A Single-Exponential Time 2-Approximation Algorithm for Treewidth

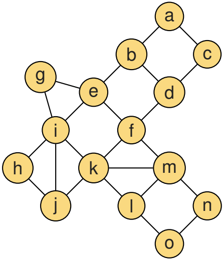
Tuukka Korhonen

University of Bergen

Online seminar of AlGCo
Jan 20, 2022

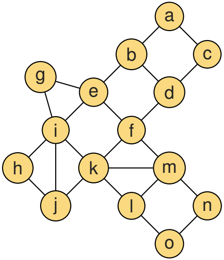
Treewidth

- Measures how close a graph is to a tree



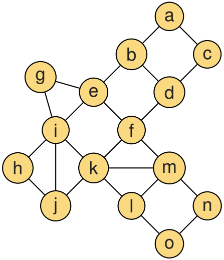
Treewidth

- Measures how close a graph is to a tree
  - Trees have treewidth 1



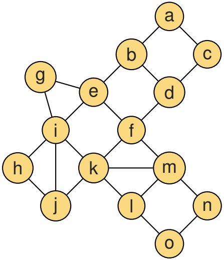
Treewidth

- Measures how close a graph is to a tree
  - Trees have treewidth 1
  - The example graph has treewidth 2



Treewidth

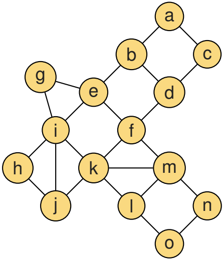
- Measures how close a graph is to a tree
  - Trees have treewidth 1
  - The example graph has treewidth 2
  - The $n \times n$ -grid has treewidth n

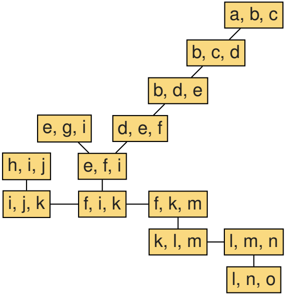


Treewidth

- Measures how close a graph is to a tree
  - Trees have treewidth 1
  - The example graph has treewidth 2
  - The $n \times n$ -grid has treewidth n

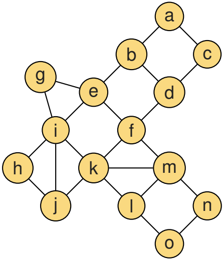
- Treewidth is defined as the minimum width of a *tree decomposition*





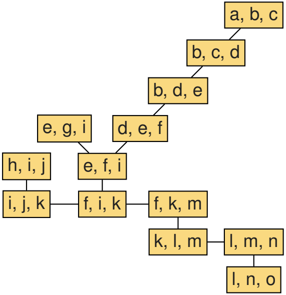
Treewidth

- Measures how close a graph is to a tree
  - Trees have treewidth 1
  - The example graph has treewidth 2
  - The $n \times n$ -grid has treewidth n



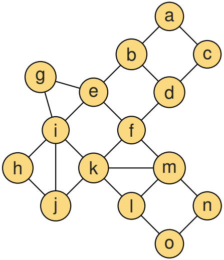
- Treewidth is defined as the minimum width of a *tree decomposition*

- A tree decomposition of a graph G is a tree T of bags $B_i \subseteq V(G)$ so that:
  1. every vertex of G is in a bag
  2. every edge of G is in a bag
  3. for each vertex of G, the bags containing it form a connected subtree of T (connectedness condition)



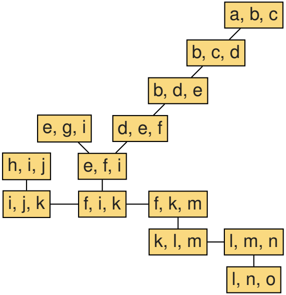
Treewidth

- Measures how close a graph is to a tree
  - Trees have treewidth 1
  - The example graph has treewidth 2
  - The $n \times n$ -grid has treewidth n



- Treewidth is defined as the minimum width of a *tree decomposition*

- A tree decomposition of a graph G is a tree T of bags $B_i \subseteq V(G)$ so that:
  1. every vertex of G is in a bag
  2. every edge of G is in a bag
  3. for each vertex of G, the bags containing it form a connected subtree of T (connectedness condition)



Width 2

- The width of a tree decomposition is $\max |B_i| - 1$

Algorithms using tree decompositions

Hundreds of results of form:

Given an n-vertex graph with a tree decomposition of width k, some combinatorial problem can be solved in time $f(k)n^c$

Algorithms using tree decompositions

Hundreds of results of form:

Given an n-vertex graph with a tree decomposition of width k, some combinatorial problem can be solved in time $f(k)n^c$

Often $f(k) = 2^{\mathcal{O}(k)}$ and $c = 1$

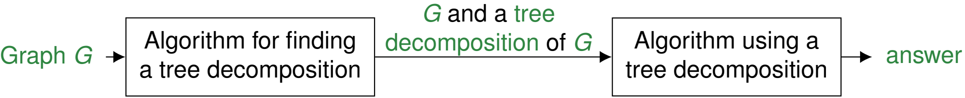
Algorithms using tree decompositions

Hundreds of results of form:

Given an n-vertex graph with a tree decomposition of width k, some combinatorial problem can be solved in time $f(k)n^c$

Often $f(k) = 2^{\mathcal{O}(k)}$ and $c = 1$

Here, the algorithm will look as follows:



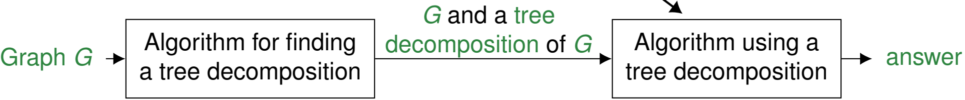
Algorithms using tree decompositions

Hundreds of results of form:

> Given an n-vertex graph with a tree decomposition of width k, some combinatorial problem can be solved in time $f(k)n^c$

Often $f(k) = 2^{\mathcal{O}(k)}$ and $c = 1$

Here, the algorithm will look as follows:



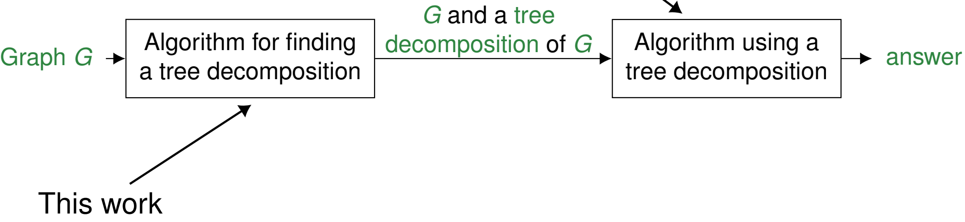
Algorithms using tree decompositions

Hundreds of results of form:

> Given an n-vertex graph with a tree decomposition of width k, some combinatorial problem can be solved in time $f(k)n^c$

Often $f(k) = 2^{\mathcal{O}(k)}$ and $c = 1$

Here, the algorithm will look as follows:



Algorithms for finding tree decompositions

Long history of algorithms for finding tree decompositions

Algorithms for finding tree decompositions

Long history of algorithms for finding tree decompositions

- Arnborg, Corneil, Proskurowski '87: Optimal tree decomposition in time $\mathcal{O}(n^{k+2})$

Algorithms for finding tree decompositions

Long history of algorithms for finding tree decompositions

- Arnborg, Corneil, Proskurowski '87: Optimal tree decomposition in time $\mathcal{O}(n^{k+2})$
- Robertson and Seymour [GM XIII,'95]: 4-approximation in time $2^{\mathcal{O}(k)}n^2$

Algorithms for finding tree decompositions

Long history of algorithms for finding tree decompositions

- Arnborg, Corneil, Proskurowski '87: Optimal tree decomposition in time $\mathcal{O}(n^{k+2})$
- Robertson and Seymour [GM XIII,'95]: 4-approximation in time $2^{\mathcal{O}(k)} n^2$
- Reed '92: 8-approximation in time $2^{\mathcal{O}(k \log k)} n \log n$

Algorithms for finding tree decompositions

Long history of algorithms for finding tree decompositions

- Arnborg, Corneil, Proskurowski ’87: Optimal tree decomposition in time $\mathcal{O}(n^{k+2})$
- Robertson and Seymour [GM XIII,’95]: 4-approximation in time $2^{\mathcal{O}(k)}n^2$
- Reed ’92: 8-approximation in time $2^{\mathcal{O}(k \log k)} n \log n$
- Bodlaender ’96: Optimal in time $2^{\mathcal{O}(k^3)} n$

