CPM 2024 Summer School

– Phylogenetic Consensus Trees –

Lecturer: Jesper Jansson
Kyoto University
2024-06-20

PART I : Introduction

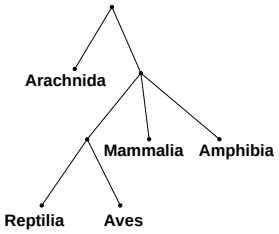
Phylogenetic tree

Definition

A phylogenetic tree is a rooted, unordered tree whose leaves are uniquely labeled and in which every internal node has ≥ 2 children.

Can describe divergent evolutionary history for a set of objects, where:

"objects" = Biological species, proteins, types of tumor cells in a patient, natural languages, hand-copied manuscripts, SARS-CoV-2 strains, or ...



Main idea:

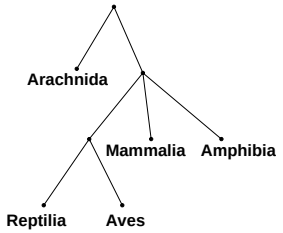
- Represent objects by *leaves* in the tree.
- Select branching structure so that *internal nodes* correspond to common ancestors.

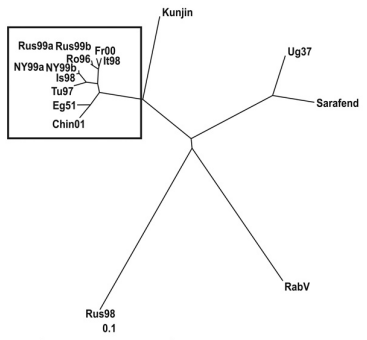
Phylogenetic tree

Variants

Depending on the application, phylogenetic trees may:

- be *rooted* or *unrooted*
- have *weighted* or *unweighted* edges
- have *bounded degree* (maximum # of children of each internal node)





Consensus methods

- During the last 150 years, numerous methods for reconstructing phylogenetic trees have been proposed.
- For various reasons, inferring an accurate phylogenetic tree can be a difficult problem.
- For example, small changes in the data may produce trees with very different structures.
- Furthermore, many of the underlying computational problems are $\mathcal{NP}$-hard optimization problems.

Consensus methods, cont.

One approach:

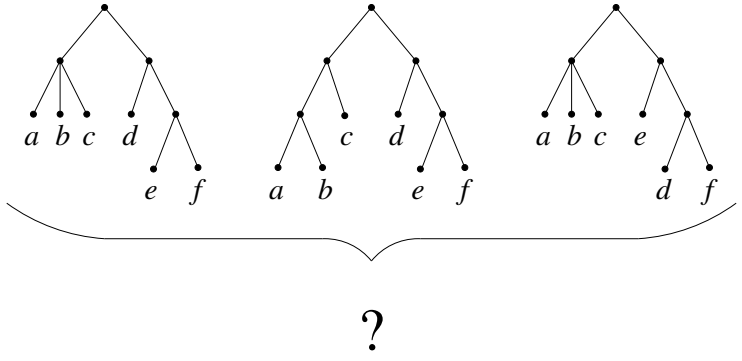
- Multiple data sets.
- Apply resampling techniques like bootstrapping to the same data set.
- Apply different tree reconstruction algorithms.
- Assume different models of evolution.
- Using heuristics for maximizing parsimony.

$\Rightarrow$ A collection of alternative trees for the same leaf label set.

Then, represent all of the obtained trees by one tree.
"consensus tree"

Consensus trees, example

Ideally, a consensus tree should summarize all the branching information contained in the input set $\mathcal{S}$ in the best way possible.



(Example from `https://www.geol.umd.edu/%7etholtz/G331/`)

Different types of consensus trees

Many different definitions of "in the best way" exist...
(Depends on which criteria are used to resolve conflicts.)

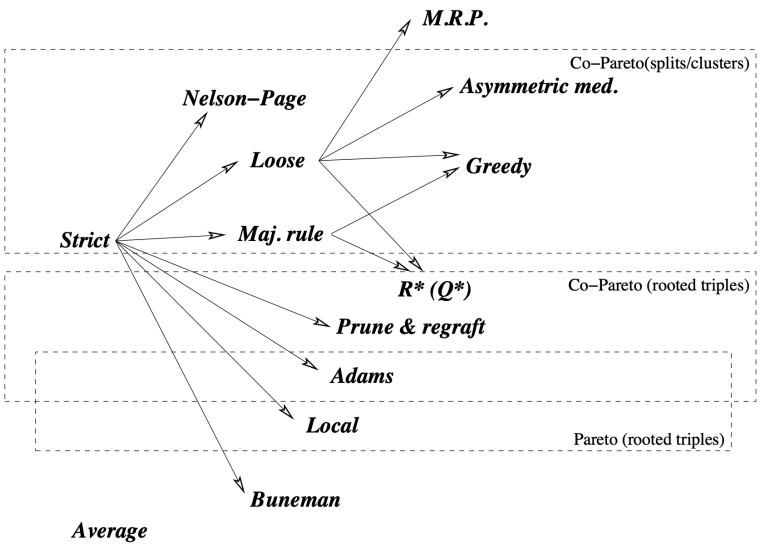
$\Rightarrow$

- Strict consensus [Sokal, Rohlf; 1981]
- Majority rule consensus [Margush, McMorris; 1981]
- Greedy consensus [Felsenstein; 1993]
- Loose consensus [Bremer; 1990]
- Adams consensus [Adams; 1972]
- Q* consensus [Berry, Gascuel; 1997] / R* consensus [Bryant; 2003]
- Local consensus (RV-I, RV-II, RV-III) [Kannan *et al.*; 1998]
- Frequency difference consensus [Goloboff *et al.*; 2003]
- etc.

Each type of consensus tree has some advantages & disadvantages. See:

- D. Bryant. A classification of consensus methods for phylogenetics. Vol. 61 of DIMACS Series in DMTCS, pp. 163–184, AMS, 2003.

Bryant's classification of consensus trees



(Figure from D. Bryant, Vol. 61 of DIMACS Series in DMTCS, pp. 163–184, 2003.)

In one of our ongoing research projects, we are developing fast algorithms for constructing various types of consensus trees.

Joint work with Zhaoxian Li, Ramesh Rajaby, Chuanqi Shen, Wing-Kin Sung, Ali Tabatabaee, and Yutong Yang.

Today's talk will introduce some of the most popular consensus trees, look at how they are related to each other, and present some fast algorithms.

Remainder of the talk:

- PART II : Cluster-Based Consensus Trees
- PART III : Rooted Triplet-Based Consensus Trees

PART II : Cluster-Based Consensus Trees

Notation

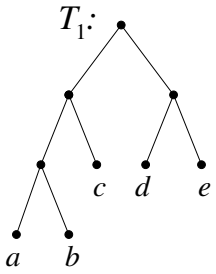
Let T be a phylogenetic tree.

- $V(T)$ = the set of all nodes in T
- $\Lambda(T)$ = the set of all leaf labels in T
- Any subset C of $\Lambda(T)$ is called a *cluster of* $\Lambda(T)$.
  If $|C| = 1$ or $C = \Lambda(T)$ then C is a *trivial cluster*.
- For every node u in T, define $T[u]$ = the subtree of T rooted at u (i.e., the subtree of T induced by u and all of u's descendants).
  $\Lambda(T[u])$ is called *the cluster associated with* u.
- The *cluster collection of* T is the set
  $\mathcal{C}(T) = \bigcup_{u \in V(T)} \{\Lambda(T[u])\}$.

Notation, cont.

- The *cluster collection of* T is the set
$\mathcal{C}(T) = \bigcup_{u \in V(T)} \{\Lambda(T[u])\}$.

Example:



$$\mathcal{C}(T_1) = \Big\{\{a\},\{b\},\{c\},\{d\},\{e\},\{a,b\},\{a,b,c\},\{d,e\},\{a,b,c,d,e\}\Big\}$$

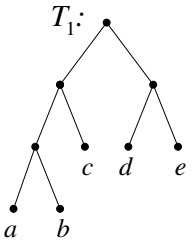
- When a cluster C belongs to $\mathcal{C}(T)$, we say that C *occurs in* T.
Example: The cluster $\{a,b,c\}$ occurs in the tree T_1 above.

Notation, cont.

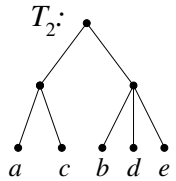
- Two clusters C_1, C_2 are *compatible* if $C_1 \subseteq C_2$, $C_2 \subseteq C_1$, or $C_1 \cap C_2 = \emptyset$. In this case, we write $C_1 \smile C_2$; otherwise, $C_1 \not\smile C_2$.

- Any cluster C is said to be *compatible with* the tree T if $C \smile \Lambda(T[u])$ for every node $u \in V(T)$, and we write $C \smile T$.

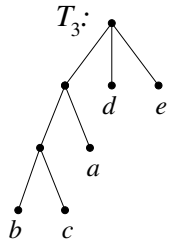
Example: The cluster $\{d,e\}$ occurs in T_1 but not in T_2 and T_3. However, $\{d,e\} \smile T_2$ and $\{d,e\} \smile T_3$.



ab, *abc*, *de*



ac, *bde*



bc, *abc*

Cluster-based consensus trees

These concepts are enough to define (at least) five types of consensus trees!

1. Strict consensus tree
[Sokal, Rohlf; 1981]
2. Majority rule consensus tree
[Margush, McMorris; 1981]
3. Loose consensus tree
[Bremer; 1990]
4. Frequency difference consensus tree
[Goloboff, Farris, Källersjö, Oxelman, Ramírez, Szumik; 2003]
5. Greedy consensus tree
[Felsenstein; 1989]

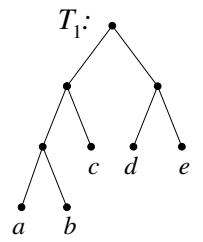
We'll refer to them collectively as cluster-based consensus trees.

1. Strict consensus tree [Sokal, Rohlf; 1981]

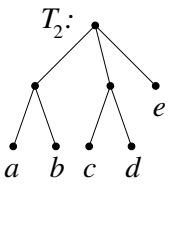
Let $\mathcal{S} = \{T_1, \ldots, T_k\}$ be a set of trees with $\Lambda(T_1) = \ldots = \Lambda(T_k) = L$ for some leaf label set L. **Note:** All trees have the same leaf label set L.

The strict consensus tree of $\mathcal{S}$ is the (unique) tree T such that $\Lambda(T) = L$ and $\mathcal{C}(T)$ consists of those clusters that occur in every tree in $\mathcal{S}$, i.e., $\mathcal{C}(T) = \bigcap_{i=1}^{k} \mathcal{C}(T_i)$.

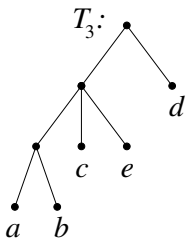
Example:



ab, *abc*, *de*

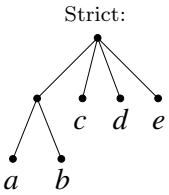


ab, *cd*



ab, *abce*

⇒



ab

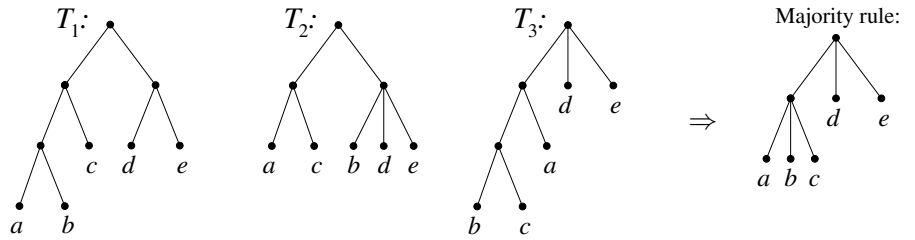
2. Majority rule consensus tree [Margush, McMorris; 1981]

Let $\mathcal{S} = \{T_1, \ldots, T_k\}$ be a set of trees with $\Lambda(T_1) = \ldots = \Lambda(T_k) = L$ for some leaf label set L. **Note:** All trees have the same leaf label set L.

A cluster that occurs in more than $k/2$ of the trees in $\mathcal{S}$ is a *majority cluster*.

The majority rule consensus tree of $\mathcal{S}$ is the (unique) tree T such that $\Lambda(T) = L$ and $\mathcal{C}(T)$ consists of all majority clusters.

