Fast FPT-Approximation of Branchwidth

Fedor V. Fomin, Tuukka Korhonen

University of Bergen, Norway

STOC 2022

In this work

- Framework for designing FPT 2-approximation algorithms for branchwidth of symmetric submodular functions

In this work

- Framework for designing FPT 2-approximation algorithms for branchwidth of symmetric submodular functions
- Applications:

Theorem

There is a $2^{2^{\mathcal{O}(k)}} n^2$ time 2-approximation algorithm for rankwidth.

In this work

- Framework for designing FPT 2-approximation algorithms for branchwidth of symmetric submodular functions
- Applications:

Theorem

There is a $2^{2^{\mathcal{O}(k)}} n^2$ time 2-approximation algorithm for rankwidth.

Theorem

There is a $2^{\mathcal{O}(k)} n$ time 2-approximation algorithm for graph branchwidth.

In this work

- Framework for designing FPT 2-approximation algorithms for branchwidth of symmetric submodular functions
- Applications:

Theorem

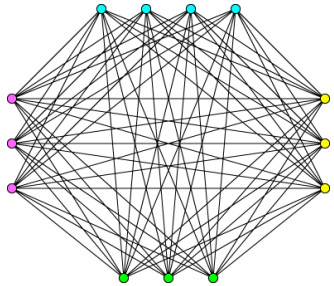
There is a $2^{2^{\mathcal{O}(k)}} n^2$ time 2-approximation algorithm for rankwidth.

Theorem

There is a $2^{\mathcal{O}(k)} n$ time 2-approximation algorithm for graph branchwidth.

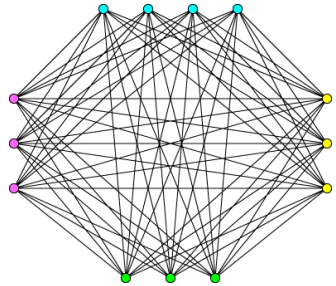
Rankwidth

- Measures graph decomposition by low-rank cuts



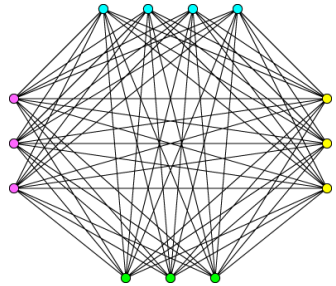
Rankwidth

- Measures graph decomposition by low-rank cuts
- Generalization of treewidth, but can be bounded also for dense graphs



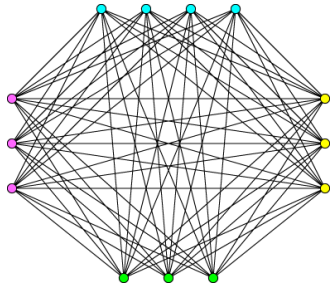
Rankwidth

- Measures graph decomposition by low-rank cuts
- Generalization of treewidth, but can be bounded also for dense graphs
  - cliques, cographs, distance-hereditary, k-leaf-power...

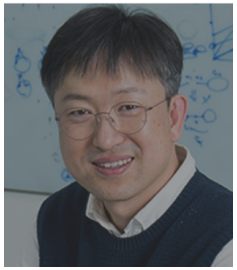


Rankwidth

- Measures graph decomposition by low-rank cuts
- Generalization of treewidth, but can be bounded also for dense graphs
  - cliques, cographs, distance-hereditary, k-leaf-power...

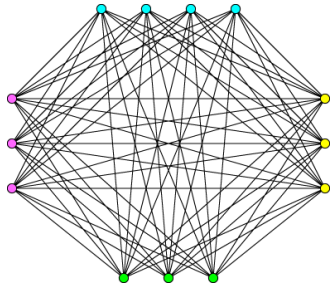


- Introduced by [Oum & Seymour, '06] to approximate cliquewidth:

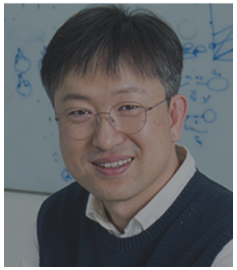


Rankwidth

- Measures graph decomposition by low-rank cuts
- Generalization of treewidth, but can be bounded also for dense graphs
  - cliques, cographs, distance-hereditary, k-leaf-power...

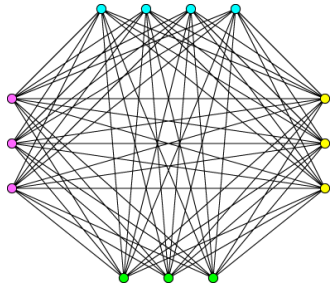


- Introduced by [Oum & Seymour, '06] to approximate cliquewidth:
  - $\mathtt{rw}(G) \leq \mathtt{cw}(G) \leq 2^{\mathtt{rw}(G)+1}$

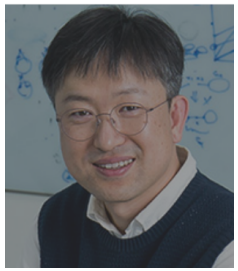


Rankwidth

- Measures graph decomposition by low-rank cuts
- Generalization of treewidth, but can be bounded also for dense graphs
  - cliques, cographs, distance-hereditary, k-leaf-power...

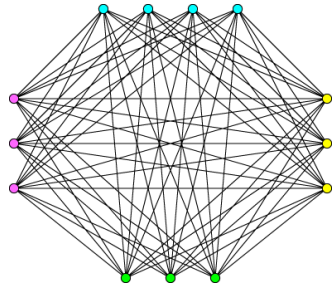


- Introduced by [Oum & Seymour, '06] to approximate cliquewidth:
  - $\mathtt{rw}(G) \leq \mathtt{cw}(G) \leq 2^{\mathtt{rw}(G)+1}$
  - $f(k)n^9 \log n$ time 3-approximation algorithm for rankwidth

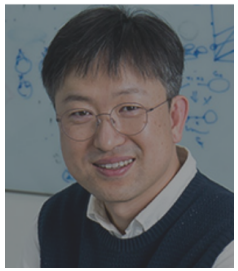


Rankwidth

- Measures graph decomposition by low-rank cuts
- Generalization of treewidth, but can be bounded also for dense graphs
  - cliques, cographs, distance-hereditary, k-leaf-power...



- Introduced by [Oum & Seymour, '06] to approximate cliquewidth:
  - $\mathtt{rw}(G) \leq \mathtt{cw}(G) \leq 2^{\mathtt{rw}(G)+1}$
  - $f(k)n^9 \log n$ time 3-approximation algorithm for rankwidth



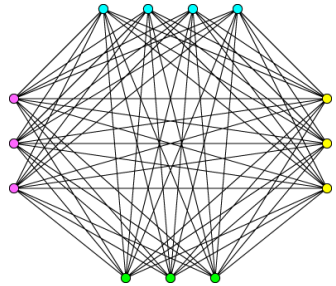
"Courcelle's theorem" for rankwidth/cliquewidth

[Courcelle, Makowsky, & Rotics, '00], [Oum & Seymour, '06]

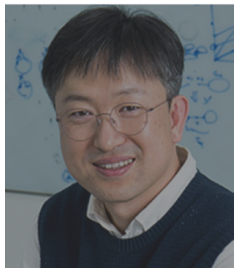
Given a graph with a rank decomposition of width k, any $\mathbf{MSO}_1$-definable problem can be solved in $f(k)n^2$ time

Rankwidth

- Measures graph decomposition by low-rank cuts
- Generalization of treewidth, but can be bounded also for dense graphs
  - cliques, cographs, distance-hereditary, k-leaf-power...



