Fast FPT-Approximation of Branchwidth

Fedor V. Fomin, Tuukka Korhonen

Department of Informatics, University of Bergen, Norway

IBS Virtual Discrete Math Colloquium
November 25, 2021

In this work

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

In this work

- Framework for designing fast 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 fast 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 fast 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.

Plan

1. Definitions and background
2. Overview of techniques for rankwidth
3. Combinatorial part of our framework
4. Algorithmic part of our framework

Plan

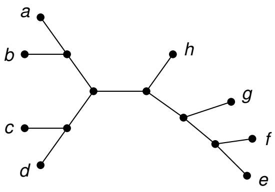
1. **Definitions and background**
2. Overview of techniques for rankwidth
3. Combinatorial part of our framework
4. Algorithmic part of our framework

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 $\overline{A} = V \setminus A$

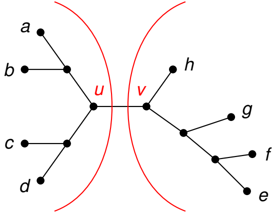
Branchwidth

- 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

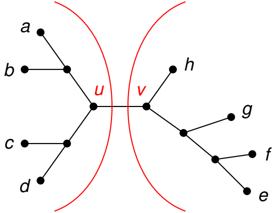
- Let V be a set and $f : 2^V \to \mathbb{Z}_{\geq 0}$ a symmetric set function.
  - Symmetric: For any $A \subseteq V$, it holds that $f(A) = f(\overline{A})$, where $\overline{A} = V \setminus A$
- Branch decomposition of f is a cubic tree whose leaves correspond to V
- Example with $V = \{a, b, c,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



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

Branchwidth

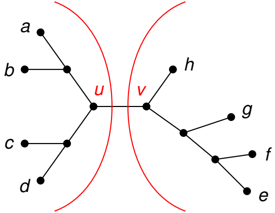
- Let V be a set and $f : 2^V \to \mathbb{Z}_{\geq 0}$ a symmetric set function.
  - Symmetric: For any $A \subseteq V$, it holds that $f(A) = f(\overline{A})$, where $\overline{A} = V \setminus A$
- Branch decomposition of f is a cubic tree whose leaves correspond to V
- Example with $V = \{a, b, c,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



- 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 edge set: $V = E(G)$, for any $A \subseteq V$, $f(A)$ is the number of vertices adjacent to edges in both A and $\overline{A}$
  - Branchwidth of 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 edge set: $V = E(G)$, for any $A \subseteq V$, $f(A)$ is the number of vertices adjacent to edges in both A and $\overline{A}$
  - Branchwidth of G
- Cut-rank: $V = V(G)$, for any $A \subseteq V$, $f(A)$ is the GF(2) rank of the $|A| \times |\overline{A}|$ matrix representing $G[A, \overline{A}]$
  - Rankwidth of G

History of rankwidth – Cliquewidth

- Cliquewidth – dense generalization of treewidth defined by [Courcelle, Engelfriet, and Rozenberg, 1993]

History of rankwidth – Cliquewidth

- Cliquewidth – dense generalization of treewidth defined by [Courcelle, Engelfriet, and Rozenberg, 1993]
- Graph classes with bounded treewidth have bounded cliquewidth

History of rankwidth – Cliquewidth

- Cliquewidth – dense generalization of treewidth defined by [Courcelle, Engelfriet, and Rozenberg, 1993]
- Graph classes with bounded treewidth have bounded cliquewidth
- But cliquewidth can be bounded also for dense graphs, for example:
  - cliques, cographs, distance hereditary graphs, k-leaf-powers...

History of rankwidth – Cliquewidth

- Cliquewidth – dense generalization of treewidth defined by [Courcelle, Engelfriet, and Rozenberg, 1993]
- Graph classes with bounded treewidth have bounded cliquewidth
- But cliquewidth can be bounded also for dense graphs, for example:
  - cliques, cographs, distance hereditary graphs, *k*-leaf-powers...

“Courcelle’s theorem” for cliquewidth

[Courcelle, Makowsky, and Rotics, 2000]



Given a graph with a decomposition witnessing cliquewidth $\leq k$, any $\mathbf{MSO}_1$-definable graph problem can be solved in $f(k)(n+m)$ time

History of rankwidth – Cliquewidth

- Cliquewidth – dense generalization of treewidth defined by [Courcelle, Engelfriet, and Rozenberg, 1993]
- Graph classes with bounded treewidth have bounded cliquewidth
- But cliquewidth can be bounded also for dense graphs, for example:
  - cliques, cographs, distance hereditary graphs, k-leaf-powers...

