Fast FPT-Approximation of Branchwidth

Fedor V. Fomin, Tuukka Korhonen

University of Bergen, Norway

Parametrized complexity and discrete optimization
December 10, 2021

Branchwidth

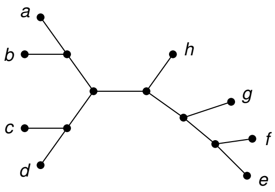
- Let V be a set and $f : 2^V \to \mathbb{Z}_{\geq 0}$ a symmetric set function.
  - Symmetric: For any $A \subseteq V$, it holds that $f(A) = f(\overline{A})$, where $\overline{A} = V \setminus A$

Branchwidth

- Let V be a set and $f : 2^V \to \mathbb{Z}_{\geq 0}$ a symmetric set function.
  - Symmetric: For any $A \subseteq V$, it holds that $f(A) = f(\overline{A})$, where $\overline{A} = V \setminus A$

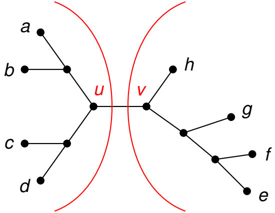
- Branch decomposition of f is a cubic tree whose leaves correspond to V

- Example with $V = \{a, b, c, d, e, f, g, h\}$:



Branchwidth

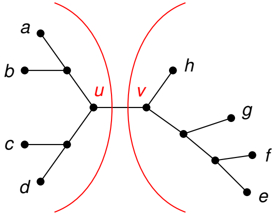
- Let V be a set and $f : 2^V \to \mathbb{Z}_{\geq 0}$ a symmetric set function.
  - Symmetric: For any $A \subseteq V$, it holds that $f(A) = f(\overline{A})$, where $\overline{A} = V \setminus A$
- Branch decomposition of f is a cubic tree whose leaves correspond to V
- Example with $V = \{a, b, c, d, e, f, g, h\}$:



- Each edge of decomposition corresponds to a bipartition of V
- Example: uv corresponds to $(\{a, b, c, d\}, \{e, f, g, h\})$

Branchwidth

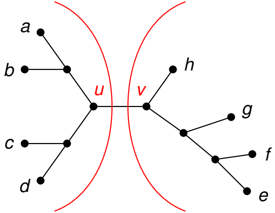
- Let V be a set and $f : 2^V \to \mathbb{Z}_{\geq 0}$ a symmetric set function.
  - Symmetric: For any $A \subseteq V$, it holds that $f(A) = f(\overline{A})$, where $\overline{A} = V \setminus A$
- Branch decomposition of f is a cubic tree whose leaves correspond to V
- Example with $V = \{a, b, c, d, e, f, g, h\}$:



- Each edge of decomposition corresponds to a bipartition of V
- Example: uv corresponds to $(\{a, b, c, d\}, \{e, f, g, h\})$
- 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

- Let V be a set and $f : 2^V \to \mathbb{Z}_{\geq 0}$ a symmetric set function.
  - Symmetric: For any $A \subseteq V$, it holds that $f(A) = f(\overline{A})$, where $\overline{A} = V \setminus A$
- Branch decomposition of f is a cubic tree whose leaves correspond to V
- Example with $V = \{a, b, c, d, e, f, g, h\}$:



- Each edge of decomposition corresponds to a bipartition of V
- Example: uv corresponds to $(\{a, b, c, d\}, \{e, f, g, h\})$
- 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

Connectivity functions

- Function $f : 2^V \to \mathbb{Z}_{\geq 0}$ is a connectivity function if for any $A, B \subseteq V$:
  - $f(A) = f(\overline{A})$ (symmetric)
  - $f(A \cup B) + f(A \cap B) \leq f(A) + f(B)$ (submodular)

Connectivity functions

- Function $f : 2^V \to \mathbb{Z}_{\geq 0}$ is a connectivity function if for any $A, B \subseteq V$:
  - $f(A) = f(\overline{A})$ (symmetric)
  - $f(A \cup B) + f(A \cap B) \leq f(A) + f(B)$ (submodular)

Examples:

- Border of an edge set in a graph G:
  - For any edge set $A \subseteq E(G)$ let $\delta_G(A)$ be the number of vertices incident to both A and $\overline{A}$.
  - The branchwidth of G is the branchwidth of δ_G.

Connectivity functions

- Function $f : 2^V \to \mathbb{Z}_{\geq 0}$ is a connectivity function if for any $A, B \subseteq V$:
  - $f(A) = f(\overline{A})$ (symmetric)
  - $f(A \cup B) + f(A \cap B) \leq f(A) + f(B)$ (submodular)

Examples:

- Border of an edge set in a graph G:
  - For any edge set $A \subseteq E(G)$ let $\delta_G(A)$ be the number of vertices incident to both A and $\overline{A}$.
  - The branchwidth of G is the branchwidth of δ_G.
- Cut-rank in a graph G:
  - For any vertex set $A \subseteq V(G)$ let $\mathrm{cutrk}_G(A)$ be the GF(2)-rank of the $|A| \times |\overline{A}|$ matrix representing edges between A and $\overline{A}$.
  - The rankwidth of G is the branchwidth of cutrk_G.

