Introduction to Parametrized Algorithms
Part 2

Łukasz Kowalik
some slides by Marek Cygan, Marcin Pilipczuk and Michał Pilipczuk

UNIVERSITY
OF WARSAW

Gdańsk, Poland,
11.09.2017

Motivation of treewidth

- Independent Set: given a graph, find the maximum possible size of a subset of vertices that are pairwise non-adjacent.

Motivation of treewidth

- Independent Set: given a graph, find the maximum possible size of a subset of vertices that are pairwise non-adjacent.
- NP-hard in general, tractable in many special cases.

Motivation of treewidth

- Independent Set: given a graph, find the maximum possible size of a subset of vertices that are pairwise non-adjacent.
- **NP**-hard in general, tractable in many special cases.
- **Easy exercise**: linear-time algorithm on trees.

Motivation of treewidth

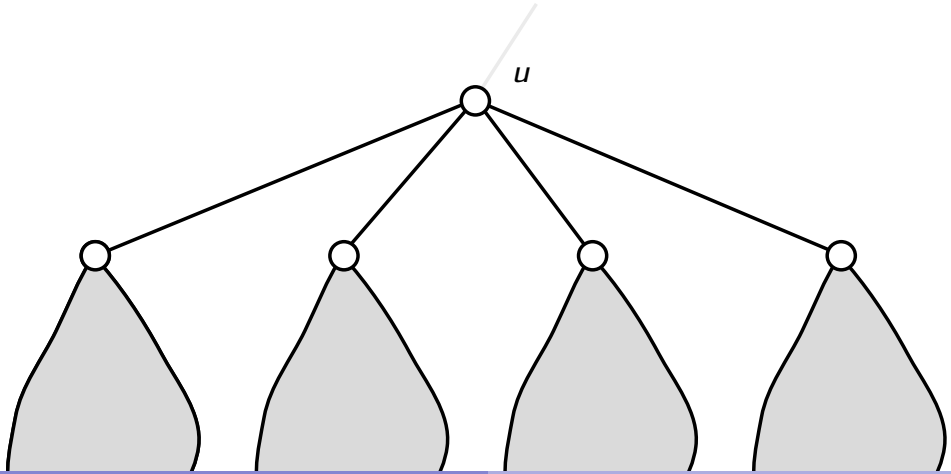
- Independent Set: given a graph, find the maximum possible size of a subset of vertices that are pairwise non-adjacent.
- **NP**-hard in general, tractable in many special cases.
- **Easy exercise**: linear-time algorithm on trees.
  - Principle of **dynamic programming**.

Independent Set on trees

- Root the tree in an arbitrary vertex r.

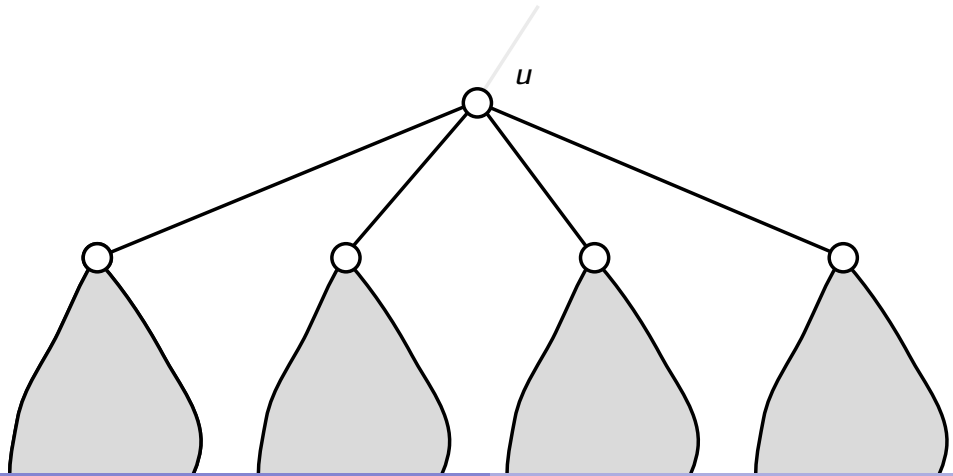
Independent Set on trees

- Root the tree in an arbitrary vertex r.
  - T_u is the subtree rooted at u.



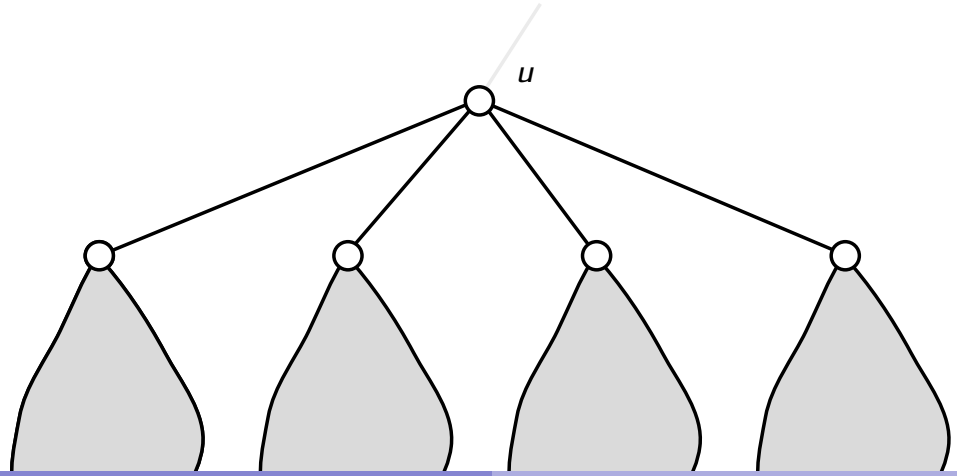
Independent Set on trees

- Root the tree in an arbitrary vertex r.
  - T_u is the subtree rooted at u.
- Compute dynamic programming tables:



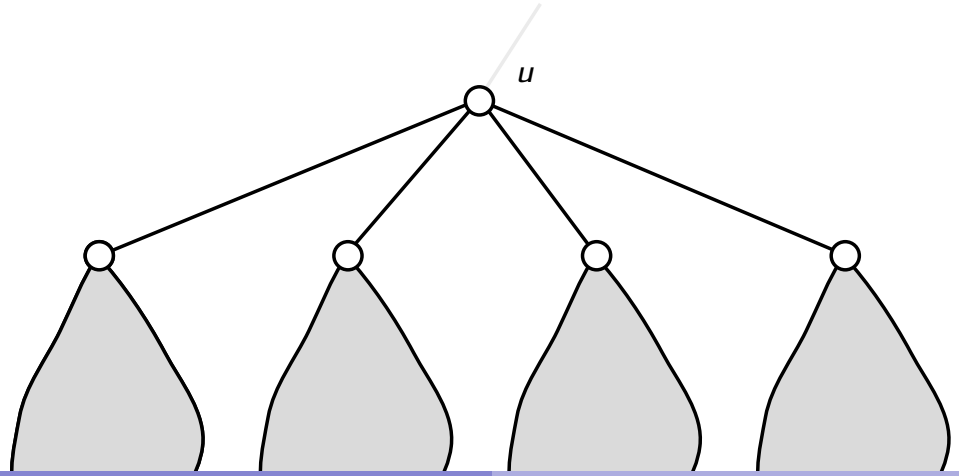
Independent Set on trees

- Root the tree in an arbitrary vertex r.
  - T_u is the subtree rooted at u.
- Compute dynamic programming tables:
  - $A[u]$: maximum size of an IS in T_u.



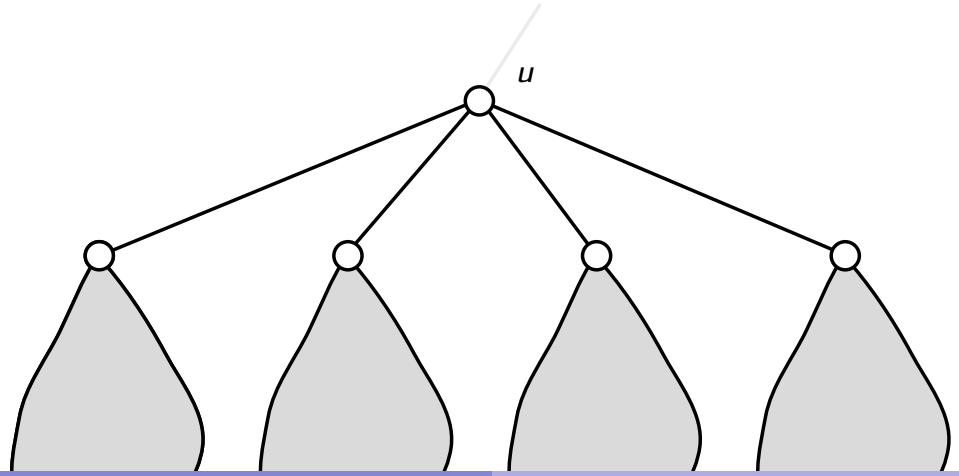
Independent Set on trees

- Root the tree in an arbitrary vertex r.
  - T_u is the subtree rooted at u.
- Compute dynamic programming tables:
  - $A[u]$: maximum size of an IS in T_u.
  - $B[u]$: maximum size of an IS in T_u that contains u.



