Dynamic Treewidth in Logarithmic Time

Tuukka Korhonen

IBS DIMAG seminar

9 December 2025

Dynamic graph algorithms

Dynamic graph algorithms

- Setting: Design a data structure that maintains a graph G and supports the following operations:

Dynamic graph algorithms

- Setting: Design a data structure that maintains a graph G and supports the following operations:
  1. Initialize(n): Create G as an edgeless n-vertex graph
  2. Insert(u, v): Insert an edge between u and v
  3. Delete(u, v): Delete an edge between u and v
  4. Query: Ask something about the graph G

Dynamic graph algorithms

- Setting: Design a data structure that maintains a graph G and supports the following operations:
  1. Initialize(n): Create G as an edgeless n-vertex graph
  2. Insert(u, v): Insert an edge between u and v
  3. Delete(u, v): Delete an edge between u and v
  4. Query: Ask something about the graph G

Question

Can we support the operations faster than by re-computing from scratch every time?

Dynamic graph algorithms

- Setting: Design a data structure that maintains a graph G and supports the following operations:
  1. Initialize(n): Create G as an edgeless n-vertex graph
  2. Insert(u, v): Insert an edge between u and v
  3. Delete(u, v): Delete an edge between u and v
  4. Query: Ask something about the graph G

Question

Can we support the operations faster than by re-computing from scratch every time?

Example: Connectivity (Query: Are s and t in the same connected component?)

Dynamic graph algorithms

- Setting: Design a data structure that maintains a graph G and supports the following operations:
  1. Initialize(n): Create G as an edgeless n-vertex graph
  2. Insert(u, v): Insert an edge between u and v
  3. Delete(u, v): Delete an edge between u and v
  4. Query: Ask something about the graph G

Question

Can we support the operations faster than by re-computing from scratch every time?

Example: Connectivity (Query: Are s and t in the same connected component?)

1. Naive: $\mathcal{O}(m)$ worst-case time per operation

Dynamic graph algorithms

- Setting: Design a data structure that maintains a graph G and supports the following operations:
  1. Initialize(n): Create G as an edgeless n-vertex graph
  2. Insert(u, v): Insert an edge between u and v
  3. Delete(u, v): Delete an edge between u and v
  4. Query: Ask something about the graph G

Question

Can we support the operations faster than by re-computing from scratch every time?

Example: Connectivity (Query: Are s and t in the same connected component?)

1. Naive: $\mathcal{O}(m)$ worst-case time per operation
2. Union-find: $\mathcal{O}(\alpha(n))$ amortized time per operation, but deletions not allowed [Tarjan'75]

Dynamic graph algorithms

- Setting: Design a data structure that maintains a graph G and supports the following operations:
  1. Initialize(n): Create G as an edgeless n-vertex graph
  2. Insert(u, v): Insert an edge between u and v
  3. Delete(u, v): Delete an edge between u and v
  4. Query: Ask something about the graph G

Question

Can we support the operations faster than by re-computing from scratch every time?

Example: Connectivity (Query: Are s and t in the same connected component?)

1. Naive: $\mathcal{O}(m)$ worst-case time per operation
2. Union-find: $\mathcal{O}(\alpha(n))$ amortized time per operation, but deletions not allowed [Tarjan'75]
3. Link/cut tree: $\mathcal{O}(\log n)$ amortized time when G is a forest [Sleator&Tarjan'81]

Dynamic graph algorithms

- Setting: Design a data structure that maintains a graph G and supports the following operations:
  1. Initialize(n): Create G as an edgeless n-vertex graph
  2. Insert(u, v): Insert an edge between u and v
  3. Delete(u, v): Delete an edge between u and v
  4. Query: Ask something about the graph G

Question

Can we support the operations faster than by re-computing from scratch every time?

Example: Connectivity (Query: Are s and t in the same connected component?)

1. Naive: $\mathcal{O}(m)$ worst-case time per operation
2. Union-find: $\mathcal{O}(\alpha(n))$ amortized time per operation, but deletions not allowed [Tarjan'75]
3. Link/cut tree: $\mathcal{O}(\log n)$ amortized time when G is a forest [Sleator&Tarjan'81] (optimal)

Dynamic graph algorithms

- Setting: Design a data structure that maintains a graph G and supports the following operations:
  1. Initialize(n): Create G as an edgeless n-vertex graph
  2. Insert(u, v): Insert an edge between u and v
  3. Delete(u, v): Delete an edge between u and v
  4. Query: Ask something about the graph G

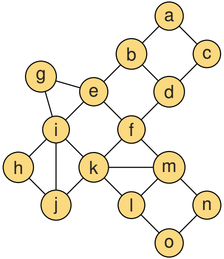
Question

Can we support the operations faster than by re-computing from scratch every time?

Example: Connectivity (Query: Are s and t in the same connected component?)

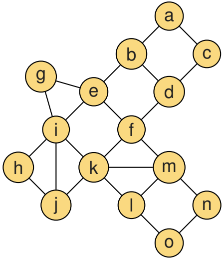
1. Naive: $\mathcal{O}(m)$ worst-case time per operation
2. Union-find: $\mathcal{O}(\alpha(n))$ amortized time per operation, but deletions not allowed [Tarjan'75]
3. Link/cut tree: $\mathcal{O}(\log n)$ amortized time when G is a forest [Sleator&Tarjan'81] (optimal)
4. [Henzinger&King'99]: $\mathcal{O}(\log^3 n)$ amortized time

