Fast FPT 2-Approximation Algorithms for Width Parameters

Tuukka Korhonen

UNIVERSITY OF BERGEN

PAAW 2022

July 4, 2022

FPT-Approximation of Treewidth

FPT-Approximation of Treewidth

Robertson & Seymour, Graph Minors XIII. ('86)

- 4-approximation of treewidth in time $\mathcal{O}(8^k n^2)$


FPT-Approximation of Treewidth

Robertson & Seymour, Graph Minors XIII. ('86)


- 4-approximation of treewidth in time $\mathcal{O}(8^k n^2)$

$\Rightarrow$ Problems parameterized by treewidth can be solved in time $f(k)n^2$

FPT-Approximation of Treewidth

Robertson & Seymour, Graph Minors XIII. ('86)


- 4-approximation of treewidth in time $\mathcal{O}(8^k n^2)$

$\Rightarrow$ Problems parameterized by treewidth can be solved in time $f(k)n^2$ (many in time $2^{\mathcal{O}(k)}n^2$)

FPT-Approximation of Treewidth

Robertson & Seymour, Graph Minors XIII. ('86)


- 4-approximation of treewidth in time $\mathcal{O}(8^k n^2)$

$\Rightarrow$ Problems parameterized by treewidth can be solved in time $f(k)n^2$ (many in time $2^{\mathcal{O}(k)}n^2$)

Test of time versus exact FPT algorithms

- Bodlaender'93: $2^{\mathcal{O}(k^3)}n$ time exact treewidth

FPT-Approximation of Treewidth

Robertson & Seymour, Graph Minors XIII. ('86)


- 4-approximation of treewidth in time $\mathcal{O}(8^k n^2)$

$\Rightarrow$ Problems parameterized by treewidth can be solved in time $f(k)n^2$ (many in time $2^{\mathcal{O}(k)}n^2$)

Test of time versus exact FPT algorithms

- Bodlaender'93: $2^{\mathcal{O}(k^3)}n$ time exact treewidth
- No more progress on exact FPT-algorithms for treewidth!

FPT-Approximation of Treewidth

Robertson & Seymour, Graph Minors XIII. ('86)

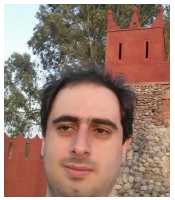

- 4-approximation of treewidth in time $\mathcal{O}(8^k n^2)$

$\Rightarrow$ Problems parameterized by treewidth can be solved in time $f(k)n^2$ (many in time $2^{\mathcal{O}(k)}n^2$)

Test of time versus exact FPT algorithms

- Bodlaender'93: $2^{\mathcal{O}(k^3)}n$ time exact treewidth
- No more progress on exact FPT-algorithms for treewidth!

Versus polynomial-time approximation


- Feige, Hajiaghayi, Lee'05: polynomial-time $\mathcal{O}(\sqrt{\log k})$-approximation

FPT-Approximation of Treewidth

Robertson & Seymour, Graph Minors XIII. ('86)

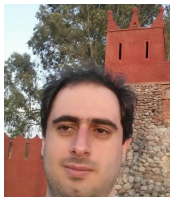

- 4-approximation of treewidth in time $\mathcal{O}(8^k n^2)$

$\Rightarrow$ Problems parameterized by treewidth can be solved in time $f(k)n^2$ (many in time $2^{\mathcal{O}(k)}n^2$)

Test of time versus exact FPT algorithms

- Bodlaender'93: $2^{\mathcal{O}(k^3)}n$ time exact treewidth
- No more progress on exact FPT-algorithms for treewidth!

Versus polynomial-time approximation


- Feige, Hajiaghayi, Lee'05: polynomial-time $\mathcal{O}(\sqrt{\log k})$-approximation



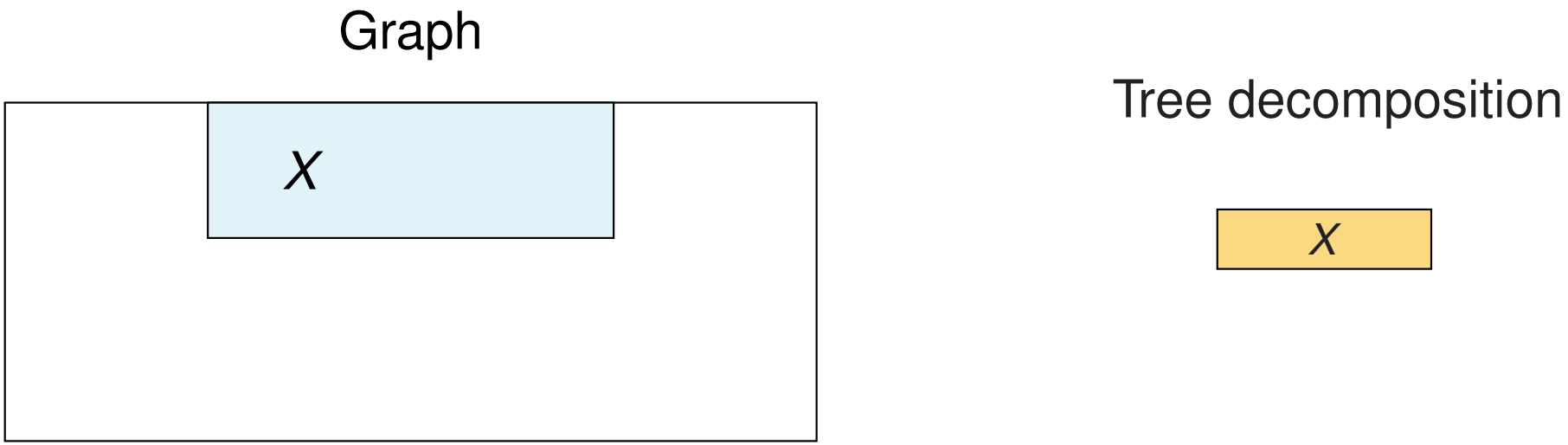

- Wu, Austrin, Pitassi, Liu'14: No polynomial-time constant-factor approximation assuming the small set expansion hypothesis

Robertson-Seymour 4-Approximation for Treewidth

Construct the decomposition “greedily” in top-down manner:

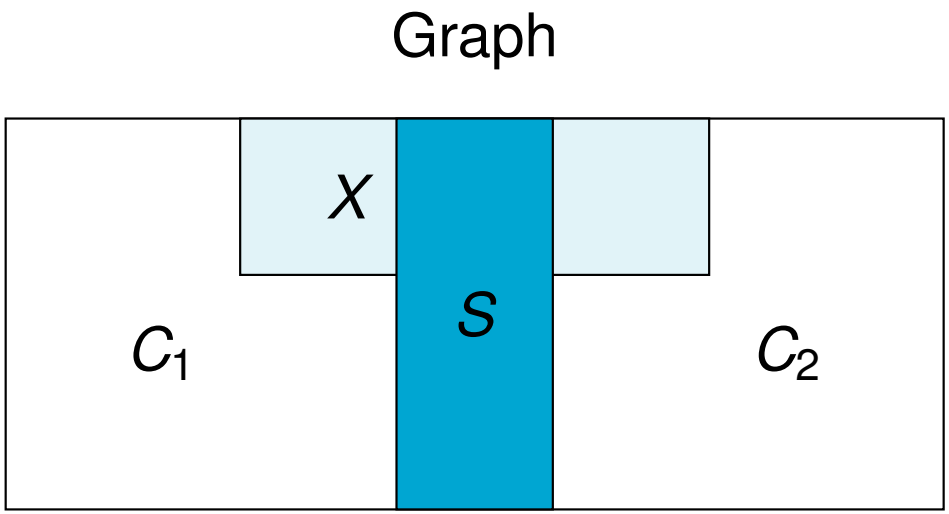
Robertson-Seymour 4-Approximation for Treewidth

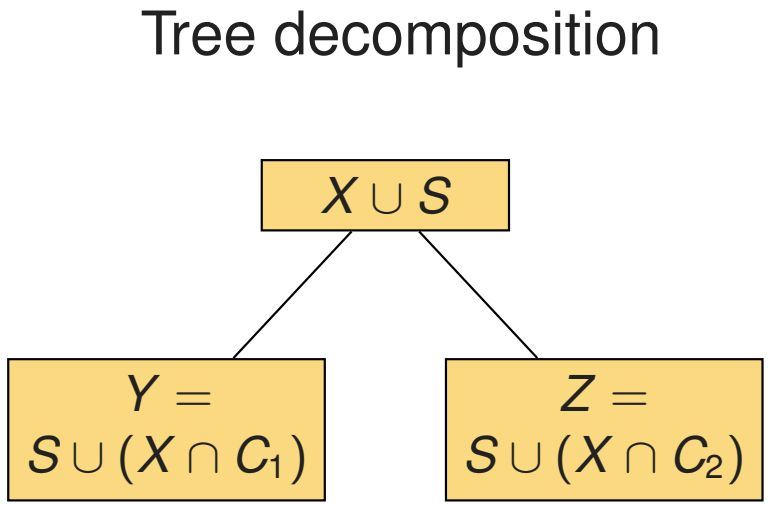
Construct the decomposition “greedily” in top-down manner:



Robertson-Seymour 4-Approximation for Treewidth

Construct the decomposition “greedily” in top-down manner:

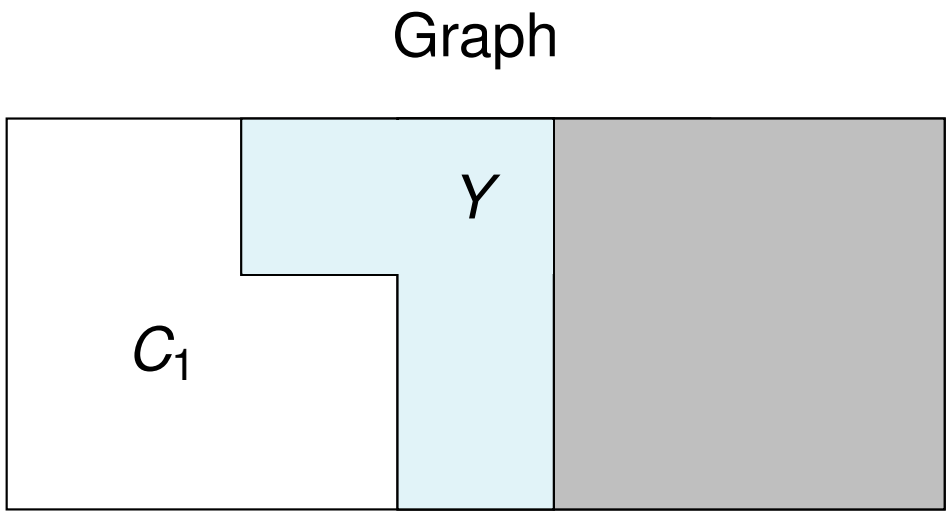


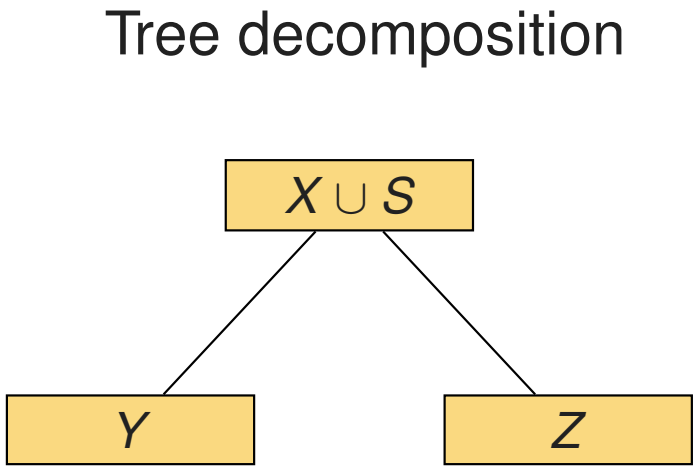


Separator S with components C_1 and C_2

Robertson-Seymour 4-Approximation for Treewidth

Construct the decomposition “greedily” in top-down manner:

