Tutorial: New algorithms for computing treewidth

Tuukka Korhonen

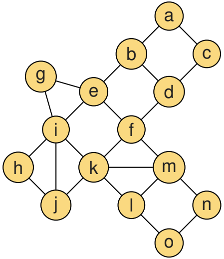
UNIVERSITY OF BERGEN

STWOR

30 September 2023

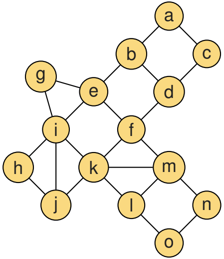
Treewidth

- Measures how close a graph is to a tree



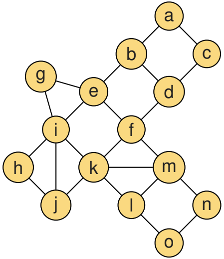
Treewidth

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



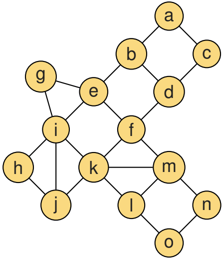
Treewidth

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



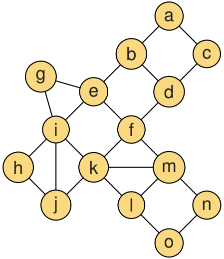
Treewidth

- 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



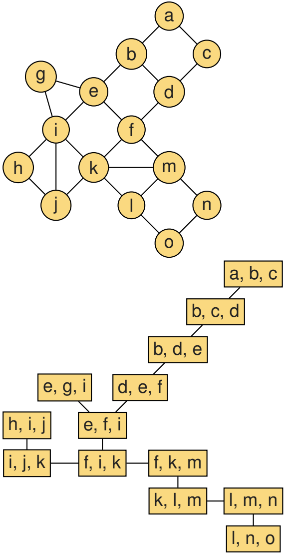
Treewidth

- 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
  - K_n has treewidth $n-1$



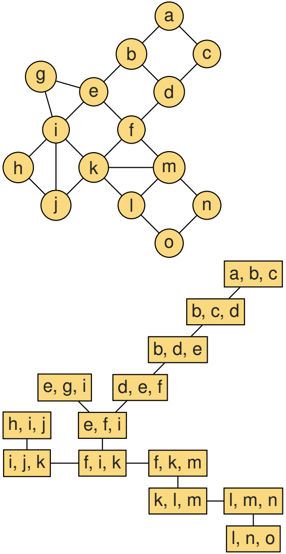
Treewidth

- 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
  - K_n has treewidth $n - 1$
- Treewidth = minimum width of a tree decomposition



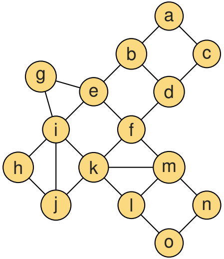
Treewidth

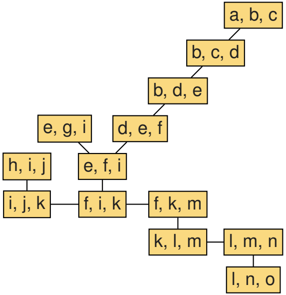
- Measures how close a graph is to a tree
  - Trees have treewidth 1
  - The example graph has treewidth 2
  - The $n \times n$-grid has treewidth n
  - K_n has treewidth $n - 1$
- Treewidth = minimum width of a tree decomposition
- Tree decomposition is a tree of bags so that:



Treewidth

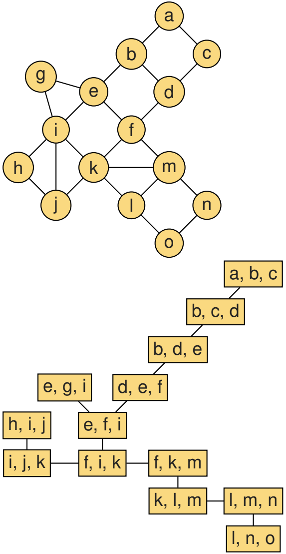
- Measures how close a graph is to a tree
  - Trees have treewidth 1
  - The example graph has treewidth 2
  - The $n \times n$-grid has treewidth n
  - K_n has treewidth $n - 1$
- Treewidth = minimum width of a tree decomposition
- Tree decomposition is a tree of bags so that:
  1. every vertex is in some bag





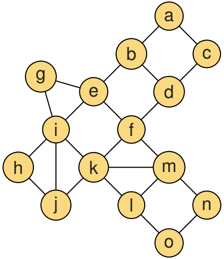
Treewidth

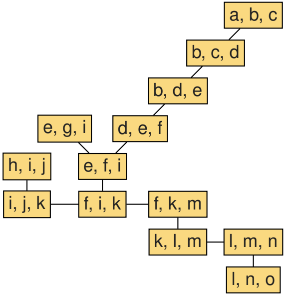
- Measures how close a graph is to a tree
  - Trees have treewidth 1
  - The example graph has treewidth 2
  - The $n \times n$-grid has treewidth n
  - K_n has treewidth $n-1$
- Treewidth = minimum width of a tree decomposition
- Tree decomposition is a tree of bags so that:
  1. every vertex is in some bag
  2. every edge is in some bag



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
  - K_n has treewidth $n - 1$
- Treewidth = minimum width of a tree decomposition
- Tree decomposition is a tree of bags so that:
  1. every vertex is in some bag
  2. every edge is in some bag
  3. bags containing a vertex form a connected subtree





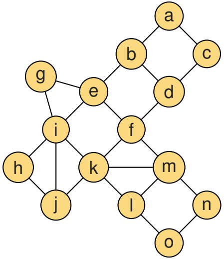
Treewidth

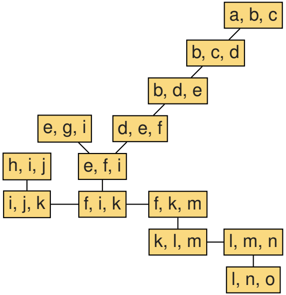
- Measures how close a graph is to a tree
  - Trees have treewidth 1
  - The example graph has treewidth 2
  - The $n \times n$-grid has treewidth n
  - K_n has treewidth $n - 1$

- Treewidth = minimum width of a tree decomposition

- Tree decomposition is a tree of bags so that:
  1. every vertex is in some bag
  2. every edge is in some bag
  3. bags containing a vertex form a connected subtree

- Width = max bag size -1



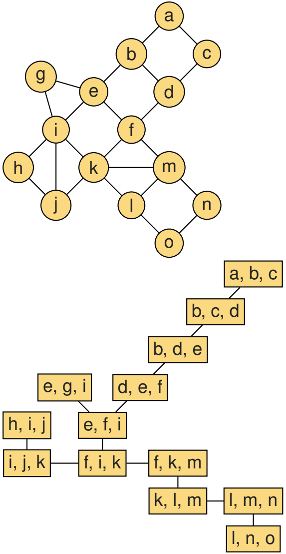


Width 2

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
  - K_n has treewidth $n - 1$
- Treewidth = minimum width of a tree decomposition
- Tree decomposition is a tree of bags so that:
  1. every vertex is in some bag
  2. every edge is in some bag
  3. bags containing a vertex form a connected subtree
- Width = max bag size -1

[Robertson & Seymour '84, Arnborg & Proskurowski '89, Bertele & Brioschi '72, Halin '76]



Width 2

Computing treewidth

Computing treewidth: Classical results

Computing treewidth: Classical results

Theorem (Robertson & Seymour, Graph minors XIII, '86)

There is a $2^{\mathcal{O}(k)}n^2$ time 4-approximation algorithm for treewidth.

Computing treewidth: Classical results

Theorem (Robertson & Seymour, Graph minors XIII, '86)

There is a $2^{\mathcal{O}(k)} n^2$ time 4-approximation algorithm for treewidth.

Theorem (Bodlaender '93)

There is a $2^{\mathcal{O}(k^3)} n$ time algorithm for treewidth.

Computing treewidth: Classical results

Theorem (Robertson & Seymour, Graph minors XIII, '86)

There is a $2^{\mathcal{O}(k)} n^2$ time 4-approximation algorithm for treewidth.