Algorithms for finding tree decompositions

Long history of algorithms for finding tree decompositions

- Arnborg, Corneil, Proskurowski '87: Optimal tree decomposition in time $\mathcal{O}(n^{k+2})$
- Robertson and Seymour [GM XIII,'95]: 4-approximation in time $2^{\mathcal{O}(k)} n^2$
- Reed '92: 8-approximation in time $2^{\mathcal{O}(k \log k)} n \log n$
- Bodlaender '96: Optimal in time $2^{\mathcal{O}(k^3)} n$
- Bodlaender, Drange, Dregi, Fomin, Lokshtanov, Pilipczuk '13:
  5-approximation in time $2^{\mathcal{O}(k)} n$ and 3-approximation in time $2^{\mathcal{O}(k)} n \log n$

Algorithms for finding tree decompositions

Long history of algorithms for finding tree decompositions

- Arnborg, Corneil, Proskurowski '87: Optimal tree decomposition in time $\mathcal{O}(n^{k+2})$
- Robertson and Seymour [GM XIII,'95]: 4-approximation in time $2^{\mathcal{O}(k)} n^2$
- Reed '92: 8-approximation in time $2^{\mathcal{O}(k \log k)} n \log n$
- Bodlaender '96: Optimal in time $2^{\mathcal{O}(k^3)} n$
- Bodlaender, Drange, Dregi, Fomin, Lokshtanov, Pilipczuk '13:
  5-approximation in time $2^{\mathcal{O}(k)} n$ and 3-approximation in time $2^{\mathcal{O}(k)} n \log n$
- And many others [Lag96, FHL08, Ami10, EJT10, FTV15, FLS+18, BF21]

Algorithms for finding tree decompositions

Long history of algorithms for finding tree decompositions

- Arnborg, Corneil, Proskurowski '87: Optimal tree decomposition in time $\mathcal{O}(n^{k+2})$
- Robertson and Seymour [GM XIII,'95]: 4-approximation in time $2^{\mathcal{O}(k)} n^2$
- Reed '92: 8-approximation in time $2^{\mathcal{O}(k \log k)} n \log n$
- Bodlaender '96: Optimal in time $2^{\mathcal{O}(k^3)} n$
- Bodlaender, Drange, Dregi, Fomin, Lokshtanov, Pilipczuk '13: 5-approximation in time $2^{\mathcal{O}(k)} n$ and 3-approximation in time $2^{\mathcal{O}(k)} n \log n$
- And many others [Lag96, FHL08, Ami10, EJT10, FTV15, FLS$^+$18, BF21]
- This work: 2-approximation in time $2^{\mathcal{O}(k)} n$

Results

Theorem (K., 2021)

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

A new approach for approximating width parameters

Results

Theorem (K., 2021)

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

A new approach for approximating width parameters

We extended this approach to branchwidth of symmetric submodular functions:

Results

Theorem (K., 2021)

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

A new approach for approximating width parameters

We extended this approach to branchwidth of symmetric submodular functions:

Theorem (Fomin and K., 2021)

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

Results

Theorem (K., 2021)

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

A new approach for approximating width parameters

We extended this approach to branchwidth of symmetric submodular functions:

Theorem (Fomin and K., 2021)

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

Improves algorithms on graphs of rankwidth k from $f(k)n^3$ time to $f(k)n^2$ time

Results

Theorem (K., 2021)

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

A new approach for approximating width parameters

We extended this approach to branchwidth of symmetric submodular functions:

Theorem (Fomin and K., 2021)

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

Improves algorithms on graphs of rankwidth k from $f(k)n^3$ time to $f(k)n^2$ time

Theorem (Fomin and K., 2021)

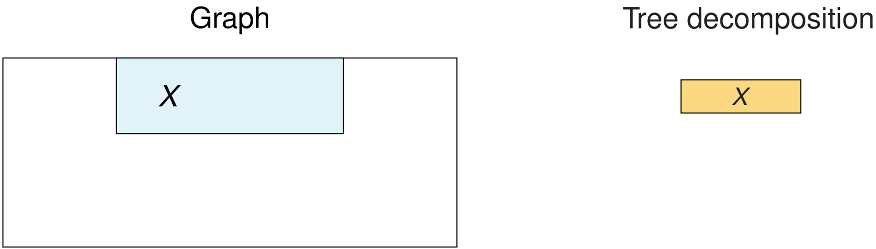
There is a $2^{\mathcal{O}(k)} n$ time 2-approximation algorithm for branchwidth of graphs

Previous algorithms

Previous approximation algorithms for treewidth
[RS95, Lag96, Ree92, FHL08, Ami10, BDD$^+$16, FLS$^+$18, BF21]
build the decomposition in a top-down manner, following [RS95]:

Previous algorithms

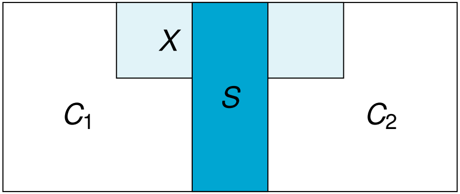
Previous approximation algorithms for treewidth
[RS95, Lag96, Ree92, FHL08, Ami10, BDD+16, FLS+18, BF21]
build the decomposition in a top-down manner, following [RS95]:



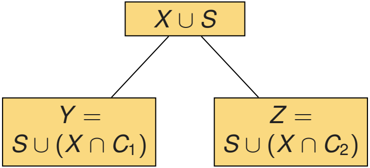
Previous algorithms

Previous approximation algorithms for treewidth [RS95, Lag96, Ree92, FHL08, Ami10, BDD$^+$16, FLS$^+$18, BF21] build the decomposition in a top-down manner, following [RS95]:

Graph



Tree decomposition

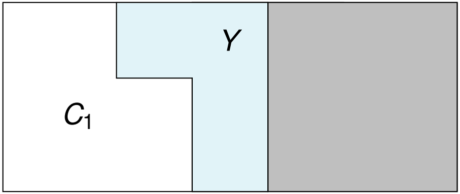


Separator S with components C_1 and C_2

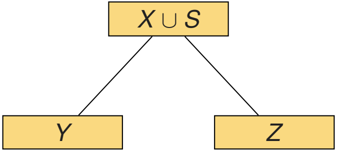
Previous algorithms

Previous approximation algorithms for treewidth [RS95, Lag96, Ree92, FHL08, Ami10, BDD+16, FLS+18, BF21] build the decomposition in a top-down manner, following [RS95]:

Graph

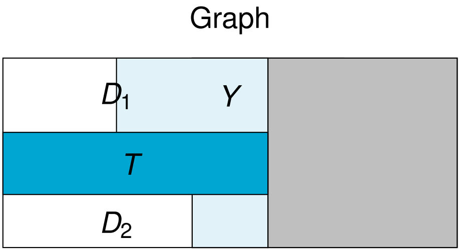


Tree decomposition

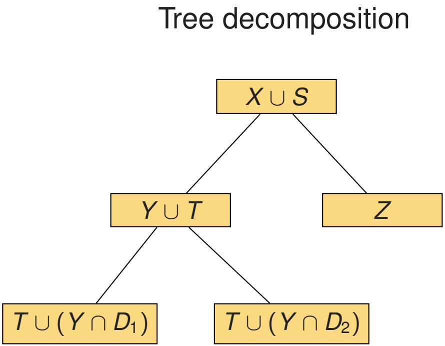


Previous algorithms

Previous approximation algorithms for treewidth [RS95, Lag96, Ree92, FHL08, Ami10, BDD$^+$16, FLS$^+$18, BF21] build the decomposition in a top-down manner, following [RS95]:

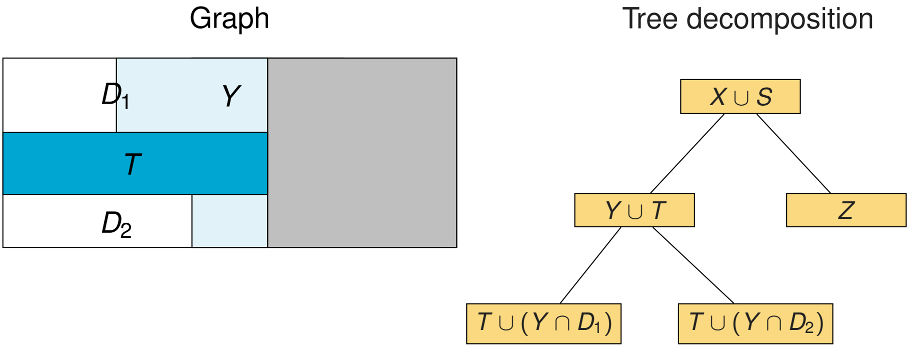


Separator T with components D_1 and D_2



Previous algorithms

Previous approximation algorithms for treewidth [RS95, Lag96, Ree92, FHL08, Ami10, BDD$^+$16, FLS$^+$18, BF21] build the decomposition in a top-down manner, following [RS95]:



- Barrier at approximation ratio 3

Outline of the algorithm

By the self-reduction technique of [Bod96] we can focus on the following:

Input: An n-vertex graph G and a tree decomposition of G of width w

Output: A tree decomposition of G of width $< w$ or conclusion that $w \leq 2\text{tw}(G) + 1$

Time complexity: $2^{O(w)} n$

Outline of the algorithm

By the self-reduction technique of [Bod96] we can focus on the following:

Input: An n-vertex graph G and a tree decomposition of G of width w

Output: A tree decomposition of G of width $< w$ or conclusion that $w \leq 2\text{tw}(G) + 1$

Time complexity: $2^{O(w)}n$

Let T be the tree decomposition given in input and W its largest bag ($|W| = w + 1$).