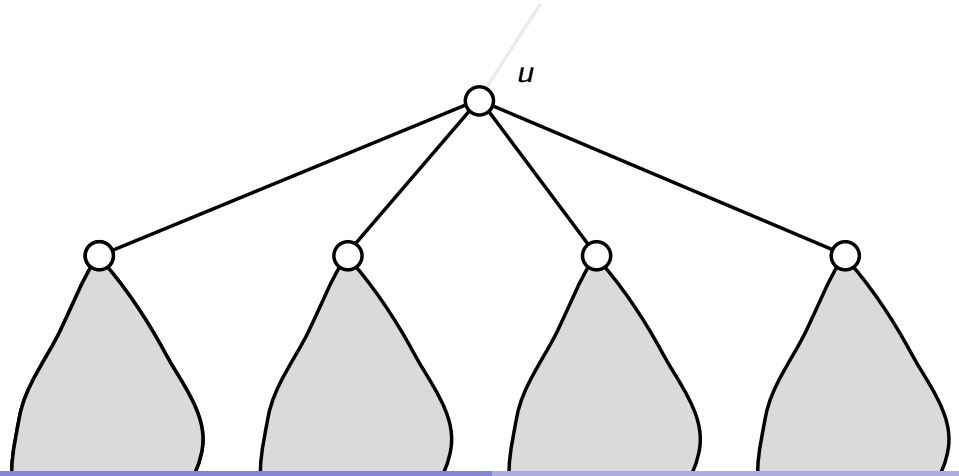
Independent Set on trees

- Root the tree in an arbitrary vertex r.
  - T_u is the subtree rooted at u.
- Compute dynamic programming tables:
  - $A[u]$: maximum size of an IS in T_u.
  - $B[u]$: maximum size of an IS in T_u that contains u.
  - $C[u]$: maximum size of an IS in T_u that excludes u.



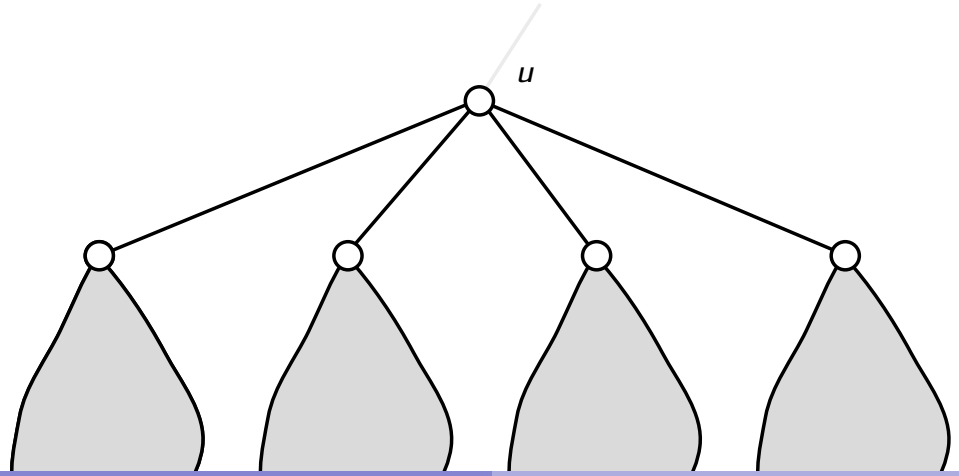
Independent Set on trees

- Root the tree in an arbitrary vertex r.
  - T_u is the subtree rooted at u.
- Compute dynamic programming tables:
  - $A[u]$: maximum size of an IS in T_u.
  - $B[u]$: maximum size of an IS in T_u that contains u.
  - $C[u]$: maximum size of an IS in T_u that excludes u.
- Recursive formulas:



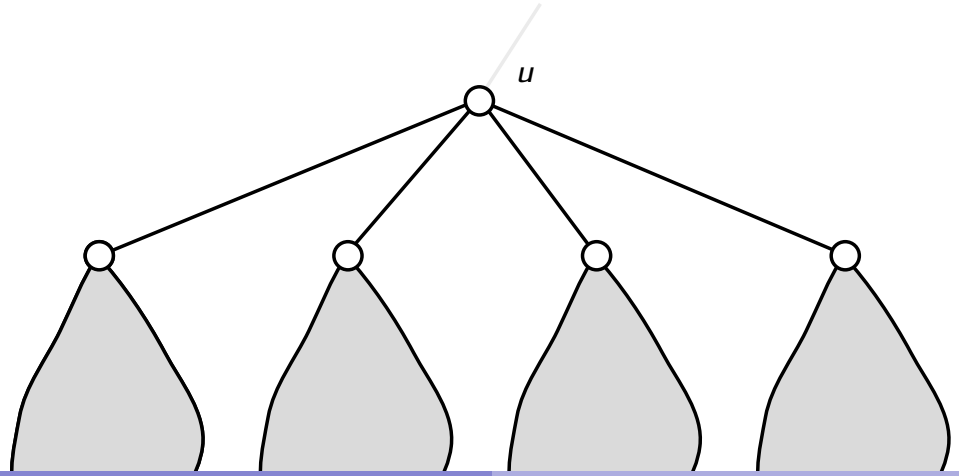
Independent Set on trees

- Root the tree in an arbitrary vertex r.
  - T_u is the subtree rooted at u.
- Compute dynamic programming tables:
  - $A[u]$: maximum size of an IS in T_u.
  - $B[u]$: maximum size of an IS in T_u that contains u.
  - $C[u]$: maximum size of an IS in T_u that excludes u.
- Recursive formulas:
  - $A[u] = \max(B[u], C[u])$.



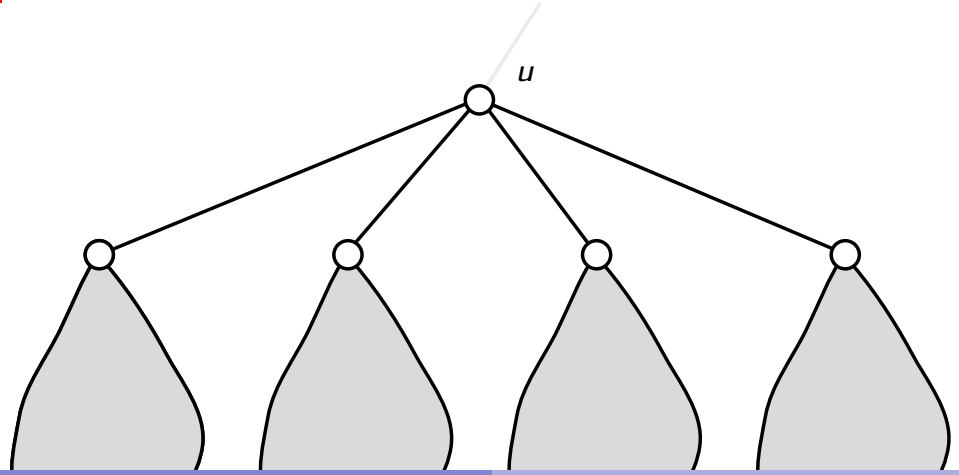
Independent Set on trees

- Root the tree in an arbitrary vertex r.
  - T_u is the subtree rooted at u.
- Compute dynamic programming tables:
  - $A[u]$: maximum size of an IS in T_u.
  - $B[u]$: maximum size of an IS in T_u that contains u.
  - $C[u]$: maximum size of an IS in T_u that excludes u.
- Recursive formulas:
  - $A[u] = \max(B[u], C[u])$.
  - $B[u] = 1 + \sum_{v \in \mathrm{chld}(u)} C[v]$.



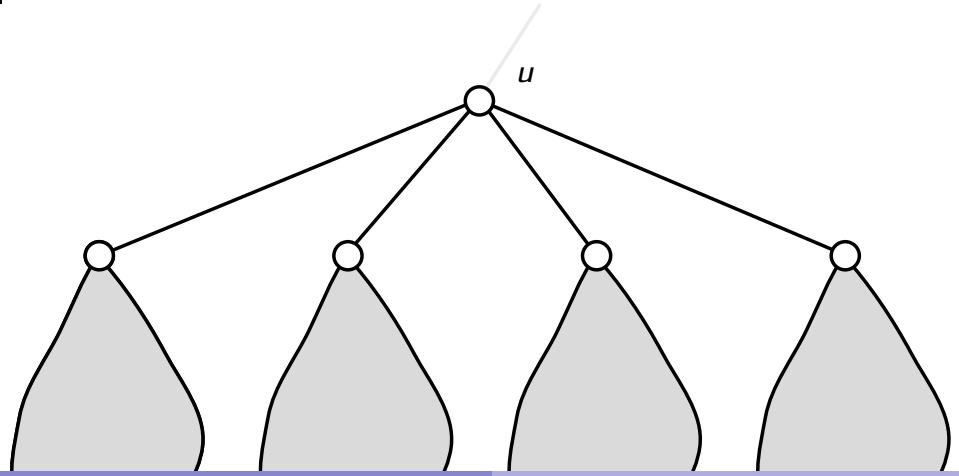
Independent Set on trees

- Root the tree in an arbitrary vertex r.
  - T_u is the subtree rooted at u.
- Compute dynamic programming tables:
  - $A[u]$: maximum size of an IS in T_u.
  - $B[u]$: maximum size of an IS in T_u that contains u.
  - $C[u]$: maximum size of an IS in T_u that excludes u.
- Recursive formulas:
  - $A[u] = \max(B[u], C[u])$.
  - $B[u] = 1 + \sum_{v \in \mathrm{chld}(u)} C[v]$.
  - $C[u] = \sum_{v \in \mathrm{chld}(u)} A[v]$.



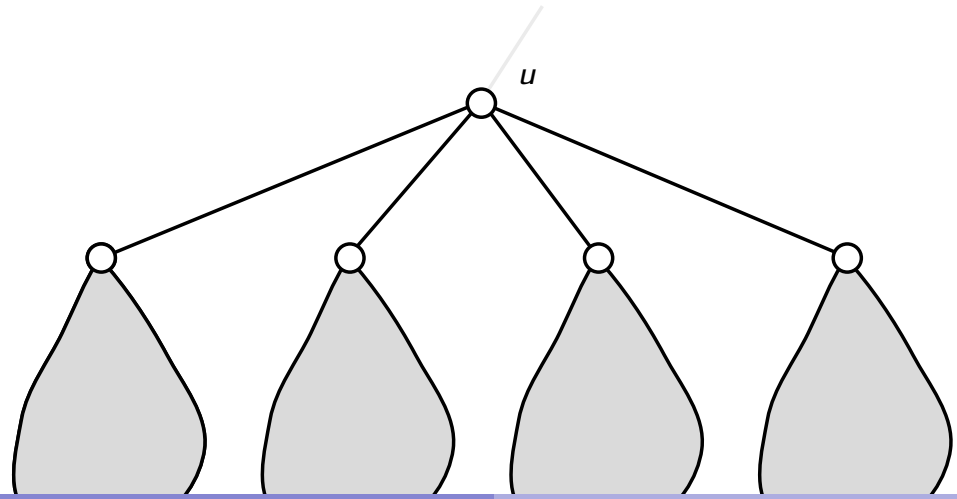
Independent Set on trees

- Root the tree in an arbitrary vertex r.
  - T_u is the subtree rooted at u.
- Compute dynamic programming tables:
  - $A[u]$: maximum size of an IS in T_u.
  - $B[u]$: maximum size of an IS in T_u that contains u.
  - $C[u]$: maximum size of an IS in T_u that excludes u.
- Recursive formulas:
  - $A[u] = \max(B[u], C[u])$.
  - $B[u] = 1 + \sum_{v \in \mathrm{chld}(u)} C[v]$.
  - $C[u] = \sum_{v \in \mathrm{chld}(u)} A[v]$.
- **Answer**: $A[r]$.