"Courcelle's theorem" for cliquewidth

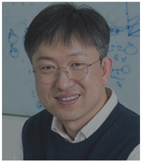
[Courcelle, Makowsky, and Rotics, 2000]



Given a graph with a decomposition witnessing cliquewidth $\leq k$, any $\mathbf{MSO}_1$-definable graph problem can be solved in $f(k)(n+m)$ time

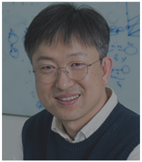
Given a graph of cliquewidth k, how to construct such decomposition?

History of rankwidth – How to approximate cliquewidth?



[Oum and Seymour, 2006]: Use rankwidth!

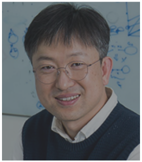
History of rankwidth – How to approximate cliquewidth?



[Oum and Seymour, 2006]: Use rankwidth!

- $\mathtt{rw}(G) \leq \mathtt{cwd}(G) \leq 2^{\mathtt{rw}(G)+1} - 1$

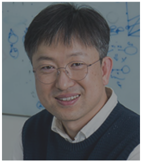
History of rankwidth – How to approximate cliquewidth?




[Oum and Seymour, 2006]: Use rankwidth!

- $\mathtt{rw}(G) \leq \mathtt{cwd}(G) \leq 2^{\mathtt{rw}(G)+1} - 1$
- $\mathcal{O}(8^k n^9 \log n)$ time 3-approximation algorithm for rankwidth

History of rankwidth – How to approximate cliquewidth?

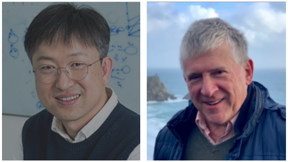
 

[Oum and Seymour, 2006]: Use rankwidth!

- $\mathtt{rw}(G) \leq \mathtt{cwd}(G) \leq 2^{\mathtt{rw}(G)+1} - 1$
- $\mathcal{O}(8^k n^9 \log n)$ time 3-approximation algorithm for rankwidth

  $\Longrightarrow$ $(2^{3k+2} - 1)$-approximation for cliquewidth

History of rankwidth – How to approximate cliquewidth?



[Oum and Seymour, 2006]: Use rankwidth!

- $\mathtt{rw}(G) \leq \mathtt{cwd}(G) \leq 2^{\mathtt{rw}(G)+1} - 1$
- $\mathcal{O}(8^k n^9 \log n)$ time 3-approximation algorithm for rankwidth
  - $\Longrightarrow$ $(2^{3k+2} - 1)$-approximation for cliquewidth
  - $\Longrightarrow$ "Courcelle's theorem" for cliquewidth with time complexity $f(k)n^9 \log n$

History of rankwidth

Reference	APX	TIME	Remarks
Oum & Seymour, 2006	$3k+1$	$8^k n^9 \log n$	Works for any connectivity function
Oum, 2008	$3k+1$	$8^k n^4$	
Oum, 2008	$3k-1$	$f(k)n^3$	Uses MSO
Courcelle & Oum, 2007	exact	$f(k)n^3$	Does not provide decomposition
Hlinený & Oum, 2008	exact	$f(k)n^3$	Uses forbidden minors
Jeong, Kim & Oum, 2021	exact	$2^{2^{O(k^2)}} n^3$	
This work	2k	$2^{2^{\mathcal{O}(k)}} n^2$	

- After the algorithm of Oum and Seymour, there has been several improvements

History of rankwidth

Reference	APX	TIME	Remarks
Oum & Seymour, 2006	$3k+1$	$8^k n^9 \log n$	Works for any connectivity function
Oum, 2008	$3k+1$	$8^k n^4$	
Oum, 2008	$3k-1$	$f(k)n^3$	Uses MSO
Courcelle & Oum, 2007	exact	$f(k)n^3$	Does not provide decomposition
Hlinený & Oum, 2008	exact	$f(k)n^3$	Uses forbidden minors
Jeong, Kim & Oum, 2021	exact	$2^{2^{O(k^2)}} n^3$	
This work	2k	$2^{2^{\mathcal{O}(k)}} n^2$	