- Introduced by [Oum & Seymour, '06] to approximate cliquewidth:
  - $\mathtt{rw}(G) \leq \mathtt{cw}(G) \leq 2^{\mathtt{rw}(G)+1}$
  - $f(k)n^9 \log n$ time 3-approximation algorithm for rankwidth



"Courcelle's theorem" for rankwidth/cliquewidth

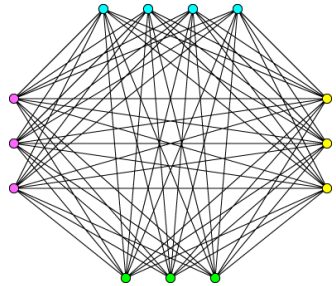
[Courcelle, Makowsky, & Rotics, '00], [Oum & Seymour, '06]

Given a graph with a rank decomposition of width k, any $\mathbf{MSO}_1$-definable problem can be solved in $f(k)n^2$ time

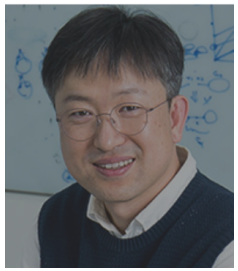
- $f(k)n^3$ 3-approximation for rankwidth [Oum '08]

Rankwidth

- Measures graph decomposition by low-rank cuts
- Generalization of treewidth, but can be bounded also for dense graphs
  - cliques, cographs, distance-hereditary, k-leaf-power...



- Introduced by [Oum & Seymour, '06] to approximate cliquewidth:
  - $\mathtt{rw}(G) \leq \mathtt{cw}(G) \leq 2^{\mathtt{rw}(G)+1}$
  - $f(k)n^9 \log n$ time 3-approximation algorithm for rankwidth



"Courcelle's theorem" for rankwidth/cliquewidth

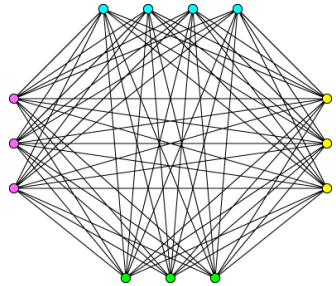
[Courcelle, Makowsky, & Rotics, '00], [Oum & Seymour, '06]

Given a graph with a rank decomposition of width k, any $\mathbf{MSO}_1$-definable problem can be solved in $f(k)n^2$ time

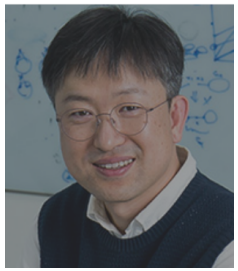
- $f(k)n^3$ 3-approximation for rankwidth [Oum '08]
- $f(k)n^3$ exact algorithm for rankwidth [Hlineny & Oum, '08]

Rankwidth

- Measures graph decomposition by low-rank cuts
- Generalization of treewidth, but can be bounded also for dense graphs
  - cliques, cographs, distance-hereditary, k-leaf-power...



- Introduced by [Oum & Seymour, '06] to approximate cliquewidth:
  - $\mathtt{rw}(G) \leq \mathtt{cw}(G) \leq 2^{\mathtt{rw}(G)+1}$
  - $f(k)n^9 \log n$ time 3-approximation algorithm for rankwidth



"Courcelle's theorem" for rankwidth/cliquewidth

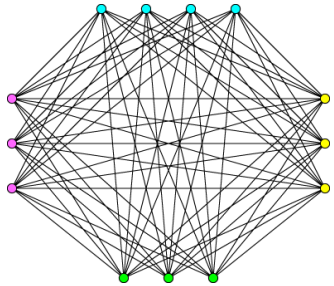
[Courcelle, Makowsky, & Rotics, '00], [Oum & Seymour, '06]

Given a graph with a rank decomposition of width k, any $\mathbf{MSO}_1$-definable problem can be solved in $f(k)n^2$ time

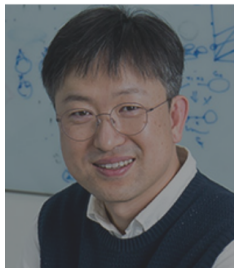
- $f(k)n^3$ 3-approximation for rankwidth [Oum '08]
- $f(k)n^3$ exact algorithm for rankwidth [Hlineny & Oum, '08]
- In this work: $f(k)n^2$ time 2-approximation

Rankwidth

- Measures graph decomposition by low-rank cuts
- Generalization of treewidth, but can be bounded also for dense graphs
  - cliques, cographs, distance-hereditary, k-leaf-power...



- Introduced by [Oum & Seymour, '06] to approximate cliquewidth:
  - $\mathtt{rw}(G) \leq \mathtt{cw}(G) \leq 2^{\mathtt{rw}(G)+1}$
  - $f(k)n^9 \log n$ time 3-approximation algorithm for rankwidth



"Courcelle's theorem" for rankwidth/cliquewidth

[Courcelle, Makowsky, & Rotics, '00], [Oum & Seymour, '06]

Given a graph with a rank decomposition of width k, any **MSO**$_1$-definable problem can be solved in $f(k)n^2$ time

- $f(k)n^3$ 3-approximation for rankwidth [Oum '08]
- $f(k)n^3$ exact algorithm for rankwidth [Hlineny & Oum, '08]
- In this work: $f(k)n^2$ time 2-approximation $\Rightarrow$

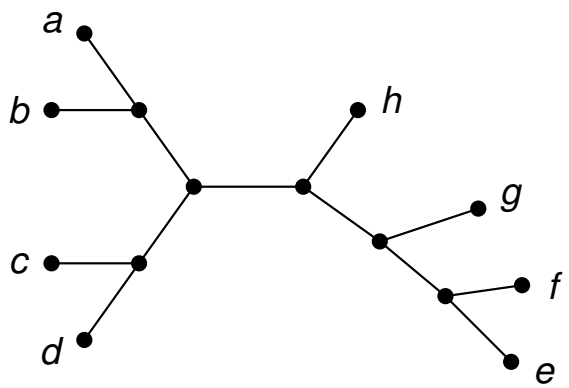
Given a graph of rankwidth k, any **MSO**$_1$-definable problem can be solved in $f(k)n^2$ time

