Dynamic Treewidth

Tuukka Korhonen

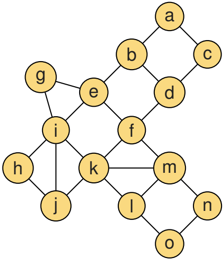
UNIVERSITY OF BERGEN

based on joint work with Konrad Majewski, Wojciech Nadara,
Michał Pilipczuk, and Marek Sokołowski from University of Warsaw

BARC talk

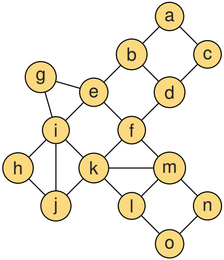
21 November 2023

Treewidth

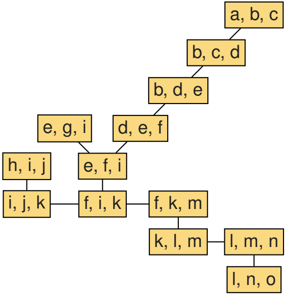


Graph G

Treewidth

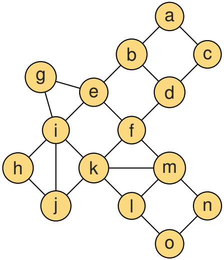


Graph G

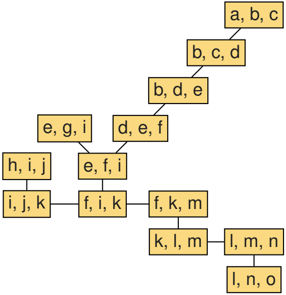


A tree decomposition of G

Treewidth



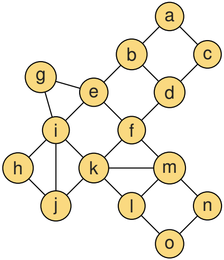
Graph G



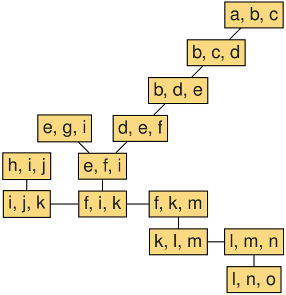
A tree decomposition of G

1. Every vertex should be in a bag

Treewidth



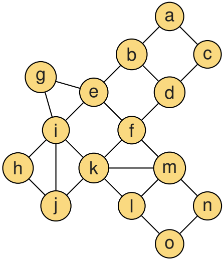
Graph G



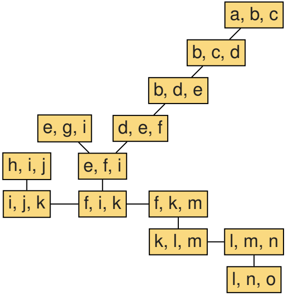
A tree decomposition of G

1. Every vertex should be in a bag
2. Every edge should be in a bag

Treewidth



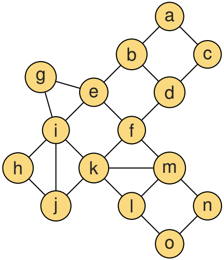
Graph G



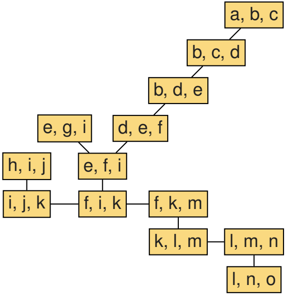
A tree decomposition of G

1. Every vertex should be in a bag
2. Every edge should be in a bag
3. Bags containing a vertex should form a connected subtree

Treewidth



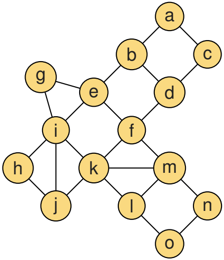
Graph G



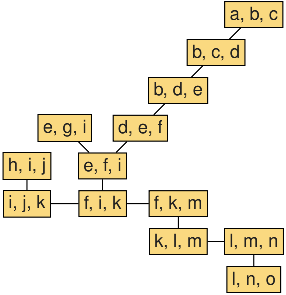
A tree decomposition of G

1. Every vertex should be in a bag
2. Every edge should be in a bag
3. Bags containing a vertex should form a connected subtree
4. Width = maximum bag size -1

Treewidth



Graph G

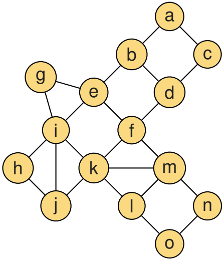


A tree decomposition of G

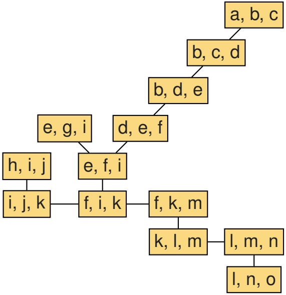
Width = 2

1. Every vertex should be in a bag
2. Every edge should be in a bag
3. Bags containing a vertex should form a connected subtree
4. Width = maximum bag size -1

