On selecting leaves with disjoint neighborhoods in embedded trees

Kolja Junginger **Ioannis Mantas** Evanthia Papadopoulou

Faculty of Informatics, USI Università della Svizzera italiana, Lugano, Switzerland

14/2/2019 - IIT Kharagpur, India - CALDAM 2019

Introduction

This work focuses on a **generalization** of a **combinatorial result** by A. Aggarwal, L. Giubas, J. Saxe and P. Shor [DCG 1987].

Given an embedded tree, the goal is to select in **linear time** a **constant fraction** of the leaves.

Introduction

This work focuses on a **generalization** of a **combinatorial result** by A. Aggarwal, L. Giubas, J. Saxe and P. Shor [DCG 1987].

Given an embedded tree, the goal is to select in **linear time** a **constant fraction** of the leaves.

Part of an algorithm to construct in deterministic **linear time** the: **Voronoi Diagram of points in convex position**, given the convex hull.

Introduction

This work focuses on a **generalization** of a **combinatorial result** by A. Aggarwal, L. Giubas, J. Saxe and P. Shor [DCG 1987].

Given an embedded tree, the goal is to select in **linear time** a **constant fraction** of the leaves.

Part of an algorithm to construct in deterministic **linear time** the: **Voronoi Diagram of points in convex position**, given the convex hull.

Can also be extended to other Voronoi diagrams with **tree structure**:

- → Farthest point VD, given the convex hull.
- → Update of a VD, after deleting a point.
- → Order-k VD, given the order-(k-1) VD.

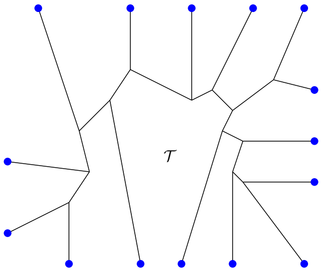
Applications

- The algorithmic scheme has been used to derive linear time algorithms for many problems, e.g.:
  - → **Medial axis** of a simple polygon in $O(n)$.
    [Chin et al. - DCG 1999]
  - → **Order-**k VD in $O(nk^2 + n \log n)$.
    [D.T. Lee - IEEE Trans. Comput. 1982]
  - → **Hamiltonian Abstract** VD in $O(n)$.
    [Klein and Lingas - ISAAC 1994]
  - → **Forest-like Abstract** VD in $O(n)$.
    [Bohler et al. - Comp. Geom. 2014]

Combinatorial result

Theorem [Aggarwal et al. 1987]

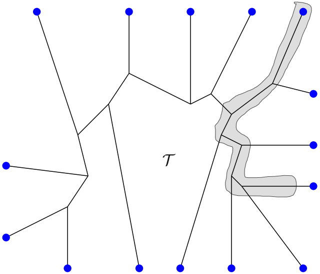
Let $\mathcal{T}$ be an embedded binary tree with n leaves



Combinatorial result

Theorem [Aggarwal et al. 1987]

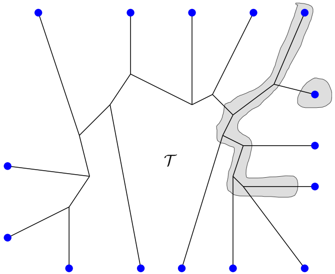
Let $\mathcal{T}$ be an embedded binary tree with n leaves where:
i) Each leaf of $\mathcal{T}$ has a neighborhood - *(a subtree of $\mathcal{T}$)*.



Combinatorial result

Theorem [Aggarwal et al. 1987]

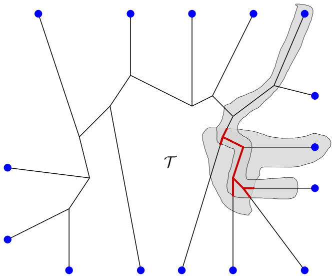
Let $\mathcal{T}$ be an embedded binary tree with n leaves where:
i) Each leaf of $\mathcal{T}$ has a neighborhood - *(a subtree of $\mathcal{T}$)*.
ii) Topologically consecutive leaves have disjoint neighborhoods.



Combinatorial result

Theorem [Aggarwal et al. 1987]

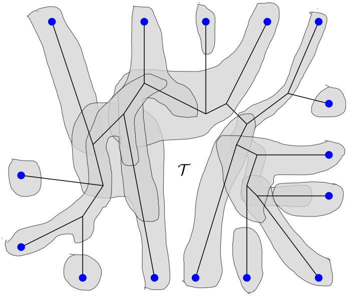
Let $\mathcal{T}$ be an embedded binary tree with n leaves where:
i) Each leaf of $\mathcal{T}$ has a neighborhood - *(a subtree of $\mathcal{T}$)*.
ii) Topologically consecutive leaves have disjoint neighborhoods.



Combinatorial result

Theorem [Aggarwal et al. 1987]

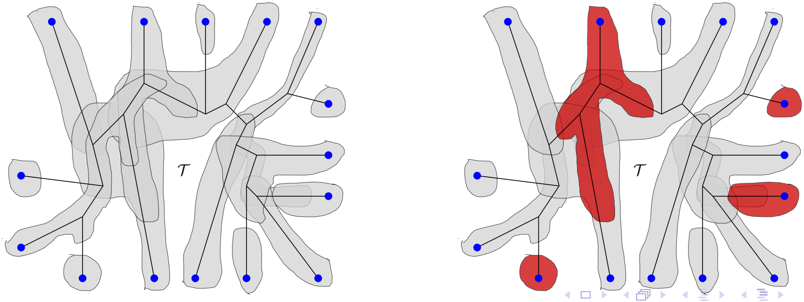
Let $\mathcal{T}$ be an embedded binary tree with n leaves where:
i) Each leaf of $\mathcal{T}$ has a neighborhood - *(a subtree of $\mathcal{T}$)*.
ii) Topologically consecutive leaves have disjoint neighborhoods.



Combinatorial result

Theorem [Aggarwal et al. 1987]

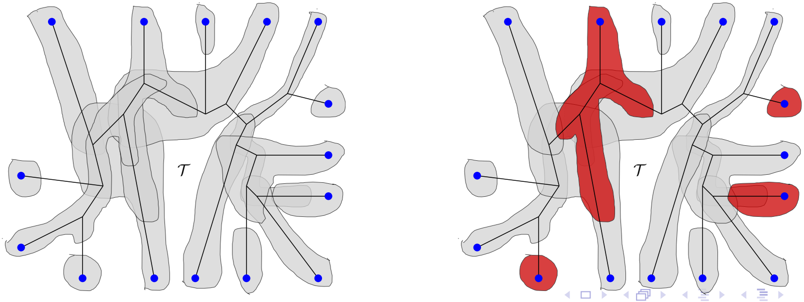
Let $\mathcal{T}$ be an embedded binary tree with n leaves where:
i) Each leaf of $\mathcal{T}$ has a neighborhood - *(a subtree of $\mathcal{T}$)*.
ii) Topologically consecutive leaves have disjoint neighborhoods.
Then:
i) $\exists \geq \frac{1}{10}n$ leaves with pairwise disjoint neighborhoods.



Combinatorial result

Theorem [Aggarwal et al. 1987]

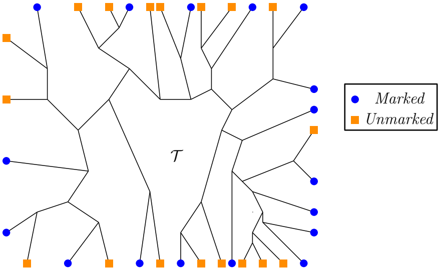
Let $\mathcal{T}$ be an embedded binary tree with n leaves where:
i) Each leaf of $\mathcal{T}$ has a neighborhood - *(a subtree of $\mathcal{T}$)*.
ii) Topologically consecutive leaves have disjoint neighborhoods.
Then:
i) $\exists \geq \frac{1}{10}n$ leaves with pairwise disjoint neighborhoods.
ii) These leaves can be found in $O(n)$ time.



Our result

Theorem - Generalized

Let $\mathcal{T}$ be an embedded binary tree with n leaves where:
i) m of the leaves have been marked.



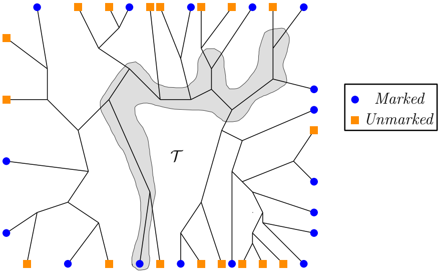
Our result

Theorem - Generalized

Let $\mathcal{T}$ be an embedded binary tree with n leaves where:

i) m of the leaves have been marked.