Branchwidth of Connectivity function

- Let V be a set and $f : 2^V \to \mathbb{Z}$ a connectivity function:
  - Symmetric: For any $A \subseteq V$, it holds that $f(A) = f(\overline{A})$, where $\overline{A} = V \setminus A$
  - Submodular: For any $A, B \subseteq V$, it holds that $f(A \cup B) + f(A \cap B) \leq f(A) + f(B)$

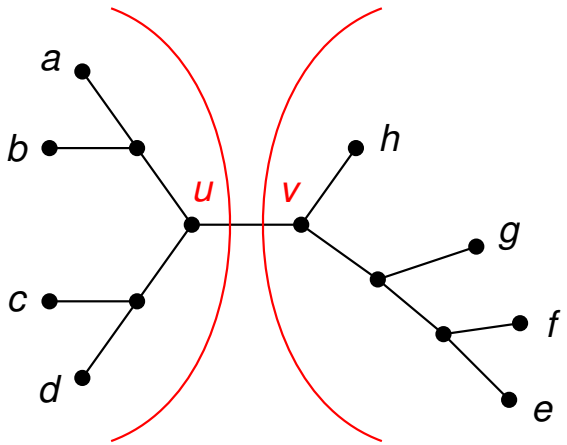
Branchwidth of Connectivity function

- Let V be a set and $f : 2^V \to \mathbb{Z}$ a connectivity function:
  - Symmetric: For any $A \subseteq V$, it holds that $f(A) = f(\overline{A})$, where $\overline{A} = V \setminus A$
  - Submodular: For any $A, B \subseteq V$, it holds that $f(A \cup B) + f(A \cap B) \leq f(A) + f(B)$
- Branch decomposition of f is a cubic tree whose leaves are the elements of V
  - Example with $V = \{a, b, c, d, e, f, g, h\}$:



Branchwidth of Connectivity function

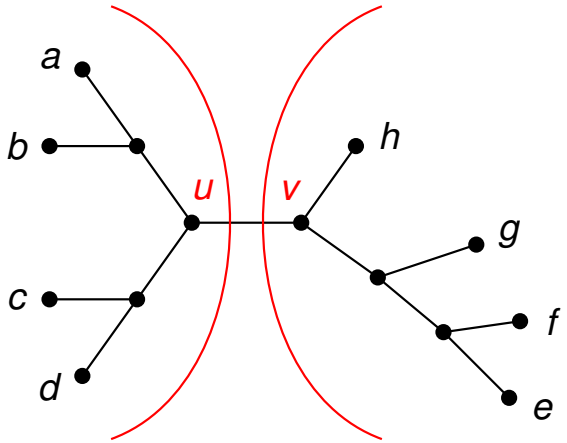
- Let V be a set and $f : 2^V \to \mathbb{Z}$ a connectivity function:
  - Symmetric: For any $A \subseteq V$, it holds that $f(A) = f(\overline{A})$, where $\overline{A} = V \setminus A$
  - Submodular: For any $A, B \subseteq V$, it holds that $f(A \cup B) + f(A \cap B) \leq f(A) + f(B)$
- Branch decomposition of f is a cubic tree whose leaves are the elements of V
  - Example with $V = \{a, b, c, d, e, f, g, h\}$:



- We denote $f(uv) = f(\{a, b, c, d\}) = f(\{e, f, g, h\})$
- The width of the decomposition is $\max_{uv \in E(T)} f(uv)$

Branchwidth of Connectivity function

- Let V be a set and $f : 2^V \to \mathbb{Z}$ a connectivity function:
  - Symmetric: For any $A \subseteq V$, it holds that $f(A) = f(\overline{A})$, where $\overline{A} = V \setminus A$
  - Submodular: For any $A, B \subseteq V$, it holds that $f(A \cup B) + f(A \cap B) \leq f(A) + f(B)$
- Branch decomposition of f is a cubic tree whose leaves are the elements of V
  - Example with $V = \{a, b, c, d, e, f, g, h\}$:



- We denote $f(uv) = f(\{a, b, c, d\}) = f(\{e, f, g, h\})$
- The width of the decomposition is $\max_{uv \in E(T)} f(uv)$
- The branchwidth of f is minimum width of a branch decomposition of f

Examples

- Branchwidth of a graph:
  - $V = E(G)$
  - $f(A)$ is the number of vertices incident to edges in both A and $\overline{A}$

Examples

- Branchwidth of a graph:
  - $V = E(G)$
  - $f(A)$ is the number of vertices incident to edges in both A and $\overline{A}$
- Rankwidth of a graph:
  - $V = V(G)$
  - $f(A)$ is the GF(2) rank of the $|A| \times |\overline{A}|$ matrix representing $G[A, \overline{A}]$

Examples

- Branchwidth of a graph:
  - $V = E(G)$
  - $f(A)$ is the number of vertices incident to edges in both A and $\overline{A}$
- Rankwidth of a graph:
  - $V = V(G)$
  - $f(A)$ is the GF(2) rank of the $|A| \times |\overline{A}|$ matrix representing $G[A, \overline{A}]$
- Also carving-width, matroid branchwidth, rankwidth in different fields...

Our Framework

Outline

Framework for $f(k)n$ time 2-approximation *compression* algorithms:

Outline

Framework for $f(k)n$ time 2-approximation *compression* algorithms:

- Given a branch decomposition T of f of width k, either output branch decomposition of width $< k$, or conclude that $k \leq 2\mathrm{bw}(f)$

Outline

Framework for $f(k)n$ time 2-approximation *compression* algorithms:

- Given a branch decomposition T of f of width k, either output branch decomposition of width $< k$, or conclude that $k \leq 2\mathrm{bw}(f)$

Specifically:

- For rankwidth: $2^{2^{\mathcal{O}(k)}}n$ time compression algorithm where input/output decompositions are *augmented*

Outline

Framework for $f(k)n$ time 2-approximation *compression* algorithms:

- Given a branch decomposition T of f of width k, either output branch decomposition of width $< k$, or conclude that $k \leq 2\mathrm{bw}(f)$

Specifically:

- For rankwidth: $2^{2^{\mathcal{O}(k)}}n$ time compression algorithm where input/output decompositions are *augmented*
  - Apply n times to get $2^{2^{\mathcal{O}(k)}}n^2$ time algorithm

Outline

Framework for $f(k)n$ time 2-approximation *compression* algorithms:

- Given a branch decomposition T of f of width k, either output branch decomposition of width $< k$, or conclude that $k \leq 2\mathrm{bw}(f)$

Specifically:

- For rankwidth: $2^{2^{\mathcal{O}(k)}}n$ time compression algorithm where input/output decompositions are *augmented*
  - Apply n times to get $2^{2^{\mathcal{O}(k)}}n^2$ time algorithm
- For graph branchwidth: $2^{\mathcal{O}(k)}n$ time compression algorithm

Outline

Framework for $f(k)n$ time 2-approximation *compression* algorithms:

- Given a branch decomposition T of f of width k, either output branch decomposition of width $< k$, or conclude that $k \leq 2\mathrm{bw}(f)$

Specifically:

- For rankwidth: $2^{2^{\mathcal{O}(k)}}n$ time compression algorithm where input/output decompositions are *augmented*
  - Apply n times to get $2^{2^{\mathcal{O}(k)}}n^2$ time algorithm
- For graph branchwidth: $2^{\mathcal{O}(k)}n$ time compression algorithm
  - Apply with treewidth approximation to get $2^{\mathcal{O}(k)}n$ time algorithm

Framework

- Input: Branch decomposition T of function f of width k

Framework

- Input: Branch decomposition T of function f of width k

- Combinatorial result:
  - An edge uv of the decomposition is *heavy* if $f(uv) = k$
  - If $k > 2\mathrm{bw}(f)$, then a **refinement operation** can be applied, which decreases the number of heavy edges and does not increase the width

Framework

- Input: Branch decomposition T of function f of width k
- Combinatorial result:
  - An edge uv of the decomposition is *heavy* if $f(uv) = k$
  - If $k > 2\mathrm{bw}(f)$, then a **refinement operation** can be applied, which decreases the number of heavy edges and does not increase the width
- Algorithmic result:
  - Assume dynamic programming in time $t(k)$ per node

$\Rightarrow$ A sequence of refinement operations, either improving width or concluding $k \leq 2\mathrm{bw}(f)$, can be performed in time $t(k)2^{\mathcal{O}(k)}n$

Framework

- Input: Branch decomposition T of function f of width k
- Combinatorial result:
  - An edge uv of the decomposition is *heavy* if $f(uv) = k$
  - If $k > 2\mathrm{bw}(f)$, then a **refinement operation** can be applied, which decreases the number of heavy edges and does not increase the width
- Algorithmic result:
  - Assume dynamic programming in time $t(k)$ per node

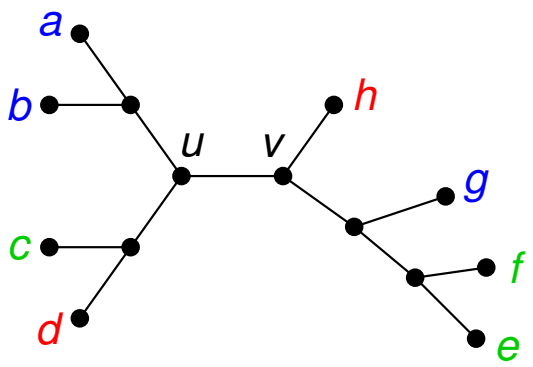
  $\Rightarrow$ A sequence of refinement operations, either improving width or concluding $k \leq 2\mathrm{bw}(f)$, can be performed in time $t(k)2^{\mathcal{O}(k)}n$
  - For rankwidth $t(k) = 2^{2^{\mathcal{O}(k)}}$, for graph branchwidth $t(k) = 2^{\mathcal{O}(k)}$

Refinement operation

Specified by 4-tuple (uv, C_1, C_2, C_3), where $uv \in E(T)$ and (C_1, C_2, C_3) tripartition of V

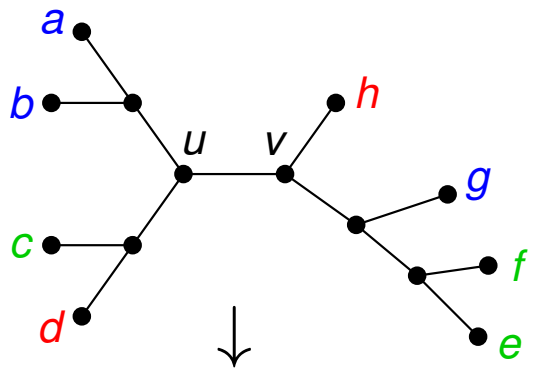
Refinement operation

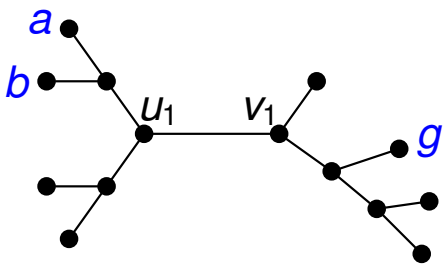
Specified by 4-tuple (uv, C_1, C_2, C_3), where $uv \in E(T)$ and (C_1, C_2, C_3) tripartition of V
Example with $(uv, C_1, C_2, C_3) = (uv, \{a, b, g\}, \{c, e, f\}, \{d, h\})$:

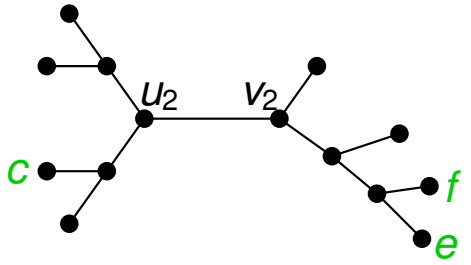


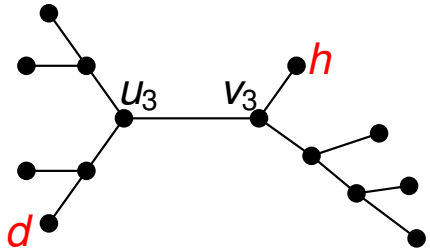
Refinement operation

Specified by 4-tuple (uv, C_1, C_2, C_3), where $uv \in E(T)$ and (C_1, C_2, C_3) tripartition of V
Example with $(uv, C_1, C_2, C_3) = (uv, \{a, b, g\}, \{c, e, f\}, \{d, h\})$:



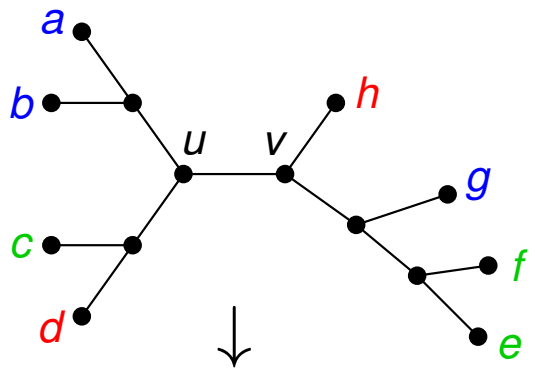


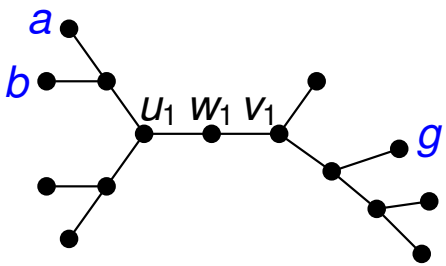


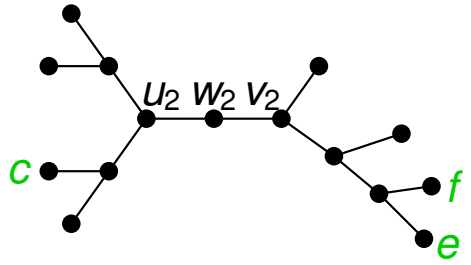


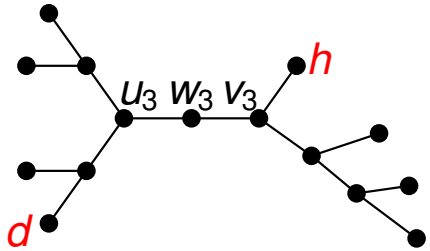
Refinement operation

Specified by 4-tuple (uv, C_1, C_2, C_3), where $uv \in E(T)$ and (C_1, C_2, C_3) tripartition of V
Example with $(uv, C_1, C_2, C_3) = (uv, \{a, b, g\}, \{c, e, f\}, \{d, h\})$:



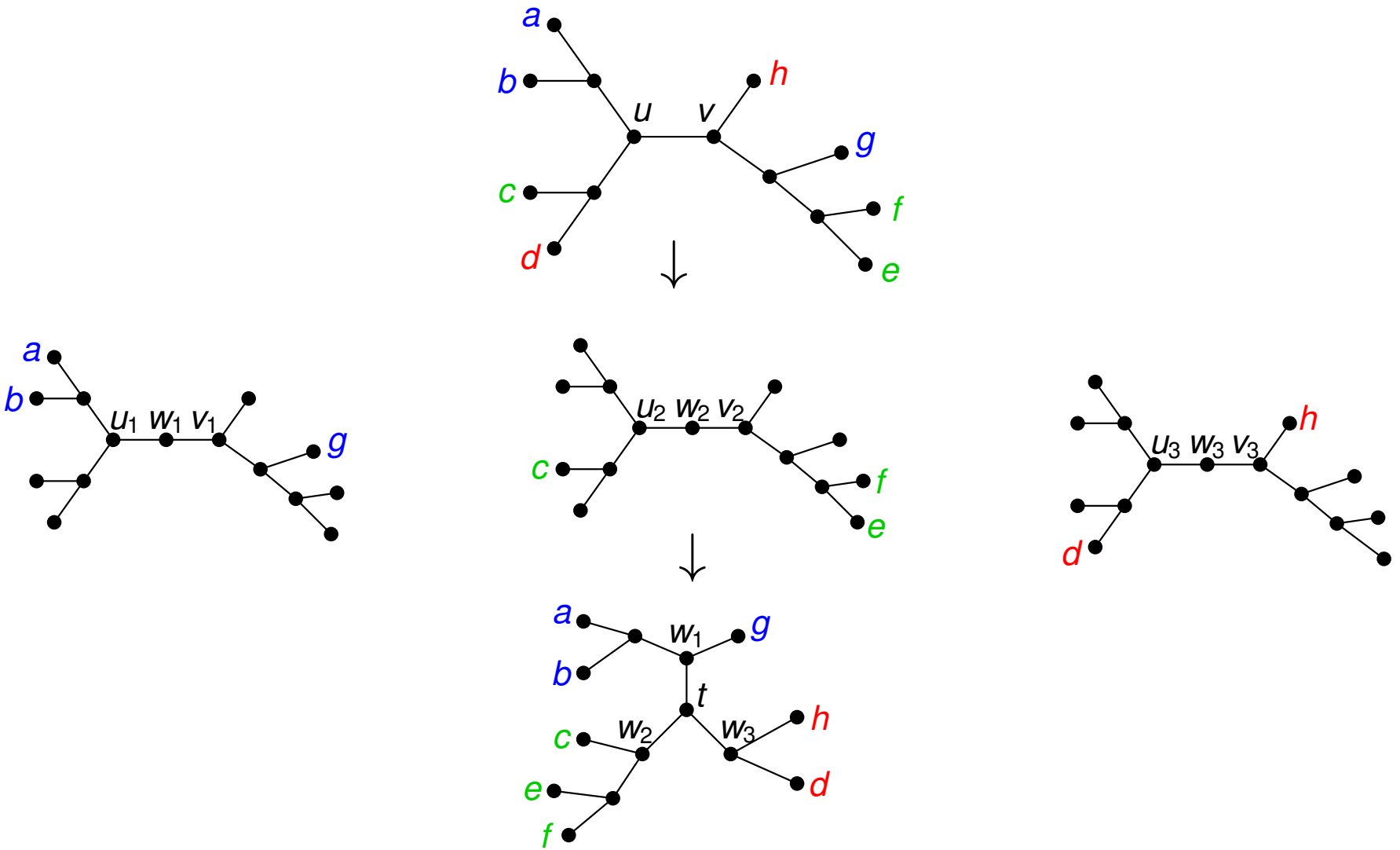






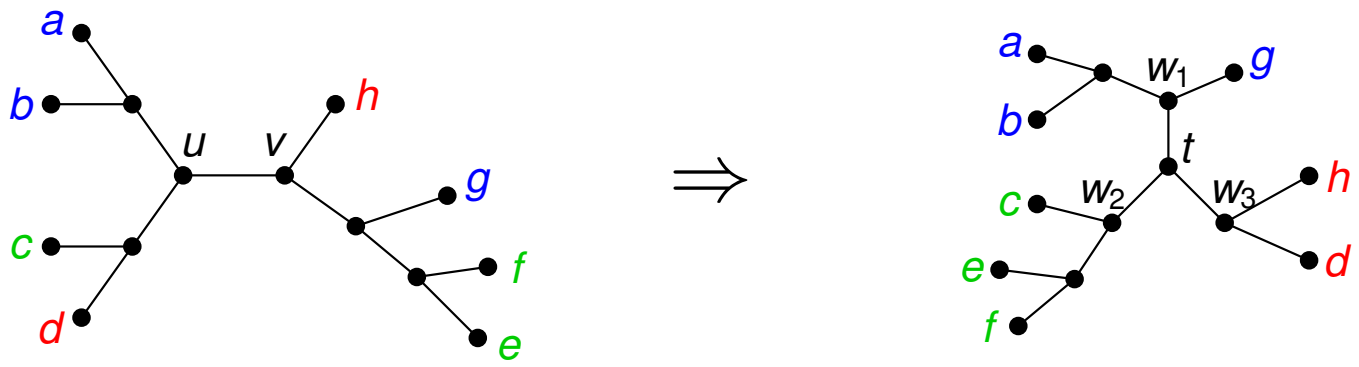
Refinement operation

Specified by 4-tuple (uv, C_1, C_2, C_3), where $uv \in E(T)$ and (C_1, C_2, C_3) tripartition of V
Example with $(uv, C_1, C_2, C_3) = (uv, \{a, b, g\}, \{c, e, f\}, \{d, h\})$:



Combinatorial result

Example with $(uv, C_1, C_2, C_3) = (uv, \{a, b, g\}, \{c, e, f\}, \{d, h\})$:

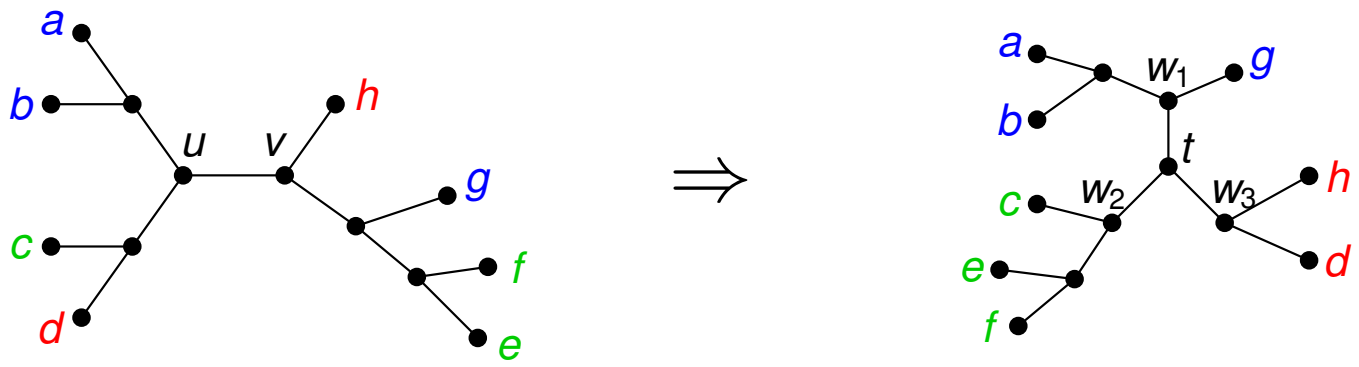


Theorem (Informal)

If $f(uv) > 2\mathtt{bw}(f)$, there exists a refinement (uv, C_1, C_2, C_3) that "locally improves" T.

Combinatorial result

Example with $(uv, C_1, C_2, C_3) = (uv, \{a, b, g\}, \{c, e, f\}, \{d, h\})$:



Theorem (Informal)

If $f(uv) > 2\mathtt{bw}(f)$, there exists a refinement (uv, C_1, C_2, C_3) that "locally improves" T.

Theorem (Informal)

If there exists a refinement (uv, C_1, C_2, C_3) that "locally improves" T, then if the partition (C_1, C_2, C_3) is selected to optimize certain criteria, the refinement globally improves T.

First Algorithm

- General compression algorithm:

1. Let T have width k, select edge uv with $f(uv) = k$

First Algorithm

- General compression algorithm:

1. Let T have width k, select edge uv with $f(uv) = k$
2. Root T at uv, denote $(W, \overline{W})$ the cut of uv

First Algorithm

- General compression algorithm:

1. Let T have width k, select edge uv with $f(uv) = k$
2. Root T at uv, denote $(W, \overline{W})$ the cut of uv
3. Use dynamic programming to find (uv, C_1, C_2, C_3) or conclude $k \leq 2\mathtt{bw}(f)$

First Algorithm

- General compression algorithm:

1. Let T have width k, select edge uv with $f(uv) = k$
2. Root T at uv, denote $(W, \overline{W})$ the cut of uv
3. Use dynamic programming to find (uv, C_1, C_2, C_3) or conclude $k \leq 2\mathtt{bw}(f)$
4. If (uv, C_1, C_2, C_3) found, refine T using it

First Algorithm

- General compression algorithm:

1. Let T have width k, select edge uv with $f(uv) = k$
2. Root T at uv, denote $(W, \overline{W})$ the cut of uv
3. Use dynamic programming to find (uv, C_1, C_2, C_3) or conclude $k \leq 2\mathtt{bw}(f)$
4. If (uv, C_1, C_2, C_3) found, refine T using it
5. Repeat 1-4 until the width of T decreases (at most n iterations)

First Algorithm

- General compression algorithm:

1. Let T have width k, select edge uv with $f(uv) = k$
2. Root T at uv, denote $(W, \overline{W})$ the cut of uv
3. Use dynamic programming to find (uv, C_1, C_2, C_3) or conclude $k \leq 2\mathtt{bw}(f)$
4. If (uv, C_1, C_2, C_3) found, refine T using it
5. Repeat 1-4 until the width of T decreases (at most n iterations)

$\Rightarrow$ Total time complexity $t(k) \cdot n^2$, where $t(k)$ time complexity of dynamic programming per node

First Algorithm

- General compression algorithm:

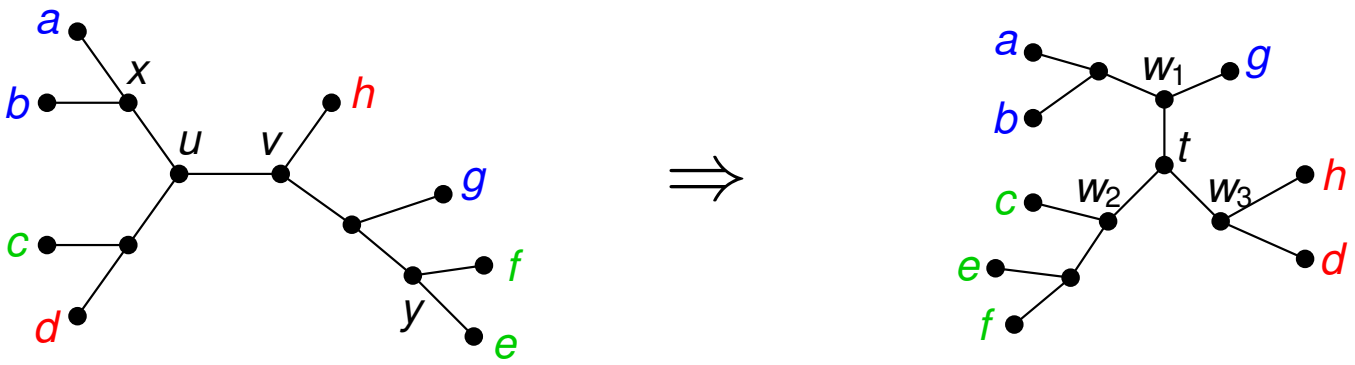
1. Let T have width k, select edge uv with $f(uv) = k$
2. Root T at uv, denote $(W, \overline{W})$ the cut of uv
3. Use dynamic programming to find (uv, C_1, C_2, C_3) or conclude $k \leq 2\mathtt{bw}(f)$
4. If (uv, C_1, C_2, C_3) found, refine T using it
5. Repeat 1-4 until the width of T decreases (at most n iterations)

$\Rightarrow$ Total time complexity $t(k) \cdot n^2$, where $t(k)$ time complexity of dynamic programming per node