In this work

- Framework for designing FPT 2-approximation algorithms for branchwidth of connectivity functions

In this work

- Framework for designing FPT 2-approximation algorithms for branchwidth of connectivity 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 connectivity 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 connectivity 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.

Why rankwidth?

- Rankwidth is more general parameter than treewidth:
  - $\mathtt{rw}(G) \leq \mathtt{tw}(G) + 1$

Why rankwidth?

- Rankwidth is more general parameter than treewidth:
  - $\mathtt{rw}(G) \leq \mathtt{tw}(G) + 1$
  - $\mathtt{tw}(G) \geq m/n$, but rankwidth can be bounded for dense graph classes:
  - cliques, cographs, distance hereditary graphs, k-leaf-powers...

Why rankwidth?

- Rankwidth is more general parameter than treewidth:
  - $\mathtt{rw}(G) \leq \mathtt{tw}(G) + 1$
  - $\mathtt{tw}(G) \geq m/n$, but rankwidth can be bounded for dense graph classes:
  - cliques, cographs, distance hereditary graphs, k-leaf-powers...

- Rankwidth is approximately equivalent to cliquewidth, only known way to approximate cliquewidth is via rankwidth

Why rankwidth?

- Rankwidth is more general parameter than treewidth:
  - $\texttt{rw}(G) \leq \texttt{tw}(G) + 1$
  - $\texttt{tw}(G) \geq m/n$, but rankwidth can be bounded for dense graph classes:
  - cliques, cographs, distance hereditary graphs, k-leaf-powers...
- Rankwidth is approximately equivalent to cliquewidth, only known way to approximate cliquewidth is via rankwidth

"Courcelle's theorem" for cliquewidth/rankwidth

[Courcelle, Makowsky, and Rotics, 2000], [Oum and Seymour 2006]

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

Why rankwidth?

- Rankwidth is more general parameter than treewidth:
  - $\mathtt{rw}(G) \leq \mathtt{tw}(G) + 1$
  - $\mathtt{tw}(G) \geq m/n$, but rankwidth can be bounded for dense graph classes:
  - cliques, cographs, distance hereditary graphs, k-leaf-powers...
- Rankwidth is approximately equivalent to cliquewidth, only known way to approximate cliquewidth is via rankwidth

"Courcelle's theorem" for cliquewidth/rankwidth

[Courcelle, Makowsky, and Rotics, 2000], [Oum and Seymour 2006]

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

- Previous best rankwidth approximation algorithm $f(k)n^3$ time [Oum, 2008]

Why rankwidth?

- Rankwidth is more general parameter than treewidth:
  - $\texttt{rw}(G) \leq \texttt{tw}(G) + 1$
  - $\texttt{tw}(G) \geq m/n$, but rankwidth can be bounded for dense graph classes:
  - cliques, cographs, distance hereditary graphs, k-leaf-powers...
- Rankwidth is approximately equivalent to cliquewidth, only known way to approximate cliquewidth is via rankwidth

"Courcelle's theorem" for cliquewidth/rankwidth

[Courcelle, Makowsky, and Rotics, 2000], [Oum and Seymour 2006]

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

- Previous best rankwidth approximation algorithm $f(k)n^3$ time [Oum, 2008]
- In this work, improvement to $f(k)n^2$ time.

Why rankwidth?

- Rankwidth is more general parameter than treewidth:
  - $\mathtt{rw}(G) \leq \mathtt{tw}(G) + 1$
  - $\mathtt{tw}(G) \geq m/n$, but rankwidth can be bounded for dense graph classes:
  - cliques, cographs, distance hereditary graphs, k-leaf-powers...
- Rankwidth is approximately equivalent to cliquewidth, only known way to approximate cliquewidth is via rankwidth

"Courcelle's theorem" for cliquewidth/rankwidth

[Courcelle, Makowsky, and Rotics, 2000], [Oum and Seymour 2006]

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

- Previous best rankwidth approximation algorithm $f(k)n^3$ time [Oum, 2008]
- In this work, improvement to $f(k)n^2$ time.