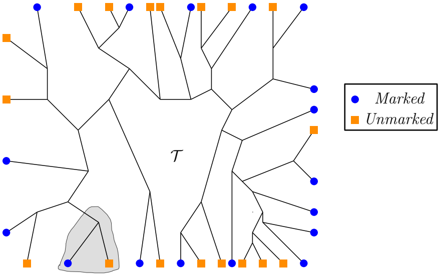
ii) Each marked leaf of $\mathcal{T}$ has a neighborhood.



Our result

Theorem - Generalized

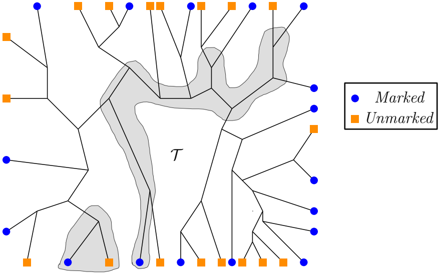
Let $\mathcal{T}$ be an embedded binary tree with n leaves where:
i) m of the leaves have been marked.
ii) Each marked leaf of $\mathcal{T}$ has a neighborhood.



Our result

Theorem - Generalized

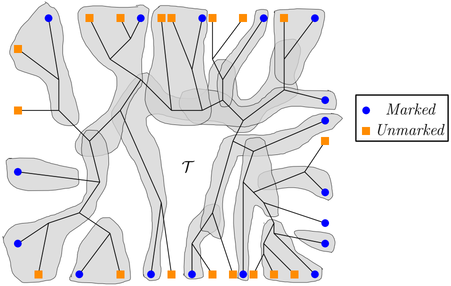
Let $\mathcal{T}$ be an embedded binary tree with n leaves where:
i) m of the leaves have been marked.
ii) Each marked leaf of $\mathcal{T}$ has a neighborhood.
iii) Topologically consecutive marked leaves have disjoint neighborhoods.



Our result

Theorem - Generalized

Let $\mathcal{T}$ be an embedded binary tree with n leaves where:
i) *m* of the leaves have been marked.
ii) Each marked leaf of $\mathcal{T}$ has a neighborhood.
iii) Topologically consecutive marked leaves have disjoint neighborhoods.



Our result

Theorem - Generalized

Let $\mathcal{T}$ be an embedded binary tree with n leaves where:

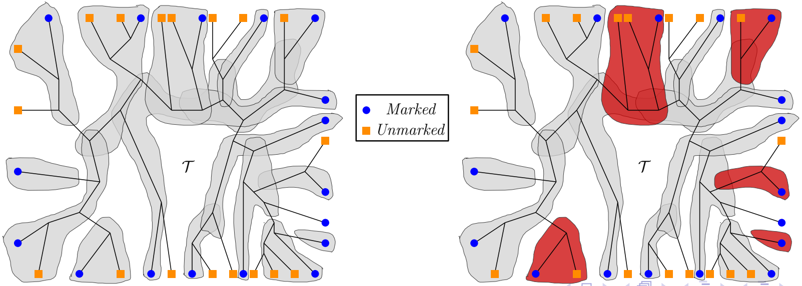
i) m of the leaves have been marked.

ii) Each marked leaf of $\mathcal{T}$ has a neighborhood.

iii) Topologically consecutive marked leaves have disjoint neighborhoods.

Then:

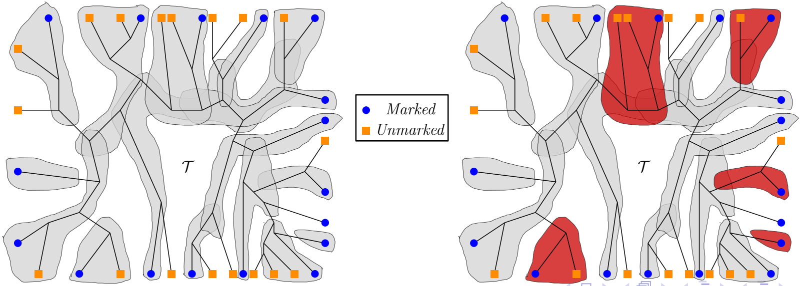
i) $\exists \geq \frac{1}{10}m$ marked leaves with pairwise disjoint neighborhoods.



Our result

Theorem - Generalized

Let $\mathcal{T}$ be an embedded binary tree with n leaves where:
i) m of the leaves have been marked.
ii) Each marked leaf of $\mathcal{T}$ has a neighborhood.
iii) Topologically consecutive marked leaves have disjoint neighborhoods.
Then:
i) $\exists \geq \frac{1}{10} m$ marked leaves with pairwise disjoint neighborhoods.
ii) $\geq \frac{p}{10} m$ marked leaves can be found in $O(\frac{1}{1-p} n)$ time, for any $p \in (0, 1)$.



Our result

Theorem - Generalized

Let $\mathcal{T}$ be an embedded binary tree with n leaves where:
i) m of the leaves have been marked.
ii) Each marked leaf of $\mathcal{T}$ has a neighborhood.
iii) Topologically consecutive marked leaves have disjoint neighborhoods.
Then:
i) $\exists \geq \frac{1}{10}m$ marked leaves with pairwise disjoint neighborhoods.
ii) $\geq \frac{p}{10}m$ marked leaves can be found in $O(\frac{1}{1-p}n)$ time, for any $p \in (0,1)$.

Remarks:

- If the solution is required to be a constant fraction of m, then it suffices to choose any constant for $p \in (0,1)$.

Our result

Theorem - Generalized

Let $\mathcal{T}$ be an embedded binary tree with n leaves where:
i) m of the leaves have been marked.
ii) Each marked leaf of $\mathcal{T}$ has a neighborhood.
iii) Topologically consecutive marked leaves have disjoint neighborhoods.
Then:
i) $\exists \geq \frac{1}{10}m$ marked leaves with pairwise disjoint neighborhoods.
ii) $\geq \frac{p}{10}m$ marked leaves can be found in $O(\frac{1}{1-p}n)$ time, for any $p \in (0,1)$.

Remarks:

- If the solution is required to be a constant fraction of m, then it suffices to choose any constant for $p \in (0,1)$.
- If p is a constant, then the algorithm has $O(n)$ time complexity.

Motivation

Linear-time algorithms for problems mentioned *(e.g. deletion of a site, construction of order-k, etc.)* remain open for:

- **Voronoi diagram of non-point sites**
  ...even for simple sites as circles, line segments, etc.
- **Abstract Voronoi diagrams**

Motivation

Linear-time algorithms for problems mentioned *(e.g. deletion of a site, construction of order-k, etc.)* remain open for:

- **Voronoi diagram of non-point sites**
  ...even for simple sites as circles, line segments, etc.
- **Abstract Voronoi diagrams**

Recent work on **randomized linear constructions** of these diagrams:

→ Construction of the farthest line-segment VD.
[Khramtcova & Papadopoulou - arXiv 2017]

→ Update of an abstract VD, after the deletion of a site.
[Junginger & Papadopoulou - SoCG 2018]

Motivation

Linear-time algorithms for problems mentioned *(e.g. deletion of a site, construction of order-k, etc.)* remain open for:

- **Voronoi diagram of non-point sites**
  ...even for simple sites as circles, line segments, etc.
- **Abstract Voronoi diagrams**

Recent work on **randomized linear constructions** of these diagrams:

→ Construction of the farthest line-segment VD.
[Khramtcova & Papadopoulou - arXiv 2017]

→ Update of an abstract VD, after the deletion of a site.
[Junginger & Papadopoulou - SoCG 2018]

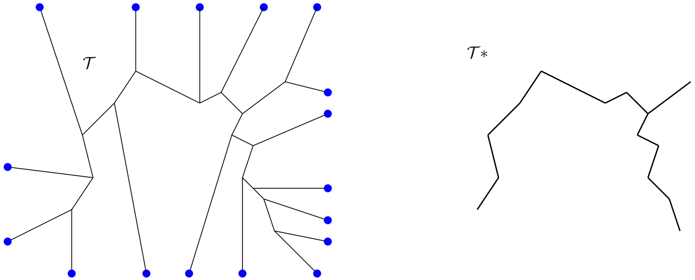
Suggests that, to potentially apply the linear-time framework...
... We first need this **generalized combinatorial result.**

Outline of results

1. Present some necessary **preliminaries**.
2. Show the first part of the theorem, the **existence**.
3. Show the second part of the theorem, the **algorithm**.