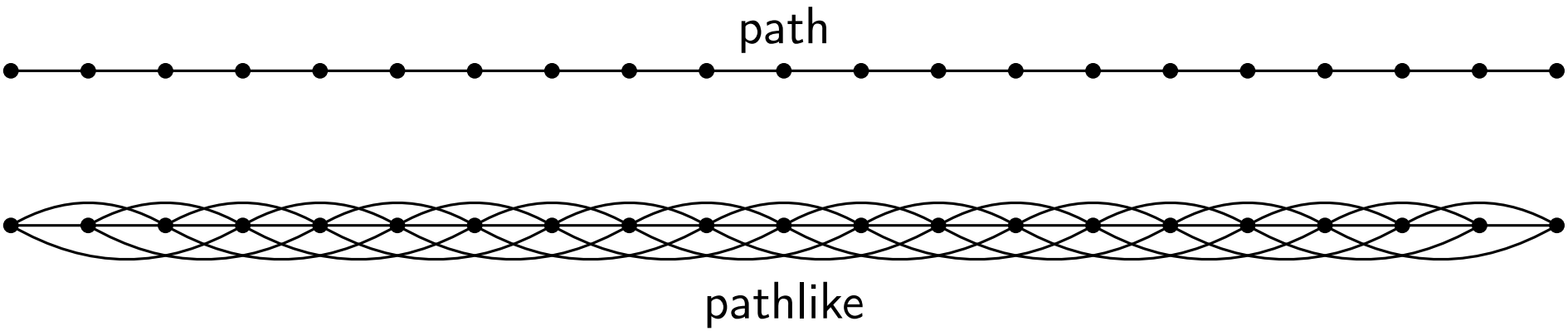


Independent Set on trees: Why does the DP work?

- Very limited dependence between vertices
- Essential information about subtree stored in a compact way.



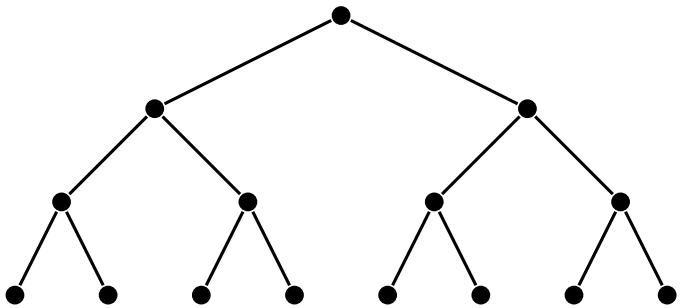
Generalizations of paths and trees

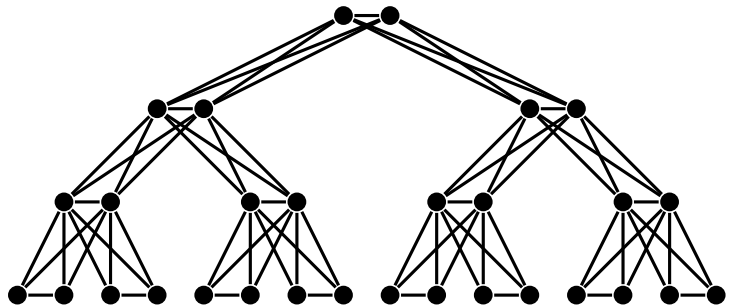


Pathwidth

Graph of small pathwidth is path-like.
We hope that problems tractable on paths are also tractable on graphs of small pathwidth.

Generalizations of paths and trees

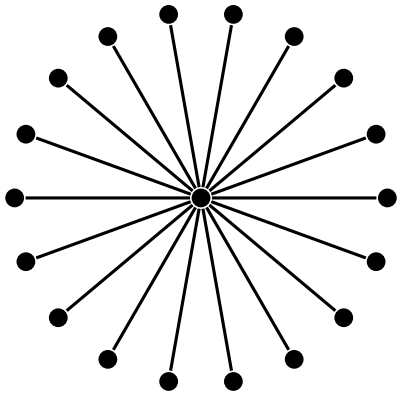


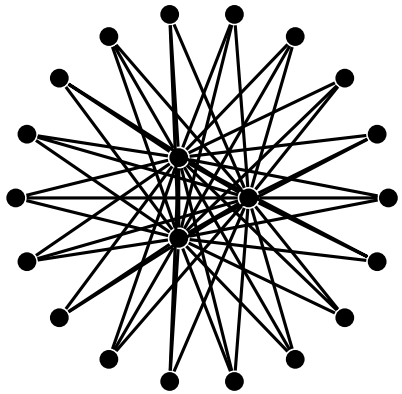


Treewidth

Graph of small treewidth is tree-like.
We hope that problems tractable on trees are also tractable on graphs of small treewidth.

Generalizations of paths and trees


tree


treelike

Treewidth

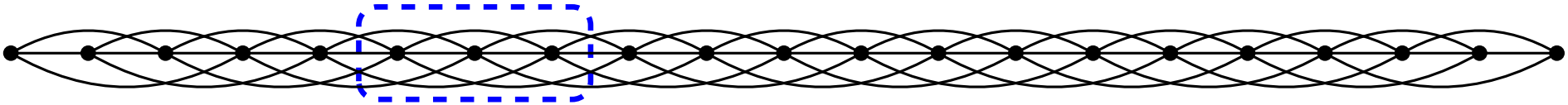
Graph of small treewidth is tree-like.
We hope that problems tractable on trees are also tractable on graphs of small treewidth.

Introduction

Crucial property of small pathwidth/treewidth graphs: separators.

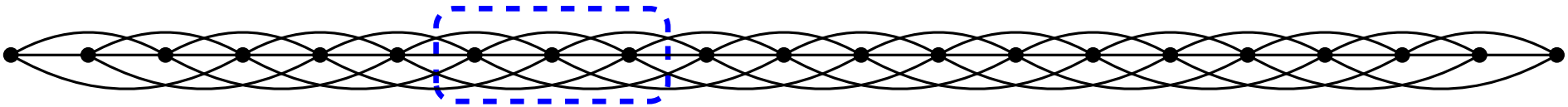
Introduction

Crucial property of small pathwidth/treewidth graphs: separators.



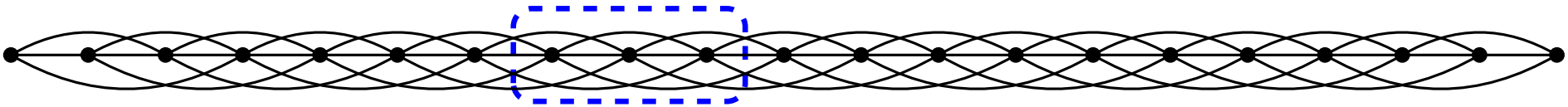
Introduction

Crucial property of small pathwidth/treewidth graphs: separators.



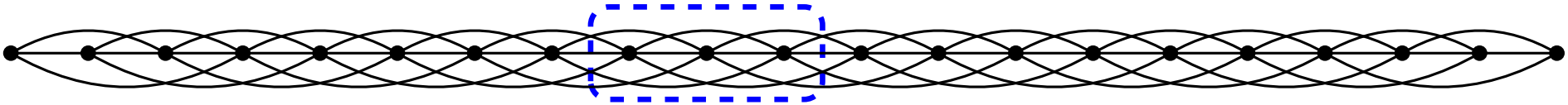
Introduction

Crucial property of small pathwidth/treewidth graphs: separators.



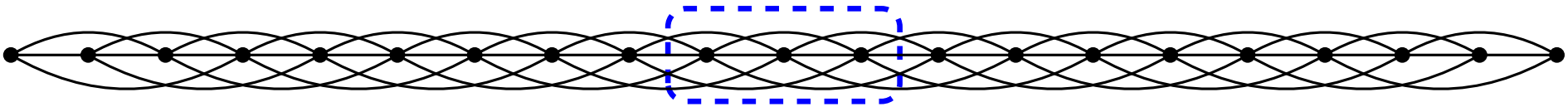
Introduction

Crucial property of small pathwidth/treewidth graphs: separators.



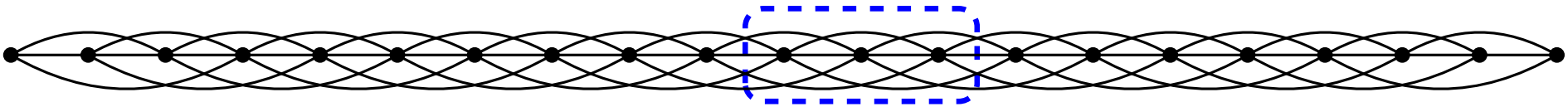
Introduction

Crucial property of small pathwidth/treewidth graphs: separators.



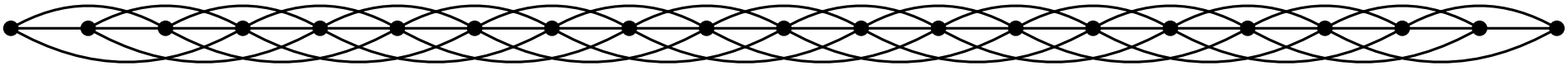
Introduction

Crucial property of small pathwidth/treewidth graphs: separators.



