A Faster Construction of Greedy Consensus Trees

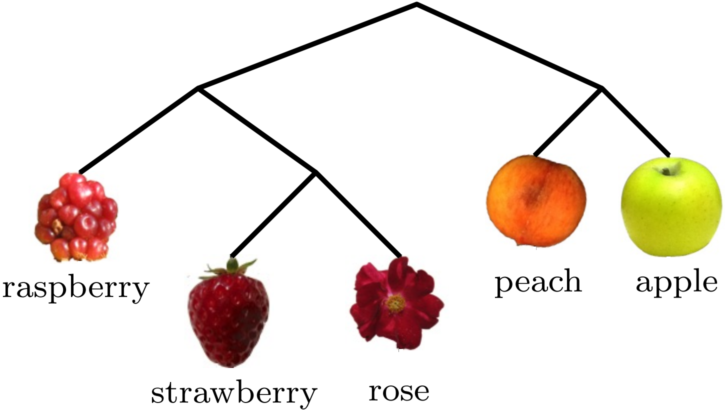
Paweł Gawrychowski[1] Gad M. Landau[2] Wing-Kin Sung[3]
Oren Weimann[2]

[1]University of Wrocław
[2]University of Haifa
[3]National University of Singapore

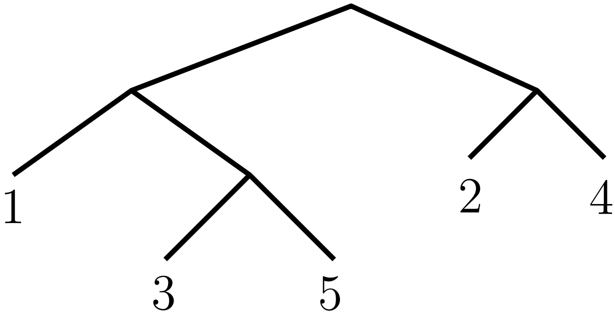
July 15, 2018

Slides by Paweł Gawrychowski

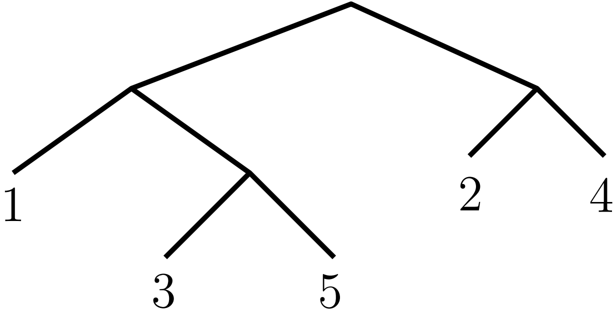
Phylogenetic trees



Phylogenetic trees

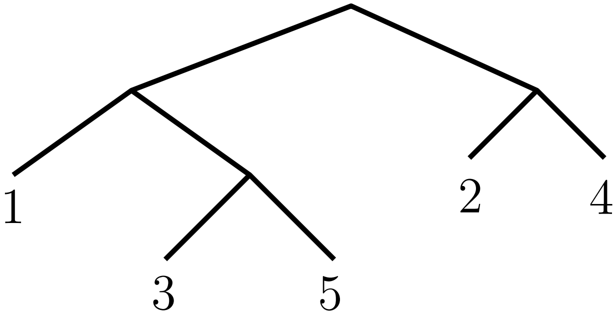


Phylogenetic trees



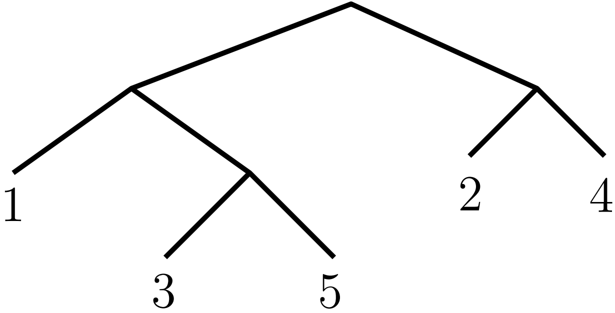
1. Rooted and unordered tree.
2. No unary nodes, but the degrees are unbounded.
3. Each leaf correspond to a species and has a distinct label from $[n]$.

Phylogenetic trees



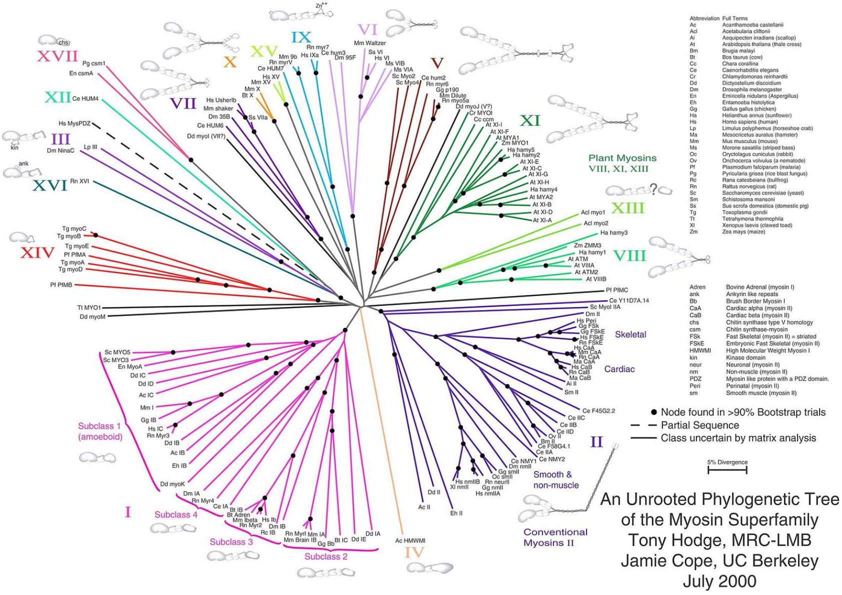
1. Rooted and unordered tree.
2. No unary nodes, but the degrees are unbounded.
3. Each leaf correspond to a species and has a distinct label from $[n]$.

Phylogenetic trees

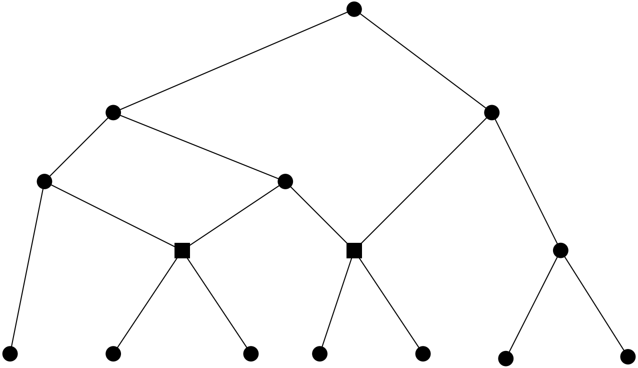


1. Rooted and unordered tree.
2. No unary nodes, but the degrees are unbounded.
3. Each leaf correspond to a species and has a distinct label from $[n]$.

