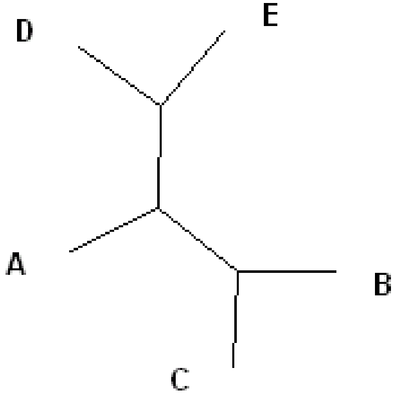
- The median tree T for T_1, T_2, …, T_m minimizes
  - $\Sigma_{i=1..m}$ d(T, T_i).

- Barthelemy and McMorris showed that majroity rule tree is the same as the median tree.

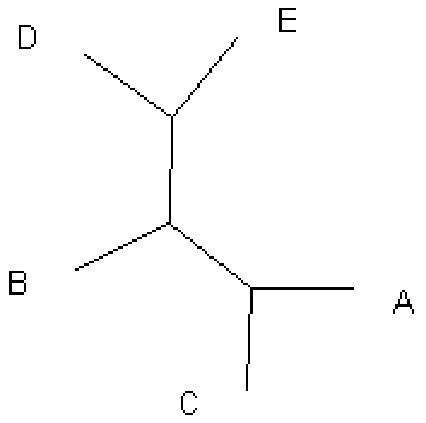
Asymmetric median consensus tree

- For every split, its weight is defined to be the number of input trees containing it.
- The asymmetric median tree a set of splits which maximizes the total weight.
- The asymmetric tree always exists.

- Example: Both T_1 and T_2 are also the asymmetric median trees of T_1 and T_2.



T_1

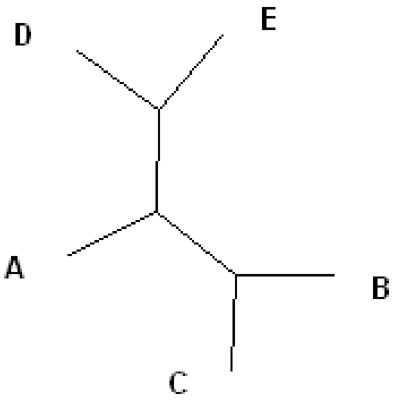


T_2

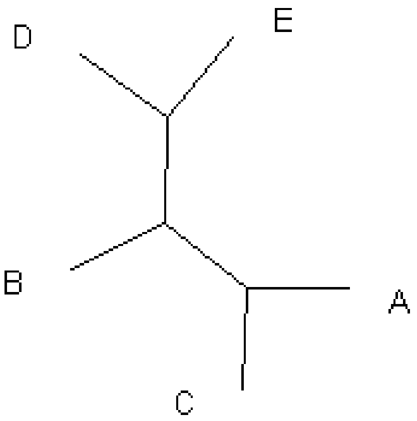
Asymmetric difference distance

- Denote $d_a(T_1, T_2)$ be the symmetric difference between T_1 and T_2.
  - The number of splits appearing in T_2 but T_1.

- Example: For T_1 and T_2, ({A,C}, {B,D,E}) only appears in T_2 but not T_1. Hence, $d_a(T_1, T_2) = 1$.



T_1



T_2

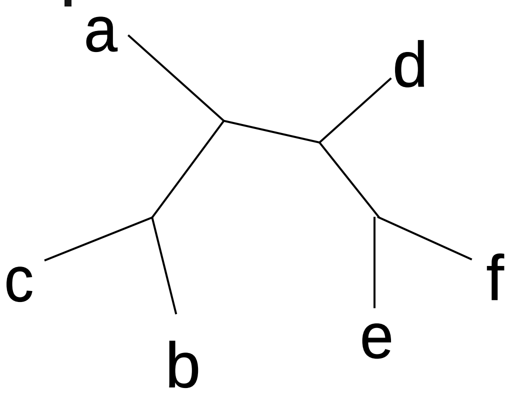
Property of asymmetric median tree

- The asymmetric median tree T for T_1, T_2, …, T_m minimizes
  - $\Sigma_{i=1..m}\, d_a(T, T_i)$.