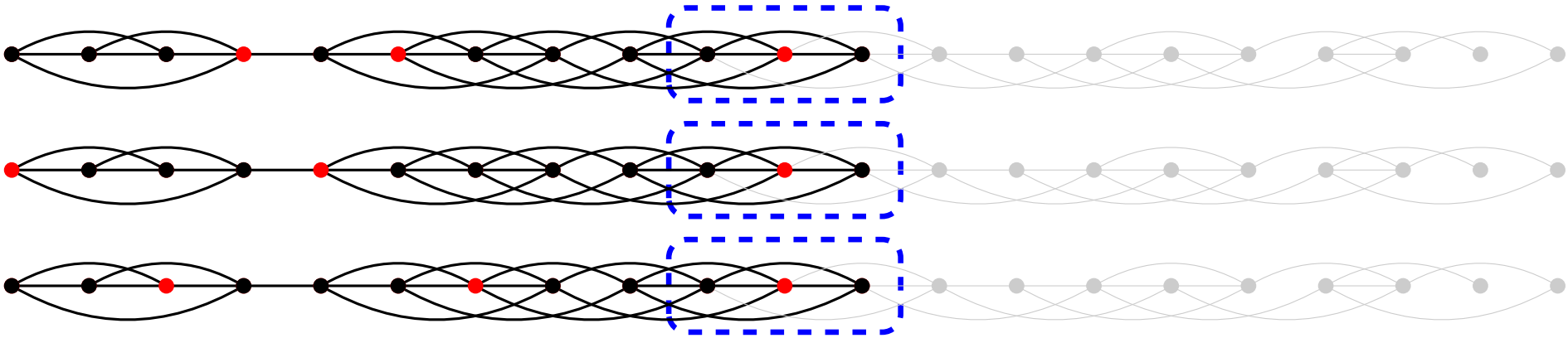
Introduction

Crucial property of small pathwidth/treewidth graphs: separators.



Introduction

Crucial property of small pathwidth/treewidth graphs: separators.



Definition (mapping to intervals)

Definition

A graph G has *pathwidth* $\leq \mathrm{pw}$ if there exists a function $f : V \to \textit{intervals}$ such that:

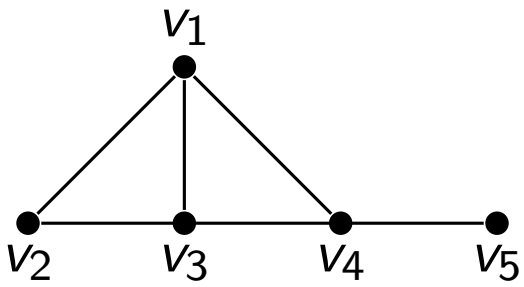
1. if $(u, v) \in E$ then $f(u) \cap f(v) \neq \emptyset$
2. $\forall_{x \in \mathbb{Z}} |\{v \in V : x \in f(v)\}| \leq \mathrm{pw} + 1$

Definition (mapping to intervals)

Definition

A graph G has *pathwidth* $\leq \mathrm{pw}$ if there exists a function $f : V \to \textit{intervals}$ such that:

1. if $(u, v) \in E$ then $f(u) \cap f(v) \neq \emptyset$
2. $\forall_{x \in \mathbb{Z}} |\{v \in V : x \in f(v)\}| \leq \mathrm{pw} + 1$

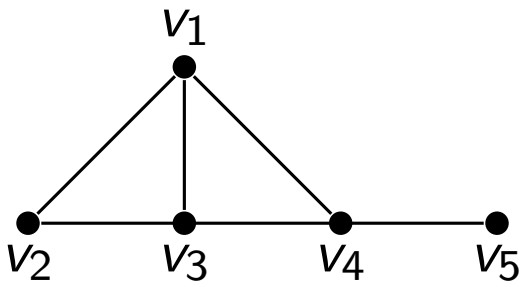


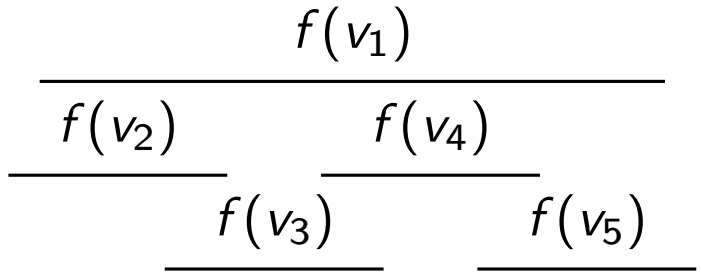
Definition (mapping to intervals)

Definition

A graph G has *pathwidth* $\leq \mathrm{pw}$ if there exists a function $f : V \to \textit{intervals}$ such that:

1. if $(u, v) \in E$ then $f(u) \cap f(v) \neq \emptyset$
2. $\forall_{x \in \mathbb{Z}} |\{v \in V : x \in f(v)\}| \leq \mathrm{pw} + 1$



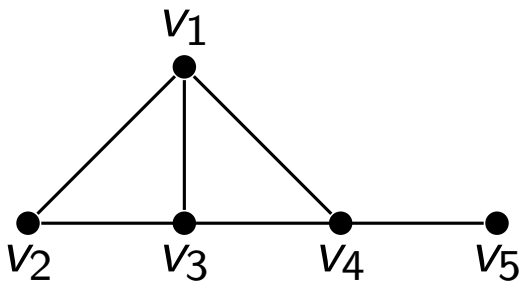


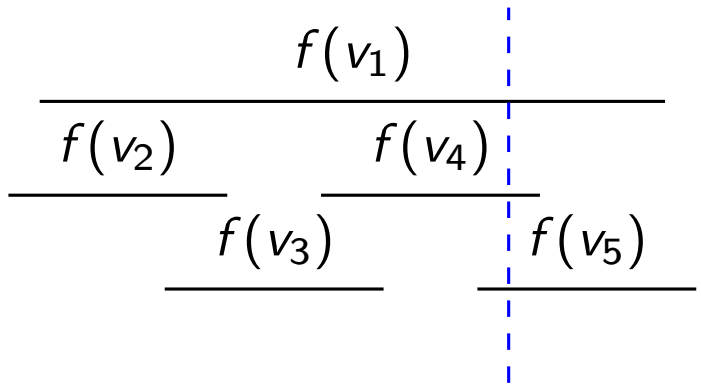
Definition (mapping to intervals)

Definition

A graph G has *pathwidth* $\leq \mathrm{pw}$ if there exists a function $f : V \to \textit{intervals}$ such that:

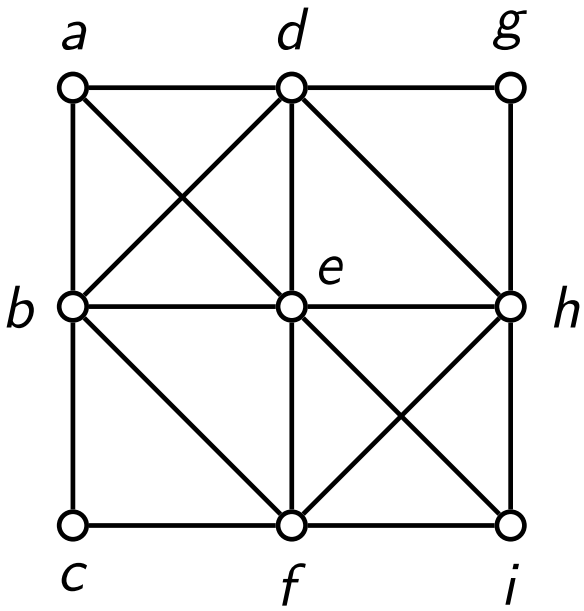
1. if $(u, v) \in E$ then $f(u) \cap f(v) \neq \emptyset$
2. $\forall_{x \in \mathbb{Z}} |\{v \in V : x \in f(v)\}| \leq \mathrm{pw} + 1$



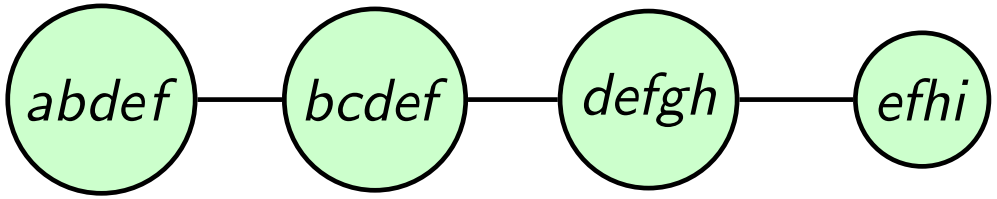


Path decompositions and pathwidth

Graph $G = (V, E)$



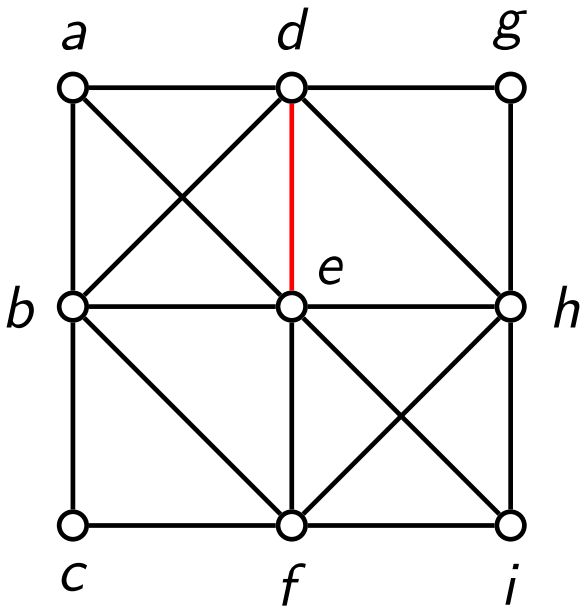
Path decomposition of G



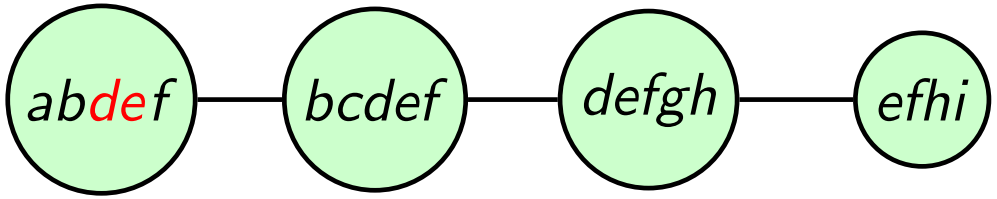
Path decomposition is a path of **bags** (subsets of V)