Labeling the nodes [Aggarwal et al. 1987]

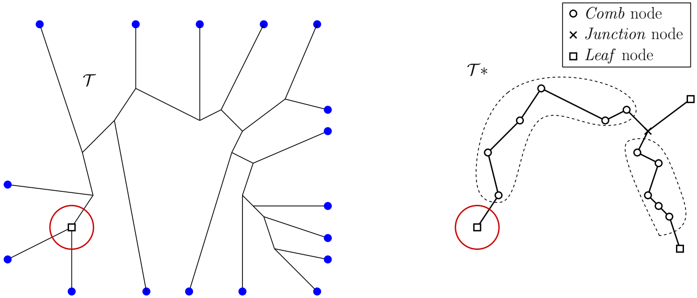
Let $\mathcal{T}^*$ be the tree obtained after **deleting all leaves** from $\mathcal{T}$.



Labeling the nodes [Aggarwal et al. 1987]

Let $\mathcal{T}^*$ be the tree obtained after **deleting all leaves** from $\mathcal{T}$.
A node $u \in \mathcal{T}$ is called:

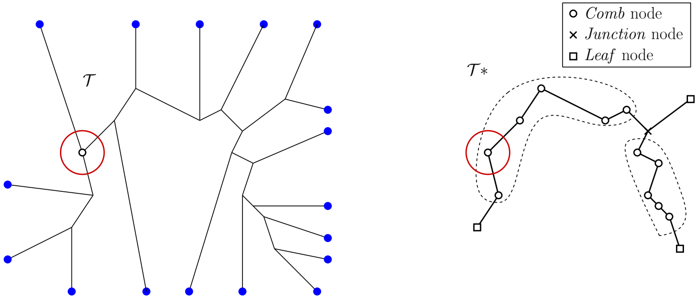
- **Leaf** node if $deg(u) = 1$ in $\mathcal{T}^*$.



Labeling the nodes [Aggarwal et al. 1987]

Let $\mathcal{T}^*$ be the tree obtained after **deleting all leaves** from $\mathcal{T}$.
A node $u \in \mathcal{T}$ is called:

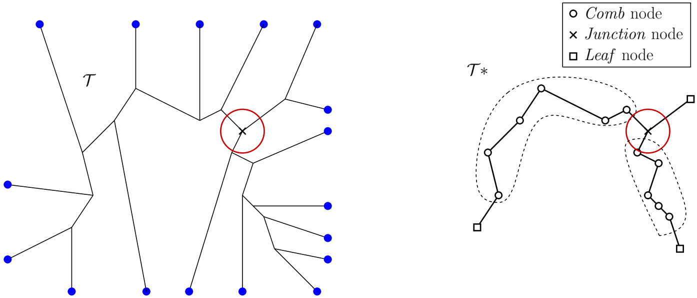
- **Leaf** node if $deg(u) = 1$ in $\mathcal{T}^*$.
- **Comb** node if $deg(u) = 2$ in $\mathcal{T}^*$.



Labeling the nodes [Aggarwal et al. 1987]

Let $\mathcal{T}^*$ be the tree obtained after **deleting all leaves** from $\mathcal{T}$.
A node $u \in \mathcal{T}$ is called:

- **Leaf** node if $deg(u) = 1$ in $\mathcal{T}^*$.
- **Comb** node if $deg(u) = 2$ in $\mathcal{T}^*$.
- **Junction** node if $deg(u) = 3$ in $\mathcal{T}^*$.

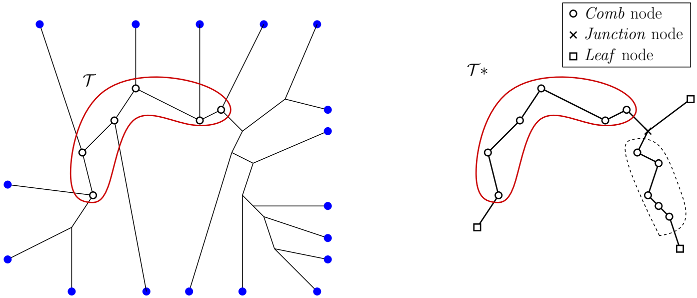


Labeling the nodes [Aggarwal et al. 1987]

Let $\mathcal{T}^*$ be the tree obtained after **deleting all leaves** from $\mathcal{T}$.
A node $u \in \mathcal{T}$ is called:

- **Leaf** node if $deg(u) = 1$ in $\mathcal{T}^*$.
- **Comb** node if $deg(u) = 2$ in $\mathcal{T}^*$.
- **Junction** node if $deg(u) = 3$ in $\mathcal{T}^*$.

A **spine** is a maximal sequence of consecutive *Comb* nodes.

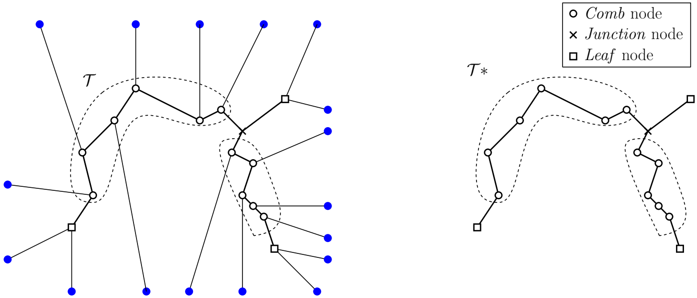


Labeling the nodes [Aggarwal et al. 1987]

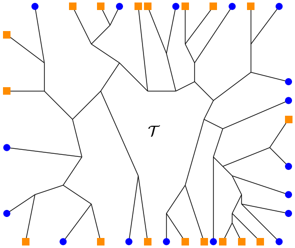
Let $\mathcal{T}^*$ be the tree obtained after **deleting all leaves** from $\mathcal{T}$.
A node $u \in \mathcal{T}$ is called:

- **Leaf** node if $deg(u) = 1$ in $\mathcal{T}^*$.
- **Comb** node if $deg(u) = 2$ in $\mathcal{T}^*$.
- **Junction** node if $deg(u) = 3$ in $\mathcal{T}^*$.

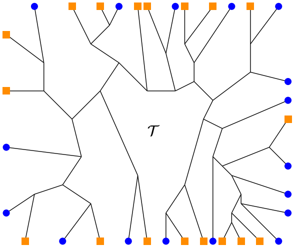
A **spine** is a maximal sequence of consecutive *Comb* nodes.



Labeling the tree $\mathcal{T}$

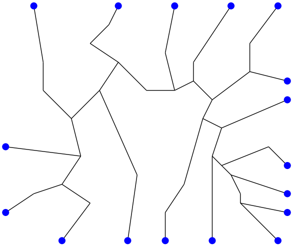


Labeling the tree $\mathcal{T}$

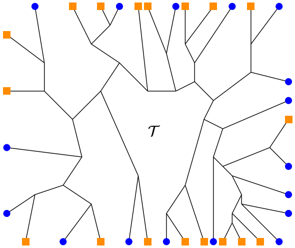


Transformation:

1. Delete unmarked leaves.

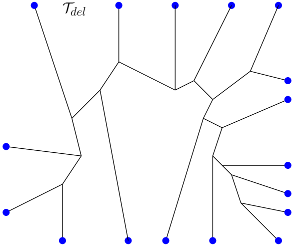


Labeling the tree $\mathcal{T}$

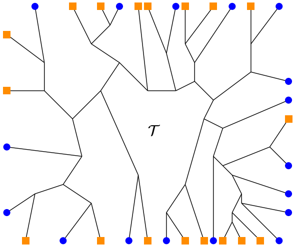


Transformation:

1. Delete unmarked leaves.
2. Contract degree 2 nodes. Obtain tree $\mathcal{T}_{del}$.

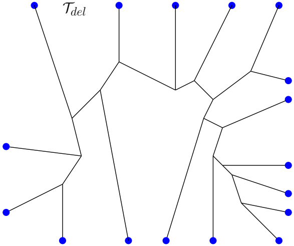


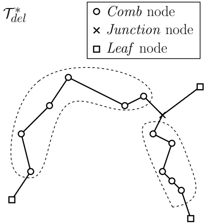
Labeling the tree $\mathcal{T}$



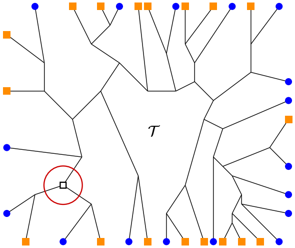
Transformation:

1. Delete unmarked leaves.
2. Contract degree 2 nodes. Obtain tree $\mathcal{T}_{del}$.
3. Use $\mathcal{T}^*_{del}$ to characterize nodes of $\mathcal{T}$.