Treewidth

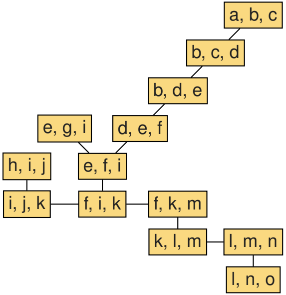


Graph G

Treewidth

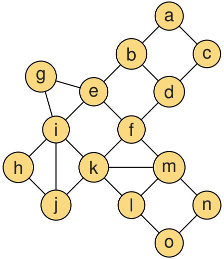


Graph G

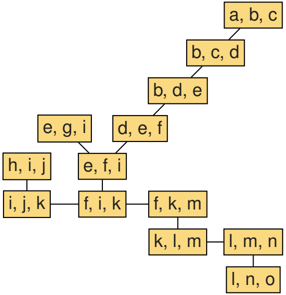


A tree decomposition of G

Treewidth



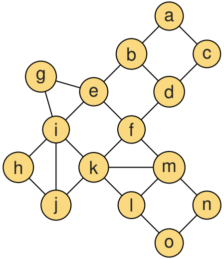
Graph G



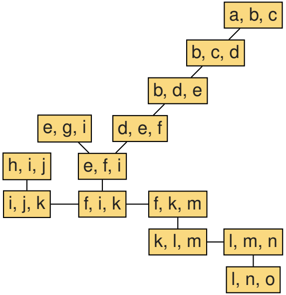
A tree decomposition of G

1. Every vertex should be in a bag

Treewidth



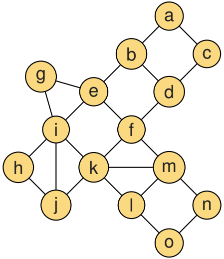
Graph G



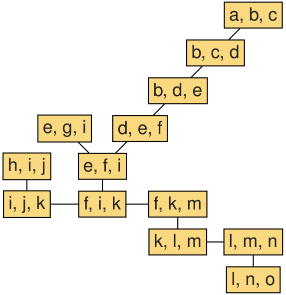
A tree decomposition of G

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

Treewidth



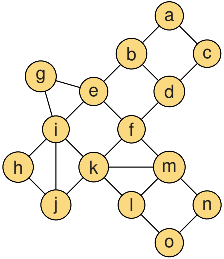
Graph G



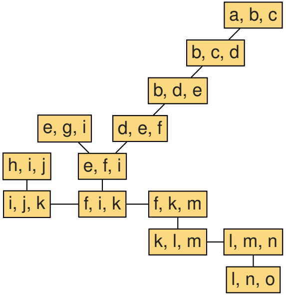
A tree decomposition of G

1. Every vertex should be in a bag
2. Every edge should be in a bag
3. For every vertex v, the bags containing v should form a connected subtree

Treewidth



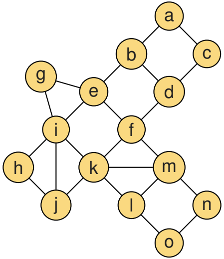
Graph G



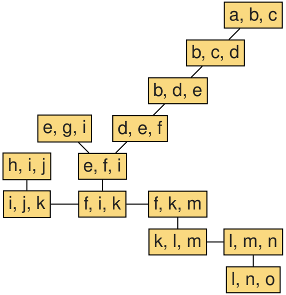
A tree decomposition of G

1. Every vertex should be in a bag
2. Every edge should be in a bag
3. For every vertex v, the bags containing v should form a connected subtree
4. Width = maximum bag size -1

Treewidth



Graph G

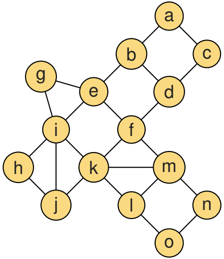


A tree decomposition of G

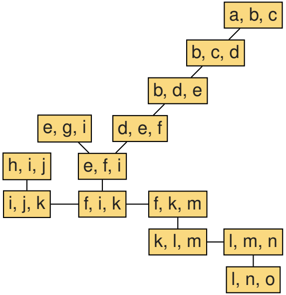
Width = 2

1. Every vertex should be in a bag
2. Every edge should be in a bag
3. For every vertex v, the bags containing v should form a connected subtree
4. Width = maximum bag size -1

Treewidth



Graph G

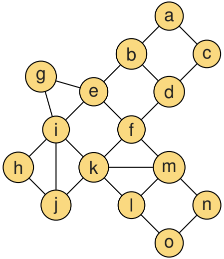


A tree decomposition of G

Width = 2

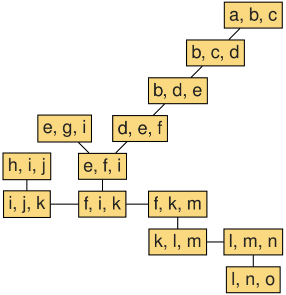
1. Every vertex should be in a bag
2. Every edge should be in a bag
3. For every vertex v, the bags containing v should form a connected subtree
4. Width = maximum bag size -1
5. Treewidth of G = minimum width of tree decomposition of G

Treewidth



Graph G

Treewidth 2

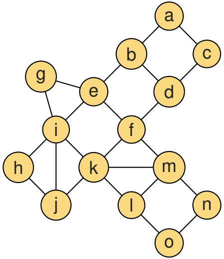


A tree decomposition of G

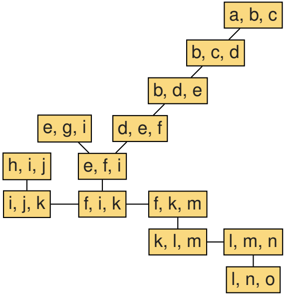
Width = 2

1. Every vertex should be in a bag
2. Every edge should be in a bag
3. For every vertex v, the bags containing v 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