Path decompositions and pathwidth

Graph $G = (V, E)$



Path decomposition of G

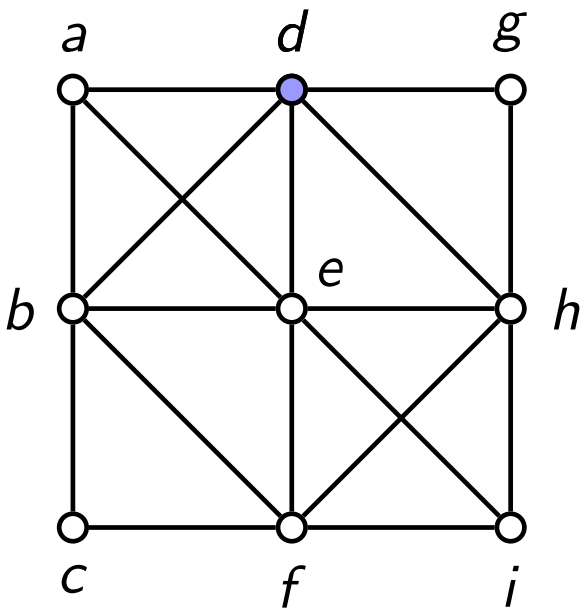


Path decomposition is a path of **bags** (subsets of V) such that

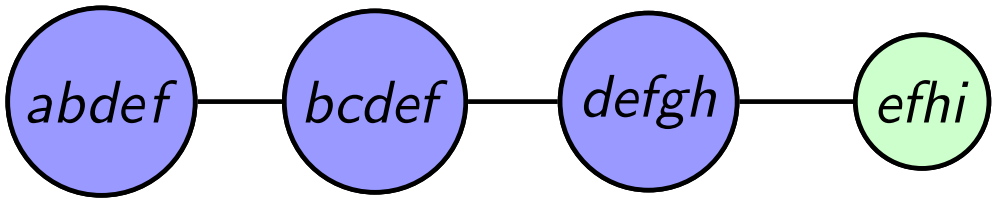
- For every edge $uv \in E$ some bag contains u and v

Path decompositions and pathwidth

Graph $G = (V, E)$



Path decomposition of G

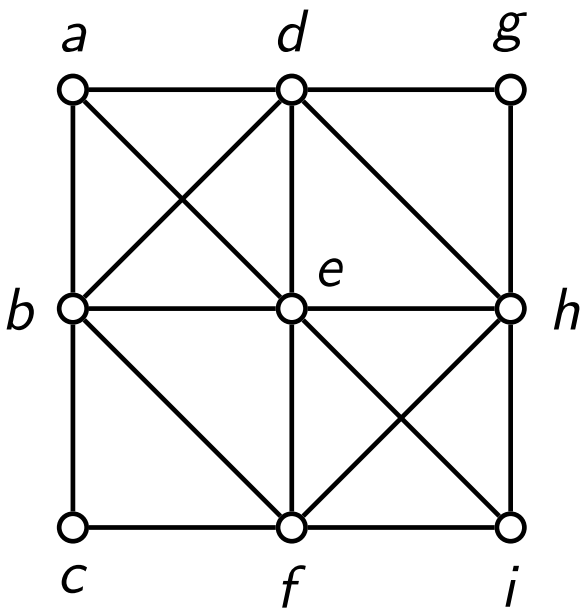


Path decomposition is a path of **bags** (subsets of V) such that

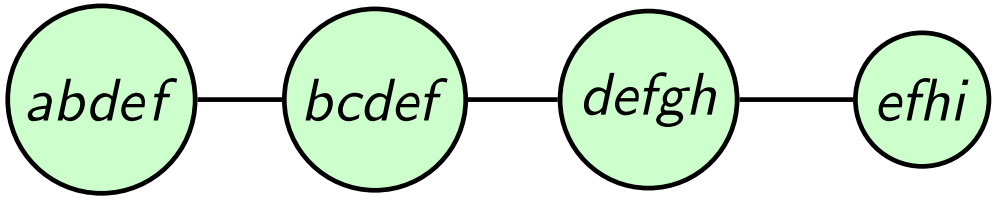
- For every edge $uv \in E$ some bag contains u and v
- For every vertex $v \in V$ bags containing v form nonempty subpath (connected!)

Path decompositions and pathwidth

Graph $G = (V, E)$



Path decomposition of G



Path decomposition is a path of **bags** (subsets of V) such that

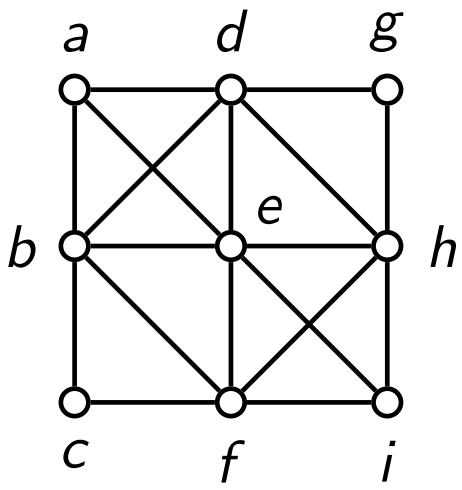
- For every edge $uv \in E$ some bag contains u and v
- For every vertex $v \in V$ bags containing v form nonempty subpath (connected!)

Width of the decomposition: maximum bag size -1 (here: 4).

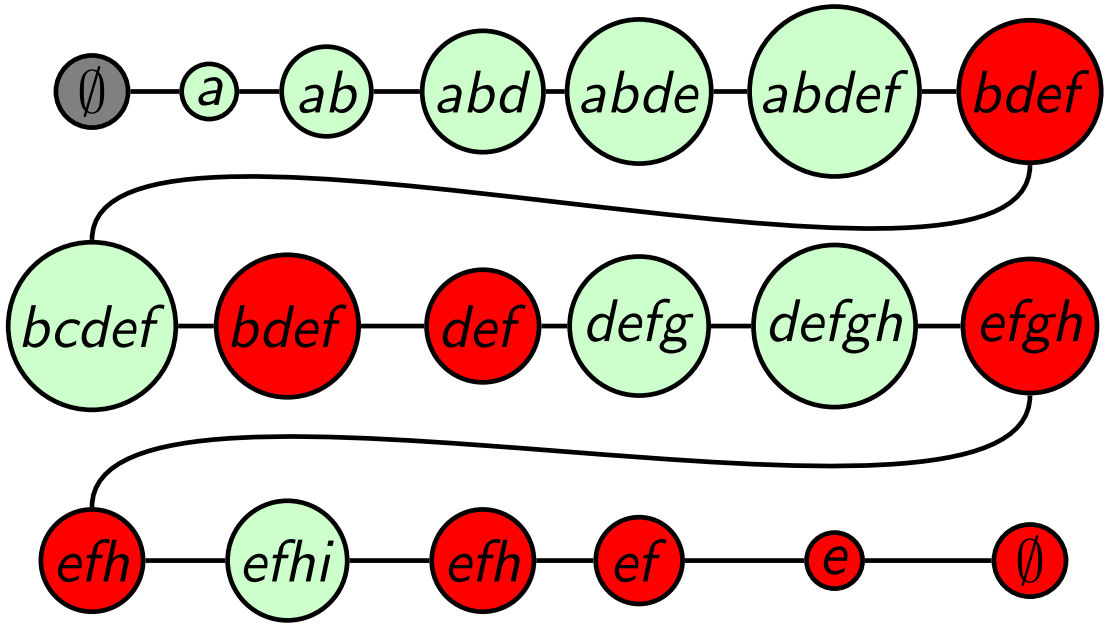
Pathwidth of G: minimum width of a decomposition of G .

Nice path decomposition

Graph $G = (V, E)$



Path decomposition of G



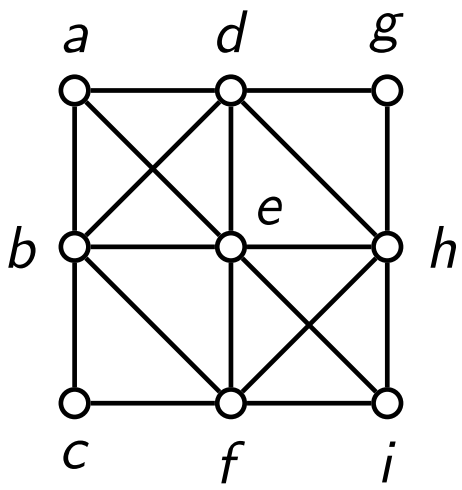
Nice path decomposition is a path decomposition $X_1, \ldots, X_r$ such that

- $X_1 = X_r = \emptyset$
- For $i = 2, \ldots, r$ bag X_i is of one of two types:
  1. Introduce bag: $X_i = X_{i-1} \cup \{v\}$ for some $v \in V \setminus X_{i-1}$,
  2. Forget bag: $X_i = X_{i-1} \setminus \{v\}$ for some $v \in X_{i-1}$.

