Computing the Arboricity

Putting the *work* back in workshop

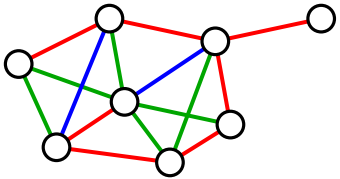
Tijn de Vos

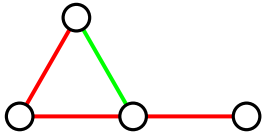
TU Graz

Arboricity

Arboricity

- The (integral) arboricity: α_i the minimum number of forests that cover E.

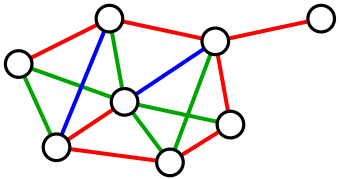


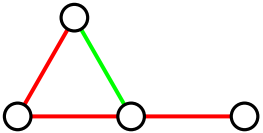


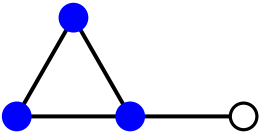
Arboricity

- The (integral) arboricity: α_i the minimum number of forests that cover E.
- The fractional arboricity:

$$\alpha_f := \max_{S \subseteq V} \frac{|E(S)|}{|S| - 1}.$$



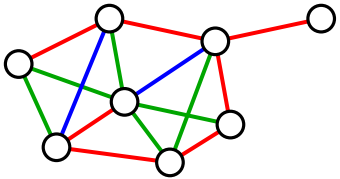


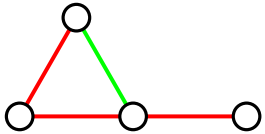


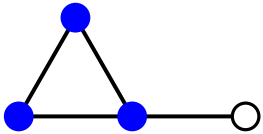
Arboricity

- The (integral) arboricity: α_i the minimum number of forests that cover E.
- The fractional arboricity:

$$\alpha_f := \max_{S \subseteq V} \frac{|E(S)|}{|S| - 1}.$$







Theorem [Nash-Williams '61]

We have $\alpha_i = \lceil \alpha_f \rceil$.

Arboricity is an Important Parameter

In LOCAL, CONGEST, MPC, dynamic, ...

- Ruling sets [Kothapalli, Pemmaraju '12, Pai, Pandurangan, Pemmaraju, Riaz, Robinson '17, Baumecker, Latypov, Maus, Uitto '26]
- MIS [Barenboim, Elkin, Kuhn '14, Ghaffari, Grunau, Jin '20, Faour, Kuhn '25]
- Correlation clustering [Cambus, Choo, Miikonen, Uitto '21]
- Out-orientations [Chekuri, Christiansen, Holm, van der Hoog, Quanrud, Rotenberg, Schwiegelshohn '24]
- Dynamic coloring [Christiansen, Nowicki, Rotenberg '23]
- Dominating set [Lenzen, Wattenhofer '10, Morgan, Solomon, Wein '21, Dory, Ghaffari, Ilchi '23]
- k-clique counting [Eden, Ron, Seshadhri '20]

Observations

1) If $\alpha_f \leq \tilde{\alpha}_f \leq (1+\varepsilon)\alpha_f$ then $\alpha_i \leq \lceil \tilde{\alpha}_f \rceil \leq (1+\varepsilon)\alpha_i + 1$.

Observations

1) If $\alpha_f \leq \tilde{\alpha}_f \leq (1+\varepsilon)\alpha_f$ then $\alpha_i \leq \lceil \tilde{\alpha}_f \rceil \leq (1+\varepsilon)\alpha_i + 1$.
2) If $\alpha_i \leq \tilde{\alpha}_i \leq (1+\varepsilon)\alpha_f$ then $\alpha_f \leq \tilde{\alpha}_i \leq (1+\varepsilon)\alpha_f + 1$.

Observations

1) If $\alpha_f \leq \tilde{\alpha}_f \leq (1+\varepsilon)\alpha_f$ then $\alpha_i \leq \lceil \tilde{\alpha}_f \rceil \leq (1+\varepsilon)\alpha_i + 1$.
2) If $\alpha_i \leq \tilde{\alpha}_i \leq (1+\varepsilon)\alpha_f$ then $\alpha_f \leq \tilde{\alpha}_i \leq (1+\varepsilon)\alpha_f + 1$.
3) $\alpha_i \leq \sqrt{m}$

Observations

1) If $\alpha_f \leq \tilde{\alpha}_f \leq (1+\varepsilon)\alpha_f$ then $\alpha_i \leq \lceil \tilde{\alpha}_f \rceil \leq (1+\varepsilon)\alpha_i + 1$.
2) If $\alpha_i \leq \tilde{\alpha}_i \leq (1+\varepsilon)\alpha_f$ then $\alpha_f \leq \tilde{\alpha}_i \leq (1+\varepsilon)\alpha_f + 1$.
3) $\alpha_i \leq \sqrt{m}$
4) $\alpha_i \geq m/(n-1)$

Part 1: Arboricity Algorithms in Static and Dynamic Models[1]

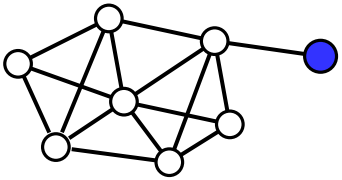
Part 2: Arboricity Algorithms in Distributed Computing

[1] Cherry picked results

Simplest Algorithm: Degeneracy

Algorithm

- $k \leftarrow 0$
- Until G is empty:
  - Let v be a node of lowest degree
  - $k \leftarrow \max\{k, \deg(v)\}$
  - Remove v from G
- Output degeneracy k

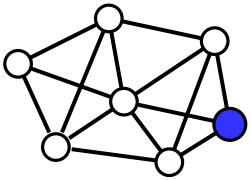


$k = 1$

Simplest Algorithm: Degeneracy

Algorithm

- $k \leftarrow 0$
- Until G is empty:
  - Let v be a node of lowest degree
  - $k \leftarrow \max\{k, \deg(v)\}$
  - Remove v from G
- Output degeneracy k

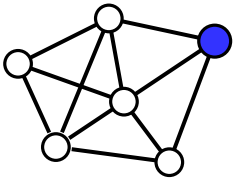


$k = 3$

Simplest Algorithm: Degeneracy

Algorithm

- $k \leftarrow 0$
- Until G is empty:
  - Let v be a node of lowest degree
  - $k \leftarrow \max\{k, \deg(v)\}$
  - Remove v from G
- Output degeneracy k

