A Single-Exponential Time 2-Approximation Algorithm for Treewidth

Tuukka Korhonen

Department of Computer Science, University of Helsinki, Finland

Frontiers of Parameterized Complexity
May 27, 2021

In this talk

Theorem. There is an algorithm with:

Input: n-vertex graph G and an integer k.
Output: Tree decomposition of G of width at most $2k+1$, or $\mathrm{tw}(G) > k$.
Running time: $2^{O(k)}n$.

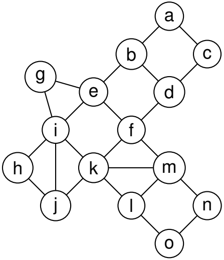
Plan

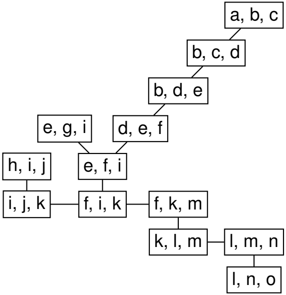
1. Preliminaries and motivation
2. Background and outline of the algorithm
3. Main concepts of the algorithm
4. Implementation in $2^{O(k)} n$ time

Tree Decompositions

A tree decomposition of a graph G is a tree T whose each node $i \in V(T)$ is associated with a bag $B_i \subseteq V(G)$, and satisfies

1. $V(G) = \bigcup_{i \in V(T)} B_i$,
2. for each $\{u, v\} \in E(G)$ there is $i \in V(T)$ with $\{u, v\} \subseteq B_i$, and
3. for each $v \in V(G)$, the induced subgraph $T[\{i \mid v \in B_i\}]$ of T is connected.



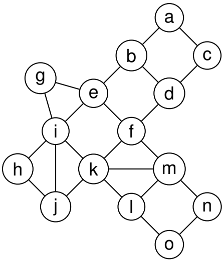


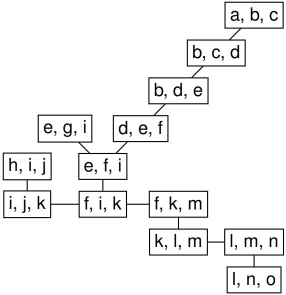
Tree Decompositions

A tree decomposition of a graph G is a tree T whose each node $i \in V(T)$ is associated with a bag $B_i \subseteq V(G)$, and satisfies

1. $V(G) = \bigcup_{i \in V(T)} B_i$,
2. for each $\{u, v\} \in E(G)$ there is $i \in V(T)$ with $\{u, v\} \subseteq B_i$, and
3. for each $v \in V(G)$, the induced subgraph $T[\{i \mid v \in B_i\}]$ of T is connected.

The width of a tree decomposition T is $\max_{i \in V(T)} |B_i| - 1$, and the treewidth of G is the minimum width of a tree decomposition of G.





Applications of treewidth

Given a graph with a tree decomposition of width k, we can solve [Bod88, BCKN15]:

- Maximum independent set in time $2^k k^{O(1)} n$
- Minimum dominating set in time $3^k k^{O(1)} n$
- Steiner tree in time $2^{O(k)} n$
- ...

Applications of treewidth

Given a graph with a tree decomposition of width k, we can solve [Bod88, BCKN15]:

- Maximum independent set in time $2^k k^{O(1)} n$
- Minimum dominating set in time $3^k k^{O(1)} n$
- Steiner tree in time $2^{O(k)} n$
- ...

To actually solve these in time $2^{O(k)} n$, where k is the treewidth, we need to find a tree decomposition of width ck in time $2^{O(k)} n$, for some constant c.

Applications of treewidth

Given a graph with a tree decomposition of width k, we can solve [Bod88, BCKN15]:

- Maximum independent set in time $2^k k^{O(1)} n$
- Minimum dominating set in time $3^k k^{O(1)} n$
- Steiner tree in time $2^{O(k)} n$
- ...

To actually solve these in time $2^{O(k)} n$, where k is the treewidth, we need to find a tree decomposition of width ck in time $2^{O(k)} n$, for some constant c.

The approximation ratio c matters:
For $c = 5$, the DP for dominating set takes $3^{5k} k^{O(1)} n = 243^k k^{O(1)} n$ time, while for $c = 2$ it takes $3^{2k} k^{O(1)} n = 9^k k^{O(1)} n$ time.

FPT Approximation Algorithms for treewidth

FPT algorithms for computing a tree decomposition of width at most $\alpha(k)$ in time $f(k)g(n)$, where k is the treewidth of the input graph.

Reference	$\alpha(k)$	$f(k)$	$g(n)$
[Bod96]	k	$k^{O(k^3)}$	n

FPT Approximation Algorithms for treewidth

FPT algorithms for computing a tree decomposition of width at most $\alpha(k)$ in time $f(k)g(n)$, where k is the treewidth of the input graph.

Reference	$\alpha(k)$	$f(k)$	$g(n)$
[Bod96]	k	$k^{O(k^3)}$	n
[FHL08]	$O(k\sqrt{\log k})$	$O(1)$	$n^{O(1)}$
[FLS+18]	$O(k^2)$	$O(k^7)$	$n \log n$

FPT Approximation Algorithms for treewidth

FPT algorithms for computing a tree decomposition of width at most $\alpha(k)$ in time $f(k)g(n)$, where k is the treewidth of the input graph.

Reference	$\alpha(k)$	$f(k)$	$g(n)$
[RS95]	$4k+3$	$O(3^{3k})$	n^2
[Lag96]	$8k+7$	$2^{O(k \log k)}$	$n \log^2 n$
[Ree92]	$8k+O(1)$	$2^{O(k \log k)}$	$n \log n$
[Bod96]	k	$k^{O(k^3)}$	n
[Ami10]	$4.5k$	$O(2^{3k} k^{3/2})$	n^2
[Ami10]	$(3+2/3)k$	$O(2^{3.6982k} k^3)$	n^2
[FHL08]	$O(k\sqrt{\log k})$	$O(1)$	$n^{O(1)}$
[FLS+18]	$O(k^2)$	$O(k^7)$	$n \log n$

FPT Approximation Algorithms for treewidth

FPT algorithms for computing a tree decomposition of width at most $\alpha(k)$ in time $f(k)g(n)$, where k is the treewidth of the input graph.