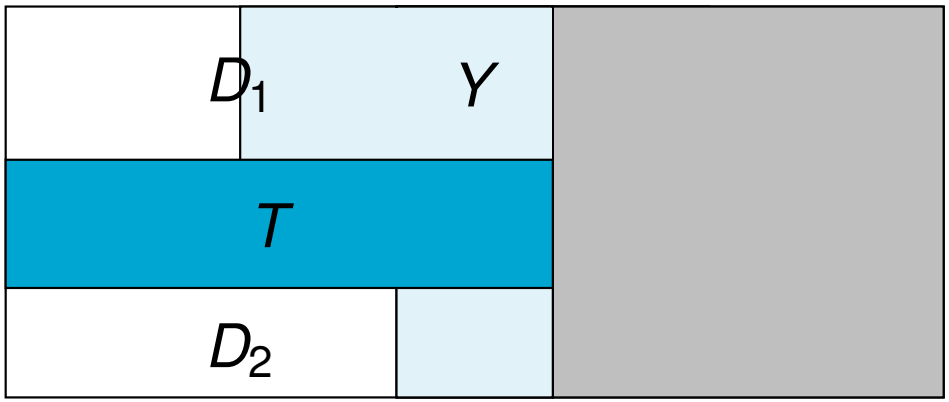




Robertson-Seymour 4-Approximation for Treewidth

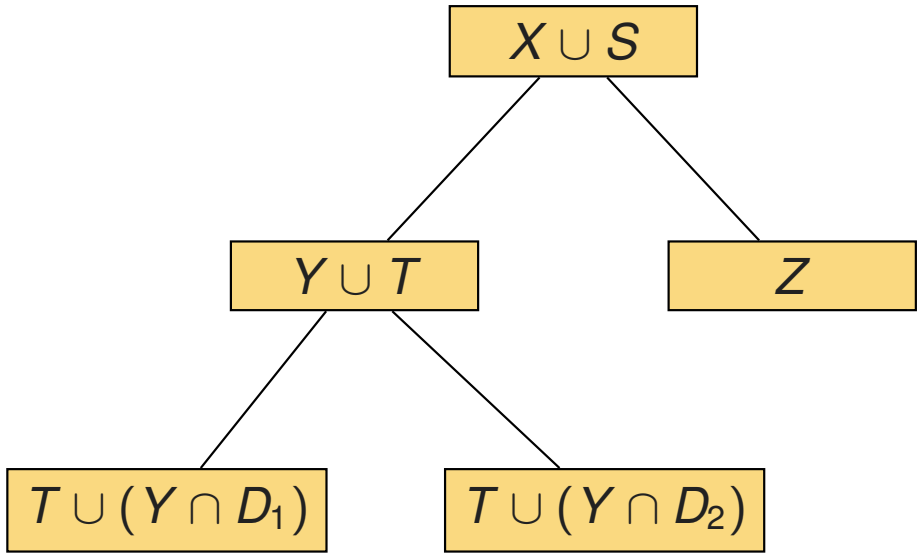
Construct the decomposition “greedily” in top-down manner:

Graph



Separator T with components D_1 and D_2

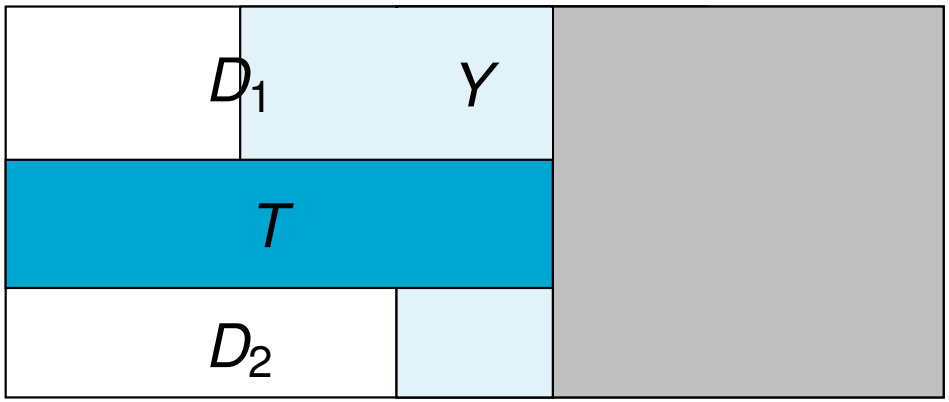
Tree decomposition



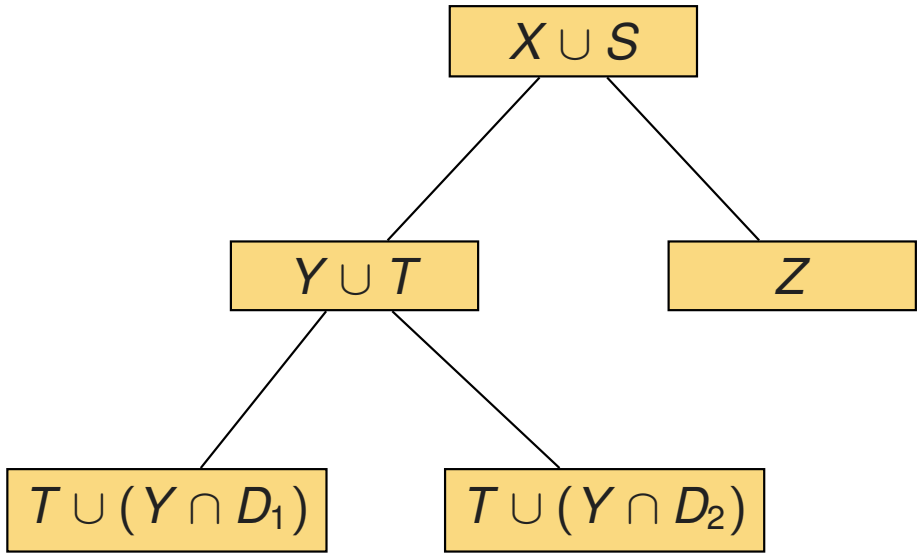
Robertson-Seymour 4-Approximation for Treewidth

Construct the decomposition “greedily” in top-down manner:

Graph



Tree decomposition



Separator T with components D_1 and D_2

Continue recursively...

Progress in FPT-Approximation of Treewidth

- Several treewidth approximation algorithms based on Robertson-Seymour

Progress in FPT-Approximation of Treewidth

- Several treewidth approximation algorithms based on Robertson-Seymour

Reed '92:

- Height of the recursion can be made $\mathcal{O}(\log n)$

Progress in FPT-Approximation of Treewidth

- Several treewidth approximation algorithms based on Robertson-Seymour

Reed '92:

- Height of the recursion can be made $\mathcal{O}(\log n)$
- $2^{\mathcal{O}(k \log k)} n \log n$ time 8-approximation

Progress in FPT-Approximation of Treewidth

- Several treewidth approximation algorithms based on Robertson-Seymour

Reed '92:

- Height of the recursion can be made $\mathcal{O}(\log n)$
- $2^{\mathcal{O}(k \log k)} n \log n$ time 8-approximation

Bodlaender, Drange, Dregi, Fomin, Lokshtanov, Pilipczuk '13:

- Implementation with several new data structures and other techniques

Progress in FPT-Approximation of Treewidth

- Several treewidth approximation algorithms based on Robertson-Seymour

Reed '92:

- Height of the recursion can be made $\mathcal{O}(\log n)$
- $2^{\mathcal{O}(k \log k)} n \log n$ time 8-approximation

Bodlaender, Drange, Dregi, Fomin, Lokshtanov, Pilipczuk '13:

- Implementation with several new data structures and other techniques
- $2^{\mathcal{O}(k)} n \log n$ time 3-approximation

Progress in FPT-Approximation of Treewidth

- Several treewidth approximation algorithms based on Robertson-Seymour

Reed '92:

- Height of the recursion can be made $\mathcal{O}(\log n)$
- $2^{\mathcal{O}(k \log k)} n \log n$ time 8-approximation

Bodlaender, Drange, Dregi, Fomin, Lokshtanov, Pilipczuk '13:

- Implementation with several new data structures and other techniques
- $2^{\mathcal{O}(k)} n \log n$ time 3-approximation
- $2^{\mathcal{O}(k)} n$ time 5-approximation

Progress in FPT-Approximation of Treewidth

- Several treewidth approximation algorithms based on Robertson-Seymour

Reed '92:

- Height of the recursion can be made $\mathcal{O}(\log n)$
- $2^{\mathcal{O}(k \log k)} n \log n$ time 8-approximation

Bodlaender, Drange, Dregi, Fomin, Lokshtanov, Pilipczuk '13:

- Implementation with several new data structures and other techniques
- $2^{\mathcal{O}(k)} n \log n$ time 3-approximation
- $2^{\mathcal{O}(k)} n$ time 5-approximation

$\Rightarrow$ $2^{\mathcal{O}(k)} n$ time algorithms parameterized by treewidth

Progress in FPT-Approximation of Treewidth

- Several treewidth approximation algorithms based on Robertson-Seymour

Reed '92:

- Height of the recursion can be made $\mathcal{O}(\log n)$
- $2^{\mathcal{O}(k \log k)} n \log n$ time 8-approximation

Bodlaender, Drange, Dregi, Fomin, Lokshtanov, Pilipczuk '13:

- Implementation with several new data structures and other techniques
- $2^{\mathcal{O}(k)} n \log n$ time 3-approximation
- $2^{\mathcal{O}(k)} n$ time 5-approximation

$\Rightarrow$ $2^{\mathcal{O}(k)} n$ time algorithms parameterized by treewidth

Also...