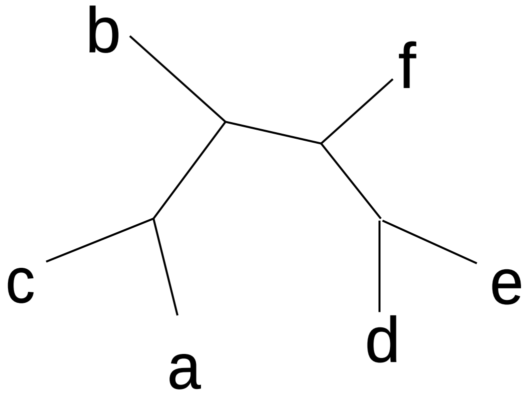
Greedy consensus tree

- Greedy consensus tree is created by
  - Sequentially include split one by one.
  - Every iteration, we include the most frequent split that is compatible with the included splits (breaking the ties randomly).
  - Do this until we cannot include any other split.

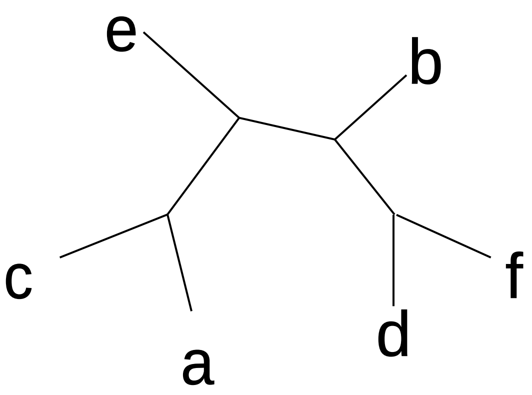
Example



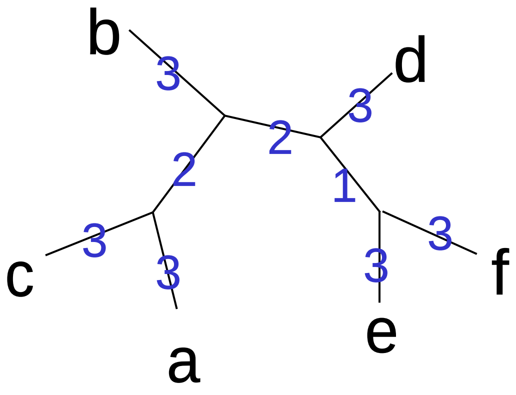
T_1



T_2



T_3



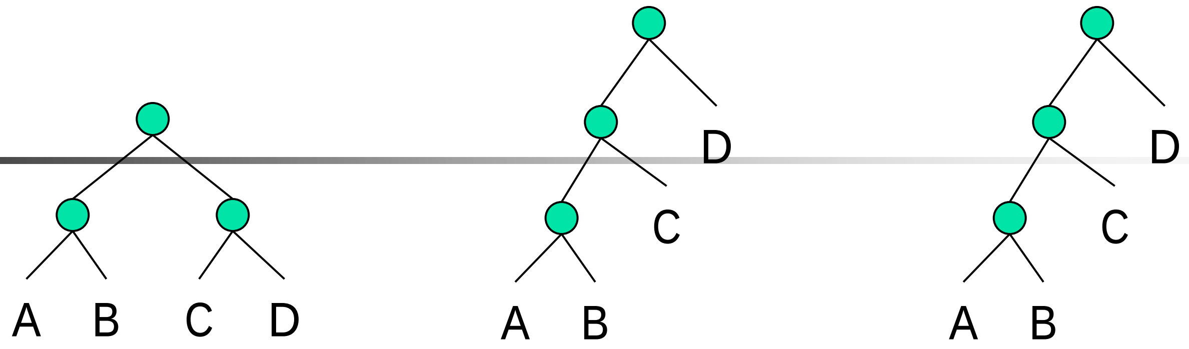
T

- Greedy consensus tree is a refinement of the majority-rule consensus tree.