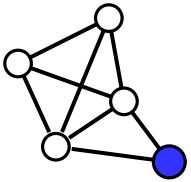


$k = 3$

Simplest Algorithm: Degeneracy

Algorithm

- $k \leftarrow 0$
- Until G is empty:
  - Let v be a node of lowest degree
  - $k \leftarrow \max\{k, \deg(v)\}$
  - Remove v from G
- Output degeneracy k

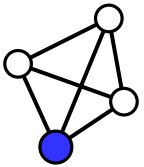


$k = 3$

Simplest Algorithm: Degeneracy

Algorithm

- $k \leftarrow 0$
- Until G is empty:
  - Let v be a node of lowest degree
  - $k \leftarrow \max\{k, \deg(v)\}$
  - Remove v from G
- Output degeneracy k

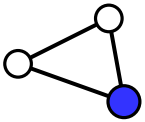


$k = 3$

Simplest Algorithm: Degeneracy

Algorithm

- $k \leftarrow 0$
- Until G is empty:
  - Let v be a node of lowest degree
  - $k \leftarrow \max\{k, \deg(v)\}$
  - Remove v from G
- Output degeneracy k

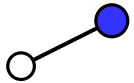


$k = 3$

Simplest Algorithm: Degeneracy

Algorithm

- $k \leftarrow 0$
- Until G is empty:
  - Let v be a node of lowest degree
  - $k \leftarrow \max\{k, \deg(v)\}$
  - Remove v from G
- Output degeneracy k



$k = 3$

Simplest Algorithm: Degeneracy

Algorithm

- $k \leftarrow 0$
- Until G is empty:
  - Let v be a node of lowest degree
  - $k \leftarrow \max\{k, \deg(v)\}$
  - Remove v from G
- Output degeneracy k



$k = 3$

- Takes $O(m)$ time.

Simplest Algorithm: Degeneracy

Algorithm

- $k \leftarrow 0$
- Until G is empty:
  - Let v be a node of lowest degree
  - $k \leftarrow \max\{k, \deg(v)\}$
  - Remove v from G
- Output degeneracy k



$k = 3$

- Takes $O(m)$ time.
- $k/2 \leq \alpha_i \leq k$.

The Densest Subgraph Problem

- Recall: $\alpha_f := \max_{S \subseteq V} \frac{|E(S)|}{|S|-1}$.

The Densest Subgraph Problem

- Recall: $\alpha_f := \max_{S \subseteq V} \frac{|E(S)|}{|S|-1}$.
- Now we define the density of G as

$$\rho := \max_{S \subseteq V} \frac{|E(S)|}{|S|}.$$

The Densest Subgraph Problem

- Recall: $\alpha_f := \max_{S \subseteq V} \frac{|E(S)|}{|S|-1}$.
- Now we define the density of G as

$$\rho := \max_{S \subseteq V} \frac{|E(S)|}{|S|}.$$

- Dual to out-orientation.

The Densest Subgraph Problem

- Recall: $\alpha_f := \max_{S \subseteq V} \frac{|E(S)|}{|S|-1}$.
- Now we define the density of G as

$$\rho := \max_{S \subseteq V} \frac{|E(S)|}{|S|}.$$

- Dual to out-orientation.

- ρ is a 2-approximation of α_f.

The Densest Subgraph Problem

- Recall: $\alpha_f := \max_{S \subseteq V} \frac{|E(S)|}{|S|-1}$.
- Now we define the density of G as

$$\rho := \max_{S \subseteq V} \frac{|E(S)|}{|S|}.$$

- Dual to out-orientation.

- ρ is a 2-approximation of α_f.
- In simple graphs:
  - $\alpha_f \leq \rho + \frac{1}{2}$.
  - $\alpha_i \leq \rho + 1$.

The Densest Subgraph Problem

- Recall: $\alpha_f := \max_{S \subseteq V} \frac{|E(S)|}{|S|-1}$.
- Now we define the density of G as

$$\rho := \max_{S \subseteq V} \frac{|E(S)|}{|S|}.$$

- Dual to out-orientation.

- ρ is a 2-approximation of α_f.
- In simple graphs:
  - $\alpha_f \leq \rho + \frac{1}{2}$.
  - $\alpha_i \leq \rho + 1$.

Algorithms

- Exact: Reduces to max flow: almost-linear time results [Goldberg '84 + Chen et al. '22]

The Densest Subgraph Problem

- Recall: $\alpha_f := \max_{S \subseteq V} \frac{|E(S)|}{|S|-1}$.
- Now we define the density of G as

$$\rho := \max_{S \subseteq V} \frac{|E(S)|}{|S|}.$$

- Dual to out-orientation.

- ρ is a 2-approximation of α_f.
- In simple graphs:
  - $\alpha_f \leq \rho + \frac{1}{2}$.
  - $\alpha_i \leq \rho + 1$.

Algorithms

- Exact: Reduces to max flow: almost-linear time results [Goldberg '84 + Chen et al. '22]
- $(1 \pm \varepsilon)$ approximation: Packing linear program $\implies$ multiplicative weight update method in near-linear time [Bahmani, Goel, Munagala '14]
- Iterative peeling: $O(m \operatorname{poly} \log \varepsilon^{-1})$ [Grigorescu, Taziki '26]

A Matroid Approach

- Exact integral arboricity: $\tilde{O}(m\alpha_i)$ time [Gabow '95]
  - Using augmenting paths.
  - SOTA until 2025

A Matroid Approach

- Exact integral arboricity: $\tilde{O}(m\alpha_i)$ time [Gabow '95]
  - Using augmenting paths.
  - SOTA until 2025
- $(1+\varepsilon)$-approximation: $\tilde{O}(m)$ time [Quanrud '24]
  - Using push-relabel.
  - SOTA

A Matroid Approach

- Exact integral arboricity: $\tilde{O}(m\alpha_i)$ time [Gabow '95]
  - Using augmenting paths.
  - SOTA until 2025
- $(1+\varepsilon)$-approximation: $\tilde{O}(m)$ time [Quanrud '24]
  - Using push-relabel.
  - SOTA
- Inherently sequential?

Tree-Packing

Tree-Packing

Definitions

- Tree-packing: $\mathcal{T} = \{T_1, T_2, \dots\}$: a family of trees in G such that [...].
- Load: $L^{\mathcal{T}}(e)$: the number of trees in $\mathcal{T}$ that contain an edge e.
- Relative load: $\ell^{\mathcal{T}}(e) := L^{\mathcal{T}}(e)/|\mathcal{T}|$.