Theorem (Bodlaender '93)

There is a $2^{\mathcal{O}(k^3)} n$ time algorithm for treewidth.

Using dynamic programming of [Bodlaender & Kloks '91]

Computing treewidth: Classical results

Theorem (Robertson & Seymour, Graph minors XIII, '86)

There is a $2^{\mathcal{O}(k)} n^2$ time 4-approximation algorithm for treewidth.

Theorem (Bodlaender '93)

There is a $2^{\mathcal{O}(k^3)} n$ time algorithm for treewidth.

Using dynamic programming of [Bodlaender & Kloks '91]

Theorem (Bodlaender, Drange, Dregi, Fomin, Lokshtanov, & Pilipczuk '13)

There is a $2^{\mathcal{O}(k)} n$ time 5-approximation for treewidth.

Computing treewidth: Classical results

Theorem (Robertson & Seymour, Graph minors XIII, '86)

There is a $2^{\mathcal{O}(k)} n^2$ time 4-approximation algorithm for treewidth.

Theorem (Bodlaender '93)

There is a $2^{\mathcal{O}(k^3)} n$ time algorithm for treewidth.

Using dynamic programming of [Bodlaender & Kloks '91]

Theorem (Bodlaender, Drange, Dregi, Fomin, Lokshtanov, & Pilipczuk '13)

There is a $2^{\mathcal{O}(k)} n$ time 5-approximation for treewidth.

Builds on both [Robertson-Seymour'86] and [Bodlaender'93]

Computing treewidth: Classical results

Theorem (Robertson & Seymour, Graph minors XIII, '86)

There is a $2^{\mathcal{O}(k)} n^2$ time 4-approximation algorithm for treewidth.

Theorem (Bodlaender '93)

There is a $2^{\mathcal{O}(k^3)} n$ time algorithm for treewidth.

Using dynamic programming of [Bodlaender & Kloks '91]

Theorem (Bodlaender, Drange, Dregi, Fomin, Lokshtanov, & Pilipczuk '13)

There is a $2^{\mathcal{O}(k)} n$ time 5-approximation for treewidth.

Builds on both [Robertson-Seymour'86] and [Bodlaender'93]

Many more: [ACP'87,MT'91,Lagergren'96,Reed'92,Amir'10,FHL'08,FTV'15,FLS'18,BF'21,BF'22]

Computing treewidth: New results

Computing treewidth: New results

Theorem (K. '21)

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

Computing treewidth: New results

Theorem (K. '21)

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

Compare to: $2^{\mathcal{O}(k)} n$ time 5-approximation of [BDDFLP '13]

Computing treewidth: New results

Theorem (K. '21)

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

Compare to: $2^{\mathcal{O}(k)} n$ time 5-approximation of [BDDFLP '13]

- Breaks the 3-approximation barrier of Robertson-Seymour-type algorithms

Computing treewidth: New results

Theorem (K. '21)

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

Compare to: $2^{\mathcal{O}(k)} n$ time 5-approximation of [BDDFLP '13]

- Breaks the 3-approximation barrier of Robertson-Seymour-type algorithms
- Improves the $2^{\mathcal{O}(k)}$ from $\approx 2^{40k}$ to 2^{11k}

Computing treewidth: New results

Theorem (K. '21)

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

Compare to: $2^{\mathcal{O}(k)} n$ time 5-approximation of [BDDFLP '13]

- Breaks the 3-approximation barrier of Robertson-Seymour-type algorithms
- Improves the $2^{\mathcal{O}(k)}$ from $\approx 2^{40k}$ to 2^{11k}

Theorem (K. & Lokshtanov '23)

There is a $2^{\mathcal{O}(k^2)} n^4$ time algorithm for treewidth.

Computing treewidth: New results

Theorem (K. '21)

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

Compare to: $2^{\mathcal{O}(k)}n$ time 5-approximation of [BDDFLP '13]

- Breaks the 3-approximation barrier of Robertson-Seymour-type algorithms
- Improves the $2^{\mathcal{O}(k)}$ from $\approx 2^{40k}$ to 2^{11k}

Theorem (K. & Lokshtanov '23)

There is a $2^{\mathcal{O}(k^2)}n^4$ time algorithm for treewidth.

Compare to: $2^{\mathcal{O}(k^3)}n$ time algorithm of [Bodlaender'93]

New method: Local improvement

- In [K.'21],[K.&Lokshtanov'23] new method for treewidth: **Local improvement**

New method: Local improvement

- In [K.'21],[K.&Lokshtanov'23] new method for treewidth: **Local improvement**
- Repeatedly re-arrange the tree decomposition to make the largest bag smaller

New method: Local improvement

- In [K.'21],[K.&Lokshtanov'23] new method for treewidth: **Local improvement**
- Repeatedly re-arrange the tree decomposition to make the largest bag smaller
- New ideas both in the graph-theoretic part of the re-arrangement and in the efficient implementation, with further applications:

New method: Local improvement

- In [K.'21],[K.&Lokshtanov'23] new method for treewidth: **Local improvement**
- Repeatedly re-arrange the tree decomposition to make the largest bag smaller
- New ideas both in the graph-theoretic part of the re-arrangement and in the efficient implementation, with further applications:

Theorem (K., Majewski, Nadara, Pilipczuk & Sokołowski '23)

There is a data structure for maintaining a tree decomposition of width $\mathcal{O}(k)$ for a fully dynamic graph of treewidth $\leq k$ with amortized update time $f(k) \cdot n^{o(1)}$.

New method: Local improvement

- In [K.'21],[K.&Lokshtanov'23] new method for treewidth: **Local improvement**
- Repeatedly re-arrange the tree decomposition to make the largest bag smaller
- New ideas both in the graph-theoretic part of the re-arrangement and in the efficient implementation, with further applications:

Theorem (K., Majewski, Nadara, Pilipczuk & Sokołowski '23)

There is a data structure for maintaining a tree decomposition of width $\mathcal{O}(k)$ for a fully dynamic graph of treewidth $\leq k$ with amortized update time $f(k) \cdot n^{o(1)}$.

(first non-trivial algorithm in this setting for $k \geq 3$)

Plan

Plan:

Plan

Plan:

1. Local improvement for FPT exact treewidth (joint work with Daniel Lokshtanov)

Plan

Plan:

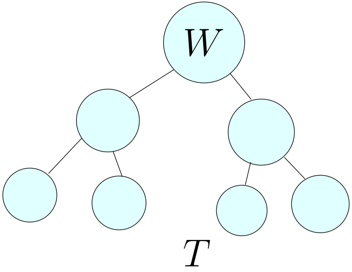
1. Local improvement for FPT exact treewidth (joint work with Daniel Lokshtanov)
2. Local improvement in dynamic treewidth (joint work with Konrad Majewski, Wojciech Nadara, Michał Pilipczuk & Marek Sokołowski)

Local improvement for FPT exact treewidth

(joint work with Daniel Lokshtanov)

Setting

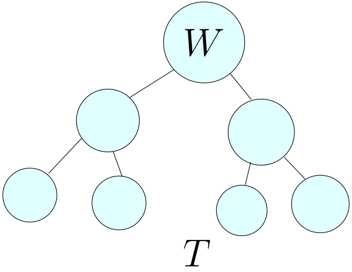
We have a tree decomposition T whose largest bag is W



Setting

We have a tree decomposition T whose largest bag is W

Goal:

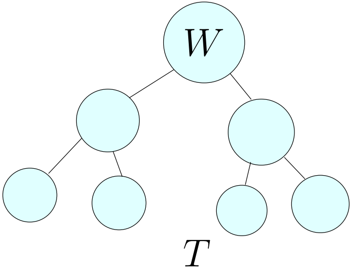


Setting

We have a tree decomposition T whose largest bag is W

Goal:

1. either decrease the number of bags of size $|W|$ while not increasing the width of T, or

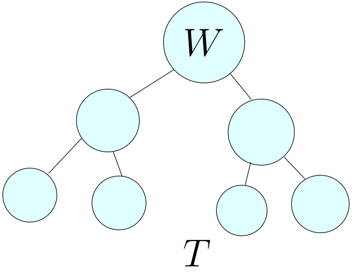


Setting

We have a tree decomposition T whose largest bag is W

Goal:

1. either decrease the number of bags of size $|W|$ while not increasing the width of T, or
2. conclude that T is optimal



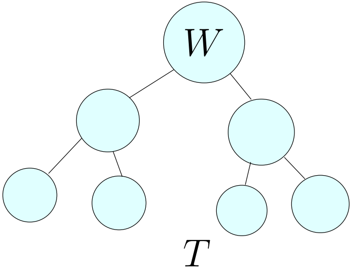
Setting

We have a tree decomposition T whose largest bag is W

Goal:

1. either decrease the number of bags of size $|W|$ while not increasing the width of T, or
2. conclude that T is optimal

Repeat for $\mathcal{O}(\mathrm{tw}(G) \cdot n)$ iterations to get an optimal tree decomposition



Setting

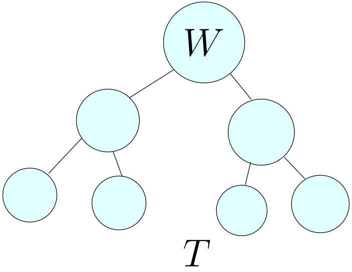
We have a tree decomposition T whose largest bag is W

Goal:

1. either decrease the number of bags of size $|W|$ while not increasing the width of T, or
2. conclude that T is optimal

Repeat for $\mathcal{O}(\text{tw}(G) \cdot n)$ iterations to get an optimal tree decomposition

(by [Bodlaender'93] we can assume to start with a decomposition of width $\mathcal{O}(\text{tw}(G))$)



Improving a tree decomposition

Let W be a largest bag of T

Improving a tree decomposition

Let W be a largest bag of T

Want to find:

Improving a tree decomposition

Let W be a largest bag of T

Want to find:

- a set X with $W \subseteq X \subseteq V(G)$, and

Improving a tree decomposition

Let W be a largest bag of T

Want to find:

- a set X with $W \subseteq X \subseteq V(G)$, and
- a tree decomposition of $\texttt{torso}(X)$ of width $\leq |W| - 2$

Improving a tree decomposition

Let W be a largest bag of T — SUBSET TREEWIDTH

Want to find:

- a set X with $W \subseteq X \subseteq V(G)$, and
- a tree decomposition of $\texttt{torso}(X)$ of width $\leq |W| - 2$

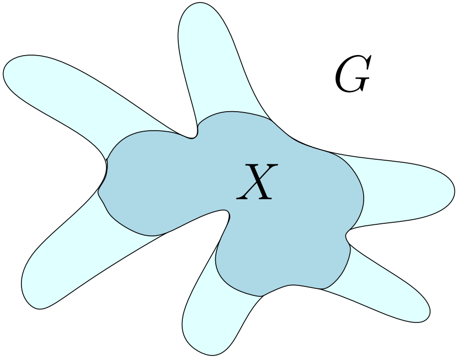
Improving a tree decomposition

Let W be a largest bag of T SUBSET TREEWIDTH

Want to find:

- a set X with $W \subseteq X \subseteq V(G)$, and
- a tree decomposition of $\mathtt{torso}(X)$ of width $\leq |W| - 2$

Torso?



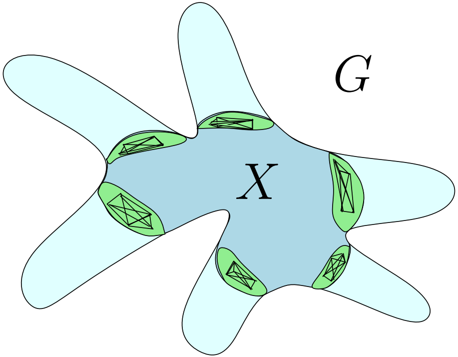
Improving a tree decomposition

Let W be a largest bag of T SUBSET TREEWIDTH

Want to find:

- a set X with $W \subseteq X \subseteq V(G)$, and
- a tree decomposition of $\texttt{torso}(X)$ of width $\leq |W| - 2$

Torso?



- Make neighborhoods of components of $G \setminus X$ into cliques

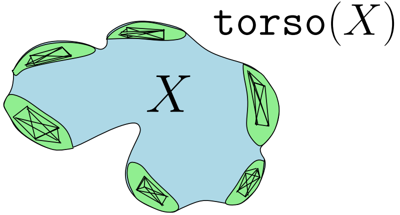
Improving a tree decomposition

Let W be a largest bag of T SUBSET TREEWIDTH

Want to find:

- a set X with $W \subseteq X \subseteq V(G)$, and
- a tree decomposition of $\mathtt{torso}(X)$ of width $\leq |W| - 2$

Torso?



- Make neighborhoods of components of $G \setminus X$ into cliques
- Delete $V(G) \setminus X$

Improving a tree decomposition

Let W be a largest bag of T	SUBSET TREEWIDTH

Want to find:

- a set X with $W \subseteq X \subseteq V(G)$, and
- a tree decomposition of $\mathtt{torso}(X)$ of width $\leq |W| - 2$

Observations:

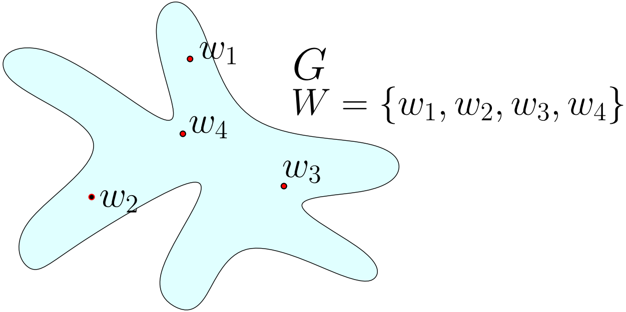
Improving a tree decomposition

Let W be a largest bag of T SUBSET TREEWIDTH

Want to find:

- a set X with $W \subseteq X \subseteq V(G)$, and
- a tree decomposition of $\texttt{torso}(X)$ of width $\leq |W| - 2$

Observations:



Improving a tree decomposition

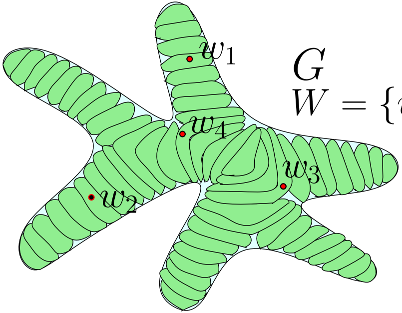
Let W be a largest bag of T SUBSET TREEWIDTH

Want to find:

- a set X with $W \subseteq X \subseteq V(G)$, and
- a tree decomposition of $\texttt{torso}(X)$ of width $\leq |W| - 2$

Observations:

- If T is not optimal, then such X exists by taking $X = V(G)$



G

$W = \{w_1, w_2, w_3, w_4\}$

Improving a tree decomposition

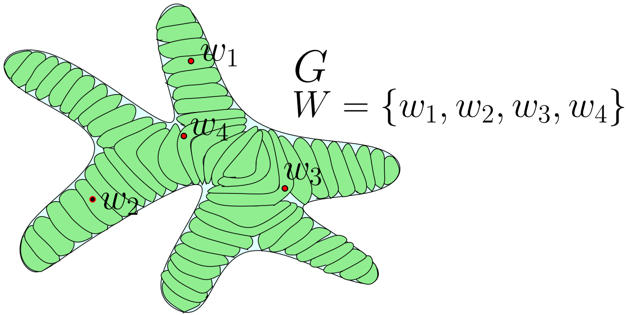
Let W be a largest bag of T — SUBSET TREEWIDTH

Want to find:

- a set X with $W \subseteq X \subseteq V(G)$, and
- a tree decomposition of $\texttt{torso}(X)$ of width $\leq |W| - 2$

Observations:

- If T is not optimal, then such X exists by taking $X = V(G)$
- Freedom to choose $X \subset V(G)$



Improving a tree decomposition

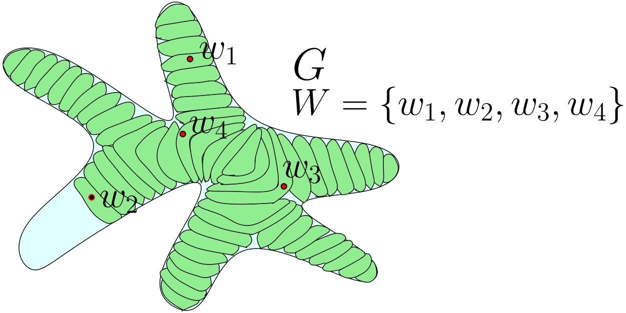
Let W be a largest bag of T SUBSET TREEWIDTH