R* tree

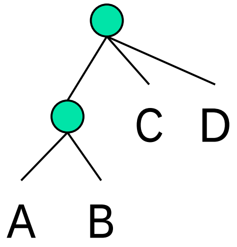
- For each set of 3 species, find the most commonly occurring triplet e.g., C|AB, B|AC or A|BC.
- Build the tree from the most commonly occurring triplets.

Example of R* tree



- C|AB – 3, A|BC – 0, B|AC – 0
- A|CD – 1, C|AD – 1, D|AC – 1
- B|CD – 1, C|BD – 1, D|BC – 1
- D|AB – 3, A|BD – 0, B|AD – 0

C|AB, D|AB



Correctness

- Lemma: Let C be the set of most commonly occurring triplets. There exists a most resolved tree which is consistent with all triplets in C. Also, such tree is unique.
- Proof:
  - Steel, M. The complexity of reconstructing trees from qualitative characters and subtrees. Journal of Classification, 9:91–116, 1992.

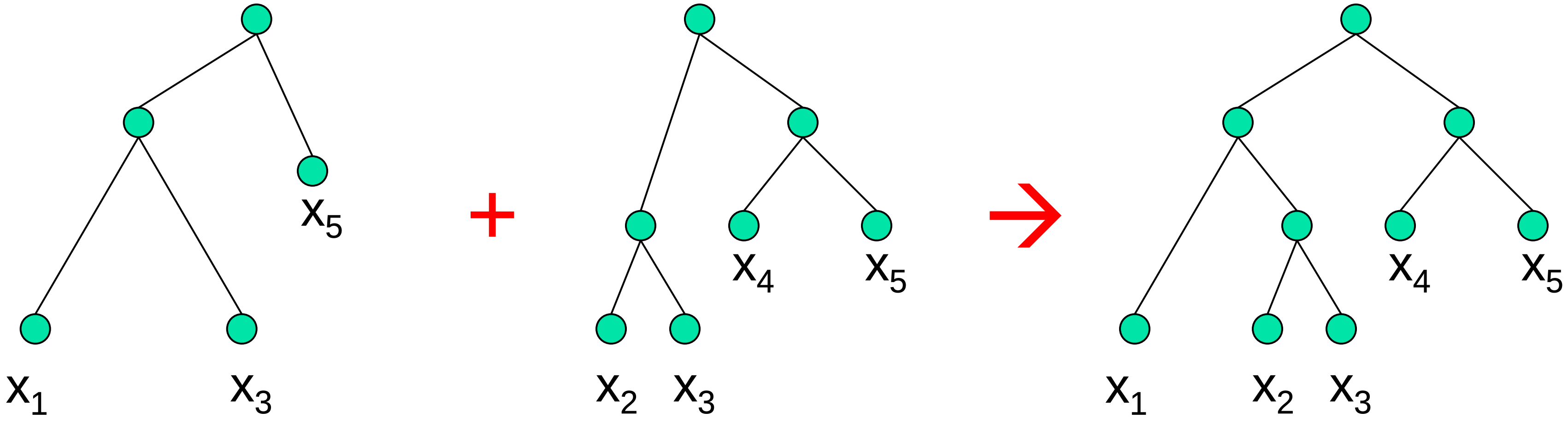
Algorithm for computing R* tree

1. Computing the number of occurrences of all triplets in the m trees.
   - There are n^3 triplets in each tree and there are m trees. Hence, it takes $O(m\ n^3)$ time.
2. For each set of 3 species {A, B, C}, find the most commonly occurring triplet.
   - This step takes $O(n^3)$ time.
3. Constructing the tree from the set C of the most commonly occurring triplets.
   - By triplet method, this step takes $O(\min\{O(k \log^2 n), O(k + n^2 \log n)\})$ where $k=|C|<n^3$. Hence, this step takes $O(n^3)$ time.