Tree-Packing

Definitions

- Tree-packing: $\mathcal{T} = \{T_1, T_2, \dots\}$: a family of trees in G such that [...].
- Load: $L^{\mathcal{T}}(e)$: the number of trees in $\mathcal{T}$ that contain an edge e.
- Relative load: $\ell^{\mathcal{T}}(e) := L^{\mathcal{T}}(e)/|\mathcal{T}|$.

Min-Cut applications

- Static [Karger '00] + [...].
- Distributed [Daga, Henzinger, Nanongkai, Saranurak '19, Dory, Efron, Mukhopadhyay, Nanongkai '21].
- Dynamic [Thorup, Karger '00, Thorup '07, **dV**, Christiansen '25].

Ideal ‘Tree-Packing’

- Define ideal load $\ell^*(e)$.

Ideal 'Tree-Packing'

- Define ideal load $\ell^*(e)$.
- We want

$$\#\text{disjoint spanning trees} = \frac{1}{\max_e \ell^*(e)}.$$

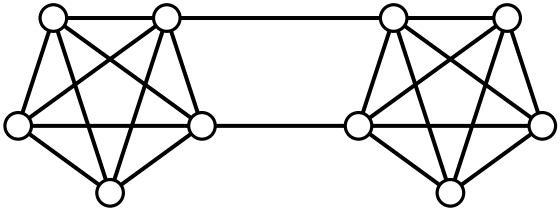
Ideal 'Tree-Packing'

- Define ideal load $\ell^*(e)$.
- We want

$$\#\text{disjoint spanning trees} = \frac{1}{\max_e \ell^*(e)}.$$

- If we fit k disjoint spanning trees in some $G/\mathcal{P}$ containing e then

$$\ell^*(e) = \frac{1}{k}.$$



Ideal Loads

- For a vertex partition $\mathcal{P}$, we define the partition value as

$$\text{part_val}(\mathcal{P}) := \frac{|\mathrm{E}(\mathrm{G}/\mathcal{P})|}{|\mathcal{P}| - 1}.$$

Ideal Loads

- For a vertex partition $\mathcal{P}$, we define the partition value as

$$\text{part_val}(\mathcal{P}) := \frac{|\mathrm{E}(\mathrm{G}/\mathcal{P})|}{|\mathcal{P}| - 1}.$$

- Rephrase [Nash-Williams '61, Tutte '61]:

$$\Phi := \max_{\mathcal{T}} \frac{1}{\max_{e \in E} \ell^{\mathcal{T}}(e)} = \min_{\mathcal{P}} \text{part_val}(\mathcal{P}).$$

Ideal Loads

- For a vertex partition $\mathcal{P}$, we define the partition value as

$$\mathrm{part_val}(\mathcal{P}) := \frac{|\mathrm{E}(\mathrm{G}/\mathcal{P})|}{|\mathcal{P}| - 1}.$$

- Rephrase [Nash-Williams '61, Tutte '61]:

$$\Phi := \max_{\mathcal{T}} \frac{1}{\max_{e \in E} \ell^{\mathcal{T}}(e)} = \min_{\mathcal{P}} \mathrm{part_val}(\mathcal{P}).$$

- Define ideal load ℓ^* recursively:

Ideal Loads

- For a vertex partition $\mathcal{P}$, we define the partition value as

$$\text{part_val}(\mathcal{P}) := \frac{|\mathrm{E}(\mathrm{G}/\mathcal{P})|}{|\mathcal{P}| - 1}.$$

- Rephrase [Nash-Williams '61, Tutte '61]:

$$\Phi := \max_{\mathcal{T}} \frac{1}{\max_{e \in E} \ell^{\mathcal{T}}(e)} = \min_{\mathcal{P}} \text{part_val}(\mathcal{P}).$$

- Define ideal load ℓ^* recursively:
  - Let $\mathcal{P}^*$ be a partition with $\text{part_val}(\mathcal{P}^*) = \Phi$.

Ideal Loads

- For a vertex partition $\mathcal{P}$, we define the partition value as

$$\mathrm{part_val}(\mathcal{P}) := \frac{|\mathrm{E}(\mathrm{G}/\mathcal{P})|}{|\mathcal{P}| - 1}.$$

- Rephrase [Nash-Williams '61, Tutte '61]:

$$\Phi := \max_{\mathcal{T}} \frac{1}{\max_{e \in E} \ell^{\mathcal{T}}(e)} = \min_{\mathcal{P}} \mathrm{part_val}(\mathcal{P}).$$

- Define ideal load ℓ^* recursively:
  - Let $\mathcal{P}^*$ be a partition with $\mathrm{part_val}(\mathcal{P}^*) = \Phi$.
  - For all $e \in E(G/\mathcal{P}^*)$, set $\ell^*(e) := 1/\Phi$.

Ideal Loads

- For a vertex partition $\mathcal{P}$, we define the partition value as

$$\mathrm{part_val}(\mathcal{P}) := \frac{|\mathrm{E}(\mathrm{G}/\mathcal{P})|}{|\mathcal{P}| - 1}.$$

- Rephrase [Nash-Williams '61, Tutte '61]:

$$\Phi := \max_{\mathcal{T}} \frac{1}{\max_{e \in E} \ell^{\mathcal{T}}(e)} = \min_{\mathcal{P}} \mathrm{part_val}(\mathcal{P}).$$

- Define ideal load ℓ^* recursively:
  - Let $\mathcal{P}^*$ be a partition with $\mathrm{part_val}(\mathcal{P}^*) = \Phi$.
  - For all $e \in E(G/\mathcal{P}^*)$, set $\ell^*(e) := 1/\Phi$.
  - For each $S \in \mathcal{P}^*$, recurse on the subgraph $G[S]$.

Arboricity and the Ideal Loads

Theorem [dV, Christiansen '25][2]

$$\alpha_f = \frac{1}{\min_{e \in E} \ell^*(e)}$$

[2] Generalized for base-packing in matroids [dV, Grilnberger '26, Arkhipov, Kolmogorov '26].

Static exact Arboricity

- Exact α_f and hence α_i in $m^{1+o(1)}$) time [Cen, Fleischmann, Li, Li, Panigrahi '25].
- Provides an algorithm to compute all ideal loads: $\tilde{O}(n)$ max flow calls.

