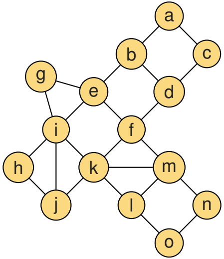
# Dynamic Treewidth in Logarithmic Time

Tuukka Korhonen

FOCS 2025

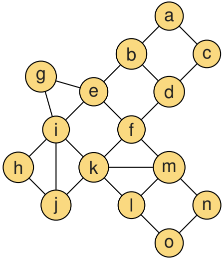
16 December 2025

# Treewidth

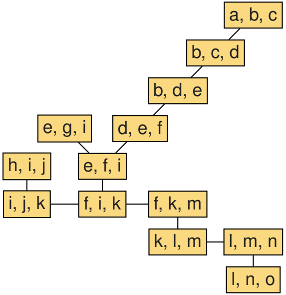



Graph $G$

# Treewidth

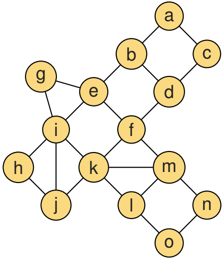



Graph *G*

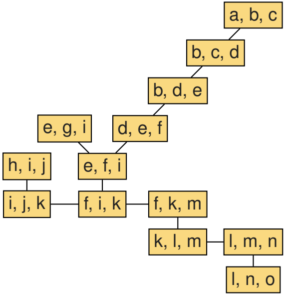



A tree decomposition of *G*

# Treewidth



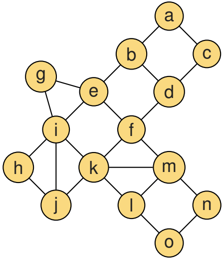

Graph $G$



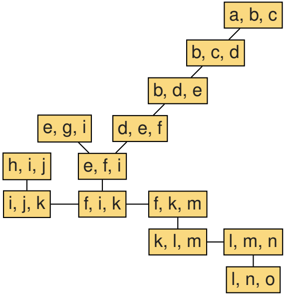

A tree decomposition of $G$

1. Every vertex should be in a bag

# Treewidth



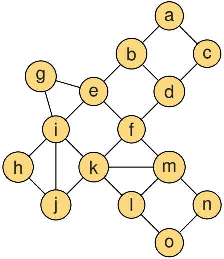

Graph $G$



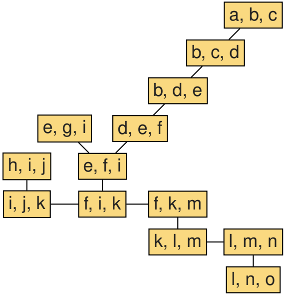

A tree decomposition of $G$

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

# Treewidth



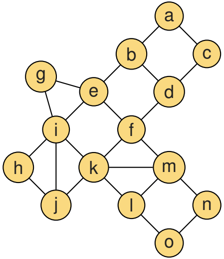

Graph $G$



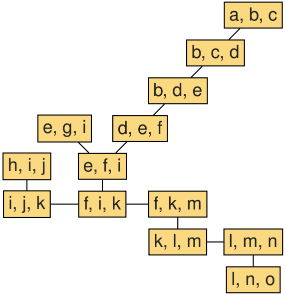

A tree decomposition of $G$

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 $G$

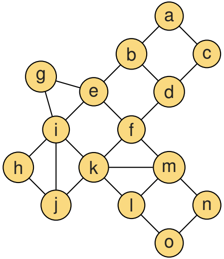



A tree decomposition of $G$

Width = 2

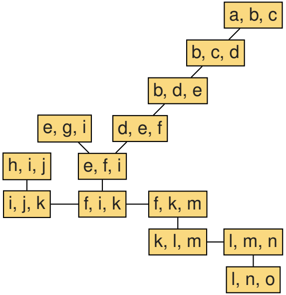
1. Every vertex should be in a bag
2. Every edge should be in a bag
3. For every vertex $v$, the bags containing $v$ should form a connected subtree
4. Width = maximum bag size $-1$