[Lagergren'90], [Feige, Hajiaghayi & Lee'05], [Amir'10],
[Fomin, Lokshtanov, Pilipczuk, Saurabh & Wrochna'17], [Belbasi & Fürer'21]

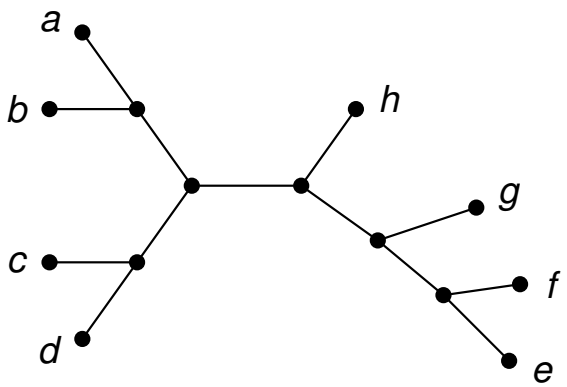
Branchwidth of Connectivity function

Branchwidth of Connectivity function

- Let V be a set and $f : 2^V \to \mathbb{Z}_{\geq 0}$ 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 $f(A \cup B) + f(A \cap B) \leq f(A) + f(B)$

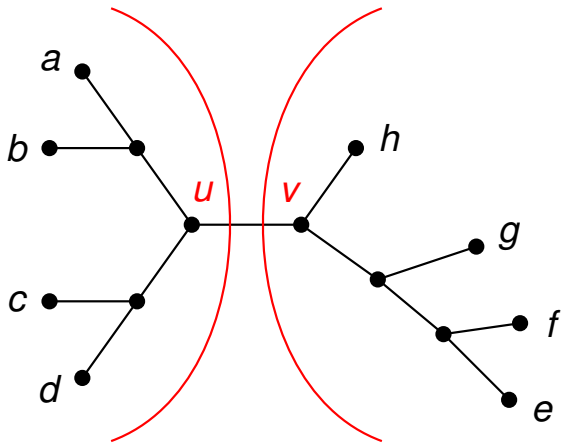
Branchwidth of Connectivity function

- Let V be a set and $f : 2^V \to \mathbb{Z}_{\geq 0}$ 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

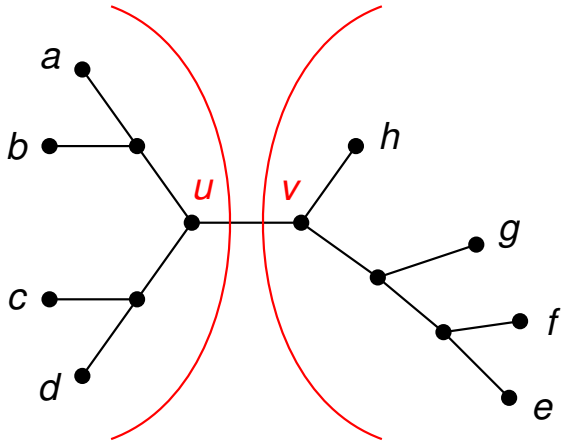
- Let V be a set and $f : 2^V \to \mathbb{Z}_{\geq 0}$ a connectivity function:
  - Symmetric: For any $A \subseteq V$, it holds that $f(A) = f(\overline{A})$, where $\overline{A} = V \setminus A$
  - Submodular: For any $A, B \subseteq V$, it holds that $f(A \cup B) + f(A \cap B) \leq f(A) + f(B)$
-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

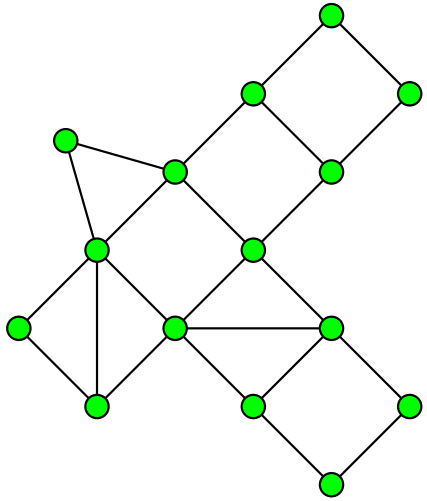
- Let V be a set and $f : 2^V \to \mathbb{Z}_{\geq 0}$ a connectivity function:
  - Symmetric: For any $A \subseteq V$, it holds that $f(A) = f(\overline{A})$, where $\overline{A} = V \setminus A$
  - Submodular: For any $A, B \subseteq V$, it holds that $f(A \cup B) + f(A \cap B) \leq f(A) + f(B)$
-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

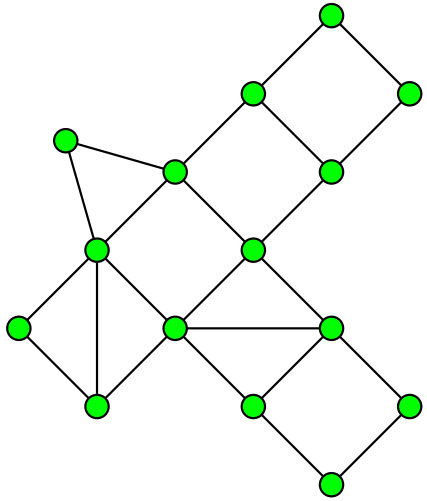
Examples of Branchwidth of Connectivity Functions

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



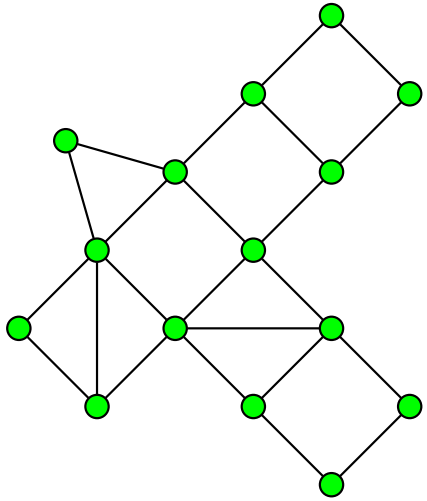
Examples of Branchwidth of Connectivity Functions

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

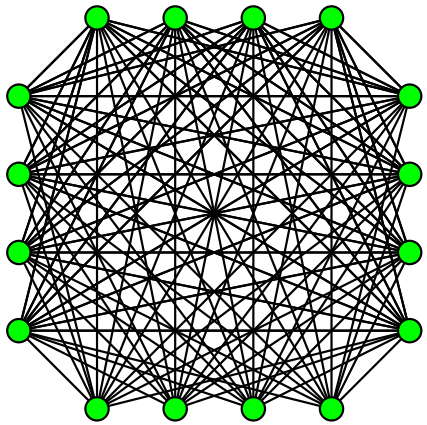


Examples of Branchwidth of Connectivity Functions

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

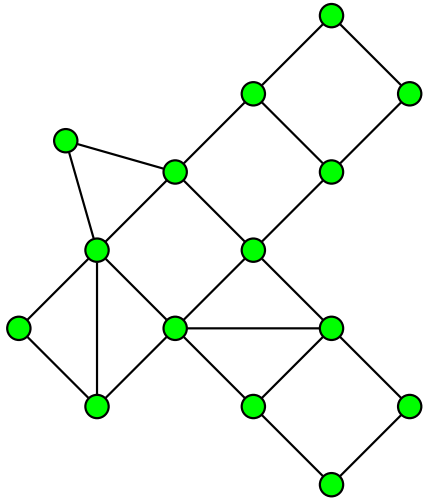


- Rankwidth of a graph:
  - $V = V(G)$
  - $f(A)$ is the GF(2)-rank of the $|A| \times |\overline{A}|$ matrix representing $G[A, \overline{A}]$

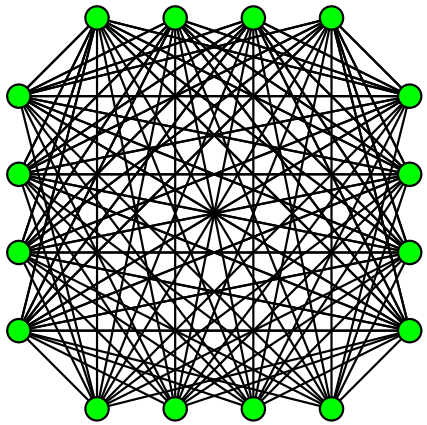


Examples of Branchwidth of Connectivity Functions

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

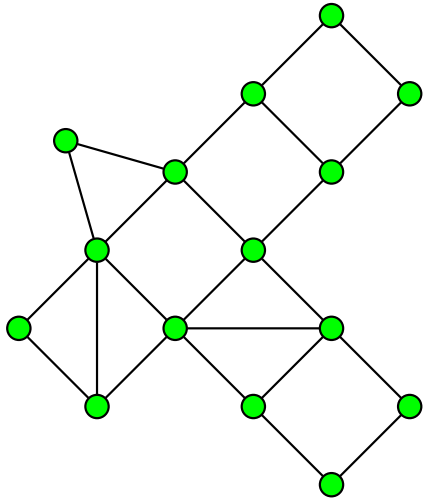


- Rankwidth of a graph:
  - $V = V(G)$
  - $f(A)$ is the GF(2)-rank of the $|A| \times |\overline{A}|$ matrix representing $G[A, \overline{A}]$
  - generalizes treewidth and also some dense graph classes, like cographs

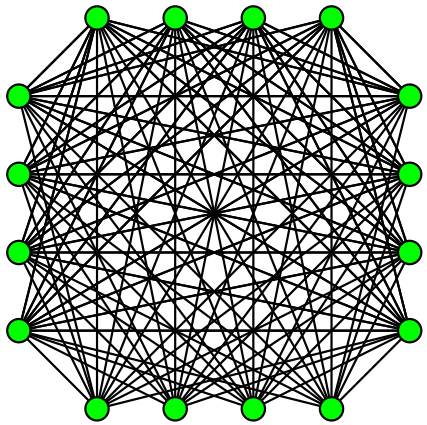


Examples of Branchwidth of Connectivity Functions

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

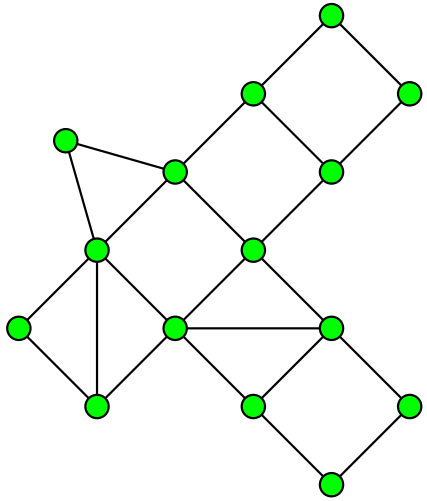


- Rankwidth of a graph:
  - $V = V(G)$
  - $f(A)$ is the GF(2)-rank of the $|A| \times |\overline{A}|$ matrix representing $G[A, \overline{A}]$
  - generalizes treewidth and also some dense graph classes, like cographs
  - Rankwidth $\approx$ cliquewidth

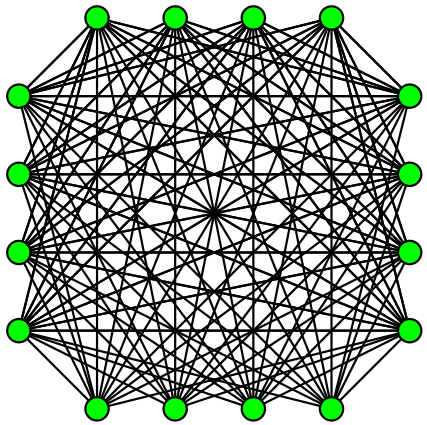


Examples of Branchwidth of Connectivity Functions

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



- Rankwidth of a graph:
  - $V = V(G)$
  - $f(A)$ is the GF(2)-rank of the $|A| \times |\overline{A}|$ matrix representing $G[A, \overline{A}]$
  - generalizes treewidth and also some dense graph classes, like cographs
  - Rankwidth $\approx$ cliquewidth

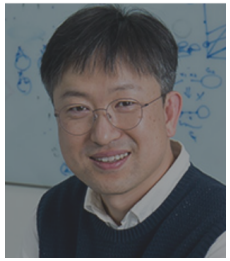


- Also carving-width, matroid branchwidth, rankwidth over different fields...