Want to find:

- a set X with $W \subseteq X \subseteq V(G)$, and
- a tree decomposition of $\texttt{torso}(X)$ of width $\leq |W| - 2$

Observations:

- If T is not optimal, then such X exists by taking $X = V(G)$
- Freedom to choose $X \subset V(G)$



Improving a tree decomposition

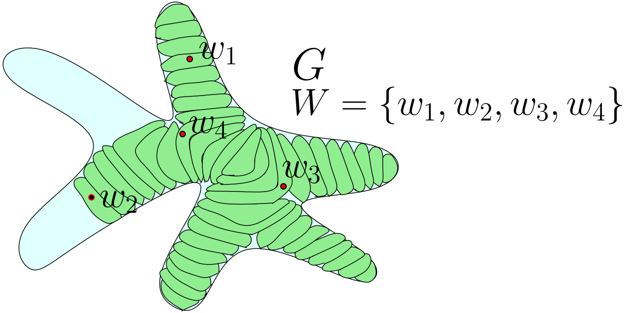
Let W be a largest bag of T SUBSET TREEWIDTH

Want to find:

- a set X with $W \subseteq X \subseteq V(G)$, and
- a tree decomposition of $\texttt{torso}(X)$ of width $\leq |W| - 2$

Observations:

- If T is not optimal, then such X exists by taking $X = V(G)$
- Freedom to choose $X \subset V(G)$



Improving a tree decomposition

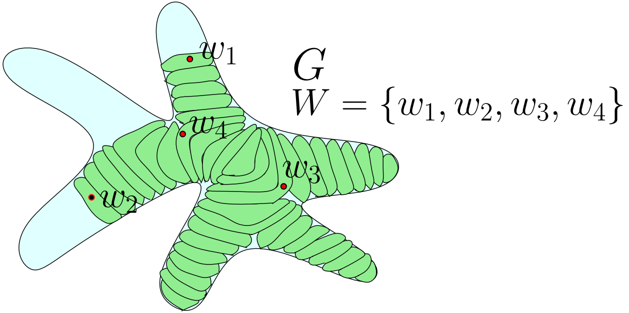
Let W be a largest bag of T — SUBSET TREEWIDTH

Want to find:

- a set X with $W \subseteq X \subseteq V(G)$, and
- a tree decomposition of $\mathtt{torso}(X)$ of width $\leq |W| - 2$

Observations:

- If T is not optimal, then such X exists by taking $X = V(G)$
- Freedom to choose $X \subset V(G)$



Improving a tree decomposition

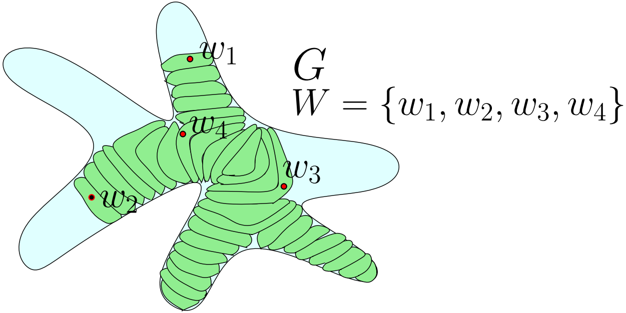
Let W be a largest bag of T — SUBSET TREEWIDTH

Want to find:

- a set X with $W \subseteq X \subseteq V(G)$, and
- a tree decomposition of $\texttt{torso}(X)$ of width $\leq |W| - 2$

Observations:

- If T is not optimal, then such X exists by taking $X = V(G)$
- Freedom to choose $X \subset V(G)$



Improving a tree decomposition

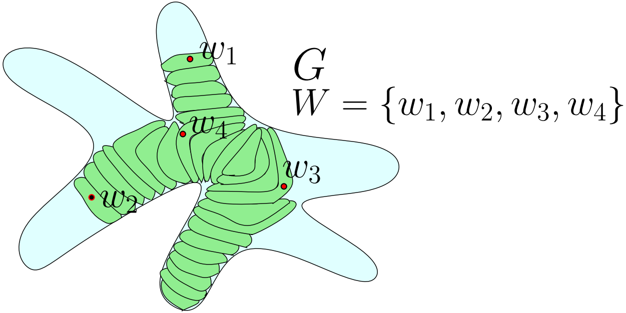
Let W be a largest bag of T — SUBSET TREEWIDTH

Want to find:

- a set X with $W \subseteq X \subseteq V(G)$, and
- a tree decomposition of $\mathtt{torso}(X)$ of width $\leq |W| - 2$

Observations:

- If T is not optimal, then such X exists by taking $X = V(G)$
- Freedom to choose $X \subset V(G)$



Improving a tree decomposition

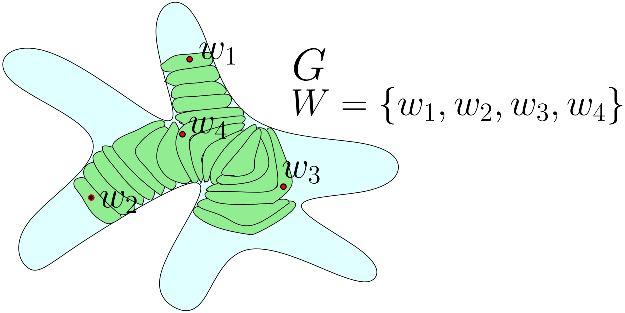
Let W be a largest bag of T — SUBSET TREEWIDTH

Want to find:

- a set X with $W \subseteq X \subseteq V(G)$, and
- a tree decomposition of $\texttt{torso}(X)$ of width $\leq |W| - 2$

Observations:

- If T is not optimal, then such X exists by taking $X = V(G)$
- Freedom to choose $X \subset V(G)$



Improving a tree decomposition

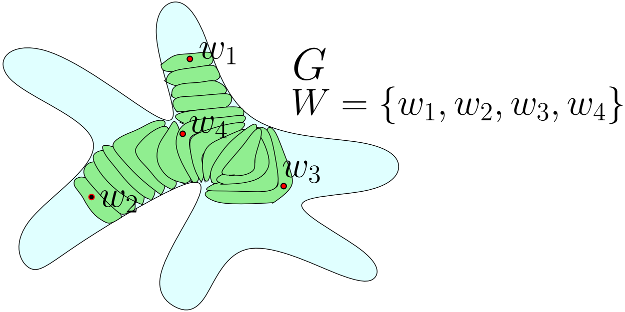
Let W be a largest bag of T SUBSET TREEWIDTH

Want to find:

- a set X with $W \subseteq X \subseteq V(G)$, and
- a tree decomposition of $\mathtt{torso}(X)$ of width $\leq |W| - 2$

Observations:

- If T is not optimal, then such X exists by taking $X = V(G)$
- Freedom to choose $X \subset V(G)$



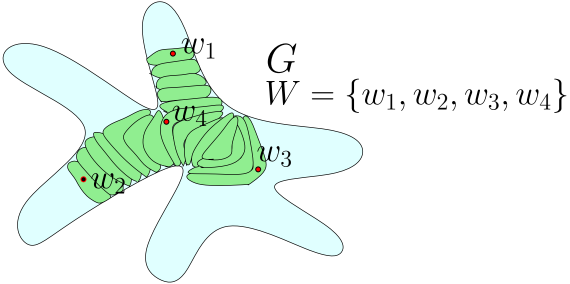
Improving a tree decomposition

Let W be a largest bag of T — SUBSET TREEWIDTH

Want to find:

- a set X with $W \subseteq X \subseteq V(G)$, and
- a tree decomposition of $\texttt{torso}(X)$ of width $\leq |W| - 2$

Big-leaf formulation:



Improving a tree decomposition

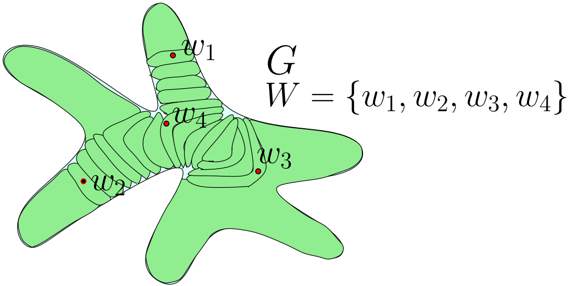
Let W be a largest bag of T — Subset treewidth

Want to find:

- a set X with $W \subseteq X \subseteq V(G)$, and
- a tree decomposition of $\mathtt{torso}(X)$ of width $\leq |W| - 2$

Big-leaf formulation:

- Find a tree decomposition of G whose internal bags have size $< |W|$ and cover W, but leaf bags can be arbitrarily large



Improving a tree decomposition

Let W be a largest bag of T — Subset treewidth

Have:

- a set X with $W \subseteq X \subseteq V(G)$, and
- a tree decomposition T_X of $\texttt{torso}(X)$ of width $\leq |W| - 2$

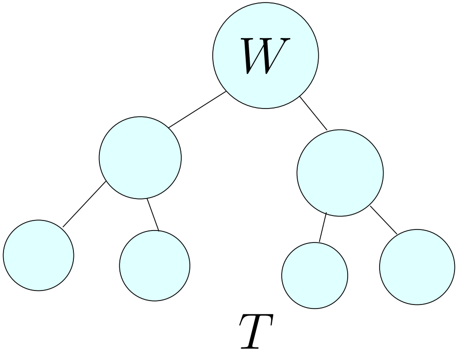
Improving a tree decomposition

Let W be a largest bag of T — SUBSET TREEWIDTH

Have:

- a set X with $W \subseteq X \subseteq V(G)$, and
- a tree decomposition T_X of $\texttt{torso}(X)$ of width $\leq |W| - 2$

Improving T:



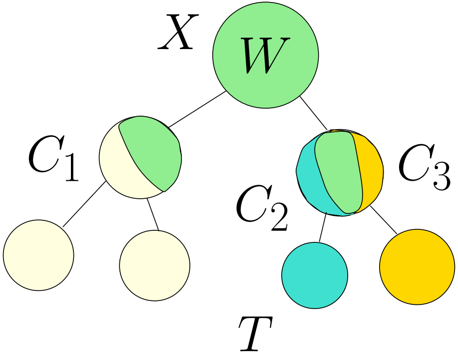
Improving a tree decomposition

Let W be a largest bag of T — SUBSET TREEWIDTH

Have:

- a set X with $W \subseteq X \subseteq V(G)$, and
- a tree decomposition T_X of $\texttt{torso}(X)$ of width $\leq |W| - 2$

Improving T:



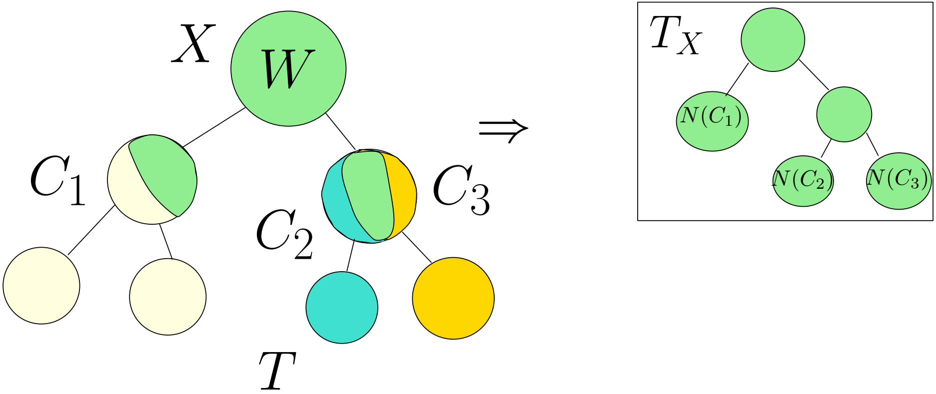
Improving a tree decomposition

Let W be a largest bag of T SUBSET TREEWIDTH

Have:

- a set X with $W \subseteq X \subseteq V(G)$, and
- a tree decomposition T_X of $\texttt{torso}(X)$ of width $\leq |W| - 2$

Improving T:



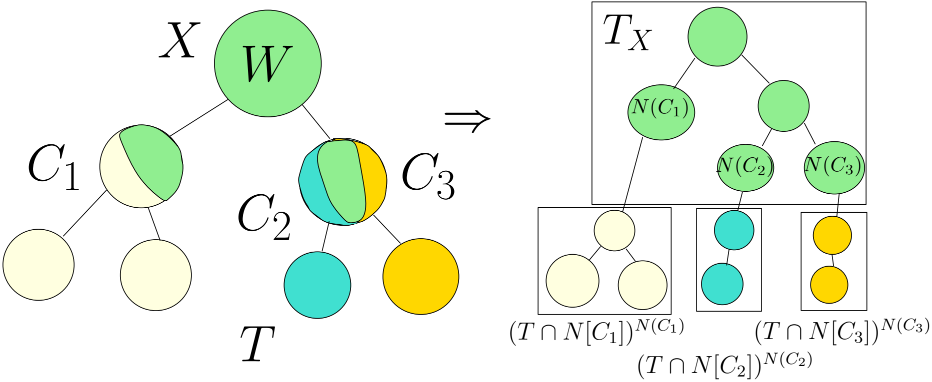
Improving a tree decomposition

Let W be a largest bag of T — SUBSET TREEWIDTH

Have:

- a set X with $W \subseteq X \subseteq V(G)$, and
- a tree decomposition T_X of $\texttt{torso}(X)$ of width $\leq |W| - 2$

Improving T:

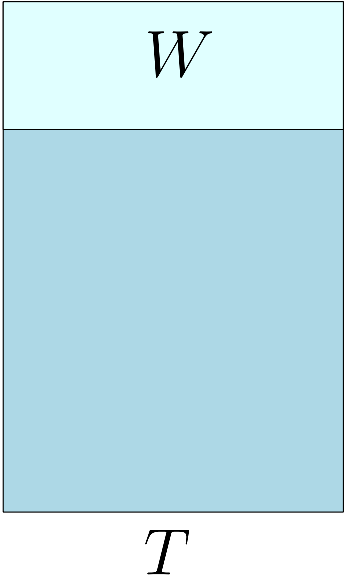


Constructing $(T \cap N[C_i])^{N(C_i)}$

Goal: For each component C_i of $G \setminus X$, construct a tree decomposition of $G[N[C_i]]$ so that $N(C_i)$ is in the root

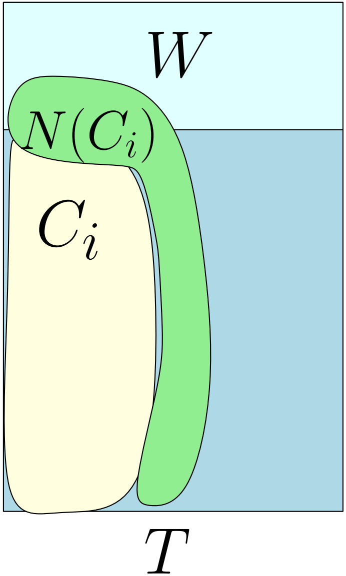
Constructing $(T \cap N[C_i])^{N(C_i)}$

Goal: For each component C_i of $G \setminus X$, construct a tree decomposition of $G[N[C_i]]$ so that $N(C_i)$ is in the root



Constructing $(T \cap N[C_i])^{N(C_i)}$

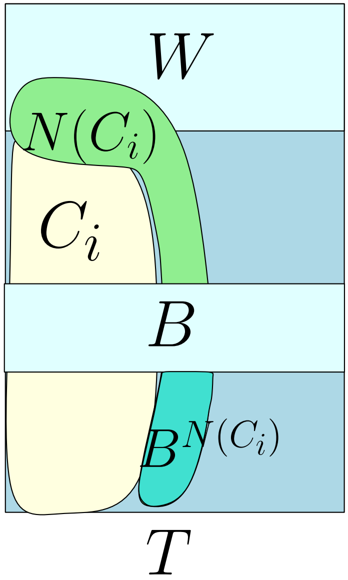
Goal: For each component C_i of $G \setminus X$, construct a tree decomposition of $G[N[C_i]]$ so that $N(C_i)$ is in the root