Labeling the tree $\mathcal{T}$

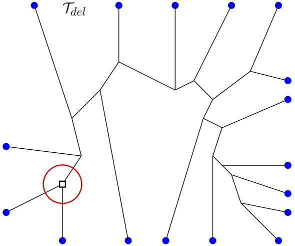


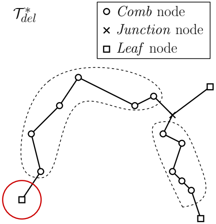
Transformation:

3. Use $\mathcal{T}^*_{del}$ to characterize nodes of $\mathcal{T}$.

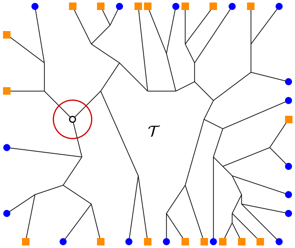
A node $u \in \mathcal{T}$ is called:

- Leaf: if $u \in \mathcal{T}^*_{del}$ and $deg(u) = 1$ in $\mathcal{T}^*_{del}$.





Labeling the tree $\mathcal{T}$

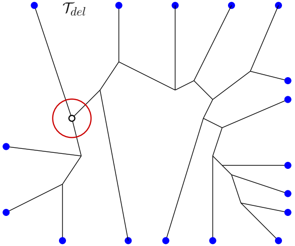


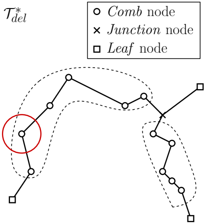
Transformation:

3. Use $\mathcal{T}^*_{del}$ to characterize nodes of $\mathcal{T}$.

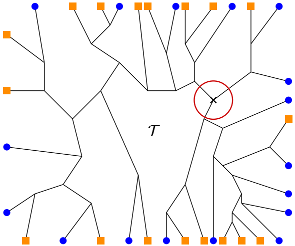
A node $u \in \mathcal{T}$ is called:

- Leaf: if $u \in \mathcal{T}^*_{del}$ and $deg(u) = 1$ in $\mathcal{T}^*_{del}$.
- Comb: if $u \in \mathcal{T}^*_{del}$ and $deg(u) = 2$ in $\mathcal{T}^*_{del}$.





Labeling the tree $\mathcal{T}$

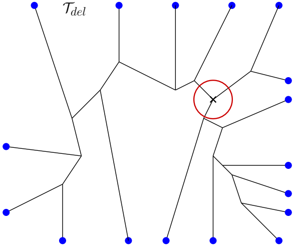


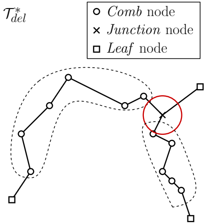
Transformation:

3. Use $\mathcal{T}^*_{del}$ to characterize nodes of $\mathcal{T}$.

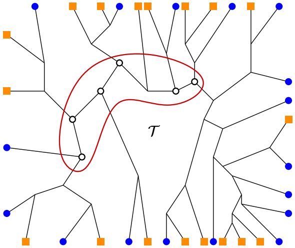
A node $u \in \mathcal{T}$ is called:

- Leaf: if $u \in \mathcal{T}^*_{del}$ and $deg(u) = 1$ in $\mathcal{T}^*_{del}$.
- Comb: if $u \in \mathcal{T}^*_{del}$ and $deg(u) = 2$ in $\mathcal{T}^*_{del}$.
- Junction: if $u \in \mathcal{T}^*_{del}$ and $deg(u) = 3$ in $\mathcal{T}^*_{del}$.





Labeling the tree $\mathcal{T}$



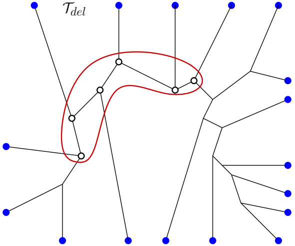
Transformation:

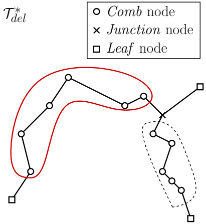
3. Use $\mathcal{T}^*_{del}$ to characterize nodes of $\mathcal{T}$.

A node $u \in \mathcal{T}$ is called:

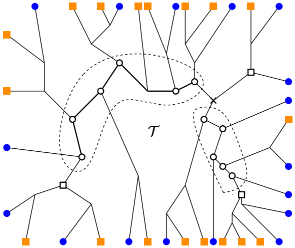
- Leaf: if $u \in \mathcal{T}^*_{del}$ and $deg(u) = 1$ in $\mathcal{T}^*_{del}$.
- Comb: if $u \in \mathcal{T}^*_{del}$ and $deg(u) = 2$ in $\mathcal{T}^*_{del}$.
- Junction: if $u \in \mathcal{T}^*_{del}$ and $deg(u) = 3$ in $\mathcal{T}^*_{del}$.

Spine: A sequence of consecutive Comb nodes.

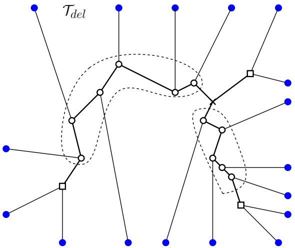


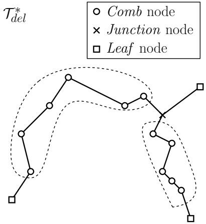


Labeling the tree $\mathcal{T}$

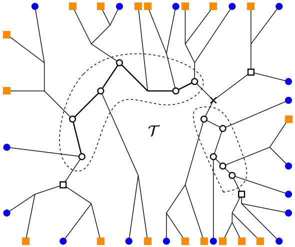


Remark:
Original: **All** internal nodes get labeled.
Generalized: Only **a subset** of the internal nodes get labeled.



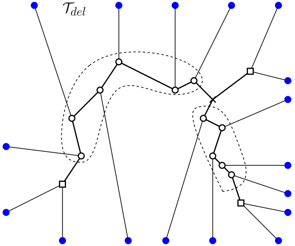


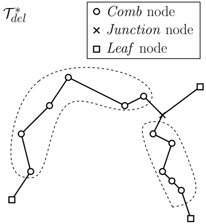
Labeling the tree $\mathcal{T}$



Remark:
Original: **All** internal nodes get labeled.
Generalized: Only **a subset** of the internal nodes get labeled.

Idea:
Pass the information of the marked leaves to **a subset of $\mathcal{T}$** to resemble [Aggarwal et al. 1987].





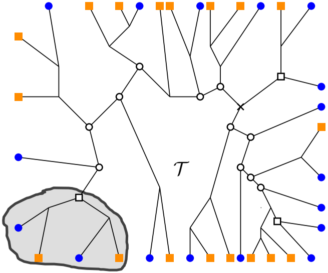
Components

Define **two types of components**, which are subtrees of $\mathcal{T}$.

Components

Define **two types of components**, which are subtrees of $\mathcal{T}$.

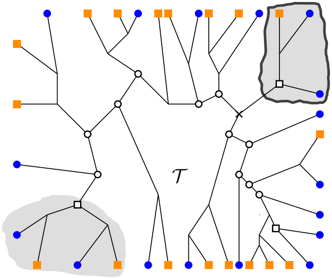
L-component: *The Leaf node and the* 2 *subtrees hanging off that node.*



Components

Define **two types of components**, which are subtrees of $\mathcal{T}$.

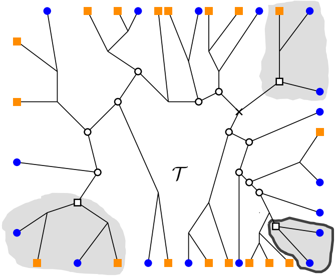
L-component: *The Leaf node and the 2 subtrees hanging off that node.*



Components

Define **two types of components**, which are subtrees of $\mathcal{T}$.

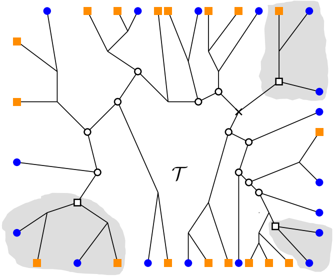
L-component: *The Leaf node and the 2 subtrees hanging off that node.*



Components

Define **two types of components**, which are subtrees of $\mathcal{T}$.