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

Techniques for rankwidth

Iterative compression

Well-known technique: Iterative compression

Iterative compression

Well-known technique: Iterative compression

- Insert vertices one-by-one, maintaining an "augmented" rank decomposition of width $\leq 2\mathtt{rw}(G)$

Iterative compression

Well-known technique: Iterative compression

- Insert vertices one-by-one, maintaining an “augmented” rank decomposition of width $\leq 2\mathtt{rw}(G)$
- Insert one vertex in $2^{\mathcal{O}(\mathtt{rw}(G))}n$ time

Iterative compression

Well-known technique: Iterative compression

- Insert vertices one-by-one, maintaining an “augmented” rank decomposition of width $\leq 2\texttt{rw}(G)$
- Insert one vertex in $2^{\mathcal{O}(\texttt{rw}(G))}n$ time
- Improve width to $\leq 2\texttt{rw}(G)$ in $2^{2^{\mathcal{O}(\texttt{rw}(G))}}n$ time

Iterative compression

Well-known technique: Iterative compression

- Insert vertices one-by-one, maintaining an "augmented" rank decomposition of width $\leq 2\mathtt{rw}(G)$
- Insert one vertex in $2^{\mathcal{O}(\mathtt{rw}(G))}n$ time
- Improve width to $\leq 2\mathtt{rw}(G)$ in $2^{2^{\mathcal{O}(\mathtt{rw}(G))}}n$ time
- Repeat n times $\rightarrow 2^{2^{\mathcal{O}(\mathtt{rw}(G))}}n^2$ time algorithm

Iterative compression

Well-known technique: Iterative compression

- Insert vertices one-by-one, maintaining an "augmented" rank decomposition of width $\leq 2\texttt{rw}(G)$
- Insert one vertex in $2^{\mathcal{O}(\texttt{rw}(G))}n$ time
- Improve width to $\leq 2\texttt{rw}(G)$ in $2^{2^{\mathcal{O}(\texttt{rw}(G))}}n$ time
- Repeat n times $\rightarrow 2^{2^{\mathcal{O}(\texttt{rw}(G))}}n^2$ time algorithm

Novel compression algorithm

Input: Augmented rank decomposition of G of width k
Output: Augmented rank decomposition of G of width $\leq k-1$ or conclusion $k \leq 2\mathtt{rw}(G)$
Time complexity: $2^{2^{\mathcal{O}(k)}} n$

Novel compression algorithm

Input: Augmented rank decomposition of G of width k
Output: Augmented rank decomposition of G of width $\leq k-1$ or conclusion $k \leq 2\mathtt{rw}(G)$
Time complexity: $2^{2^{\mathcal{O}(k)}} n$

(Assumes that the graph G is already stored as adjacency matrix)

Novel compression algorithm

Input: Augmented rank decomposition of G of width k
Output: Augmented rank decomposition of G of width $\leq k-1$ or conclusion $k \leq 2\mathtt{rw}(G)$
Time complexity: $2^{2^{\mathcal{O}(k)}} n$

(Assumes that the graph G is already stored as adjacency matrix)

Our algorithm

- **Is not** based on a Robertson-Seymour type idea of building the decomposition top-down

Novel compression algorithm

Input: Augmented rank decomposition of G of width k
Output: Augmented rank decomposition of G of width $\leq k-1$ or conclusion $k \leq 2\texttt{rw}(G)$
Time complexity: $2^{2^{\mathcal{O}(k)}} n$

(Assumes that the graph G is already stored as adjacency matrix)

Our algorithm

- **Is not** based on a Robertson-Seymour type idea of building the decomposition top-down
- Instead, iteratively improves the given rank decomposition by applying **refinement operations**

Novel compression algorithm

Input: Augmented rank decomposition of G of width k
Output: Augmented rank decomposition of G of width $\leq k-1$ or conclusion $k \leq 2\texttt{rw}(G)$
Time complexity: $2^{2^{\mathcal{O}(k)}} n$

(Assumes that the graph G is already stored as adjacency matrix)

Our algorithm

- **Is not** based on a Robertson-Seymour type idea of building the decomposition top-down
- Instead, iteratively improves the given rank decomposition by applying **refinement operations**
- Combinatorial framework: For any connectivity function f, a branch decomposition of width $> 2\texttt{bw}(f)$ can be improved by refinement operation

Novel compression algorithm

Input: Augmented rank decomposition of G of width k
Output: Augmented rank decomposition of G of width $\leq k-1$ or conclusion $k \leq 2\mathtt{rw}(G)$
Time complexity: $2^{2^{\mathcal{O}(k)}} n$

(Assumes that the graph G is already stored as adjacency matrix)

Our algorithm

- **Is not** based on a Robertson-Seymour type idea of building the decomposition top-down
- Instead, iteratively improves the given rank decomposition by applying **refinement operations**
- Combinatorial framework: For any connectivity function f, a branch decomposition of width $> 2\mathtt{bw}(f)$ can be improved by refinement operation
- Algorithmic framework:
  - Direct computation of refinements by dynamic programming $\rightarrow 2^{2^{\mathcal{O}(k)}} n^2$ time
  - Amortization techniques exploiting combinatorial results $\rightarrow 2^{2^{\mathcal{O}(k)}} n$ time