Treewidth



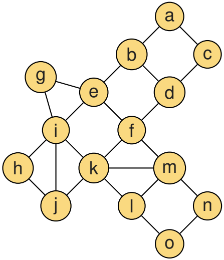
Graph G



A tree decomposition of G
Width = 2

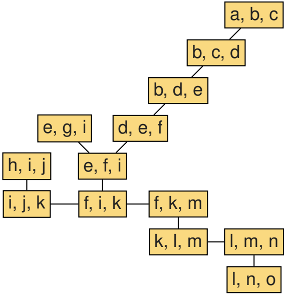
1. Every vertex should be in a bag
2. Every edge should be in a bag
3. Bags containing a vertex should form a connected subtree
4. Width = maximum bag size -1
5. Treewidth of G = minimum width of tree decomposition of G

Treewidth



Graph G

Treewidth 2

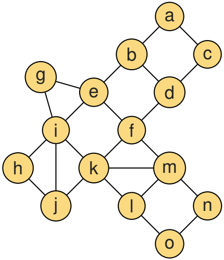


A tree decomposition of G

Width = 2

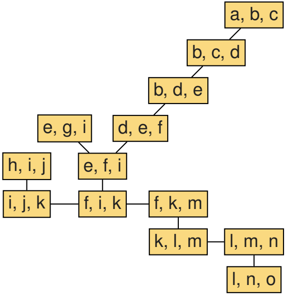
1. Every vertex should be in a bag
2. Every edge should be in a bag
3. Bags containing a vertex should form a connected subtree
4. Width = maximum bag size -1
5. Treewidth of G = minimum width of tree decomposition of G

Treewidth



Graph G

Treewidth 2



A tree decomposition of G

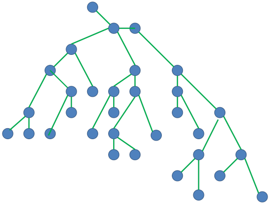
Width = 2

1. Every vertex should be in a bag
2. Every edge should be in a bag
3. Bags containing a vertex should form a connected subtree
4. Width = maximum bag size -1
5. Treewidth of G = minimum width of tree decomposition of G

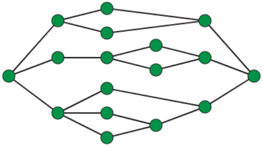
[Robertson & Seymour'84, Arnborg & Proskurowski'89, Bertele & Brioschi'72, Halin'76]

Treewidth of graphs

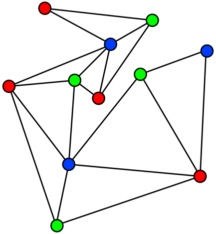
Some graphs of small treewidth:



Trees (tw ≤ 1)



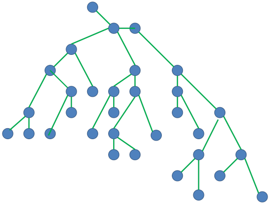
Series-parallel (tw ≤ 2)



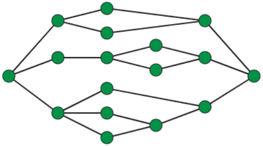
k-outerplanar (tw $\leq 3k - 1$)

Treewidth of graphs

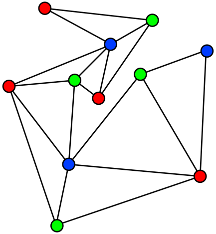
Some graphs of small treewidth:



Trees (tw ≤ 1)

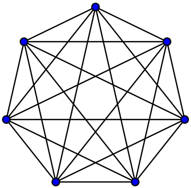


Series-parallel (tw ≤ 2)

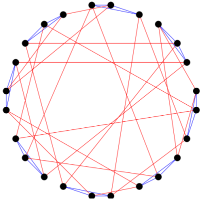


k-outerplanar (tw $\leq 3k-1$)

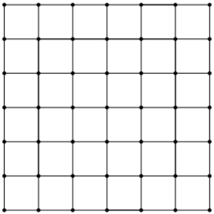
Some graphs of large treewidth:



Clique (tw $= n-1$)



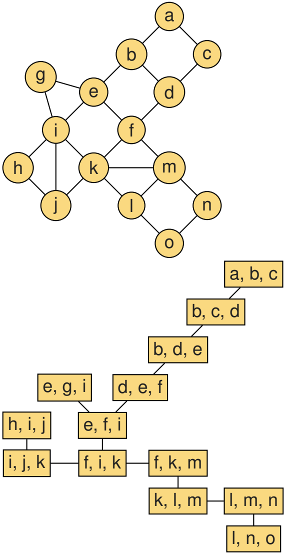
Expanders (tw $= \Theta(n)$)



$n \times m$-grid (tw $= \min(n, m)$)

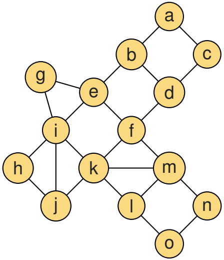
Why treewidth?

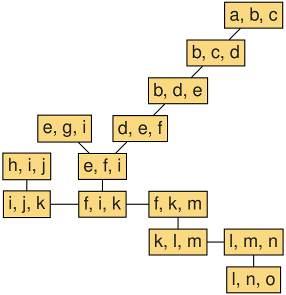
- Algorithms for trees often generalize to algorithms for graphs of small treewidth



Why treewidth?