Example:



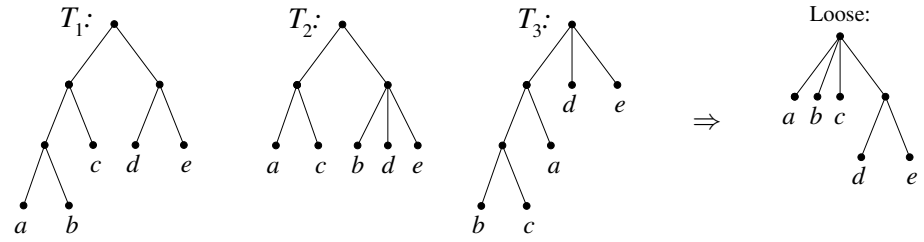
ab, *abc*, *de* | *ac*, *bde* | *bc*, *abc* | *abc*

3. Loose consensus tree [Bremer; 1990]

Let $\mathcal{S} = \{T_1, \ldots, T_k\}$ be a set of trees with $\Lambda(T_1) = \ldots = \Lambda(T_k) = L$ for some leaf label set L. **Note:** All trees have the same leaf label set L.

The loose consensus tree of $\mathcal{S}$ is the tree T such that $\Lambda(T) = L$ and $\mathcal{C}(T)$ consists of all clusters that occur in at least one tree in $\mathcal{S}$ and that are compatible with all trees in $\mathcal{S}$.

Example:



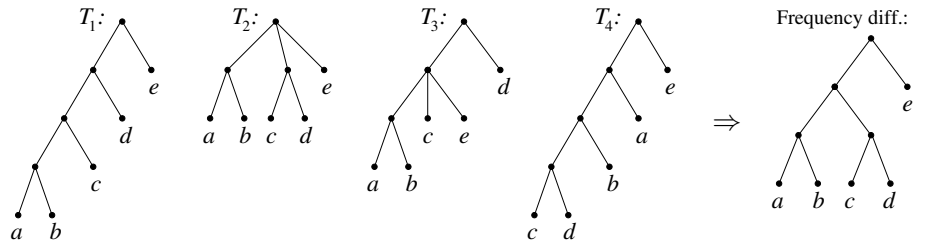
ab, *abc*, *de* | *ac*, *bde* | *bc*, *abc* | *de*

4. Frequency difference consensus tree [Goloboff *et al.*; 2003]

Let $\mathcal{S} = \{T_1, \ldots, T_k\}$ be a set of trees with $\Lambda(T_1) = \ldots = \Lambda(T_k) = L$ for some leaf label set L. **Note:** All trees have the same leaf label set L.

The frequency difference consensus tree of $\mathcal{S}$ is the tree T such that $\Lambda(T) = L$ and $\mathcal{C}(T)$ contains every cluster C occurring more often in $\mathcal{S}$ than each of the clusters in $\mathcal{S}$ that is incompatible with C.

Example:



ab, *abc*, *abcd* | *ab*, *cd* | *ab*, *abce* | *cd*, *bcd*, *abcd* | *ab* (3), *cd* (2), *abcd* (2)

5. Greedy consensus tree [Felsenstein; 1989]

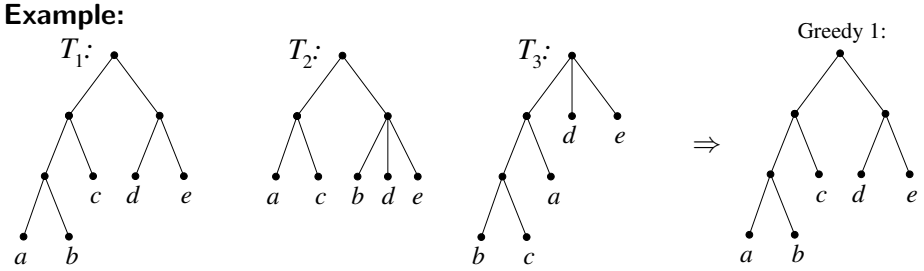
Let $\mathcal{S} = \{T_1, \ldots, T_k\}$ be a set of trees with $\Lambda(T_1) = \ldots = \Lambda(T_k) = L$ for some leaf label set L. **Note:** All trees have the same leaf label set L.

Make a list $\mathcal{X}$ of all clusters in $\mathcal{S}$, sorted by the number of occurrences in $\mathcal{S}$ in non-increasing order, and construct a set $\mathcal{Y}$ of clusters as follows:

Initialize $\mathcal{Y} := \emptyset$. Traverse $\mathcal{X}$ and for each cluster C encountered, if C and C' are compatible for all $C' \in \mathcal{Y}$ then let $\mathcal{Y} := \mathcal{Y} \cup \{C\}$.

A greedy consensus tree of $\mathcal{S}$ is a tree T such that $\Lambda(T) = L$ and $\mathcal{C}(T) = \mathcal{Y}$.

Example:



ab, *abc*, *de* | *ac*, *bde* | *bc*, *abc* | *abc*, *ab*, *de*

5. Greedy consensus tree [Felsenstein; 1989]

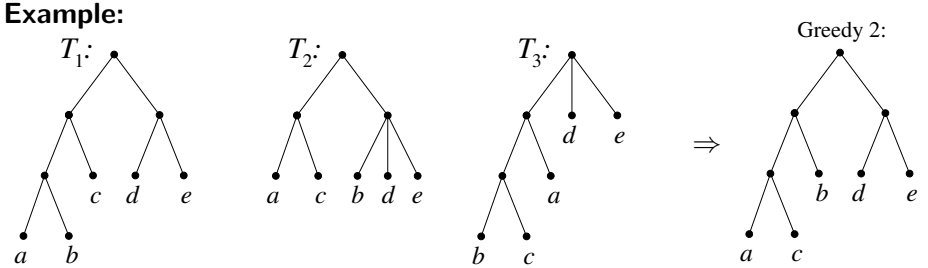
Let $\mathcal{S} = \{T_1, \ldots, T_k\}$ be a set of trees with $\Lambda(T_1) = \ldots = \Lambda(T_k) = L$ for some leaf label set L. **Note:** All trees have the same leaf label set L.

Make a list $\mathcal{X}$ of all clusters in $\mathcal{S}$, sorted by the number of occurrences in $\mathcal{S}$ in non-increasing order, and construct a set $\mathcal{Y}$ of clusters as follows:

Initialize $\mathcal{Y} := \emptyset$. Traverse $\mathcal{X}$ and for each cluster C encountered, if C and C' are compatible for all $C' \in \mathcal{Y}$ then let $\mathcal{Y} := \mathcal{Y} \cup \{C\}$.

A greedy consensus tree of $\mathcal{S}$ is a tree T such that $\Lambda(T) = L$ and $\mathcal{C}(T) = \mathcal{Y}$.

Example:



5. Greedy consensus tree [Felsenstein; 1989]

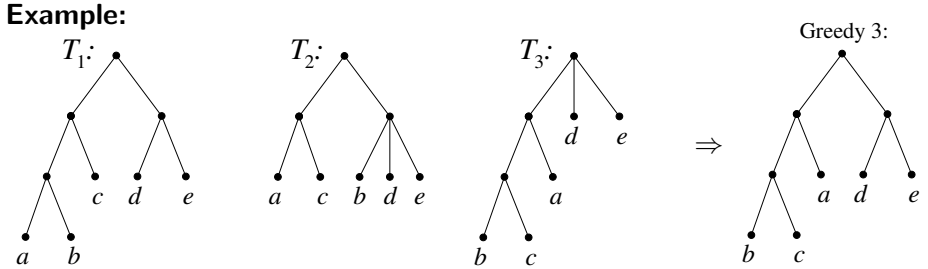
Let $\mathcal{S} = \{T_1, \ldots, T_k\}$ be a set of trees with $\Lambda(T_1) = \ldots = \Lambda(T_k) = L$ for some leaf label set L. **Note:** All trees have the same leaf label set L.

Make a list $\mathcal{X}$ of all clusters in $\mathcal{S}$, sorted by the number of occurrences in $\mathcal{S}$ in non-increasing order, and construct a set $\mathcal{Y}$ of clusters as follows:

Initialize $\mathcal{Y} := \emptyset$. Traverse $\mathcal{X}$ and for each cluster C encountered, if C and C' are compatible for all $C' \in \mathcal{Y}$ then let $\mathcal{Y} := \mathcal{Y} \cup \{C\}$.

A greedy consensus tree of $\mathcal{S}$ is a tree T such that $\Lambda(T) = L$ and $\mathcal{C}(T) = \mathcal{Y}$.

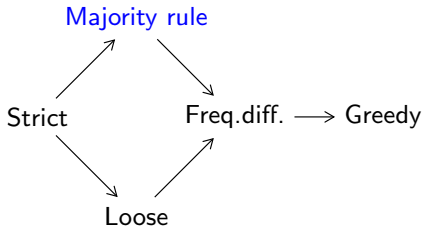
Example:



Some questions

- *How are these consensus trees related?*

  From the definitions, the following relationships hold:



  Here, a path $A \rightsquigarrow B$ means that any cluster in A is always a cluster in B.

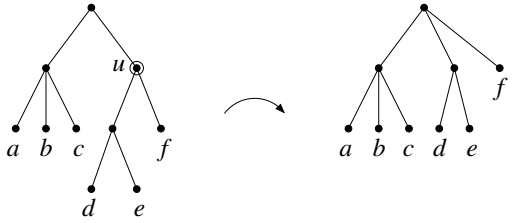
- *Which of them is used the most in practice?*

  The majority rule consensus tree is the most popular among biologists. According to Google Scholar, thousands of articles published in biology-related journals since the 1980s use it.

Preliminaries 1

The *delete* and *insert* operations on a tree:

- Let T be a tree and let u be any non-root, internal node in T. Applying the *delete* operation on u modifies T as follows: First, all children of u become children of the parent of u, and then u and the edge between u and its parent are removed.



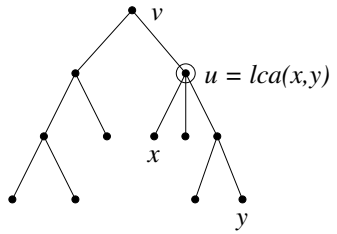
- Applying the *delete* operation on $u \Rightarrow$ The cluster $\Lambda(T[u])$ is removed from the cluster collection $\mathcal{C}(T)$ while all other clusters are preserved.
- Time for this operation: Proportional to the # of children of u.

The *insert* operation is the inverse of the *delete* operation.

Preliminaries 2

Definition

1. For any nodes u, v in a tree, if u is a descendant of v and $u \neq v$ then we write $u \prec v$ and call u a proper descendant of v.
2. Let x, y be nodes in a tree. The lowest common ancestor of x and y, denoted by $lca(x, y)$, is the unique node u such that both x and y are descendants of u and $u \prec v$ holds for every other node v which is an ancestor of both x and y.



(Straightforward generalization to $lca(C)$, where $C \subseteq \Lambda(T)$.)

Preliminaries 3

Day's algorithm:

- Takes two trees T_{ref} and T with with identical leaf label sets as input.
- After some preprocessing, the algorithm can check whether or not any specified cluster that occurs in T also occurs in T_{ref} efficiently.

Lemma 1 (Day; 1985)

Let T_{ref} and T be two given trees with $\Lambda(T_{ref}) = \Lambda(T) = L$ and $n = |L|$. After $O(n)$ time preprocessing, it is possible to determine, for any $u \in V(T)$, if $\Lambda(T[u]) \in \mathcal{C}(T_{ref})$ in $O(1)$ time.

Preliminaries 3, cont.

More precisely, the preprocessing in Day's algorithm works as follows:

- Do an $O(n)$-time depth-first traversal of T_{ref} while enumerating all the leaves as they are encountered.
  This yields a bijection f from L to the set $\{1, 2, \ldots, n\}$ under which every $C \in \mathcal{C}(T_{ref})$ forms an interval of consecutive integers.
- Assign each of the at most $n - 1$ intervals that represents a non-singleton cluster in $\mathcal{C}(T_{ref})$ to one of the n leaves in T_{ref} so that:
  (1) no leaf gets more than one interval; and
  (2) any interval $[a..b]$ is assigned to either the leaf $f^{-1}(a)$ or $f^{-1}(b)$.
  E.g., apply the rule:

  > *For each internal node u in T_{ref}, if u has no left sibling then assign u to the rightmost leaf descendant of u; otherwise, assign u to the leftmost leaf descendant of u.*

- Next, preprocess T in $O(n)$ time to store $f(x)$ in each leaf x of T. For all $u \in V(T)$, also compute $m(u) := \min_{x \in \Lambda(T[u])}\{f(x)\}$, $M(u) := \max_{x \in \Lambda(T[u])}\{f(x)\}$, and $size(u) := |\Lambda(T[u])|$.