- After the algorithm of Oum and Seymour, there has been several improvements
- In this work, improvement on the dependency on n from n^3 to n^2

History of rankwidth

Reference	APX	TIME	Remarks
Oum & Seymour, 2006	$3k+1$	$8^k n^9 \log n$	Works for any connectivity function
Oum, 2008	$3k+1$	$8^k n^4$	
Oum, 2008	$3k-1$	$f(k)n^3$	Uses MSO
Courcelle & Oum, 2007	exact	$f(k)n^3$	Does not provide decomposition
Hlinený & Oum, 2008	exact	$f(k)n^3$	Uses forbidden minors
Jeong, Kim & Oum, 2021	exact	$2^{2^{O(k^2)}} n^3$	
This work	2k	$2^{2^{\mathcal{O}(k)}} n^2$	

- After the algorithm of Oum and Seymour, there has been several improvements
- In this work, improvement on the dependency on n from n^3 to n^2

$\Longrightarrow$ "Courcelle's theorem" for cliquewidth with time complexity $f(k)n^2$

Plan

1. Definitions and background
2. **Overview of techniques for rankwidth**
3. Combinatorial part of our framework
4. Algorithmic part of our framework

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 $\leq 2\texttt{rw}(G)$

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 $k \leq 2\texttt{rw}(G)$
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 $k \leq 2\mathtt{rw}(G)$
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**

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

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

Plan

1. Definitions and background
2. Overview of techniques for rankwidth
3. **Combinatorial part of our framework**
4. Algorithmic part of our framework

General idea

- 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 $k \leq 2\mathtt{bw}(f)$

General idea

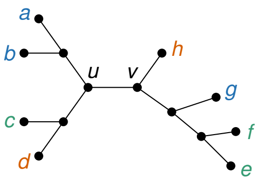
- Setting:
  - Let $f : 2^V \to \mathbb{Z}_{\geq 0}$ be a connectivity function
  - We have a branch decomposition T of f of width k
  - We want to either improve T or conclude $k \leq 2\texttt{bw}(f)$
- Strategy:
  - Let $h(T)$ be the number of edges of T of width $\geq k$ (heavy edges)
  - Either decrease $h(T)$ by using a **refinement operation**, or conclude that $k \leq 2\texttt{bw}(f)$