FPT-Approximation of Branchwidth

Oum & Seymour ’06:

- 3-approximation of branchwidth of connectivity function in time $8^k n^{\mathcal{O}(1)}$

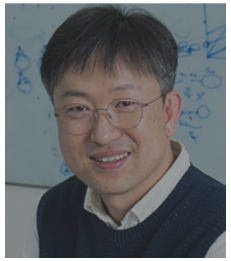



FPT-Approximation of Branchwidth

Oum & Seymour '06:

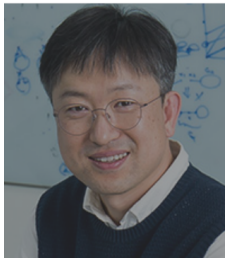
- 3-approximation of branchwidth of connectivity function in time $8^k n^{\mathcal{O}(1)}$

$\Rightarrow$ 3-approximation of rankwidth in time $\mathcal{O}(8^k n^9 \log n)$




FPT-Approximation of Branchwidth

Oum & Seymour '06:



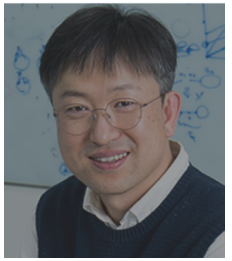

- 3-approximation of branchwidth of connectivity function in time $8^k n^{\mathcal{O}(1)}$

$\Rightarrow$ 3-approximation of rankwidth in time $\mathcal{O}(8^k n^9 \log n)$

$\Rightarrow$ 2^{3k+2}-approximation of cliquewidth in time $\mathcal{O}(8^k n^9 \log n)$

FPT-Approximation of Branchwidth

Oum & Seymour '06:




- 3-approximation of branchwidth of connectivity function in time $8^k n^{\mathcal{O}(1)}$

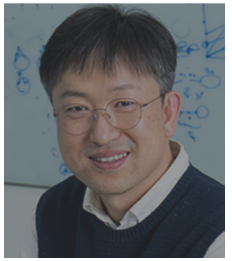
$\Rightarrow$ 3-approximation of rankwidth in time $\mathcal{O}(8^k n^9 \log n)$

$\Rightarrow$ 2^{3k+2}-approximation of cliquewidth in time $\mathcal{O}(8^k n^9 \log n)$

$\Rightarrow$ $f(k)n^9 \log n$ time algorithms parameterized by cliquewidth/rankwidth

FPT-Approximation of Branchwidth

Oum & Seymour '06:




- 3-approximation of branchwidth of connectivity function in time $8^k n^{\mathcal{O}(1)}$

$\Rightarrow$ 3-approximation of rankwidth in time $\mathcal{O}(8^k n^9 \log n)$

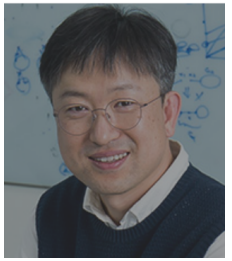
$\Rightarrow$ 2^{3k+2}-approximation of cliquewidth in time $\mathcal{O}(8^k n^9 \log n)$

$\Rightarrow$ $f(k)n^9 \log n$ time algorithms parameterized by cliquewidth/rankwidth

- Similar top-down construction as Robertson-Seymour algorithm

FPT-Approximation of Branchwidth

Oum & Seymour '06:




- 3-approximation of branchwidth of connectivity function in time $8^k n^{\mathcal{O}(1)}$

$\Rightarrow$ 3-approximation of rankwidth in time $\mathcal{O}(8^k n^9 \log n)$

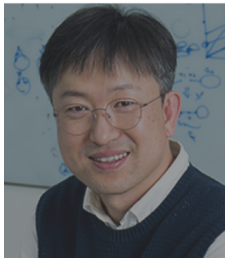
$\Rightarrow$ 2^{3k+2}-approximation of cliquewidth in time $\mathcal{O}(8^k n^9 \log n)$

$\Rightarrow$ $f(k)n^9 \log n$ time algorithms parameterized by cliquewidth/rankwidth

- Similar top-down construction as Robertson-Seymour algorithm
- Rankwidth 3-approximation improved to $f(k)n^3$ time by [Oum'08]

FPT-Approximation of Branchwidth

Oum & Seymour '06:




- 3-approximation of branchwidth of connectivity function in time $8^k n^{\mathcal{O}(1)}$

$\Rightarrow$ 3-approximation of rankwidth in time $\mathcal{O}(8^k n^9 \log n)$

$\Rightarrow$ 2^{3k+2}-approximation of cliquewidth in time $\mathcal{O}(8^k n^9 \log n)$

$\Rightarrow$ $f(k)n^9 \log n$ time algorithms parameterized by cliquewidth/rankwidth

- Similar top-down construction as Robertson-Seymour algorithm
- Rankwidth 3-approximation improved to $f(k)n^3$ time by [Oum'08]
- Further improved to $f(k)n^3$ time exact rankwidth by [Hlinený & Oum'08]

New Approach For Approximating Width Parameters

New Approach For Approximating Width Parameters

- [K, FOCS'21]:
  - 2-approximation of treewidth in time $2^{\mathcal{O}(k)}n$

New Approach For Approximating Width Parameters

- [K, FOCS'21]:
  - 2-approximation of treewidth in time $2^{\mathcal{O}(k)}n$
  - Comparison: 5-approximation in time $2^{\mathcal{O}(k)}n$ [Bodlaender, Drange, Dregi, Fomin, Lokshtanov, Pilipczuk '13]

New Approach For Approximating Width Parameters

- [K, FOCS'21]:
  - 2-approximation of treewidth in time $2^{\mathcal{O}(k)}n$
  - Comparison: 5-approximation in time $2^{\mathcal{O}(k)}n$ [Bodlaender, Drange, Dregi, Fomin, Lokshtanov, Pilipczuk '13]

- [Fomin & K, STOC'22]:
  - framework for 2-approximating branchwidth of connectivity functions

New Approach For Approximating Width Parameters

- [K, FOCS'21]:
  - 2-approximation of treewidth in time $2^{\mathcal{O}(k)}n$
  - Comparison: 5-approximation in time $2^{\mathcal{O}(k)}n$ [Bodlaender, Drange, Dregi, Fomin, Lokshtanov, Pilipczuk '13]

- [Fomin & K, STOC'22]:
  - framework for 2-approximating branchwidth of connectivity functions
  - 2-approximation of rankwidth in time $2^{2^{\mathcal{O}(k)}}n^2$

New Approach For Approximating Width Parameters

- [K, FOCS'21]:
  - 2-approximation of treewidth in time $2^{\mathcal{O}(k)}n$
  - Comparison: 5-approximation in time $2^{\mathcal{O}(k)}n$ [Bodlaender, Drange, Dregi, Fomin, Lokshtanov, Pilipczuk '13]

- [Fomin & K, STOC'22]:
  - framework for 2-approximating branchwidth of connectivity functions
  - 2-approximation of rankwidth in time $2^{2^{\mathcal{O}(k)}}n^2$
    - Comparison: prior algorithms $f(k)n^3$ time [Oum '08], [Hlinený & Oum'08]

New Approach For Approximating Width Parameters

- [K, FOCS'21]:
  - 2-approximation of treewidth in time $2^{\mathcal{O}(k)}n$
  - Comparison: 5-approximation in time $2^{\mathcal{O}(k)}n$ [Bodlaender, Drange, Dregi, Fomin, Lokshtanov, Pilipczuk '13]

- [Fomin & K, STOC'22]:
  - framework for 2-approximating branchwidth of connectivity functions
  - 2-approximation of rankwidth in time $2^{2^{\mathcal{O}(k)}}n^2$
    - Comparison: prior algorithms $f(k)n^3$ time [Oum '08], [Hlinený & Oum'08]
  - 2-approximation of graph branchwidth in time $2^{\mathcal{O}(k)}n$

New Approach For Approximating Width Parameters

- [K, FOCS'21]:
  - 2-approximation of treewidth in time $2^{\mathcal{O}(k)}n$
  - Comparison: 5-approximation in time $2^{\mathcal{O}(k)}n$ [Bodlaender, Drange, Dregi, Fomin, Lokshtanov, Pilipczuk '13]
- [Fomin & K, STOC'22]:
  - framework for 2-approximating branchwidth of connectivity functions
  - 2-approximation of rankwidth in time $2^{2^{\mathcal{O}(k)}}n^2$
    - Comparison: prior algorithms $f(k)n^3$ time [Oum '08], [Hlinený & Oum'08]
  - 2-approximation of graph branchwidth in time $2^{\mathcal{O}(k)}n$
- Instead of top-down construction, iteratively improve the decomposition

New Approach For Approximating Width Parameters

- [K, FOCS'21]:
  - 2-approximation of treewidth in time $2^{\mathcal{O}(k)}n$
  - Comparison: 5-approximation in time $2^{\mathcal{O}(k)}n$ [Bodlaender, Drange, Dregi, Fomin, Lokshtanov, Pilipczuk '13]
- [Fomin & K, STOC'22]:
  - framework for 2-approximating branchwidth of connectivity functions
  - 2-approximation of rankwidth in time $2^{2^{\mathcal{O}(k)}}n^2$
    - Comparison: prior algorithms $f(k)n^3$ time [Oum '08], [Hlinený & Oum'08]
  - 2-approximation of graph branchwidth in time $2^{\mathcal{O}(k)}n$
- Instead of top-down construction, iteratively improve the decomposition
  - Improves approximation ratios from 3 to 2

New Approach For Approximating Width Parameters

- [K, FOCS'21]:
  - 2-approximation of treewidth in time $2^{\mathcal{O}(k)}n$
  - Comparison: 5-approximation in time $2^{\mathcal{O}(k)}n$ [Bodlaender, Drange, Dregi, Fomin, Lokshtanov, Pilipczuk '13]
- [Fomin & K, STOC'22]:
  - framework for 2-approximating branchwidth of connectivity functions
  - 2-approximation of rankwidth in time $2^{2^{\mathcal{O}(k)}}n^2$
    - Comparison: prior algorithms $f(k)n^3$ time [Oum '08], [Hlinený & Oum'08]
  - 2-approximation of graph branchwidth in time $2^{\mathcal{O}(k)}n$
- Instead of top-down construction, iteratively improve the decomposition
  - Improves approximation ratios from 3 to 2
- Interplay of improvement operations and dynamic programming

New Approach For Approximating Width Parameters

- [K, FOCS'21]:
  - 2-approximation of treewidth in time $2^{\mathcal{O}(k)}n$
  - Comparison: 5-approximation in time $2^{\mathcal{O}(k)}n$ [Bodlaender, Drange, Dregi, Fomin, Lokshtanov, Pilipczuk '13]
- [Fomin & K, STOC'22]:
  - framework for 2-approximating branchwidth of connectivity functions
  - 2-approximation of rankwidth in time $2^{2^{\mathcal{O}(k)}}n^2$
    - Comparison: prior algorithms $f(k)n^3$ time [Oum '08], [Hlinený & Oum'08]
  - 2-approximation of graph branchwidth in time $2^{\mathcal{O}(k)}n$
- Instead of top-down construction, iteratively improve the decomposition
  - Improves approximation ratios from 3 to 2
- Interplay of improvement operations and dynamic programming
  - Improvements in time complexity both as a function of n and as a function of k

Compression algorithms

Our approach is for obtaining *2-approximate compression algorithms*

Compression algorithms

Our approach is for obtaining *2-approximate compression algorithms*

- Input: Decomposition T of width $\mathrm{w}(T) = k$
- Output: Either a decomposition T' of width $\mathrm{w}(T') < \mathrm{w}(T)$, or conclusion that $\mathrm{w}(T)$ is within a factor of 2 from optimal
- Parameter: $\mathrm{w}(T) = k$