Preliminaries 3, cont.

After the preprocessing is done, one can check for any specified internal node u in T if $\Lambda(T[u])$ occurs in T_{ref} in $O(1)$ time simply by checking:

- if $size(u) = M(u) - m(u) + 1$, i.e., if the interval $[m(u)..M(u)]$ is an interval of consecutive integers; and
- if either one of the two leaves $f^{-1}(m(u))$ and $f^{-1}(M(u))$ in T_{ref} was assigned the interval $[m(u)..M(u)]$.

Lemma 1 (Day; 1985)

Let T_{ref} and T be two given trees with $\Lambda(T_{ref}) = \Lambda(T) = L$ and $n = |L|$. After $O(n)$ time preprocessing, it is possible to determine, for any $u \in V(T)$, if $\Lambda(T[u]) \in \mathcal{C}(T_{ref})$ in $O(1)$ time.

Remark: This technique also gives an $O(kn)$-time algorithm for computing the strict consensus tree, which was defined by $\mathcal{C}(T) = \bigcap_{i=1}^{k} \mathcal{C}(T_i)$.

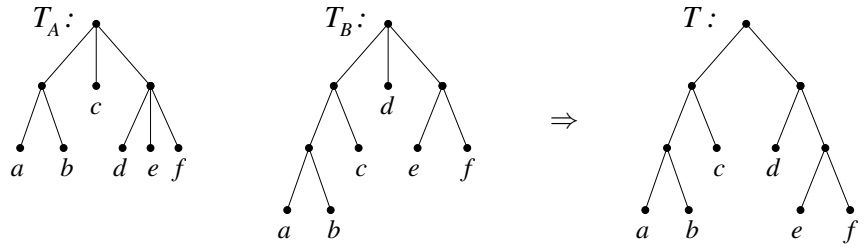
Preliminaries 4

Procedure `Merge_Trees`(T_A, T_B)**:**

- Combines all clusters from two compatible trees T_A, T_B into one tree.

Lemma 2

Let T_A and T_B be two given trees with $\Lambda(T_A) = \Lambda(T_B) = L$ that are compatible and $n = |L|$.
Procedure `Merge_Trees`(T_A, T_B) returns a tree T with $\Lambda(T) = L$ and $\mathcal{C}(T) = \mathcal{C}(T_A) \cup \mathcal{C}(T_B)$ in $O(n)$ time.



ab, *def* | *ab*, *abc*, *ef* | *ab*, *abc*, *def* , *ef*

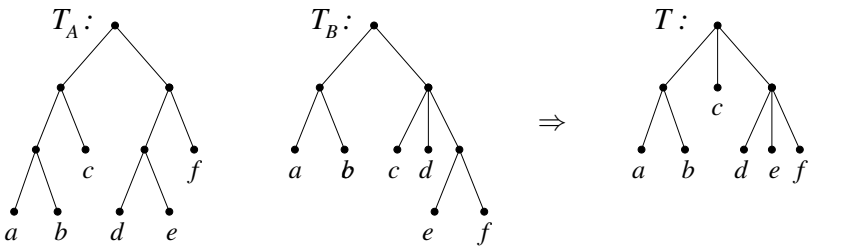
Preliminaries 4

Procedure `One-Way_Compatible`(T_A, T_B)**:**

- Output a copy of T_A in which every cluster that is not compatible with T_B has been removed. (In general, not symmetric.)

Lemma 3

Let T_A and T_B be two given trees with $\Lambda(T_A) = \Lambda(T_B) = L$ and $n = |L|$.
Procedure `One-Way_Compatible`(T_A, T_B) returns a tree T with $\Lambda(T) = L$ such that $\mathcal{C}(T) = \{C \in \mathcal{C}(T_A) : C \smile T_B\}$ in $O(n)$ time.



ab, *abc*, *de*, *def* | *ab*, *cdef* , *ef* | *ab*, *def*

Constructing the majority rule consensus tree

Recall:

- *majority cluster of* $\mathcal{S}$: Occurs in more than half of the trees in $\mathcal{S}$.
- The majority rule consensus tree of $\mathcal{S}$ is the tree T such that $\Lambda(T) = L$ and $\mathcal{C}(T)$ consists of all majority clusters of $\mathcal{S}$.

Our algorithm `Fast_Maj_Rule`:

- Inspired by the techique of Boyer and Moore (1991) for identifying a majority element (if one exists) in a list.
- Works in two phases.
  - Phase 1:
    Examine the input trees, one by one, to construct a set of candidate clusters that includes all majority clusters.
  - Phase 2:
    Remove all candidate clusters that are not majority clusters.

Algorithm `Fast_Maj_Rule`, Phase 1

- The current candidate clusters are stored as nodes in a tree T. (Importantly, we do not store any candidate clusters explicitly.)
- Every node v in T represents a current candidate cluster $\Lambda(T[v])$ and has a counter $count(v)$ that, from the iteration at which $\Lambda(T[v])$ became a candidate cluster, keeps track of the # of input trees in which it occurs minus the # of input trees in which it *doesn't* occur.
- While treating the tree T_j for any $j \in \{2, 3, \ldots, k\}$, $count(v)$ for each current candidate cluster $\Lambda(T[v])$ is updated:
  - If $\Lambda(T[v])$ occurs in T_j then $count(v)$ is incremented by 1; otherwise (i.e., if $\Lambda(T[v])$ does not occur in T_j), $count(v)$ is decremented by 1.
  - If any $count(v)$ reaches 0 then the node v is deleted from T so that $\Lambda(T[v])$ is no longer a current candidate cluster.
  - Next, every cluster occurring in T_j that is not a current candidate but compatible with T is inserted into T (thus becoming a current candidate cluster) and its counter is initialized to 1.

Algorithm Fast_Maj_Rule, Phase 1 & 2

We can prove the following:

If C is a majority cluster of $\mathcal{S}$ then $C \in \mathcal{C}(T)$ at the end of Phase 1.

Phase 2:

- Scan $\mathcal{S}$ one more time to compute the number of occurrences in $\mathcal{S}$ of every candidate cluster C.
- Remove any candidate cluster C in T that does not occur more than $\frac{k}{2}$ times.
- The clusters that remain in T are the majority clusters.

Algorithm Fast_Maj_Rule, pseudocode

Input: A set $\mathcal{S} = \{T_1, \ldots, T_k\}$ of trees with $\Lambda(T_1) = \cdots = \Lambda(T_k)$.
Output: The majority rule consensus tree of $\mathcal{S}$.

1: $T := T_1$ /* Start of Phase 1 */
2: **for** each $v \in V(T)$ **do** $count(v) := 1$
3: **for** $j := 2$ **to** k **do**
  for each $v \in V(T)$ **do**
    if $\Lambda(T[v])$ occurs in T_j **then** $count(v) := count(v) + 1$
    else $count(v) := count(v) - 1$
  for each $v \in V(T)$ in top-down order **do if** $count(v) = 0$ **then** *delete* v.
  for every $C \in \mathcal{C}(T_j)$ that is compatible with T but does not occur in T **do**
    Insert C into T; set $count(v) := 1$ for the new node v with $\Lambda(T[v]) = C$.
4: **for** each $v \in V(T)$ **do** $K(v) := 0$; /* Start of Phase 2 */
5: **for** $j := 1$ **to** k **do**
  for each $v \in V(T)$ **do**
    if $\Lambda(T[v])$ occurs in T_j **then** $K(v) := K(v) + 1$
6: **for** each $v \in V(T)$ in top-down order **do**
  if $K(v) \leq k/2$ **then** perform a *delete* operation on v.
7: **return** T

Algorithm Fast_Maj_Rule, time complexity

The above operations can be implemented efficiently by using:

- Day's algorithm (Lemma 1) to see if a current candidate cluster $\Lambda(T[v])$ occurs in the treated T_j in Phase 1, as well as to count occurrences of clusters in Phase 2.
- Procedures `Merge_Trees` (Lemma 2) and `One-Way_Compatible` (Lemma 3) to insert any clusters from T_j not currently in T but compatible with T in Phase 1.
- Top-down order for handling the *delete* operations so that every node in T is moved at most once per iteration.

⇒

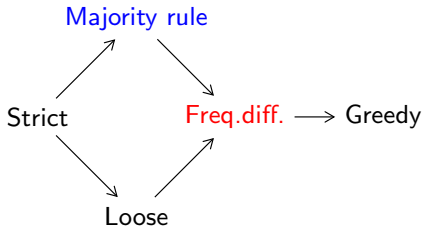
Theorem 1

Algorithm `Fast_Maj_Rule` constructs the majority rule consensus tree of $\mathcal{S}$ in $O(kn)$ time.

Some questions

- *How are these consensus trees related?*

  From the definitions, the following relationships hold:



  Here, a path $A \rightsquigarrow B$ means that any cluster in A is always a cluster in B.

- *Which of them is used the most in practice?*

  The majority rule consensus tree is the most popular among biologists. According to Google Scholar, thousands of articles published in biology-related journals since the 1980s use it.

  (The frequency difference consensus tree might be better, though...)

Constructing the frequency difference consensus tree

Recall:

- *frequency difference cluster of* $\mathcal{S}$: Occurs more often than each of the clusters that is incompatible with it.

  Written in a more formal way:
  - Let $\mathcal{S} = \{T_1, \ldots, T_k\}$ be a set of trees with $\Lambda(T_1) = \ldots = \Lambda(T_k) = L$.
  - For any cluster C of L, denote: $K_C(\mathcal{S}) = \{T_i : C \in \mathcal{C}(T_i)\}$ (i.e., the set of all trees in $\mathcal{S}$ in which C occurs).
  - If $|K_C(\mathcal{S})| > \max\{|K_D(\mathcal{S})| : D \subseteq L \text{ and } C \not\succ D\}$ then C is a *frequency difference cluster* of $\mathcal{S}$.

- The frequency difference consensus tree of $\mathcal{S}$ is the tree T such that $\Lambda(T) = L$ and $\mathcal{C}(T)$ consists of all frequency difference clusters of $\mathcal{S}$.

Can be computed naively by testing every cluster in $\mathcal{S}$ against all other clusters in $\mathcal{S}$ for compatibility. $\Rightarrow \Omega(k^2n^2)$ time

We can do it faster as follows.

Procedure `Filter_Clusters`

- For every $T_j \in \mathcal{S}$ and $u \in V(T_j)$, define the *weight of* u as $w(u) = |K_{\Lambda(T_j[u])}(\mathcal{S})|$.
  (I.e., the number of trees from $\mathcal{S}$ where the cluster $\Lambda(T_j[u])$ occurs.)
  For convenience, also define $w(C) = w(u)$, where $C = \Lambda(T_j[u])$.

- **Procedure** `Filter_Clusters`**:**
  - Takes as input two trees T_A, T_B with $\Lambda(T_A) = \Lambda(T_B) = L$ such that every cluster occurring in T_A or T_B also occurs somewhere in $\mathcal{S}$.
  - The output is a tree T with $\Lambda(T) = L$ such that $\mathcal{C}(T) = \{\Lambda(T_A[u]) : u \in V(T_A) \text{ and } w(u) > w(x) \text{ for every } x \in V(T_B) \text{ with } \Lambda(T_A[u]) \not\succ \Lambda(T_B[x])\}$.
    (I.e., a copy of T_A in which every cluster that is incompatible with some cluster in T_B with a higher weight has been removed.)

Forward frequency consensus trees

- Let $\mathcal{C}(S)$ for any set S of trees denote $\bigcup_{T_i \in S} \mathcal{C}(T_i)$.
- For any $j \in \{1, \ldots, k\}$, define a *forward frequency difference consensus tree of* $\{T_1, T_2, \ldots, T_j\}$ as any tree that includes every cluster C in $\mathcal{C}(\{T_1, T_2, \ldots, T_j\})$ satisfying $w(C) > w(X)$ for all $X \in \mathcal{C}(\{T_1, T_2, \ldots, T_j\})$ with $C \not\succ X$.

To compute forward frequency difference consensus trees:

Lemma 4

For any $j \in \{2, 3, \ldots, k\}$, suppose that T is a forward frequency difference consensus tree of $\{T_1, T_2, \ldots, T_{j-1}\}$. Let $A :=$ `Filter_Clusters`(T, T_j) and $B :=$ `Filter_Clusters`(T_j, T). Then `Merge_Trees`(A, B) is a forward frequency difference consensus tree of $\{T_1, T_2, \ldots, T_j\}$.