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

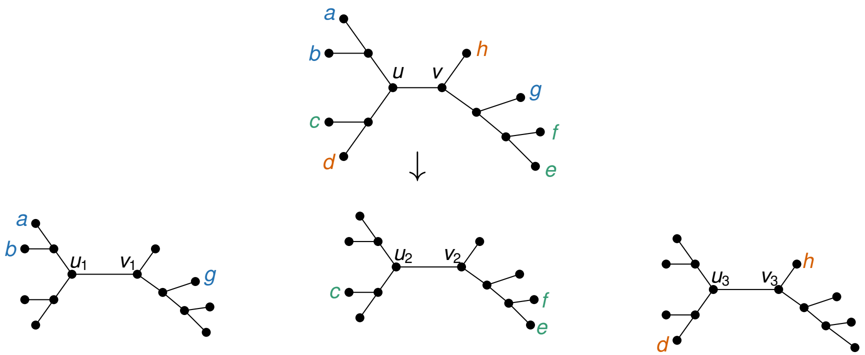
Refinement operation

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



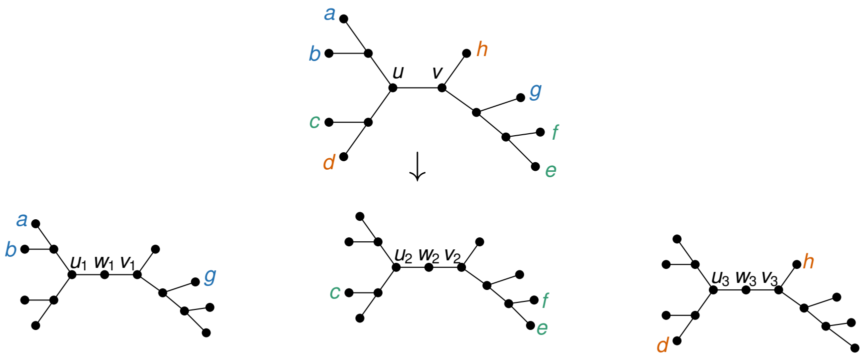
Refinement operation

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



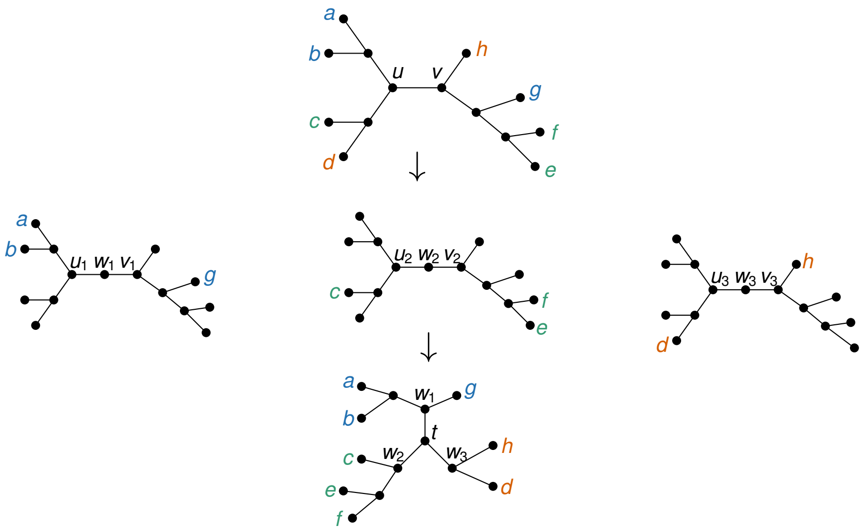
Refinement operation

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



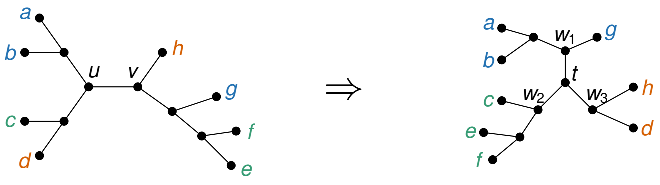
Refinement operation

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



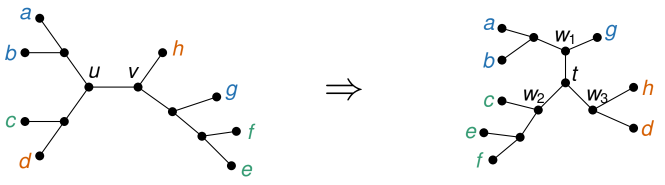
Observations on Refinement

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



Observations on Refinement

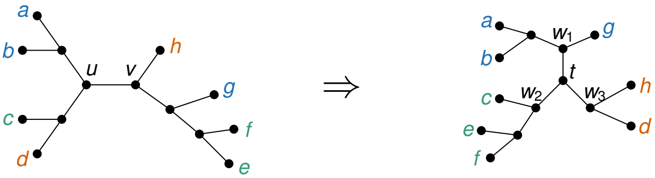
Example with $(r, C_1, C_2, C_3) = (uv, \{a, b, g\}, \{c, e, f\}, \{d, h\})$



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

Observations on Refinement

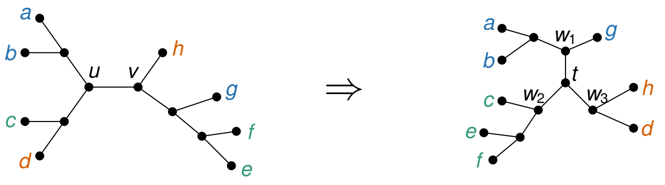
Example with $(r, C_1, C_2, C_3) = (uv, \{a, b, g\}, \{c, e, f\}, \{d, h\})$



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

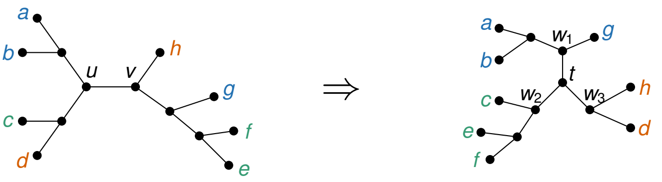


- Observation 1: For each i, there will be an edge $w_i t$ corresponding to $(C_i, \overline{C_i})$
  - (Except when C_i is empty)