Constructing $(T \cap N[C_i])^{N(C_i)}$

Goal: For each component C_i of $G \setminus X$, construct a tree decomposition of $G[N[C_i]]$ so that $N(C_i)$ is in the root

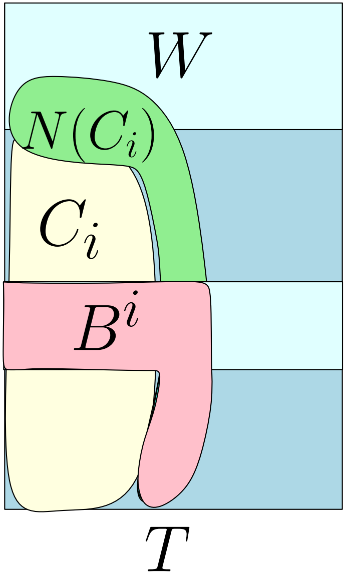
Replace each bag B by:
$B^i = (B \cap N[C_i]) \cup B^{N(C_i)}$



Constructing $(T \cap N[C_i])^{N(C_i)}$

Goal: For each component C_i of $G \setminus X$, construct a tree decomposition of $G[N[C_i]]$ so that $N(C_i)$ is in the root

Replace each bag B by:
$B^i = (B \cap N[C_i]) \cup B^{N(C_i)}$

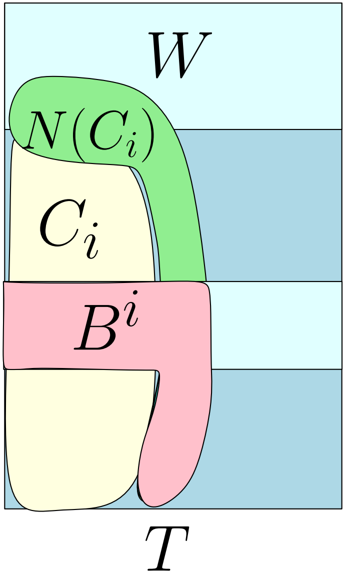


Constructing $(T \cap N[C_i])^{N(C_i)}$

Goal: For each component C_i of $G \setminus X$, construct a tree decomposition of $G[N[C_i]]$ so that $N(C_i)$ is in the root

Replace each bag B by:
$B^i = (B \cap N[C_i]) \cup B^{N(C_i)}$

What if $|B^i| > |B|$?



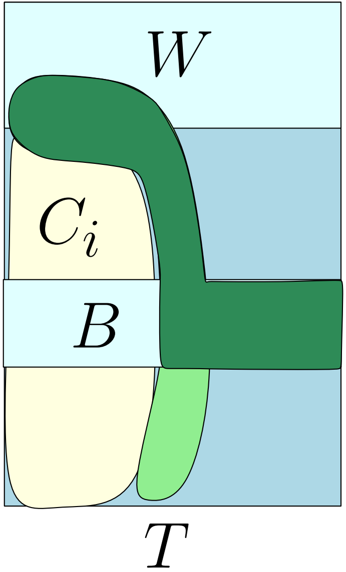
Constructing $(T \cap N[C_i])^{N(C_i)}$

Goal: For each component C_i of $G \setminus X$, construct a tree decomposition of $G[N[C_i]]$ so that $N(C_i)$ is in the root

Replace each bag B by:
$B^i = (B \cap N[C_i]) \cup B^{N(C_i)}$

What if $|B^i| > |B|$?

Then $(N(C_i) \setminus B^{N(C_i)}) \cup (B \setminus C)$ is a separator between $N(C_i)$ and W of size $< |N(C_i)|$



Constructing $(T \cap N[C_i])^{N(C_i)}$

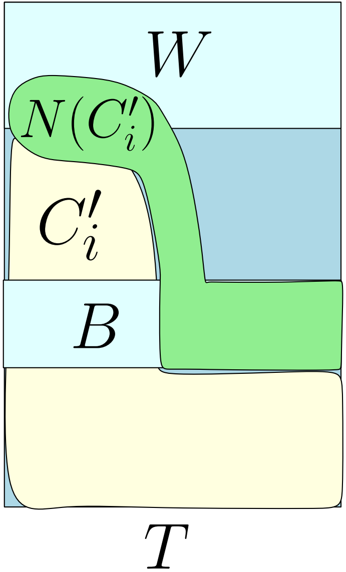
Goal: For each component C_i of $G \setminus X$, construct a tree decomposition of $G[N[C_i]]$ so that $N(C_i)$ is in the root

Replace each bag B by:
$B^i = (B \cap N[C_i]) \cup B^{N(C_i)}$

What if $|B^i| > |B|$?

Then $(N(C_i) \setminus B^{N(C_i)}) \cup (B \setminus C)$ is a separator between $N(C_i)$ and W of size $< |N(C_i)|$

$\Rightarrow$ Create new X by "pushing" $N(C_i)$ forward



Constructing $(T \cap N[C_i])^{N(C_i)}$

Goal: For each component C_i of $G \setminus X$, construct a tree decomposition of $G[N[C_i]]$ so that $N(C_i)$ is in the root

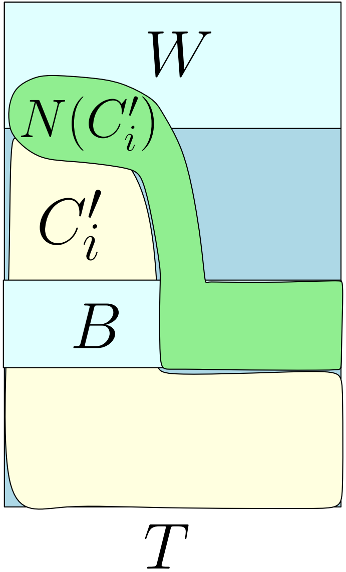
Replace each bag B by:
$B^i = (B \cap N[C_i]) \cup B^{N(C_i)}$

What if $|B^i| > |B|$?

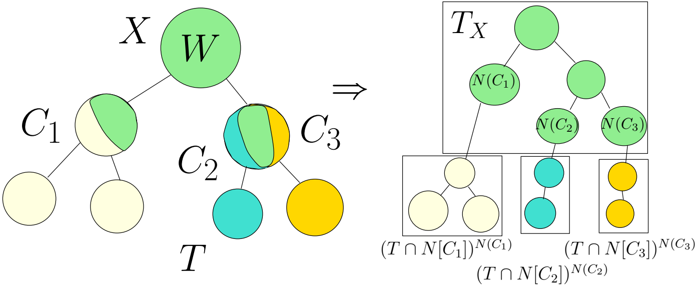
Then $(N(C_i) \setminus B^{N(C_i)}) \cup (B \setminus C)$ is a separator between $N(C_i)$ and W of size $< |N(C_i)|$

$\Rightarrow$ Create new X by "pushing" $N(C_i)$ forward

Decreases $|X|$

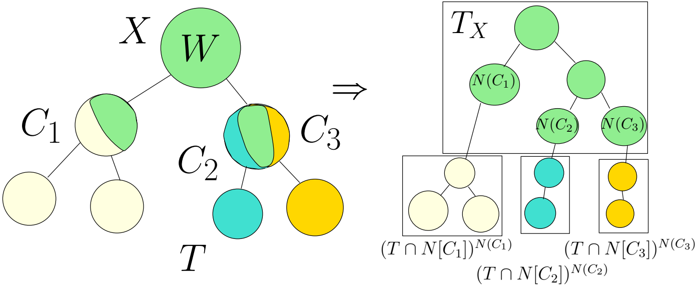


Result



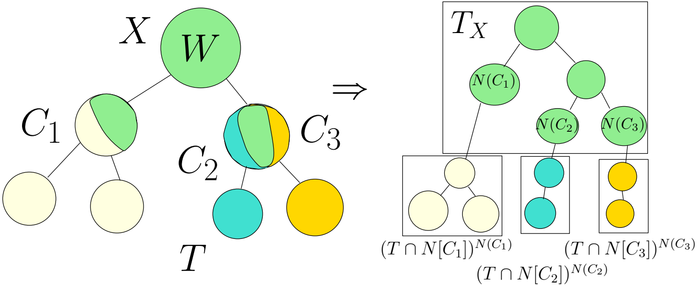
- By repeatedly applying the pushing argument, we achieve:

Result



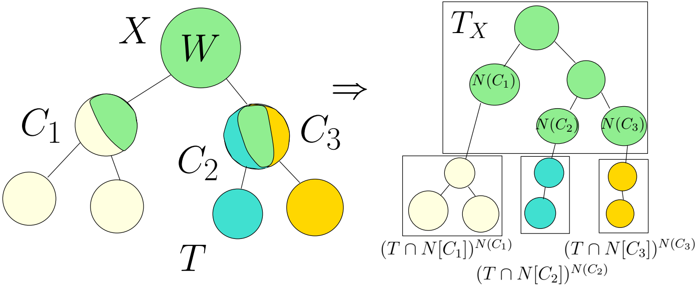
- By repeatedly applying the pushing argument, we achieve:
- The copy B^i of a bag in $(T \cap N[C_i])^{N(C_i)}$ is not larger than the original bag B

Result



- By repeatedly applying the pushing argument, we achieve:
- The copy B^i of a bag in $(T \cap N[C_i])^{N(C_i)}$ is not larger than the original bag B
  - n^4 in the running time comes from here

Result



- By repeatedly applying the pushing argument, we achieve:
- The copy B^i of a bag in $(T \cap N[C_i])^{N(C_i)}$ is not larger than the original bag B
  - n^4 in the running time comes from here
- Proof idea generalization of proofs of existence of lean tree decompositions [Thomas '90, Bellenbaum & Diestel '02]

Subset treewidth for exact FPT algorithms

Subset treewidth for exact FPT algorithms

SUBSET TREEWIDTH

Input: Graph G, integer k, set of vertices $W \subseteq V(G)$ with $|W| = k + 2$

Output: Set $X \subseteq V(G)$ with $W \subseteq X$ and tree decomposition of $\texttt{torso}(X)$ of width $\leq k$ or that the treewidth of G is $> k$

Subset treewidth for exact FPT algorithms

SUBSET TREEWIDTH

Input: Graph G, integer k, set of vertices $W \subseteq V(G)$ with $|W| = k + 2$

Output: Set $X \subseteq V(G)$ with $W \subseteq X$ and tree decomposition of $\texttt{torso}(X)$ of width $\leq k$ or that the treewidth of G is $> k$

Theorem

If there is an $f(k) \cdot n^{\mathcal{O}(1)}$ time algorithm for subset treewidth, then there is an $f(k) \cdot n^{\mathcal{O}(1)}$ time algorithm for treewidth with the same function f.

Subset treewidth for exact FPT algorithms

SUBSET TREEWIDTH

Input: Graph G, integer k, set of vertices $W \subseteq V(G)$ with $|W| = k + 2$

Output: Set $X \subseteq V(G)$ with $W \subseteq X$ and tree decomposition of $\mathtt{torso}(X)$ of width $\leq k$ or that the treewidth of G is $> k$

Theorem

If there is an $f(k) \cdot n^{\mathcal{O}(1)}$ time algorithm for subset treewidth, then there is an $f(k) \cdot n^{\mathcal{O}(1)}$ time algorithm for treewidth with the same function f.

$2^{\mathcal{O}(k^2)} n^2$ time algorithm for subset treewidth $\rightarrow$ $2^{\mathcal{O}(k^2)} n^4$ time algorithm for treewidth

How to solve subset treewidth?

How to solve subset treewidth?

Techniques:

How to solve subset treewidth?

Techniques:

1. Branching on important separators [Marx'06]

How to solve subset treewidth?

Techniques:

1. Branching on important separators [Marx'06] (poly space!)

How to solve subset treewidth?

Techniques:

1. Branching on important separators [Marx'06] (poly space!)

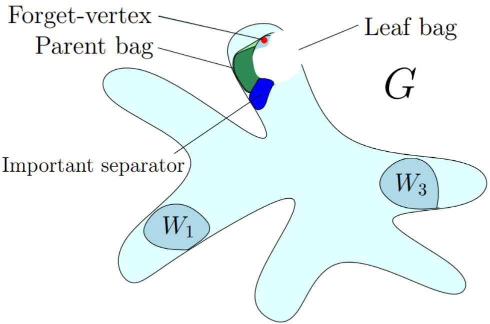
2. Lot of Bellenbaum-Diestel type “pulling arguments” to re-arrange tree decompositions

How to solve subset treewidth?

Techniques:

1. Branching on important separators [Marx'06] (poly space!)

2. Lot of Bellenbaum-Diestel type "pulling arguments" to re-arrange tree decompositions



Local improvement in dynamic treewidth

joint work with Konrad Majewski, Wojciech Nadara, Michał Pilipczuk & Marek Sokołowski

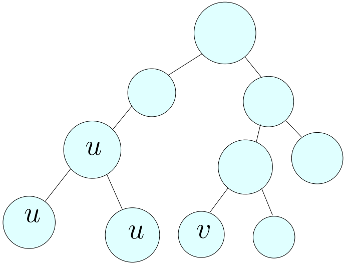
Local improvement in dynamic treewidth

Local improvement in dynamic treewidth

Goal: Maintain a tree decomposition of width $\mathcal{O}(k)$ and depth $n^{o(1)}$

Goal: Maintain a tree decomposition of width $\mathcal{O}(k)$ and depth $n^{o(1)}$

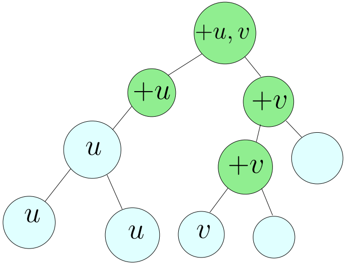
- Edge insertion:



Local improvement in dynamic treewidth

Goal: Maintain a tree decomposition of width $\mathcal{O}(k)$ and depth $n^{o(1)}$

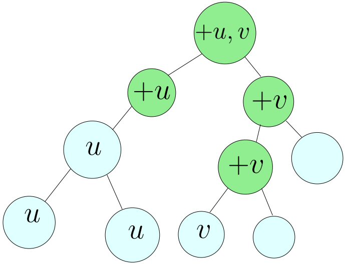
- Edge insertion: Add endpoints to all bags on the path from their subtrees to the root



Local improvement in dynamic treewidth

Goal: Maintain a tree decomposition of width $\mathcal{O}(k)$ and depth $n^{o(1)}$

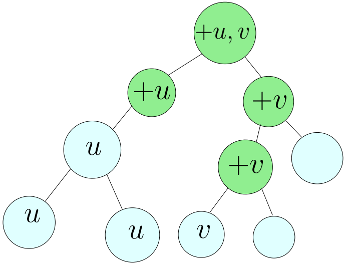
- Edge insertion: Add endpoints to all bags on the path from their subtrees to the root
- Increases width!