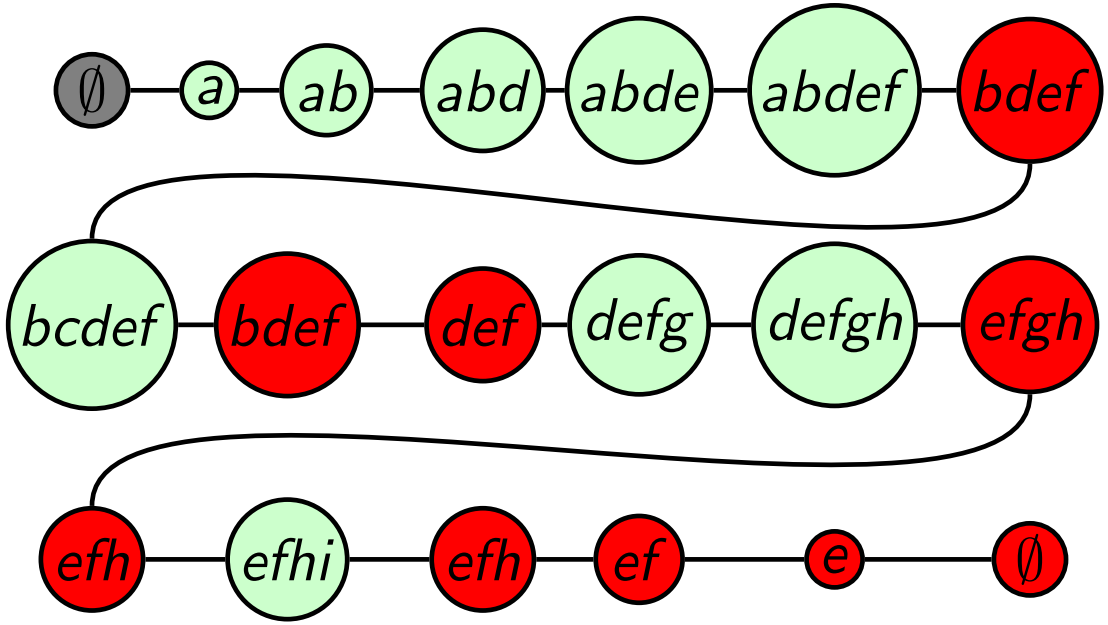
(Easy) Theorem 1: Graph has a path decomposition of width w iff it has a nice path decomposition of width w (at most $2w$ times longer).

Nice path decomposition

Graph $G = (V, E)$



Path decomposition of G



Nice path decomposition is a path decomposition $X_1, \ldots, X_r$ such that

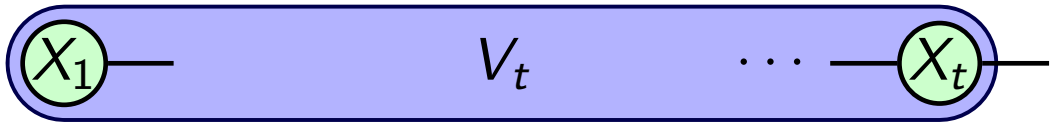
- $X_1 = X_r = \emptyset$
- For $i = 2, \ldots, r$ bag X_i is of one of two types:
  1. Introduce bag: $X_i = X_{i-1} \cup \{v\}$ for some $v \in V \setminus X_{i-1}$,
  2. Forget bag: $X_i = X_{i-1} \setminus \{v\}$ for some $v \in X_{i-1}$.

(Easy) Theorem 2: There is a mapping of G to p intervals iff G has nice path decomposition with bags of size p.

Dynamic programming for Independent Set

For every node t of a path decomposition of graph G:

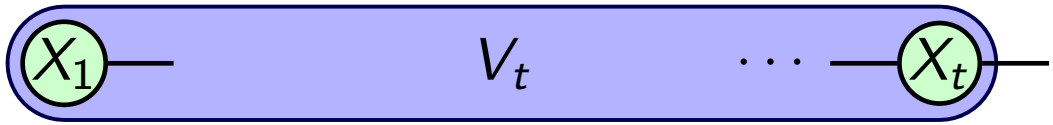
- X_t = the bag at t,
- $V_t = \bigcup_{i \leq t} X_i$



Dynamic programming for Independent Set

For every node t of a path decomposition of graph G:

- X_t = the bag at t,
- $V_t = \bigcup_{i \leq t} X_i$



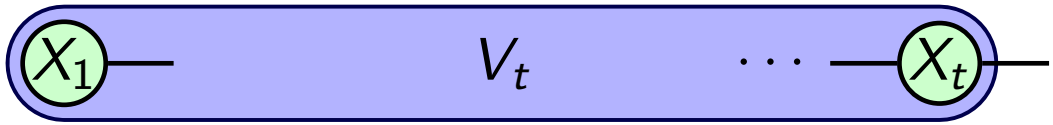
For every $f : X_i \to \{0, 1\}$ we want to compute the following.

$$\Psi[i, f] = \max\{|S| : S \text{ is an independent set in } V_i \text{ and } S \cap X_i = f^{-1}(1)\}.$$

Dynamic programming for Independent Set

For every node t of a path decomposition of graph G:

- X_t = the bag at t,
- $V_t = \bigcup_{i \leq t} X_i$



For every $f : X_i \to \{0, 1\}$ we want to compute the following.

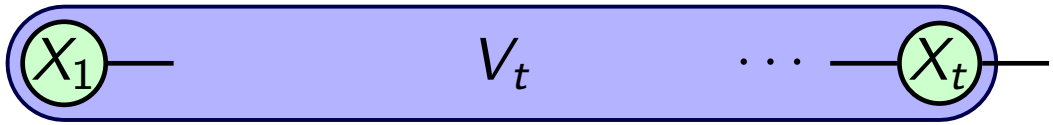
$$\Psi[i, f] = \max\{|S| : S \text{ is an independent set in } V_i \text{ and } S \cap X_i = f^{-1}(1)\}.$$

- We now give recursive equations on $\Psi[\cdot, \cdot]$.

Dynamic programming for Independent Set

For every node t of a path decomposition of graph G:

- X_t = the bag at t,
- $V_t = \bigcup_{i \leq t} X_i$



For every $f : X_i \to \{0, 1\}$ we want to compute the following.

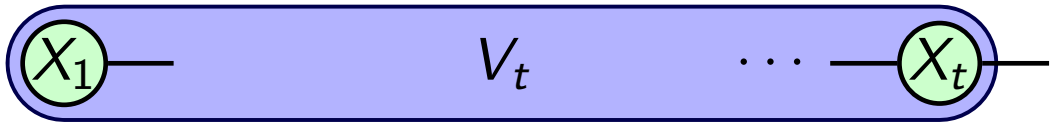
$$\Psi[i, f] = \max\{|S| : S \text{ is an independent set in } V_i \text{ and } S \cap X_i = f^{-1}(1)\}.$$

- We now give recursive equations on $\Psi[\cdot, \cdot]$.
- $\Psi[i, \emptyset] = 0$ if $i = 1$.

Dynamic programming for Independent Set

For every node t of a path decomposition of graph G:

- X_t = the bag at t,
- $V_t = \bigcup_{i \leq t} X_i$



For every $f : X_i \to \{0, 1\}$ we want to compute the following.

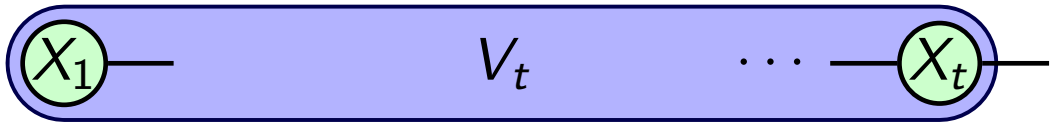
$$\Psi[i, f] = \max\{|S| : S \text{ is an independent set in } V_i \text{ and } S \cap X_i = f^{-1}(1)\}.$$

- We now give recursive equations on $\Psi[\cdot, \cdot]$.
- $\Psi[i, \emptyset] = 0$ if $i = 1$.
- The answer is $\Psi[r, \emptyset]$.

Dynamic programming for Independent Set

For every node t of a path decomposition of graph G:

- X_t = the bag at t,
- $V_t = \bigcup_{i \leq t} X_i$



For every $f : X_i \to \{0, 1\}$ we want to compute the following.

$$\Psi[i, f] = \max\{|S| : S \text{ is an independent set in } V_i \text{ and } S \cap X_i = f^{-1}(1)\}.$$

- We now give recursive equations on $\Psi[\cdot, \cdot]$.
- $\Psi[i, \emptyset] = 0$ if $i = 1$.
- The answer is $\Psi[r, \emptyset]$.
- We follow convention that $\Psi[i, f] = -\infty$ if $f^{-1}(1)$ is already not independent.

DP for Independent Set: introduce node

Introduce node i such that $X_i = X_{i-1} \cup \{v\}$.

DP for Independent Set: introduce node

Introduce node i such that $X_i = X_{i-1} \cup \{v\}$.

Goal: For $f : X_i \to \{0, 1\}$ compute

$$\Psi[i, f] = \max\{|S| : S \text{ is an independent set in } V_i \text{ and } S \cap X_i = f^{-1}(1)\}.$$

Let $f' = f|_{X_{i-1}}$.

DP for Independent Set: introduce node

Introduce node i such that $X_i = X_{i-1} \cup \{v\}$.

Goal: For $f : X_i \to \{0, 1\}$ compute

$$\Psi[i, f] = \max\{|S| : S \text{ is an independent set in } V_i \text{ and } S \cap X_i = f^{-1}(1)\}.$$

Let $f' = f|_{X_{i-1}}$.

If $f(v) = 0$ then

$$\Psi[i, f] = \Psi[i - 1, f'].$$

If $f(v) = 1$ and there is $v' \in X_i$ s.t. $vv' \in E(G)$ and $f(v') = 1$, then

$$\Psi[i, f] = -\infty.$$

Otherwise, we have

$$\Psi[i, f] = 1 + \Psi[i - 1, f'].$$

DP for Independent Set: forget node

Forget node i such that $X_i = X_{i-1} \setminus \{v\}$.

DP for Independent Set: forget node

Forget node i such that $X_i = X_{i-1} \setminus \{v\}$.