L-component: *The Leaf node and the 2 subtrees hanging off that node.*

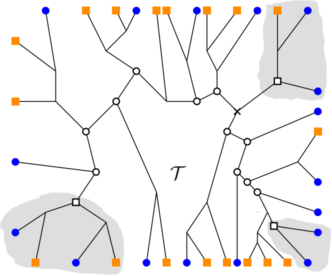


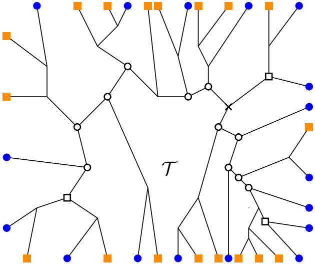
Components

Define **two types of components**, which are subtrees of $\mathcal{T}$.

L-component: *The Leaf node and the* 2 *subtrees hanging off that node.*

Subdivide each spine into groups of 5 Comb nodes.





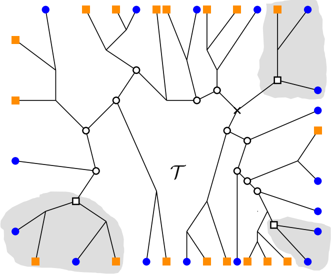
Components

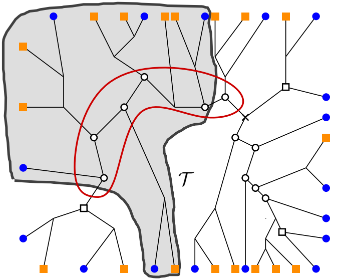
Define **two types of components**, which are subtrees of $\mathcal{T}$.

L-component: *The Leaf node and the 2 subtrees hanging off that node.*

Subdivide each spine into groups of 5 Comb nodes.

5-component: *Part of the spine containing the 5 Comb nodes and the subtress hanging off that.*

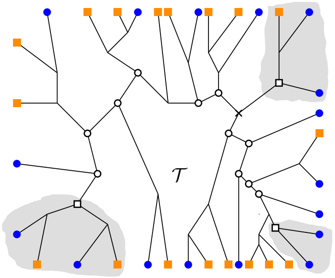




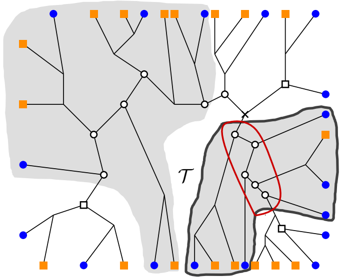
Components

Define **two types of components**, which are subtrees of $\mathcal{T}$.

L-component: *The Leaf node and the* 2 *subtrees hanging off that node.*



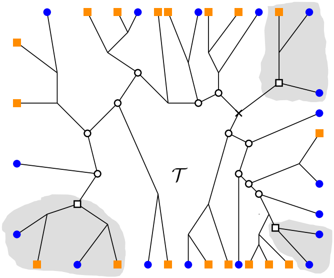
Subdivide each spine into groups of 5 Comb nodes.
5-component: *Part of the spine containing the* 5 *Comb nodes and the subtress hanging off that.*



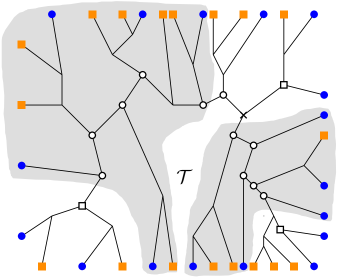
Components

Define **two types of components**, which are subtrees of $\mathcal{T}$.

L-component: *The Leaf node and the 2 subtrees hanging off that node.*



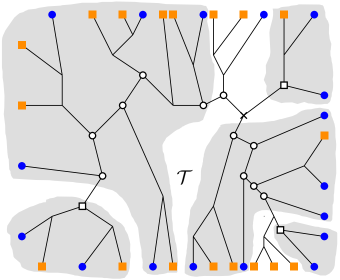
Subdivide each spine into groups of 5 Comb nodes.
5-component: *Part of the spine containing the 5 Comb nodes and the subtress hanging off that.*



Components

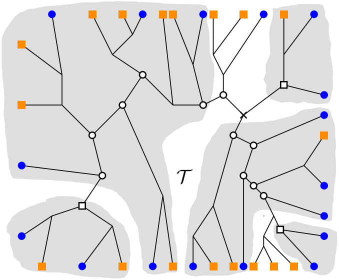
Define **two types of components**, which are subtrees of $\mathcal{T}$.

L-component: *The Leaf node and the 2 subtrees hanging off that node.*



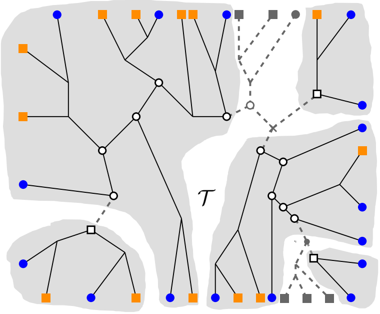
Subdivide each spine into groups of 5 Comb nodes.

5-component: *Part of the spine containing the 5 Comb nodes and the subtress hanging off that.*



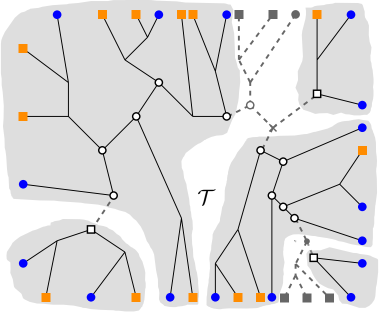
Components

- **L-component**
- **5-component**



Components

- **L-component**
- **5-component**

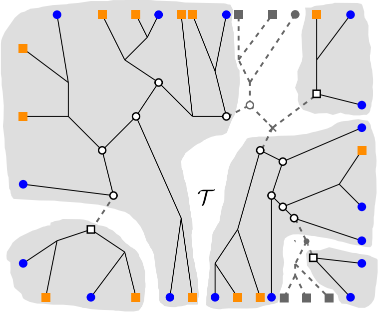


Observations:

- Components are disjoint subtrees of $\mathcal{T}$.

Components

- **L-component**
- **5-component**

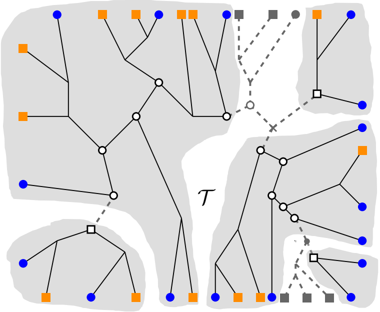


Observations:

- Components are disjoint subtrees of $\mathcal{T}$.
- Each L-component has 2 marked leaves.
- Each 5-component has 5 marked leaves.

Components

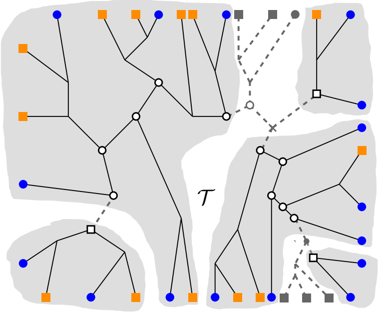
- **L-component**
- **5-component**



Observations:

- Components are disjoint subtrees of $\mathcal{T}$.
- Each L-component has 2 marked leaves.
- Each 5-component has 5 marked leaves.
- Not every node belongs to a component.

Components



- **L-component**
- **5-component**

Observations:

- Components are disjoint subtrees of $\mathcal{T}$.
- Each L-component has 2 marked leaves.
- Each 5-component has 5 marked leaves.
- Not every node belongs to a component.
- A component can have $\Theta(n)$ nodes.

Existence

We want to prove:

Lemma - Existence

At least $\frac{1}{10}m$ marked leaves of $\mathcal{T}$ have pairwise disjoint neighborhoods.

Existence proof

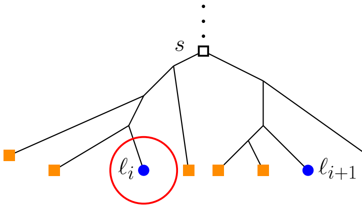
Lemma 1

In every component, there a exists at least one marked leaf ℓ with neighborhood $nh(\ell)$ confined to that component.

Existence proof

Lemma 1

In every component, there a exists at least one marked leaf ℓ with neighborhood $nh(\ell)$ confined to that component.

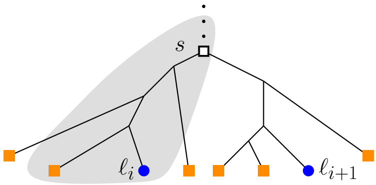


For an **L-component**: consider $nh(\ell_i)$.