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

Observations on Refinement

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



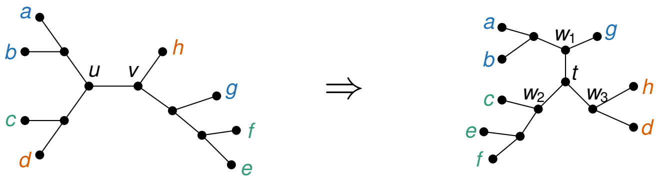
- Observation 1: For each i, there will be an edge $w_i t$ corresponding to $(C_i, \overline{C_i})$
  - (Except when C_i is empty)

- 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

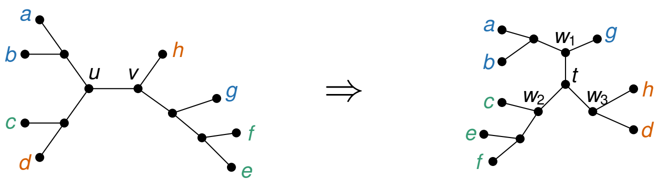
Example with $(r, C_1, C_2, C_3) = (uv, \{a, b, g\}, \{c, e, f\}, \{d, h\})$



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



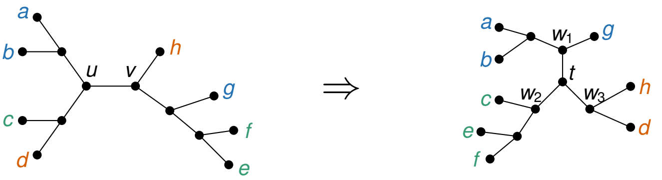
- Let $(W, \overline{W}) = (\{a, b, c, d\}, \{e, f, g, h\})$ be the cut of uv
- Combination of Observation 1 and 2:
  - The widths of edges "near the center" will be $f(C_i)$, $f(C_i \cap W)$, and $f(C_i \cap \overline{W})$ for each i

Theorem

For any set $W \subseteq V$ with $f(W) > 2\texttt{bw}(f)$ 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 $f(W) > 2\texttt{bw}(f)$ 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 $f(uv) > 2\texttt{bw}(f)$, 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 $f(uv) > 2\texttt{bw}(f)$, then W-improvement exists

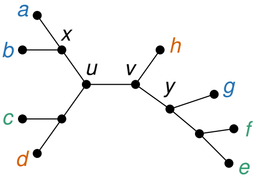
Global Improvement

- 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 $f(uv) > 2\mathtt{bw}(f)$, then W-improvement exists

Theorem

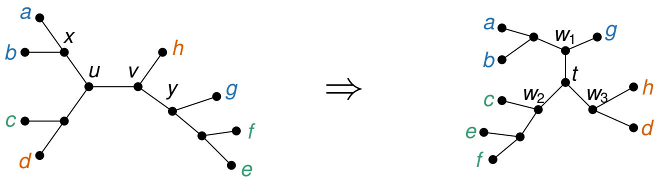
If there exists a W-improvement, then there exists a W-improvement (C_1, C_2, C_3) so that refinement with (uv, C_1, C_2, C_3) does not increase width and decreases number of edges of width k.

Global Improvement: Observation



- 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 $T_r[y] = \{e, f, g\}$

Global Improvement: Observation



- 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 $T_r[y] = \{e, f, g\}$
- Let T' be refinement of T with (r, C_1, C_2, C_3)
- Observation: Each edge of T' corresponds either to $(C_i, \overline{C_i})$ or to $(T_r[x] \cap C_i, \overline{T_r[x] \cap C_i})$ for some $x \in V(T)$

Global Improvement: Construction

- A global T-improvement is a W-improvement (C_1, C_2, C_3) that
  1. minimizes $\max(f(C_1), f(C_2), f(C_3))$ among W-improvements
  2. subject to (1), minimizes the number of non-empty C_i
  3. subject to (1,2), minimizes $f(C_1) + f(C_2) + f(C_3)$
  4. subject to (1,2,3), maximizes the number of nodes x such that $T_r[x] \subseteq C_i$ for some i

Global Improvement: Construction

- A global T-improvement is a W-improvement (C_1, C_2, C_3) that
  1. minimizes $\max(f(C_1), f(C_2), f(C_3))$ among W-improvements
  2. subject to (1), minimizes the number of non-empty C_i
  3. subject to (1,2), minimizes $f(C_1) + f(C_2) + f(C_3)$
  4. subject to (1,2,3), maximizes the number of nodes x such that $T_r[x] \subseteq C_i$ for some i

Theorem

Let (C_1, C_2, C_3) be a global T-improvement. For any $x \in V(T)$ it holds that $f(T_r[x] \cap C_i) \leq f(T_r[x])$, and moreover $f(T_r[x] \cap C_i) = f(T_r[x])$ only if $T_r[x] \subseteq C_i$.