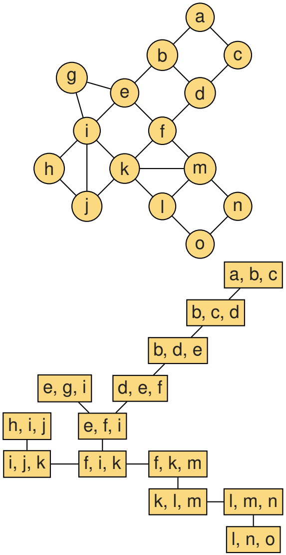
A tree decomposition of G
Width = 2

1. Every vertex should be in a bag
2. Every edge should be in a bag
3. For every vertex v, the bags containing v 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]

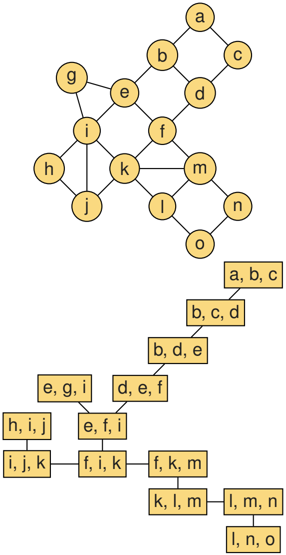
Why treewidth?

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



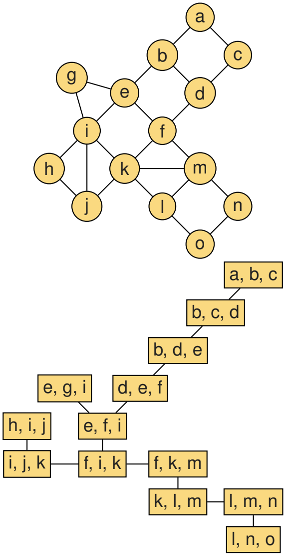
Why treewidth?

- 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



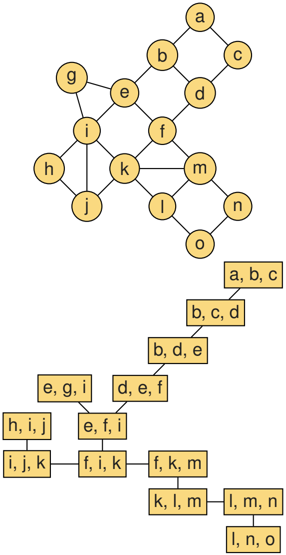
Why treewidth?

- 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 $f(k) \cdot n$ time algorithms for all problems definable in **MSO**-logic



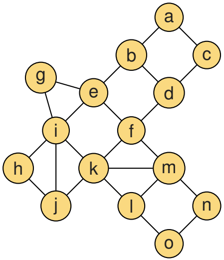
Why treewidth?

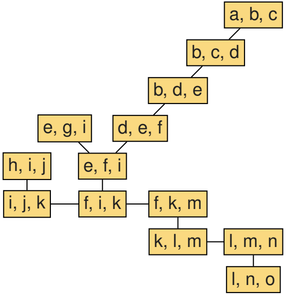
- Algorithms for trees often generalize to algorithms for graphs of small treewidth
- Example: Maximum independent set in $\mathcal{O}(2^k \cdot n)$ time on treewidth-k graphs
- Courcelle's theorem gives $f(k) \cdot n$ time algorithms for all problems definable in **MSO**-logic
- Need the tree decomposition!



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 $f(k) \cdot n$ time algorithms for all problems definable in **MSO**-logic
- Need the tree decomposition!
  - $2^{\mathcal{O}(k^3)} n$ time algorithm to compute an optimum-width tree decomposition [Bodlaender '96]
  - $2^{\mathcal{O}(k)} n$ time for 2-approximation [K. '21]
  - $n^{\mathcal{O}(1)}$ time for $\mathcal{O}(\sqrt{\log k})$-approximation [Feige,Hajiaghayi,Lee'08]





Dynamic treewidth

Question [Bodlaender '93, Dvořák, Kupec & Tůma '14, Alman, Mnich & Vassilevska Williams '20]

Can we efficiently maintain a tree decomposition of a dynamic graph with bounded treewidth?

Dynamic treewidth

Question [Bodlaender ’93, Dvořák, Kupec & Tůma ’14, Alman, Mnich & Vassilevska Williams ’20]

Can we efficiently maintain a tree decomposition of a dynamic graph with bounded treewidth?

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

Dynamic treewidth

Question [Bodlaender '93, Dvořák, Kupec & Tůma '14, Alman, Mnich & Vassilevska Williams '20]

Can we efficiently maintain a tree decomposition of a dynamic graph with 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 [Bodlaender '93, Dvořák, Kupec & Tůma '14, Alman, Mnich & Vassilevska Williams '20]

Can we efficiently maintain a tree decomposition of a dynamic graph with bounded treewidth?

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

Previous work:

- Treewidth-1 (dynamic forests): $\mathcal{O}(\log n)$ update time [Sleator & Tarjan'83, Frederickson'85,97, Alstrup, Holm, de Lichtenberg, Thorup'05...]

Dynamic treewidth

Question [Bodlaender '93, Dvořák, Kupec & Tůma '14, Alman, Mnich & Vassilevska Williams '20]

Can we efficiently maintain a tree decomposition of a dynamic graph with bounded treewidth?

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

Previous work:

- Treewidth-1 (dynamic forests): $\mathcal{O}(\log n)$ update time [Sleator & Tarjan'83, Frederickson'85,97, Alstrup, Holm, de Lichtenberg, Thorup'05...]
- Treewidth-2: $\mathcal{O}(\log n)$ update time [Bodlaender'93]

Dynamic treewidth

Question [Bodlaender '93, Dvořák, Kupec & Tůma '14, Alman, Mnich & Vassilevska Williams '20]

Can we efficiently maintain a tree decomposition of a dynamic graph with bounded treewidth?