Not in this talk: unrooted phylogenetic trees



Not in this talk: phylogenetic networks



Motivation

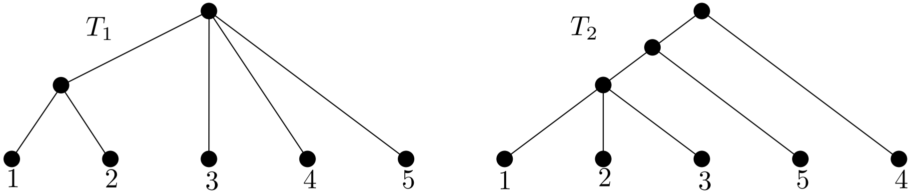
By applying different reconstruction methods or using different data sources we might obtain multiple phylogenetic trees. How to combine them into a single tree?

For any node of T_1 or T_2, there is a node of the combined tree with exactly the same set of leaf labels.

In practice, the set of leaf labels in a tree might be a proper subset of $[n]$, but we assume that it is exactly $[n]$ as in the previous work.

Motivation

By applying different reconstruction methods or using different data sources we might obtain multiple phylogenetic trees. How to combine them into a single tree?

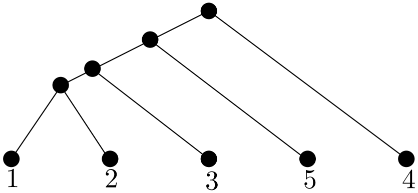


For any node of T_1 or T_2, there is a node of the combined tree with exactly the same set of leaf labels.

In practice, the set of leaf labels in a tree might be a proper subset of $[n]$, but we assume that it is exactly $[n]$ as in the previous work.

Motivation

By applying different reconstruction methods or using different data sources we might obtain multiple phylogenetic trees. How to combine them into a single tree?

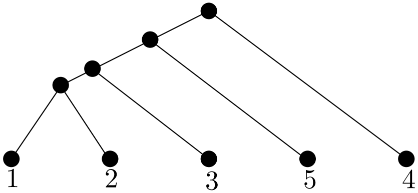


For any node of T_1 or T_2, there is a node of the combined tree with exactly the same set of leaf labels.

In practice, the set of leaf labels in a tree might be a proper subset of $[n]$, but we assume that it is exactly $[n]$ as in the previous work.

Motivation

By applying different reconstruction methods or using different data sources we might obtain multiple phylogenetic trees. How to combine them into a single tree?

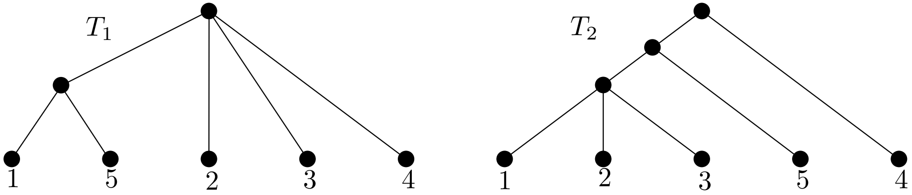


For any node of T_1 or T_2, there is a node of the combined tree with exactly the same set of leaf labels.

In practice, the set of leaf labels in a tree might be a proper subset of $[n]$, but we assume that it is exactly $[n]$ as in the previous work.

Motivation

By applying different reconstruction methods or using different data sources we might obtain multiple phylogenetic trees. How to combine them into a single tree?



For any node of T_1 or T_2, there is a node of the combined tree with exactly the same set of leaf labels.

In practice, the set of leaf labels in a tree might be a proper subset of $[n]$, but we assume that it is exactly $[n]$ as in the previous work.

Motivation

By applying different reconstruction methods or using different data sources we might obtain multiple phylogenetic trees. How to combine them into a single tree?

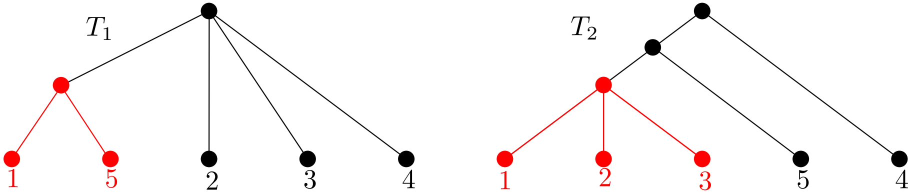
?

For any node of T_1 or T_2, there is a node of the combined tree with exactly the same set of leaf labels.

In practice, the set of leaf labels in a tree might be a proper subset of $[n]$, but we assume that it is exactly $[n]$ as in the previous work.

Motivation

By applying different reconstruction methods or using different data sources we might obtain multiple phylogenetic trees. How to combine them into a single tree?

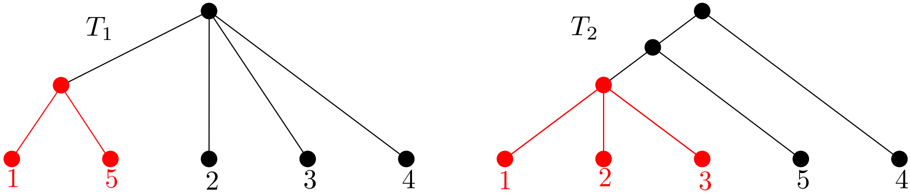


For any node of T_1 or T_2, there is a node of the combined tree with exactly the same set of leaf labels.

In practice, the set of leaf labels in a tree might be a proper subset of $[n]$, but we assume that it is exactly $[n]$ as in the previous work.

Motivation

By applying different reconstruction methods or using different data sources we might obtain multiple phylogenetic trees. How to combine them into a single tree?



For any node of T_1 or T_2, there is a node of the combined tree with exactly the same set of leaf labels.

In practice, the set of leaf labels in a tree might be a proper subset of $[n]$, but we assume that it is exactly $[n]$ as in the previous work.

Notation

Input: k trees $T_1, \ldots, T_k$ on n leaves with distinct labels from $[n]$.
Output: a single tree T_r on n leaves with distinct labels from $[n]$.

Cluster

$\mathrm{L}(u)$ = labels of all leaves in the subtree rooted at u

$$\mathrm{L}(u) = \{1, 2\}$$

We identify a tree with the set of its clusters.

Notation

Input: k trees $T_1, \ldots, T_k$ on n leaves with distinct labels from $[n]$.
Output: a single tree T_r on n leaves with distinct labels from $[n]$.