Our framework

Our Framework

- Setting:
  - Let $f : 2^V \to \mathbb{Z}_{\geq 0}$ be a connectivity function
  - We have a branch decomposition T of f of width k
  - We want to either improve T or conclude $k \leq 2\texttt{bw}(f)$

Our Framework

- Setting:
  - Let $f : 2^V \to \mathbb{Z}_{\geq 0}$ be a connectivity function
  - We have a branch decomposition T of f of width k
  - We want to either improve T or conclude $k \leq 2\texttt{bw}(f)$

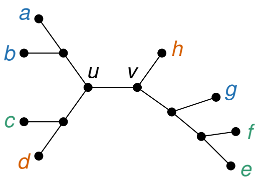
- Our structural result:
  - An edge uv of the decomposition is *heavy* if $f(uv) = k$
  - If $k > 2\texttt{bw}(f)$, then a **refinement operation** can be applied, which decreases the number of heavy edges and does not increase the width

Refinement operation

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

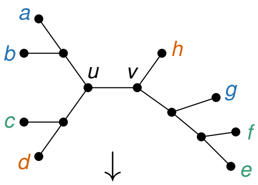
Refinement operation

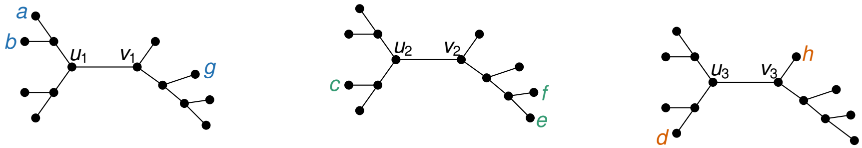
Specified by 4-tuple (r, C_1, C_2, C_3), where $r \in E(T)$ and (C_1, C_2, C_3) tripartition of V
Example with $(r, C_1, C_2, C_3) = (uv, \{a, b, g\}, \{c, e, f\}, \{d, h\})$



Refinement operation

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

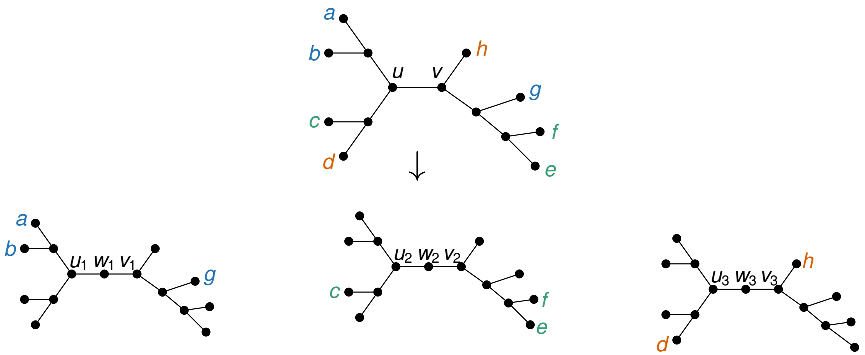




Refinement operation

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

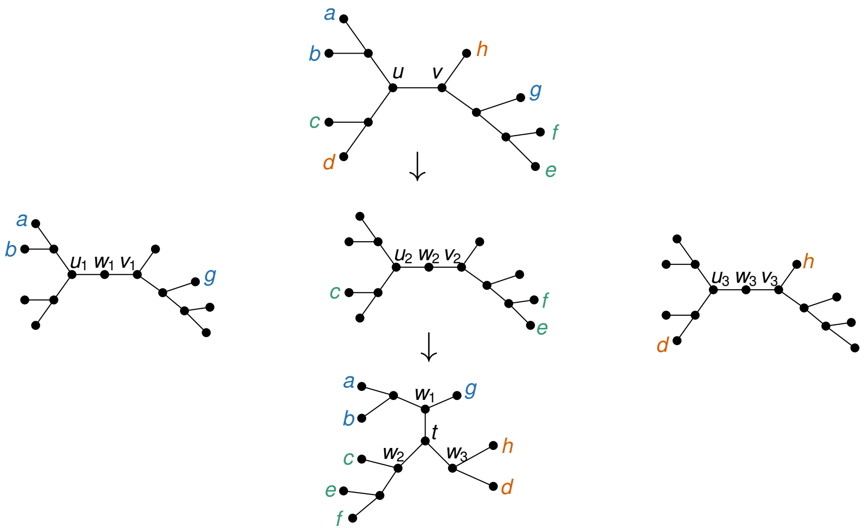
Example with $(r, C_1, C_2, C_3) = (uv, \{a, b, g\}, \{c, e, f\}, \{d, h\})$