# Treewidth




Graph $G$

Treewidth 2




A tree decomposition of $G$

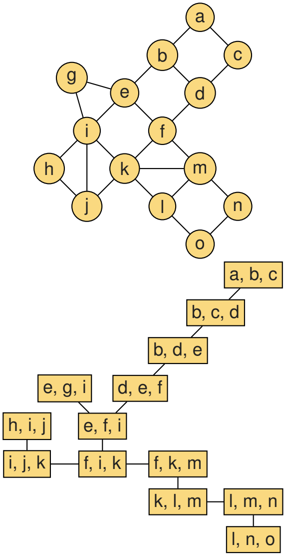
Width = 2

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]

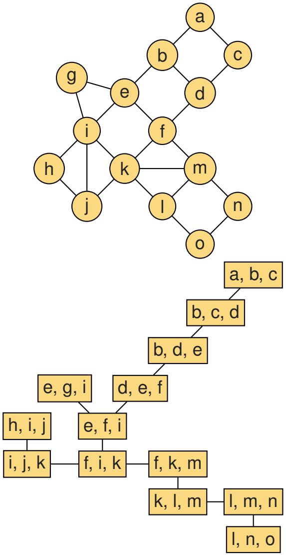
# Why treewidth?

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



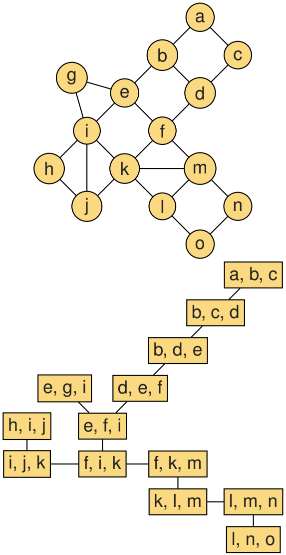
# Why treewidth?

- 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



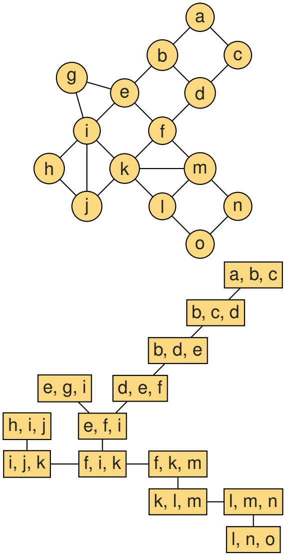
# Why treewidth?

- 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



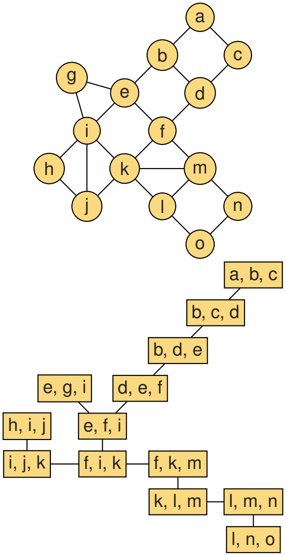
# Why treewidth?

- 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 dynamic programming on the 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 dynamic programming on the 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 dynamic programming on the 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 dynamic programming on the 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 dynamic programming on the 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]
- 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 dynamic programming on the 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]
- 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 dynamic programming on the 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]
- 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 dynamic programming on the 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]
- 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.

# 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 dynamic programming on the 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]
- 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.

- Arguably optimal update time: $\Omega(\log n)$ needed for dynamic forests [Patrascu&Demaine'04]

# Detailed Theorem Statement

**Theorem (This work)**

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

# Detailed Theorem Statement

**Theorem (This work)**

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

There is data structure that:

- is initialized with an 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 $9 \cdot \mathrm{tw}(G) + 8$

# Detailed Theorem Statement

**Theorem (This work)**

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

There is data structure that:

- is initialized with an 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 $9 \cdot \text{tw}(G) + 8$
- can also maintain any dynamic programming scheme on the decomposition within similar running time (formalized by tree decomposition automata)

# Detailed Theorem Statement

**Theorem (This work)**

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

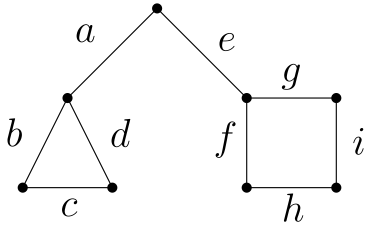
There is data structure that:

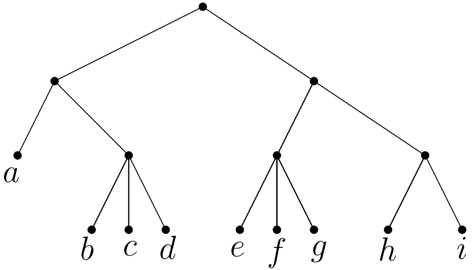
- is initialized with an integer $k$ and an edgeless $n$-vertex graph $G$
- supports edge insertions/deletions in amortized time $2^{\mathcal{O}(k)} \log n$ under the promise that $\text{tw}(G) \leq k$
- maintains a tree decomposition of $G$ of width at most $9 \cdot \text{tw}(G) + 8$
- can also maintain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

# The algorithm

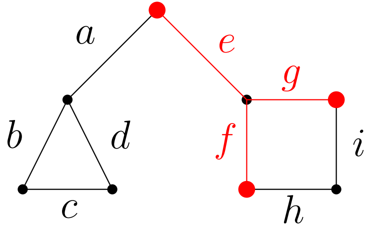
# Maintained decomposition

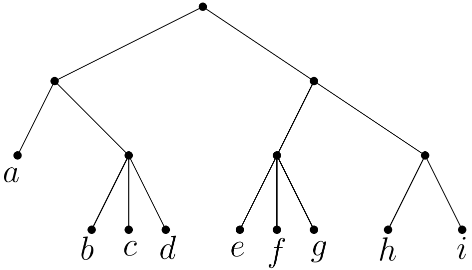



- Branch decomposition: Rooted tree whose leaves correspond to the edges of the graph

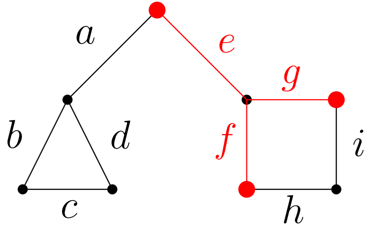
# Maintained decomposition

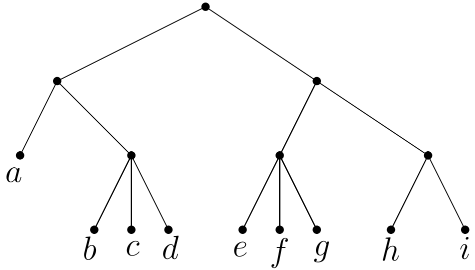



- Branch decomposition: Rooted tree whose leaves correspond to the edges of the graph
- Boundary $\partial(F)$ of a set of edges $F \subseteq E$: The vertices incident to edges from both $F$ and $E \setminus F$.

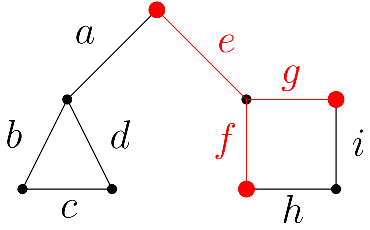
# Maintained decomposition

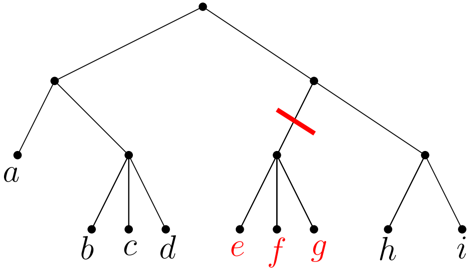



- Branch decomposition: Rooted tree whose leaves correspond to the edges of the graph
- Boundary $\partial(F)$ of a set of edges $F \subseteq E$: The vertices incident to edges from both $F$ and $E \setminus F$.
- A set of edges $F \subseteq E$ is well-linked if it cannot be partitioned to $(C_1, C_2)$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