Cluster

L(u) = labels of all leaves in the subtree rooted at u

$$L(u) = \{1, 2\}$$

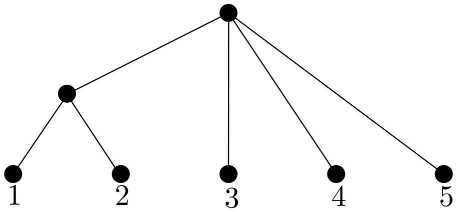
We identify a tree with the set of its clusters.

Notation

Input: k trees $T_1, \ldots, T_k$ on n leaves with distinct labels from $[n]$.
Output: a single tree T_r on n leaves with distinct labels from $[n]$.

Cluster

$\mathrm{L}(u)$ = labels of all leaves in the subtree rooted at u



$\mathrm{L}(u) = \{1, 2\}$

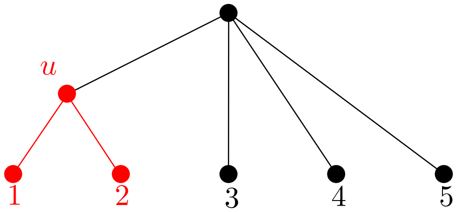
We identify a tree with the set of its clusters.

Notation

Input: k trees $T_1, \ldots, T_k$ on n leaves with distinct labels from $[n]$.
Output: a single tree T_r on n leaves with distinct labels from $[n]$.

Cluster

$\mathrm{L}(u)$ = labels of all leaves in the subtree rooted at u



$\mathrm{L}(u) = \{1, 2\}$

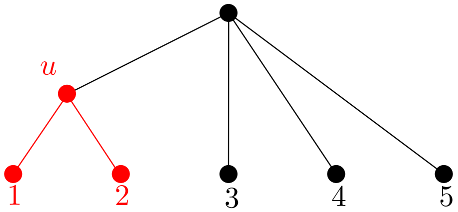
We identify a tree with the set of its clusters.

Notation

Input: k trees $T_1, \ldots, T_k$ on n leaves with distinct labels from $[n]$.
Output: a single tree T_r on n leaves with distinct labels from $[n]$.

Cluster

L(u) = labels of all leaves in the subtree rooted at u



$$\mathsf{L}(u) = \{1, 2\}$$

We identify a tree with the set of its clusters.

Different methods of combining trees

1. Majority consensus tree,
2. loose consensus tree,
3. frequency difference consensus tree,
4. greedy consensus tree.

and Adam's consensus tree, strict consensus tree, asymmetric median consensus tree...

Compatible clusters

C_1 and C_2 are compatible if $C_1 \cap C_2 = \emptyset$, $C_1 \subseteq C_2$ or $C_2 \subseteq C_1$.

$\{1, 2\}$ and $\{3, 4\}$ are compatible, and so are $\{1, 2, 3\}$ and $\{2, 3\}$, but $\{1, 2\}$ and $\{2, 3\}$ are not.

A collection of clusters corresponds to a tree iff they are pairwise compatible.

Different methods of combining trees

1. Majority consensus tree,
2. loose consensus tree,
3. frequency difference consensus tree,
4. greedy consensus tree.

and Adam's consensus tree, strict consensus tree, asymmetric median consensus tree...

Compatible clusters

C_1 and C_2 are compatible if $C_1 \cap C_2 = \emptyset$, $C_1 \subseteq C_2$ or $C_2 \subseteq C_1$.

$\{1, 2\}$ and $\{3, 4\}$ are compatible, and so are $\{1, 2, 3\}$ and $\{2, 3\}$, but $\{1, 2\}$ and $\{2, 3\}$ are not.

A collection of clusters corresponds to a tree iff they are pairwise compatible.

Different methods of combining trees

1. Majority consensus tree,
2. loose consensus tree,
3. frequency difference consensus tree,
4. greedy consensus tree.

and Adam's consensus tree, strict consensus tree, asymmetric median consensus tree...

Compatible clusters

C_1 and C_2 are compatible if $C_1 \cap C_2 = \emptyset$, $C_1 \subseteq C_2$ or $C_2 \subseteq C_1$.

$\{1, 2\}$ and $\{3, 4\}$ are compatible, and so are $\{1, 2, 3\}$ and $\{2, 3\}$, but $\{1, 2\}$ and $\{2, 3\}$ are not.

A collection of clusters corresponds to a tree iff they are pairwise compatible.

Different methods of combining trees

1. Majority consensus tree,
2. loose consensus tree,
3. frequency difference consensus tree,
4. greedy consensus tree.

and Adam's consensus tree, strict consensus tree, asymmetric median consensus tree...

Compatible clusters

C_1 and C_2 are compatible if $C_1 \cap C_2 = \emptyset$, $C_1 \subseteq C_2$ or $C_2 \subseteq C_1$.

$\{1, 2\}$ and $\{3, 4\}$ are compatible, and so are $\{1, 2, 3\}$ and $\{2, 3\}$, but $\{1, 2\}$ and $\{2, 3\}$ are not.

A collection of clusters corresponds to a tree iff they are pairwise compatible.

Different methods of combining trees

1. Majority consensus tree,
2. loose consensus tree,
3. frequency difference consensus tree,
4. greedy consensus tree.

and Adam's consensus tree, strict consensus tree, asymmetric median consensus tree...

Compatible clusters

C_1 and C_2 are compatible if $C_1 \cap C_2 = \emptyset$, $C_1 \subseteq C_2$ or $C_2 \subseteq C_1$.

$\{1,2\}$ and $\{3,4\}$ are compatible, and so are $\{1,2,3\}$ and $\{2,3\}$, but $\{1,2\}$ and $\{2,3\}$ are not.

A collection of clusters corresponds to a tree iff they are pairwise compatible.

Different methods of combining trees

1. Majority consensus tree,
2. loose consensus tree,
3. frequency difference consensus tree,
4. greedy consensus tree.

and Adam's consensus tree, strict consensus tree, asymmetric median consensus tree...

Compatible clusters

C_1 and C_2 are compatible if $C_1 \cap C_2 = \emptyset$, $C_1 \subseteq C_2$ or $C_2 \subseteq C_1$.

$\{1, 2\}$ and $\{3, 4\}$ are compatible, and so are $\{1, 2, 3\}$ and $\{2, 3\}$, but $\{1, 2\}$ and $\{2, 3\}$ are not.

A collection of clusters corresponds to a tree iff they are pairwise compatible.

Different methods of combining trees

1. Majority consensus tree,
2. loose consensus tree,
3. frequency difference consensus tree,
4. greedy consensus tree.

and Adam's consensus tree, strict consensus tree, asymmetric median consensus tree...

Compatible clusters

C_1 and C_2 are compatible if $C_1 \cap C_2 = \emptyset$, $C_1 \subseteq C_2$ or $C_2 \subseteq C_1$.