Reference	$\alpha(k)$	$f(k)$	$g(n)$
[RS95]	$4k+3$	$O(3^{3k})$	n^2
[Lag96]	$8k+7$	$2^{O(k \log k)}$	$n \log^2 n$
[Ree92]	$8k+O(1)$	$2^{O(k \log k)}$	$n \log n$
[Bod96]	k	$k^{O(k^3)}$	n
[Ami10]	$4.5k$	$O(2^{3k} k^{3/2})$	n^2
[Ami10]	$(3+2/3)k$	$O(2^{3.6982k} k^3)$	n^2
[FHL08]	$O(k\sqrt{\log k})$	$O(1)$	$n^{O(1)}$
[FLS$^+$18]	$O(k^2)$	$O(k^7)$	$n \log n$
[BDD$^+$16]	$3k+4$	$2^{O(k)}$	$n \log n$
[BDD$^+$16]	$5k+4$	$2^{O(k)}$	n

FPT Approximation Algorithms for treewidth

FPT algorithms for computing a tree decomposition of width at most $\alpha(k)$ in time $f(k)g(n)$, where k is the treewidth of the input graph.

Reference	$\alpha(k)$	$f(k)$	$g(n)$
[RS95]	$4k+3$	$O(3^{3k})$	n^2
[Lag96]	$8k+7$	$2^{O(k \log k)}$	$n \log^2 n$
[Ree92]	$8k+O(1)$	$2^{O(k \log k)}$	$n \log n$
[Bod96]	k	$k^{O(k^3)}$	n
[Ami10]	$4.5k$	$O(2^{3k} k^{3/2})$	n^2
[Ami10]	$(3+2/3)k$	$O(2^{3.6982k} k^3)$	n^2
[FHL08]	$O(k\sqrt{\log k})$	$O(1)$	$n^{O(1)}$
[FLS+18]	$O(k^2)$	$O(k^7)$	$n \log n$
[BDD+16]	$3k+4$	$2^{O(k)}$	$n \log n$
[BDD+16]	$5k+4$	$2^{O(k)}$	n
This work	$2k+1$	$2^{O(k)}$	n

FPT Approximation Algorithms for treewidth

FPT algorithms for computing a tree decomposition of width at most $\alpha(k)$ in time $f(k)g(n)$, where k is the treewidth of the input graph.

Reference	$\alpha(k)$	$f(k)$	$g(n)$	
[RS95]	$4k+3$	$O(3^{3k})$	n^2	
[Lag96]	$8k+7$	$2^{O(k \log k)}$	$n \log^2 n$	
[Ree92]	$8k+O(1)$	$2^{O(k \log k)}$	$n \log n$	
[Bod96]	k	$k^{O(k^3)}$	n	
[Ami10]	$4.5k$	$O(2^{3k} k^{3/2})$	n^2	
[Ami10]	$(3+2/3)k$	$O(2^{3.6982k} k^3)$	n^2	
[FHL08]	$O(k\sqrt{\log k})$	$O(1)$	$n^{O(1)}$	
[FLS+18]	$O(k^2)$	$O(k^7)$	$n \log n$	
[BDD+16]	$3k+4$	$2^{O(k)}$	$n \log n$	
[BDD+16]	$5k+4$	$2^{O(k)}$	n	$f(k) = \Omega(2^{40k})$
This work	$2k+1$	$2^{O(k)}$	n	$f(k) = O(2^{11k})$

Part 2: Background and outline of the algorithm

Building Blocks

Bodlaender's compression technique [Bod96]:

Suppose there is a c-approximation algorithm for treewidth with running time $f(k)n$ that requires a tree decomposition of width $2ck$ as an input.

Then there is a c-approximation algorithm for treewidth with running time $k^{O(1)}f(k)n$ without the requirement.

Building Blocks

Bodlaender's compression technique [Bod96]:

Suppose there is a c-approximation algorithm for treewidth with running time $f(k)n$ that requires a tree decomposition of width $2ck$ as an input.

Then there is a c-approximation algorithm for treewidth with running time $k^{O(1)}f(k)n$ without the requirement.

The new algorithm:

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

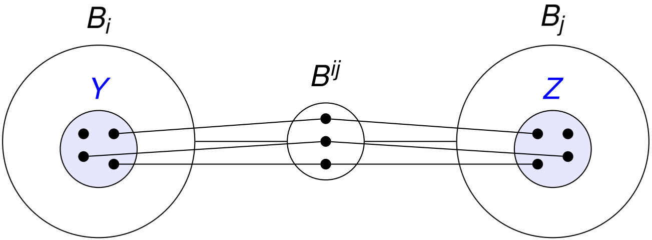
Output: A tree decomposition of G of width at most $w-1$, or $w \leq 2 \cdot tw(G)+1$.

Running time: $2^{O(w)}n$.

Background: Lean Tree Decompositions

Definition (Lean tree decomposition)

A tree decomposition T is *lean* if for any two bags B_i, B_j and vertex sets $Y \subseteq B_i$ and $Z \subseteq B_j$, the number of disjoint paths between Y and Z is $\min(|Y|, |Z|, |B^{ij}|)$, where B^{ij} is a minimum size bag in the path between i and j in T.