Existence proof

Lemma 1

In every component, there a exists at least one marked leaf ℓ with neighborhood $nh(\ell)$ confined to that component.



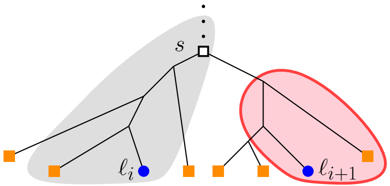
For an **L-component**:
consider $nh(\ell_i)$.

Case 1: If Leaf node $s \in nh(\ell_i)$

Existence proof

Lemma 1

In every component, there a exists at least one marked leaf ℓ with neighborhood $nh(\ell)$ confined to that component.



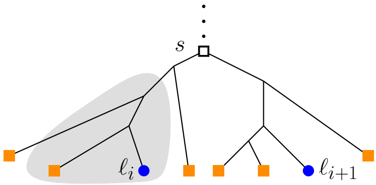
For an **L-component**: consider $nh(\ell_i)$.

Case 1: If Leaf node $s \in nh(\ell_i)$ $\Rightarrow nh(\ell_{i+1})$ is confined.

Existence proof

Lemma 1

In every component, there a exists at least one marked leaf ℓ with neighborhood $nh(\ell)$ confined to that component.



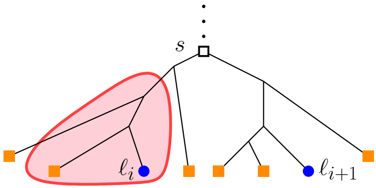
For an **L-component**: consider $nh(\ell_i)$.

Case 2: If Leaf node $s \notin nh(\ell_i)$

Existence proof

Lemma 1

In every component, there a exists at least one marked leaf ℓ with neighborhood $nh(\ell)$ confined to that component.



For an **L-component**: consider $nh(\ell_i)$.

Case 2: If Leaf node $s \notin nh(\ell_i)$ $\Rightarrow nh(\ell_i)$ is confined.

Existence proof

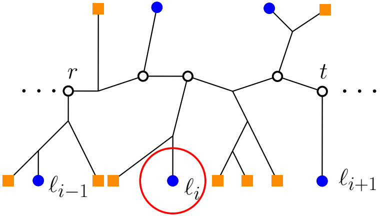
Lemma 1

In every component, there a exists at least one marked leaf ℓ with neighborhood $nh(\ell)$ confined to that component.

Existence proof

Lemma 1

In every component, there a exists at least one marked leaf ℓ with neighborhood $nh(\ell)$ confined to that component.

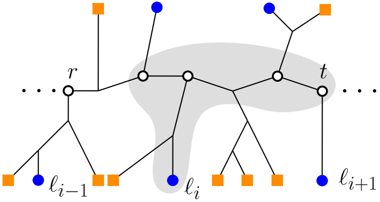


For a **5-component**: consider $nh(\ell_i)$.

Existence proof

Lemma 1

In every component, there a exists at least one marked leaf ℓ with neighborhood $nh(\ell)$ confined to that component.



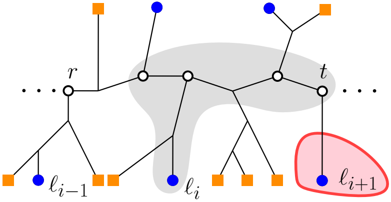
For a **5-component**:
consider $nh(\ell_i)$.

Case 1: If Comb node $t \in nh(\ell_i)$

Existence proof

Lemma 1

In every component, there a exists at least one marked leaf ℓ with neighborhood $nh(\ell)$ confined to that component.



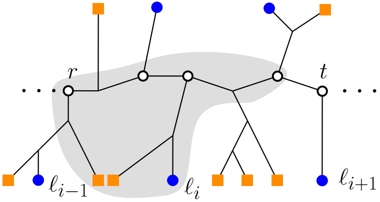
For a **5-component**:
consider $nh(\ell_i)$.

Case 1: If Comb node $t \in nh(\ell_i)$
$\Rightarrow nh(\ell_{i+1})$ is confined.

Existence proof

Lemma 1

In every component, there a exists at least one marked leaf ℓ with neighborhood $nh(\ell)$ confined to that component.



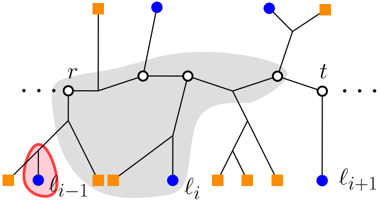
For a **5-component**: consider $nh(\ell_i)$.

Case 2: If Comb node $r \in nh(\ell_i)$

Existence proof

Lemma 1

In every component, there a exists at least one marked leaf ℓ with neighborhood $nh(\ell)$ confined to that component.



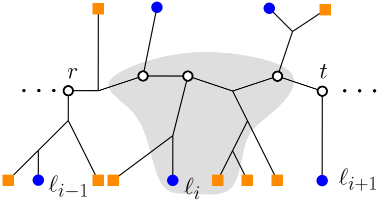
For a **5-component**:
consider $nh(\ell_i)$.

Case 2: If Comb node $r \in nh(\ell_i)$
$\Rightarrow nh(\ell_{i-1})$ is confined.

Existence proof

Lemma 1

In every component, there a exists at least one marked leaf ℓ with neighborhood $nh(\ell)$ confined to that component.



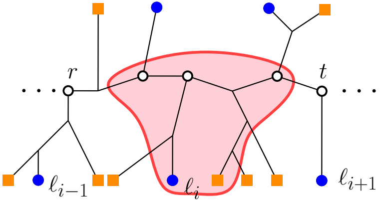
For a **5-component**: consider $nh(\ell_i)$.

Case 3: If Comb nodes $r, t \notin nh(\ell_i)$

Existence proof

Lemma 1

In every component, there a exists at least one marked leaf ℓ with neighborhood $nh(\ell)$ confined to that component.



For a **5-component**: consider $nh(\ell_i)$.

Case 3: If Comb nodes $r, t \notin nh(\ell_i)$ $\Rightarrow nh(\ell_i)$ is confined.

Existence

Corrolary - Lemma 1

The number of marked leaves with a confined neighborhood is:
$\rightarrow$ At least 1 out of 5 in every 5-component.
$\rightarrow$ At least 1 out of 2 in every L-component.

Existence

Corrolary - Lemma 1

The number of marked leaves with a confined neighborhood is:
$\rightarrow$ At least 1 out of 5 in every 5-component.
$\rightarrow$ At least 1 out of 2 in every L-component.

Observation

Each spine has at most 4 ungrouped Comb nodes.

Lemma 2

For every 8 ungrouped *Comb* nodes there exists at least 1 L-component.

Existence

Corrolary - Lemma 1

The number of marked leaves with a confined neighborhood is:
$\rightarrow$ At least 1 out of 5 in every 5-component.
$\rightarrow$ At least 1 out of 2 in every L-component.

Observation

Each spine has at most 4 ungrouped Comb nodes.

Lemma 2

For every 8 ungrouped *Comb* nodes there exists at least 1 L-component.

Combining the above, we conclude:

Lemma - Existence

At least $\frac{1}{10}m$ marked leaves have pairwise disjoint neighborhoods.

Designing an algorithm

Goal: Design an algorithm to return a fraction of the marked leaves with pairwise disjoint neighborhoods.

Designing an algorithm

Goal: Design an algorithm to return a fraction of the marked leaves with pairwise disjoint neighborhoods.

Challenge: Arbitrary distribution of unmarked leaves among marked leaves in the topological ordering.

Designing an algorithm

Goal: Design an algorithm to return a fraction of the marked leaves with pairwise disjoint neighborhoods.

Challenge: Arbitrary distribution of unmarked leaves among marked leaves in the topological ordering. This implies that:
$\rightarrow$ A component can have $\Theta(n)$ size.

Designing an algorithm

Goal: Design an algorithm to return a fraction of the marked leaves with pairwise disjoint neighborhoods.

Challenge: Arbitrary distribution of unmarked leaves among marked leaves in the topological ordering. This implies that:

$\rightarrow$ A component can have $\Theta(n)$ size.

$\rightarrow$ A confined neighborhood can have $\Theta(n)$ size.

Designing an algorithm

Goal: Design an algorithm to return a fraction of the marked leaves with pairwise disjoint neighborhoods.

Challenge: Arbitrary distribution of unmarked leaves among marked leaves in the topological ordering. This implies that:
→ A component can have $\Theta(n)$ size.
→ A confined neighborhood can have $\Theta(n)$ size.
→ A single neighborhood can require $\Theta(n)$ time to be identified.