- So, by repeatedly using `Filter_Clusters` & `Merge_Trees`, we end up with a forward frequency difference consensus tree T of $\{T_1, \ldots, T_k\}$.
- $\mathcal{C}(T)$ contains all frequency difference clusters of $\mathcal{S}$ but possibly some other clusters as well, so we apply `Filter_Clusters` again.

Algorithm `Fast_Frequency_Difference`, pseudocode

Input: A set $\mathcal{S} = \{T_1, \ldots, T_k\}$ of trees with $\Lambda(T_1) = \cdots = \Lambda(T_k)$.
Output: The frequency difference consensus tree of $\mathcal{S}$.

1: Compute $w(C)$ for every cluster C occurring in $\mathcal{S}$.
2: $T := T_1$
3: **for** $j := 2$ **to** k **do**
   $A :=$ `Filter_Clusters`(T, T_j); $B :=$ `Filter_Clusters`(T_j, T)
   $T :=$ `Merge_Trees`(A, B)
4: **for** $j := 1$ **to** k **do**
   $T :=$ `Filter_Clusters`(T, T_j)
5: **return** T

Algorithm Fast_Frequency_Difference, time complexity

Time complexity analysis:

- Step 1 takes $O(kn \log n)$ time by divide-and-conquer + counting sort.
- Every execution of Merge_Trees takes $O(n)$ time.
- Assume that every execution of Filter_Clusters takes $f(n)$ time.
  $\Rightarrow$ Step 3 takes $O(k \cdot f(n))$ time, and Step 4 takes $O(k \cdot f(n))$ time.
  - $f(n) = O(n^2)$ is relatively easy. $\Rightarrow$ Total running time: $O(kn^2)$
  - $f(n) = O(n \log^2 n)$ is possible by the centroid path decomposition technique to break the cluster collection of T_A into smaller sets that can be checked more easily and then put together again at the end.
  - The above can be refined to $f(n) = O(n \log n)$ by interpreting clusters as intervals and solving instances of the Max-Manhattan Skyline Problem to find which clusters to remove at each stage.

Theorem 2

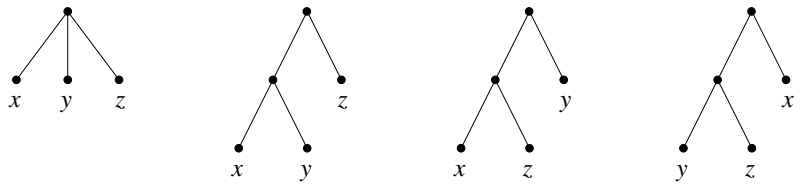
Algorithm Fast_Frequency_Difference constructs the frequency difference consensus tree of $\mathcal{S}$ in $O(kn \log n)$ time.

PART III : Rooted Triplet-Based Consensus Trees

Rooted triplets

A phylogenetic tree with exactly three leaves is called a rooted triplet.

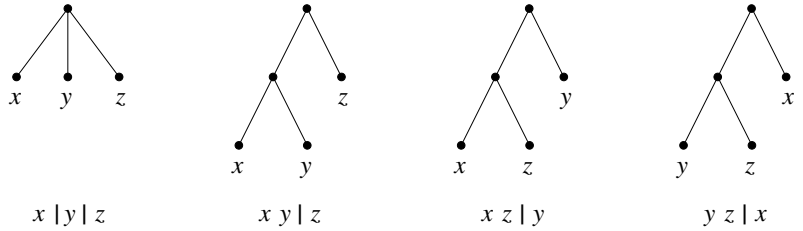
Let $\{x, y, z\}$ be a leaf label set of cardinality 3.
There are four possible phylogenetic trees leaf-labeled by $\{x, y, z\}$:



Two kinds of rooted triplets:

- *Fan triplet* = One internal node $(x|y|z)$
- *Resolved triplet* = Two internal nodes $(xy|z,\ xz|y$, and $yz|x)$

Rooted triplet-based consensus trees



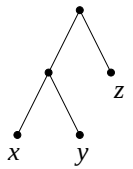
Intuition: Smallest unit of branching information... Useful concept because any phylogenetic tree can be represented by a set of rooted triplets.

This part of the talk is about three consensus trees related to rooted triplets:

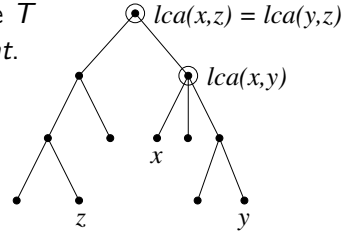
1. Local consensus tree
   [Kannan, Warnow, Yooseph; 1998]
2. R* consensus tree
   [Bryant; 2003]
3. Adams consensus tree
   [Adams; 1972]

Local consensus tree, notation

1. $xy|z$ = The unique resolved triplet with $lca(x,y) \prec lca(x,z) = lca(y,z)$.



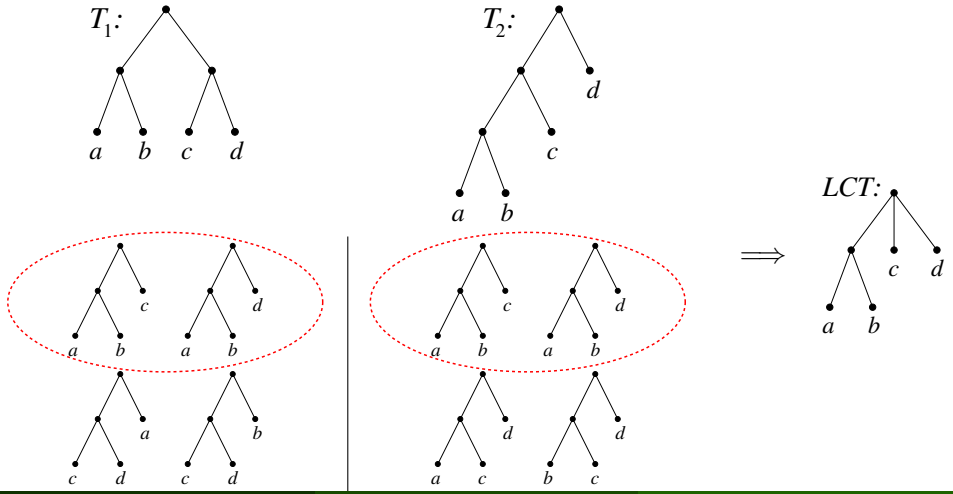
2. If $lca(x,y) \prec lca(x,z) = lca(y,z)$ in a tree T then we say that T and $xy|z$ *are consistent*.



3. $r(T)$ = the set of all resolved triplets consistent with the tree T
4. A set $\mathcal{R}$ of resolved triplets is consistent if $\exists T$ such that $\mathcal{R} \subseteq r(T)$.

Local consensus tree, main idea

1. Represent every input tree T_i by its set of resolved triplets $r(T_i)$.
2. Compute the intersection of the resolved triplet-sets.
3. Construct a smallest tree that contains at least these resolved triplets.



Local consensus tree, problem definitions

The minimally resolved local consensus tree problem (MinRLC):

Input: A set $\mathcal{S} = \{T_1, T_2, \ldots, T_k\}$ of trees, each with the same leaf label set L.

Output: A tree T with leaves labeled by L satisfying $\bigcap_{i=1}^{k} r(T_i) \subseteq r(T)$ with as few internal nodes as possible.

The minimally rooted-triplet-inducing local consensus tree problem (MinILC):

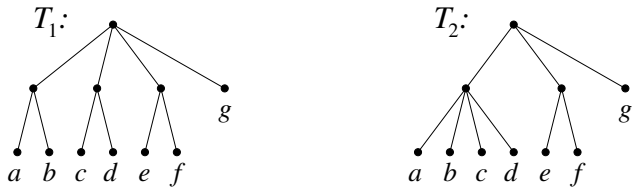
Input: A set $\mathcal{S} = \{T_1, T_2, \ldots, T_k\}$ of trees, each with the same leaf label set L.

Output: A tree T with leaves labeled by L satisfying $\bigcap_{i=1}^{k} r(T_i) \subseteq r(T)$ that minimizes the value $|r(T)|$.

Remark: If we just want $\bigcap_{i=1}^{k} r(T_i) \subseteq r(T)$ then outputting T_1 would do.

"Minimal" $\Rightarrow$ simpler overview, more compact, avoids false groupings

MinRLC and MinILC are not always the same. Example:



$r(T_1) \cap r(T_2) =$
$\{ab|e, ab|f, ab|g, cd|e, cd|f, cd|g, ef|a, ef|b, ef|c, ef|d, ef|g\}$

- T_2 is an optimal solution to MinRLC.
- On the other hand, $|r(T_1)| = 15$ while $|r(T_2)| = 23$, so T_2 cannot be an optimal solution to MinILC.

Local consensus tree, previous work

- MinRLC: The closely related (sub-)problem in which the input is a consistent set $\mathcal{R}$ of rooted triplets and the output is a tree containing all of $\mathcal{R}$ having the minimum number of nodes was studied previously in [Jansson, Lemence, Lingas; *SIAM Journal on Computing*; 2012].
- MinILC and some other "local consensus trees" were introduced by Kannan, Warnow, and Yooseph [*SIAM Journal on Computing*; 1998]. (RV-II tree = "relaxed version II" tree)

Kannan *et al.* claimed that applying the BUILD algorithm [Aho, Sagiv, Szymanski, Ullman; *SIAM Journal on Computing*; 1981] to $\mathcal{R} = \bigcap_{i=1}^{k} r(T_i)$ produces a minimally rooted-triplet-inducing local consensus tree.

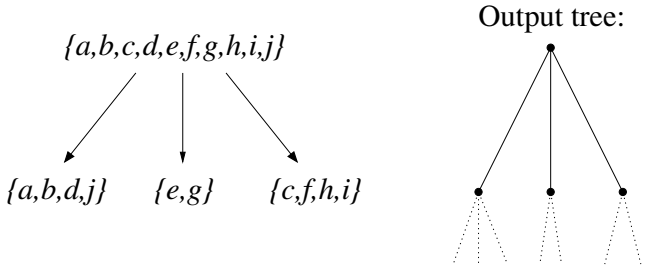
This would imply that MinILC is solvable in polynomial time.

Unfortunately, the claim is not correct.

Algorithm BUILD by Aho, Sagiv, Szymanski, Ullman [1981]

Top-down, recursive algorithm for constructing a tree consistent with an input set $\mathcal{R}$ of resolved triplets, or determining that no such tree exists. Runs in polynomial time.

Strategy:
Partition L into blocks according to $\mathcal{R}$. Output a tree consisting of a root whose children are roots of the trees obtained by recursing on each block.

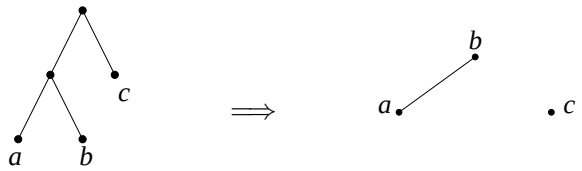


(Base case of the recursion: $|L'| = 1$)

Algorithm BUILD, cont.

Use "auxiliary graph" $\mathcal{G}(L)$ to find the partition into blocks.
For any $L' \subseteq L$, define $\mathcal{G}(L') = (L', E)$, where E contains edge $\{x, y\}$ iff there is some $xy|z$ in $\mathcal{R}$ with $x, y, z \in L'$.

Example:



The rooted triplet $ab|c$ in $\mathcal{R}$ gives edge $\{a, b\}$ in $\mathcal{G}(L)$.

Crucial observation: If $ab|c$ is consistent with a tree T then the leaves labeled by a and b *cannot* descend from two different children of the root of T, i.e., a and b must belong to the same block.

Therefore, the algorithm defines the partition of L by:

Blocks of leaves $\longleftrightarrow$ connected components in $\mathcal{G}(L)$

Algorithm BUILD, cont.

How can the algorithm detect conflicts?

Lemma [Aho, Sagiv, Szymanski, Ullman; 1981]

$\mathcal{R}$ is not consistent with any phylogenetic tree if and only if some $\mathcal{G}(L')$ has only one connected component and $|L'| > 1$.

Algorithm BUILD, pseudocode

Input: A set $\mathcal{R}$ of resolved triplets and a leaf label set L.
Output: A phylogenetic tree with leaves labeled by L that is consistent with all resolved triplets in $\mathcal{R}$, if one exists; otherwise, *null*.