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

Previous work:

- Treewidth-1 (dynamic forests): $\mathcal{O}(\log n)$ update time [Sleator & Tarjan'83, Frederickson'85,97, Alstrup, Holm, de Lichtenberg, Thorup'05...]
- Treewidth-2: $\mathcal{O}(\log n)$ update time [Bodlaender'93] (writeup missing details)

Dynamic treewidth

Question [Bodlaender '93, Dvořák, Kupec & Tůma '14, Alman, Mnich & Vassilevska Williams '20]

Can we efficiently maintain a tree decomposition of a dynamic graph with bounded treewidth?

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

Previous work:

- Treewidth-1 (dynamic forests): $\mathcal{O}(\log n)$ update time [Sleator & Tarjan'83, Frederickson'85,97, Alstrup, Holm, de Lichtenberg, Thorup'05...]
- Treewidth-2: $\mathcal{O}(\log n)$ update time [Bodlaender'93] (writeup missing details)
- Treewidth-k: $n^{o(1)}$ amortized update time $n^{o(1)}$-approximate tree decomposition on bounded-degree graphs. [Goranci, Räcke, Saranurak, Tan '21]

Dynamic treewidth

Question [Bodlaender '93, Dvořák, Kupec & Tůma '14, Alman, Mnich & Vassilevska Williams '20]

Can we efficiently maintain a tree decomposition of a dynamic graph with bounded treewidth?

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

Previous work:

- Treewidth-1 (dynamic forests): $\mathcal{O}(\log n)$ update time [Sleator & Tarjan'83, Frederickson'85,97, Alstrup, Holm, de Lichtenberg, Thorup'05...]
- Treewidth-2: $\mathcal{O}(\log n)$ update time [Bodlaender'93] (writeup missing details)
- Treewidth-k: $n^{o(1)}$ amortized update time $n^{o(1)}$-approximate tree decomposition on bounded-degree graphs. [Goranci, Räcke, Saranurak, Tan '21] (not suitable for dynamic Courcelle's theorem)

Dynamic treewidth

Question [Bodlaender '93, Dvořák, Kupec & Tůma '14, Alman, Mnich & Vassilevska Williams '20]

Can we efficiently maintain a tree decomposition of a dynamic graph with bounded treewidth?

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

Previous work:

- Treewidth-1 (dynamic forests): $\mathcal{O}(\log n)$ update time [Sleator & Tarjan'83, Frederickson'85,97, Alstrup, Holm, de Lichtenberg, Thorup'05...]
- Treewidth-2: $\mathcal{O}(\log n)$ update time [Bodlaender'93] (writeup missing details)
- Treewidth-k: $n^{o(1)}$ amortized update time $n^{o(1)}$-approximate tree decomposition on bounded-degree graphs. [Goranci, Räcke, Saranurak, Tan '21] (not suitable for dynamic Courcelle's theorem)
- Treewidth-k: $2^{k^{\mathcal{O}(1)}} n^{o(1)}$ amortized update time 6-approximate tree decomposition. [K., Majewski, Nadara, Pilipczuk, Sokołowski '23]

Dynamic treewidth

Question [Bodlaender '93, Dvořák, Kupec & Tůma '14, Alman, Mnich & Vassilevska Williams '20]

Can we efficiently maintain a tree decomposition of a dynamic graph with bounded treewidth?

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

Previous work:

- Treewidth-1 (dynamic forests): $\mathcal{O}(\log n)$ update time [Sleator & Tarjan'83, Frederickson'85,97, Alstrup, Holm, de Lichtenberg, Thorup'05...]
- Treewidth-2: $\mathcal{O}(\log n)$ update time [Bodlaender'93] (writeup missing details)
- Treewidth-k: $n^{o(1)}$ amortized update time $n^{o(1)}$-approximate tree decomposition on bounded-degree graphs. [Goranci, Räcke, Saranurak, Tan '21] (not suitable for dynamic Courcelle's theorem)
- Treewidth-k: $2^{k^{\mathcal{O}(1)}} n^{o(1)}$ amortized update time 6-approximate tree decomposition. [K., Majewski, Nadara, Pilipczuk, Sokołowski '23]

Theorem (This work)

$2^{\mathcal{O}(k)} \log n$ amortized update time 9-approximate tree decomposition.

Theorem Statement

Theorem (this work):

Theorem Statement

Theorem (this work):

There is data structure that

- is initialized with integer k and an edgeless n-vertex graph G
- supports edge insertions/deletions in amortized time $2^{\mathcal{O}(k)} \log n$ under the promise that $\text{tw}(G) \leq k$
- maintains a tree decomposition of G of width at most $9 \cdot \text{tw}(G) + 8$

Theorem Statement

Theorem (this work):

There is data structure that

- is initialized with integer k and an edgeless n-vertex graph G
- supports edge insertions/deletions in amortized time $2^{\mathcal{O}(k)} \log n$ under the promise that $\mathrm{tw}(G) \leq k$
- maintains a tree decomposition of G of width at most $9 \cdot \mathrm{tw}(G) + 8$
- can also maintain any dynamic programming scheme on the decomposition within similar running time (formalized by tree decomposition automata)

Theorem Statement

Theorem (this work):

There is data structure that

- is initialized with integer k and an edgeless n-vertex graph G
- supports edge insertions/deletions in amortized time $2^{\mathcal{O}(k)} \log n$ under the promise that $\mathrm{tw}(G) \leq k$
- maintains a tree decomposition of G of width at most $9 \cdot \mathrm{tw}(G) + 8$
- can also maintain any dynamic programming scheme on the decomposition within similar running time (formalized by tree decomposition automata)

$\Rightarrow$ Dynamic Courcelle's theorem in $f(k) \cdot \log n$ amortized update time