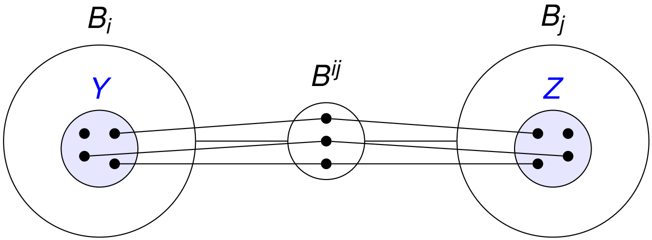
Background: Lean Tree Decompositions

Definition (Lean tree decomposition)

A tree decomposition T is *lean* if for any two bags B_i, B_j and vertex sets $Y \subseteq B_i$ and $Z \subseteq B_j$, the number of disjoint paths between Y and Z is $\min(|Y|, |Z|, |B^{ij}|)$, where B^{ij} is a minimum size bag in the path between i and j in T.

Theorem ([Tho90, BD02])

Every graph of treewidth k has a lean tree decomposition of width k.



Background: Lean Tree Decompositions

Definition (Lean tree decomposition)

A tree decomposition T is *lean* if for any two bags B_i, B_j and vertex sets $Y \subseteq B_i$ and $Z \subseteq B_j$, the number of disjoint paths between Y and Z is $\min(|Y|, |Z|, |B^{ij}|)$, where B^{ij} is a minimum size bag in the path between i and j in T.

Theorem ([Tho90, BD02])

Every graph of treewidth k has a lean tree decomposition of width k.

We are not interested in the theorem, but on the proof idea of [BD02]:

- Let T be a tree decomposition that is not lean
- $\Rightarrow$ we can apply an improvement step to T
- This does not increase the width of T, and decreases an invariant on T.

Background: Lean Tree Decompositions

Definition (Lean tree decomposition)

A tree decomposition T is *lean* if for any two bags B_i, B_j and vertex sets $Y \subseteq B_i$ and $Z \subseteq B_j$, the number of disjoint paths between Y and Z is $\min(|Y|, |Z|, |B^{ij}|)$, where B^{ij} is a minimum size bag in the path between i and j in T.

Theorem ([Tho90, BD02])

Every graph of treewidth k has a lean tree decomposition of width k.

We are not interested in the theorem, but on the proof idea of [BD02]:

- Let T be a tree decomposition that is not lean
- $\Rightarrow$ we can apply an improvement step to T
- This does not increase the width of T, and decreases an invariant on T.
- Given the "lean witness" (Y, Z), the improvement can be implemented in polynomial time.

The idea: Adaptation of Bellenbaum–Diestel

Observation

If there is a bag B_i of size $|B_i| \geq 3 \cdot tw(G) + 3$, then there is a lean witness (Y, Z) with $Y \subseteq B_i$, $Z \subseteq B_i$.

$\Rightarrow$ 3-approximation algorithm with time complexity $2^{O(k)} n^{O(1)}$ follows quite directly from Bellenbaum–Diestel.

The idea: Adaptation of Bellenbaum–Diestel

Observation

If there is a bag B_i of size $|B_i| \geq 3 \cdot tw(G) + 3$, then there is a lean witness (Y, Z) with $Y \subseteq B_i$, $Z \subseteq B_i$.

$\Rightarrow$ 3-approximation algorithm with time complexity $2^{O(k)} n^{O(1)}$ follows quite directly from Bellenbaum–Diestel.

- To achieve approximation ratio 2 instead of 3, instead of lean witnesses we use 3-way separations of bags

The idea: Adaptation of Bellenbaum–Diestel

Observation

If there is a bag B_i of size $|B_i| \geq 3 \cdot tw(G) + 3$, then there is a lean witness (Y, Z) with $Y \subseteq B_i$, $Z \subseteq B_i$.

$\Rightarrow$ 3-approximation algorithm with time complexity $2^{O(k)} n^{O(1)}$ follows quite directly from Bellenbaum–Diestel.

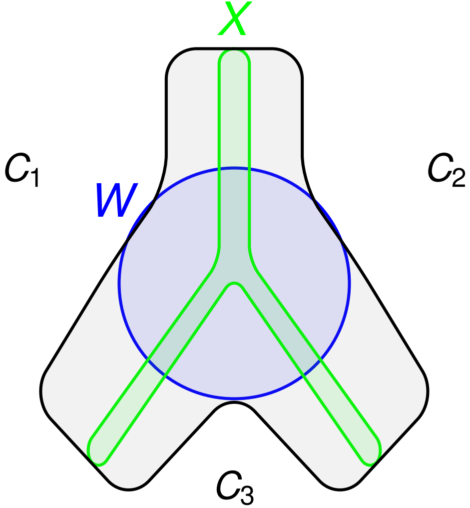
- To achieve approximation ratio 2 instead of 3, instead of lean witnesses we use 3-way separations of bags
- To achieve time complexity $2^{O(k)} n$, we
  - use a "local" version of the improvement
  - refine the invariant analysis
  - use dynamic programming on T to find the separations

Part 3: Main concepts of the algorithm

Splittable Bags

Definition (Splittable bag)

A bag $W \subseteq V(G)$ is *splittable* if $V(G)$ can be partitioned into (C_1, C_2, C_3, X) with no edges between C_i and C_j for $i \neq j$ and $|(W \cap C_i) \cup X| < |W|$ for all $i \in \{1, 2, 3\}$.

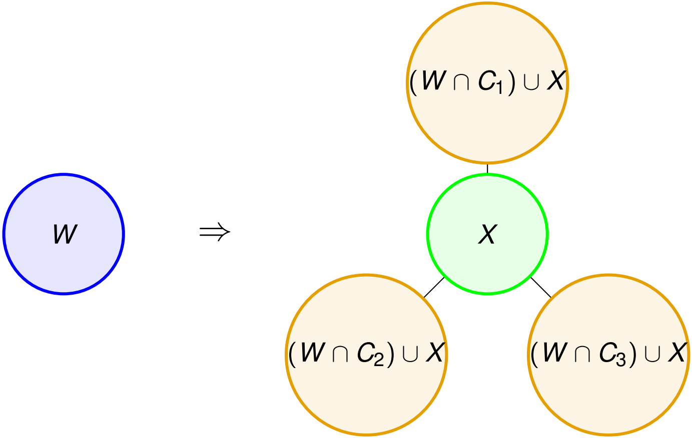


Splittable Bags

Definition (Splittable bag)

A bag $W \subseteq V(G)$ is *splittable* if $V(G)$ can be partitioned into (C_1, C_2, C_3, X) 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 local construction improves the tree decomposition



Splittable Bags

Definition (Splittable bag)

A bag $W \subseteq V(G)$ is *splittable* if $V(G)$ can be partitioned into (C_1, C_2, C_3, X) with no edges between C_i and C_j for $i \neq j$ and $|(W \cap C_i) \cup X| < |W|$ for all i.