Outline of the algorithm

By the self-reduction technique of [Bod96] we can focus on the following:

Input: An n-vertex graph G and a tree decomposition of G of width w

Output: A tree decomposition of G of width $< w$ or conclusion that $w \leq 2\text{tw}(G) + 1$

Time complexity: $2^{O(w)} n$

Let T be the tree decomposition given in input and W its largest bag ($|W| = w + 1$).

1. If $|W| > 2\text{tw}(G) + 2$ then T can be improved by a certain improvement operation

Outline of the algorithm

By the self-reduction technique of [Bod96] we can focus on the following:

Input: An n-vertex graph G and a tree decomposition of G of width w

Output: A tree decomposition of G of width $< w$ or conclusion that $w \leq 2\text{tw}(G) + 1$

Time complexity: $2^{O(w)} n$

Let T be the tree decomposition given in input and W its largest bag ($|W| = w + 1$).

1. If $|W| > 2\text{tw}(G) + 2$ then T can be improved by a certain improvement operation
   - Decreases the number of bags of size $|W|$ and does not increase the width of T

Outline of the algorithm

By the self-reduction technique of [Bod96] we can focus on the following:

Input: An n-vertex graph G and a tree decomposition of G of width w

Output: A tree decomposition of G of width $< w$ or conclusion that $w \leq 2\text{tw}(G) + 1$

Time complexity: $2^{O(w)}n$

Let T be the tree decomposition given in input and W its largest bag ($|W| = w + 1$).

1. If $|W| > 2\text{tw}(G) + 2$ then T can be improved by a certain improvement operation
   - Decreases the number of bags of size $|W|$ and does not increase the width of T
   - Inspired by a proof of Bellenbaum and Diestel on lean tree decompositions [BD02]

Outline of the algorithm

By the self-reduction technique of [Bod96] we can focus on the following:

Input: An n-vertex graph G and a tree decomposition of G of width w

Output: A tree decomposition of G of width $< w$ or conclusion that $w \leq 2\text{tw}(G) + 1$

Time complexity: $2^{O(w)}n$

Let T be the tree decomposition given in input and W its largest bag ($|W| = w + 1$).

1. If $|W| > 2\text{tw}(G) + 2$ then T can be improved by a certain improvement operation
   - Decreases the number of bags of size $|W|$ and does not increase the width of T
   - Inspired by a proof of Bellenbaum and Diestel on lean tree decompositions [BD02]
2. To improve width by one, $\Omega(n)$ improvement operations may be needed

Outline of the algorithm

By the self-reduction technique of [Bod96] we can focus on the following:

Input: An n-vertex graph G and a tree decomposition of G of width w

Output: A tree decomposition of G of width $< w$ or conclusion that $w \leq 2\text{tw}(G) + 1$

Time complexity: $2^{O(w)} n$

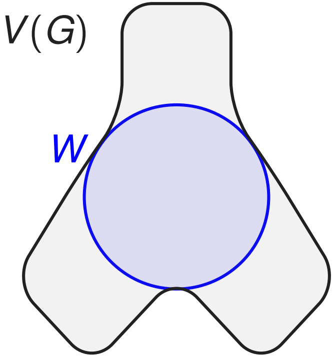
Let T be the tree decomposition given in input and W its largest bag ($|W| = w + 1$).

1. If $|W| > 2\text{tw}(G) + 2$ then T can be improved by a certain improvement operation
   - Decreases the number of bags of size $|W|$ and does not increase the width of T
   - Inspired by a proof of Bellenbaum and Diestel on lean tree decompositions [BD02]
2. To improve width by one, $\Omega(n)$ improvement operations may be needed
   - Efficient implementation by amortizatized analysis of the improvements and dynamic programming over the tree decomposition

Details of the algorithm

Splitting a bag

Let $W \subseteq V(G)$ be the largest bag and $|W| > 2\text{tw}(G) + 2$.

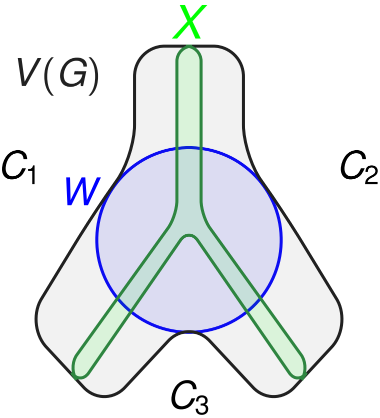


Splitting a bag

Let $W \subseteq V(G)$ be the largest bag and $|W| > 2\text{tw}(G) + 2$.

Lemma

There is a partition (C_1, C_2, C_3, X) of $V(G)$ with no edges between C_i and C_j for $i \neq j$ and $|(W \cap C_i) \cup X| < |W|$ for all $i \in \{1, 2, 3\}$.



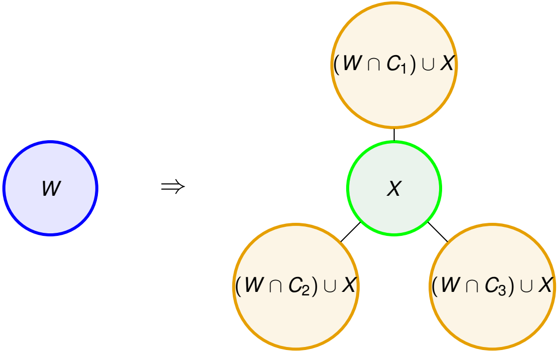
Splitting a bag

Let $W \subseteq V(G)$ be the largest bag and $|W| > 2\text{tw}(G) + 2$.

Lemma

There is a partition (C_1, C_2, C_3, X) of $V(G)$ with no edges between C_i and C_j for $i \neq j$ and $|(W \cap C_i) \cup X| < |W|$ for all $i \in \{1, 2, 3\}$.

Intuition: Now the following construction "locally improves" the tree decomposition



Splitting a bag

Let $W \subseteq V(G)$ be the largest bag and $|W| > 2\text{tw}(G) + 2$.

Lemma

There is a partition (C_1, C_2, C_3, X) of $V(G)$ with no edges between C_i and C_j for $i \neq j$ and $|(W \cap C_i) \cup X| < |W|$ for all $i \in \{1, 2, 3\}$.

Make the lemma into a definition:

Definition (Split)

A split of $W \subseteq V(G)$ is a partition (C_1, C_2, C_3, X) of $V(G)$ with no edges between C_i and C_j for $i \neq j$ and $|(W \cap C_i) \cup X| < |W|$ for all $i \in \{1, 2, 3\}$.

Splitting a bag

Let $W \subseteq V(G)$ be the largest bag and $|W| > 2\text{tw}(G) + 2$.

Lemma

There is a partition (C_1, C_2, C_3, X) of $V(G)$ with no edges between C_i and C_j for $i \neq j$ and $|(W \cap C_i) \cup X| < |W|$ for all $i \in \{1, 2, 3\}$.

Make the lemma into a definition:

Definition (Split)

A split of $W \subseteq V(G)$ is a partition (C_1, C_2, C_3, X) of $V(G)$ with no edges between C_i and C_j for $i \neq j$ and $|(W \cap C_i) \cup X| < |W|$ for all $i \in \{1, 2, 3\}$.

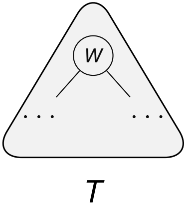
$\Rightarrow$

Lemma

Any set of vertices $W \subseteq V(G)$ of size $|W| > 2\text{tw}(G) + 2$ has a split.