Compression algorithms

Our approach is for obtaining *2-approximate compression algorithms*

- Input: Decomposition T of width $\mathrm{w}(T) = k$
- Output: Either a decomposition T' of width $\mathrm{w}(T') < \mathrm{w}(T)$, or conclusion that $\mathrm{w}(T)$ is within a factor of 2 from optimal
- Parameter: $\mathrm{w}(T) = k$

- For treewidth and graph branchwidth: $f(k) \cdot n$ time c-approximate compression algorithm $\rightarrow$ $k^{\mathcal{O}(1)} \cdot f(k) \cdot n$ time c-approximation algorithm [Bodlaender'93]

Compression algorithms

Our approach is for obtaining *2-approximate compression algorithms*

- Input: Decomposition T of width $\mathrm{w}(T) = k$
- Output: Either a decomposition T' of width $\mathrm{w}(T') < \mathrm{w}(T)$, or conclusion that $\mathrm{w}(T)$ is within a factor of 2 from optimal
- Parameter: $\mathrm{w}(T) = k$

- For treewidth and graph branchwidth: $f(k) \cdot n$ time c-approximate compression algorithm $\rightarrow k^{\mathcal{O}(1)} \cdot f(k) \cdot n$ time c-approximation algorithm [Bodlaender'93]
- For rankwidth: $f(k) \cdot n$ time c-approximate compression algorithm $\rightarrow 2^{\mathcal{O}(k)} \cdot f(k) \cdot n^2$ time c-approximation algorithm

Plan

- 2-Approximating branchwidth of connectivity functions
  1. Combinatorial framework
  2. Algorithmic framework

Combinatorial 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 $k \leq 2\text{bw}(f)$

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 $k \leq 2\text{bw}(f)$
- Strategy:
  - Let $h(T)$ be the number of edges of T of width k (heavy edges)
  - If $k > 2 \cdot \text{bw}(f)$, then a **refinement operation** can be applied, which decreases $h(T)$ and does not increase $\text{w}(T)$

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 $k \leq 2\text{bw}(f)$

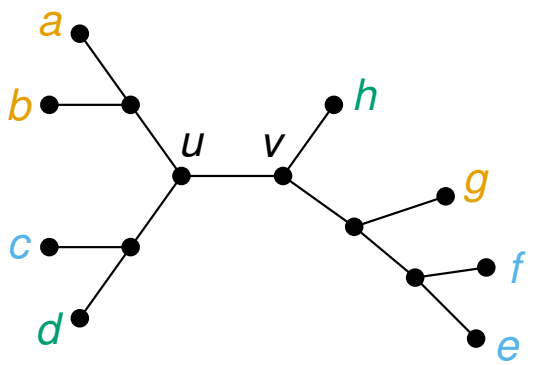
- Strategy:
  - Let $h(T)$ be the number of edges of T of width k (heavy edges)
  - If $k > 2 \cdot \text{bw}(f)$, then a **refinement operation** can be applied, which decreases $h(T)$ and does not increase $\text{w}(T)$

$\Rightarrow$ after at most n refinement operations, $\text{w}(T)$ decreases

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

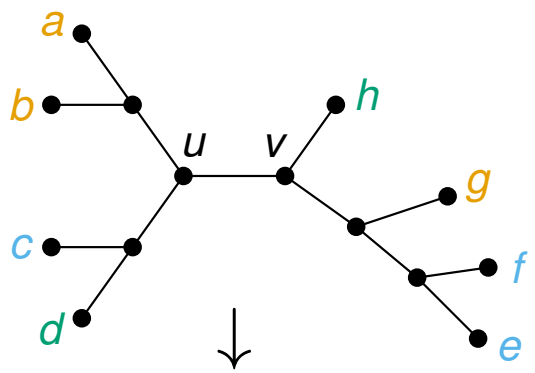
Example with $(uv, C_1, C_2, C_3) = (uv, \{a, b, g\}, \{c, e, f\}, \{d, h\})$

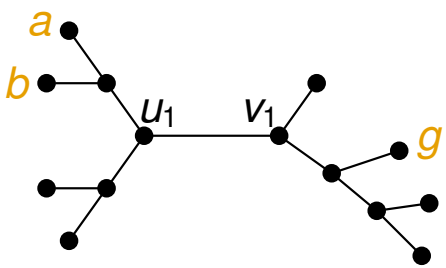


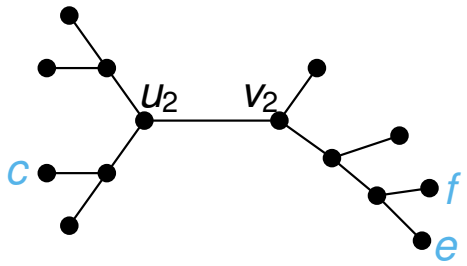
Refinement operation

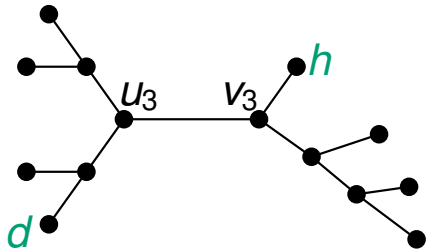
Specified by 4-tuple (uv, C_1, C_2, C_3), where $uv \in E(T)$ and (C_1, C_2, C_3) tripartition of V

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





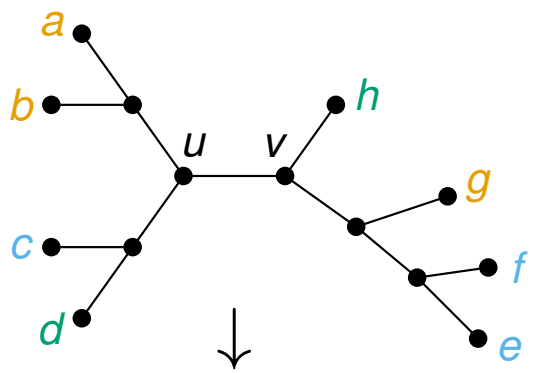


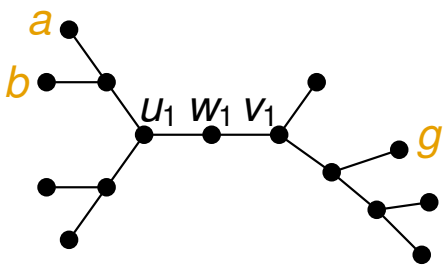


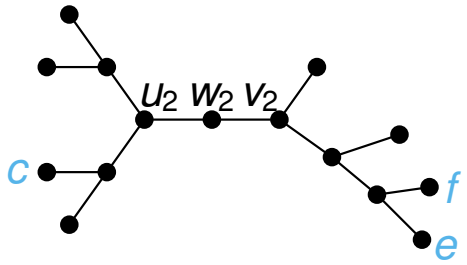
Refinement operation

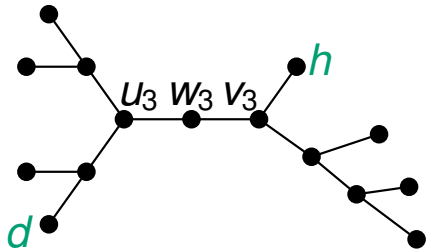
Specified by 4-tuple (uv, C_1, C_2, C_3), where $uv \in E(T)$ and (C_1, C_2, C_3) tripartition of V

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





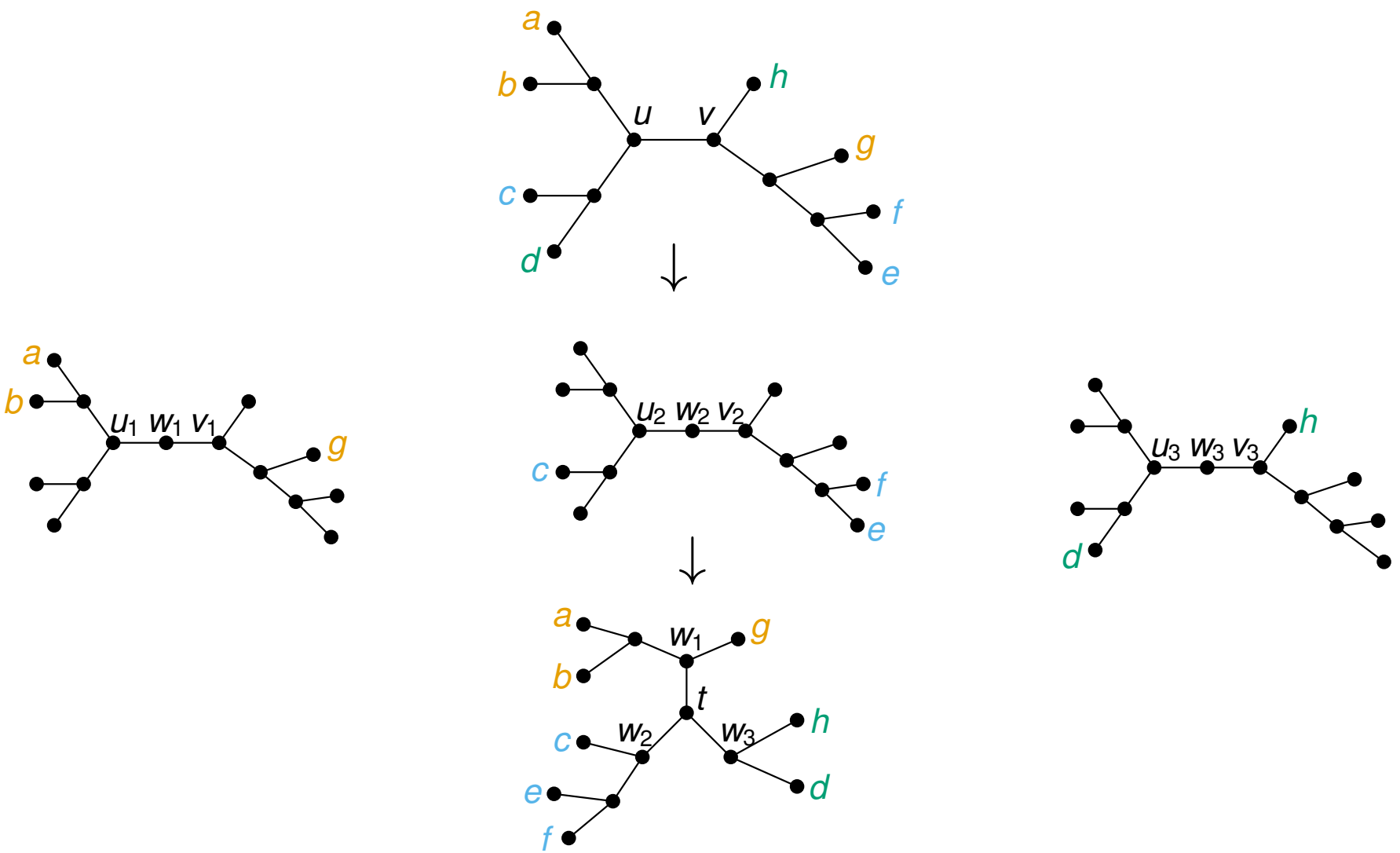




Refinement operation

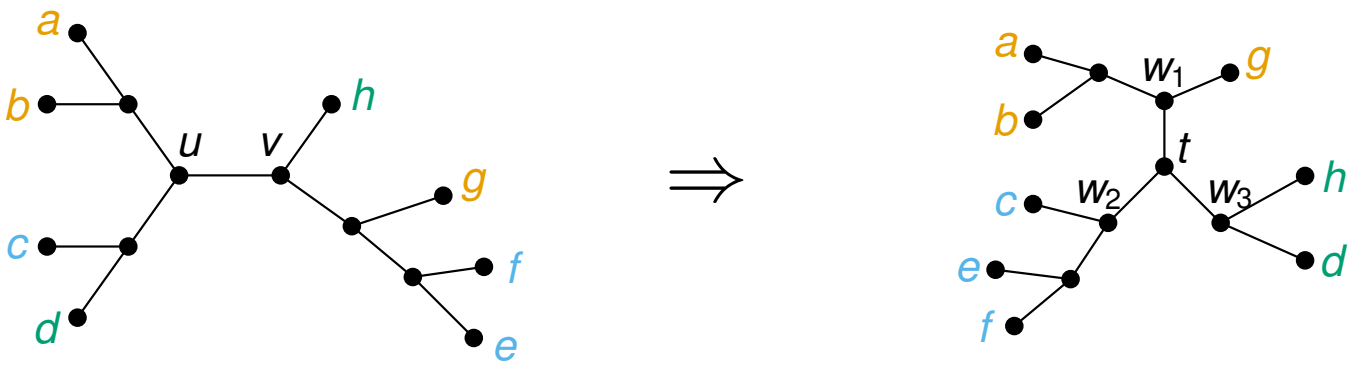
Specified by 4-tuple (uv, C_1, C_2, C_3), where $uv \in E(T)$ and (C_1, C_2, C_3) tripartition of V