Lemma

Any bag W of size $|W| \geq 2 \cdot tw(G) + 3$ is splittable.

Proof:
Let X be a $\frac{1}{2}$-balanced separator of W of size $|X| \leq tw(G) + 1$.

Splitting a Tree Decomposition T

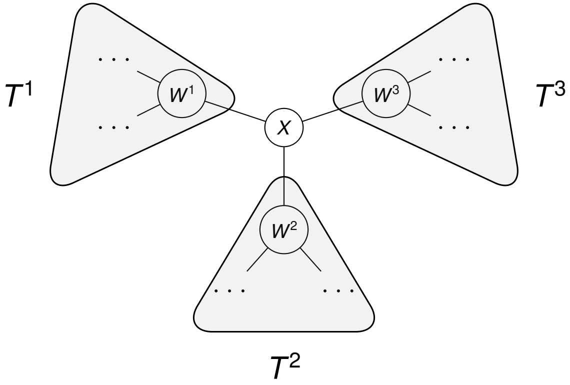
- Let W be the root bag of T and let (C_1, C_2, C_3, X) be a split of W.

Splitting a Tree Decomposition T

- Let W be the root bag of T and let (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 of $G[C_i \cup X]$ by setting $B^i = B \cap (C_i \cup X)$ for each bag B of T.

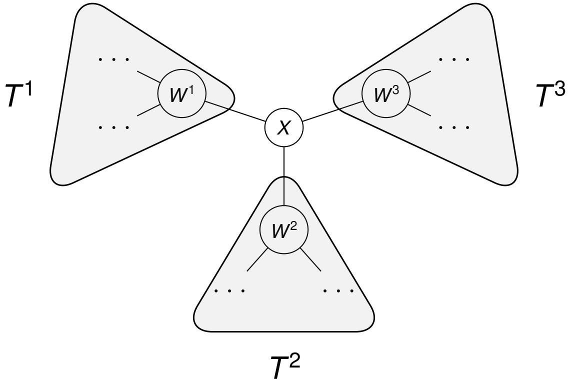
Splitting a Tree Decomposition T

- Let W be the root bag of T and let (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 of $G[C_i \cup X]$ by setting $B^i = B \cap (C_i \cup X)$ for each bag B of T.
- Now, the following is (almost) a tree decomposition of G:



Splitting a Tree Decomposition T

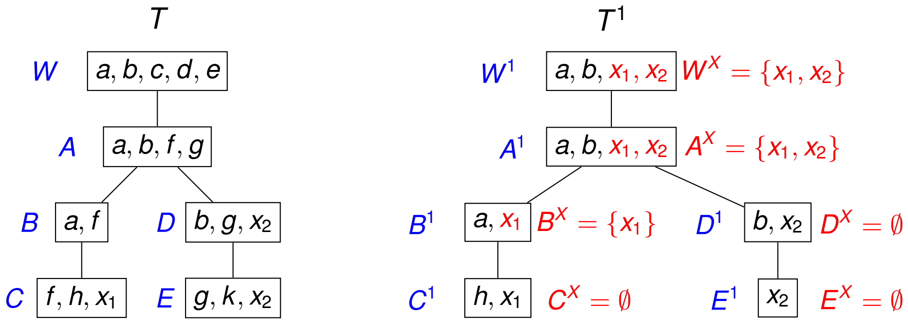
- Let W be the root bag of T and let (C_1, C_2, C_3, X) be a split of W.
- For each $i \in \{1, 2, 3\}$, obtain a tree decomposition T^i of $G[C_i \cup X]$ by setting $B^i = B \cap (C_i \cup X)$ for each bag B of T.
- Now, the following is (almost) a tree decomposition of G:



Except that vertices $x_j \in X$ may violate the connected subtree condition

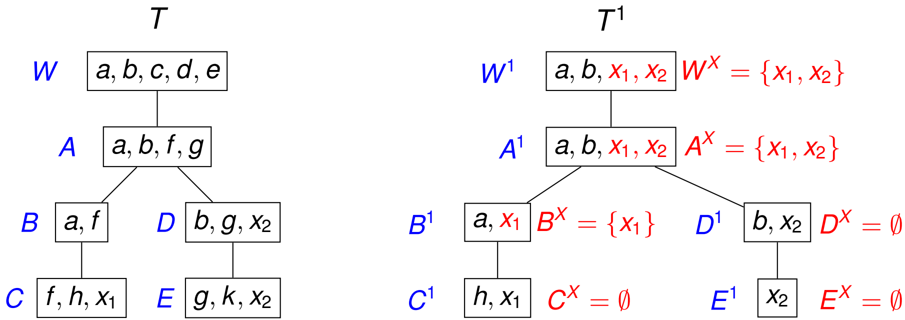
Fixing a tree decomposition

- We fix the connectedness condition by inserting vertices $x_j \in X$ to bags
- Let $B^i = (B \cap (C_i \cup X)) \cup B^X$
- Example: Consider a split $(\{a, b, h\}, \{c, d, f\}, \{e, g, k\}, \{x_1, x_2\})$ of W:



Fixing a tree decomposition

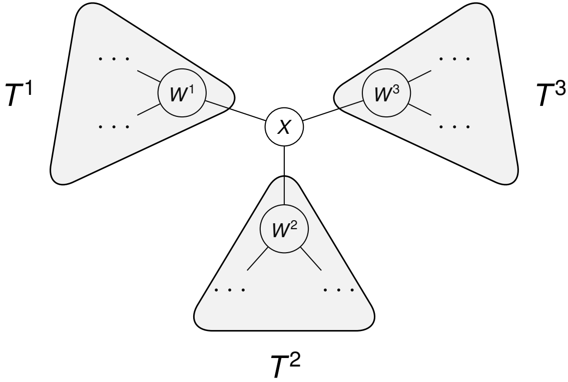
- We fix the connectedness condition by inserting vertices $x_j \in X$ to bags
- Let $B^i = (B \cap (C_i \cup X)) \cup B^X$
- Example: Consider a split $(\{a, b, h\}, \{c, d, f\}, \{e, g, k\}, \{x_1, x_2\})$ of W:



The *home bag* of x_1 is C and the home bag of x_2 is D.

Splitting a Tree Decomposition

Now, with $B^i = (B \cap (C_i \cup X)) \cup B^X$, the following is a tree decomposition of G:



- $|W^i| < |W|$ (and $|X| < |W|$) by the definition of a split.
- It remains to bound $|B^i|$ for the other bags B.

Minimum Splits

We need some extra properties from the split that we use

Definition (Minimum Split)

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

1. primarily minimizes $|X|$, and
2. secondarily minimizes ...

Bounding $|B^i|$

Lemma

Let (C_1, C_2, C_3, X) be a minimum split of W. For all bags B and distinct i, j it holds that $|B^X| \leq |B \cap (C_i \cup C_j)|$.

Bounding $|B^i|$

Lemma

Let (C_1, C_2, C_3, X) be a minimum split of W. For all bags B and distinct i, j it holds that $|B^X| \leq |B \cap (C_i \cup C_j)|$.

$\Rightarrow |B^i| = |(B \cap (C_i \cup X)) \cup B^X| \leq |B|$.

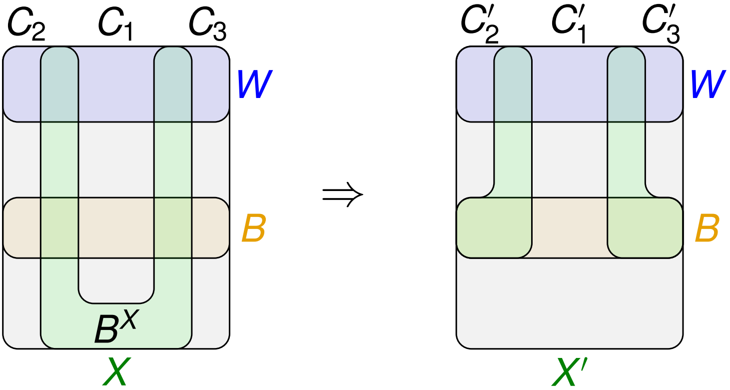
Bounding $|B^i|$

Lemma

Let (C_1, C_2, C_3, X) be a minimum split of W. For all bags B and distinct i, j it holds that $|B^X| \leq |B \cap (C_i \cup C_j)|$.

$\Rightarrow |B^i| = |(B \cap (C_i \cup X)) \cup B^X| \leq |B|$.

Proof: Suppose $|B^X| > |B \cap (C_2 \cup C_3)|$. Take $X' = (X \setminus B^X) \cup (B \cap (C_2 \cup C_3))$. We have $|X'| < |X|$, so a split (C_1', C_2', C_3', X') contradicts the minimality.



Minimum Splits: The Secondary Condition

We need to require extra properties from the split that we use

Definition (Minimum Split)

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

1. primarily minimizes $|X|$, and
2. secondarily minimizes $\sum_{x_j \in X} d_T(x_j, W)$.

Where $d_T(x, W)$ is the distance in T from the home bag $H(x)$ of x to W.

Minimum Splits: The Secondary Condition

We need to require extra properties from the split that we use

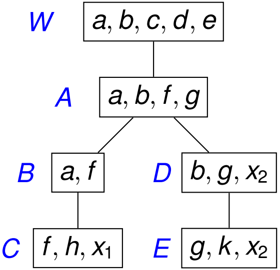
Definition (Minimum Split)

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

1. primarily minimizes $|X|$, and
2. secondarily minimizes $\sum_{x_j \in X} d_T(x_j, W)$.

Where $d_T(x, W)$ is the distance in T from the home bag $H(x)$ of x to W.

Example: Here $H(x_1) = C$ and thus $d_T(x_1, W) = 3$, and $H(x_2) = D$ and thus $d_T(x_2, W) = 2$. Therefore $\sum_{x_j \in X} d_T(x_j, W) = 5$.



Bounding $|B^i|$: Stronger Bound

Lemma

Let (C_1, C_2, C_3, X) be a minimum split of W. For all bags B with non-empty B^X and distinct i, j it holds that $|B^X| < |B \cap (C_i \cup C_j)|$

Bounding $|B^i|$: Stronger Bound

Lemma

Let (C_1, C_2, C_3, X) be a minimum split of W. For all bags B with non-empty B^X and distinct i, j it holds that $|B^X| < |B \cap (C_i \cup C_j)|$

$\Rightarrow |B^i| = |(B \cap (C_i \cup X)) \cup B^X| < |B|$ if B^X is non-empty.