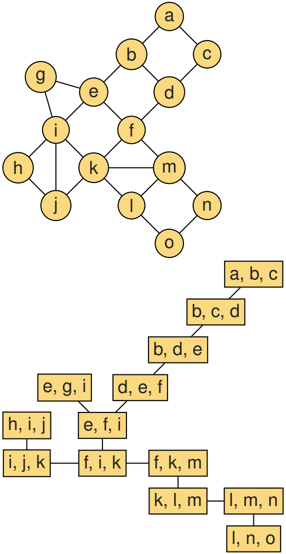
- Algorithms for trees often generalize to algorithms for graphs of small treewidth
- Example: Maximum independent set in $\mathcal{O}(2^k \cdot n)$ time on treewidth-k graphs





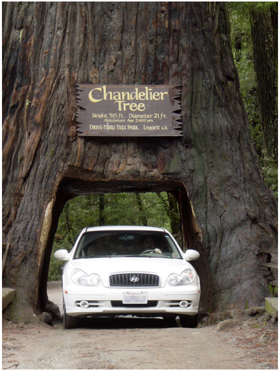
Why treewidth?

- Algorithms for trees often generalize to algorithms for graphs of small treewidth
- Example: Maximum independent set in $\mathcal{O}(2^k \cdot n)$ time on treewidth-k graphs
- Courcelle's theorem gives $f(k) \cdot n$ algorithms for all problems definable in **MSO**-logic



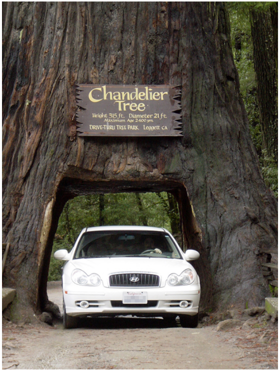
Computing treewidth

- Most algorithms using treewidth need a tree decomposition as an input



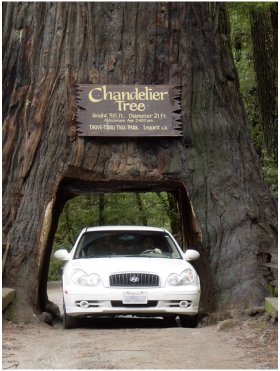
Computing treewidth

- Most algorithms using treewidth need a tree decomposition as an input
- Optimum-width tree decomposition in $2^{\mathcal{O}(k^3)}n$ time [Bodlaender '96]



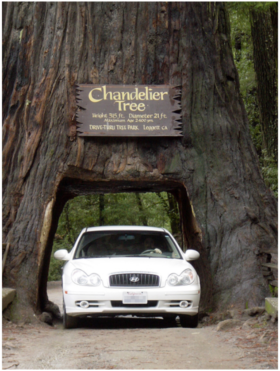
Computing treewidth

- Most algorithms using treewidth need a tree decomposition as an input
- Optimum-width tree decomposition in $2^{\mathcal{O}(k^3)}n$ time [Bodlaender '96]
- 2-approximation in $2^{\mathcal{O}(k)}n$ time [**K.** '21]



Computing treewidth

- Most algorithms using treewidth need a tree decomposition as an input
- Optimum-width tree decomposition in $2^{\mathcal{O}(k^3)}n$ time [Bodlaender '96]
- 2-approximation in $2^{\mathcal{O}(k)}n$ time [**K.** '21]
- $\mathcal{O}(\sqrt{\log k})$-approximation in polynomial-time [Feige, Hajiaghayi & Lee '08]



Dynamic treewidth

Question

Can we maintain tree decompositions of dynamic graphs of bounded treewidth?

Dynamic treewidth

Question

Can we maintain tree decompositions of dynamic graphs of bounded treewidth?

- Would also like to maintain any "finite-state" dynamic programming scheme on the tree decomposition (dynamic Courcelle's theorem)

Dynamic treewidth

Question

Can we maintain tree decompositions of dynamic graphs of bounded treewidth?

- Would also like to maintain any "finite-state" dynamic programming scheme on the tree decomposition (dynamic Courcelle's theorem)

Previous work:

Dynamic treewidth

Question

Can we maintain tree decompositions of dynamic graphs of bounded treewidth?

- Would also like to maintain any "finite-state" dynamic programming scheme on the tree decomposition (dynamic Courcelle's theorem)

Previous work:

- [Bodlaender '93]: $\mathcal{O}(\log n)$ worst-case time for treewidth-2 in the fully dynamic setting

Dynamic treewidth

Question

Can we maintain tree decompositions of dynamic graphs of bounded treewidth?

- Would also like to maintain any "finite-state" dynamic programming scheme on the tree decomposition (dynamic Courcelle's theorem)

Previous work:

- [Bodlaender '93]: $\mathcal{O}(\log n)$ worst-case time for treewidth-2 in the fully dynamic setting and $f(k) \cdot \log n$ for treewidth-k in the decremental setting

Dynamic treewidth

Question

Can we maintain tree decompositions of dynamic graphs of bounded treewidth?

- Would also like to maintain any "finite-state" dynamic programming scheme on the tree decomposition (dynamic Courcelle's theorem)

Previous work:

- [Bodlaender '93]: $\mathcal{O}(\log n)$ worst-case time for treewidth-2 in the fully dynamic setting and $f(k) \cdot \log n$ for treewidth-k in the decremental setting
- [Cohen, Sairam, Tamassia & Vitter '93]: $\mathcal{O}(\log n)$ amortized time for treewidth-3 in the incremental setting

Dynamic treewidth

Question

Can we maintain tree decompositions of dynamic graphs of bounded treewidth?