$\{1,2\}$ and $\{3,4\}$ are compatible, and so are $\{1,2,3\}$ and $\{2,3\}$, but $\{1,2\}$ and $\{2,3\}$ are not.

A collection of clusters corresponds to a tree iff they are pairwise compatible.

Different methods of combining trees

1. Majority consensus tree,
2. loose consensus tree,
3. frequency difference consensus tree,
4. greedy consensus tree.

and Adam's consensus tree, strict consensus tree, asymmetric median consensus tree...

Compatible clusters

C_1 and C_2 are compatible if $C_1 \cap C_2 = \emptyset$, $C_1 \subseteq C_2$ or $C_2 \subseteq C_1$.

$\{1,2\}$ and $\{3,4\}$ are compatible, and so are $\{1,2,3\}$ and $\{2,3\}$, but $\{1,2\}$ and $\{2,3\}$ are not.

A collection of clusters corresponds to a tree iff they are pairwise compatible.

Majority consensus tree

We choose all clusters that appear in more than $k/2$ of the trees.

For any two chosen clusters C_1 and C_2, there is a tree T_i containing both C_1 and C_2, so they must be compatible.

Hence, chosen clusters correspond to a single tree T_r.

Majority consensus tree

We choose all clusters that appear in more than $k/2$ of the trees.

For any two chosen clusters C_1 and C_2, there is a tree T_i containing both C_1 and C_2, so they must be compatible.

Hence, chosen clusters correspond to a single tree T_r.

Majority consensus tree

We choose all clusters that appear in more than $k/2$ of the trees.

For any two chosen clusters C_1 and C_2, there is a tree T_i containing both C_1 and C_2, so they must be compatible.

Hence, chosen clusters correspond to a single tree T_r.

Loose consensus tree

Compatible cluster

A cluster C is compatible with a tree T if it is compatible with cluster $\mathrm{L}(u)$, for every $u \in T$.

We choose all clusters that appear in at least one tree and are compatible with all trees. By definition, chosen clusters correspond to a single tree T_r.

Loose consensus tree

Compatible cluster

A cluster C is compatible with a tree T if it is compatible with cluster $\mathrm{L}(u)$, for every $u \in T$.

We choose all clusters that appear in at least one tree and are compatible with all trees. By definition, chosen clusters correspond to a single tree T_r.

Frequency difference consensus tree

Frequency

The frequency of a cluster C is the number of trees T_i such that $C = \mathrm{L}(u)$ for some $u \in T_i$.

For every cluster $\mathrm{L}(u)$, where $u \in T_i$ for some i, we choose $\mathrm{L}(u)$ if its frequency is strictly larger than the frequency of any cluster $\mathrm{L}(v)$, where $v \in T_j$ for some j, such that $\mathrm{L}(u)$ is not compatible with $\mathrm{L}(v)$.

Frequency difference consensus tree

Frequency

The frequency of a cluster C is the number of trees T_i such that $C = \mathrm{L}(u)$ for some $u \in T_i$.

For every cluster $\mathrm{L}(u)$, where $u \in T_i$ for some i, we choose $\mathrm{L}(u)$ if its frequency is strictly larger than the frequency of any cluster $\mathrm{L}(v)$, where $v \in T_j$ for some j, such that $\mathrm{L}(u)$ is not compatible with $\mathrm{L}(v)$.

Greedy consensus tree

1. We consider all clusters that appear in at least one tree in decreasing order of their frequencies.
2. Consider one such cluster $\mathrm{L}(u)$, where $u \in T_i$ for some i. If $\mathrm{L}(u)$ is consistent with $\mathcal{C}$, add $\mathrm{L}(u)$ to $\mathcal{C}$.
3. Return the tree corresponding to $\mathcal{C}$.

Greedy consensus tree

1. We consider all clusters that appear in at least one tree in decreasing order of their frequencies.
2. Consider one such cluster $\mathrm{L}(u)$, where $u \in T_i$ for some i. If $\mathrm{L}(u)$ is consistent with $\mathcal{C}$, add $\mathrm{L}(u)$ to $\mathcal{C}$.
3. Return the tree corresponding to $\mathcal{C}$.

Greedy consensus tree

1. We consider all clusters that appear in at least one tree in decreasing order of their frequencies.
2. Consider one such cluster $L(u)$, where $u \in T_i$ for some i. If $L(u)$ is consistent with $\mathcal{C}$, add $L(u)$ to $\mathcal{C}$.
3. Return the tree corresponding to $\mathcal{C}$.

Known bounds

Majority	$\mathcal{O}(k \cdot n)$	Jansson, Shen, Sung JACM 2016
Loose	$\mathcal{O}(k \cdot n)$	Jansson, Shen, Sung JACM 2016
Frequency	$\tilde{\mathcal{O}}(\min\{n, k\} \cdot k \cdot n)$	Jansson et al. TCBB 2016
Greedy	$\mathcal{O}(k \cdot n^2)$	Jansson, Shen, Sung JACM 2016
Frequency	$\tilde{\mathcal{O}}(k \cdot n)$	This paper
Greedy	$\tilde{\mathcal{O}}(k \cdot n^{1.5})$	This paper

Known bounds

Majority	$\mathcal{O}(k \cdot n)$	Jansson, Shen, Sung JACM 2016
Loose	$\mathcal{O}(k \cdot n)$	Jansson, Shen, Sung JACM 2016
Frequency	$\tilde{\mathcal{O}}(\min\{n, k\} \cdot k \cdot n)$	Jansson et al. TCBB 2016
Greedy	$\mathcal{O}(k \cdot n^2)$	Jansson, Shen, Sung JACM 2016
Frequency	$\tilde{\mathcal{O}}(k \cdot n)$	This paper
Greedy	$\tilde{\mathcal{O}}(k \cdot n^{1.5})$	This paper

Known bounds

Majority	$\mathcal{O}(k \cdot n)$	Jansson, Shen, Sung JACM 2016
Loose	$\mathcal{O}(k \cdot n)$	Jansson, Shen, Sung JACM 2016
Frequency	$\tilde{\mathcal{O}}(\min\{n, k\} \cdot k \cdot n)$	Jansson et al. TCBB 2016
Greedy	$\mathcal{O}(k \cdot n^2)$	Jansson, Shen, Sung JACM 2016
Frequency	$\tilde{\mathcal{O}}(k \cdot n)$	This paper
Greedy	$\tilde{\mathcal{O}}(k \cdot n^{1.5})$	This paper