Static exact Arboricity

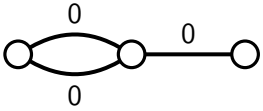
- Exact α_f and hence α_i in $m^{1+o(1)}$) time [Cen, Fleischmann, Li, Li, Panigrahi '25].
- Provides an algorithm to compute all ideal loads: $\tilde{O}(n)$ max flow calls.
- For minimum only: start at the bottom of the hierarchy.
- Reduces it to $\log n$ calls to max flow and directed min-cut.
  - Directed min-cut takes $m^{1+o(1)}$ time [Fleischmann, Li, Saranurak, Wang '26]

Static exact Arboricity

- Exact α_f and hence α_i in $m^{1+o(1)}$) time [Cen, Fleischmann, Li, Li, Panigrahi '25].
- Provides an algorithm to compute all ideal loads: $\tilde{O}(n)$ max flow calls.
- For minimum only: start at the bottom of the hierarchy.
- Reduces it to $\log n$ calls to max flow and directed min-cut.
  - Directed min-cut takes $m^{1+o(1)}$ time [Fleischmann, Li, Saranurak, Wang '26]
- $n^{2+o(1)}$ time algorithm that computes the forest decomposition [Gholami, Hajiaghayi, JafariRaviz, Kaviani, Mohammadkhani, Shahrezaei '26].

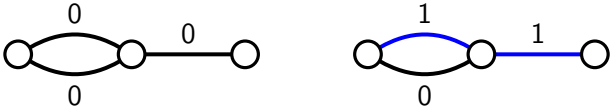
Greedy Tree-Packing

- Let the weight of an edge be the number of trees an edge belongs to. In a greedy tree-packing $\mathcal{T}$, the spanning trees form successive minimum spanning trees.



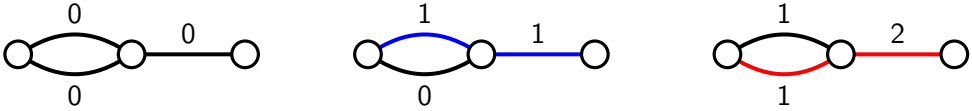
Greedy Tree-Packing

- Let the weight of an edge be the number of trees an edge belongs to. In a greedy tree-packing $\mathcal{T}$, the spanning trees form successive minimum spanning trees.



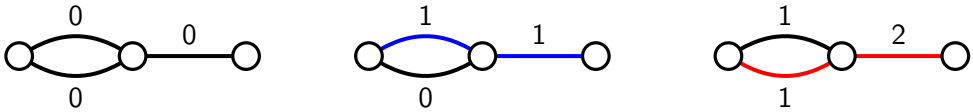
Greedy Tree-Packing

- Let the weight of an edge be the number of trees an edge belongs to. In a greedy tree-packing $\mathcal{T}$, the spanning trees form successive minimum spanning trees.



Greedy Tree-Packing

- Let the weight of an edge be the number of trees an edge belongs to. In a greedy tree-packing $\mathcal{T}$, the spanning trees form successive minimum spanning trees.



Theorem [dV, Grilnberger '26][3]

If $|\mathcal{T}| \geq \Theta(\alpha_f \log m/\varepsilon^2)$ then

$$|\ell^{\mathcal{T}}(e) - \ell^*(e)| \leq \varepsilon \ell^*(e),$$

for all $e \in E$.

[3] Generalizing [Thorup '07].

Simple Static Algorithm

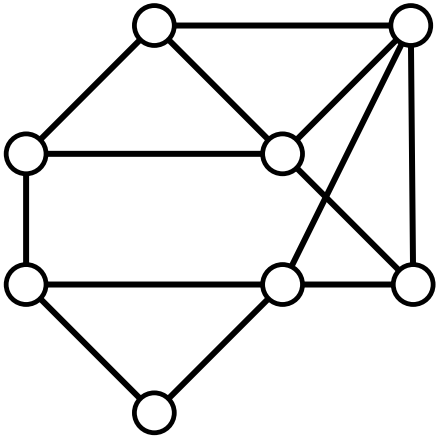
Algorithm

- Initialize $w(e) \leftarrow 0$ for all $e \in E$.
- For $i = 1, \ldots, \Theta(\alpha_f \log n/\varepsilon^2)$:
  - Compute a minimum spanning forest F.
  - Increase the weights $w(e)$ of all edges in $e \in F$ by 1.
- Output $\tilde{\alpha}_f = 1/\min_{e \in E} w(e)$.

- Produces a $(1 \pm \varepsilon)$-approximation
- We can assume $\alpha_f = O(\log n/\varepsilon^2)$ by uniform sampling.
- Corresponds the multiplicative weight update method on the linear program [Plotkin, Shmoys, Tardos '91].

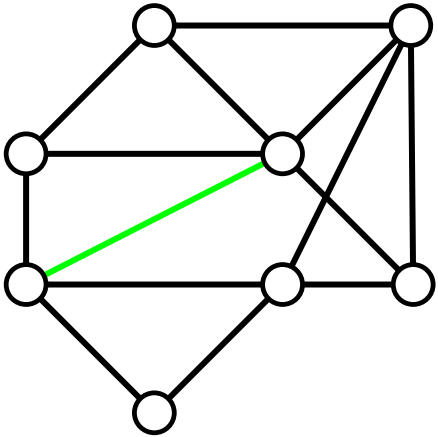
Dynamic Graph Algorithms

- Graph $G = (V, E)$ with edge updates: insertions/deletions.



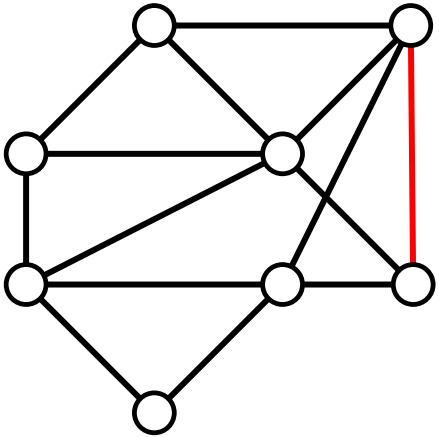
Dynamic Graph Algorithms

- Graph $G = (V, E)$ with edge updates: insertions/deletions.



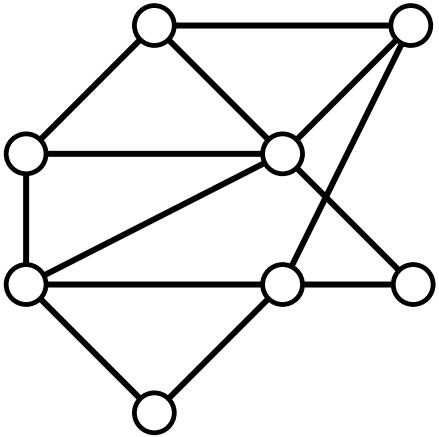
Dynamic Graph Algorithms

- Graph $G = (V, E)$ with edge updates: insertions/deletions.