Applications

Applications

- $f(k) \cdot m^{1+o(1)}$ time algorithm for k-disjoint paths and H-minor-containment [K., Pilipczuk, Stamoulis, '24]

Applications

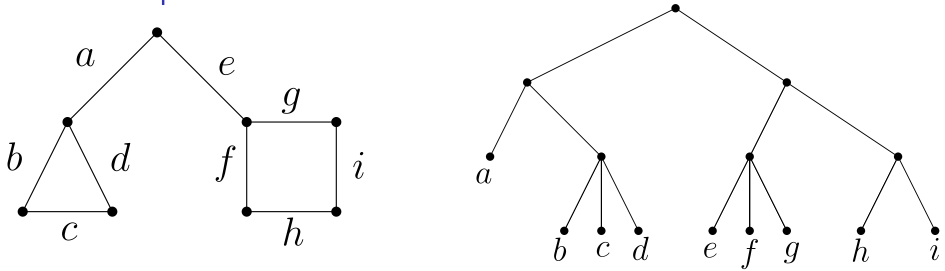
- $f(k) \cdot m^{1+o(1)}$ time algorithm for k-disjoint paths and H-minor-containment [K., Pilipczuk, Stamoulis, '24]
- Dynamic rankwidth $\Rightarrow$ rankwidth in $f(k) \cdot n^{1+o(1)} + \mathcal{O}(m)$ time [K., Sokolowski, '24]

Applications

- $f(k) \cdot m^{1+o(1)}$ time algorithm for k-disjoint paths and H-minor-containment [K., Pilipczuk, Stamoulis, '24]
- Dynamic rankwidth $\Rightarrow$ rankwidth in $f(k) \cdot n^{1+o(1)} + \mathcal{O}(m)$ time [K., Sokolowski, '24]
- Dynamic kernelization with $\mathcal{O}(\log n)$ amortized update time, e.g., for planar dominating set [Bertram, Haun, Jensen, K., '25]

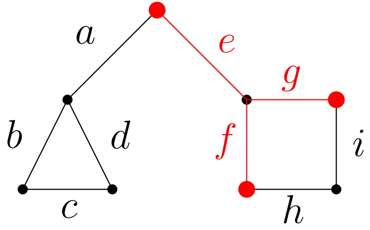
The algorithm

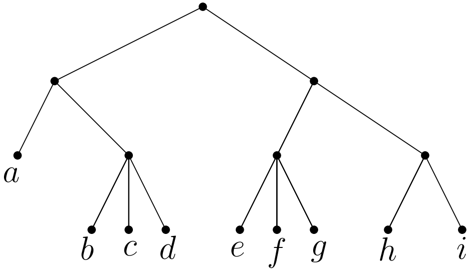
Maintained decomposition



- Branch decomposition: Rooted tree whose leaves correspond to the edges of the graph

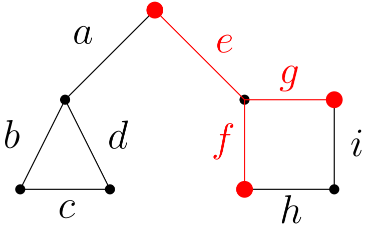
Maintained decomposition

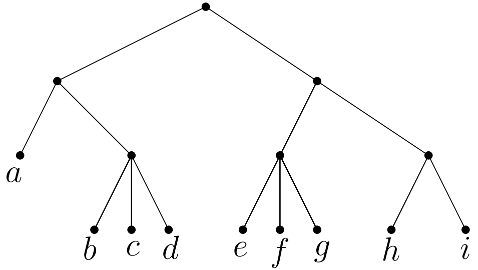




- Branch decomposition: Rooted tree whose leaves correspond to the edges of the graph
- Boundary $\partial(F)$ of a set of edges $F \subseteq E$: The vertices incident to edges from both F and $E \setminus F$.

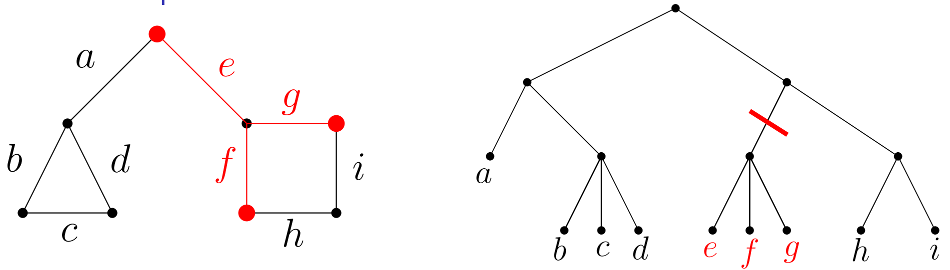
Maintained decomposition





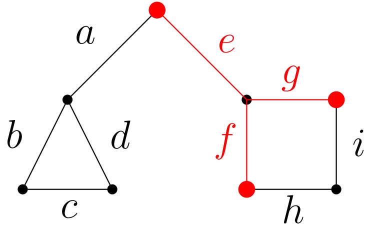
- Branch decomposition: Rooted tree whose leaves correspond to the edges of the graph
- Boundary $\partial(F)$ of a set of edges $F \subseteq E$: The vertices incident to edges from both F and $E \setminus F$.
- A set of edges $F \subseteq E$ is well-linked if it cannot be partitioned to (C_1, C_2) so that $|\partial(C_1)| < |\partial(F)|$ and $|\partial(C_2)| < |\partial(F)|$