Goal: For $f : X_i \to \{0, 1\}$ compute

$$\Psi[i, f] = \max\{|S| : S \text{ is an independent set in } V_i \text{ and } S \cap X_i = f^{-1}(1)\}.$$

If there is $w' \in X_i$ such that $ww' \in E(G)$ and $f(w') = 1$, then

$$\Psi[i, f] = \Psi[i - 1, f[i \to 0]],$$

where

$$f[i \to \alpha](x) = \begin{cases} f(x) & x \neq v \\ \alpha & x = v. \end{cases}$$

Otherwise, we have

$$\Psi[i, f] = \max(\Psi[i - 1, f[w \to 0]], \Psi[i - 1, f[w \to 1]]).$$

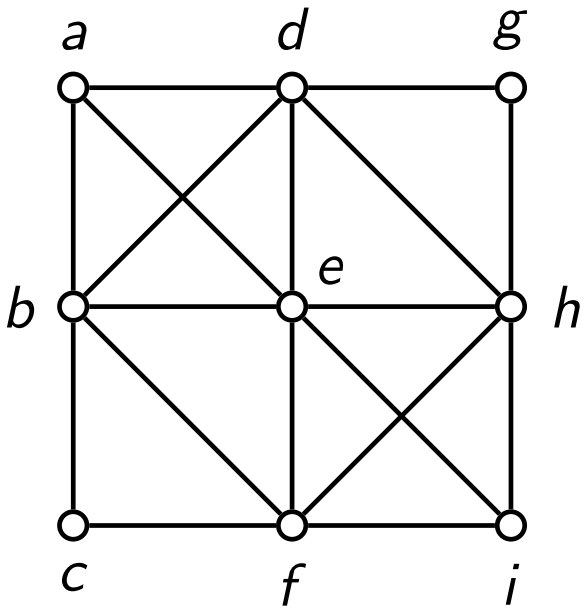
DP for Independent Set: conclusion for pathwidth

Theorem

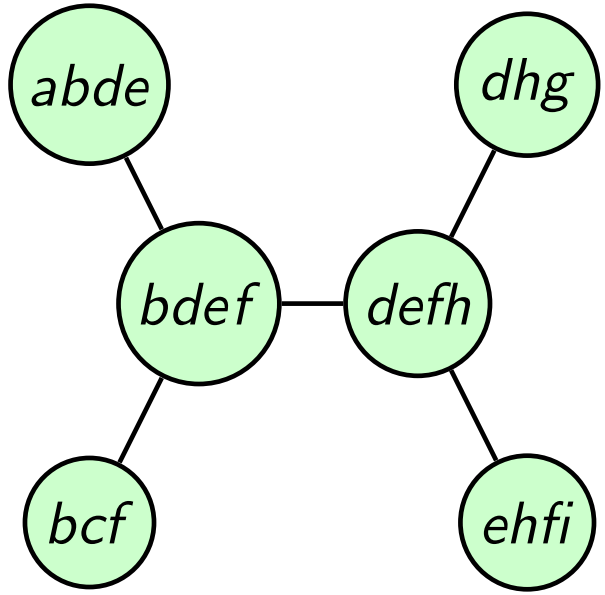
Given an n-vertex graph and its path decomposition of width p, Independent Set *can be solved in time* $2^p n^{O(1)} p^{O(1)}$.

Tree decompositions and treewidth

Graph $G = (V, E)$



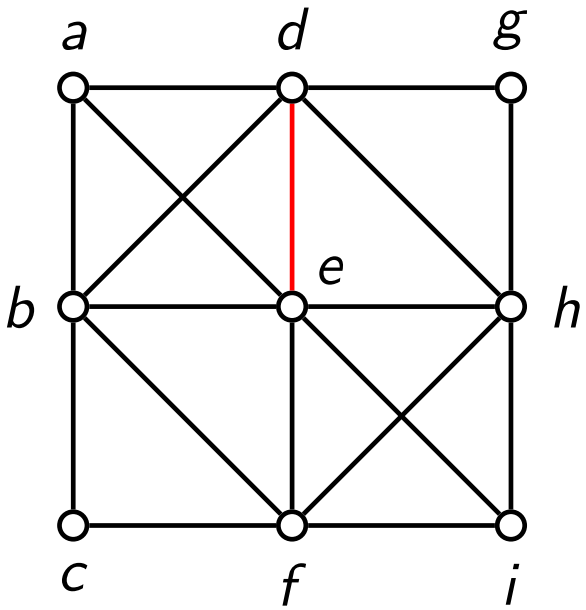
Tree decomposition of G



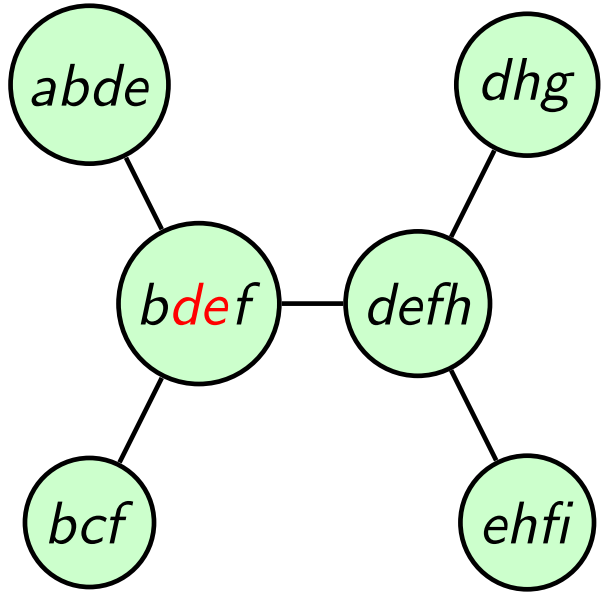
Tree decomposition is a tree of **bags** (subsets of V)

Tree decompositions and treewidth

Graph $G = (V, E)$



Tree decomposition of G

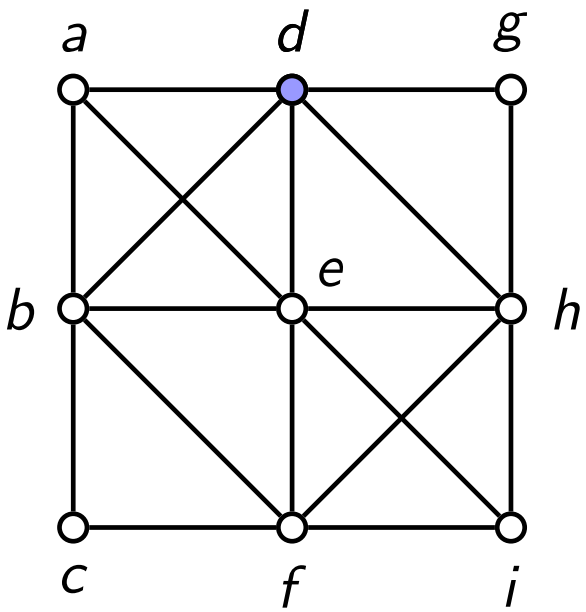


Tree decomposition is a tree of **bags** (subsets of V) such that

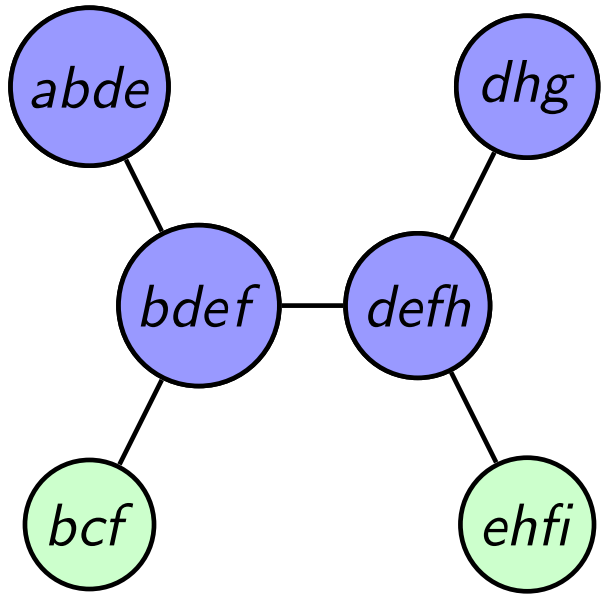
- For every edge $uv \in E$ some bag contains u and v

Tree decompositions and treewidth

Graph $G = (V, E)$



Tree decomposition of G

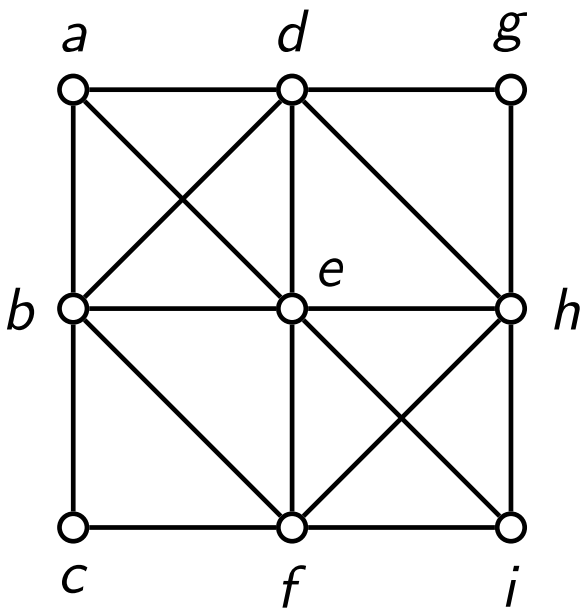


Tree decomposition is a tree of **bags** (subsets of V) such that

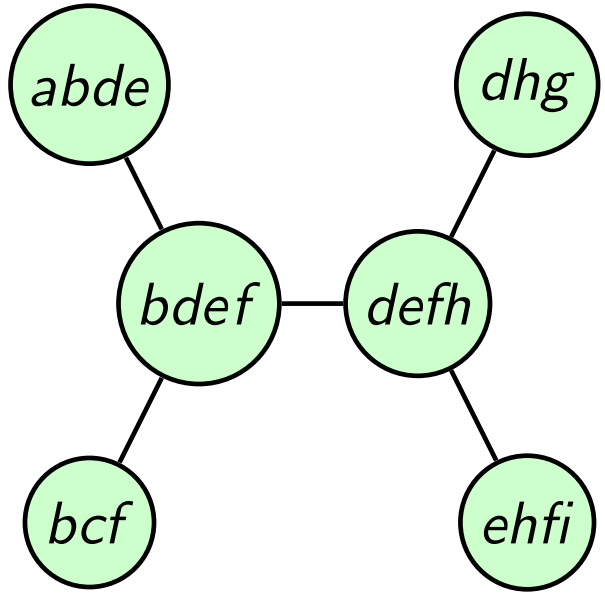
- For every edge $uv \in E$ some bag contains u and v
- For every vertex $v \in V$ bags containing v form nonempty subtree (connected!)