Global Improvement: Construction

- A global T-improvement is a W-improvement (C_1, C_2, C_3) that
  1. minimizes $\max(f(C_1), f(C_2), f(C_3))$ among W-improvements
  2. subject to (1), minimizes the number of non-empty C_i
  3. subject to (1,2), minimizes $f(C_1) + f(C_2) + f(C_3)$
  4. subject to (1,2,3), maximizes the number of nodes x such that $T_r[x] \subseteq C_i$ for some i

Theorem

Let (C_1, C_2, C_3) be a global T-improvement. For any $x \in V(T)$ it holds that $f(T_r[x] \cap C_i) \leq f(T_r[x])$, and moreover $f(T_r[x] \cap C_i) = f(T_r[x])$ only if $T_r[x] \subseteq C_i$.

- Observation 1: For each edge e of T corresponding to $T_r[x]$,
  - each of the new edges corresponding to $T_r[x] \cap C_i$ has width at most $f(e)$

Global Improvement: Construction

- A global T-improvement is a W-improvement (C_1, C_2, C_3) that
  1. minimizes $\max(f(C_1), f(C_2), f(C_3))$ among W-improvements
  2. subject to (1), minimizes the number of non-empty C_i
  3. subject to (1,2), minimizes $f(C_1) + f(C_2) + f(C_3)$
  4. subject to (1,2,3), maximizes the number of nodes x such that $T_r[x] \subseteq C_i$ for some i

Theorem

Let (C_1, C_2, C_3) be a global T-improvement. For any $x \in V(T)$ it holds that $f(T_r[x] \cap C_i) \leq f(T_r[x])$, and moreover $f(T_r[x] \cap C_i) = f(T_r[x])$ only if $T_r[x] \subseteq C_i$.

- Observation 1: For each edge e of T corresponding to $T_r[x]$,
  - each of the new edges corresponding to $T_r[x] \cap C_i$ has width at most $f(e)$
  - at most one of the new edges corresponding to $T_r[x] \cap C_i$ has width $f(e)$

Global Improvement: Construction

- A global T-improvement is a W-improvement (C_1, C_2, C_3) that
  1. minimizes $\max(f(C_1), f(C_2), f(C_3))$ among W-improvements
  2. subject to (1), minimizes the number of non-empty C_i
  3. subject to (1,2), minimizes $f(C_1) + f(C_2) + f(C_3)$
  4. subject to (1,2,3), maximizes the number of nodes x such that $T_r[x] \subseteq C_i$ for some i

Theorem

Let (C_1, C_2, C_3) be a global T-improvement. For any $x \in V(T)$ it holds that $f(T_r[x] \cap C_i) \leq f(T_r[x])$, and moreover $f(T_r[x] \cap C_i) = f(T_r[x])$ only if $T_r[x] \subseteq C_i$.

- Observation 1: For each edge e of T corresponding to $T_r[x]$,
  - each of the new edges corresponding to $T_r[x] \cap C_i$ has width at most $f(e)$
  - at most one of the new edges corresponding to $T_r[x] \cap C_i$ has width $f(e)$
- Observation 2: For the edge uv, none of the new edges corresponding to it has width $f(uv)$

  $\Longrightarrow$ Strict improvement

Plan

1. Definitions and background
2. Overview of techniques for rankwidth
3. Combinatorial part of our framework
4. **Algorithmic part of our framework**

First Algorithm

- Now, we have a following generic algorithm for 2-approximating branchwidth of connectivity function

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

First Algorithm

- Now, we have a following generic algorithm for 2-approximating branchwidth of connectivity function

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

First Algorithm

- Now, we have a following generic algorithm for 2-approximating branchwidth of connectivity function

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 compute a global T-improvement or conclude $k \leq 2\mathtt{bw}(f)$

First Algorithm

- Now, we have a following generic algorithm for 2-approximating branchwidth of connectivity function

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 compute a global T-improvement or conclude $k \leq 2\texttt{bw}(f)$
4. If global T-improvement found, refine T using it