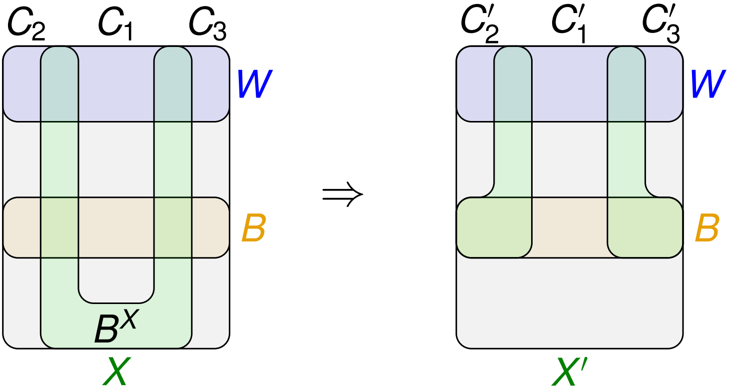
Bounding $|B^i|$: Stronger Bound

Lemma

Let (C_1, C_2, C_3, X) be a minimum split of W. For all bags B with non-empty B^X and distinct i, j it holds that $|B^X| < |B \cap (C_i \cup C_j)|$

$\Rightarrow |B^i| = |(B \cap (C_i \cup X)) \cup B^X| < |B|$ if B^X is non-empty.

The proof is exactly same as before, with the observation that $d_T(y, W) < d_T(x, W)$ for all $y \in B$ and $x \in B^X$.



Bounding $|B^i|$: Summary

By using minimum splits we have that:

1. $|B^i| \leq |B|$ for all bags B, and
2. $|B^i| < |B|$ if B^X is non-empty, implying
3. $|B^i| < |B|$ if B intersects more than one of C_1, C_2, C_3.

Bounding $|B^i|$: Summary

By using minimum splits we have that:

1. $|B^i| \leq |B|$ for all bags B, and
2. $|B^i| < |B|$ if B^X is non-empty, implying
3. $|B^i| < |B|$ if B intersects more than one of C_1, C_2, C_3.

The only case when $|B^1| = |B|$ is when $B^X = \emptyset$ and $B \cap (C_2 \cup C_3) = \emptyset$, in which case $B^2 \subseteq X$ and $B^3 \subseteq X$.
$\Rightarrow$ at most one of B^1, B^2, B^3 has size $|W|$.

Bounding $|B^i|$: Summary

By using minimum splits we have that:

1. $|B^i| \leq |B|$ for all bags B, and
2. $|B^i| < |B|$ if B^X is non-empty, implying
3. $|B^i| < |B|$ if B intersects more than one of C_1, C_2, C_3.

The only case when $|B^1| = |B|$ is when $B^X = \emptyset$ and $B \cap (C_2 \cup C_3) = \emptyset$, in which case $B^2 \subseteq X$ and $B^3 \subseteq X$.
$\Rightarrow$ at most one of B^1, B^2, B^3 has size $|W|$.

$\Rightarrow$Splitting a largest bag W of T does not increase the width of T, and decreases the number of bags of size $|W|$.

Bounding $|B^i|$: Summary

By using minimum splits we have that:

1. $|B^i| \leq |B|$ for all bags B, and
2. $|B^i| < |B|$ if B^X is non-empty, implying
3. $|B^i| < |B|$ if B intersects more than one of C_1, C_2, C_3.

The only case when $|B^1| = |B|$ is when $B^X = \emptyset$ and $B \cap (C_2 \cup C_3) = \emptyset$, in which case $B^2 \subseteq X$ and $B^3 \subseteq X$.
$\Rightarrow$ at most one of B^1, B^2, B^3 has size $|W|$.

$\Rightarrow$Splitting a largest bag W of T does not increase the width of T, and decreases the number of bags of size $|W|$.

$\Rightarrow$If we implement splitting in time $2^{O(w)}n$, we obtain a $2^{O(k)}n^2$ time 2-approximation algorithm for treewidth.

Part 4: Implementation in $2^{O(k)}n$ time

Issue 1: Number of bag edits

Issue: By implementing the splits as described before, solely the bag editing will take $\Omega(n^2)$ time over the course of the algorithm.

Issue 1: Number of bag edits

Issue: By implementing the splits as described before, solely the bag editing will take $\Omega(n^2)$ time over the course of the algorithm.

Solution: Modified bag editing process:

- We define *editable* bag
- Each splitting operation will take $w^{O(1)}t$ time, where t is the number of editable bags of the split.
- $t \leq \phi(T) - \phi(T')$, where T is the old tree decomposition, T' is the new tree decomposition, and ϕ is a potential function.
- In particular, $\phi(T) = \sum_{i \in V(T)} 7^{|B_i|}$.

Issue 1: Number of bag edits

Issue: By implementing the splits as described before, solely the bag editing will take $\Omega(n^2)$ time over the course of the algorithm.

Solution: Modified bag editing process:

- We define *editable* bag
- Each splitting operation will take $w^{O(1)}t$ time, where t is the number of editable bags of the split.
- $t \leq \phi(T) - \phi(T')$, where T is the old tree decomposition, T' is the new tree decomposition, and ϕ is a potential function.
- In particular, $\phi(T) = \sum_{i \in V(T)} 7^{|B_i|}$.

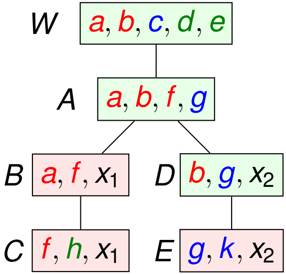
$\Rightarrow$ Time taken by bag editing will be bounded by $\phi(T)w^{O(1)} = 2^{O(w)}n$.

Editable bags

Definition

Let (C_1, C_2, C_3, X) be a split of a root bag W. A bag B is *editable* if it intersects at least two of C_1, C_2, C_3, and its parent is editable.

Example: Consider a split $(\{a, b, f\}, \{c, g, k\}, \{d, e, h\}, \{x_1, x_2\})$.



Here bags W, A, and D are editable.