Known bounds

Majority	$\mathcal{O}(k \cdot n)$	Jansson, Shen, Sung JACM 2016
Loose	$\mathcal{O}(k \cdot n)$	Jansson, Shen, Sung JACM 2016
Frequency	$\tilde{\mathcal{O}}(\min\{n, k\} \cdot k \cdot n)$	Jansson et al. TCBB 2016
Greedy	$\mathcal{O}(k \cdot n^2)$	Jansson, Shen, Sung JACM 2016
Frequency	$\tilde{\mathcal{O}}(k \cdot n)$	This paper
Greedy	$\tilde{\mathcal{O}}(k \cdot n^{1.5})$	This paper

Known and new bounds

Majority	$\mathcal{O}(k \cdot n)$	Jansson, Shen, Sung JACM 2016
Loose	$\mathcal{O}(k \cdot n)$	Jansson, Shen, Sung JACM 2016
Frequency	$\tilde{\mathcal{O}}(\min\{n, k\} \cdot k \cdot n)$	Jansson et al. TCBB 2016
Greedy	$\mathcal{O}(k \cdot n^2)$	Jansson, Shen, Sung JACM 2016

Frequency	$\tilde{\mathcal{O}}(k \cdot n)$	This paper
Greedy	$\tilde{\mathcal{O}}(k \cdot n^{1.5})$	This paper

Known and new bounds

Majority	$\mathcal{O}(k \cdot n)$	Jansson, Shen, Sung JACM 2016
Loose	$\mathcal{O}(k \cdot n)$	Jansson, Shen, Sung JACM 2016
Frequency	$\tilde{\mathcal{O}}(\min\{n, k\} \cdot k \cdot n)$	Jansson et al. TCBB 2016
Greedy	$\mathcal{O}(k \cdot n^2)$	Jansson, Shen, Sung JACM 2016
Frequency	$\tilde{\mathcal{O}}(k \cdot n)$	This paper
Greedy	$\tilde{\mathcal{O}}(k \cdot n^{1.5})$	This paper

Known and new bounds

Majority	$\mathcal{O}(k \cdot n)$	Jansson, Shen, Sung JACM 2016
Loose	$\mathcal{O}(k \cdot n)$	Jansson, Shen, Sung JACM 2016
Frequency	$\tilde{\mathcal{O}}(\min\{n, k\} \cdot k \cdot n)$	Jansson et al. TCBB 2016
Greedy	$\mathcal{O}(k \cdot n^2)$	Jansson, Shen, Sung JACM 2016
Frequency	$\tilde{\mathcal{O}}(k \cdot n)$	This paper
Greedy	$\tilde{\mathcal{O}}(k \cdot n^{1.5})$	This paper

Frequency difference consensus tree

Previous algorithm:

1. Compute the frequency of every cluster $\mathrm{L}(u)$, where $u \in T_i$ for some i, in $\mathcal{O}(\min\{n,k\} \cdot k \cdot n)$ time.
2. Given the frequency of every cluster, construct the frequency difference consensus tree in additional $\mathcal{O}(k \cdot n \log^2 n)$ time.

Only need to compute identifiers $\mathrm{id}(u)$, where $u \in T_i$ for some i, such that $\mathrm{id}(u) = \mathrm{id}(v)$, where $u \in T_i$, $v \in T_j$ for some i and j, iff $\mathrm{L}(u) = \mathrm{L}(v)$.

Frequency difference consensus tree

Previous algorithm:

1. Compute the frequency of every cluster $\mathrm{L}(u)$, where $u \in T_i$ for some i, in $\mathcal{O}(\min\{n, k\} \cdot k \cdot n)$ time.
2. Given the frequency of every cluster, construct the frequency difference consensus tree in additional $\mathcal{O}(k \cdot n \log^2 n)$ time.

Only need to compute identifiers $\mathrm{id}(u)$, where $u \in T_i$ for some i, such that $\mathrm{id}(u) = \mathrm{id}(v)$, where $u \in T_i$, $v \in T_j$ for some i and j, iff $\mathrm{L}(u) = \mathrm{L}(v)$.

Frequency difference consensus tree

Previous algorithm:

1. Compute the frequency of every cluster $\mathrm{L}(u)$, where $u \in T_i$ for some i, in $\mathcal{O}(\min\{n,k\} \cdot k \cdot n)$ time.
2. Given the frequency of every cluster, construct the frequency difference consensus tree in additional $\mathcal{O}(k \cdot n \log^2 n)$ time.

Only need to compute identifiers $\mathrm{id}(u)$, where $u \in T_i$ for some i, such that $\mathrm{id}(u) = \mathrm{id}(v)$, where $u \in T_i$, $v \in T_j$ for some i and j, iff $\mathrm{L}(u) = \mathrm{L}(v)$.

Frequency difference consensus tree

Previous algorithm:

1. Compute the frequency of every cluster $\mathsf{L}(u)$, where $u \in T_i$ for some i, in $\mathcal{O}(\min\{n, k\} \cdot k \cdot n)$ time.
2. Given the frequency of every cluster, construct the frequency difference consensus tree in additional $\mathcal{O}(k \cdot n \log^2 n)$ time.

Only need to compute identifiers $\mathrm{id}(u)$, where $u \in T_i$ for some i, such that $\mathrm{id}(u) = \mathrm{id}(v)$, where $u \in T_i$, $v \in T_j$ for some i and j, iff $\mathsf{L}(u) = \mathsf{L}(v)$.

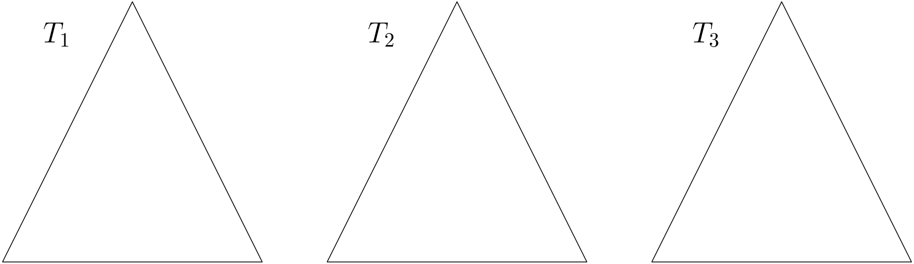
Computing identifiers

We proceed in phases, in the ℓ-th phase assigning ids to all nodes u with $|\mathrm{L}(u)| \in [2^{\ell}, 2^{\ell+1})$, where $u \in T_i$ for some i.

Total number of artificial nodes over all phases = $\mathcal{O}(k \cdot n \log n)$.