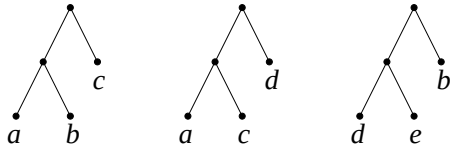
1: Construct the auxiliary graph $\mathcal{G}(L)$.
2: Compute the connected components $C_1, C_2, \ldots, C_s$ of $\mathcal{G}(L)$.
3:
- If $s = 1$ and $\mathcal{G}(L)$ consists of exactly one vertex i then let T be a tree with a single leaf labeled by i.
- If $s = 1$ and $\mathcal{G}(L)$ contains > 1 vertex then $T :=$ *null*.
- Otherwise, for $i \in \{1, 2, \ldots, s\}$, build tree $T_i :=$ BUILD$(\mathcal{R}|V(C_i), V(C_i))$. If every $T_i \neq$ *null* then attach all of these trees to a common parent node and let T be the resulting tree; else $T :=$ *null*.

4: **return** T.

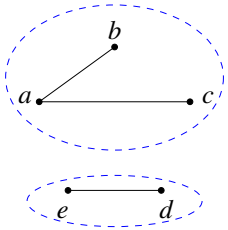
(Here, $V(C_i) =$ the set of vertices in C_i and
$\mathcal{R}|V(C_i) =$ all rooted triplets in $\mathcal{R}$ whose leaves belong to $V(C_i)$ only.)

Algorithm BUILD, example 1 (no conflicts)

$\mathcal{R}$:



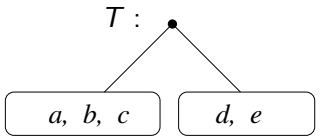
Construct the auxiliary graph $\mathcal{G}(L)$:



The connected components' vertex sets are:
$V(C_1) = \{a, b, c\}$,
$V(C_2) = \{d, e\}$

Algorithm BUILD, example 1 (no conflicts), cont.

So far, we know that the tree must look like this:



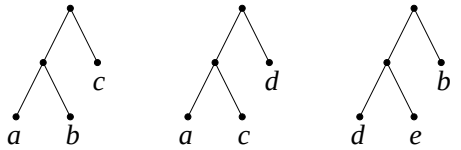
Next, recurse on C_1 and C_2.

For C_1, consider input resolved triplets that involve $\{a, b, c\}$ only.

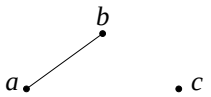
For C_2, consider input resolved triplets that involve $\{d, e\}$ only.

Algorithm BUILD, example 1 (no conflicts), cont.

$\mathcal{R}$:



$\mathcal{G}(\{a, b, c\})$:



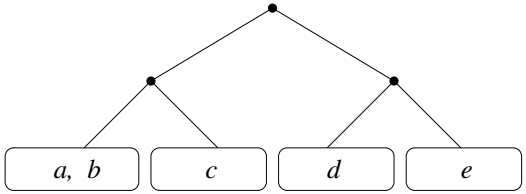
$V(C_{1,1}) = \{a, b\}$,
$V(C_{1,2}) = \{c\}$

$\mathcal{G}(\{d, e\})$:

$V(C_{2,1}) = \{d\}$,
$V(C_{2,2}) = \{e\}$

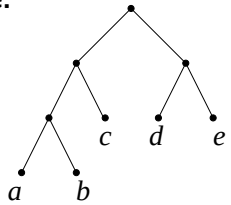
Algorithm BUILD, example 1 (no conflicts), cont.

This yields:



⋮

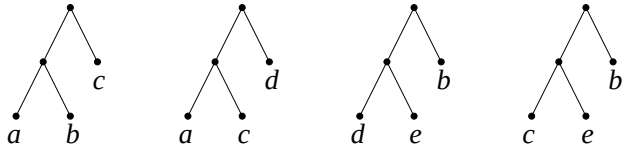
Continue $\Longrightarrow$ **Final tree:**



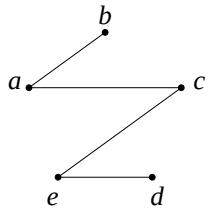
□

Algorithm BUILD, example 2 (conflict)

$\mathcal{R}$:



$\mathcal{G}(L)$:

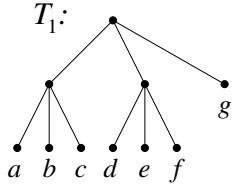


Here, $\mathcal{G}(L)$ has exactly one connected component and $|L| > 1$.
$\Rightarrow$ No tree is consistent with $\mathcal{R}$. □

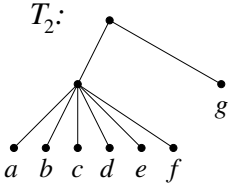
BUILD does not solve MinILC

Based on an observation by Bryant [1997], we have the following counterexample to Kannan *et al.*'s claim that BUILD produces a minimally rooted-triplet-inducing local consensus tree:

T_1:

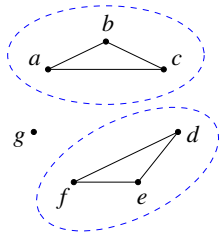


T_2:



Then $\mathcal{R} = r(T_1) \cap r(T_2) = \{ab|g, ac|g, bc|g, de|g, df|g, ef|g\}$

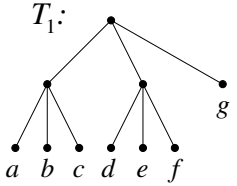
$\mathcal{G}(L)$:



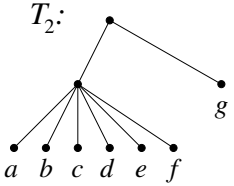
BUILD does not solve MinILC

Based on an observation by Bryant [1997], we have the following counterexample to Kannan *et al.*'s claim that BUILD produces a minimally rooted-triplet-inducing local consensus tree:

T_1:



T_2:



Then $\mathcal{R} = r(T_1) \cap r(T_2) = \{ab|g, ac|g, bc|g, de|g, df|g, ef|g\}$ and the output of BUILD($\mathcal{R}$) is T_1.

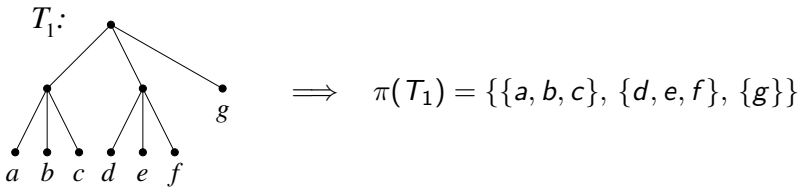
However, $|r(T_1)| = 24$ and $|r(T_2)| = 15$ so T_1 is not an optimal solution to MinILC.

(The same example also shows that BUILD does not solve MinRLC.)

Exponential-time algorithms for MinRLC & MinILC

Instead, we can use dynamic programming together with Semple's lemma.

Notation: For any tree T, $\pi(T)$ is the partition of T's leaf label set according to the subtrees attached to the root of T.



$\Longrightarrow \quad \pi(T_1) = \{\{a, b, c\}, \{d, e, f\}, \{g\}\}$

The next lemma relates the auxiliary graph $\mathcal{G}(L)$ used in the BUILD algorithm to *every* tree consistent with $\mathcal{R}$:

Lemma 5 [Semple; 2003]

Let T be any tree that is consistent with $\mathcal{R}$. For each connected component C in $\mathcal{G}(L)$, $\Lambda(C) \subseteq B$ for some $B \in \pi(T)$, where $\Lambda(C) =$ the leaf labels in C.

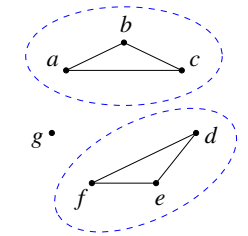
Local consensus tree, applying Semple's lemma

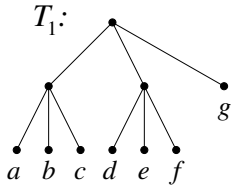
Lemma 5 [Semple; 2003]

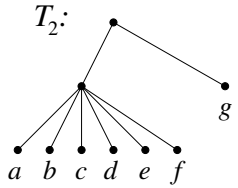
Let T be any tree that is consistent with $\mathcal{R}$. For each connected component C in $\mathcal{G}(L)$, $\Lambda(C) \subseteq B$ for some $B \in \pi(T)$, where $\Lambda(C) =$ the leaf labels in C.

Example:
$\mathcal{R} = \{ab|g,\ ac|g,\ bc|g,\ de|g,\ df|g,\ ef|g\}$

$\mathcal{G}(L)$:







Local consensus tree, applying Semple's lemma

Lemma 5 [Semple; 2003]

Let T be any tree that is consistent with $\mathcal{R}$. For each connected component C in $\mathcal{G}(L)$, $\Lambda(C) \subseteq B$ for some $B \in \pi(T)$, where $\Lambda(C) =$ the leaf labels in C.

Importantly, if T is consistent with $\mathcal{R}$ then $\pi(T)$ can be obtained by performing zero of more mergings of $\mathcal{G}(L)$'s connected components.

$\Rightarrow$ Every tree consistent with $\mathcal{R}$ can be recovered by trying all possible mergings of the connected components in $\mathcal{G}(L)$ at each recursion level.

Algorithm for MinRLC

First construct $\mathcal{R} = \bigcap_{i=1}^{k} r(T_i)$ in $O(kn^3)$ time.

Then:
For all $L' \subseteq L$ in order of increasing cardinality, use Lemma 6 to compute $opt(L') = \#$ of internal nodes in an optimal solution for $\mathcal{R}$ restricted to L'.

Lemma 6

For any $L' \subseteq L$ with $|L'| \geq 2$, it holds that $opt(L') = DP(\mathcal{C}_{L'}) + 1$.

where:

- $\mathcal{C}_{L'}$ is the set of connected components in $\mathcal{G}(L')$.
- $DP(\mathcal{D})$ for every $\mathcal{D} \subseteq \mathcal{C}_{L'}$ is the minimum value of $\sum_{\mathcal{X} \in \mathcal{Q}} opt(Merge(\mathcal{X}))$ taken over all true partitions $\mathcal{Q}$ of $\mathcal{D}$.
- $Merge(\mathcal{D}) =$ the set of all leaf labels belonging to components in $\mathcal{D}$.

Also use dynamic programming over the subsets $\mathcal{D}$ of $\mathcal{C}_{L'}$ in an inner loop to get the $DP(\mathcal{D})$-values for each fixed L'.

Finally, do a traceback to retrieve a corresponding optimal solution.

Algorithm for MinRLC, pseudocode

Algorithm `MinRS_exact`

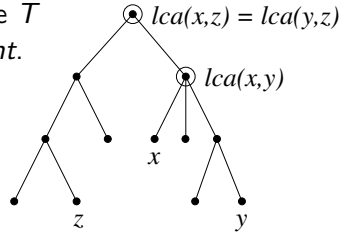
Input: A consistent set $\mathcal{R}$ of rooted triplets over a leaf label set L.
Output: The number of internal nodes in a minimally resolved supertree consistent with $\mathcal{R}$ and leaf-labeled by L.

For every $x \in L$, initialize $opt(\{x\}) := 0$;
for $i := 2$ to n **do**
  for every cardinality-i subset L' of L **do**
    Construct $\mathcal{G}(L')$. Let $\mathcal{C}$ and $\mathcal{U}$ be the set of connected components and the set of singleton components, respectively, in $\mathcal{G}(L')$;
    Let $DP(\emptyset) := 0$. For every $X \in \mathcal{C} \setminus \mathcal{U}$, let $DP(\{X\}) := opt(\Lambda(X))$;
    for $j := 2$ to $|\mathcal{C}| - |\mathcal{U}|$ **do**
      for every cardinality-j subset $\mathcal{D}$ of $\mathcal{C} \setminus \mathcal{U}$ **do**
        $DP(\mathcal{D}) := \min_{\emptyset \neq \mathcal{X} \subsetneq \mathcal{D}} \{opt(\bigcup_{Q \in \mathcal{X}} \Lambda(Q)) + \min\{DP(\mathcal{D} \setminus \mathcal{X}), opt(\bigcup_{Q \in \mathcal{D} \setminus \mathcal{X}} \Lambda(Q))\}\}$;
      end for
    end for
    $opt(L') := DP(\mathcal{C} \setminus \mathcal{U}) + 1$;
  end for
end for
return $opt(L)$;

Figure: Algorithm `MinRS_exact`.

R* consensus tree, notation

- If $lca(x,y) \prec lca(x,z) = lca(y,z)$ in a tree T then we say that T and $xy|z$ *are consistent*.