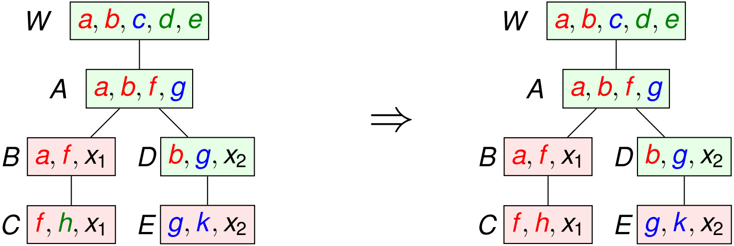
Re-Assigning Vertices

Definition

Let (C_1, C_2, C_3, X) be a split of a root bag W. A bag B is *editable* if it intersects at least two of C_1, C_2, C_3, and its parent is editable.

Consider a non-editable bag B whose parent A is editable

$(\{a, b, f\}, \{c, g, k\}, \{d, e, h\}, \{x_1, x_2\})$ $\quad$ $(\{a, b, f, h\}, \{c, g, k\}, \{d, e\}, \{x_1, x_2\})$



We re-assign h from C_3 to C_1.

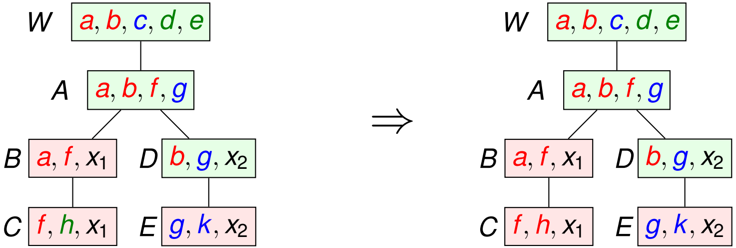
Re-Assigning Vertices

Definition

Let (C_1, C_2, C_3, X) be a split of a root bag W. A bag B is *editable* if it intersects at least two of C_1, C_2, C_3, and its parent is editable.

Consider a non-editable bag B whose parent A is editable

$(\{a, b, f\}, \{c, g, k\}, \{d, e, h\}, \{x_1, x_2\})$ $\quad$ $(\{a, b, f, h\}, \{c, g, k\}, \{d, e\}, \{x_1, x_2\})$



We re-assign h from C_3 to C_1.

With this re-assignment, a bag is editable if and only if it intersects at least two of C_1, C_2, C_3. Also, the editable bags form a connected subtree containing W.

Editing Editable Bags

Observation: For non-editable bags in minimum splits, $B^X = \emptyset$.

Editing Editable Bags

Observation: For non-editable bags in minimum splits, $B^X = \emptyset$.

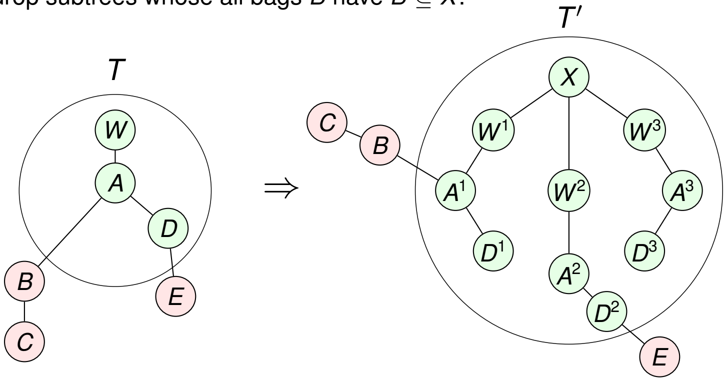
$\Rightarrow$ in minimum split, if B is non-editable and intersects C_1, then $B^1 = B$, $B^2 \subseteq X$ and $B^3 \subseteq X$, and this holds for all bags in the subtree below B.

Editing Editable Bags

Observation: For non-editable bags in minimum splits, $B^X = \emptyset$.

$\Rightarrow$ in minimum split, if B is non-editable and intersects C_1, then $B^1 = B$, $B^2 \subseteq X$ and $B^3 \subseteq X$, and this holds for all bags in the subtree below B.

We can drop subtrees whose all bags B have $B \subseteq X$.



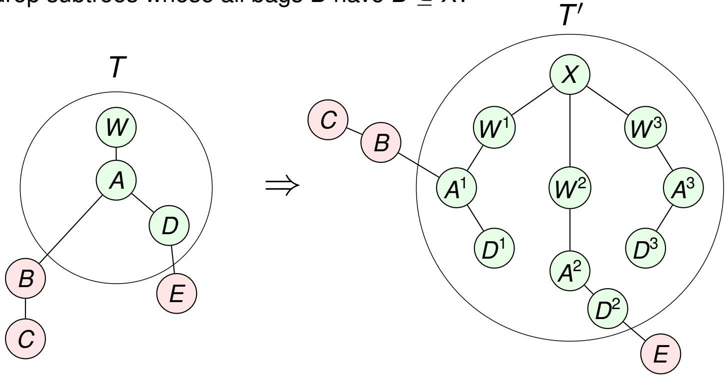
Now the splitting can be done by removing the t editable bags, inserting $3t+1$ new bags, and connecting the non-editable subtrees accordingly.

Editing Editable Bags

Observation: For non-editable bags in minimum splits, $B^X = \emptyset$.

$\Rightarrow$ in minimum split, if B is non-editable and intersects C_1, then $B^1 = B$, $B^2 \subseteq X$ and $B^3 \subseteq X$, and this holds for all bags in the subtree below B.

We can drop subtrees whose all bags B have $B \subseteq X$.



Now the splitting can be done by removing the t editable bags, inserting $3t + 1$ new bags, and connecting the non-editable subtrees accordingly.

Detail: To do this in $w^{O(1)}t$ time, we will maintain that T has maximum degree 3 (this increases the number of new bags inserted to $3t + 4$).

Analysis of the Potential Function

Consider the potential function $\phi(T) = \sum_{i \in V(T)} 7^{|B_i|}$.

Because $|B^i| < |B|$ for editable bags, simple computation shows $\phi(T) - \phi(T') \geq t$ where t is the number of editable bags.