- Would also like to maintain any "finite-state" dynamic programming scheme on the tree decomposition (dynamic Courcelle's theorem)

Previous work:

- [Bodlaender '93]: $\mathcal{O}(\log n)$ worst-case time for treewidth-2 in the fully dynamic setting and $f(k) \cdot \log n$ for treewidth-k in the decremental setting
- [Cohen, Sairam, Tamassia & Vitter '93]: $\mathcal{O}(\log n)$ amortized time for treewidth-3 in the incremental setting
- [Dvořák, Kupec & Tůma '14]: $f(d)$ worst-case time for treedepth-d

Dynamic treewidth

Question

Can we maintain tree decompositions of dynamic graphs of bounded treewidth?

- Would also like to maintain any "finite-state" dynamic programming scheme on the tree decomposition (dynamic Courcelle's theorem)

Previous work:

- [Bodlaender '93]: $\mathcal{O}(\log n)$ worst-case time for treewidth-2 in the fully dynamic setting and $f(k) \cdot \log n$ for treewidth-k in the decremental setting
- [Cohen, Sairam, Tamassia & Vitter '93]: $\mathcal{O}(\log n)$ amortized time for treewidth-3 in the incremental setting
- [Dvořák, Kupec & Tůma '14]: $f(d)$ worst-case time for treedepth-d
- [Majewski, Pilipczuk & Sokołowski '23]: $f(\ell) \cdot \log n$ amortized time for feedback vertex number ℓ

Dynamic treewidth

Question

Can we maintain tree decompositions of dynamic graphs of bounded treewidth?

- Would also like to maintain any "finite-state" dynamic programming scheme on the tree decomposition (dynamic Courcelle's theorem)

Previous work:

- [Bodlaender '93]: $\mathcal{O}(\log n)$ worst-case time for treewidth-2 in the fully dynamic setting and $f(k) \cdot \log n$ for treewidth-k in the decremental setting
- [Cohen, Sairam, Tamassia & Vitter '93]: $\mathcal{O}(\log n)$ amortized time for treewidth-3 in the incremental setting
- [Dvořák, Kupec & Tůma '14]: $f(d)$ worst-case time for treedepth-d
- [Majewski, Pilipczuk & Sokołowski '23]: $f(\ell) \cdot \log n$ amortized time for feedback vertex number ℓ
- [Goranci, Räcke, Saranurak & Tan '21]: $n^{o(1)}$ amortized time $n^{o(1)}$-approximate tree decomposition on bounded-degree graphs. Not suitable for dynamic programming.

Our result

Summary of previous results

No sublinear dynamic algorithms for maintaining tree decompositions of width $f(k)$ for graphs of treewidth $k \geq 3$ in the fully dynamic setting.

Our result

Summary of previous results

No sublinear dynamic algorithms for maintaining tree decompositions of width $f(k)$ for graphs of treewidth $k \geq 3$ in the fully dynamic setting.

Theorem (this work):

There is data structure that

- is initialized with integer k and empty n-vertex graph G
- supports edge insertions and deletions in amortized time $f(k) \cdot 2^{\sqrt{\log n} \log \log n}$ under the promise that the treewidth of G never exceeds k
- maintains a tree decomposition of G of width at most $6k + 5$

Our result

Summary of previous results

No sublinear dynamic algorithms for maintaining tree decompositions of width $f(k)$ for graphs of treewidth $k \geq 3$ in the fully dynamic setting.

Theorem (this work):

There is data structure that

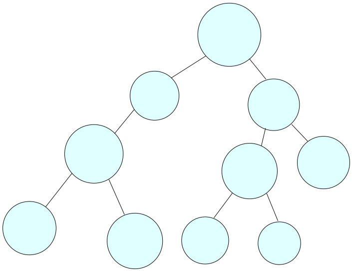
- is initialized with integer k and empty n-vertex graph G
- supports edge insertions and deletions in amortized time $f(k) \cdot 2^{\sqrt{\log n} \log \log n}$ under the promise that the treewidth of G never exceeds k
- maintains a tree decomposition of G of width at most $6k + 5$
- can also maintain any dynamic programming scheme on the decomposition within similar running time (formalized by tree-automata)

The algorithm

High-level plan

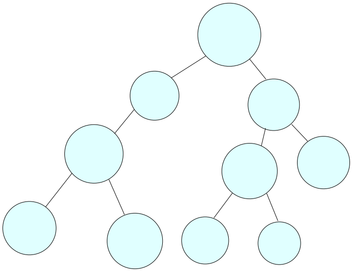
High-level plan

- Goal: Maintain a rooted binary tree decomposition of width $6k+5$ and depth $d = 2^{\mathcal{O}_k(\sqrt{\log n \log \log n})}$



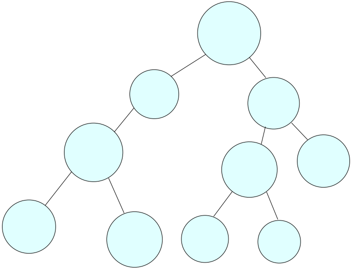
High-level plan

- Goal: Maintain a rooted binary tree decomposition of width $6k+5$ and depth $d = 2^{\mathcal{O}_k(\sqrt{\log n \log \log n})}$
- [Bodlaender & Hagerup '98]: Any tree decomposition of width k can be turned into rooted binary tree decomposition of depth $\mathcal{O}(\log n)$ and width $3k+2$