Local improvement in dynamic treewidth

Goal: Maintain a tree decomposition of width $\mathcal{O}(k)$ and depth $n^{o(1)}$

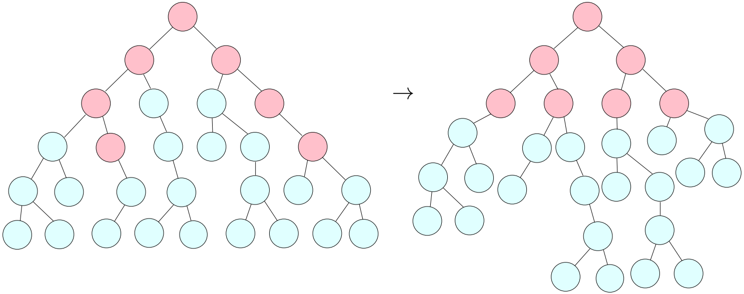
- Edge insertion: Add endpoints to all bags on the path from their subtrees to the root
- Increases width! But only in a subtree of size $\mathcal{O}(\text{depth}) = n^{o(1)}$



Local improvement in dynamic treewidth

Goal: Maintain a tree decomposition of width $\mathcal{O}(k)$ and depth $n^{o(1)}$

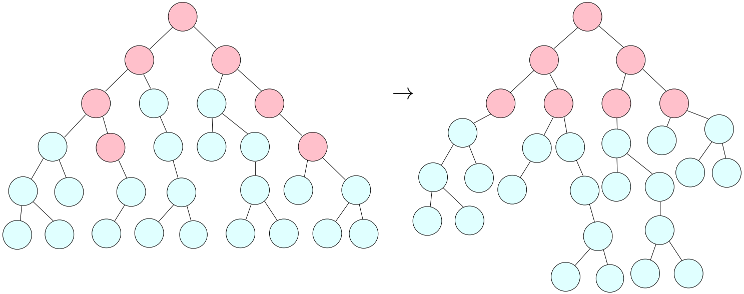
- Edge insertion: Add endpoints to all bags on the path from their subtrees to the root
- Increases width! But only in a subtree of size $\mathcal{O}(\text{depth}) = n^{o(1)}$
- Refinement operation: Rebuild a subtree T in amortized time $f(k) \cdot |T|$



Local improvement in dynamic treewidth

Goal: Maintain a tree decomposition of width $\mathcal{O}(k)$ and depth $n^{o(1)}$

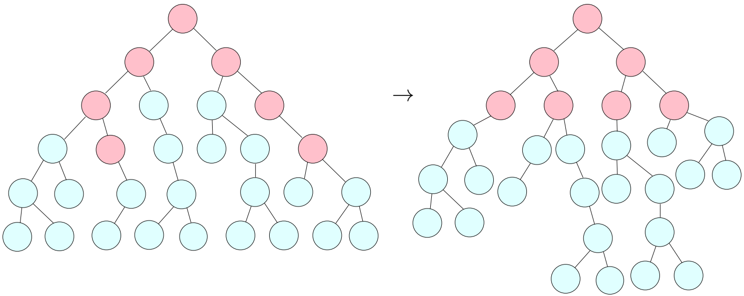
- Edge insertion: Add endpoints to all bags on the path from their subtrees to the root
- Increases width! But only in a subtree of size $\mathcal{O}(\text{depth}) = n^{o(1)}$
- Refinement operation: Rebuild a subtree T in amortized time $f(k) \cdot |T|$
- Re-arranges given subtree into depth $\mathcal{O}(\log n)$ and width $\leq 6k + 5$



Local improvement in dynamic treewidth

Goal: Maintain a tree decomposition of width $\mathcal{O}(k)$ and depth $n^{o(1)}$

- Edge insertion: Add endpoints to all bags on the path from their subtrees to the root
- Increases width! But only in a subtree of size $\mathcal{O}(\text{depth}) = n^{o(1)}$
- Refinement operation: Rebuild a subtree T in amortized time $f(k) \cdot |T|$
- Re-arranges given subtree into depth $\mathcal{O}(\log n)$ and width $\leq 6k + 5$
- Builds on subset treewidth, log-depth decompositions [Bodlaender & Hagerup '98], and the "dealternation lemma" [Bojańczyk & Pilipczuk '22]



Conclusion

New method for FPT algorithms for treewidth: Local improvement

Conclusion

New method for FPT algorithms for treewidth: Local improvement

- Introduced in [K. '21] for 2-approximation in $2^{\mathcal{O}(k)} n$ time

Conclusion

New method for FPT algorithms for treewidth: Local improvement

- Introduced in [K. '21] for 2-approximation in $2^{\mathcal{O}(k)}n$ time
- Generalized in [K. & Lokshtanov '23] for exact in $2^{\mathcal{O}(k^2)}n^4$ time and $(1+\varepsilon)$-approximation in $k^{\mathcal{O}(k/\varepsilon)}n^4$ time

Conclusion

New method for FPT algorithms for treewidth: Local improvement

- Introduced in [K. '21] for 2-approximation in $2^{\mathcal{O}(k)} n$ time
- Generalized in [K. & Lokshtanov '23] for exact in $2^{\mathcal{O}(k^2)} n^4$ time and $(1+\varepsilon)$-approximation in $k^{\mathcal{O}(k/\varepsilon)} n^4$ time
- Used in [K., Majewski, Nadara, Pilipczuk & Sokołowski '23] for fully dynamic treewidth in $f(k) \cdot n^{o(1)}$ amortized update time

Conclusion

New method for FPT algorithms for treewidth: Local improvement

- Introduced in [K. '21] for 2-approximation in $2^{\mathcal{O}(k)} n$ time
- Generalized in [K. & Lokshtanov '23] for exact in $2^{\mathcal{O}(k^2)} n^4$ time and $(1+\varepsilon)$-approximation in $k^{\mathcal{O}(k/\varepsilon)} n^4$ time
- Used in [K., Majewski, Nadara, Pilipczuk & Sokołowski '23] for fully dynamic treewidth in $f(k) \cdot n^{o(1)}$ amortized update time

Open problems:

Conclusion

New method for FPT algorithms for treewidth: Local improvement

- Introduced in [K. '21] for 2-approximation in $2^{\mathcal{O}(k)} n$ time
- Generalized in [K. & Lokshtanov '23] for exact in $2^{\mathcal{O}(k^2)} n^4$ time and $(1+\varepsilon)$-approximation in $k^{\mathcal{O}(k/\varepsilon)} n^4$ time
- Used in [K., Majewski, Nadara, Pilipczuk & Sokołowski '23] for fully dynamic treewidth in $f(k) \cdot n^{o(1)}$ amortized update time

Open problems:

- Prove $2^{\Omega(k)}$ lower bound for treewidth under ETH ($2^{\Omega(\sqrt{k})}$ known)

Conclusion

New method for FPT algorithms for treewidth: Local improvement

- Introduced in [K. '21] for 2-approximation in $2^{\mathcal{O}(k)}n$ time
- Generalized in [K. & Lokshtanov '23] for exact in $2^{\mathcal{O}(k^2)}n^4$ time and $(1+\varepsilon)$-approximation in $k^{\mathcal{O}(k/\varepsilon)}n^4$ time
- Used in [K., Majewski, Nadara, Pilipczuk & Sokołowski '23] for fully dynamic treewidth in $f(k) \cdot n^{o(1)}$ amortized update time

Open problems:

- Prove $2^{\Omega(k)}$ lower bound for treewidth under ETH ($2^{\Omega(\sqrt{k})}$ known)
- Treewidth 1.9-approximation in $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$ time?

Conclusion

New method for FPT algorithms for treewidth: Local improvement

- Introduced in [K. '21] for 2-approximation in $2^{\mathcal{O}(k)} n$ time
- Generalized in [K. & Lokshtanov '23] for exact in $2^{\mathcal{O}(k^2)} n^4$ time and $(1+\varepsilon)$-approximation in $k^{\mathcal{O}(k/\varepsilon)} n^4$ time
- Used in [K., Majewski, Nadara, Pilipczuk & Sokołowski '23] for fully dynamic treewidth in $f(k) \cdot n^{o(1)}$ amortized update time

Open problems:

- Prove $2^{\Omega(k)}$ lower bound for treewidth under ETH ($2^{\Omega(\sqrt{k})}$ known)
- Treewidth 1.9-approximation in $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$ time?
- Dynamic treewidth in amortized $f(k) \cdot \text{polylog}(n)$ time?

Conclusion

New method for FPT algorithms for treewidth: Local improvement

- Introduced in [K. '21] for 2-approximation in $2^{\mathcal{O}(k)} n$ time
- Generalized in [K. & Lokshtanov '23] for exact in $2^{\mathcal{O}(k^2)} n^4$ time and $(1+\varepsilon)$-approximation in $k^{\mathcal{O}(k/\varepsilon)} n^4$ time
- Used in [K., Majewski, Nadara, Pilipczuk & Sokołowski '23] for fully dynamic treewidth in $f(k) \cdot n^{o(1)}$ amortized update time

Open problems:

- Prove $2^{\Omega(k)}$ lower bound for treewidth under ETH ($2^{\Omega(\sqrt{k})}$ known)
- Treewidth 1.9-approximation in $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$ time?
- Dynamic treewidth in amortized $f(k) \cdot \text{polylog}(n)$ time?

Thank you!