Analysis of the Potential Function

Consider the potential function $\phi(T) = \sum_{i \in V(T)} 7^{|B_i|}$.

Because $|B^i| < |B|$ for editable bags, simple computation shows $\phi(T) - \phi(T') \geq t$ where t is the number of editable bags.

$\Rightarrow$ The algorithm can be implemented so that we edit $2^{O(w)}n$ bags in total over the course of the algorithm.

Issue 2: Finding the splits

Issue: We may need to perform $\Omega(n)$ split operations to decrease the width, so each split should be found in (amortized) $2^{O(w)}$ time.

Issue 2: Finding the splits

Issue: We may need to perform $\Omega(n)$ split operations to decrease the width, so each split should be found in (amortized) $2^{O(w)}$ time.

Solution: Use dynamic programming on the tree decomposition while editing it

- Using dynamic programming on the tree decomposition, we can compute a minimum split of the root bag in $2^{O(w)}n$ time.
- We can re-root the dynamic programming to an adjacent node in $2^{O(w)}$ time
- DFS-style procedure to with a dynamic programming-based data structure to implement everything in $2^{O(w)}n$ total time.

Data Structure

We maintain a degree-3 width $\leq w$ tree decomposition T rooted at node r.

Operation	Time	Description
Init(T, r)	$2^{O(w)}n$	Given a degree-3 width w tree decomposition T and a root node $r \in V(T)$, initialize the data structure.
Move(s)	$2^{O(w)}$	Move the root r to a neighboring node s.
Split()	$2^{O(w)}$	If the bag W of r is splittable set the internal state to a minimum split (C_1, C_2, C_3, X) of W and return $\top$. Otherwise return $\perp$.
State()	$w^{O(1)}$	Returns the internal state restricted to the bag W of r, i.e., the partition $(C_1 \cap W, C_2 \cap W, C_3 \cap W, X \cap W)$.
Edit(T_1, T_2, r')	$2^{O(w)}(\lvert T_1\rvert + \lvert T_2\rvert)$	Replaces a subtree T_1 with a given subtree T_2. Assumes that $r \in V(T_1)$, and places new r to r'.

Dynamic Programming

Let i be a node of T and $G[T_i]$ the subgraph of G induced by vertices in the subtree of T rooted at i.

$$dp[i][(C_1 \cap B_i, C_2 \cap B_i, C_3 \cap B_i, X \cap B_i)][h]$$

Stores either

1. The minimum value of $\sum_{x \in X} d_T(H(x), B_i)$ such that there is a partition (C_1, C_2, C_3, X) of $G[T_i]$ with no edges between C_i, C_j when $i \neq j$ and $|X| = h$ or
2. $\perp$ if no such partition exists.

Because T has degree-3, $dp[i][\ldots][\ldots]$ can be computed in $2^{O(w)}$ time based on its children

Dynamic Programming

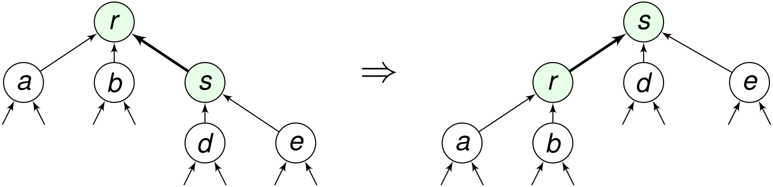
Let i be a node of T and $G[T_i]$ the subgraph of G induced by vertices in the subtree of T rooted at i.

$$dp[i][(C_1 \cap B_i, C_2 \cap B_i, C_3 \cap B_i, X \cap B_i)][h]$$

Stores either

1. The minimum value of $\sum_{x \in X} d_T(H(x), B_i)$ such that there is a partition (C_1, C_2, C_3, X) of $G[T_i]$ with no edges between C_i, C_j when $i \neq j$ and $|X| = h$ or
2. $\perp$ if no such partition exists.

Because T has degree-3, $dp[i][\ldots][\ldots]$ can be computed in $2^{O(w)}$ time based on its children



Moving the root from r to s requires updating tables of r and s.

Putting All Together

- DFS over the tree decomposition, every time we see a bag of size $w + 1$ we apply the splitting operation.
- By the potential function, the total number of nodes considered will be $2^{O(w)}n$
- Each move in the tree decomposition takes $2^{O(w)}$ time
- Total time complexity $2^{O(w)} \cdot 2^{O(w)}n = 2^{O(k)}n$.

Conclusion

- We gave a $2^{O(k)}n$ time 2-approximation algorithm for treewidth

Conclusion

- We gave a $2^{O(k)}n$ time 2-approximation algorithm for treewidth
- Future ideas: The approximation ratio 2 appears only in the lemma that guarantees every bag of size $\geq 2 \cdot tw(G) + 3$ is splittable. Can we do better analysis about what happens if we just continue splitting bags until no bag is splittable?
- For this purpose, the bag splitting can be generalized to arbitrary number of parts.

The end

Thank you for your attention!

Bibliography

Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski.
Complexity of finding embeddings in a k-tree.
SIAM Journal on Algebraic Discrete Methods, 8(2):277–284, 1987.

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

Hans L. Bodlaender, Marek Cygan, Stefan Kratsch, and Jesper Nederlof.
Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth.
Inf. Comput., 243:86–111, 2015.

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.

Hans L. Bodlaender.
Dynamic programming on graphs with bounded treewidth.
In Timo Lepistö and Arto Salomaa, editors, *Automata, Languages and Programming, 15th International Colloquium, ICALP88, Tampere, Finland, July 11-15, 1988, Proceedings*, volume 317 of *Lecture Notes in Computer Science*, pages 105–118. Springer, 1988.

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

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

Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, Michal Pilipczuk, and Marcin Wrochna.
Fully polynomial-time parameterized computations for graphs and matrices of low treewidth.
ACM Trans. Algorithms, 14(3):34:1–34:45, 2018.