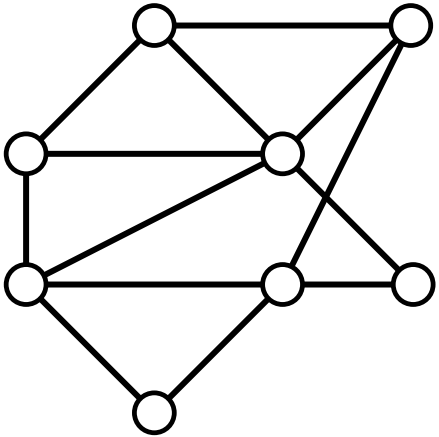
Dynamic Graph Algorithms

- Graph $G = (V, E)$ with edge updates: insertions/deletions.



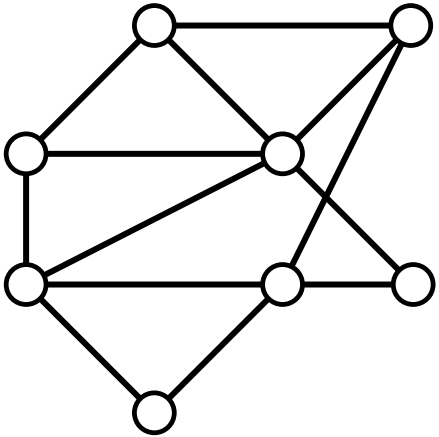
Dynamic Graph Algorithms

- Graph $G = (V, E)$ with edge updates: insertions/deletions.
- Maintain the answer to a problem.



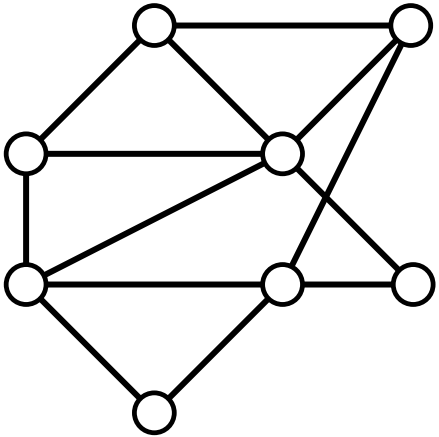
Dynamic Graph Algorithms

- Graph $G = (V, E)$ with edge updates: insertions/deletions.
- Maintain the answer to a problem.
- Goal: low amortized (average) update time.



Dynamic Graph Algorithms

- Graph $G = (V, E)$ with edge updates: insertions/deletions.
- Maintain the answer to a problem.
- Goal: low amortized (average) update time.
- Adversary determines updates:
  - Oblivious Adversary: updates fixed beforehand.
  - Adaptive Adversary: updates can depend on (random) choices of the algorithm.



Dynamic Arboricity

Theorem [dV, Christiansen '25]

Unweighted, undirected (multi-)graph $G = (V, E)$. There exists a deterministic dynamic algorithm that maintains a $(1+\varepsilon)$-approximation of the fractional arboricity α_f when $\alpha_f \leq \alpha_{\max}$ in $O(\alpha_{\max} \operatorname{poly}(\log n, \varepsilon^{-1}))$ amortized update time.

Dynamic Arboricity

Theorem [dV, Christiansen '25]

Unweighted, undirected (multi-)graph $G = (V, E)$. There exists a deterministic dynamic algorithm that maintains a $(1+\varepsilon)$-approximation of the fractional arboricity α_f when $\alpha_f \leq \alpha_{\max}$ in $O(\alpha_{\max} \operatorname{poly}(\log n, \varepsilon^{-1}))$ amortized update time.

- By maintaining MSTs with tricks to lower the recourse.
- Simple graphs: $\operatorname{poly}(\log n, \varepsilon^{-1})$ update time (deterministic)
  - Use densest subgraph/out-orientations for large values of α [Chekuri, Christiansen, Holm, van der Hoog, Quanrud, Rotenberg, Schwiegelshohn '24].

Dynamic Arboricity

Theorem [dV, Christiansen '25]

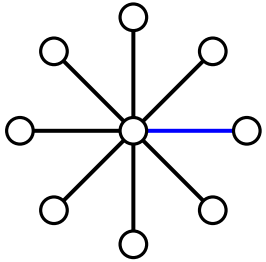
Unweighted, undirected (multi-)graph $G = (V, E)$. There exists a deterministic dynamic algorithm that maintains a $(1+\varepsilon)$-approximation of the fractional arboricity α_f when $\alpha_f \leq \alpha_{\max}$ in $O(\alpha_{\max} \operatorname{poly}(\log n, \varepsilon^{-1}))$ amortized update time.

- By maintaining MSTs with tricks to lower the recourse.
- Simple graphs: $\operatorname{poly}(\log n, \varepsilon^{-1})$ update time (deterministic)
  - Use densest subgraph/out-orientations for large values of α [Chekuri, Christiansen, Holm, van der Hoog, Quanrud, Rotenberg, Schwiegelshohn '24].
- Multi-graphs $\operatorname{poly}(\log n, \varepsilon^{-1})$ update time (randomized)
  - Oblivious adversary: uniform sampling.
  - Adaptive adversary: fancy sampling.

Fancy Sampling

Uniform sampling $\sim \frac{\log n}{\alpha_f \varepsilon^2}$.

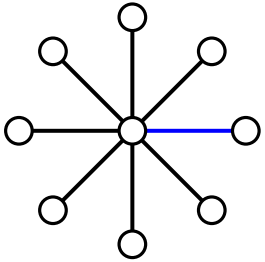
- Problem: Adaptive adversary can delete sampled edges.

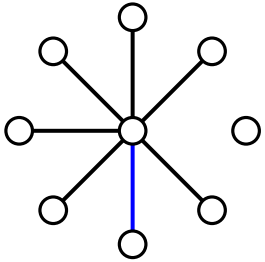


Fancy Sampling

Uniform sampling $\sim \frac{\log n}{\alpha_f \varepsilon^2}$.

- Problem: Adaptive adversary can delete sampled edges.
- Solution: A vertex resamples its out-edges when affected.