- The whole algorithm runs in $O(m\ n^3)$ time.

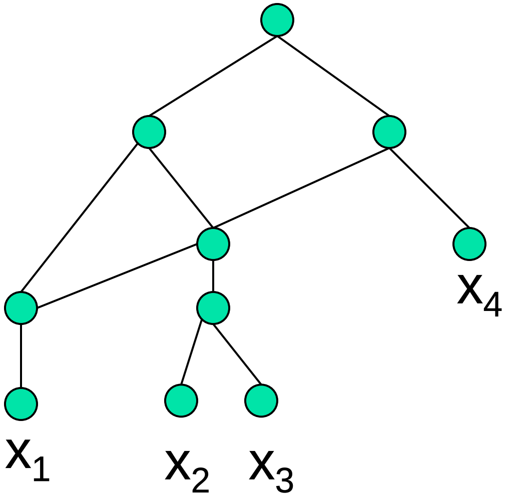
Other directions of Phylogenetic study

- Supertree
  - No method can find the phylogenetic tree for all species
  - To find the phylogenetic tree for all species, one method is to combine a number of phylogenetic trees
  - The combined tree is called supertree.
  - The difficulties of this problem is to resolve the conflicts among the trees.



Other directions of Phylogenetic study

- Phylogenetic network
  - Evolution is in fact more than a point mutation. We have other types of evolutions. Like:
    - Hybridization.
      - E.g. tiger + lion → tiglion
    - Horizontal gene transfer
      - E.g. Bovine Corona Virus (genbank ID NC_003045) + Murine Hepatitis Virus (genbank ID AF201929) → SARS
  - Phylogenetic tree cannot model those types of evolutions.



Reference (Robinson-Foulds distance and Day's algorithm)

- D. F. Robinson and L. R. Foulds. Comparison of phylogenetic trees. Mathematical Biosciences, 53:131-147, 1981.
- W. H. E. Day. Optimal algorithms for comparing trees with labeled leaves. Journal of Classification, 2:7-28, 1985.

Reference (NNI-distance and Subtree-transfer distance)

- M. Li, J. Tromp, and L. X. Zhang. Some notes on the nearest neighbour interchange distance. Journal of Theoretical Biology, 182:463-467, 1996.
- B. DasGupta, X. He, T. Jiang, M. Li, and J. Tromp. On the linear-cost subtree-transfer distance between phylogenetic trees. Algorithmica, 25(2):176-195, 1999.
- B. Das Gupta, X. He, T. Jiang, M. Li, J. Tromp, and L. Zhang. On distance between phylogenetic trees. In Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 427-436, 1997.
- J. Hein. Reconstructing evolution of sequences subject to recombination using parsimony. Mathematical Biosciences, 98:185-200, 1990.
- J. Hein. A heuristic method to reconstruct the history of sequences subject to recombination. Journal of Molecular Evolution, 36:396-405, 1993.
- G. W. Moore, M. Goodman, and J. Barnabas. An iterative approach from teh standpoint of the additive hypothesis to the dendrogram problem posed by molecular data sets. Journal of Theoretical Biology, 38:423-457, 1973.
- D. F. Robinson. Comparison of labeled trees with valency three. Journal of Combinatorial Theory, 11:105-119, 1971.

Reference for consensus tree

- Nina Amenta, Frederick Clarke, and Katherine St. John. A linear-time majority tree algorithm. WABI, 216-227, 2003.
- T. Margush and F.R. McMorris. Consensus n-trees. Bulletin of Mathematical Biology, 43:239–244, 1981.