Refinement operation

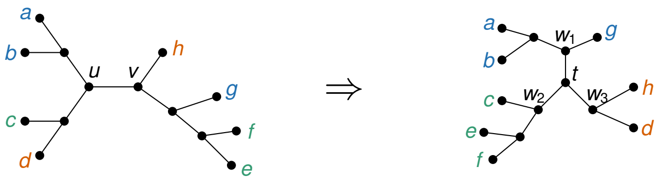
Specified by 4-tuple (r, C_1, C_2, C_3), where $r \in E(T)$ and (C_1, C_2, C_3) tripartition of V

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



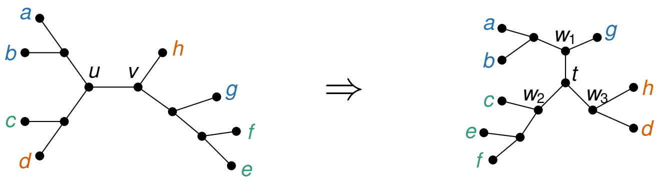
Observations on Refinement

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



Observations on Refinement

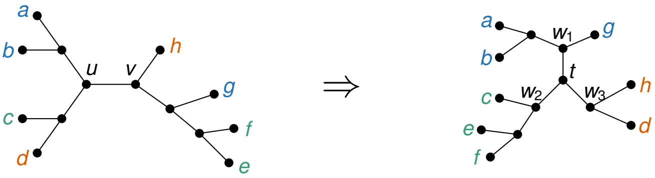
Example with $(r, C_1, C_2, C_3) = (uv, \{a, b, g\}, \{c, e, f\}, \{d, h\})$



- Observation 1: For each i, there will be an edge $w_i t$ corresponding to $(C_i, \overline{C_i})$

Observations on Refinement

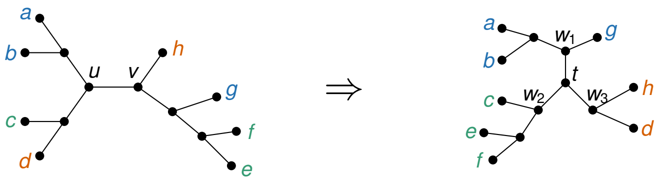
Example with $(r, C_1, C_2, C_3) = (uv, \{a, b, g\}, \{c, e, f\}, \{d, h\})$



- Observation 1: For each i, there will be an edge $w_i t$ corresponding to $(C_i, \overline{C_i})$
  - (Except when C_i is empty)

Observations on Refinement

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

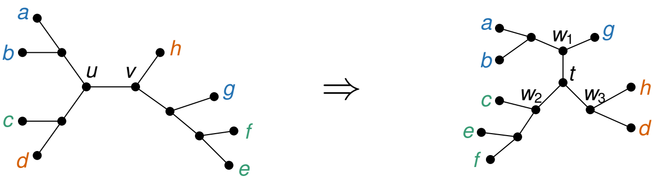


- Observation 1: For each i, there will be an edge $w_i t$ corresponding to $(C_i, \overline{C_i})$
  - (Except when C_i is empty)

- Let $(W, \overline{W}) = (\{a, b, c, d\}, \{e, f, g, h\})$ be the cut of uv

Observations on Refinement

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



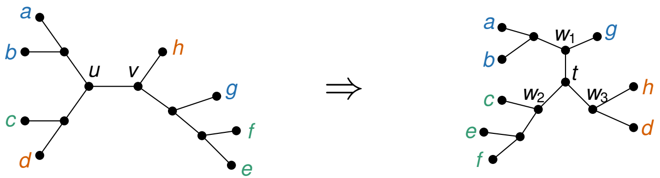
- Observation 1: For each i, there will be an edge $w_i t$ corresponding to $(C_i, \overline{C_i})$
  - (Except when C_i is empty)

- Let $(W, \overline{W}) = (\{a, b, c, d\}, \{e, f, g, h\})$ be the cut of uv

- Observation 2: For each i, there will be edges corresponding to $(C_i \cap W, \overline{C_i \cap W})$ and $(C_i \cap \overline{W}, \overline{C_i \cap \overline{W}})$

Local Improvement

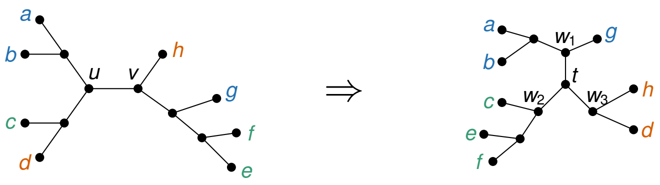
Example with $(r, C_1, C_2, C_3) = (uv, \{a, b, g\}, \{c, e, f\}, \{d, h\})$