- On the other hand, if $lca(x,y) = lca(x,z) = lca(y,z)$ in T then T and the fan triplet $x|y|z$ are *consistent*.
- $r(T)$ = the set of *resolved* triplets consistent with T
- $t(T)$ = the set of *all* triplets consistent with T (Thus, $r(T) \subseteq t(T)$. When T is binary, $r(T) = t(T)$.)
- $\Lambda(T)$ = the set of leaves in T

Local consensus tree, summary

Theorem 3

- MinRLC is solvable in $O(kn^3 + 2.733^n)$ time.
- MinILC is solvable in $O(kn^3 + 4^n \cdot poly(n))$ time.

where
$k = |\mathcal{S}|$ is the number of input trees, $n = |L|$ is the number of leaf labels.

But why not polynomial time? Actually:

Theorem 4

- MinRLC is NP-hard.
- MinILC is NP-hard.

Remark: To prove the NP-hardness, reduce from Chromatic Number and Clique, respectively.

R* consensus tree, problem definition

Let $\mathcal{S} = \{T_1, T_2, \ldots, T_k\}$ be a given set of trees with $\Lambda(T_1) = \Lambda(T_2) = \ldots = \Lambda(T_k) = L$.

- For any $\{a,b,c\} \subseteq L$, define $\#ab|c$ as the number of trees T_i for which $ab|c \in t(T_i)$. (Here, $0 \leq \#ab|c \leq k$.)
- The set of majority resolved triplets is defined as $\mathcal{R}_{maj} = \left\{ab|c : a,b,c \in L \text{ and } \#ab|c > \max\{\#ac|b, \#bc|a\}\right\}$.

Definition

The R* consensus tree of $\mathcal{S}$ is the tree τ with $\Lambda(\tau) = L$ that satisfies $r(\tau) \subseteq \mathcal{R}_{maj}$ and that maximizes the number of internal nodes.

Remark 1: The local consensus tree from before asked for a tree T such that $\bigcap_{i=1}^{k} r(T_i) \subseteq r(T)$, but now we want a τ such that $r(\tau) \subseteq \mathcal{R}_{maj}$.

Remark 2: Also observe that $\mathcal{R}_{maj}$ doesn't have to be consistent.

Lemma 9 [Bryant; 2003] The R* consensus tree exists and is unique.

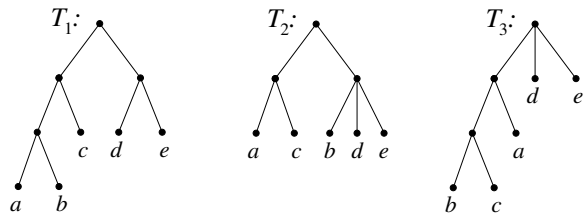
R* consensus tree, examples

Definition

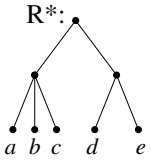
The R* consensus tree of $\mathcal{S}$ is the tree τ with $\Lambda(\tau) = L$ that satisfies $r(\tau) \subseteq \mathcal{R}_{maj}$ and that maximizes the number of internal nodes.

(Every rooted triplet consistent with τ must also belong to $\mathcal{R}_{maj}$.)

Example 1:



Then
$\mathcal{R}_{maj} = \{ab|d,\ ab|e,\ ac|d,\ ac|e,\ de|a,\ bc|d,\ bc|e,\ de|b,\ de|c\}$,
and the R* consensus tree is:



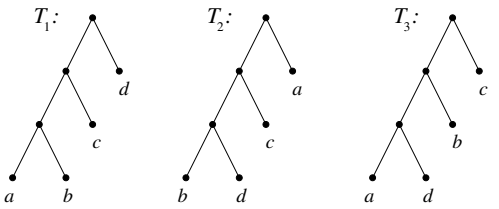
R* consensus tree, examples

Definition

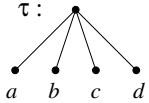
The R* consensus tree of $\mathcal{S}$ is the tree τ with $\Lambda(\tau) = L$ that satisfies $r(\tau) \subseteq \mathcal{R}_{maj}$ and that maximizes the number of internal nodes.

(Every rooted triplet consistent with τ must also belong to $\mathcal{R}_{maj}$.)

Example 2:



Then $\mathcal{R}_{maj} = \{ab|c,\ bd|c\}$, and the R* consensus tree is:



Note that $r(\tau) = \emptyset$. Any other tree with leaf set $\{a, b, c, d\}$ would introduce additional resolved triplets not in $\mathcal{R}_{maj}$!

R* consensus tree, motivation

The R* consensus tree provides a statistically consistent estimator of the species tree topology when combining a set of gene trees:

- J.H. Degnan, M. DeGiorgio, D. Bryant, N.A. Rosenberg: "Properties of consensus methods for inferring species trees from gene trees", *Systematic Biology*, Vol. 58, pp. 35–54, 2009.

(In their study, R* consensus outperformed other popular consensus methods such as *majority rule consensus*.)

From an algorithmic point of view, the R* consensus tree is also interesting because it generalizes the *RV-III tree* from:

- S. Kannan, T. Warnow, S. Yooseph: "Computing the local consensus of trees", *SIAM Journal on Computing*, Vol. 27, pp. 1695–1724, 1998.

to more than two input trees.

R* consensus tree, previous results

$k = |\mathcal{S}|$ be the number of input trees; $n = |L|$ the number of leaf labels

- For $k = 2$, the R* consensus tree can be computed in $O(n^3)$ time [Kannan, Warnow, Yooseph; *SIAM Journal on Computing*, 1998].
- For unbounded k, the R* consensus tree can be computed in $O(kn^3)$ time [Bryant; Vol. 61 of DIMACS Series in DMTCS, 2003].

Remark:
$|\mathcal{R}_{maj}| = \Omega(n^3)$ when the trees have similar branching structures.
$\Rightarrow$ To obtain a faster algorithm, we have to avoid explicitly constructing the set $\mathcal{R}_{maj}$.

Algorithm R*_consensus_tree, preliminaries

Let $\mathcal{R}$ be a set of triplets over a leaf label set $L = \bigcup_{t \in \mathcal{R}} \Lambda(t)$ s.t. for each $\{x, y, z\} \subseteq L$, at most one of $x|yz$, $xy|z$, $xz|y$, and $yz|x$ belongs to $\mathcal{R}$.

- A cluster of L is any subset of L.

Two special types of clusters:

- A cluster A of L is called a strong cluster of $\mathcal{R}$ if $aa'|x \in \mathcal{R}$ for all $a, a' \in A$ with $a \neq a'$ and all $x \in L \setminus A$. Furthermore, L as well as every singleton set of L is also defined to be a strong cluster of $\mathcal{R}$.
- For each $a, b \in L$ with $a \neq b$, define $s_{\mathcal{R}}(a, b) = |\{w : ab|w \in \mathcal{R}\}|$. For each $a \in L$, define $s_{\mathcal{R}}(a, a) = |L| - 1$. A cluster A of L is called an Apresjan cluster of $s_{\mathcal{R}}$ if $s_{\mathcal{R}}(a, a') > s_{\mathcal{R}}(a, x)$ for all $a, a' \in A$ and all $x \in L \setminus A$.

Lemma 10

Every strong cluster of $\mathcal{R}$ is an Apresjan cluster of $s_{\mathcal{R}}$.

Proof sketch: Let C be any strong cluster. Then for every $a, a' \in C$ and $x \in L \setminus C$, it holds that $s_{\mathcal{R}}(a, a') \geq |L \setminus C|$ and $s_{\mathcal{R}}(a, x) < |L \setminus C|$. □

- **Strong clusters** are useful because:
  The strong clusters of $\mathcal{R}_{maj}$ determine the R* consensus tree:

Lemma 11 [Bryant; 2003]

The R* consensus tree includes every strong cluster of $\mathcal{R}_{maj}$ and no other clusters.

- **Apresjan clusters** are useful because:
  By Lemma 10, the set of strong clusters of $\mathcal{R}_{maj}$ is contained within the set of Apresjan clusters of $s_{\mathcal{R}_{maj}}$.

  And: There are only $O(n)$ Apresjan clusters of $s_{\mathcal{R}_{maj}}$.

  And: If we know all values of $s_{\mathcal{R}_{maj}}(a, b)$, the Apresjan clusters of $s_{\mathcal{R}_{maj}}$ can be computed quickly.
  (Use a fast algorithm by Bryant & Berry [2001].)

Algorithm R*_consensus_tree, pseudocode

Algorithm R*_consensus_tree
Input: A set $\mathcal{S} = \{T_1, \ldots, T_k\}$ of trees with $\Lambda(T_1) = \ldots = \Lambda(T_k) = L$
Output: The R* consensus tree of $\mathcal{S}$

1: Compute and store $s_{\mathcal{R}_{maj}}(a, b)$ for all $a, b \in L$.
2: Compute the Apresjan clusters of $s_{\mathcal{R}_{maj}}$.
3: **for** each Apresjan cluster A of $s_{\mathcal{R}_{maj}}$ **do**
4: Determine if A is a strong cluster of $\mathcal{R}_{maj}$.
5: **end for**
6: Construct the R* consensus tree using all the strong clusters of $\mathcal{R}_{maj}$.

Time complexity? For any k:

- Step 2: Apply an $O(n^2)$-time algorithm by Bryant & Berry [2001].
- Step 6: Can be done in $O(n^2)$ time by Gusfield's algorithm for the "perfect phylogeny problem with binary characters" [1991].
- The time complexity of the other steps depends on k.

R* consensus tree, summary

Theorem 5

Let $\mathcal{S}$ be an input set of k trees with n leaves each and identical leaf label sets. The R* consensus tree of $\mathcal{S}$ can be computed in:

- $O(n^2)$ time when $k = 2$;
- $O(n^2 \log^{4/3} n)$ time when $k = 3$; and
- $\min\{O(n^2 \log^{k+2} n),\ O(kn^3)\}$ time when k is unbounded.

Adams consensus tree, notation

Let T be a phylogenetic tree.

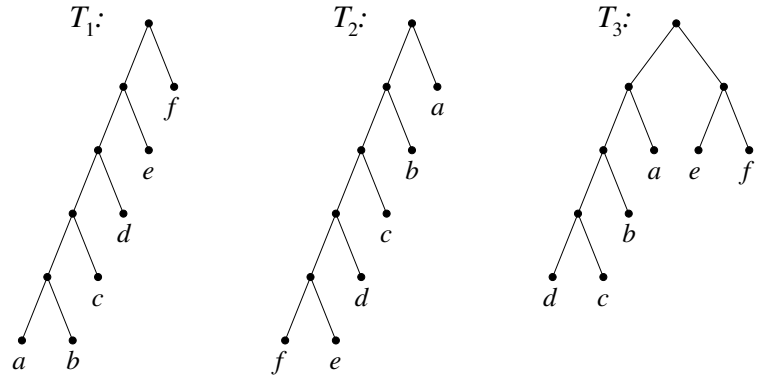
- $V(T)$ = the set of all nodes in T
- $\Lambda(T)$ = the set of all leaf labels in T
- For every $u \in V(T)$, define $T[u]$ = the subtree of T rooted at u (= the subgraph of T induced by u and all of u's proper descendants).
- $\pi(T) = \{\Lambda(T[c]) : c \in \mathit{Child}(r)$, where r is the root of $T\}$. Observe that $\pi(T)$ is a partition of $\Lambda(T)$.

Next, let $\mathcal{S} = \{T_1, T_2, \ldots, T_k\}$ be a set of phylogenetic trees with $\Lambda(T_1) = \Lambda(T_2) = \ldots = \Lambda(T_k) = L$ for a leaf label set L.

- $\pi(\mathcal{S})$ = the product of the partitions $\pi(T_1), \pi(T_2), \ldots, \pi(T_k)$

I.e., every part in $\pi(\mathcal{S})$ is of the form $\bigcap_{j=1}^{k} \Lambda(T_j[c_j])$ for some child c_j of the root of T_j.

Example: $\mathcal{S} = \{T_1, T_2, T_3\}$ and $\Lambda(T_1) = \Lambda(T_2) = \Lambda(T_3) = \{a, b, c, d, e, f\}$

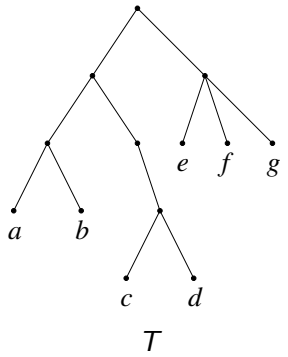


Then:
$\pi(T_1) = \big\{\{a, b, c, d, e\}, \{f\}\big\}$,
$\pi(T_2) = \big\{\{a\}, \{b, c, d, e, f\}\big\}$,
$\pi(T_3) = \big\{\{a, b, c, d\}, \{e, f\}\big\}$,
and $\pi(\mathcal{S}) = \big\{\{a\}, \{b, c, d\}, \{e\}, \{f\}\big\}$.