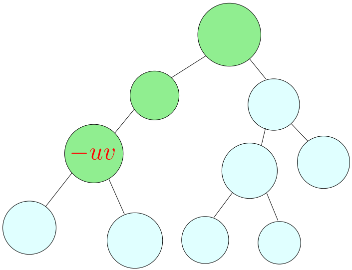
High-level plan

- Goal: Maintain a rooted binary tree decomposition of width $6k + 5$ and depth $d = 2^{\mathcal{O}_k(\sqrt{\log n \log \log n})}$
- [Bodlaender & Hagerup '98]: Any tree decomposition of width k can be turned into rooted binary tree decomposition of depth $\mathcal{O}(\log n)$ and width $3k + 2$
- Maintain also dynamic programming tables directed towards the root



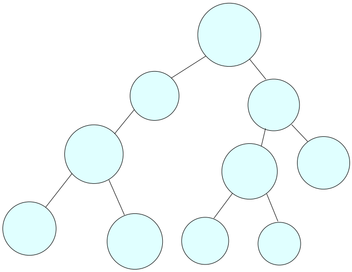
High-level plan

- Goal: Maintain a rooted binary tree decomposition of width $6k+5$ and depth $d = 2^{\mathcal{O}_k(\sqrt{\log n \log\log n})}$
- [Bodlaender & Hagerup '98]: Any tree decomposition of width k can be turned into rooted binary tree decomposition of depth $\mathcal{O}(\log n)$ and width $3k+2$
- Maintain also dynamic programming tables directed towards the root
- Edge deletion: Re-compute dynamic programming tables in time $\mathcal{O}_k(d)$



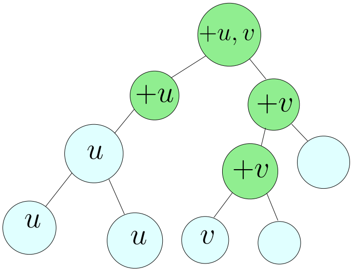
High-level plan

- Goal: Maintain a rooted binary tree decomposition of width $6k+5$ and depth $d = 2^{\mathcal{O}_k(\sqrt{\log n \log \log n})}$
- [Bodlaender & Hagerup '98]: Any tree decomposition of width k can be turned into rooted binary tree decomposition of depth $\mathcal{O}(\log n)$ and width $3k+2$
- Maintain also dynamic programming tables directed towards the root
- Edge deletion: Re-compute dynamic programming tables in time $\mathcal{O}_k(d)$

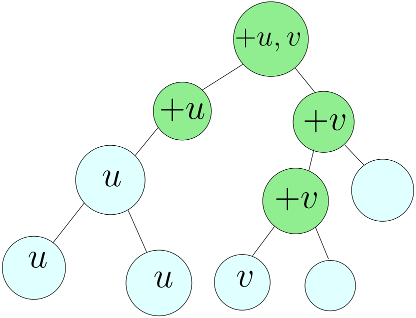


High-level plan

- Goal: Maintain a rooted binary tree decomposition of width $6k+5$ and depth $d = 2^{\mathcal{O}_k(\sqrt{\log n \log\log n})}$
- [Bodlaender & Hagerup '98]: Any tree decomposition of width k can be turned into rooted binary tree decomposition of depth $\mathcal{O}(\log n)$ and width $3k+2$
- Maintain also dynamic programming tables directed towards the root
- Edge deletion: Re-compute dynamic programming tables in time $\mathcal{O}_k(d)$
- Edge insertion: Add u and v to all bags on the path from their subtrees to the root, and re-compute dynamic programming tables in time $\mathcal{O}_k(d)$

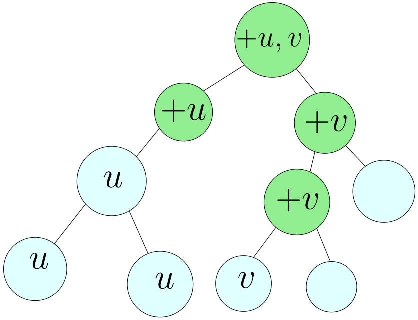


What can go wrong?



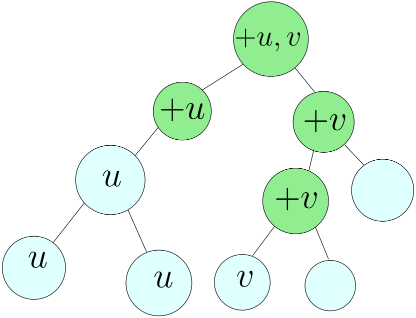
What can go wrong?

- The width can become more than $6k + 5$ on the green bags!



What can go wrong?

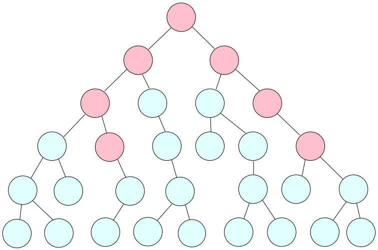
- The width can become more than $6k + 5$ on the green bags!
- Solution: a *Refinement operation* to re-compute the tree decomposition on these bags



Refinement operation

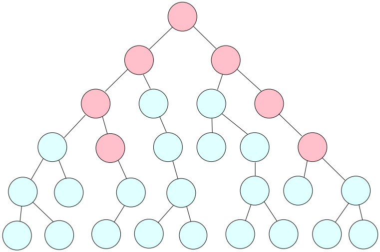
Refinement operation

- Refinement operation is given a *prefix* P of tree decomposition that contains all bags of width $> 6k + 5$