- Let $(W, \overline{W}) = (\{a, b, c, d\}, \{e, f, g, h\})$ be the cut of uv
- Combination of Observation 1 and 2:
  - The widths of edges "near the center" will be $f(C_i)$, $f(C_i \cap W)$, and $f(C_i \cap \overline{W})$ for each i

Local Improvement

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



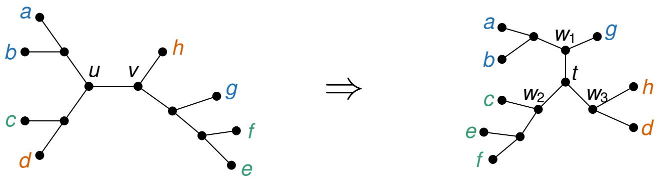
- Let $(W, \overline{W}) = (\{a, b, c, d\}, \{e, f, g, h\})$ be the cut of uv
- Combination of Observation 1 and 2:
  - The widths of edges "near the center" will be $f(C_i)$, $f(C_i \cap W)$, and $f(C_i \cap \overline{W})$ for each i

Theorem

For any set $W \subseteq V$ with $f(W) > 2\mathtt{bw}(f)$ there exists tripartition (C_1, C_2, C_3) of V so that for each i it holds that $f(C_i) < f(W)/2$, $f(C_i \cap W) < f(W)$, and $f(C_i \cap \overline{W}) < f(W)$.

Local Improvement

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



- Let $(W, \overline{W}) = (\{a, b, c, d\}, \{e, f, g, h\})$ be the cut of uv
- Combination of Observation 1 and 2:
  - The widths of edges "near the center" will be $f(C_i)$, $f(C_i \cap W)$, and $f(C_i \cap \overline{W})$ for each i

Theorem

For any set $W \subseteq V$ with $f(W) > 2\mathtt{bw}(f)$ there exists tripartition (C_1, C_2, C_3) of V so that for each i it holds that $f(C_i) < f(W)/2$, $f(C_i \cap W) < f(W)$, and $f(C_i \cap \overline{W}) < f(W)$.

$\Longrightarrow$ If $f(uv) > 2\mathtt{bw}(f)$, there exists refinement with uv that locally improves T

Global Improvement

- Let $uv \in E(T)$, $(W, \overline{W})$ the cut of uv, and $f(uv) = k$

Global Improvement

- Let $uv \in E(T)$, $(W, \overline{W})$ the cut of uv, and $f(uv) = k$
- W-improvement is any tripartition of (C_1, C_2, C_3) of V with
  1. $f(C_i) < f(W)/2$
  2. $f(C_i \cap W) < f(W)$
  3. $f(C_i \cap \overline{W}) < f(W)$

Global Improvement

- Let $uv \in E(T)$, $(W, \overline{W})$ the cut of uv, and $f(uv) = k$
- W-improvement is any tripartition of (C_1, C_2, C_3) of V with
  1. $f(C_i) < f(W)/2$
  2. $f(C_i \cap W) < f(W)$
  3. $f(C_i \cap \overline{W}) < f(W)$
- Recall: If $f(uv) > 2\mathtt{bw}(f)$, then W-improvement exists

Global Improvement

- Let $uv \in E(T)$, $(W, \overline{W})$ the cut of uv, and $f(uv) = k$
- W-improvement is any tripartition of (C_1, C_2, C_3) of V with
  1. $f(C_i) < f(W)/2$
  2. $f(C_i \cap W) < f(W)$
  3. $f(C_i \cap \overline{W}) < f(W)$
- Recall: If $f(uv) > 2\texttt{bw}(f)$, then W-improvement exists

Theorem

If there exists a W-improvement, then there exists a W-improvement (C_1, C_2, C_3) so that refinement with (uv, C_1, C_2, C_3) does not increase width and decreases number of edges of width k.

Global Improvement

- Let $uv \in E(T)$, $(W, \overline{W})$ the cut of uv, and $f(uv) = k$
- W-improvement is any tripartition of (C_1, C_2, C_3) of V with
  1. $f(C_i) < f(W)/2$
  2. $f(C_i \cap W) < f(W)$
  3. $f(C_i \cap \overline{W}) < f(W)$
- Recall: If $f(uv) > 2\mathtt{bw}(f)$, then W-improvement exists

Theorem

If there exists a W-improvement, then there exists a W-improvement (C_1, C_2, C_3) so that refinement with (uv, C_1, C_2, C_3) does not increase width and decreases number of edges of width k.