- Too slow! Goal is linear in n

Linear-time Algorithm

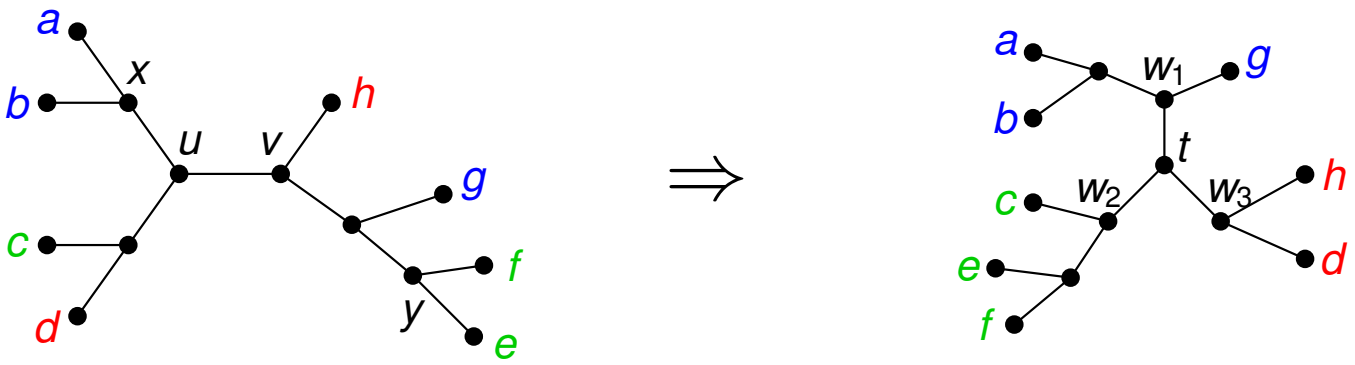
Example with $(uv, C_1, C_2, C_3) = (uv, \{a, b, g\}, \{c, e, f\}, \{d, h\})$



- Consider T rooted at uv, for a node x denote by $T_{uv}[x]$ the leafs below x

Linear-time Algorithm

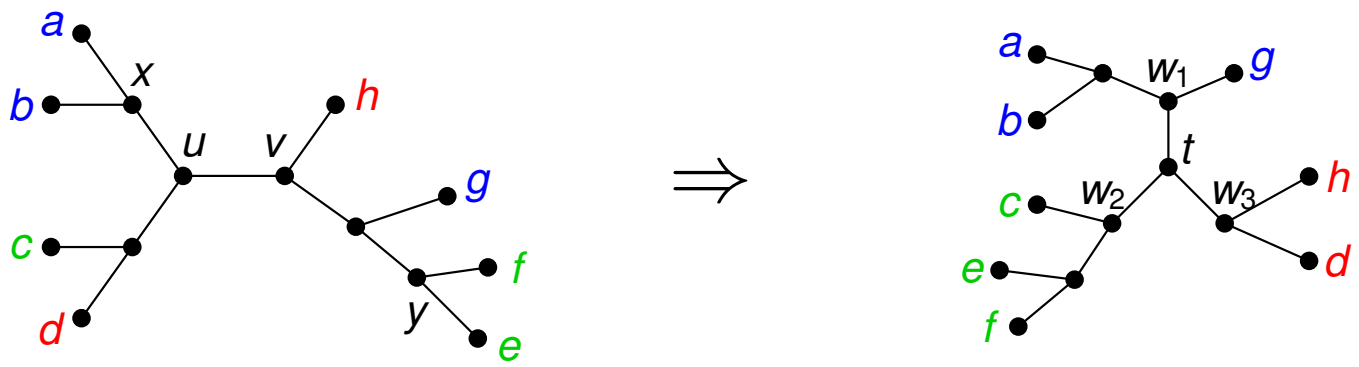
Example with $(uv, C_1, C_2, C_3) = (uv, \{a, b, g\}, \{c, e, f\}, \{d, h\})$



- Consider T rooted at uv, for a node x denote by $T_{uv}[x]$ the leafs below x
  - Example: $T_{uv}[x] = \{a, b\}$ and $T_{uv}[y] = \{e, f\}$

Linear-time Algorithm

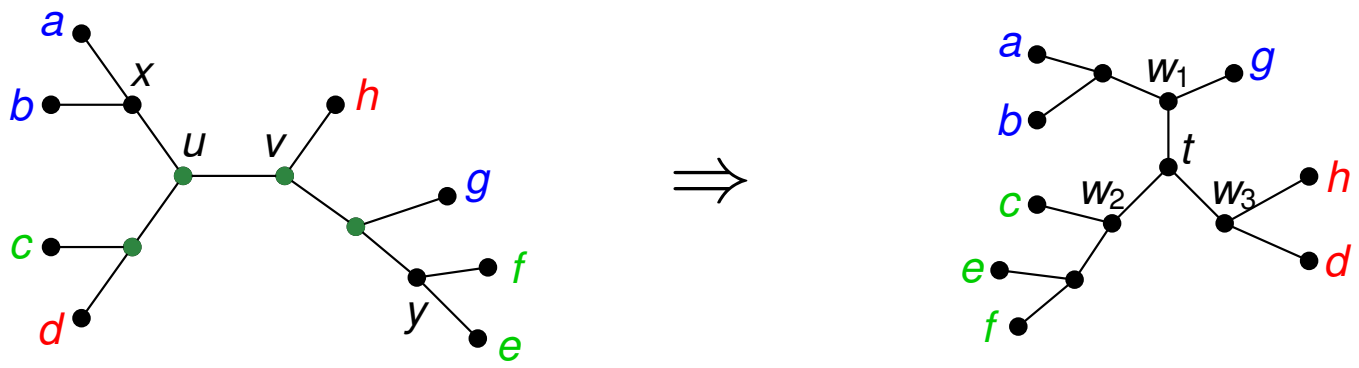
Example with $(uv, C_1, C_2, C_3) = (uv, \{a, b, g\}, \{c, e, f\}, \{d, h\})$



- Consider T rooted at uv, for a node x denote by $T_{uv}[x]$ the leafs below x
  - Example: $T_{uv}[x] = \{a, b\}$ and $T_{uv}[y] = \{e, f\}$
- Observation: If $T_{uv}[x] \subseteq C_i$, then the subtree of x appears identically in refinement

Linear-time Algorithm

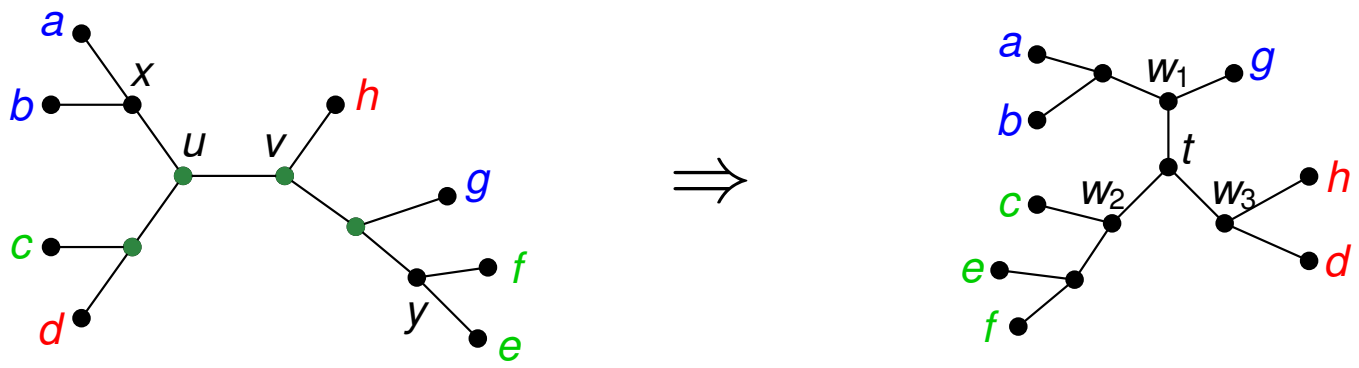
Example with $(uv, C_1, C_2, C_3) = (uv, \{a, b, g\}, \{c, e, f\}, \{d, h\})$