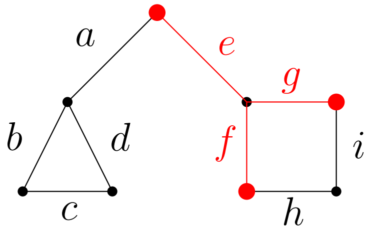
# Maintained decomposition

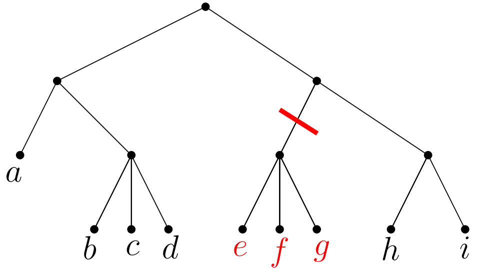



- Branch decomposition: Rooted tree whose leaves correspond to the edges of the graph
- Boundary $\partial(F)$ of a set of edges $F \subseteq E$: The vertices incident to edges from both $F$ and $E \setminus F$.
- A set of edges $F \subseteq E$ is well-linked if it cannot be partitioned to $(C_1, C_2)$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

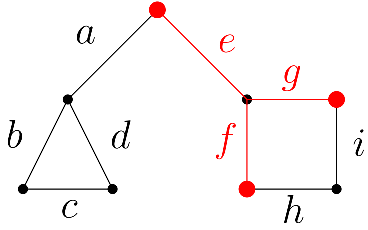
# Maintained decomposition

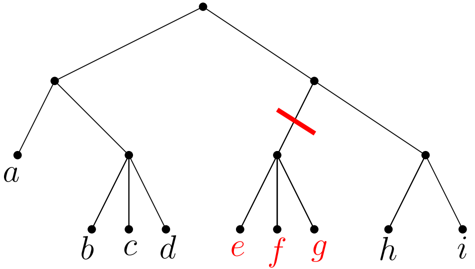



- Branch decomposition: Rooted tree whose leaves correspond to the edges of the graph
- Boundary $\partial(F)$ of a set of edges $F \subseteq E$: The vertices incident to edges from both $F$ and $E \setminus F$.
- A set of edges $F \subseteq E$ is well-linked if it cannot be partitioned to $(C_1, C_2)$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

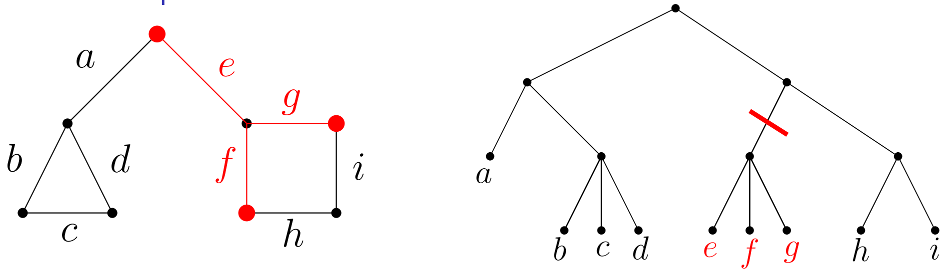
# Maintained decomposition



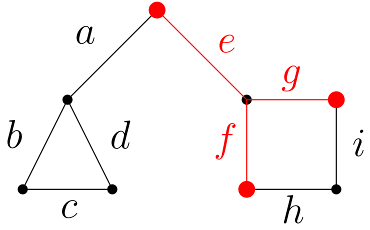

- Branch decomposition: Rooted tree whose leaves correspond to the edges of the graph
- Boundary $\partial(F)$ of a set of edges $F \subseteq E$: The vertices incident to edges from both $F$ and $E \setminus F$.
- A set of edges $F \subseteq E$ is well-linked if it cannot be partitioned to $(C_1, C_2)$ 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 $2^{\mathcal{O}(k)}$

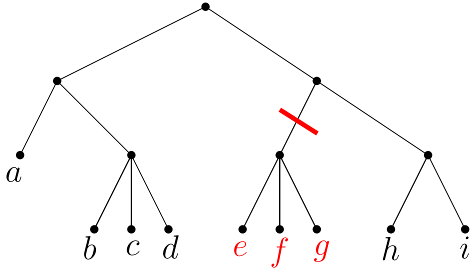
# Maintained decomposition




- 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 $2^{\mathcal{O}(k)}$
  - $\Rightarrow$ Tree decomposition of width $2^{\mathcal{O}(k)}$ (later optimize to $\mathcal{O}(k)$)