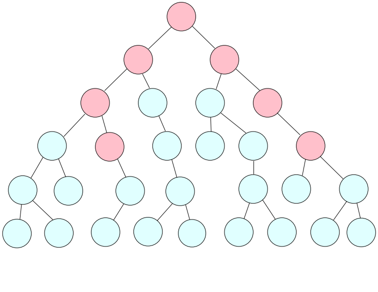
Refinement operation

- Refinement operation is given a *prefix* P of tree decomposition that contains all bags of width $> 6k + 5$
- Re-arranges P into new prefix P' of width $\leq 6k + 5$ and depth $\leq \mathcal{O}(\log n)$



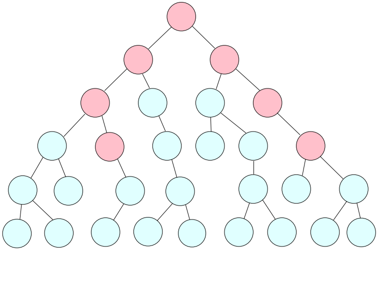
Refinement operation

- Refinement operation is given a *prefix* P of tree decomposition that contains all bags of width $> 6k + 5$
- Re-arranges P into new prefix P' of width $\leq 6k + 5$ and depth $\leq \mathcal{O}(\log n)$



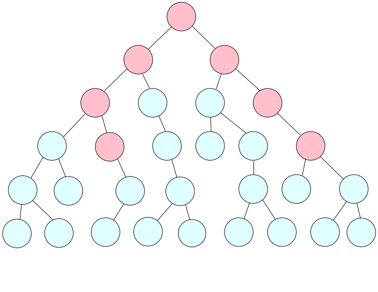
Refinement operation

- Refinement operation is given a *prefix* P of tree decomposition that contains all bags of width $> 6k + 5$
- Re-arranges P into new prefix P' of width $\leq 6k + 5$ and depth $\leq \mathcal{O}(\log n)$
- Changes also other parts of the decomposition, but only improves the width, and the amortized complexity of the operation is $\mathcal{O}_k(|P|)$

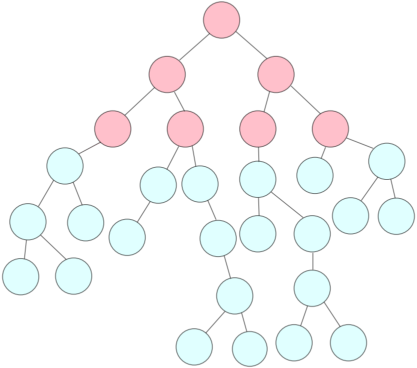


Refinement operation

- Refinement operation is given a *prefix* P of tree decomposition that contains all bags of width $> 6k + 5$
- Re-arranges P into new prefix P' of width $\leq 6k + 5$ and depth $\leq \mathcal{O}(\log n)$
- Changes also other parts of the decomposition, but only improves the width, and the amortized complexity of the operation is $\mathcal{O}_k(|P|)$
- Builds on the improvement operation of [**K.**&Lokshtanov'23], also uses the dealternation lemma of [Bojańczyk&Pilipczuk'22] and Bodlaender-Hagerup-lemma

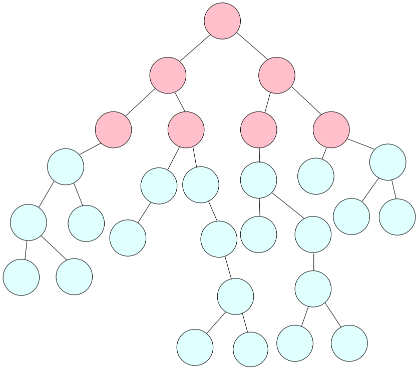


What can go wrong?



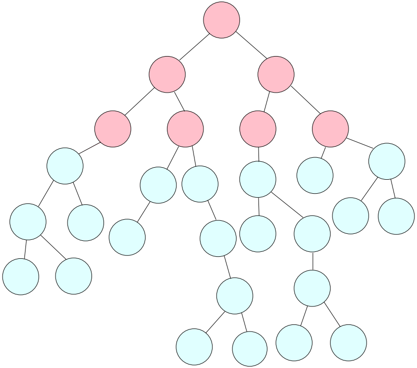
What can go wrong?

- Refinement operation can increase the depth by $\mathcal{O}(\log n)$



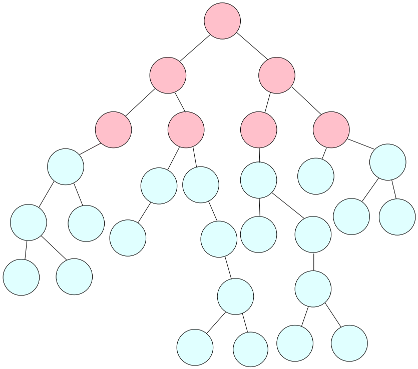
What can go wrong?

- Refinement operation can increase the depth by $\mathcal{O}(\log n)$
- Once depth becomes more than $2^{\mathcal{O}_k(\sqrt{\log n \log \log n})}$, need to reduce it



What can go wrong?