Designing an algorithm

Goal: Design an algorithm to return a fraction of the marked leaves with pairwise disjoint neighborhoods.

Challenge: Arbitrary distribution of unmarked leaves among marked leaves in the topological ordering. This implies that:
$\rightarrow$ A component can have $\Theta(n)$ size.
$\rightarrow$ A confined neighborhood can have $\Theta(n)$ size.
$\rightarrow$ A single neighborhood can require $\Theta(n)$ time to be identified.

Introduce a parameter $p \in (0, 1)$ in the algorithm.
Trade-off between time complexity and number of selected leaves.

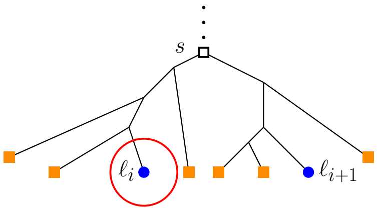
Algorithm description

1. Label the tree $\mathcal{T}$ and obtain the components.
2. **For each component** K check up to a **fixed number of steps** $O(z)$:

Algorithm description

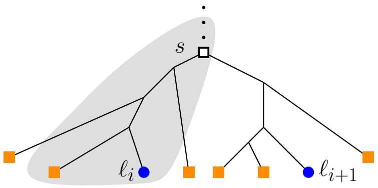
1. Label the tree $\mathcal{T}$ and obtain the components.
2. **For each component** K check up to a **fixed number of steps** $O(z)$:

3. **If** K is L-component **then:**
   trace $nh(\ell_i)$ for $\leq$ **4z** steps:



Algorithm description

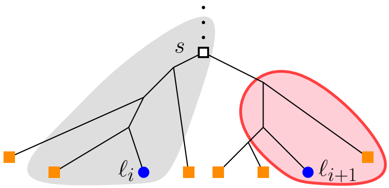
1. Label the tree $\mathcal{T}$ and obtain the components.
2. **For each component** K check up to a **fixed number of steps** $O(z)$:
3. **If** K is L-component **then:**
   trace $nh(\ell_i)$ for $\leq$ **4z** steps:
   If s is visited **then:**



Algorithm description

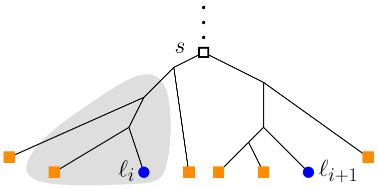
1. Label the tree $\mathcal{T}$ and obtain the components.
2. **For each component** K check up to a **fixed number of steps** $O(z)$:

3. **If** K is L-component **then:**
   trace $nh(\ell_i)$ for $\leq$ **4z** steps:
   If s is visited **then:**
   select ℓ_{i+1}



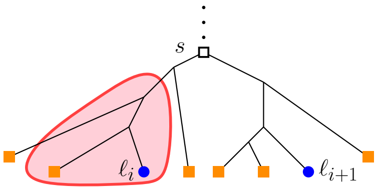
Algorithm description

1. Label the tree $\mathcal{T}$ and obtain the components.
2. **For each component** K check up to a **fixed number of steps** $O(z)$:
3. **If** K is L-component **then:**
   trace $nh(\ell_i)$ for $\leq$ **4z** steps:
   If s is visited **then:**
   select ℓ_{i+1}
   If $nh(\ell_i)$ is found **and**
   s is not visited **then:**



Algorithm description

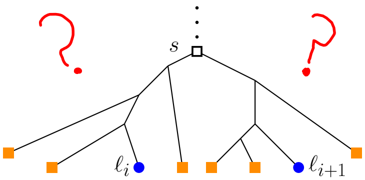
1. Label the tree $\mathcal{T}$ and obtain the components.
2. **For each component** K check up to a **fixed number of steps** $O(z)$:
3. **If** K is L-component **then:**
   trace $nh(\ell_i)$ for $\leq$ **4z** steps:
   If s is visited **then:**
   select ℓ_{i+1}
   If $nh(\ell_i)$ is found **and**
   s is not visited **then:**
   select ℓ_i



Algorithm description

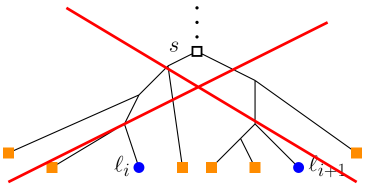
1. Label the tree $\mathcal{T}$ and obtain the components.
2. **For each component** K check up to a **fixed number of steps** $O(z)$:

3. **If** K is L-component **then:**
   trace $nh(\ell_i)$ for $\leq$ **4z** steps:
   If s is visited **then:**
   select ℓ_{i+1}
   If $nh(\ell_i)$ is found **and**
   s is not visited **then:**
   select ℓ_i
   If $nh(\ell_i)$ is not found **then:**



Algorithm description

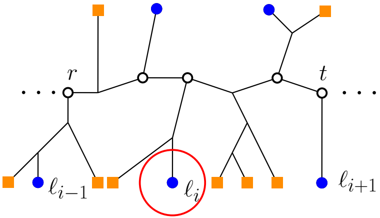
1. Label the tree $\mathcal{T}$ and obtain the components.
2. **For each component** K check up to a **fixed number of steps** $O(z)$:
3. **If** K is L-component **then:**
   trace $nh(\ell_i)$ for $\leq$ **4z** steps:
   If s is visited **then:**
   select ℓ_{i+1}
   If $nh(\ell_i)$ is found **and**
   s is not visited **then:**
   select ℓ_i
   If $nh(\ell_i)$ is not found **then:**
   abandon K



Algorithm description

1. Label the tree $\mathcal{T}$ and obtain the components.
2. **For each component** K check up to a **fixed number of steps** $O(z)$:

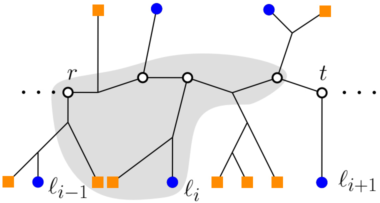
4. **If** K is 5-component **then:**
   trace $nh(\ell_i)$ for $\leq$ **10z** steps:



Algorithm description

1. Label the tree $\mathcal{T}$ and obtain the components.
2. **For each component** K check up to a **fixed number of steps** $O(z)$:

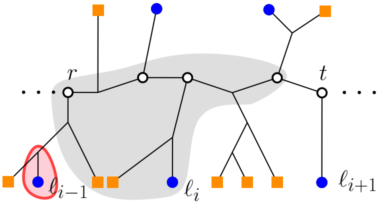
4. **If** K is 5-component **then:**
   trace $nh(\ell_i)$ for $\leq$ **10z** steps:
   If r is visited **then:**



Algorithm description

1. Label the tree $\mathcal{T}$ and obtain the components.
2. **For each component** K check up to a **fixed number of steps** $O(z)$:

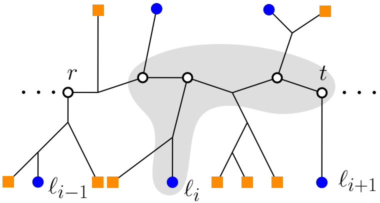
4. **If** K is 5-component **then:**
   trace $nh(\ell_i)$ for $\leq$ **10z** steps:
   If r is visited **then:**
   select ℓ_{i-1}



Algorithm description

1. Label the tree $\mathcal{T}$ and obtain the components.
2. **For each component** K check up to a **fixed number of steps** $O(z)$:

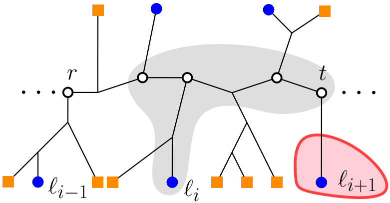
4. **If** K is 5-component **then:**
   trace $nh(\ell_i)$ for $\leq$ **10z** steps:
   If r is visited **then:**
   select ℓ_{i-1}
   If t is visited **then:**



Algorithm description

1. Label the tree $\mathcal{T}$ and obtain the components.
2. **For each component** K check up to a **fixed number of steps** $O(z)$:

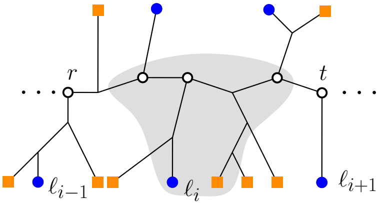
4. **If** K is 5-component **then:**
   trace $nh(\ell_i)$ for $\leq$ **10z** steps:
   If r is visited **then:**
   select ℓ_{i-1}
   If t is visited **then:**
   select ℓ_{i+1}