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



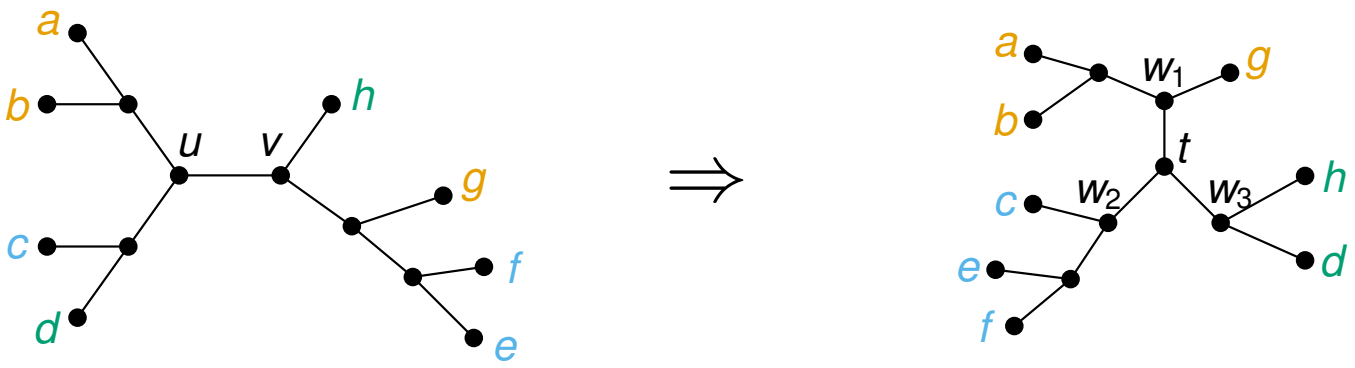
Observations on Refinement

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



Observations on Refinement

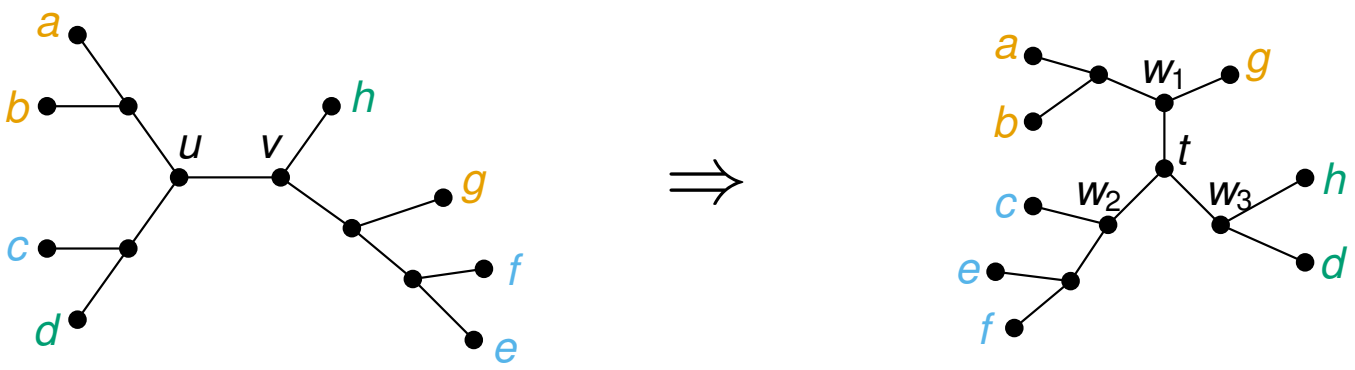
Example with $(r, C_1, C_2, C_3) = (uv, \{a, b, g\}, \{c, e, f\}, \{d, h\})$



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

Observations on Refinement

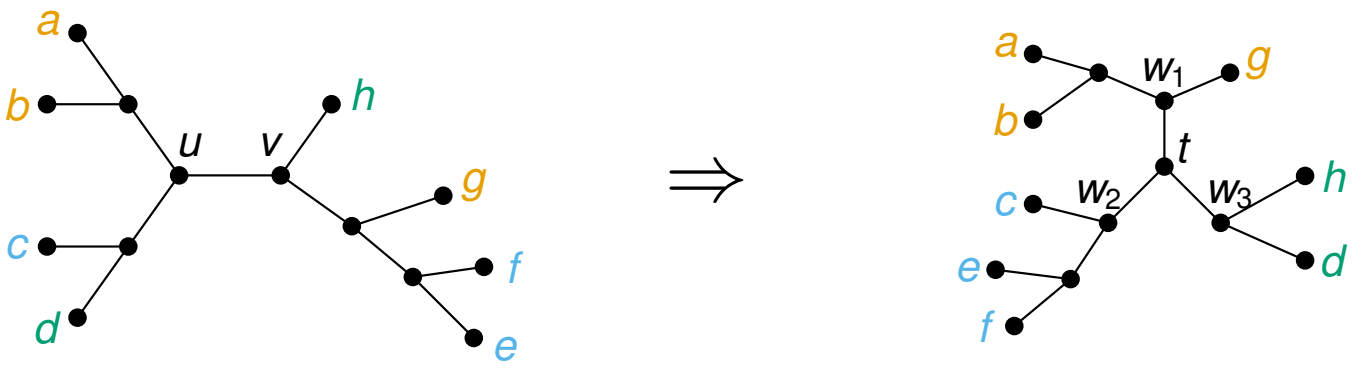
Example with $(r, C_1, C_2, C_3) = (uv, \{a, b, g\}, \{c, e, f\}, \{d, h\})$



- Observation 1: For each i, there will be an edge $w_i t$ corresponding to $(C_i, \overline{C_i})$
- Let $(W, \overline{W}) = (\{a, b, c, d\}, \{e, f, g, h\})$ be the cut of uv

Observations on Refinement

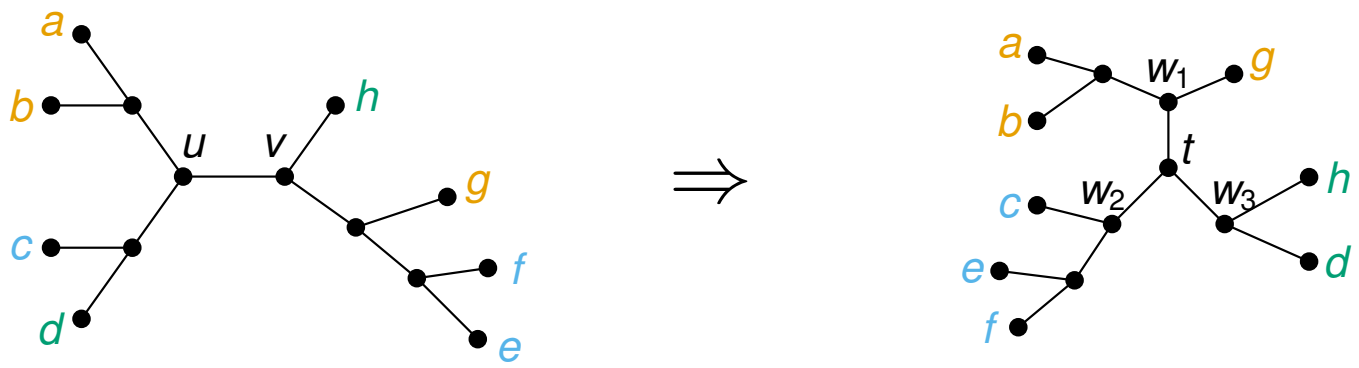
Example with $(r, C_1, C_2, C_3) = (uv, \{a, b, g\}, \{c, e, f\}, \{d, h\})$



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

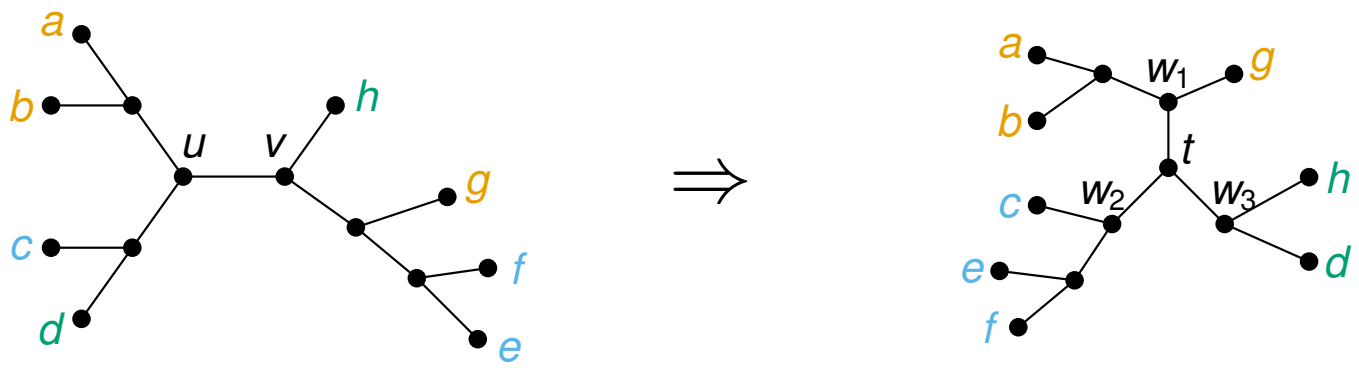
Example with $(uv, C_1, C_2, C_3) = (uv, \{a, b, g\}, \{c, e, f\}, \{d, h\})$