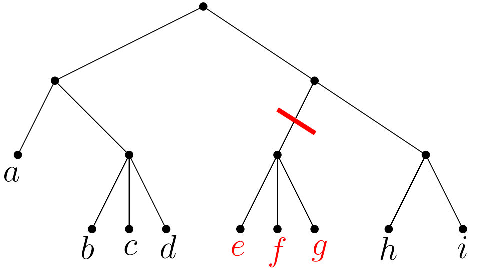
Maintained decomposition



- Branch decomposition: Rooted tree whose leaves correspond to the edges of the graph
- Boundary $\partial(F)$ of a set of edges $F \subseteq E$: The vertices incident to edges from both F and $E \setminus F$.
- A set of edges $F \subseteq E$ is well-linked if it cannot be partitioned to (C_1, C_2) so that $|\partial(C_1)| < |\partial(F)|$ and $|\partial(C_2)| < |\partial(F)|$
- Want to maintain: Every edge set corresponding to a subtree is well-linked “downwards well-linkedness”

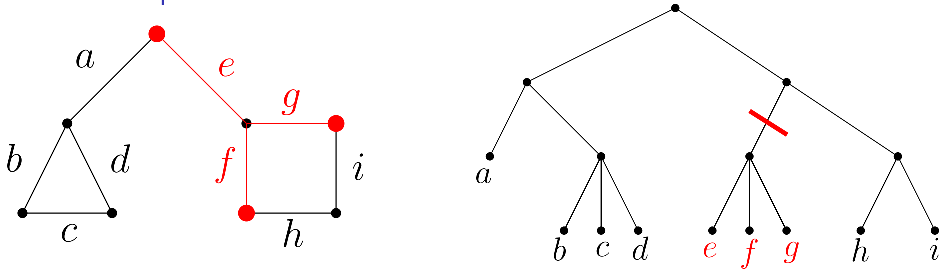
Maintained decomposition





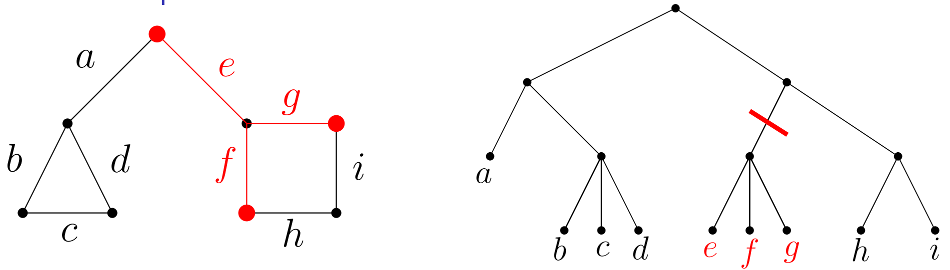
- Branch decomposition: Rooted tree whose leaves correspond to the edges of the graph
- Boundary $\partial(F)$ of a set of edges $F \subseteq E$: The vertices incident to edges from both F and $E \setminus F$.
- A set of edges $F \subseteq E$ is well-linked if it cannot be partitioned to (C_1, C_2) so that $|\partial(C_1)| < |\partial(F)|$ and $|\partial(C_2)| < |\partial(F)|$
- Want to maintain: Every edge set corresponding to a subtree is well-linked "downwards well-linkedness"
  $\Rightarrow$ Boundaries have size $\mathcal{O}(k)$

Maintained decomposition



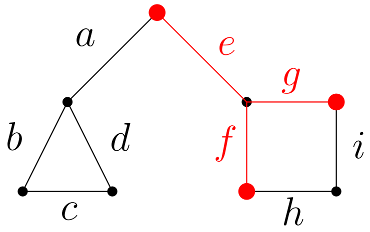
- Branch decomposition: Rooted tree whose leaves correspond to the edges of the graph
- Boundary $\partial(F)$ of a set of edges $F \subseteq E$: The vertices incident to edges from both F and $E \setminus F$.
- A set of edges $F \subseteq E$ is well-linked if it cannot be partitioned to (C_1, C_2) so that $|\partial(C_1)| < |\partial(F)|$ and $|\partial(C_2)| < |\partial(F)|$
- Want to maintain: Every edge set corresponding to a subtree is well-linked "downwards well-linkedness"
  $\Rightarrow$ Boundaries have size $\mathcal{O}(k)$
- Also: Degree at most $2^{\mathcal{O}(k)}$

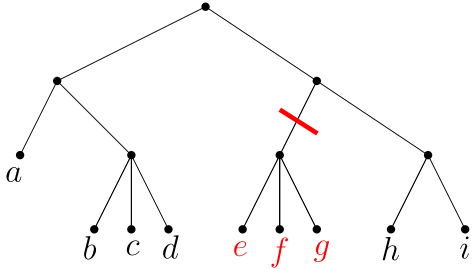
Maintained decomposition



- Branch decomposition: Rooted tree whose leaves correspond to the edges of the graph
- Boundary $\partial(F)$ of a set of edges $F \subseteq E$: The vertices incident to edges from both F and $E \setminus F$.
- A set of edges $F \subseteq E$ is well-linked if it cannot be partitioned to (C_1, C_2) so that $|\partial(C_1)| < |\partial(F)|$ and $|\partial(C_2)| < |\partial(F)|$
- Want to maintain: Every edge set corresponding to a subtree is well-linked "downwards well-linkedness"
  $\Rightarrow$ Boundaries have size $\mathcal{O}(k)$
- Also: Degree at most $2^{\mathcal{O}(k)}$
  $\Rightarrow$ Corresponds to a tree decomposition of width $2^{\mathcal{O}(k)}$ (later optimize to $\mathcal{O}(k)$)