- Refinement operation can increase the depth by $\mathcal{O}(\log n)$
- Once depth becomes more than $2^{\mathcal{O}_k(\sqrt{\log n \log \log n})}$, need to reduce it
- Solution: A depth-reduction scheme by using the refinement operation and a potential function



Depth-reduction scheme

Depth-reduction scheme

- Potential function of form $\Phi(T) = \sum_{t \in V(T)} k^{10 \cdot |\mathrm{bag}(t)|} \cdot \mathrm{height}(t)$

Depth-reduction scheme

- Potential function of form $\Phi(T) = \sum_{t \in V(T)} k^{10 \cdot |\text{bag}(t)|} \cdot \text{height}(t)$
- The $k^{10 \cdot |\text{bag}(t)|}$ factor is for amortized analysis of the refinement, the $\text{height}(t)$ factor for depth-reduction

Depth-reduction scheme

- Potential function of form $\Phi(T) = \sum_{t \in V(T)} k^{10 \cdot |\mathrm{bag}(t)|} \cdot \mathrm{height}(t)$
- The $k^{10 \cdot |\mathrm{bag}(t)|}$ factor is for amortized analysis of the refinement, the $\mathrm{height}(t)$ factor for depth-reduction
- Edge insertion increses potential by $\mathcal{O}_k(d^2) = 2^{\mathcal{O}_k(\sqrt{\log n \log \log n})}$

Depth-reduction scheme

- Potential function of form $\Phi(T) = \sum_{t \in V(T)} k^{10 \cdot |\text{bag}(t)|} \cdot \text{height}(t)$
- The $k^{10 \cdot |\text{bag}(t)|}$ factor is for amortized analysis of the refinement, the $\text{height}(t)$ factor for depth-reduction
- Edge insertion increses potential by $\mathcal{O}_k(d^2) = 2^{\mathcal{O}_k(\sqrt{\log n \log \log n})}$
- Wish to argue that if depth is more than $2^{\mathcal{O}_k(\sqrt{\log n \log \log n})}$, then exists prefix P s.t.
  - refining on P produces decomposition T' with $\Phi(T') < \Phi(T)$ and
  - runs in time $\mathcal{O}_k(\Phi(T) - \Phi(T'))$

Depth-reduction scheme

- Potential function of form $\Phi(T) = \sum_{t \in V(T)} k^{10 \cdot |\text{bag}(t)|} \cdot \text{height}(t)$
- The $k^{10 \cdot |\text{bag}(t)|}$ factor is for amortized analysis of the refinement, the $\text{height}(t)$ factor for depth-reduction
- Edge insertion increses potential by $\mathcal{O}_k(d^2) = 2^{\mathcal{O}_k(\sqrt{\log n \log \log n})}$
- Wish to argue that if depth is more than $2^{\mathcal{O}_k(\sqrt{\log n \log \log n})}$, then exists prefix P s.t.
  - refining on P produces decomposition T' with $\Phi(T') < \Phi(T)$ and
  - runs in time $\mathcal{O}_k(\Phi(T) - \Phi(T'))$

$\Rightarrow$ Can control the height in amortized $2^{\mathcal{O}_k(\sqrt{\log n \log \log n})}$ time

Lemma on trees

Lemma

Let $c \geq 2$ and T be a binary tree width n nodes. If the depth of T is at least $2^{\Omega(\sqrt{\log n \log c})}$ then there exists a prefix P of T so that

$$c \cdot \left(|P| + \sum_{t \in \mathrm{App}(P)} \mathrm{height}(t) \right) < \sum_{t \in P} \mathrm{height}(t),$$

where $\mathrm{App}(P)$ is the set of nodes not in P but with parent in P.

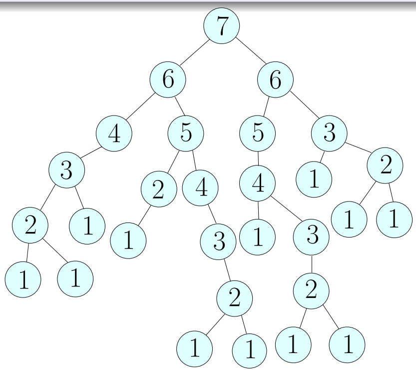
Lemma on trees

Lemma

Let $c \geq 2$ and T be a binary tree width n nodes. If the depth of T is at least $2^{\Omega(\sqrt{\log n \log c})}$ then there exists a prefix P of T so that

$$c \cdot \left(|P| + \sum_{t \in \mathrm{App}(P)} \mathrm{height}(t) \right) < \sum_{t \in P} \mathrm{height}(t),$$

where App(P) is the set of nodes not in P but with parent in P.



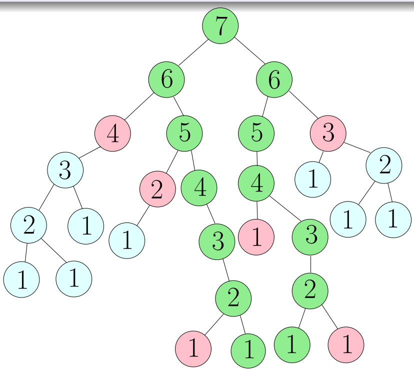
Lemma on trees

Lemma

Let $c \geq 2$ and T be a binary tree width n nodes. If the depth of T is at least $2^{\Omega(\sqrt{\log n \log c})}$ then there exists a prefix P of T so that

$$c \cdot \left(|P| + \sum_{t \in \mathrm{App}(P)} \mathrm{height}(t) \right) < \sum_{t \in P} \mathrm{height}(t),$$

where App(P) is the set of nodes not in P but with parent in P.