- 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 $i \in \{1, 2, 3\}$

Local Improvement

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



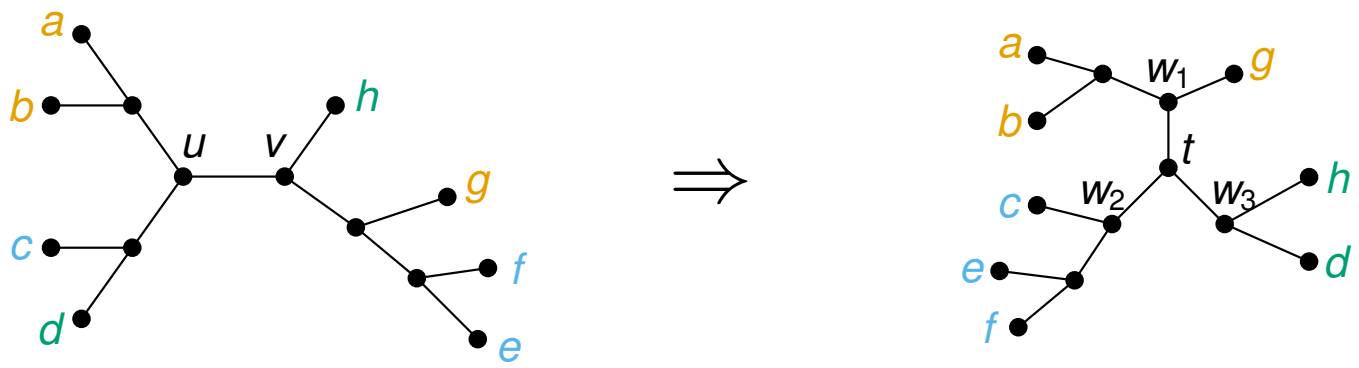
- Let $(W, \overline{W}) = (\{a, b, c, d\}, \{e, f, g, h\})$ be the cut of uv
- Combination of Observation 1 and 2:
  - The widths of edges "near the center" will be $f(C_i)$, $f(C_i \cap W)$, and $f(C_i \cap \overline{W})$ for each $i \in \{1, 2, 3\}$

Theorem

For any set $W \subseteq V$ with $f(W) > 2\text{bw}(f)$ 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 $(uv, C_1, C_2, C_3) = (uv, \{a, b, g\}, \{c, e, f\}, \{d, h\})$



- 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 $i \in \{1, 2, 3\}$

Theorem

For any set $W \subseteq V$ with $f(W) > 2\text{bw}(f)$ 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 $f(C_i \cap \overline{W}) < f(W)$.

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

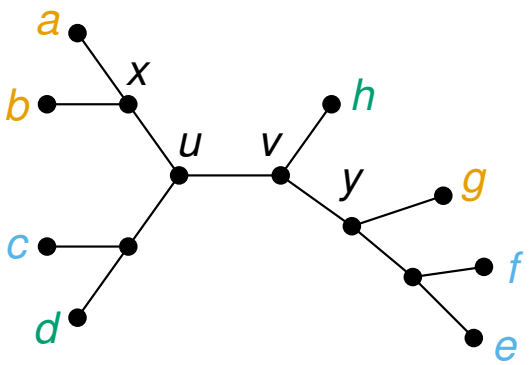
Global Improvement

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

Theorem

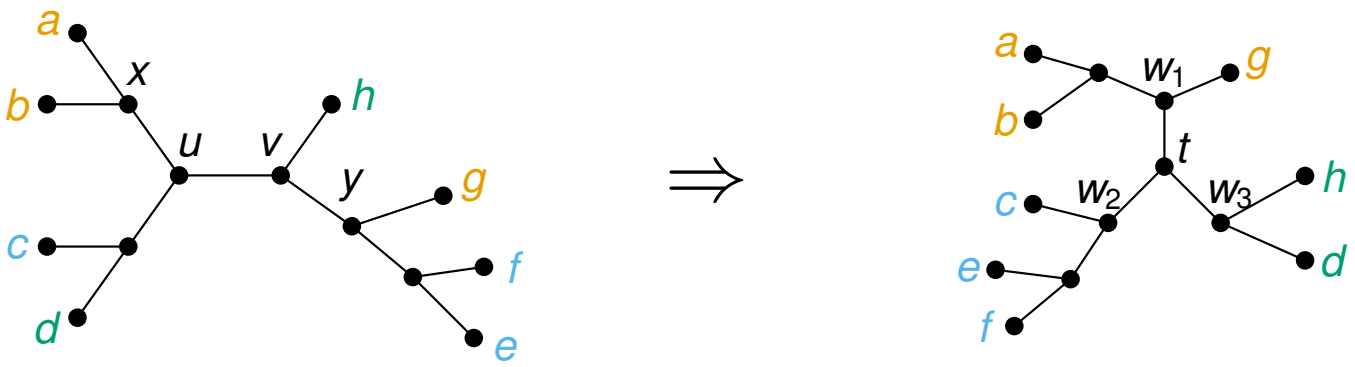
If there exists a W-improvement, then there exists a W-improvement (C_1, C_2, C_3) so that refinement with (uv, C_1, C_2, C_3) does not increase width and decreases the number of heavy edges.

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 (uv, 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

- Let $(W, \overline{W})$ be a cut corresponding to an edge uv of the decomposition

Global Improvement: Construction

- Let $(W, \overline{W})$ be a cut corresponding to an edge uv of the decomposition
- A minimum W-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

- Let $(W, \overline{W})$ be a cut corresponding to an edge uv of the decomposition
- A minimum W-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 minimum W-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

- Let $(W, \overline{W})$ be a cut corresponding to an edge uv of the decomposition
- A minimum W-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 minimum W-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 $T_r[x] \subseteq C_i$.

- For each edge e of T, at most one of the new edges corresponding to e has width $f(e)$, others have width $< f(e)$

Global Improvement: Construction

- Let $(W, \overline{W})$ be a cut corresponding to an edge uv of the decomposition
- A minimum W-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 minimum W-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 $T_r[x] \subseteq C_i$.

- For each edge e of T, at most one of the new edges corresponding to e has width $f(e)$, others have width $< f(e)$
- For the edge uv, none of the new edges corresponding to uv has width $f(uv)$
  $\Rightarrow$ Strict improvement

Algorithmic Framework

First Algorithm

- General algorithm for improving a branch decomposition:

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

First Algorithm

- General algorithm for improving a branch decomposition:

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 algorithm for improving a branch decomposition:

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 compute minimum W-improvement or conclude $k \leq 2\text{bw}(f)$

First Algorithm

- General algorithm for improving a branch decomposition:

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

First Algorithm

- General algorithm for improving a branch decomposition:

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

$\Rightarrow$ The number of heavy edges decreased

First Algorithm

- General algorithm for improving a branch decomposition:

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

$\Rightarrow$ The number of heavy edges decreased

5. Repeat until the width of T decreases (at most n iterations)

First Algorithm

- General algorithm for improving a branch decomposition:

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

$\Rightarrow$ The number of heavy edges decreased

5. Repeat 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

First Algorithm

- General algorithm for improving a branch decomposition:

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

$\Rightarrow$ The number of heavy edges decreased

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 $t(k) \cdot n$

Framework for Fast Algorithms

Assume dynamic programming data structure for computing minimum W-improvements with time complexity $t(k)$ per node

Framework for Fast Algorithms

Assume dynamic programming data structure for computing minimum W-improvements with time complexity $t(k)$ per node

$\Rightarrow$

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 $k \leq 2\text{bw}(f)$.

Framework for Fast Algorithms

Assume dynamic programming data structure for computing minimum W-improvements with time complexity $t(k)$ per node

$\Rightarrow$

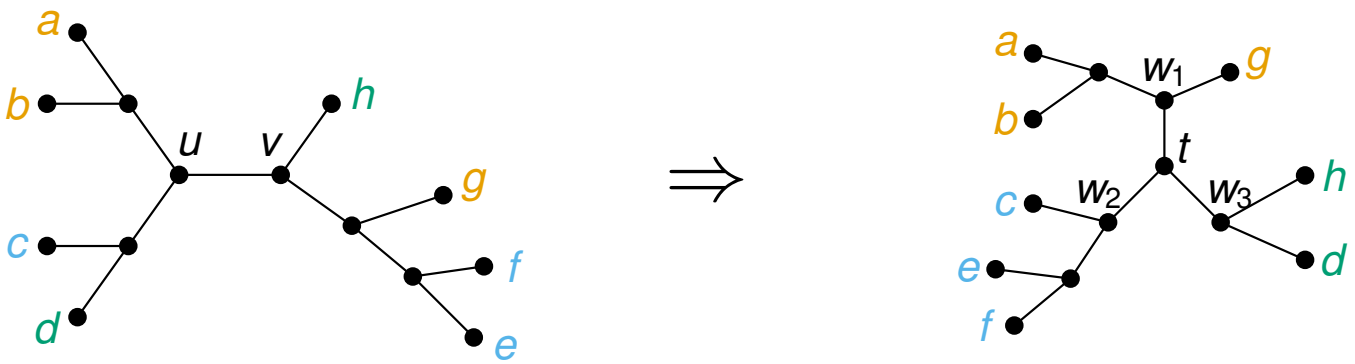
Theorem

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 $k \leq 2\text{bw}(f)$.

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

Amortization technique

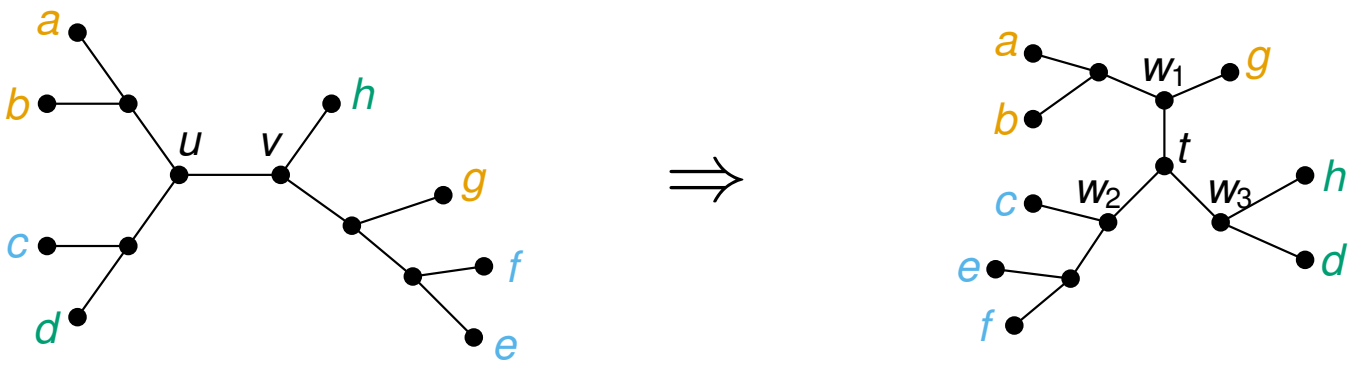
Example with $(uv, C_1, C_2, C_3) = (uv, \{a, b, g\}, \{c, e, f\}, \{d, h\})$