Computing identifiers

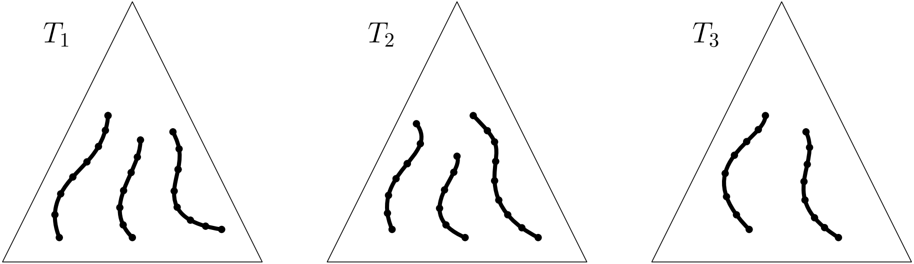
We proceed in phases, in the ℓ-th phase assigning ids to all nodes u with $|\mathsf{L}(u)| \in [2^\ell, 2^{\ell+1})$, where $u \in T_i$ for some i.



Total number of artificial nodes over all phases = $\mathcal{O}(k \cdot n \log n)$.

Computing identifiers

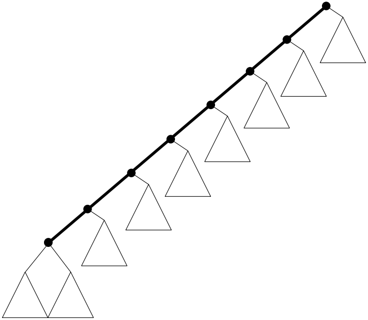
We proceed in phases, in the ℓ-th phase assigning ids to all nodes u with $|\mathsf{L}(u)| \in [2^\ell, 2^{\ell+1})$, where $u \in T_i$ for some i.



Total number of artificial nodes over all phases = $\mathcal{O}(k \cdot n \log n)$.

Computing identifiers

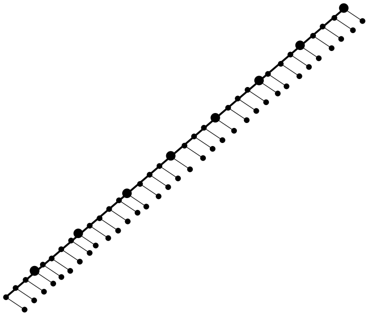
We proceed in phases, in the ℓ-th phase assigning ids to all nodes u with $|\mathrm{L}(u)| \in [2^{\ell}, 2^{\ell+1})$, where $u \in T_i$ for some i.



Total number of artificial nodes over all phases = $\mathcal{O}(k \cdot n \log n)$.

Computing identifiers

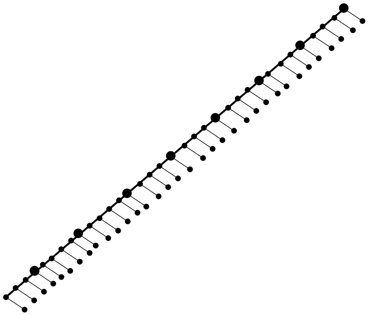
We proceed in phases, in the ℓ-th phase assigning ids to all nodes u with $|\mathsf{L}(u)| \in [2^{\ell}, 2^{\ell+1})$, where $u \in T_i$ for some i.



Total number of artificial nodes over all phases = $\mathcal{O}(k \cdot n \log n)$.

Computing identifiers

We proceed in phases, in the ℓ-th phase assigning ids to all nodes u with $|\mathsf{L}(u)| \in [2^\ell, 2^{\ell+1})$, where $u \in T_i$ for some i.



Total number of artificial nodes over all phases = $\mathcal{O}(k \cdot n \log n)$.

Dynamic set structure

We need to maintain subsets of $[n]$ under:

1. inserting elements,
2. returning the id (a small integer) of the current subset,

so that two subsets are equal iff their ids are the same.

Dynamic set structure

We need to maintain subsets of $[n]$ under:

1. inserting elements,
2. returning the id (a small integer) of the current subset,

so that two subsets are equal iff their ids are the same.

Dynamic set structure

We need to maintain subsets of $[n]$ under:

1. inserting elements,
2. returning the id (a small integer) of the current subset,

so that two subsets are equal iff their ids are the same.

Dynamic set structure

We need to maintain subsets of $[n]$ under:

1. inserting elements,
2. returning the id (a small integer) of the current subset,

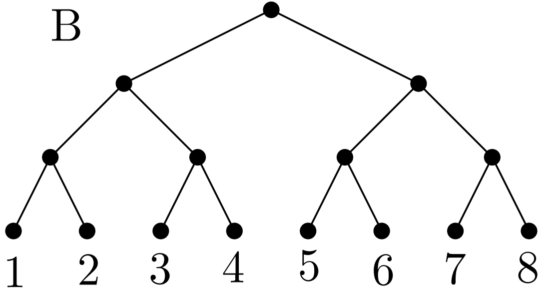
so that two subsets are equal iff their ids are the same.

Dynamic set structure

We need to maintain subsets of $[n]$ under:

1. inserting elements,
2. returning the id (a small integer) of the current subset,

so that two subsets are equal iff their ids are the same.

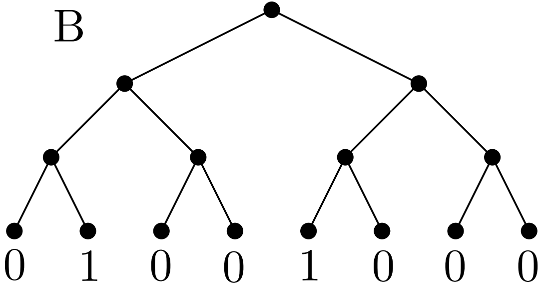


Dynamic set structure

We need to maintain subsets of $[n]$ under:

1. inserting elements,
2. returning the id (a small integer) of the current subset,

so that two subsets are equal iff their ids are the same.

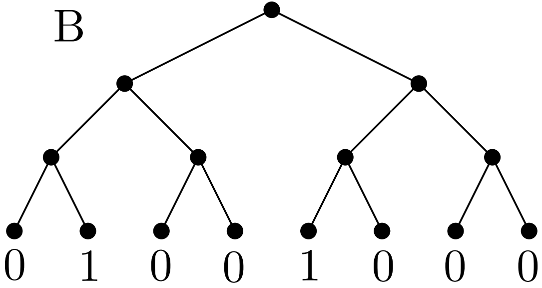


Dynamic set structure

We need to maintain subsets of $[n]$ under:

1. inserting elements,
2. returning the id (a small integer) of the current subset,

so that two subsets are equal iff their ids are the same.



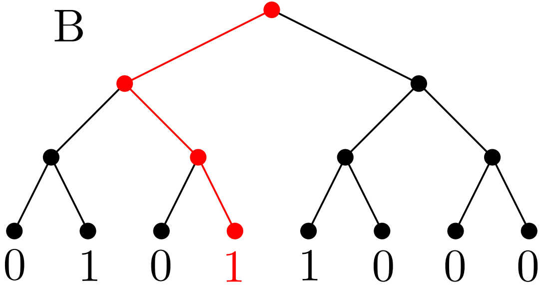
Maintain the ids of the intersections with every range corresponding to a node of B.