- Such W-improvement can be found by selecting a W-improvement that optimizes some explicit criteria among all W-improvements

Global Improvement

- Let $uv \in E(T)$, $(W, \overline{W})$ the cut of uv, and $f(uv) = k$
- W-improvement is any tripartition of (C_1, C_2, C_3) of V with
  1. $f(C_i) < f(W)/2$
  2. $f(C_i \cap W) < f(W)$
  3. $f(C_i \cap \overline{W}) < f(W)$
- Recall: If $f(uv) > 2\mathtt{bw}(f)$, then W-improvement exists

Theorem

If there exists a W-improvement, then there exists a W-improvement (C_1, C_2, C_3) so that refinement with (uv, C_1, C_2, C_3) does not increase width and decreases number of edges of width k.

- Such W-improvement can be found by selecting a W-improvement that optimizes some explicit criteria among all W-improvements
  - i.e., primarily minimize $\max_i f(C_i)$, secondarily minimize number of non-empty C_i, tertiarily...

First Algorithm

- Now, we have a following meta-algorithm for connectivity functions that allow efficient dynamic programming

1. Let T have width k, select edge uv with $f(uv) = k$
2. Root T at uv
3. Use dynamic programming on T to find a W-improvement optimizing the criteria or conclude $k \leq 2\mathtt{bw}(f)$ if no W-improvement found
4. Refine T using the W-improvement
5. Repeat until the width of T decreases (at most n iterations)

First Algorithm

- Now, we have a following meta-algorithm for connectivity functions that allow efficient dynamic programming

1. Let T have width k, select edge uv with $f(uv) = k$
2. Root T at uv
3. Use dynamic programming on T to find a W-improvement optimizing the criteria or conclude $k \leq 2\mathtt{bw}(f)$ if no W-improvement found
4. Refine T using the W-improvement
5. Repeat until the width of T decreases (at most n iterations)

$\Rightarrow$ Time complexity $t(k) \cdot n^2$ to decrease the width by one, where $t(k)$ is the time complexity of dynamic programming per node

First Algorithm

- Now, we have a following meta-algorithm for connectivity functions that allow efficient dynamic programming

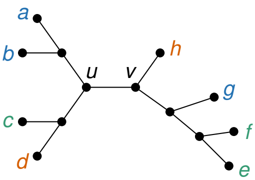
1. Let T have width k, select edge uv with $f(uv) = k$
2. Root T at uv
3. Use dynamic programming on T to find a W-improvement optimizing the criteria or conclude $k \leq 2\mathtt{bw}(f)$ if no W-improvement found
4. Refine T using the W-improvement
5. Repeat until the width of T decreases (at most n iterations)

$\Longrightarrow$ Time complexity $t(k) \cdot n^2$ to decrease the width by one, where $t(k)$ is the time complexity of dynamic programming per node

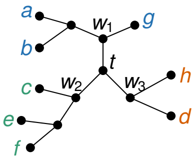
- Too slow! Applications require $t(k) \cdot n$

Amortization technique

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



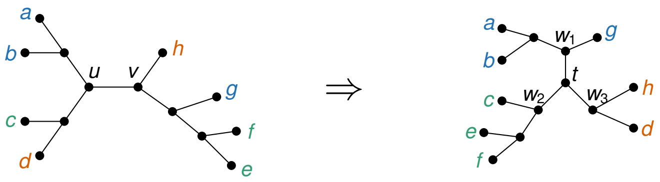
$\Rightarrow$



- Consider T rooted at $r = uv$

Amortization technique

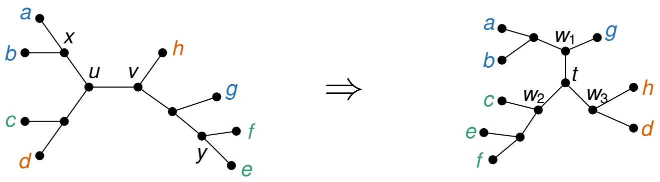
Example with $(r, C_1, C_2, C_3) = (uv, \{a, b, g\}, \{c, e, f\}, \{d, h\})$



- Consider T rooted at $r = uv$
- For a node $x \in V(T)$, denote by $T_r[x] \subseteq V$ the leaves in the subtree below x
  - Example: $T_r[x] = \{a, b\}$ and $T_r[y] = \{e, f\}$

Amortization technique

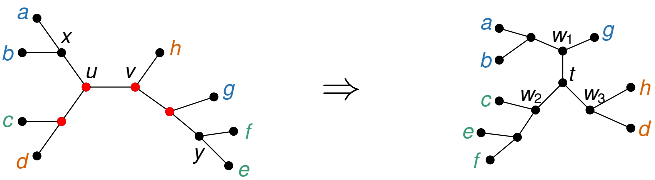
Example with $(r, C_1, C_2, C_3) = (uv, \{a, b, g\}, \{c, e, f\}, \{d, h\})$