- Consider T rooted at uv, for a node x denote by $T_{uv}[x]$ the leafs below x
  - Example: $T_{uv}[x] = \{a, b\}$ and $T_{uv}[y] = \{e, f\}$
- Observation: If $T_{uv}[x] \subseteq C_i$, then the subtree of x appears identically in refinement
- Call the nodes for which this does **not** happen the edit set R of the refinement

Linear-time Algorithm

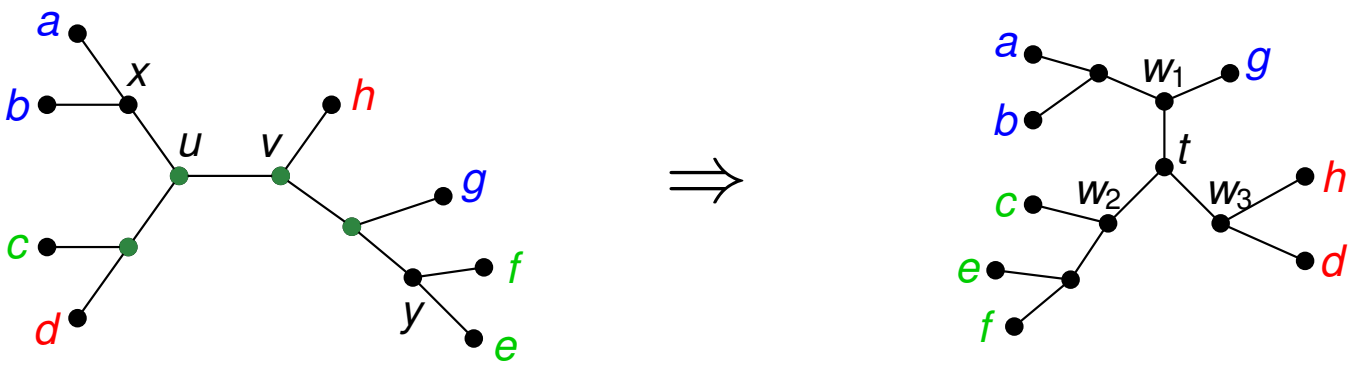
Example with $(uv, C_1, C_2, C_3) = (uv, \{a, b, g\}, \{c, e, f\}, \{d, h\})$



- Consider T rooted at uv, for a node x denote by $T_{uv}[x]$ the leafs below x
  - Example: $T_{uv}[x] = \{a, b\}$ and $T_{uv}[y] = \{e, f\}$
- Observation: If $T_{uv}[x] \subseteq C_i$, then the subtree of x appears identically in refinement
- Call the nodes for which this does **not** happen the edit set R of the refinement
  - Implement refinement by changing only R, in time $\mathcal{O}(t(n) \cdot |R|)$

Linear-time Algorithm

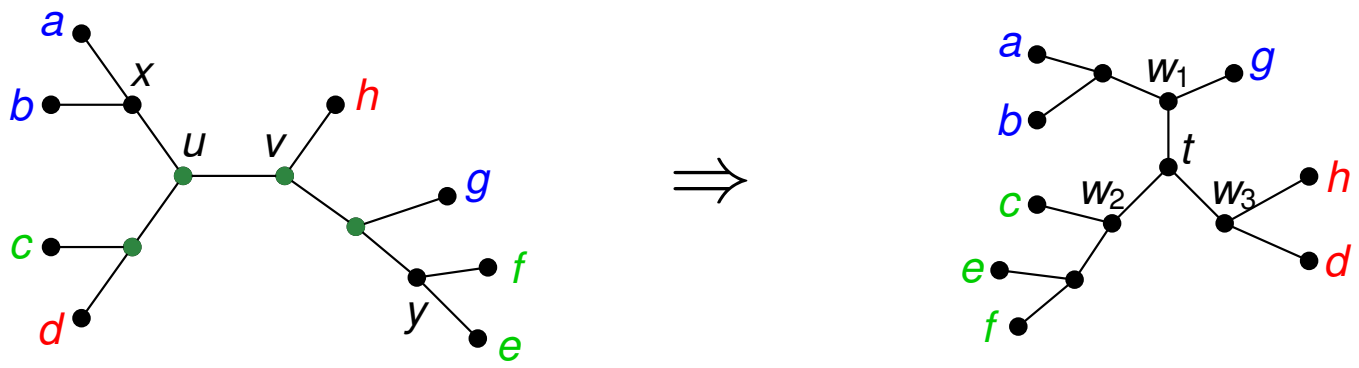
Example with $(uv, C_1, C_2, C_3) = (uv, \{a, b, g\}, \{c, e, f\}, \{d, h\})$



- Consider T rooted at uv, for a node x denote by $T_{uv}[x]$ the leafs below x
  - Example: $T_{uv}[x] = \{a, b\}$ and $T_{uv}[y] = \{e, f\}$
- Observation: If $T_{uv}[x] \subseteq C_i$, then the subtree of x appears identically in refinement
- Call the nodes for which this does **not** happen the edit set R of the refinement
  - Implement refinement by changing only R, in time $\mathcal{O}(t(n) \cdot |R|)$
  - Over any sequence of refinements, $\sum |R| = \mathcal{O}(3^k \cdot k \cdot n)$

Linear-time Algorithm

Example with $(uv, C_1, C_2, C_3) = (uv, \{a, b, g\}, \{c, e, f\}, \{d, h\})$



- Consider T rooted at uv, for a node x denote by $T_{uv}[x]$ the leafs below x
  - Example: $T_{uv}[x] = \{a, b\}$ and $T_{uv}[y] = \{e, f\}$
- Observation: If $T_{uv}[x] \subseteq C_i$, then the subtree of x appears identically in refinement
- Call the nodes for which this does **not** happen the edit set R of the refinement
  - Implement refinement by changing only R, in time $\mathcal{O}(t(n) \cdot |R|)$
  - Over any sequence of refinements, $\sum |R| = \mathcal{O}(3^k \cdot k \cdot n)$
- Walk over the decomposition and refine whenever seeing edge uv with $f(uv) = k$

The end

Thanks for watching!

Paper: https://arxiv.org/abs/2111.03492

Slides: https://tuukkakorhonen.com