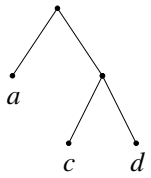
Adams consensus tree, restriction of trees

Let T be a phylogenetic tree and $B \subseteq \Lambda(T)$.

- The *restriction of T to B*, denoted by $T|B$, is the tree T' with leaf label set B and node set $\{lca^T(\{u, v\}) : u, v \in B\}$ that preserves the ancestor relations from T, i.e., that satisfies $lca^T(B') = lca^{T'}(B')$ for all nonempty $B' \subseteq B$.



T



$T \,|\, \{a, c, d\}$

Adams consensus tree, definition

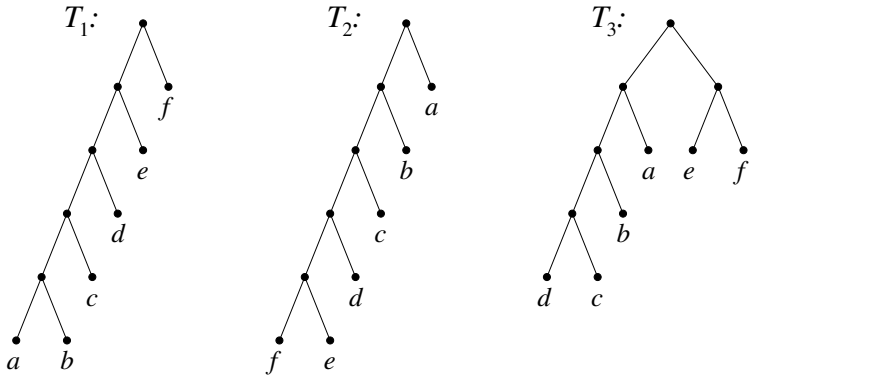
Let $\mathcal{S} = \{T_1, T_2, \ldots, T_k\}$ be any set of trees satisfying $\Lambda(T_1) = \Lambda(T_2) = \ldots = \Lambda(T_k)$.

The Adams consensus tree of $\mathcal{S}$ is the output of the following algorithm [Adams; 1972]:

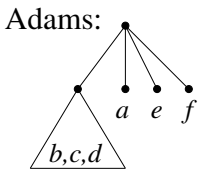
Algorithm `Old_Adams_consensus`
Input: Set $\mathcal{S} = \{T_1, T_2, \ldots, T_k\}$ of trees with $\Lambda(T_1) = \Lambda(T_2) = \ldots = \Lambda(T_k)$.

1: **if** T_1 has only one leaf **then** let $T := T_1$; /* Base case of the recursion */
2: **else** /* General case of the recursion */
3: $\pi := \pi(\mathcal{S})$;
4: **for** $B \in \pi$ **do** $T_B :=$ `Old_Adams_consensus`$(\{T_1|B, T_2|B, \ldots, T_k|B\})$;
5: Create tree T whose root is the parent of the root of T_B for every $B \in \pi$;
6: **end if**
7: **return** T;

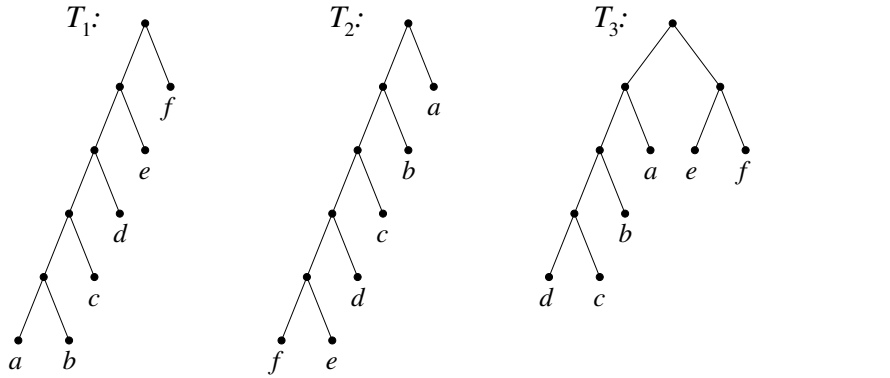
Example: $\mathcal{S} = \{T_1, T_2, T_3\}$ and $\Lambda(T_1) = \Lambda(T_2) = \Lambda(T_3) = \{a, b, c, d, e, f\}$



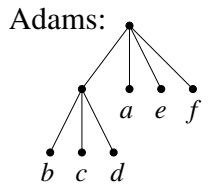
From before: $\pi(\mathcal{S}) = \{\{a\}, \{b, c, d\}, \{e\}, \{f\}\}$ $\Longrightarrow$



Example: $\mathcal{S} = \{T_1, T_2, T_3\}$ and $\Lambda(T_1) = \Lambda(T_2) = \Lambda(T_3) = \{a, b, c, d, e, f\}$



From before: $\pi(\mathcal{S}) = \{\{a\}, \{b, c, d\}, \{e\}, \{f\}\}$ $\Longrightarrow$



CPM 2024 Summer School Phylogenetic Consensus Trees 110 / 132

Adams consensus tree, alternative definition

Remark:
Equivalently, the Adams consensus tree of $\mathcal{S}$ can be defined as the unique tree T with $\Lambda(T) = L$ for which the following two properties hold:

- For any $A, B \subseteq L$, if $lca^{T_j}(A) \prec lca^{T_j}(B)$ in every $T_j \in \mathcal{S}$ then $lca^{T}(A) \prec lca^{T}(B)$.
- For any $u, v \in V(T)$, if $u \prec v$ in T then $lca^{T_j}(\Lambda(T[u])) \prec lca^{T_j}(\Lambda(T[v]))$ in every $T_j \in \mathcal{S}$.

(Proved by Adams in 1986.)

In this sense, the Adams consensus tree preserves the nesting information common to all input trees.

Connection to rooted triplets: $\bigcap_{i=1}^{k} r(T_i) \subseteq r(T^{\text{Adams}}) \subseteq \bigcup_{i=1}^{k} r(T_i)$

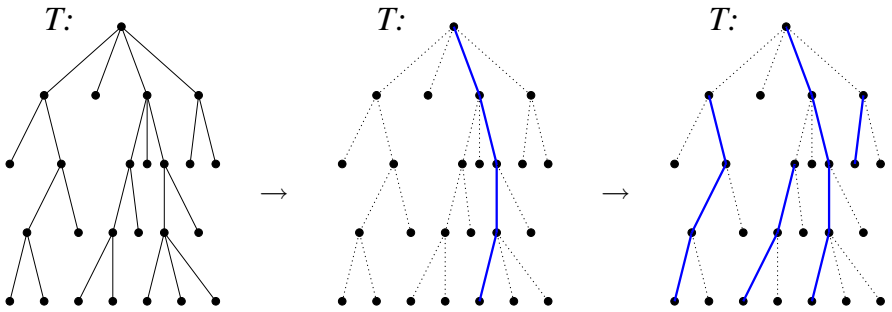
Adams consensus tree, new algorithm 1

Our first new algorithm runs in $\boxed{O(kn \log n)}$ time.

- Recall that the old algorithm computes a partition $\pi(\mathcal{S})$ of the leaf labels equal to the product of the partitions $\pi(T_1), \pi(T_2), \ldots, \pi(T_k)$, where $\pi(T_i) = \{\Lambda(T_i[c]) : c \in Child(r_i)\}$, and makes a recursive call to each part in $\pi(\mathcal{S})$.
- Old algorithm's time complexity: $\boxed{O(kn^2)}$
  Difficult to improve directly because in the worst case, some parts in $\pi(\mathcal{S})$ may be of size $\Omega(n)$.
- **Main idea of the first new algorithm:**
  Use the centroid path decomposition technique to avoid making recursive calls to large subproblems, and treat them iteratively instead.

Adams consensus tree, centroid path decompositions

A centroid path in a tree T [Cole *et al.*; *SIAM Journal on Computing*; 2000] is a path in T of the form $P = \langle p_\alpha, p_{\alpha-1}, \ldots, p_1 \rangle$, where p_{w-1} is any child of p_w with the maximum number of leaf descendants, and p_1 is a leaf.

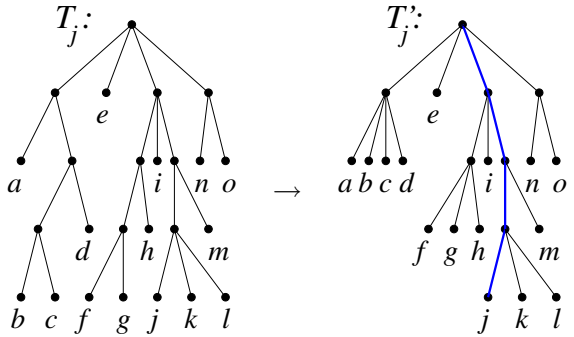


For any $u \in V(T)$ such that u does not belong to P but the parent of u does, the subtree $T[u]$ is called a side tree of P.
For any side tree τ of a centroid path starting at the root of T, the property $|\Lambda(\tau)| \leq |\Lambda(T)|/2$ holds.

Adams consensus tree, T'_j-trees

For each $j \in \{1, 2, \ldots, k\}$, build a tree T'_j from T_j in $O(n)$ time as follows:

1. Let P_j be a centroid path in T_j that starts at the root of T_j.
2. Let T'_j be a copy of T_j. For each non-root, internal node in T'_j whose parent does not belong to P_j, contract its parent edge.



$\Rightarrow$ T'_j is a useful summary of T_j that helps us to quickly retrieve the leaves in any side tree of P_j or check which side tree a specified leaf belongs to.

Adams consensus tree, restricted partitions

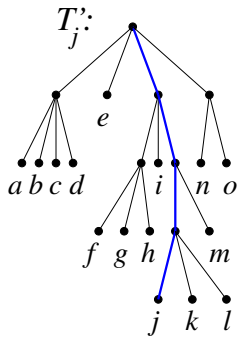
We use the T'_j-trees to compute the partition $\pi(\mathcal{S})$ of L at the top level of the Adams consensus tree. More precisely:

Let $X = \{x \in L : x$ belongs to a side tree attached to the root of T_j for some $j \in \{1, 2, \ldots, k\}\}$.

Define the restricted partition: $\pi(\mathcal{S}; X) = \{B \cap X : B \in \pi(\mathcal{S}), |B \cap X| \geq 1\}$
(In other words, $\pi(\mathcal{S}; X)$ is the partition $\pi(\mathcal{S})$ restricted to elements in X.)

Lemma 15
If $X \neq L$ then $\pi(\mathcal{S}) = \pi(\mathcal{S}; X) \cup \{L \setminus X\}$.
If $X = L$ then $\pi(\mathcal{S}) = \pi(\mathcal{S}; X)$.

To compute X, we can use the T'_j-trees.
Moreover, $\pi(\mathcal{S}; X) = \pi(\{T'_1, T'_2, \ldots, T'_k\}; X)$, where the latter can be computed efficiently.
Plug into Lemma 15 $\Rightarrow$ We obtain $\pi(\mathcal{S})$.



Algorithm `New_Adams_consensus_k`, pseudocode

Base case: If there is only one leaf then return it.

Otherwise:

1. For each $j \in \{1, 2, \ldots, k\}$: compute P_j and the tree T'_j.
2. (Main loop:)
   i. Use the T'_j-trees to compute $X_1 = \{x \in L : x$ belongs to a side tree attached to the root of T_j for some $j \in \{1, 2, \ldots, k\}\}$ as well as the restricted partition $\pi_{X_1} = \pi(\{T'_1, T'_2, \ldots, T'_k\}; X_1)$.
   (By Lemma 15, the parts in π_{X_1} along with $\{L \setminus X_1\}$ yield the partition at the top level of the Adams consensus tree.)
   ii. Remove the leaves belonging to X_1 from all T'_j-trees and contract.
   iii. Repeat the process (getting $X_2, X_3, \ldots, X_h$) until the T'_j-trees are empty.
3. For $w := h$ downto 1 do:
   i. For each part B in $\pi_{X_w} = \pi(\{T'_1, T'_2, \ldots, T'_k\}; X_w)$, construct $T_1|B$, $T_2|B, \ldots, T_k|B$ and recursively compute the Adams consensus tree T_B.
   ii. Let Q_w be a tree with a root whose children are the T_B-trees just computed and (if $w < h$) Q_{w+1} (corresponding to the part $\{L \setminus X_w\}$).
4. Return Q_1.