Conclusion

- $\mathcal{O}_k(2^{\sqrt{\log n} \log \log n})$ amortized update time for maintaining a tree decomposition of width at most $6k + 5$ of dynamic graph of treewidth $\leq k$

Conclusion

- $\mathcal{O}_k(2^{\sqrt{\log n}\log\log n})$ amortized update time for maintaining a tree decomposition of width at most $6k+5$ of dynamic graph of treewidth $\leq k$
  - Can also maintain any dynamic programming on the tree decomposition

Conclusion

- $\mathcal{O}_k(2^{\sqrt{\log n} \log \log n})$ amortized update time for maintaining a tree decomposition of width at most $6k + 5$ of dynamic graph of treewidth $\leq k$
  - Can also maintain any dynamic programming on the tree decomposition
- Follow up works:

Conclusion

- $\mathcal{O}_k(2^{\sqrt{\log n}\log\log n})$ amortized update time for maintaining a tree decomposition of width at most $6k+5$ of dynamic graph of treewidth $\leq k$
  - Can also maintain any dynamic programming on the tree decomposition
- Follow up works:
  - [**K.**, Nadara, Pilipczuk & Sokołowski '24]: Dynamic Baker's scheme: Dynamic $f(\varepsilon) \cdot n^{o(1)}$ time $(1+\varepsilon)$-approximations on planar graphs

Conclusion

- $\mathcal{O}_k(2^{\sqrt{\log n}\log\log n})$ amortized update time for maintaining a tree decomposition of width at most $6k+5$ of dynamic graph of treewidth $\leq k$
  - Can also maintain any dynamic programming on the tree decomposition
- Follow up works:
  - [**K.**, Nadara, Pilipczuk & Sokołowski '24]: Dynamic Baker's scheme: Dynamic $f(\varepsilon) \cdot n^{o(1)}$ time $(1+\varepsilon)$-approximations on planar graphs
  - [**K.** & Sokołowski '24+]: Dynamic rankwidth: Generalization of dynamic treewidth to a "dense" generalization of treewidth called rankwidth

Conclusion

- $\mathcal{O}_k(2^{\sqrt{\log n}\log\log n})$ amortized update time for maintaining a tree decomposition of width at most $6k+5$ of dynamic graph of treewidth $\leq k$
  - Can also maintain any dynamic programming on the tree decomposition
- Follow up works:
  - [**K.**, Nadara, Pilipczuk & Sokołowski '24]: Dynamic Baker's scheme: Dynamic $f(\varepsilon) \cdot n^{o(1)}$ time $(1+\varepsilon)$-approximations on planar graphs
  - [**K.** & Sokołowski '24+]: Dynamic rankwidth: Generalization of dynamic treewidth to a "dense" generalization of treewidth called rankwidth
- Open problems and directions:

Conclusion

- $\mathcal{O}_k(2^{\sqrt{\log n}\log\log n})$ amortized update time for maintaining a tree decomposition of width at most $6k+5$ of dynamic graph of treewidth $\leq k$
  - Can also maintain any dynamic programming on the tree decomposition
- Follow up works:
  - [**K.**, Nadara, Pilipczuk & Sokołowski '24]: Dynamic Baker's scheme: Dynamic $f(\varepsilon) \cdot n^{o(1)}$ time $(1+\varepsilon)$-approximations on planar graphs
  - [**K.** & Sokołowski '24+]: Dynamic rankwidth: Generalization of dynamic treewidth to a "dense" generalization of treewidth called rankwidth
- Open problems and directions:
  - Improve to $\mathcal{O}_k(\text{poly}\log n)$

Conclusion

- $\mathcal{O}_k(2^{\sqrt{\log n}\log\log n})$ amortized update time for maintaining a tree decomposition of width at most $6k+5$ of dynamic graph of treewidth $\leq k$
  - Can also maintain any dynamic programming on the tree decomposition
- Follow up works:
  - [**K.**, Nadara, Pilipczuk & Sokołowski '24]: Dynamic Baker's scheme: Dynamic $f(\varepsilon) \cdot n^{o(1)}$ time $(1+\varepsilon)$-approximations on planar graphs
  - [**K.** & Sokołowski '24+]: Dynamic rankwidth: Generalization of dynamic treewidth to a "dense" generalization of treewidth called rankwidth
- Open problems and directions:
  - Improve to $\mathcal{O}_k(\text{poly}\log n)$
  - More applications, for example: Dynamic k-DISJOINT PATHS on planar graphs?

Conclusion

- $\mathcal{O}_k(2^{\sqrt{\log n}\log\log n})$ amortized update time for maintaining a tree decomposition of width at most $6k+5$ of dynamic graph of treewidth $\leq k$
  - Can also maintain any dynamic programming on the tree decomposition
- Follow up works:
  - [**K.**, Nadara, Pilipczuk & Sokołowski '24]: Dynamic Baker's scheme: Dynamic $f(\varepsilon) \cdot n^{o(1)}$ time $(1+\varepsilon)$-approximations on planar graphs
  - [**K.** & Sokołowski '24+]: Dynamic rankwidth: Generalization of dynamic treewidth to a "dense" generalization of treewidth called rankwidth
- Open problems and directions:
  - Improve to $\mathcal{O}_k(\text{poly}\log n)$
  - More applications, for example: Dynamic k-DISJOINT PATHS on planar graphs?

Thank you!