- Consider T rooted at $r = uv$
- For a node $x \in V(T)$, denote by $T_r[x] \subseteq V$ the leaves in the subtree below x
  - Example: $T_r[x] = \{a, b\}$ and $T_r[y] = \{e, f\}$
- Observation: If $T_r[x] \subseteq C_i$, then the subtree of x appears identically in refinement

Amortization technique

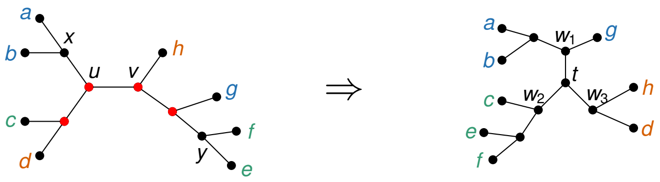
Example with $(r, C_1, C_2, C_3) = (uv, \{a, b, g\}, \{c, e, f\}, \{d, h\})$



- Consider T rooted at $r = uv$
- For a node $x \in V(T)$, denote by $T_r[x] \subseteq V$ the leaves in the subtree below x
  - Example: $T_r[x] = \{a, b\}$ and $T_r[y] = \{e, f\}$
- Observation: If $T_r[x] \subseteq C_i$, 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

Amortization technique

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



- Consider T rooted at $r = uv$

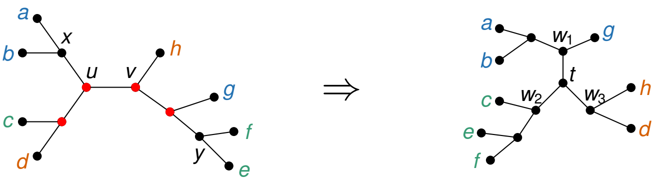
- For a node $x \in V(T)$, denote by $T_r[x] \subseteq V$ the leaves in the subtree below x
  - Example: $T_r[x] = \{a, b\}$ and $T_r[y] = \{e, f\}$

- Observation: If $T_r[x] \subseteq C_i$, 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
  - R forms a connected subtree around uv, and refinement can be implemented by removing R and inserting $|R|$ nodes in its place

Amortization technique

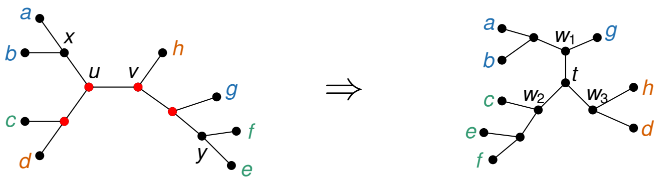
Example with $(r, C_1, C_2, C_3) = (uv, \{a, b, g\}, \{c, e, f\}, \{d, h\})$



- Consider T rooted at $r = uv$
- For a node $x \in V(T)$, denote by $T_r[x] \subseteq V$ the leaves in the subtree below x
  - Example: $T_r[x] = \{a, b\}$ and $T_r[y] = \{e, f\}$
- Observation: If $T_r[x] \subseteq C_i$, 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
  - R forms a connected subtree around uv, and refinement can be implemented by removing R and inserting $|R|$ nodes in its place
  - Over sequence of refinements, it holds that $\sum |R| \leq \mathcal{O}(3^k \cdot k \cdot n)$

Amortization technique

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



- Consider T rooted at $r = uv$
- For a node $x \in V(T)$, denote by $T_r[x] \subseteq V$ the leaves in the subtree below x
  - Example: $T_r[x] = \{a, b\}$ and $T_r[y] = \{e, f\}$
- Observation: If $T_r[x] \subseteq C_i$, 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
  - R forms a connected subtree around uv, and refinement can be implemented by removing R and inserting $|R|$ nodes in its place
  - Over sequence of refinements, it holds that $\sum |R| \leq \mathcal{O}(3^k \cdot k \cdot n)$

  $\Rightarrow$ Time complexity $t(k)2^{\mathcal{O}(k)}n$ for connectivity functions with $t(k)$ time dynamic programming per node

Conclusion

- Framework for 2-approximating branchwidth of connectivity functions

Conclusion

- Framework for 2-approximating branchwidth of connectivity functions
- Main application: $2^{2^{\mathcal{O}(k)}} n^2$ time 2-approximation algorithm for rankwidth
  - Solves the open problem of breaking the n^3 barrier for rankwidth

Conclusion

- Framework for 2-approximating branchwidth of connectivity functions
- Main application: $2^{2^{\mathcal{O}(k)}} n^2$ time 2-approximation algorithm for rankwidth
  - Solves the open problem of breaking the n^3 barrier for rankwidth
- Open problem: Is there a $f(k)(n+m)^{1.9}$ time $g(k)$-approximation algorithm for rankwidth?