First Algorithm

- Now, we have a following generic algorithm for 2-approximating branchwidth of connectivity function

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 compute a global T-improvement or conclude $k \leq 2\texttt{bw}(f)$
4. If global T-improvement found, refine T using it
5. Repeat until the width of T decreases (at most n iterations)

First Algorithm

- Now, we have a following generic algorithm for 2-approximating branchwidth of connectivity function

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 compute a global T-improvement or conclude $k \leq 2\mathtt{bw}(f)$
4. If global T-improvement found, refine T using it
5. Repeat until the width of T decreases (at most n iterations)

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

First Algorithm

- Now, we have a following generic algorithm for 2-approximating branchwidth of connectivity function

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 compute a global T-improvement or conclude $k \leq 2\mathtt{bw}(f)$
4. If global T-improvement found, refine T using it
5. Repeat until the width of T decreases (at most n iterations)

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

- Too slow! Target is $t(k) \cdot n$

Algorithmic Framework

- Let f be a connectivity function for which there exists dynamic programming data structure with time complexity $t(k)$ per node, where k is the width of the decomposition

Algorithmic Framework

- Let f be a connectivity function for which there exists dynamic programming data structure with time complexity $t(k)$ per node, where k is the width of the decomposition

Theorem

There is an algorithm, that given a branch decomposition of width k, in time $t(k)2^{\mathcal{O}(k)}n$ either outputs a branch decomposition of width at most $k-1$, or concludes $k \leq 2\mathtt{bw}(f)$.

Algorithmic Framework

- Let f be a connectivity function for which there exists dynamic programming data structure with time complexity $t(k)$ per node, where k is the width of the decomposition

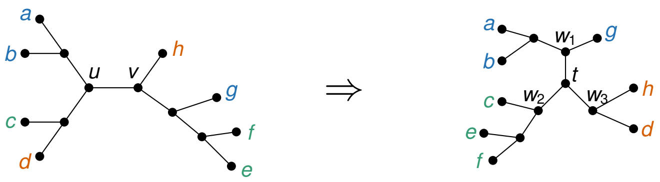
Theorem

There is an algorithm, that given a branch decomposition of width k, in time $t(k)2^{\mathcal{O}(k)}n$ either outputs a branch decomposition of width at most $k-1$, or concludes $k \leq 2\mathtt{bw}(f)$.

- For rankwidth, $t(k) = 2^{2^{\mathcal{O}(k)}}$
- For graph branchwidth $t(k) = 2^{\mathcal{O}(k)}$

Amortization technique

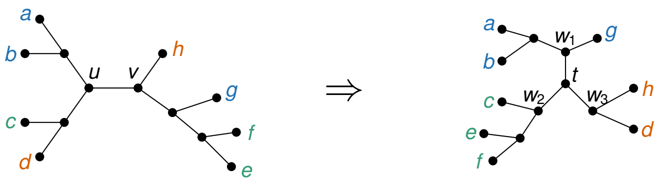
Example with $(r, C_1, C_2, C_3) = (uv, \{a, b, g\}, \{c, e, f\}, \{d, h\})$



- Consider T rooted at $r = uv$

Amortization technique

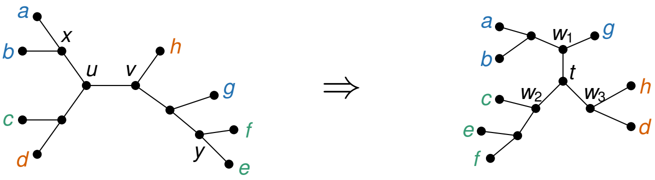
Example with $(r, C_1, C_2, C_3) = (uv, \{a, b, g\}, \{c, e, f\}, \{d, h\})$



- Consider T rooted at $r = uv$
- Observation: If $T_r[x] \subseteq C_i$, then the subtree of x appears identically in refinement

Amortization technique

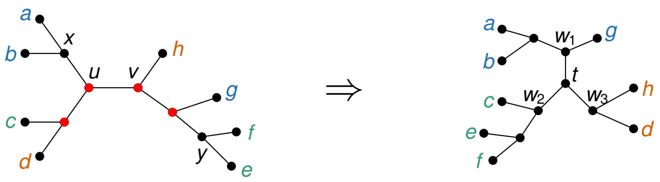
Example with $(r, C_1, C_2, C_3) = (uv, \{a, b, g\}, \{c, e, f\}, \{d, h\})$