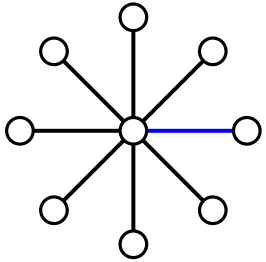


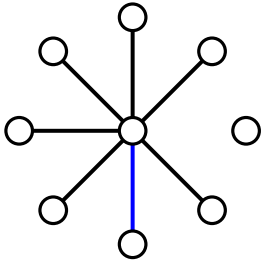


Fancy Sampling

Uniform sampling $\sim \frac{\log n}{\alpha_f \varepsilon^2}$.

- Problem: Adaptive adversary can delete sampled edges.
- Solution: A vertex resamples its out-edges when affected.
- Problem: Maximum degree $n - 1$.
  - In sampled graph $\approx n/\alpha_f$.

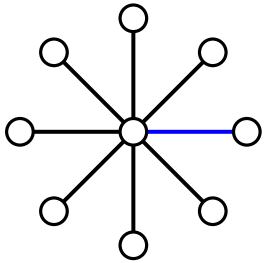


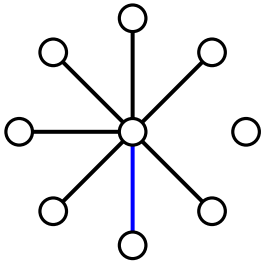


Fancy Sampling

Uniform sampling $\sim \frac{\log n}{\alpha_f \varepsilon^2}$.

- Problem: Adaptive adversary can delete sampled edges.
- Solution: A vertex resamples its out-edges when affected.
- Problem: Maximum degree $n - 1$.
  - In sampled graph $\approx n/\alpha_f$.
- Solution: Use out-orientations such that each vertex 'owns' $O(\alpha_f)$ edges.
  - In sampled graph $O(\alpha_f \cdot \frac{\log n}{\alpha_f \varepsilon^2}) = O(\frac{\log n}{\varepsilon^2})$.

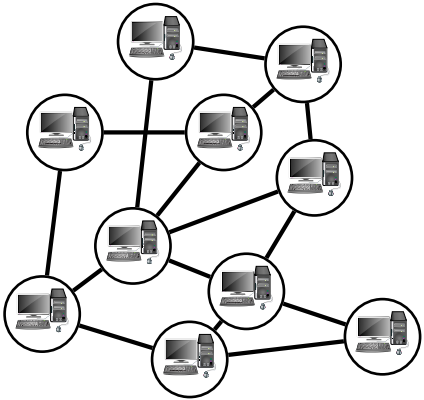




Part 1: Arboricity Algorithms in Static and Dynamic Models

Part 2: Arboricity Algorithms in Distributed Computing

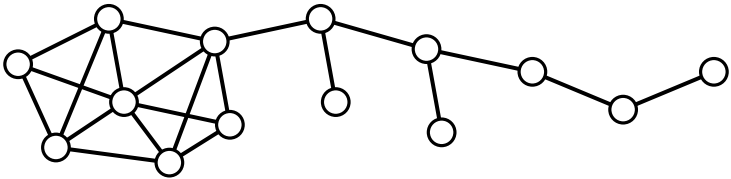
The LOCAL Model



- $G = (V, E)$, $|V| = n$, $|E| = m$
- G unknown: each node knows its neighbors.
- Communication over edges in synchronous rounds.

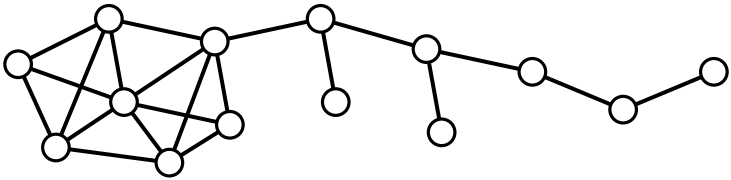
Problem Definition

- Goal: find α_i or α_f in a distributed model
- Inherently $\Omega(D)$



Problem Definition

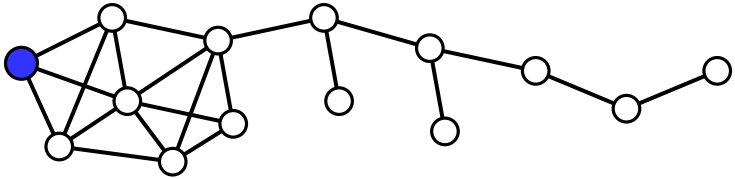
- Goal: find α_i or α_f in a distributed model
- Inherently $\Omega(D)$
- Local arboricity:
  - At least one node correct;
  - Others under-estimate.
- See also [Christiansen, van der Hoog, Rotenberg '25] for 'local density'



A LOCAL Algorithm

Theorem [unpublished]

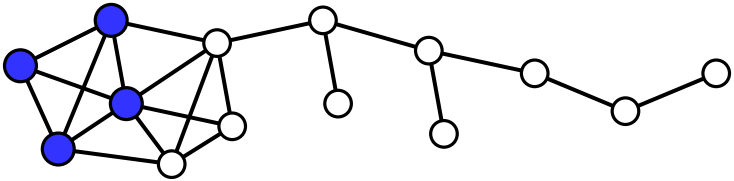
We can compute a $(1 \pm \varepsilon)$-approximation of α_f in $O(\log n/\varepsilon)$ rounds.



A LOCAL Algorithm

Theorem [unpublished]

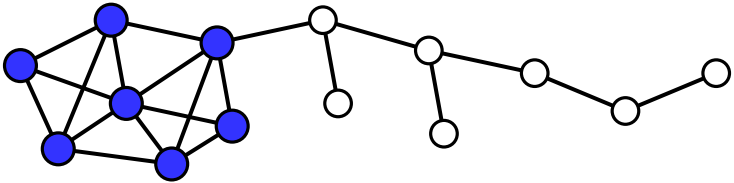
We can compute a $(1 \pm \varepsilon)$-approximation of α_f in $O(\log n/\varepsilon)$ rounds.



A LOCAL Algorithm

Theorem [unpublished]

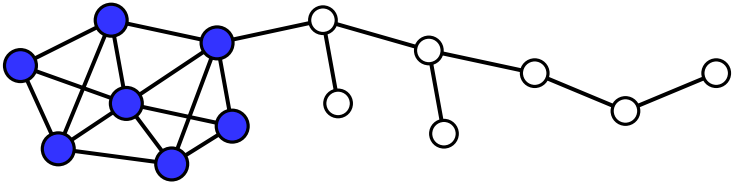
We can compute a $(1 \pm \varepsilon)$-approximation of α_f in $O(\log n/\varepsilon)$ rounds.



A LOCAL Algorithm

Theorem [unpublished]

We can compute a $(1 \pm \varepsilon)$-approximation of α_f in $O(\log n/\varepsilon)$ rounds.