# Maintained decomposition



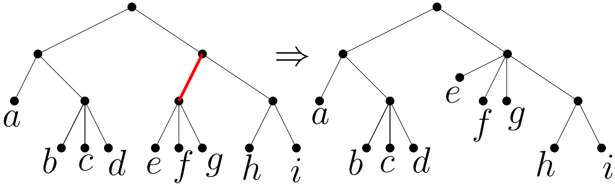




- Branch decomposition: Rooted tree whose leaves correspond to the edges of the graph
- Boundary $\partial(F)$ of a set of edges $F \subseteq E$: The vertices incident to edges from both $F$ and $E \setminus F$.
- A set of edges $F \subseteq E$ is well-linked if it cannot be partitioned to $(C_1, C_2)$ 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 $2^{\mathcal{O}(k)}$
  $\Rightarrow$ 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

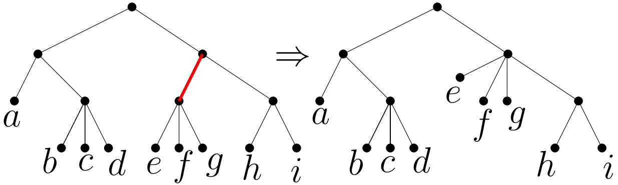
# Local rotations

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



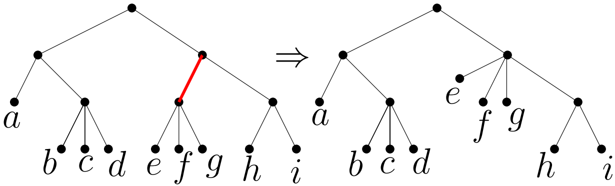
# Local rotations

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

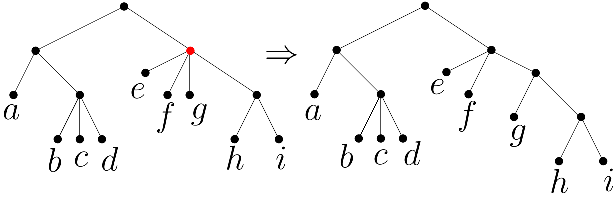


# Local rotations

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

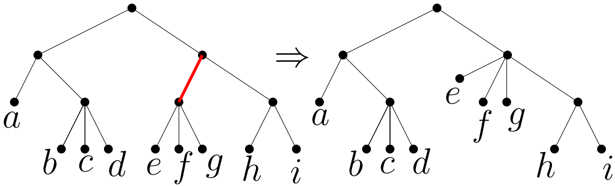



2. **Splitting**: Given a node of degree $> f(k) = 2^{\mathcal{O}(k)}$, locally split it to multiple nodes

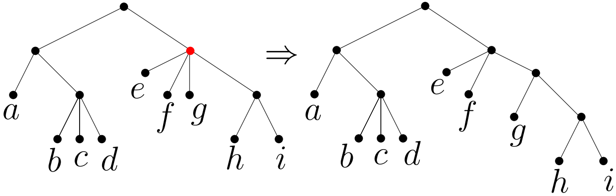


# Local rotations

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

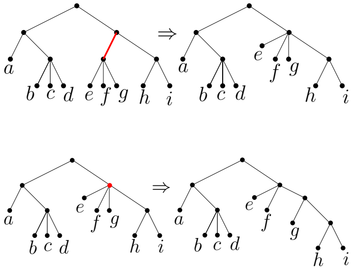



2. **Splitting**: Given a node of degree $> f(k) = 2^{\mathcal{O}(k)}$, locally split it to multiple nodes
   - Lemma: Can be done so that downwards well-linkedness is maintained



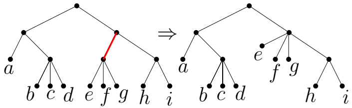
# Approach

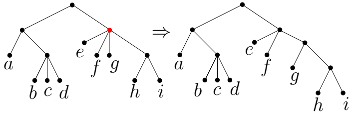
- **Idea**: Implement splay-tree-like rotations with the local rotations