Maintained decomposition



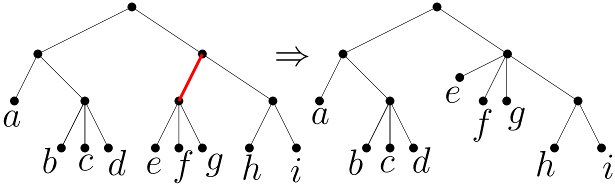


- Branch decomposition: Rooted tree whose leaves correspond to the edges of the graph
- Boundary $\partial(F)$ of a set of edges $F \subseteq E$: The vertices incident to edges from both F and $E \setminus F$.
- A set of edges $F \subseteq E$ is well-linked if it cannot be partitioned to (C_1, C_2) so that $|\partial(C_1)| < |\partial(F)|$ and $|\partial(C_2)| < |\partial(F)|$
- Want to maintain: Every edge set corresponding to a subtree is well-linked "downwards well-linkedness"
  - $\Rightarrow$ Boundaries have size $\mathcal{O}(k)$
- Also: Degree at most $2^{\mathcal{O}(k)}$
  - $\Rightarrow$ Corresponds to a tree decomposition of width $2^{\mathcal{O}(k)}$ (later optimize to $\mathcal{O}(k)$)
- Depth at most $2^{\mathcal{O}(k)} \log n$

Local rotations

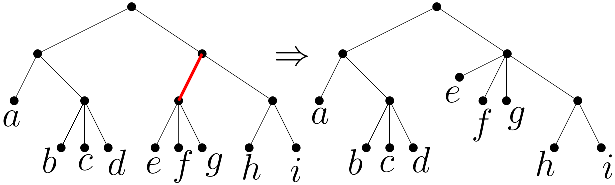
Local rotations

1. **Contraction**: Given an edge *uv* of the decomposition, contract it.



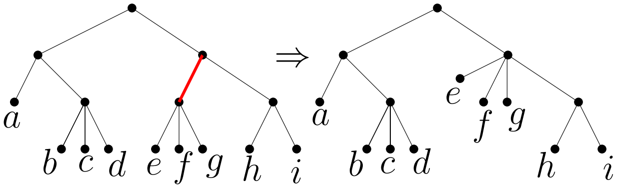
Local rotations

1. **Contraction**: Given an edge *uv* of the decomposition, contract it.
   - Maintains downwards well-linkedness, but increases degree

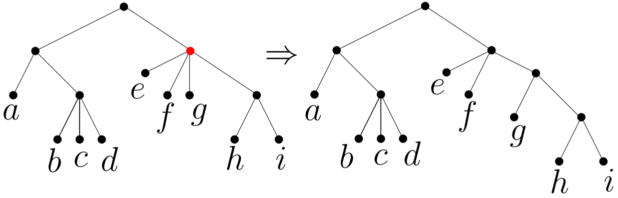


Local rotations

1. **Contraction**: Given an edge *uv* of the decomposition, contract it.
   - Maintains downwards well-linkedness, but increases degree

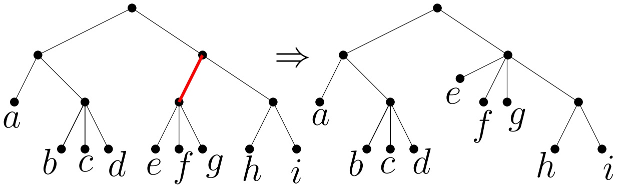


2. **Splitting**: Given a node of degree $> f(k) = 2^{\mathcal{O}(k)}$, locally split it to multiple nodes

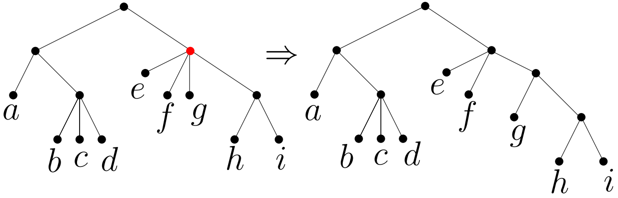


Local rotations

1. **Contraction**: Given an edge *uv* of the decomposition, contract it.
   - Maintains downwards well-linkedness, but increases degree



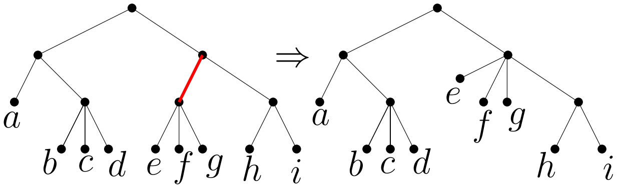
2. **Splitting**: Given a node of degree $> f(k) = 2^{\mathcal{O}(k)}$, locally split it to multiple nodes
   - Lemma: Can be done so that downwards well-linkedness is maintained



Local rotations

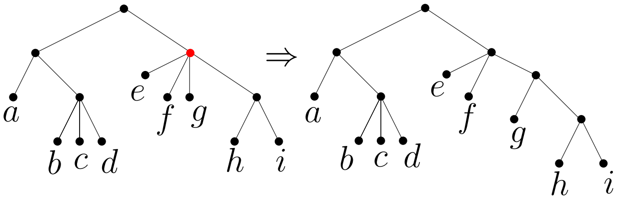
1. **Contraction**: Given an edge *uv* of the decomposition, contract it.