Dynamic set structure

We need to maintain subsets of $[n]$ under:

1. inserting elements,
2. returning the id (a small integer) of the current subset,

so that two subsets are equal iff their ids are the same.



Maintain the ids of the intersections with every range corresponding to a node of B.

Dynamic set structure

1. B is implemented persistently, so after an insert we need to recompute only $\log n$ identifiers.
2. Every identifier can be calculated using the identifiers of its two children, we need to store the mapping in a BST to make sure that two subsets are equal iff their ids are the same.
3. This would give us insertions $\mathcal{O}(\log^2 n)$, so $\mathcal{O}(k \cdot n \log^3 n)$ overall.
4. ...however, we know all insertions in advance! Instead of a BST, we process them together and use radix sort in $\mathcal{O}(k \cdot n \log^2 n)$ time.

Dynamic set structure

1. B is implemented persistently, so after an insert we need to recompute only $\log n$ identifiers.
2. Every identifier can be calculated using the identifiers of its two children, we need to store the mapping in a BST to make sure that two subsets are equal iff their ids are the same.
3. This would give us insertions $\mathcal{O}(\log^2 n)$, so $\mathcal{O}(k \cdot n \log^3 n)$ overall.
4. ...however, we know all insertions in advance! Instead of a BST, we process them together and use radix sort in $\mathcal{O}(k \cdot n \log^2 n)$ time.

Dynamic set structure

1. B is implemented persistently, so after an insert we need to recompute only $\log n$ identifiers.
2. Every identifier can be calculated using the identifiers of its two children, we need to store the mapping in a BST to make sure that two subsets are equal iff their ids are the same.
3. This would give us insertions $\mathcal{O}(\log^2 n)$, so $\mathcal{O}(k \cdot n \log^3 n)$ overall.
4. ...however, we know all insertions in advance! Instead of a BST, we process them together and use radix sort in $\mathcal{O}(k \cdot n \log^2 n)$ time.

Dynamic set structure

1. B is implemented persistently, so after an insert we need to recompute only $\log n$ identifiers.
2. Every identifier can be calculated using the identifiers of its two children, we need to store the mapping in a BST to make sure that two subsets are equal iff their ids are the same.
3. This would give us insertions $\mathcal{O}(\log^2 n)$, so $\mathcal{O}(k \cdot n \log^3 n)$ overall.
4. ...however, we know all insertions in advance! Instead of a BST, we process them together and use radix sort in $\mathcal{O}(k \cdot n \log^2 n)$ time.

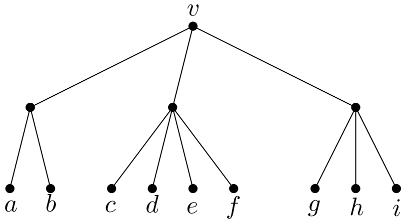
Greedy consensus tree

We consider the clusters $\mathrm{L}(u)$, where $u \in T_i$ for some i in the appropriate order and maintain the current tree T_c. We need to:

1. Efficiently check if $\mathrm{L}(u)$ is compatible with all clusters of T_c,
2. if so update T_c.

Updating T_c

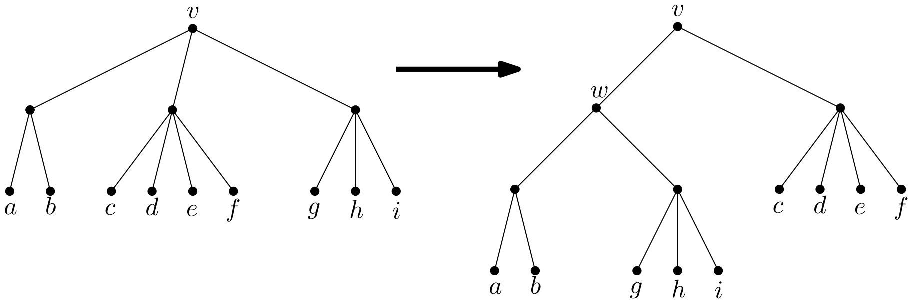
Adding $\{a, b, g, h, i\}$:



1. We always need to add a new child v' to some node v and reconnect some of the children of v to v'.
2. We implement this in time proportional to $\min\{\#\text{ reconnected children}, \#\text{ not reconnected children}\}$.
3. Then, the overall complexity of updating T_c is $O(n \log n)$.

Updating T_c

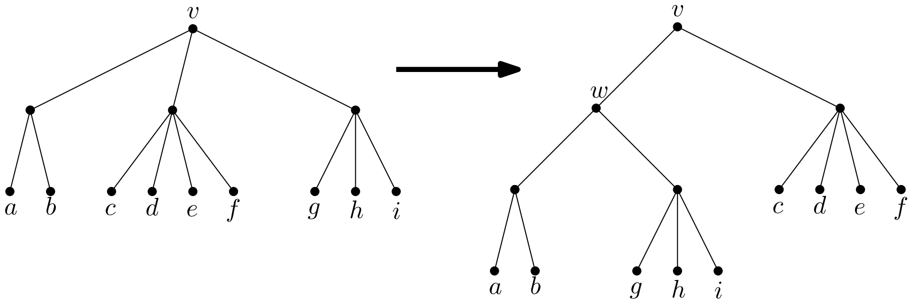
Adding $\{a, b, g, h, i\}$:



1. We always need to add a new child v' to some node v and reconnect some of the children of v to v'.
2. We implement this in time proportional to $\min\{\#\text{ reconnected children}, \#\text{ not reconnected children}\}$.
3. Then, the overall complexity of updating T_c is $O(n \log n)$.

Updating T_c

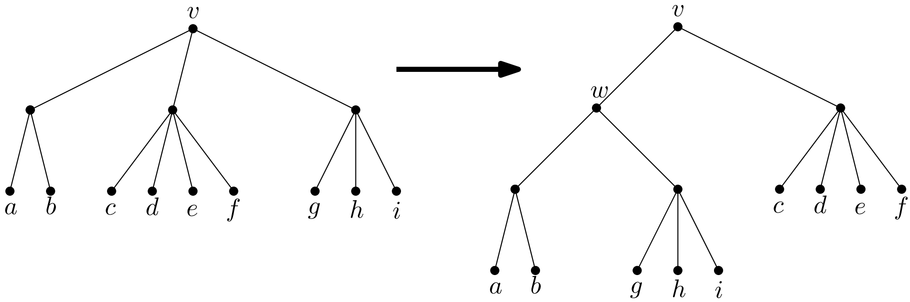
Adding $\{a, b, g, h, i\}$:



1. We always need to add a new child v' to some node v and reconnect some of the children of v to v'.
2. We implement this in time proportional to $\min\{\#\text{ reconnected children}, \#\text{ not reconnected children}\}$.
3. Then, the overall complexity of updating T_c is $\mathcal{O}(n \log n)$.