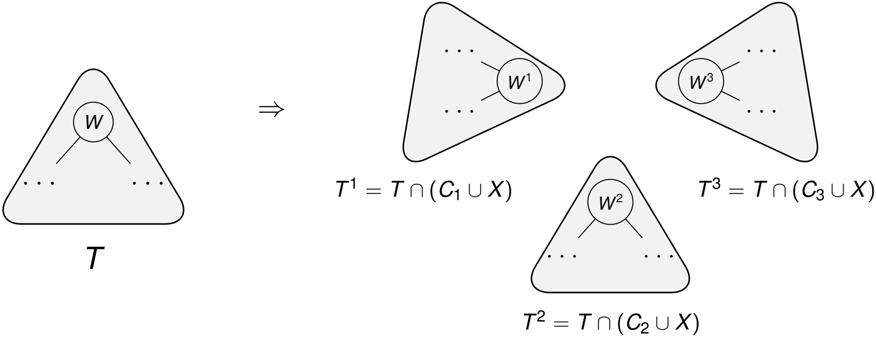
Improving a Tree Decomposition T

- Let W be the largest bag and (C_1, C_2, C_3, X) be a split of W



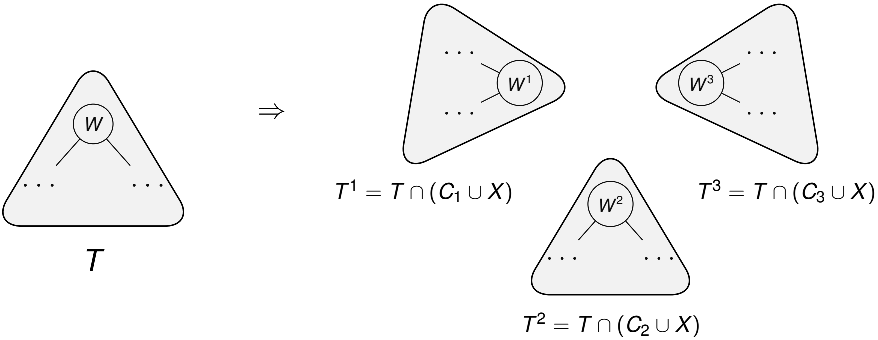
Improving a Tree Decomposition T

- Let W be the largest bag and (C_1, C_2, C_3, X) be a split of W
- For each $i \in \{1, 2, 3\}$, obtain a tree decomposition $T^i = T \cap (C_i \cup X)$ by setting $B^i = B \cap (C_i \cup X)$ for each bag B of T.



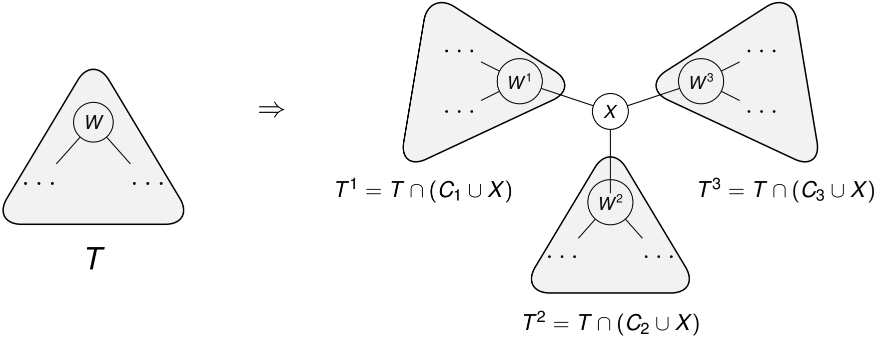
Improving a Tree Decomposition T

- Let W be the largest bag and (C_1, C_2, C_3, X) be a split of W
- For each $i \in \{1, 2, 3\}$, obtain a tree decomposition $T^i = T \cap (C_i \cup X)$ by setting $B^i = B \cap (C_i \cup X)$ for each bag B of T.
- T^i is a tree decomposition of $G[C_i \cup X]$



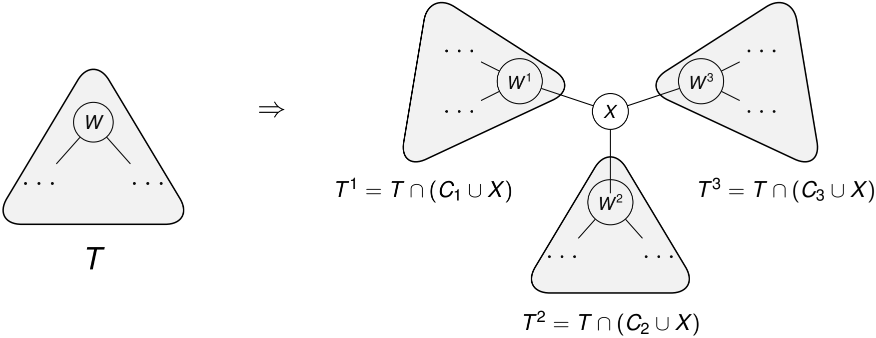
Improving a Tree Decomposition T

- Let W be the largest bag and (C_1, C_2, C_3, X) be a split of W
- For each $i \in \{1, 2, 3\}$, obtain a tree decomposition $T^i = T \cap (C_i \cup X)$ by setting $B^i = B \cap (C_i \cup X)$ for each bag B of T.
- T^i is a tree decomposition of $G[C_i \cup X]$
- The following is almost a tree decomposition of G:



Improving a Tree Decomposition T

- Let W be the largest bag and (C_1, C_2, C_3, X) be a split of W
- For each $i \in \{1, 2, 3\}$, obtain a tree decomposition $T^i = T \cap (C_i \cup X)$ by setting $B^i = B \cap (C_i \cup X)$ for each bag B of T.
- T^i is a tree decomposition of $G[C_i \cup X]$
- The following is almost a tree decomposition of G:



Except that vertices in X may violate the connectedness condition

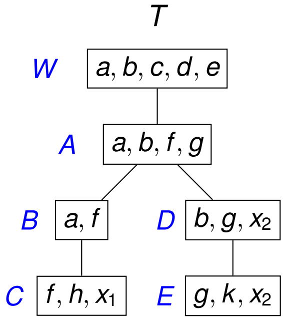
Fixing a tree decomposition

- Fix the connectedness condition by inserting vertices of X to bags

Fixing a tree decomposition

- Fix the connectedness condition by inserting vertices of X to bags

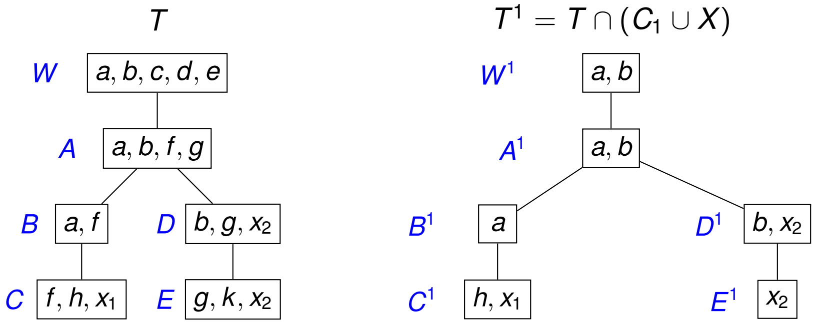
Example: Let $(C_1, C_2, C_3, X) = (\{a, b, h\}, \{c, d, f\}, \{e, g, k\}, \{x_1, x_2\})$ be a split of W:



Fixing a tree decomposition

- Fix the connectedness condition by inserting vertices of X to bags

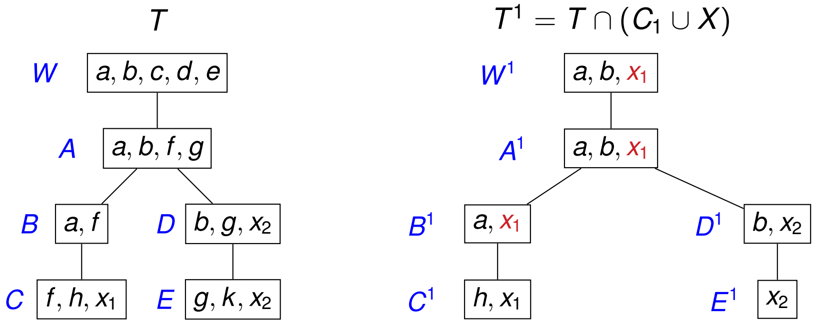
Example: Let $(C_1, C_2, C_3, X) = (\{a, b, h\}, \{c, d, f\}, \{e, g, k\}, \{x_1, x_2\})$ be a split of W:



Fixing a tree decomposition

- Fix the connectedness condition by inserting vertices of X to bags

Example: Let $(C_1, C_2, C_3, X) = (\{a, b, h\}, \{c, d, f\}, \{e, g, k\}, \{x_1, x_2\})$ be a split of W:



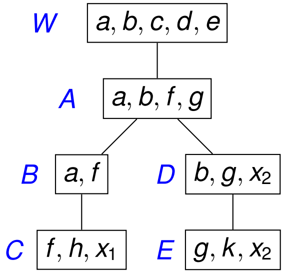
- Insert x_1 to B^1, A^1, and W^1

Fixing a tree decomposition

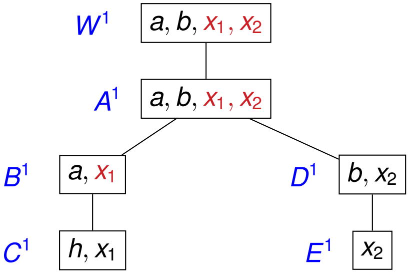
- Fix the connectedness condition by inserting vertices of X to bags

Example: Let $(C_1, C_2, C_3, X) = (\{a, b, h\}, \{c, d, f\}, \{e, g, k\}, \{x_1, x_2\})$ be a split of W:

T



$T^1 = T \cap (C_1 \cup X)$

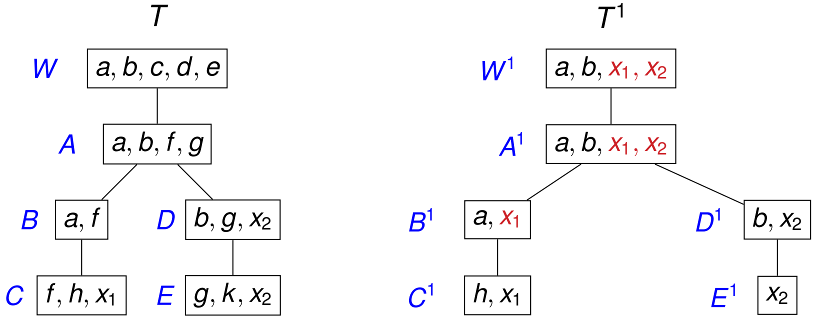


- Insert x_1 to B^1, A^1, and W^1
- Insert x_2 to A^1 and W^1

Fixing a tree decomposition

- Fix the connectedness condition by inserting vertices of X to bags

Example: Let $(C_1, C_2, C_3, X) = (\{a, b, h\}, \{c, d, f\}, \{e, g, k\}, \{x_1, x_2\})$ be a split of W:

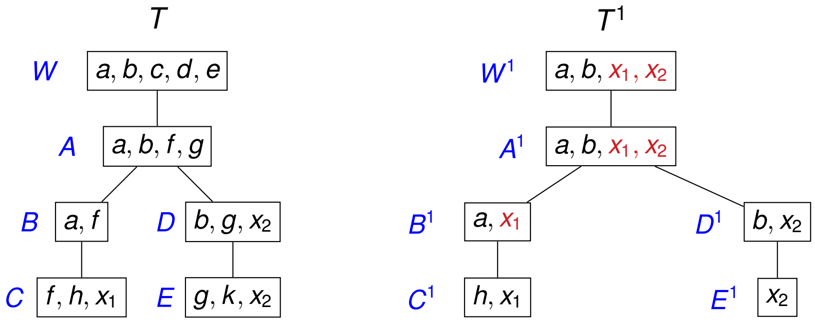


- Insert x_1 to B^1, A^1, and W^1
- Insert x_2 to A^1 and W^1
- Now $X \subseteq W^1$ and T^1 satisfies the connectedness condition

Fixing a tree decomposition

- Fix the connectedness condition by inserting vertices of X to bags

Example: Let $(C_1, C_2, C_3, X) = (\{a, b, h\}, \{c, d, f\}, \{e, g, k\}, \{x_1, x_2\})$ be a split of W:

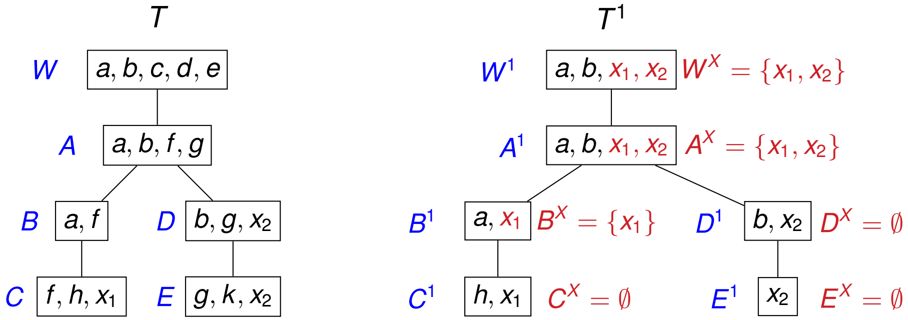


- Insert x_1 to B^1, A^1, and W^1
- Insert x_2 to A^1 and W^1
- Now $X \subseteq W^1$ and T^1 satisfies the connectedness condition

$\Rightarrow$ The construction with T^1, T^2, and T^3 satisfies the connectedness condition

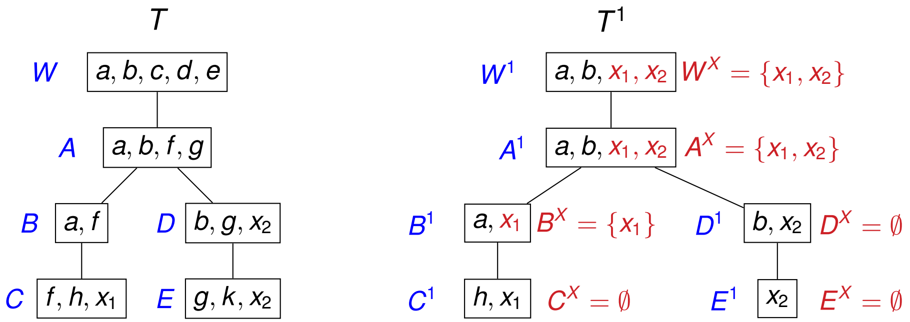
Fixing a tree decomposition: Analysis

- Now the bags of T^i are of form $B^i = (B \cap (C_i \cup X)) \cup B^X$, where $B^X \subseteq X$ are the inserted vertices



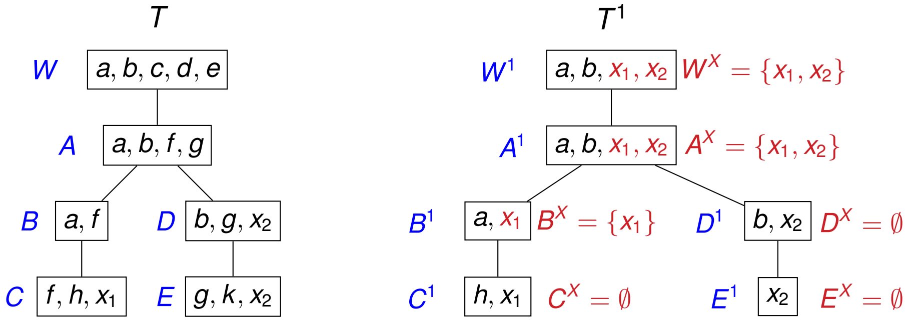
Fixing a tree decomposition: Analysis

- Now the bags of T^i are of form $B^i = (B \cap (C_i \cup X)) \cup B^X$, where $B^X \subseteq X$ are the inserted vertices
- For the root, $|W^i| < |W|$ by the definition of a split



Fixing a tree decomposition: Analysis

- Now the bags of T^i are of form $B^i = (B \cap (C_i \cup X)) \cup B^X$, where $B^X \subseteq X$ are the inserted vertices
- For the root, $|W^i| < |W|$ by the definition of a split
- Goal: Show that $|B^i| \leq |B|$ for all bags B



The main proof

Definition (Minimum split)

A split (C_1, C_2, C_3, X) of W is a *minimum split* if it minimizes $|X|$ among all splits of W

The main proof

Definition (Minimum split)

A split (C_1, C_2, C_3, X) of W is a *minimum split* if it minimizes $|X|$ among all splits of W

Lemma

If (C_1, C_2, C_3, X) is a minimum split, then $|B^i| \leq |B|$ for all bags B and all i

The main proof

Definition (Minimum split)

A split (C_1, C_2, C_3, X) of W is a *minimum split* if it minimizes $|X|$ among all splits of W

Lemma

If (C_1, C_2, C_3, X) is a minimum split, then $|B^i| \leq |B|$ for all bags B and all i

- Suppose $i = 1$ and $|B^1| > |B|$

The main proof

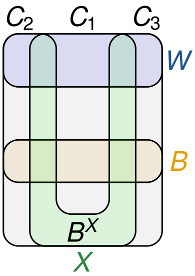
Definition (Minimum split)

A split (C_1, C_2, C_3, X) of W is a *minimum split* if it minimizes $|X|$ among all splits of W

Lemma

If (C_1, C_2, C_3, X) is a minimum split, then $|B^i| \leq |B|$ for all bags B and all i

- Suppose $i = 1$ and $|B^1| > |B|$
- Note that $B^1 = (B \cap (C_1 \cup X)) \cup B^X = (B \setminus (C_2 \cup C_3)) \cup B^X$



The main proof

Definition (Minimum split)

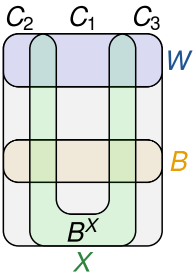
A split (C_1, C_2, C_3, X) of W is a *minimum split* if it minimizes $|X|$ among all splits of W

Lemma

If (C_1, C_2, C_3, X) is a minimum split, then $|B^i| \leq |B|$ for all bags B and all i

- Suppose $i = 1$ and $|B^1| > |B|$
- Note that $B^1 = (B \cap (C_1 \cup X)) \cup B^X = (B \setminus (C_2 \cup C_3)) \cup B^X$

$\Rightarrow |B^X| > |B \cap (C_2 \cup C_3)|$



The main proof

Definition (Minimum split)

A split (C_1, C_2, C_3, X) of W is a *minimum split* if it minimizes $|X|$ among all splits of W