Algorithm description

1. Label the tree $\mathcal{T}$ and obtain the components.
2. **For each component** K check up to a **fixed number of steps** $O(z)$:

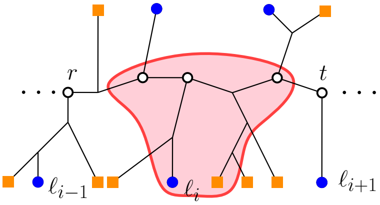
4. **If** K is 5-component **then:**
   trace $nh(\ell_i)$ for $\leq$ **10z** steps:
      If r is visited **then:**
         select ℓ_{i-1}
      If t is visited **then:**
         select ℓ_{i+1}
   If $nh(\ell_i)$ is found **and**
   r, t are not visited **then:**



Algorithm description

1. Label the tree $\mathcal{T}$ and obtain the components.
2. **For each component** K check up to a **fixed number of steps** $O(z)$:

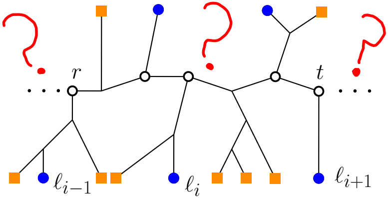
4. **If** K is 5-component **then:**
   trace $nh(\ell_i)$ for $\leq$ **10z** steps:
   If r is visited **then:**
   select ℓ_{i-1}
   If t is visited **then:**
   select ℓ_{i+1}
   If $nh(\ell_i)$ is found **and**
   r, t are not visited **then:**
   select ℓ_i



Algorithm description

1. Label the tree $\mathcal{T}$ and obtain the components.
2. **For each component** K check up to a **fixed number of steps** $O(z)$:

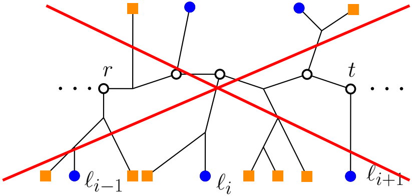
4. **If** K is 5-component **then:**
   trace $nh(\ell_i)$ for $\leq$ **10z** steps:
   If r is visited **then:**
   select ℓ_{i-1}
   If t is visited **then:**
   select ℓ_{i+1}
   If $nh(\ell_i)$ is found **and**
   r, t are not visited **then:**
   select ℓ_i
   If $nh(\ell_i)$ is not found **then:**



Algorithm description

1. Label the tree $\mathcal{T}$ and obtain the components.
2. **For each component** K check up to a **fixed number of steps** $O(z)$:

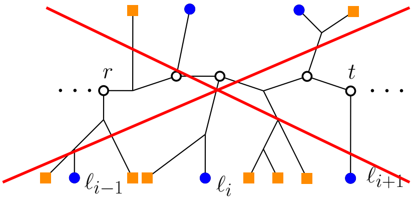
4. **If** K is 5-component **then:**
   trace $nh(\ell_i)$ for $\leq$ **10z** steps:
   If r is visited **then:**
   select ℓ_{i-1}
   If t is visited **then:**
   select ℓ_{i+1}
   If $nh(\ell_i)$ is found **and** r, t are not visited **then:**
   select ℓ_i
   If $nh(\ell_i)$ is not found **then:**
   abandon K



Algorithm description

1. Label the tree $\mathcal{T}$ and obtain the components.
2. **For each component** K check up to a **fixed number of steps** $O(z)$:

4. **If** K is 5-component **then:**
   trace $nh(\ell_i)$ for $\leq$ **10z** steps:
   If r is visited **then:**
   select ℓ_{i-1}
   If t is visited **then:**
   select ℓ_{i+1}
   If $nh(\ell_i)$ is found **and** r, t are not visited **then:**
   select ℓ_i
   If $nh(\ell_i)$ is not found **then:**
   abandon K



5. **Return** selected leaves.

Algorithm proofs

Need to show:

Algorithm Correctness

Lemma - Correctness

The algorithm returns at least $\frac{p}{10}m$ leaves with pairwise disjoint neighborhoods.

Algorithm proofs

Need to show:

Algorithm Correctness

Lemma - Correctness

The algorithm returns at least $\frac{p}{10}m$ leaves with pairwise disjoint neighborhoods.

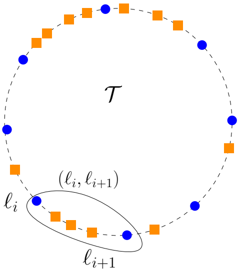
Algorithm time complexity

Lemma -Time complexity

The algorithm has time complexity $O(\frac{1}{1-p}n)$.

Correctness proof

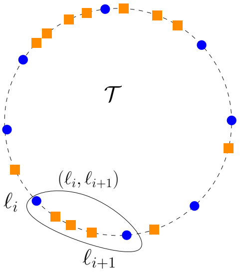
Intervals



Correctness proof

Idea:
Lower bound the number of intervals that do not have *many* unmarked leaves.

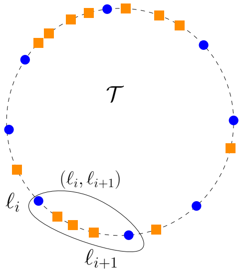
Intervals



Correctness proof

Idea:
Lower bound the number of intervals that do not have *many* unmarked leaves.

Intervals



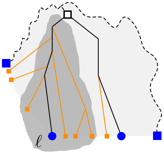
Lemma - *Pigeonhole*

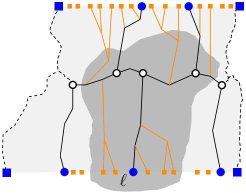
Let M_x be the number of marked leaves whose intervals have at most x unmarked leaves, $x \in \mathbb{N}$. Then $|M_x| \geq \frac{x - c + 1}{x + 1} m$ holds.

c is the ratio between unmarked and marked leaves, $c = \lceil \frac{n-m}{m} \rceil$.

Correctness proof

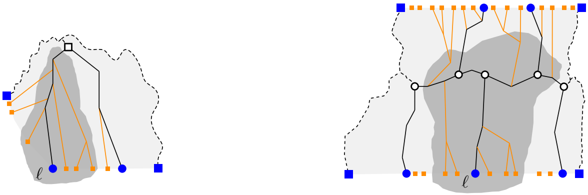
Idea:
Upper bound the size of a confined neighborhood by the number of unmarked leaves in the intervals related to the component.





Correctness proof

Idea:
Upper bound the size of a confined neighborhood by the number of unmarked leaves in the intervals related to the component.



Lemma - Size of confined neighborhoods

Let K be component and a marked leaf ℓ with neighborhood $nh(\ell)$ confined K. Then, $|nh(\ell)| < 10\delta_K$.

δ_K is the maximum size of intervals related to the component K.

Time complexity proof

Lemma -Time complexity

The algorithm has time complexity $O(\frac{1}{1-p}n)$.

- There are $\Theta(m)$ components.

Time complexity proof

Lemma -Time complexity

The algorithm has time complexity $O(\frac{1}{1-p}n)$.

- There are $\Theta(m)$ components.
- For each component, the algorithm does a fixed number of steps ($\leq 10z$).
  By using $z = \left\lceil \frac{10c}{1-p} \right\rceil = \Theta(\frac{c}{1-p})$, the claim follows.

Conclusion

Theorem - Generalized

Let $\mathcal{T}$ be an embedded binary tree with n leaves where:
i) m of the leaves have been marked.
ii) Each marked leaf of $\mathcal{T}$ has a neighborhood.
iii) Topologically consecutive marked leaves have disjoint neighborhoods.
Then:
i) $\exists \geq \frac{1}{10}m$ marked leaves with pairwise disjoint neighborhoods.
ii) $\geq \frac{p}{10}m$ marked leaves can be found in $O(\frac{1}{1-p}n)$ time, $p \in (0,1)$.

Conclusion

Theorem - Generalized

Let $\mathcal{T}$ be an embedded binary tree with n leaves where:
$i)$ m of the leaves have been marked.
$ii)$ Each marked leaf of $\mathcal{T}$ has a neighborhood.
$iii)$ Topologically consecutive marked leaves have disjoint neighborhoods.
Then:
$i)$ $\exists \geq \frac{1}{10}m$ marked leaves with pairwise disjoint neighborhoods.
$ii)$ $\geq \frac{p}{10}m$ marked leaves can be found in $O(\frac{1}{1-p}n)$ time, $p \in (0,1)$.

Expect it to be helpful in designing deterministic **linear time algorithms** for problems related to **abstract Voronoi diagrams** and other **generalized Voronoi diagrams**.

Thank you for your attention!