- Consider T rooted at $r = uv$

Amortization technique

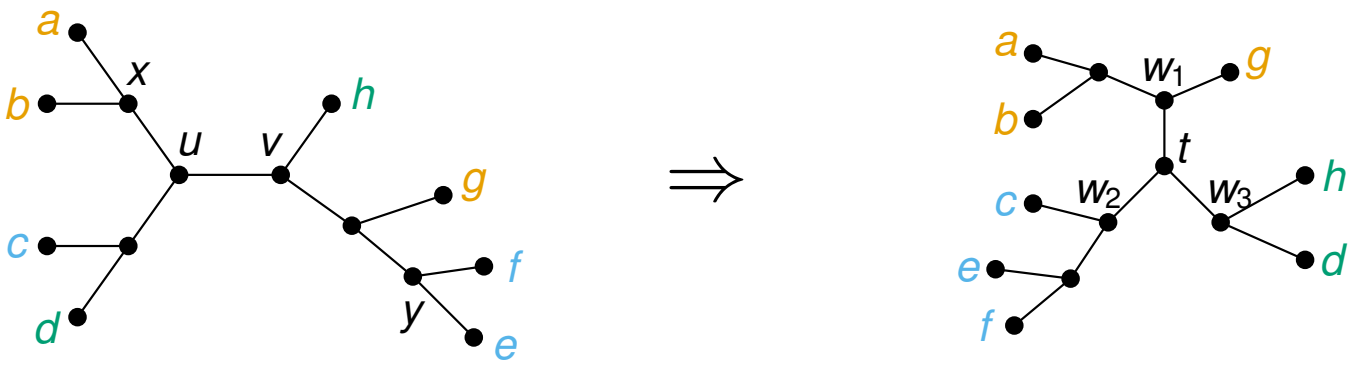
Example with $(uv, C_1, C_2, C_3) = (uv, \{a, b, g\}, \{c, e, f\}, \{d, h\})$



- 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

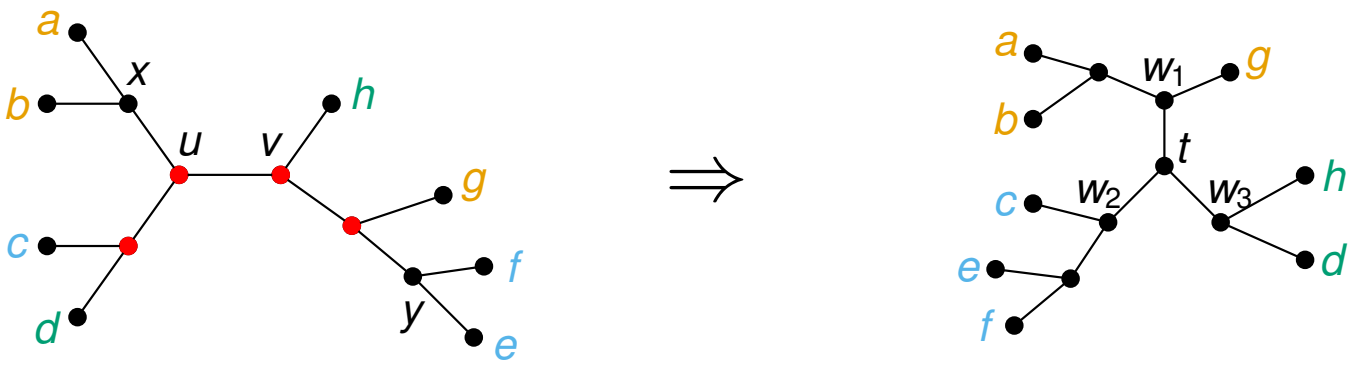
Example with $(uv, C_1, C_2, C_3) = (uv, \{a, b, g\}, \{c, e, f\}, \{d, h\})$



- 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

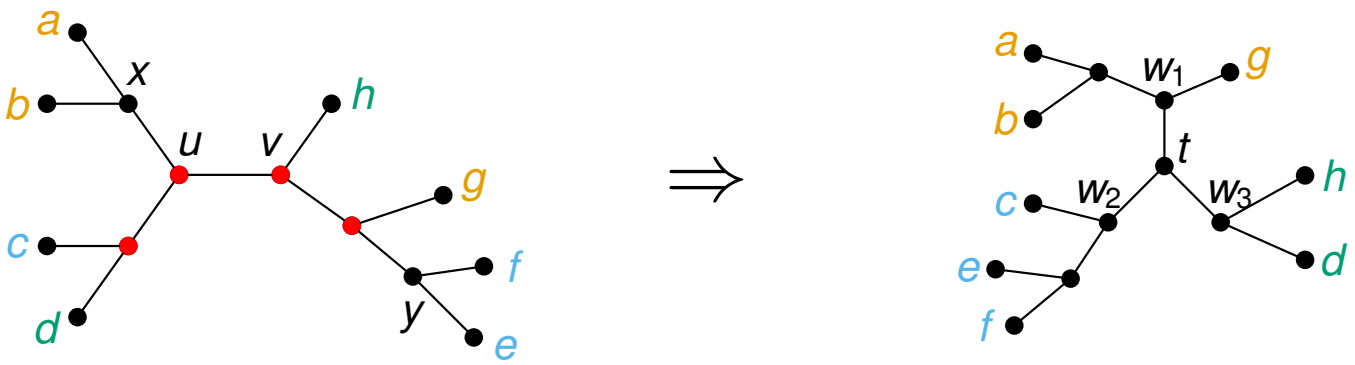
Example with $(uv, C_1, C_2, C_3) = (uv, \{a, b, g\}, \{c, e, f\}, \{d, h\})$



- 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

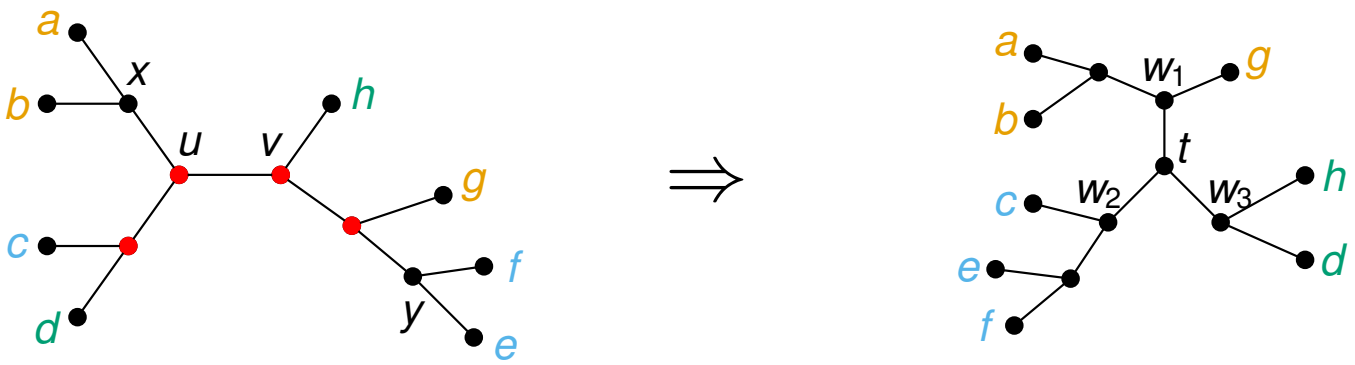
Example with $(uv, C_1, C_2, C_3) = (uv, \{a, b, g\}, \{c, e, f\}, \{d, h\})$



- 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
  - Implement refinement by changing only R, in time $\mathcal{O}(t(k) \cdot |R|)$

Amortization technique

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



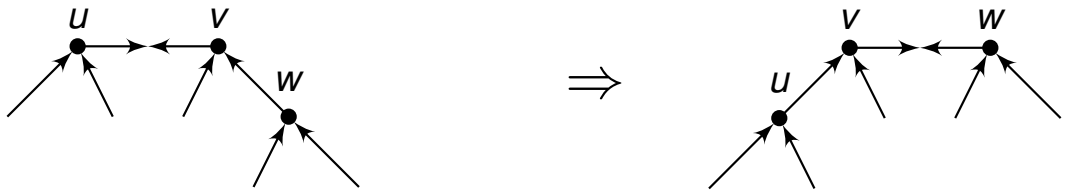
- Consider T rooted at $r = uv$
- Observation: If $T_r[x] \subseteq C_i$, then the subtree of x appears identically in refinement
  - Example: $T_r[x] = \{a, b\} \subseteq C_1$ and $T_r[y] = \{e, f\} \subseteq C_2$
- Call the nodes for which this does **not** happen the edit set R of the refinement
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

The Algorithm

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

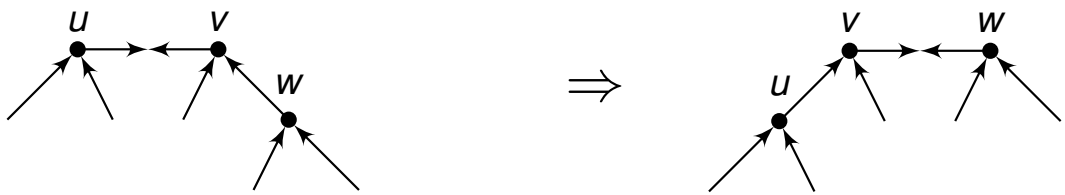
The Algorithm

- 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

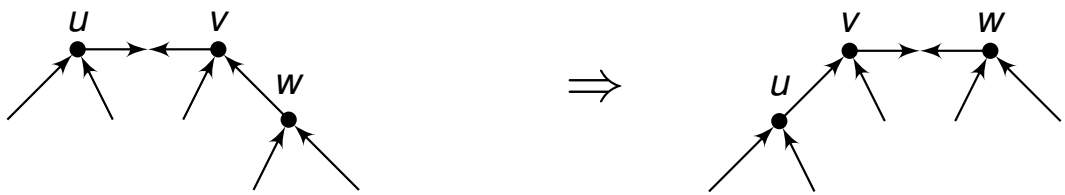
- Maintain dynamic programming tables towards a root edge $r = uv$
- When changing $r = uv$ to an incident edge $r' = vw$, only the table of v needs to be recomputed



- 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 time complexity $t(k)2^{\mathcal{O}(k)}n$

Conclusion

- New approach for 2-approximation of width parameters

Conclusion

- New approach for 2-approximation of width parameters

- Main applications:
  - $2^{\mathcal{O}(k)}n$ time 2-approximation algorithm for treewidth
  - $2^{2^{\mathcal{O}(k)}}n^2$ time 2-approximation algorithm for rankwidth

Conclusion

- New approach for 2-approximation of width parameters
- Main applications:
  - $2^{\mathcal{O}(k)}n$ time 2-approximation algorithm for treewidth
  - $2^{2^{\mathcal{O}(k)}}n^2$ time 2-approximation algorithm for rankwidth
- Open problems:

Conclusion

- New approach for 2-approximation of width parameters
- Main applications:
  - $2^{\mathcal{O}(k)}n$ time 2-approximation algorithm for treewidth
  - $2^{2^{\mathcal{O}(k)}}n^2$ time 2-approximation algorithm for rankwidth
- Open problems:
  - Better than 2-approximation of treewidth in time $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$?

Conclusion

- New approach for 2-approximation of width parameters
- Main applications:
  - $2^{\mathcal{O}(k)}n$ time 2-approximation algorithm for treewidth
  - $2^{2^{\mathcal{O}(k)}}n^2$ time 2-approximation algorithm for rankwidth
- Open problems:
  - Better than 2-approximation of treewidth in time $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$?
  - $f(k)m^{1.99}$ time $g(k)$-approximation of rankwidth?

Conclusion

- New approach for 2-approximation of width parameters
- Main applications:
  - $2^{\mathcal{O}(k)}n$ time 2-approximation algorithm for treewidth
  - $2^{2^{\mathcal{O}(k)}}n^2$ time 2-approximation algorithm for rankwidth
- Open problems:
  - Better than 2-approximation of treewidth in time $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$?
  - $f(k)m^{1.99}$ time $g(k)$-approximation of rankwidth?
  - XP-approximation of mim-width, twin-width?

Thank you!