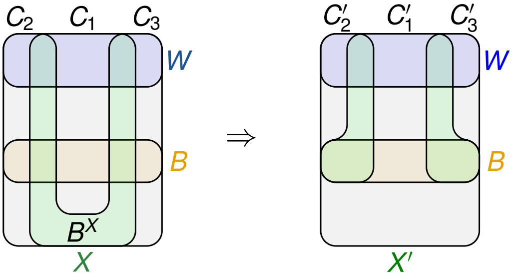
Lemma

If (C_1, C_2, C_3, X) is a minimum split, then $|B^i| \leq |B|$ for all bags B and all i

- Suppose $i = 1$ and $|B^1| > |B|$
- Note that $B^1 = (B \cap (C_1 \cup X)) \cup B^X = (B \setminus (C_2 \cup C_3)) \cup B^X$

$\Rightarrow |B^X| > |B \cap (C_2 \cup C_3)|$

- Take a split with $X' = (X \setminus B^X) \cup (B \cap (C_2 \cup C_3))$



The main proof

Definition (Minimum split)

A split (C_1, C_2, C_3, X) of W is a *minimum split* if it minimizes $|X|$ among all splits of W

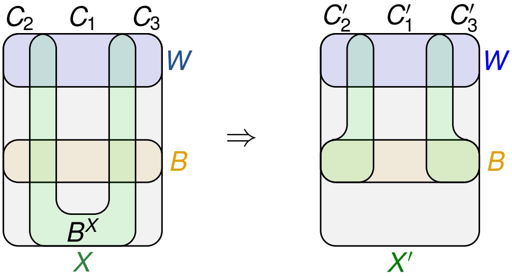
Lemma

If (C_1, C_2, C_3, X) is a minimum split, then $|B^i| \leq |B|$ for all bags B and all i

- Suppose $i = 1$ and $|B^1| > |B|$
- Note that $B^1 = (B \cap (C_1 \cup X)) \cup B^X = (B \setminus (C_2 \cup C_3)) \cup B^X$

$\Rightarrow$ $|B^X| > |B \cap (C_2 \cup C_3)|$

- Take a split with $X' = (X \setminus B^X) \cup (B \cap (C_2 \cup C_3))$
- $|X'| < |X|$ so this contradicts the minimality



Stronger minimum split

Previous definition of minimum splits not sufficient!

Stronger minimum split

Previous definition of minimum splits not sufficient!

Definition (Minimum split, the real definition)

A split (C_1, C_2, C_3, X) of W is a *minimum split* if

1. it minimizes $|X|$ among all splits of W
2. subject to (1), minimizes $\sum_{x \in X} d_T(x)$

Where $d_T(x)$ is the distance in T from the root to the closest bag containing x

Stronger minimum split

Previous definition of minimum splits not sufficient!

Definition (Minimum split, the real definition)

A split (C_1, C_2, C_3, X) of W is a *minimum split* if

1. it minimizes $|X|$ among all splits of W
2. subject to (1), minimizes $\sum_{x \in X} d_T(x)$

Where $d_T(x)$ is the distance in T from the root to the closest bag containing x

Lemma

Let (C_1, C_2, C_3, X) be a minimum split of W. For all bags B either $B^X = \emptyset$ or $|B^i| < |B|$

Stronger minimum split

Previous definition of minimum splits not sufficient!

Definition (Minimum split, the real definition)

A split (C_1, C_2, C_3, X) of W is a *minimum split* if

1. it minimizes $|X|$ among all splits of W
2. subject to (1), minimizes $\sum_{x \in X} d_T(x)$

Where $d_T(x)$ is the distance in T from the root to the closest bag containing x

Lemma

Let (C_1, C_2, C_3, X) be a minimum split of W. For all bags B either $B^X = \emptyset$ or $|B^i| < |B|$

The proof uses similar argument as earlier

Stronger minimum split

Previous definition of minimum splits not sufficient!

Definition (Minimum split, the real definition)

A split (C_1, C_2, C_3, X) of W is a *minimum split* if

1. it minimizes $|X|$ among all splits of W
2. subject to (1), minimizes $\sum_{x \in X} d_T(x)$

Where $d_T(x)$ is the distance in T from the root to the closest bag containing x

Lemma

Let (C_1, C_2, C_3, X) be a minimum split of W. For all bags B either $B^X = \emptyset$ or $|B^i| < |B|$

The proof uses similar argument as earlier

- Now, for each bag B, either of the following happens:

Stronger minimum split

Previous definition of minimum splits not sufficient!

Definition (Minimum split, the real definition)

A split (C_1, C_2, C_3, X) of W is a *minimum split* if

1. it minimizes $|X|$ among all splits of W
2. subject to (1), minimizes $\sum_{x \in X} d_T(x)$

Where $d_T(x)$ is the distance in T from the root to the closest bag containing x

Lemma

Let (C_1, C_2, C_3, X) be a minimum split of W. For all bags B either $B^X = \emptyset$ or $|B^i| < |B|$

The proof uses similar argument as earlier

- Now, for each bag B, either of the following happens:
  1. $|B^i| < |B|$ holds for all i

Stronger minimum split

Previous definition of minimum splits not sufficient!

Definition (Minimum split, the real definition)

A split (C_1, C_2, C_3, X) of W is a *minimum split* if

1. it minimizes $|X|$ among all splits of W
2. subject to (1), minimizes $\sum_{x \in X} d_T(x)$

Where $d_T(x)$ is the distance in T from the root to the closest bag containing x

Lemma

Let (C_1, C_2, C_3, X) be a minimum split of W. For all bags B either $B^X = \emptyset$ or $|B^i| < |B|$

The proof uses similar argument as earlier

- Now, for each bag B, either of the following happens:
  1. $|B^i| < |B|$ holds for all i
  2. $B^X = \emptyset$ and $B \subseteq C_i \cup X$, implying $B^i = B$ and $B^j \subseteq X$ for $j \neq i$

Stronger minimum split

Previous definition of minimum splits not sufficient!

Definition (Minimum split, the real definition)

A split (C_1, C_2, C_3, X) of W is a *minimum split* if

1. it minimizes $|X|$ among all splits of W
2. subject to (1), minimizes $\sum_{x \in X} d_T(x)$

Where $d_T(x)$ is the distance in T from the root to the closest bag containing x

Lemma

Let (C_1, C_2, C_3, X) be a minimum split of W. For all bags B either $B^X = \emptyset$ or $|B^i| < |B|$

The proof uses similar argument as earlier

- Now, for each bag B, either of the following happens:
  1. $|B^i| < |B|$ holds for all i
  2. $B^X = \emptyset$ and $B \subseteq C_i \cup X$, implying $B^i = B$ and $B^j \subseteq X$ for $j \neq i$
- For the root bag W, $|W^i| < |W|$ by definition of split

Stronger minimum split

Previous definition of minimum splits not sufficient!

Definition (Minimum split, the real definition)

A split (C_1, C_2, C_3, X) of W is a *minimum split* if

1. it minimizes $|X|$ among all splits of W
2. subject to (1), minimizes $\sum_{x \in X} d_T(x)$

Where $d_T(x)$ is the distance in T from the root to the closest bag containing x

Lemma

Let (C_1, C_2, C_3, X) be a minimum split of W. For all bags B either $B^X = \emptyset$ or $|B^i| < |B|$

The proof uses similar argument as earlier

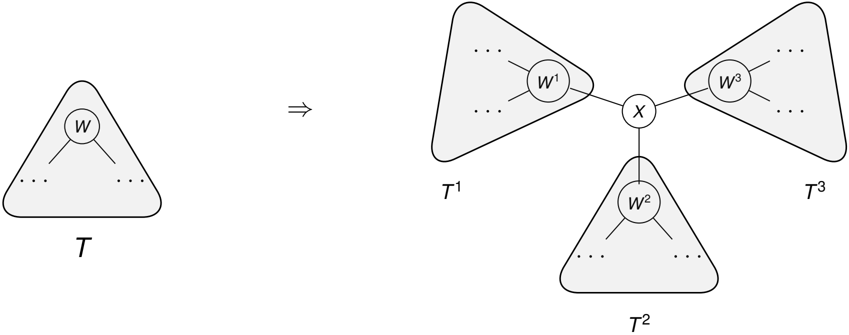
- Now, for each bag B, either of the following happens:
  1. $|B^i| < |B|$ holds for all i
  2. $B^X = \emptyset$ and $B \subseteq C_i \cup X$, implying $B^i = B$ and $B^j \subseteq X$ for $j \neq i$
- For the root bag W, $|W^i| < |W|$ by definition of split

$\Rightarrow$ The number of bags of size $|W|$ decreases and width does not increase

Recap

Now we have an algorithm:

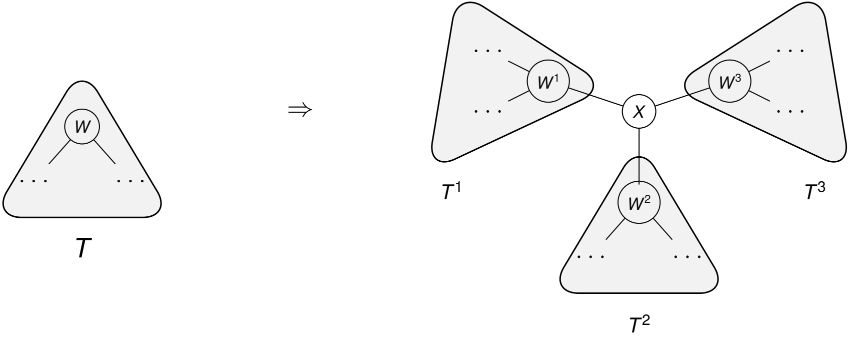
1. Take the largest bag W of T



Recap

Now we have an algorithm:

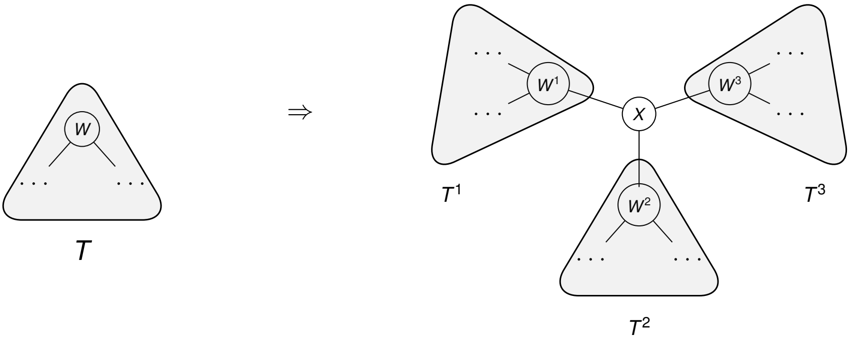
1. Take the largest bag W of T
2. Test if there exists a split (C_1, C_2, C_3, X) of W



Recap

Now we have an algorithm:

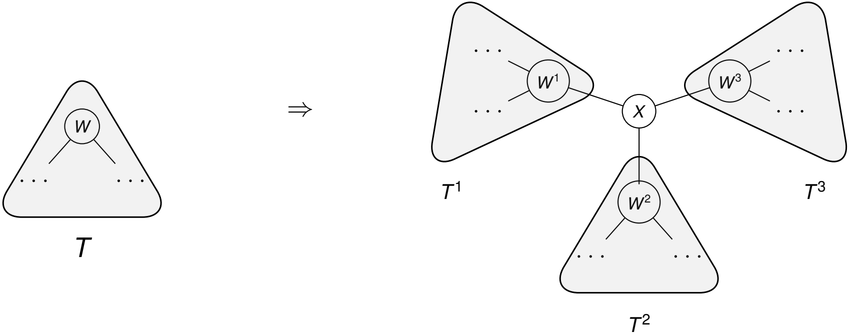
1. Take the largest bag W of T
2. Test if there exists a split (C_1, C_2, C_3, X) of W
   2.1 Yes $\Rightarrow$ use a minimum split to apply the improvement operation on T



Recap

Now we have an algorithm:

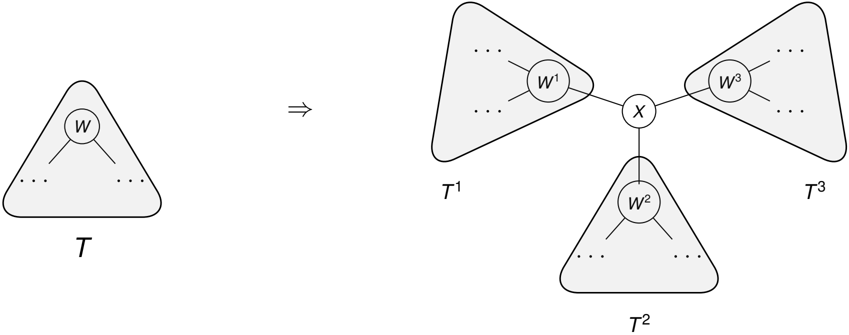
1. Take the largest bag W of T
2. Test if there exists a split (C_1, C_2, C_3, X) of W
   2.1 Yes $\Rightarrow$ use a minimum split to apply the improvement operation on T
   2.2 No $\Rightarrow$ return that T has width $\leq 2\text{tw}(G) + 1$



Recap

Now we have an algorithm:

1. Take the largest bag W of T
2. Test if there exists a split (C_1, C_2, C_3, X) of W
   2.1 Yes $\Rightarrow$ use a minimum split to apply the improvement operation on T
   2.2 No $\Rightarrow$ return that T has width $\leq 2\text{tw}(G) + 1$
3. Repeat (1-2) until the width has decreased, taking at most n iterations

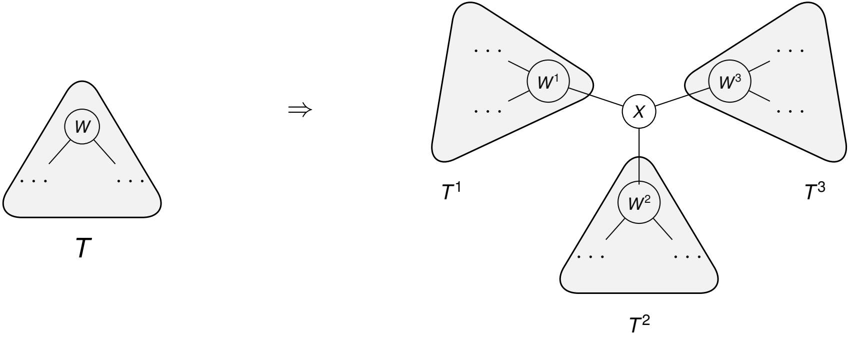


Recap

Now we have an algorithm:

1. Take the largest bag W of T
2. Test if there exists a split (C_1, C_2, C_3, X) of W
   - 2.1 Yes $\Rightarrow$ use a minimum split to apply the improvement operation on T
   - 2.2 No $\Rightarrow$ return that T has width $\leq 2\text{tw}(G) + 1$
3. Repeat (1-2) until the width has decreased, taking at most n iterations

With standard techniques, total time complexity $2^{\mathcal{O}(k)} n^2$



Optimizing to linear time

Recall: Two types of bags B:

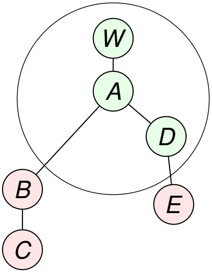
1. $|B^i| < |B|$ for all i
2. $B^X = \emptyset$ and $B \subseteq C_i \cup X$ for some i

Optimizing to linear time

Recall: Two types of bags B:

1. $|B^i| < |B|$ for all i
2. $B^X = \emptyset$ and $B \subseteq C_i \cup X$ for some i

- Bags of type 1 form a connected subtree around root W

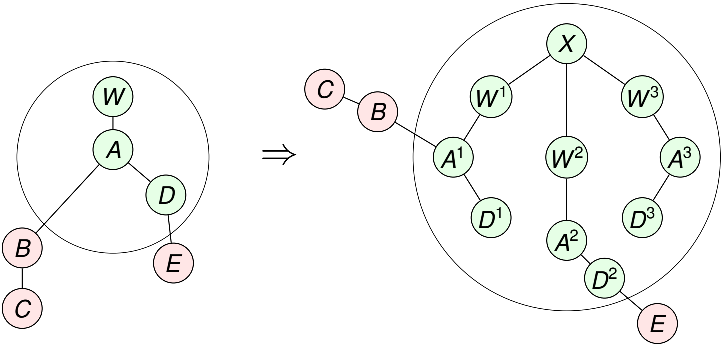


Optimizing to linear time

Recall: Two types of bags B:

1. $|B^i| < |B|$ for all i
2. $B^X = \emptyset$ and $B \subseteq C_i \cup X$ for some i

- Bags of type 1 form a connected subtree around root W
- If B is type 2 with $B \subseteq C_i \cup X$, then $B^i = B$ and $B^j \subseteq X$, and the same holds for all bags below it

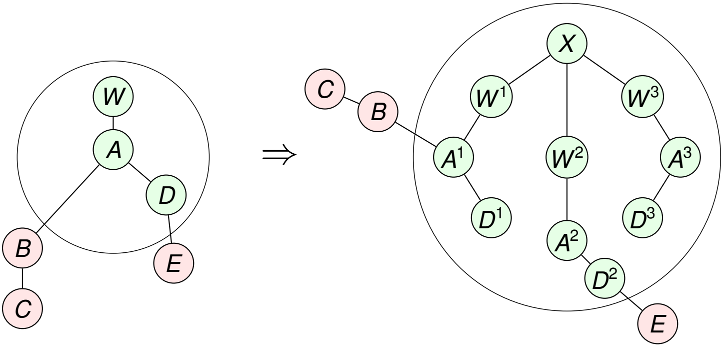


Optimizing to linear time

Recall: Two types of bags B:

1. $|B^i| < |B|$ for all i
2. $B^X = \emptyset$ and $B \subseteq C_i \cup X$ for some i

- Bags of type 1 form a connected subtree around root W
- If B is type 2 with $B \subseteq C_i \cup X$, then $B^i = B$ and $B^j \subseteq X$, and the same holds for all bags below it
  - Drop the copies $B^j \subseteq X$

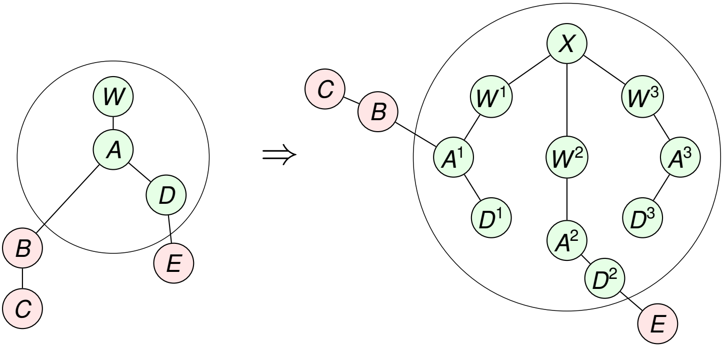


Optimizing to linear time

Recall: Two types of bags B:

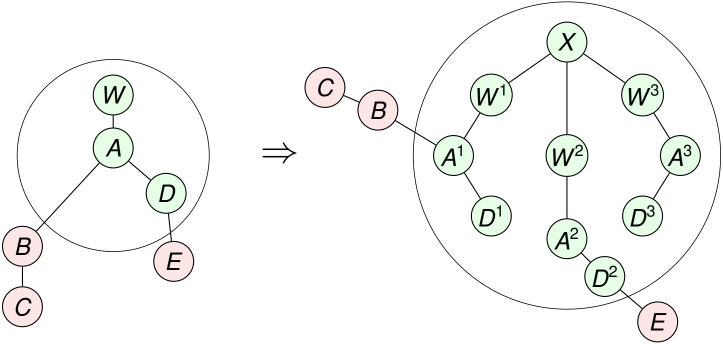
1. $|B^i| < |B|$ for all i
2. $B^X = \emptyset$ and $B \subseteq C_i \cup X$ for some i

- Bags of type 1 form a connected subtree around root W
- If B is type 2 with $B \subseteq C_i \cup X$, then $B^i = B$ and $B^j \subseteq X$, and the same holds for all bags below it
  - Drop the copies $B^j \subseteq X$
  - Only one copy $B^i = B$ in the resulting decomposition



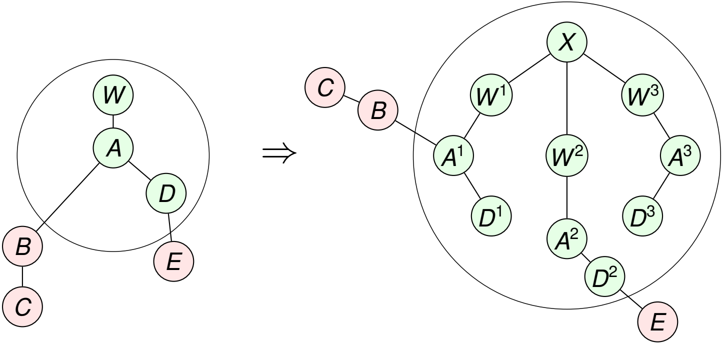
Optimizing to linear time

- Main idea: Do work only for the bags of type 1



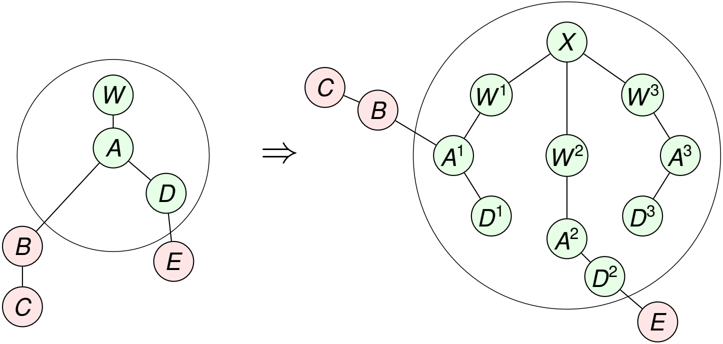
Optimizing to linear time

- Main idea: Do work only for the bags of type 1
- Because $|B^i| < |B|$ for each $i \in \{1, 2, 3\}$ for these bags, this amortizes to $2^{\mathcal{O}(k)} n$



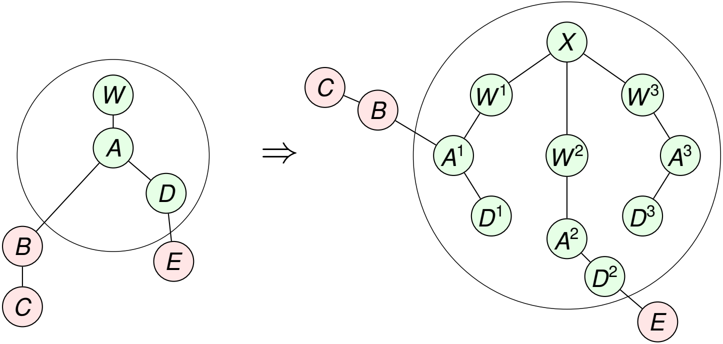
Optimizing to linear time

- Main idea: Do work only for the bags of type 1
- Because $|B^i| < |B|$ for each $i \in \{1, 2, 3\}$ for these bags, this amortizes to $2^{\mathcal{O}(k)} n$
- Maintain dynamic programming that finds the split, recompute only for bags of type 1



Optimizing to linear time

- Main idea: Do work only for the bags of type 1
- Because $|B^i| < |B|$ for each $i \in \{1, 2, 3\}$ for these bags, this amortizes to $2^{\mathcal{O}(k)} n$
- Maintain dynamic programming that finds the split, recompute only for bags of type 1
- Go trough the decomposition with DFS while maintaining the dynamic programming by "rerooting operations"



Conclusion

- A $2^{\mathcal{O}(k)} n$ time 2-approximation algorithm for treewidth

Conclusion

- A $2^{\mathcal{O}(k)} n$ time 2-approximation algorithm for treewidth
- A new approach for width parameters that has already been extended to rankwidth and branchwidth

Conclusion

- A $2^{\mathcal{O}(k)}n$ time 2-approximation algorithm for treewidth
- A new approach for width parameters that has already been extended to rankwidth and branchwidth
- Open problems:

Conclusion

- A $2^{\mathcal{O}(k)}n$ time 2-approximation algorithm for treewidth
- A new approach for width parameters that has already been extended to rankwidth and branchwidth
- Open problems:
  - Classic: Is there a $2^{o(k^3)}n^{\mathcal{O}(1)}$ time exact algorithm for treewidth?

Conclusion

- A $2^{\mathcal{O}(k)}n$ time 2-approximation algorithm for treewidth
- A new approach for width parameters that has already been extended to rankwidth and branchwidth
- Open problems:
  - Classic: Is there a $2^{o(k^3)}n^{\mathcal{O}(1)}$ time exact algorithm for treewidth?
  - Is the approximation ratio 2 the best possible for the approach of this work?

Bibliography

Eyal Amir.
Approximation algorithms for treewidth.
Algorithmica, 56(4):448–479, 2010.

Stefan Arnborg and Andrzej Proskurowski.
Linear time algorithms for NP-hard problems restricted to partial k-trees.
Discret. Appl. Math., 23(1):11–24, 1989.

Umberto Bertelè and Francesco Brioschi.
On non-serial dynamic programming.
J. Comb. Theory, Ser. A, 14(2):137–148, 1973.

Patrick Bellenbaum and Reinhard Diestel.
Two short proofs concerning tree-decompositions.
Comb. Probab. Comput., 11(6):541–547, 2002.

Hans L. Bodlaender, Pål Grønås Drange, Markus S. Dregi, Fedor V. Fomin, Daniel Lokshtanov, and Michal Pilipczuk.
A c^k n 5-approximation algorithm for treewidth.
SIAM J. Comput., 45(2):317–378, 2016.

Mahdi Belbasi and Martin Fürer.
An improvement of reed's treewidth approximation.
In *International Workshop on Algorithms and Computation*, pages 166–181. Springer, 2021.

Hans L. Bodlaender.
A linear-time algorithm for finding tree-decompositions of small treewidth.
SIAM J. Comput., 25(6):1305–1317, 1996.

Michael Elberfeld, Andreas Jakoby, and Till Tantau.
Logspace versions of the theorems of bodlaender and courcelle.
In *Proceedings of the 51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010*, pages 143–152. IEEE Computer Society, 2010.

Uriel Feige, MohammadTaghi Hajiaghayi, and James R. Lee.
Improved approximation algorithms for minimum weight vertex separators.
SIAM J. Comput., 38(2):629–657, 2008.