Adams consensus tree, new algorithm 2

Our second new algorithm only works for the special case $k = 2$.
It runs in $O(n \cdot \frac{\log n}{\log \log n})$ time.

- Consider any recursive call `Old_Adams_consensus`$(\{T_1|B,\ T_2|B\})$, where $B \subseteq L$.
  $\Omega(|B|)$ time is used to obtain the partition π of the leaves in B.
- The second new algorithm uses a faster way to do the partitioning based on two simple observations:
  1. First, observe that B always satisfies $B = \Lambda(T_1[u]) \cap \Lambda(T_2[v])$ for some pair of nodes $u \in V(T_1)$, $v \in V(T_2)$.
     $\Rightarrow$ Successive recursive calls to the algorithm can be specified by pairs of vertices from T_1 and T_2.
  2. Secondly, observe that one needs to proceed recursively from (u, v) only to those (u', v'), where $u' \in Child^{T_1}(u)$ and $v' \in Child^{T_2}(v)$, for which $|\Lambda(T_1[u']) \cap \Lambda(T_2[v'])| > 0$.

Algorithm `New_Adams_consensus_2`, pseudocode

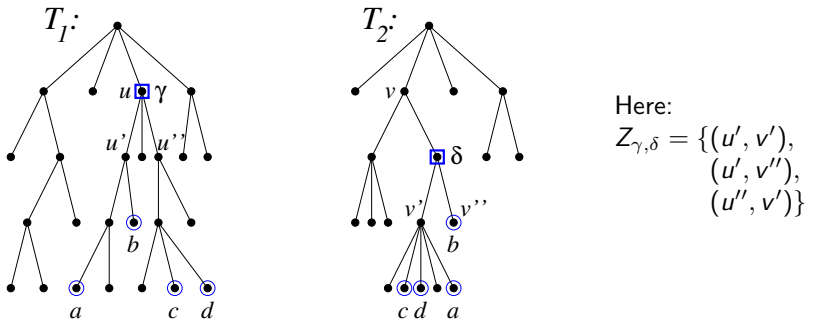
For $u \in V(T_1)$, $v \in V(T_2)$, define:

$$Z_{u,v} = \{(u', v') : u' \in Child^{T_1}(u),\ v' \in Child^{T_2}(v),\ |\Lambda(T_1[u']) \cap \Lambda(T_2[v'])| > 0\}$$

Then, for any $u \in V(T_1)$ and $v \in V(T_2)$:

Lemma 16

Let $B = \Lambda(T_1[u]) \cap \Lambda(T_2[v])$, $\gamma = lca^{T_1}(B)$, and $\delta = lca^{T_2}(B)$. If $|B| > 1$ then $\pi(\{T_1[u]|B,\ T_2[v]|B\}) = \{\Lambda(T_1[u']) \cap \Lambda(T_2[v']) : (u', v') \in Z_{\gamma,\delta}\}$.



Algorithm `New_Adams_consensus_2`, pseudocode

For $u \in V(T_1)$, $v \in V(T_2)$, define:

$$Z_{u,v} = \{(u', v') : u' \in Child^{T_1}(u),\ v' \in Child^{T_2}(v),\ |\Lambda(T_1[u']) \cap \Lambda(T_2[v'])| > 0\}$$

Then, for any $u \in V(T_1)$ and $v \in V(T_2)$:

Lemma 16

Let $B = \Lambda(T_1[u]) \cap \Lambda(T_2[v])$, $\gamma = lca^{T_1}(B)$, and $\delta = lca^{T_2}(B)$. If $|B| > 1$ then $\pi(\{T_1[u]|B,\ T_2[v]|B\}) = \{\Lambda(T_1[u']) \cap \Lambda(T_2[v']) : (u', v') \in Z_{\gamma,\delta}\}$.

$\Rightarrow$ Algorithm:

Base case: If $|\Lambda(T_1[u]) \cap \Lambda(T_2[v])| = 1$ then return this shared leaf.

Otherwise:

1. Compute $\gamma = lca^{T_1}(B)$ and $\delta = lca^{T_2}(B)$, where $B = \Lambda(T_1[u]) \cap \Lambda(T_2[v])$.
2. Construct $Z_{\gamma,\delta}$.
3. For $(u', v') \in Z_{\gamma,\delta}$, recursively compute its Adams consensus tree $T_{u',v'}$.
4. Return the tree obtained by attaching all of the computed $T_{u',v'}$-trees to a newly created root node.

To achieve a good time complexity for `New_Adams_consensus_2`:

- Fix an arbitrary left-to-right ordering of T_1 and T_2.
- Preprocess the two trees in $O(n)$ time so that any $lca^{T_j}(B)$-query can be answered in $O(|B|)$ time [Bender, Farach-Colton; 2000].
- Use a data structure for orthogonal range counting on a grid to quickly find the leftmost (and rightmost) leaf in $T_1 \,|\, (\Lambda(T_1[u]) \cap \Lambda(T_2[v]))$ and in $T_2 \,|\, (\Lambda(T_1[u]) \cap \Lambda(T_2[v]))$, as well as to quickly construct $Z_{\gamma,\delta}$.

Auxiliary data structure for orthogonal range counting

Let N be a set of n points on an $n \times n$ grid such that every column contains exactly one point and every row contains exactly one point.

Lemma 17

We can build a data structure $D(N)$ in $O(n \cdot \frac{\log n}{\log \log n})$ time after which:

- Counting the number of points in any query rectangle $[x..x'] \times [y..y']$ takes $O(\frac{\log n}{\log \log n})$ time.
- Reporting the point with the maximum (or minimum) x-coordinate inside any query rectangle $[x..x'] \times [y..y']$ takes $O(\frac{\log n}{\log \log n})$ time.

$D(N)$ is an extension of the wavelet tree-based data structure in [Bose, He, Maheshwari, Morin; WADS 2009] for supporting orthogonal range counting queries on a grid to also support truncated range maximum queries.

Remark: The time needed to construct $D(N)$ is bounded in the same way.

How to apply the auxiliary data structure

In the algorithm, represent each leaf label in the trees as a 2D point:

- For $1 \leq i \leq n$, let $L_1(i)$ and $L_2(i)$ be the ith leaf in T_1 and T_2, respectively, in the fixed left-to-right ordering.
- Define $N = \{(L_1^{-1}(\ell), L_2^{-1}(\ell)) : \ell \in L\}$ and build $D(N)$.

For any pair of siblings u, u' in a tree T, let $T[u..u']$ denote the set of all rooted subtrees of the form $T[x]$, where $x \in [u, \ldots, u']$ in T.

$\Rightarrow$ Each $\ell \in \Lambda(T_1[v..v'])$ satisfies $L_1^{-1}(l_u) \leq L_1^{-1}(\ell) \leq L_1^{-1}(r_{u'})$, where l_u is the leftmost leaf in $T_1[u]$ and $r_{u'}$ the rightmost leaf in $T_1[u']$. (Same for T_2.)

Lemma 18

Given $D(N)$, for any siblings u and u' in T_1 and any siblings v and v' in T_2, $|\Lambda(T_1[u..u']) \cap \Lambda(T_2[v..v'])|$ can be found in $O(\frac{\log n}{\log \log n})$ time.

Furthermore, the leftmost and rightmost leaves in T_1 (or T_2) among all leaves in $\Lambda(T_1[u..u']) \cap \Lambda(T_2[v..v'])$ can be reported in $O(\frac{\log n}{\log \log n})$ time.

Adams consensus tree, summary

Theorems 6 and 7

The Adams consensus tree can be computed in:

- $O(n \cdot \frac{\log n}{\log \log n})$ time when $k = 2$; and
- $O(kn \log n)$ time when $k \geq 3$.

PART IV : Conclusion

FACT: Fast Algorithms for Consensus Trees

- Let $k = |\mathcal{S}|$, $n = |L|$, $p =$ the number of different clusters occurring in $\mathcal{S}$, and $q =$ the total number of clusters occurring in $\mathcal{S}$.

Consensus tree:	**Previously fastest algorithm**	**Our project**
Strict	$O(kn)$ time [Day; 1985]	—
Majority rule	$O(n^2+k^2n)$ time [Wareham; 1985]	$O(kn)$ time *J. of the ACM*; 2016
Majority rule (+)	Polynomial time [Dong *et al.*; 2010]	$O(kn)$ time *IEEE TCBB*; 2017
Loose	$O(q^2n) = O(k^2n^3)$ time [McMorris, Wilkinson; 2011]	$O(kn)$ time *J. of the ACM*; 2016
Frequency difference	$O(kn\log^2 n)$ time [Gawrychowski *et al.*; 2018]	$O(kn\log n)$ time STACS 2024
Greedy	$O(qn + pn^2) = O(kn^3)$ time [Bryant; 2003]	~~$O(qn) = O(kn^2)$ time *J. of the ACM*; 2016~~

Remark 1: The input size is $\Omega(kn)$ $\Rightarrow$ $O(kn)$ time is optimal.

Remark 2: The currently fastest algorithm for the greedy consensus tree runs in $O(kn(\log k + \log^2 n))$ time [H. Wu; ICALP 2020].

FACT: Fast Algorithms for Consensus Trees

Consensus tree:	**Previously fastest algorithm**	**Our project**
Adams for $k = 2$	$O(n^2)$ time [Adams; 1972]	$O(n \cdot \frac{\log n}{\log\log n})$ time *Inf. & Comput.*; 2017
Adams for $k \geq 3$	$O(kn^2)$ time [Adams; 1972]	$O(kn\log n)$ time *Inf. & Comput.*; 2017
Minimally resolved local	— —	$O(kn^3 + 2.733^n)$ time *AIMS Med. Sci.*; 2018
Minimally rooted-triplet-induc. local	— —	$O(kn^3 + 4^n \cdot poly(n))$ time *AIMS Med. Sci.*; 2018
R* for $k = 2$	$O(n^3)$ time [Kannan *et al.*; 1998]	$O(n^2)$ time *Algorithmica*; 2016
R* for $k \geq 3$	$O(kn^3)$ time [Bryant; 2003]	$O(n^2\log^{k+2} n)$ time *Algorithmica*; 2016
Asymmetric median for $k = 2$	$O(n^{2.5})$ time [Phillips, Warnow; 1996]	Work in progress

Note: Source code (for about half of our algorithms) is available from:
`https://github.com/Mesh89/FACT` & `https://github.com/Mesh89/FACT2`

Open problems

1. Develop $O(kn)$-time (i.e., optimal) algorithms for computing:
   - the frequency difference consensus tree
   - a greedy consensus tree
   - the Adams consensus tree
   - the R* consensus tree
2. Define new types of consensus trees that can be computed efficiently and are even more informative than the existing ones.
3. Generalizations to consensus supertrees, where the input trees are allowed to have *different* leaf label sets?
4. Generalizations to MUL-trees, where the same leaf label can appear many times in a tree?
5. Generalizations to consensus phylogenetic networks, where nodes are allowed to have more than one parent?
6. Write an up-to-date survey.

References

- J. Jansson, C. Shen, and W.-K. Sung. Improved Algorithms for Constructing Consensus Trees. *Journal of the ACM*, Vol. 63, No. 3, Article 28, 2016.
- J. Jansson, W.-K. Sung, H. Vu, and S.-M. Yiu. Faster Algorithms for Computing the R* Consensus Tree. *Algorithmica*, Vol. 76, No. 4, pp. 1224–1244, 2016.
- J. Jansson, Z. Li, and W.-K. Sung. On Finding the Adams Consensus Tree. *Information and Computation*, Vol. 256, pp. 334–347, 2017.
- J. Jansson, R. Rajaby, C. Shen, and W.-K. Sung. Algorithms for the Majority Rule (+) Consensus Tree and the Frequency Difference Consensus Tree. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, Vol. 15, No. 1, pp. 15–26, 2018.
- J. Jansson, R. Rajaby, and W.-K. Sung. Minimal Phylogenetic Supertrees and Local Consensus Trees. *AIMS Medical Science*, Vol. 5, No. 2, pp. 181–203, 2018.
- J. Jansson, W.-K. Sung, S. A. Tabatabaee, and Y. Yang. A Faster Algorithm for Constructing the Frequency Difference Consensus Tree. In *Proceedings of the Forty-First International Symposium on Theoretical Aspects of Computer Science* (STACS 2024), LIPIcs, Vol. 289, Article No. 43, pp. 43:1–43:17, 2024.