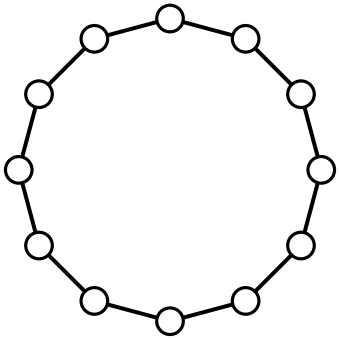
Theorem [found with Claude, unpublished]

A $(1 \pm \varepsilon)$-approximation of α_f requires $\Omega(\log n/\varepsilon)$ rounds.

Integer Arboricity

Observation

We need D rounds to discern between a long path and a cycle.

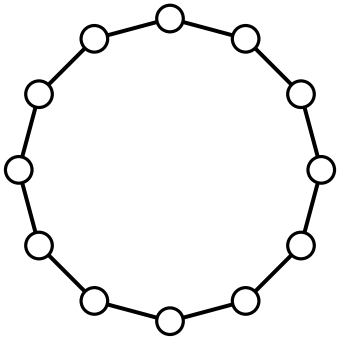


Integer Arboricity

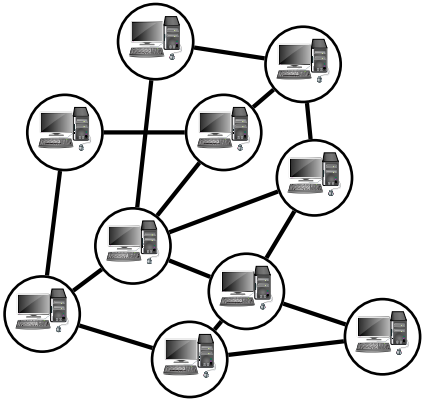
Observation

We need D rounds to discern between a long path and a cycle.

- Hence $+1$ approximation expected for α_i
- For the *value*: can approximate via α_f and in simple graphs via ρ



The CONGEST Model



- $G = (V, E)$, $|V| = n$, $|E| = m$
- G unknown: each node knows its neighbors.
- Communication over edges in synchronous rounds.
- Bandwidth $O(\log n)$ bits per edge.

A CONGEST Algorithm

Simple Graphs

- Densest subgraph [Su, Vu '20]:
  - LOCAL: $O(\log n/\varepsilon)$ rounds;
  - CONGEST: $O(\log^4 n/\varepsilon^4)$ rounds.

A CONGEST Algorithm

Simple Graphs

- Densest subgraph [Su, Vu '20]:
  - LOCAL: $O(\log n/\varepsilon)$ rounds;
  - CONGEST: $O(\log^4 n/\varepsilon^4)$ rounds.

Algorithm for multi-graphs [unpublished]

3. Compute an approximation of α_f via $O(\alpha_f \cdot \text{poly}(\log n, \varepsilon^{-1}))$ MSTs, taking $\tilde{O}(D + \sqrt{n})$ rounds each.

A CONGEST Algorithm

Simple Graphs

- Densest subgraph [Su, Vu '20]:
  - LOCAL: $O(\log n/\varepsilon)$ rounds;
  - CONGEST: $O(\log^4 n/\varepsilon^4)$ rounds.

Algorithm for multi-graphs [unpublished]

1. Use network decompositions to only consider low diameter

3. Compute an approximation of α_f via $O(\alpha_f \cdot \text{poly}(\log n, \varepsilon^{-1}))$ MSTs, taking $\tilde{O}(\sqrt{n})$ rounds each.

A CONGEST Algorithm

Simple Graphs

- Densest subgraph [Su, Vu '20]:
  - LOCAL: $O(\log n/\varepsilon)$ rounds;
  - CONGEST: $O(\log^4 n/\varepsilon^4)$ rounds.

Algorithm for multi-graphs [unpublished]

1. Use network decompositions to only consider low diameter
2. Sample uniformly such that $\alpha_f = O(\log n/\varepsilon^2)$
3. Compute an approximation of α_f via $O(\text{poly}(\log n, \varepsilon^{-1}))$ MSTs, taking $\tilde{O}(\sqrt{n})$ rounds each.

A CONGEST Algorithm

Simple Graphs

- Densest subgraph [Su, Vu '20]:
  - LOCAL: $O(\log n/\varepsilon)$ rounds;
  - CONGEST: $O(\log^4 n/\varepsilon^4)$ rounds.

Algorithm for multi-graphs [unpublished]

1. Use network decompositions to only consider low diameter
2. Sample uniformly such that $\alpha_f = O(\log n/\varepsilon^2)$
3. Compute an approximation of α_f via $O(\text{poly}(\log n, \varepsilon^{-1}))$ MSTs, taking $\tilde{O}(\sqrt{n})$ rounds each.

- Compare to $O(\log n/\varepsilon)$ LOCAL algorithm
- Q: Can we not do much better?

A CONGEST Algorithm

Simple Graphs

- Densest subgraph [Su, Vu '20]:
  - LOCAL: $O(\log n/\varepsilon)$ rounds;
  - CONGEST: $O(\log^4 n/\varepsilon^4)$ rounds.

Algorithm for multi-graphs [unpublished]

1. Use network decompositions to only consider low diameter
2. Sample uniformly such that $\alpha_f = O(\log n/\varepsilon^2)$
3. Compute an approximation of α_f via $O(\text{poly}(\log n, \varepsilon^{-1}))$ MSTs, taking $\tilde{O}(\sqrt{n})$ rounds each.

- Compare to $O(\log n/\varepsilon)$ LOCAL algorithm
- Q: Can we not do much better?
- Q: What about the exact integer arboricity with $o(n^2)$ rounds?

Open Questions

Give a CONGEST algorithm that computes a $(1+\varepsilon)$-approximations of α_f in $o(\sqrt{n})$ rounds (poly $\log n$ rounds?) in a graph with $\alpha_f = O(\log n/\varepsilon^2)$ and $D = O(\log n)$ or show an $\Omega(\sqrt{n})$ lower bound.

Finding the Forests

- Also known as a "Nash-Williams decomposition"
- Goal: Find $c \cdot \alpha_i$ forests to cover G

Finding the Forests

- Also known as a "Nash-Williams decomposition"
- Goal: Find $c \cdot \alpha_i$ forests to cover G
- $O(\log n/\varepsilon)$ CONGEST rounds for $c = 2 + \varepsilon$ [Barenboim, Elkin '08]
  - Akin to the degeneracy algorithm.