Updating T_c

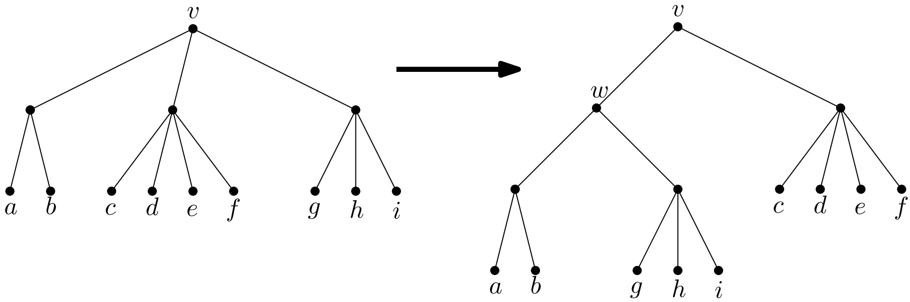
Adding $\{a, b, g, h, i\}$:



1. We always need to add a new child v' to some node v and reconnect some of the children of v to v'.
2. We implement this in time proportional to $\min\{\#\text{ reconnected children}, \#\text{ not reconnected children}\}$.
3. Then, the overall complexity of updating T_c is $\mathcal{O}(n \log n)$.

Updating T_c

Adding $\{a, b, g, h, i\}$:



1. We always need to add a new child v' to some node v and reconnect some of the children of v to v'.
2. We implement this in time proportional to $\min\{\#\text{ reconnected children}, \#\text{ not reconnected children}\}$.
3. Then, the overall complexity of updating T_c is $\mathcal{O}(n \log n)$.

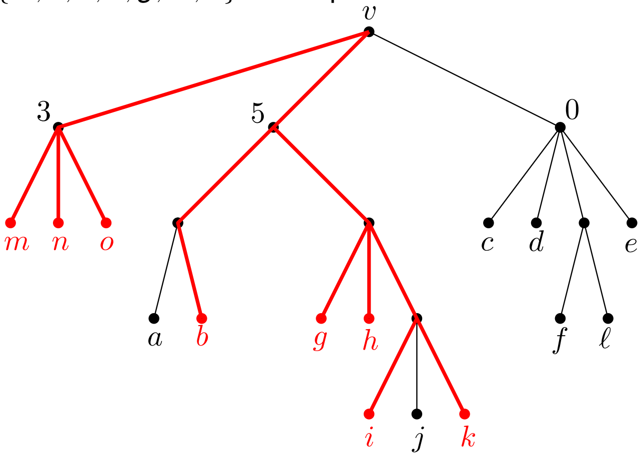
Checking if L(u) is compatible with all clusters of T_c

Checking $\{m, n, o, b, g, hi, k\}$ is compatible with all clusters of T_c:

We essentially need to compute the LCA of all leaves labeled with $x \in$ L(u).

Checking if L(u) is compatible with all clusters of T_c

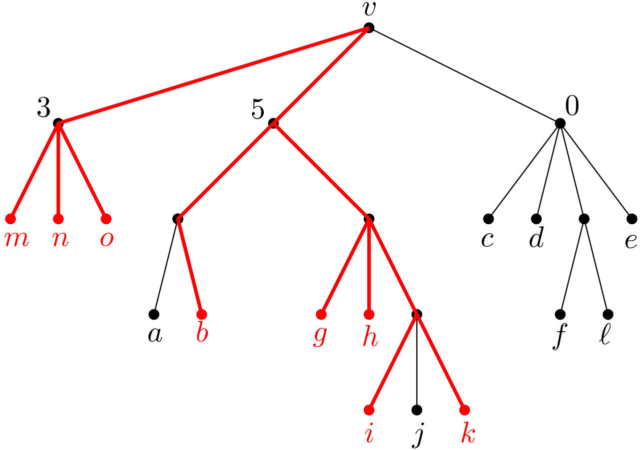
Checking $\{m, n, o, b, g, hi, k\}$ is compatible with all clusters of T_c:



We essentially need to compute the LCA of all leaves labeled with $x \in$ L(u).

Checking if L(u) is compatible with all clusters of T_c

Checking $\{m, n, o, b, g, hi, k\}$ is compatible with all clusters of T_c:



We essentially need to compute the LCA of all leaves labeled with $x \in$ L(u).

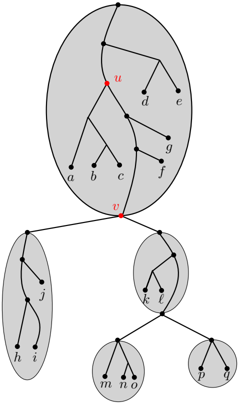
Avoiding considering every $x \in \mathrm{L}(u)$

We apply micro-macro decomposition of every T_i into $\mathcal{O}(n^{0.5})$ micro-trees of size $\mathcal{O}(n^{0.5})$:

We maintain the LCA for all leaves in a subtree of every boundary node. This requires some bookkeeping.

Avoiding considering every $x \in \mathrm{L}(u)$

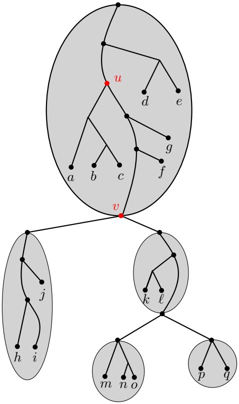
We apply micro-macro decomposition of every T_i into $\mathcal{O}(n^{0.5})$ micro-trees of size $\mathcal{O}(n^{0.5})$:



We maintain the LCA for all leaves in a subtree of every boundary node. This requires some bookkeeping.

Avoiding considering every $x \in \mathrm{L}(u)$

We apply micro-macro decomposition of every T_i into $\mathcal{O}(n^{0.5})$ micro-trees of size $\mathcal{O}(n^{0.5})$:



We maintain the LCA for all leaves in a subtree of every boundary node. This requires some bookkeeping.

1. Is there an $\tilde{\mathcal{O}}(k \cdot n)$ algorithm, maybe by using multiple levels of micro-macro decomposition?
2. ...or is there a conditional lower bound?

Questions?

1. Is there an $\tilde{\mathcal{O}}(k \cdot n)$ algorithm, maybe by using multiple levels of micro-macro decomposition?
2. ...or is there a conditional lower bound?

Questions?

1. Is there an $\tilde{\mathcal{O}}(k \cdot n)$ algorithm, maybe by using multiple levels of micro-macro decomposition?
2. ...or is there a conditional lower bound?

Questions?