- Consider T rooted at $r = uv$
- Observation: If $T_r[x] \subseteq C_i$, then the subtree of x appears identically in refinement
  - Example: $T_r[x] = \{a, b\} \subseteq C_1$ and $T_r[y] = \{e, f\} \subseteq C_2$

Amortization technique

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



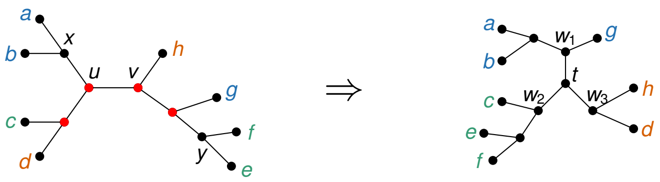
- Consider T rooted at $r = uv$

- Observation: If $T_r[x] \subseteq C_i$, then the subtree of x appears identically in refinement
  - Example: $T_r[x] = \{a, b\} \subseteq C_1$ and $T_r[y] = \{e, f\} \subseteq C_2$

- Call the nodes for which this does **not** happen the *edit set* R of the refinement

Amortization technique

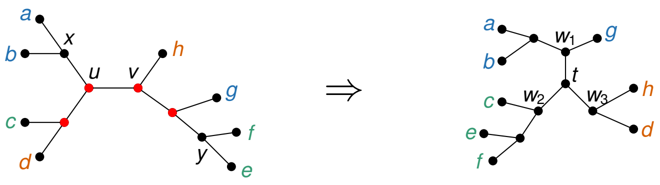
Example with $(r, C_1, C_2, C_3) = (uv, \{a, b, g\}, \{c, e, f\}, \{d, h\})$



- Consider T rooted at $r = uv$
- Observation: If $T_r[x] \subseteq C_i$, then the subtree of x appears identically in refinement
  - Example: $T_r[x] = \{a, b\} \subseteq C_1$ and $T_r[y] = \{e, f\} \subseteq C_2$
- 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 $(r, C_1, C_2, C_3) = (uv, \{a, b, g\}, \{c, e, f\}, \{d, h\})$



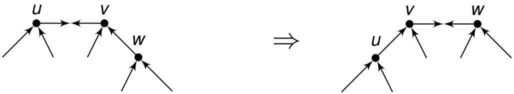
- Consider T rooted at $r = uv$
- Observation: If $T_r[x] \subseteq C_i$, then the subtree of x appears identically in refinement
  - Example: $T_r[x] = \{a, b\} \subseteq C_1$ and $T_r[y] = \{e, f\} \subseteq C_2$
- Call the nodes for which this does **not** happen the *edit set* R of the refinement
  - 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 $\sum |R| \leq \mathcal{O}(3^k \cdot k \cdot n)$

The Algorithm

- Maintain dynamic programming tables towards a root edge $r = uv$

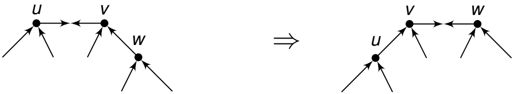
The Algorithm

- Maintain dynamic programming tables towards a root edge $r = uv$
- When changing $r = uv$ to an incident edge $r' = vw$, only the table of v needs to be recomputed



The Algorithm

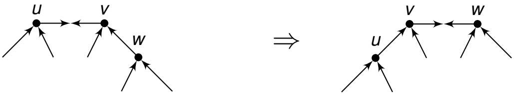
- Maintain dynamic programming tables towards a root edge $r = uv$
- When changing $r = uv$ to an incident edge $r' = vw$, only the table of v needs to be recomputed



- Use DFS to traverse the tree and refine when necessary, total amount of re-computed DP-tables will be $2^{\mathcal{O}(k)}n$ by refinement amortization

The Algorithm

- Maintain dynamic programming tables towards a root edge $r = uv$
- When changing $r = uv$ to an incident edge $r' = vw$, only the table of v needs to be recomputed



- Use DFS to traverse the tree and refine when necessary, total amount of re-computed DP-tables will be $2^{\mathcal{O}(k)}n$ by refinement amortization

$\Rightarrow$ Total complexity $t(k)2^{\mathcal{O}(k)}n$

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 $f(k)(n+m)$ time $g(k)$-approximation algorithm for rankwidth?

The end

Thank you for your attention!