Finding the Forests

- Also known as a "Nash-Williams decomposition"
- Goal: Find $c \cdot \alpha_i$ forests to cover G
- $O(\log n/\varepsilon)$ CONGEST rounds for $c = 2 + \varepsilon$ [Barenboim, Elkin '08]
  - Akin to the degeneracy algorithm.
- $O(\log^4 n/\varepsilon)$ LOCAL rounds for $c = 1 + \varepsilon$ and $\alpha_i = \Omega(\log n/\varepsilon)$ [Ghaffari, Su, Vu '21]
  - $+$ results parametrized by Δ

Finding the Forests

- Also known as a "Nash-Williams decomposition"
- Goal: Find $c \cdot \alpha_i$ forests to cover G
- $O(\log n/\varepsilon)$ CONGEST rounds for $c = 2 + \varepsilon$ [Barenboim, Elkin '08]
  - Akin to the degeneracy algorithm.
- $O(\log^4 n/\varepsilon)$ LOCAL rounds for $c = 1 + \varepsilon$ and $\alpha_i = \Omega(\log n/\varepsilon)$ [Ghaffari, Su, Vu '21]
  - $+$ results parametrized by Δ
- Q: what about $c = 1 + \varepsilon$ for $\alpha_i = o(\log n)$?
- Q: what about CONGEST? Bottleneck is finding a $(1 + \varepsilon)\alpha_i$-out orientation

Parallel/MPC/Congested Clique

- Fast approximations of α_f

Parallel/MPC/Congested Clique

- Fast approximations of α_f
  - PRAM: [Plotkin, Shmoys, Tardos '91].

Parallel/MPC/Congested Clique

- Fast approximations of α_f
  - PRAM: [Plotkin, Shmoys, Tardos '91].
  - Can do parallel MST in $O(\log n)$ depth, $O(1)$ CC rounds

Parallel/MPC/Congested Clique

- Fast approximations of α_f
  - PRAM: [Plotkin, Shmoys, Tardos '91].
  - Can do parallel MST in $O(\log n)$ depth, $O(1)$ CC rounds
- Exact α_f and α_i in PRAM

Parallel/MPC/Congested Clique

- Fast approximations of α_f
  - PRAM: [Plotkin, Shmoys, Tardos '91].
  - Can do parallel MST in $O(\log n)$ depth, $O(1)$ CC rounds
- Exact α_f and α_i in PRAM
  - poly log n depth, $\tilde{O}(\alpha^{1+\omega} n^{\omega})$ work [Narayanan, Huzur, Vazirani '92]

Parallel/MPC/Congested Clique

- Fast approximations of α_f
  - PRAM: [Plotkin, Shmoys, Tardos '91].
  - Can do parallel MST in $O(\log n)$ depth, $O(1)$ CC rounds
- Exact α_f and α_i in PRAM
  - poly log n depth, $\tilde{O}(\alpha^{1+\omega} n^{\omega})$ work [Narayanan, Huzur, Vazirani '92]
- Q: Exact in MPC/CC?

Parallel/MPC/Congested Clique

- Fast approximations of α_f
  - PRAM: [Plotkin, Shmoys, Tardos '91].
  - Can do parallel MST in $O(\log n)$ depth, $O(1)$ CC rounds
- Exact α_f and α_i in PRAM
  - poly log n depth, $\tilde{O}(\alpha^{1+\omega} n^{\omega})$ work [Narayanan, Huzur, Vazirani '92]
- Q: Exact in MPC/CC?
- Q: Forest decomposition?

Parallel/MPC/Congested Clique

- Fast approximations of α_f
  - PRAM: [Plotkin, Shmoys, Tardos '91].
  - Can do parallel MST in $O(\log n)$ depth, $O(1)$ CC rounds
- Exact α_f and α_i in PRAM
  - poly log n depth, $\tilde{O}(\alpha^{1+\omega} n^{\omega})$ work [Narayanan, Huzur, Vazirani '92]
- Q: Exact in MPC/CC?
- Q: Forest decomposition?
- Q: Other models? Eg Dynamic graph algorithms

Open Questions on Tree-Packing

- Unweighted, undirected (multi-)graph $G = (V, E)$.
- Goal: $|\ell^{\mathcal{T}}(e) - \ell^*(e)| \leq \varepsilon/\Phi$ for all $e \in E$.[4]
- [Thorup '07] $|\mathcal{T}| = O(\log m \cdot \Phi/\varepsilon^2)$ greedy trees.
- Can we do better?

[4] Recall: Φ is the maximum number of disjoint trees in the graph.

Open Questions on Tree-Packing

- Unweighted, undirected (multi-)graph $G = (V, E)$.
- Goal: $|\ell^{\mathcal{T}}(e) - \ell^*(e)| \leq \varepsilon/\Phi$ for all $e \in E$.[4]
- [Thorup '07] $|\mathcal{T}| = O(\log m \cdot \Phi/\varepsilon^2)$ greedy trees.
- Can we do better?

Theorem [Arkhipov, Kolmogorov '26]

In general, such a greedy tree-packing $\mathcal{T}$ needs $|\mathcal{T}| = \Omega(\Phi/\varepsilon^2)$ trees.

[4] Recall: Φ is the maximum number of disjoint trees in the graph.

Open Questions on Tree-Packing

- Unweighted, undirected (multi-)graph $G = (V, E)$.
- Goal: $|\ell^{\mathcal{T}}(e) - \ell^*(e)| \leq \varepsilon/\Phi$ for all $e \in E$.[4]
- [Thorup '07] $|\mathcal{T}| = O(\log m \cdot \Phi/\varepsilon^2)$ greedy trees.
- Can we do better?

Theorem [Arkhipov, Kolmogorov '26]

In general, such a greedy tree-packing $\mathcal{T}$ needs $|\mathcal{T}| = \Omega(\Phi/\varepsilon^2)$ trees.

Theorem [dV, Christiansen '25]

There exists such a tree-packing $\mathcal{T}$ with $|\mathcal{T}| = \Theta(\Phi/\varepsilon)$ trees.

[4] Recall: Φ is the maximum number of disjoint trees in the graph.

Conclusion

- Arboricity can be computed (approximately) with
  - Augmenting paths
  - Max Flow
  - Directed Min Cut
  - Minimum Spanning Forests
  - Via Out-Orientations
- But still many open questions!

Conclusion

- Arboricity can be computed (approximately) with
  - Augmenting paths
  - Max Flow
  - Directed Min Cut
  - Minimum Spanning Forests
  - Via Out-Orientations
- But still many open questions!

Thank you!