# Approach

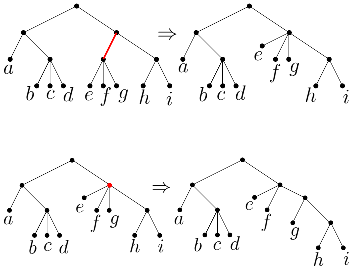
- **Idea**: Implement splay-tree-like rotations with the local rotations
- Potential-function: $\Phi(t) = (\#\text{children}(t) - 1) \cdot \log(\text{size}(t))$



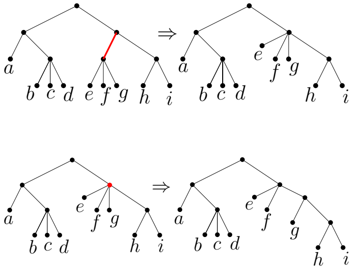
# Approach

- **Idea**: Implement splay-tree-like rotations with the local rotations
- Potential-function: $\Phi(t) = (\#\text{children}(t) - 1) \cdot \log(\text{size}(t))$
- To facilitate edge insertions/deletions, additional self-loops on vertices



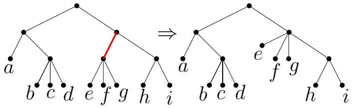
# Approach

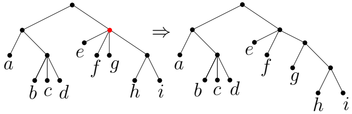
- **Idea**: Implement splay-tree-like rotations with the local rotations
- Potential-function: $\Phi(t) = (\#\text{children}(t) - 1) \cdot \log(\text{size}(t))$
- To facilitate edge insertions/deletions, additional self-loops on vertices
- To insert an edge *uv*:
  1. rotate the self-loops *u* and *v* to be children of the root
  2. insert *uv* as another child of the root
  3. re-balance



# Approach

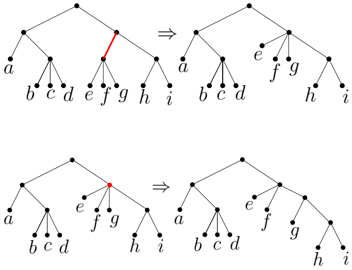
- **Idea**: Implement splay-tree-like rotations with the local rotations
- Potential-function: $\Phi(t) = (\#\text{children}(t) - 1) \cdot \log(\text{size}(t))$
- To facilitate edge insertions/deletions, additional self-loops on vertices
- To insert an edge *uv*:
  1. rotate the self-loops *u* and *v* to be children of the root
  2. insert *uv* as another child of the root
  3. re-balance
- Deletion of *uv* is similar



# Approach

- **Idea**: Implement splay-tree-like rotations with the local rotations
- Potential-function: $\Phi(t) = (\#\text{children}(t) - 1) \cdot \log(\text{size}(t))$
- To facilitate edge insertions/deletions, additional self-loops on vertices
- To insert an edge *uv*:
  1. rotate the self-loops *u* and *v* to be children of the root
  2. insert *uv* as another child of the root
  3. re-balance
- Deletion of *uv* is similar
- **Post-processing:** Replace each node of size $2^{\mathcal{O}(k)}$ by a tree decomposition of width $\mathcal{O}(k)$



# Conclusion

- Dynamic treewidth in $2^{\mathcal{O}(k)} \log n$ amortized update time

# Conclusion

- Dynamic treewidth in $2^{\mathcal{O}(k)} \log n$ amortized update time
  - Tree decomposition of width at most $9 \cdot \text{tw}(G) + 8$ of dynamic graph of treewidth $\text{tw}(G) \leq k$
  - Can also maintain dynamic programming schemes on the tree decomposition

# Conclusion

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

# Conclusion

- Dynamic treewidth in $2^{\mathcal{O}(k)} \log n$ amortized update time
  - Tree decomposition of width at most $9 \cdot \text{tw}(G) + 8$ of dynamic graph of treewidth $\text{tw}(G) \leq k$
  - 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)

# Conclusion

- Dynamic treewidth in $2^{\mathcal{O}(k)} \log n$ amortized update time
  - Tree decomposition of width at most $9 \cdot \text{tw}(G) + 8$ of dynamic graph of treewidth $\text{tw}(G) \leq k$
  - 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)

Thank you!