   - Maintains downwards well-linkedness, but increases degree



2. **Splitting**: Given a node of degree $> f(k) = 2^{\mathcal{O}(k)}$, locally split it to multiple nodes

   - Lemma: Can be done so that downwards well-linkedness is maintained
   - Can choose a set of ≤ 3 children that will not drop deeper



Inserting and deleting edges

Inserting and deleting edges

- In addition to normal edges, have also self-loops for every vertex

Inserting and deleting edges

- In addition to normal edges, have also self-loops for every vertex
- To insert an edge uv:

Inserting and deleting edges

- In addition to normal edges, have also self-loops for every vertex
- To insert an edge uv:
  1. Rotate the self-loops u and v to be children of the root
  2. Insert uv as an additional child of the root
  3. Decomposition is easy to update

Inserting and deleting edges

- In addition to normal edges, have also self-loops for every vertex
- To insert an edge uv:
  1. Rotate the self-loops u and v to be children of the root
  2. Insert uv as an additional child of the root
  3. Decomposition is easy to update
- To delete an edge uv:

Inserting and deleting edges

- In addition to normal edges, have also self-loops for every vertex
- To insert an edge uv:
  1. Rotate the self-loops u and v to be children of the root
  2. Insert uv as an additional child of the root
  3. Decomposition is easy to update
- To delete an edge uv:
  1. Rotate uv and the self-loops u and v to be children of the root
  2. Delete uv
  3. Decomposition is easy to update

Controlling the depth

Controlling the depth

- size(t): Number of leafs below t

Controlling the depth

- size(t): Number of leafs below t
- **Invariant:** If b is a descendant of a with distance $> f(k) = 2^{\mathcal{O}(k)}$ from a, then size(b) $<$ size(a)$/2$.

Controlling the depth

- size(t): Number of leafs below t
- **Invariant:** If b is a descendant of a with distance $> f(k) = 2^{\mathcal{O}(k)}$ from a, then size(b) $<$ size(a)$/2$.
- Implies depth $\leq 2^{\mathcal{O}(k)} \log n$

Controlling the depth

- size(t): Number of leafs below t
- **Invariant:** If b is a descendant of a with distance $> f(k) = 2^{\mathcal{O}(k)}$ from a, then size(b) $<$ size(a)$/2$.
- Implies depth $\leq 2^{\mathcal{O}(k)} \log n$
- Potential-function:
  - $\Phi(t) = (\text{degree}(t) - 2) \cdot \log(\text{size}(t))$
  - $\Phi(T) = \sum_{t \in V(T)} \Phi(t)$

Controlling the depth

- size(t): Number of leafs below t
- **Invariant:** If b is a descendant of a with distance $> f(k) = 2^{\mathcal{O}(k)}$ from a, then size(b) $<$ size(a)$/2$.
- Implies depth $\leq 2^{\mathcal{O}(k)} \log n$
- Potential-function:
  - $\Phi(t) = (\text{degree}(t) - 2) \cdot \log(\text{size}(t))$
  - $\Phi(T) = \sum_{t \in V(T)} \Phi(t)$

Lemma

If a pair a, b does not satisfy the invariant, then can decrease the value of $\Phi(T)$ in time proportional to the decrease

Controlling the depth

- size(t): Number of leafs below t
- **Invariant:** If b is a descendant of a with distance $> f(k) = 2^{\mathcal{O}(k)}$ from a, then size(b) $<$ size(a)$/2$.
- Implies depth $\leq 2^{\mathcal{O}(k)} \log n$
- Potential-function:
  - $\Phi(t) = (\text{degree}(t) - 2) \cdot \log(\text{size}(t))$
  - $\Phi(T) = \sum_{t \in V(T)} \Phi(t)$

Lemma

If a pair a, b does not satisfy the invariant, then can decrease the value of $\Phi(T)$ in time proportional to the decrease

Lemma

Edge insertion and deletion increase $\Phi(T)$ by $2^{\mathcal{O}(k)} \log n$

Conclusion

- $2^{\mathcal{O}(k)} \log n$ amortized update time for maintaining a tree decomposition of width at most $9k + 8$ of a dynamic graph of treewidth $\leq k$

Conclusion

- $2^{\mathcal{O}(k)} \log n$ amortized update time for maintaining a tree decomposition of width at most $9k + 8$ of a dynamic graph of treewidth $\leq k$
  - Can also maintain dynamic programming schemes on the tree decomposition

Conclusion

- $2^{\mathcal{O}(k)} \log n$ amortized update time for maintaining a tree decomposition of width at most $9k+8$ of a dynamic graph of treewidth $\leq k$
  - Can also maintain dynamic programming schemes on the tree decomposition
- Open problems:

Conclusion

- $2^{\mathcal{O}(k)} \log n$ amortized update time for maintaining a tree decomposition of width at most $9k + 8$ of a dynamic graph of treewidth $\leq k$
  - Can also maintain dynamic programming schemes on the tree decomposition
- Open problems:
  - From amortized to worst-case?

Conclusion

- $2^{\mathcal{O}(k)} \log n$ amortized update time for maintaining a tree decomposition of width at most $9k + 8$ of a dynamic graph of treewidth $\leq k$
  - Can also maintain dynamic programming schemes on the tree decomposition
- Open problems:
  - From amortized to worst-case?
  - Can we rule out $f(k) + \mathcal{O}(\log n)$ update time?

Conclusion

- $2^{\mathcal{O}(k)} \log n$ amortized update time for maintaining a tree decomposition of width at most $9k + 8$ of a dynamic graph of treewidth $\leq k$
  - Can also maintain dynamic programming schemes on the tree decomposition
- Open problems:
  - From amortized to worst-case?
  - Can we rule out $f(k) + \mathcal{O}(\log n)$ update time?
  - Improve approximation ratio (3 seems to be a lower bound for explicitly maintaining a tree decomposition)

Conclusion

- $2^{\mathcal{O}(k)} \log n$ amortized update time for maintaining a tree decomposition of width at most $9k + 8$ of a dynamic graph of treewidth $\leq k$
  - Can also maintain dynamic programming schemes on the tree decomposition
- Open problems:
  - From amortized to worst-case?
  - Can we rule out $f(k) + \mathcal{O}(\log n)$ update time?
  - Improve approximation ratio (3 seems to be a lower bound for explicitly maintaining a tree decomposition)

Thank you!