Tree decompositions and treewidth

Graph $G = (V, E)$



Tree decomposition of G



Tree decomposition is a tree of **bags** (subsets of V) such that

- For every edge $uv \in E$ some bag contains u and v
- For every vertex $v \in V$ bags containing v form nonempty subtree (connected!)

Width of the decomposition: maximum bag size -1 (here: 3).

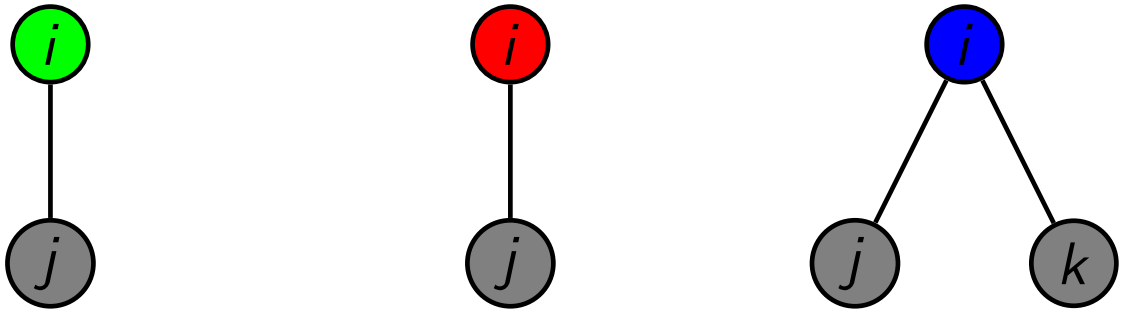
Treewidth of G: minimum width of a decomposition of G .

Nice tree decompositions

Nice tree decomposition is a rooted tree decomp. $(T, \{X_i\}_{i \in V(T)})$ s.t.

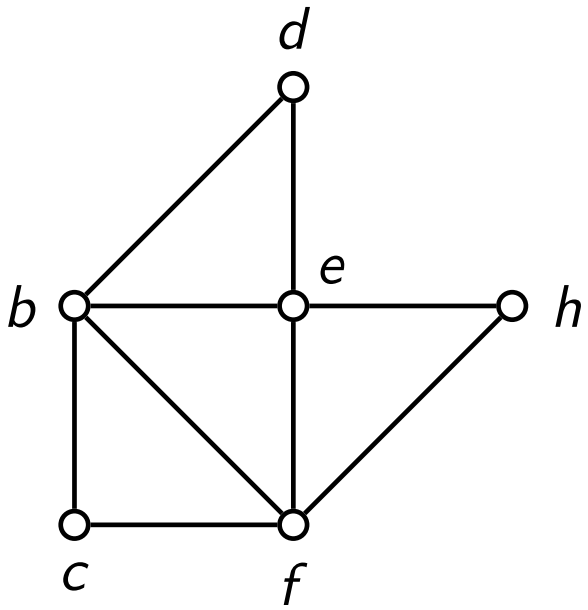
- $X_i = \emptyset$ for every leaf i,
- $X_r = \emptyset$ for the root r,
- Every non-leaf node $i \in V(T)$ is of one of three types:
  1. Introduce: $X_i = X_j \cup \{v\}$ for $v \in V \setminus X_j$, where j is the only child of i,
  2. Forget: $X_i = X_j \setminus \{v\}$ for $v \in X_j$, where j is the only child of i,
  3. Join: $X_i = X_j = X_k$ where j and k are the only two children of i.

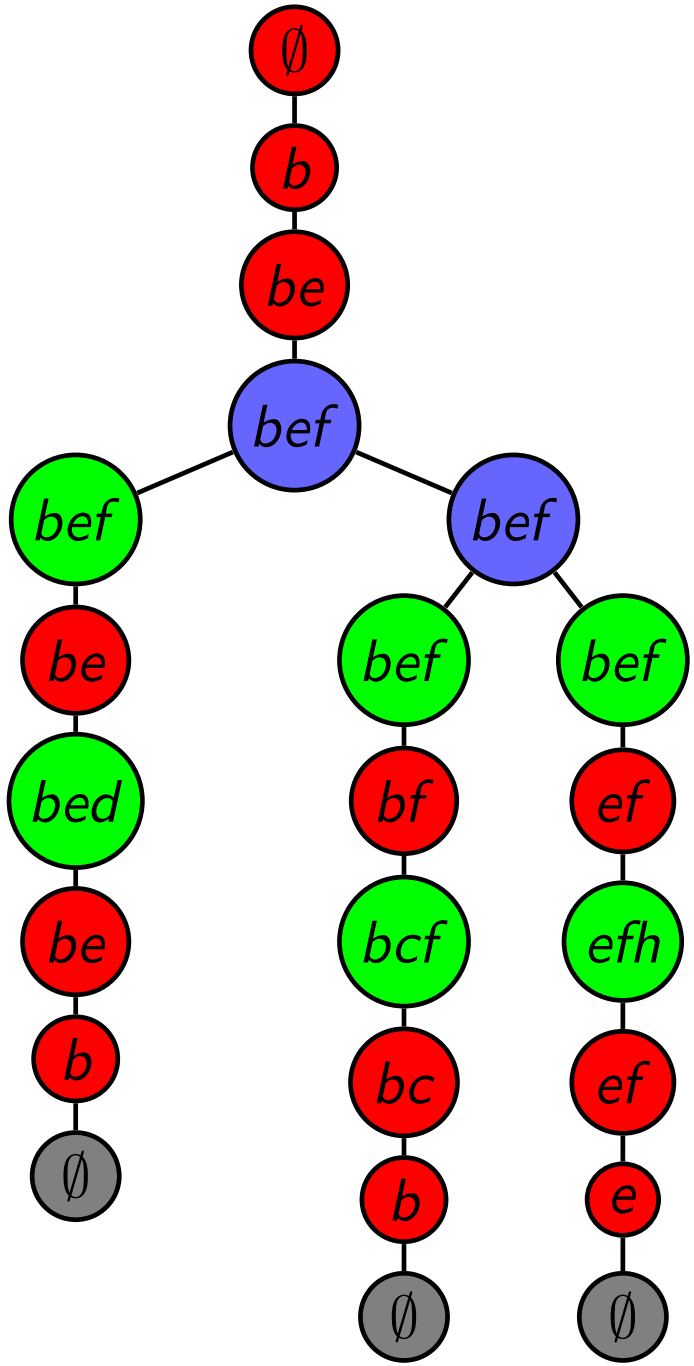
$X_i = X_j \cup \{v\}$ $X_i = X_j \setminus \{v\}$ $X_i = X_j = X_k$



Introduce node Forget node Join node

Example



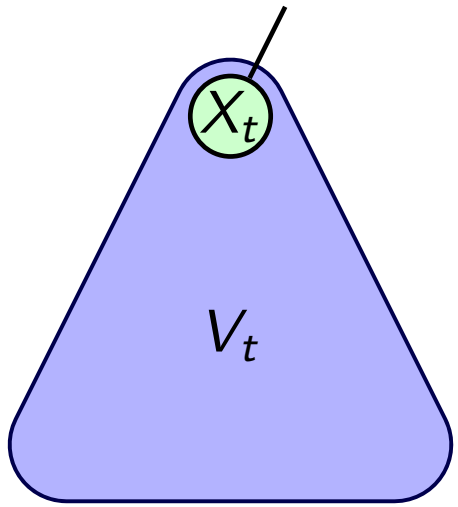


(Easy) Theorem 1: Graph has a tree decomposition of width w iff it has a nice tree decomposition of width w (at most $2w$ times larger).

Dynamic programming for Independent Set cont'd

For every node t of a tree decomposition of graph G:

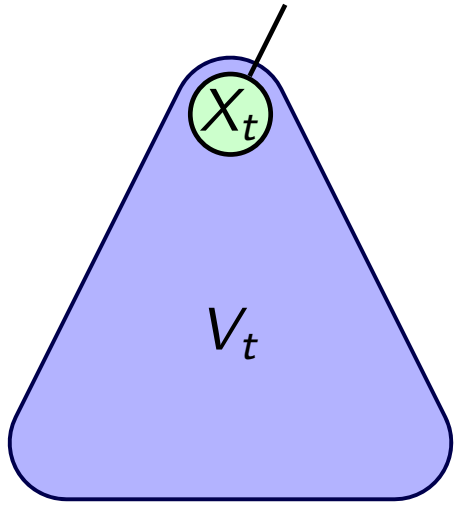
- X_t = the bag at t,
- V_t = union of all bags in the subtree rooted at t.



Dynamic programming for Independent Set cont'd

For every node t of a tree decomposition of graph G:

- X_t = the bag at t,
- V_t = union of all bags in the subtree rooted at t.



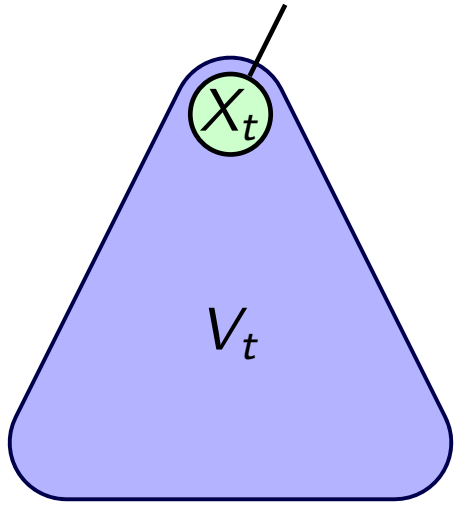
For every $f : X_i \to \{0, 1\}$ we want to compute the following.

$$\Psi[i, f] = \max\{|S| : S \text{ is an independent set in } V_i \text{ and } S \cap X_i = f^{-1}(1)\}.$$

Dynamic programming for Independent Set cont'd

For every node t of a tree decomposition of graph G:

- X_t = the bag at t,
- V_t = union of all bags in the subtree rooted at t.



For every $f : X_i \to \{0, 1\}$ we want to compute the following.

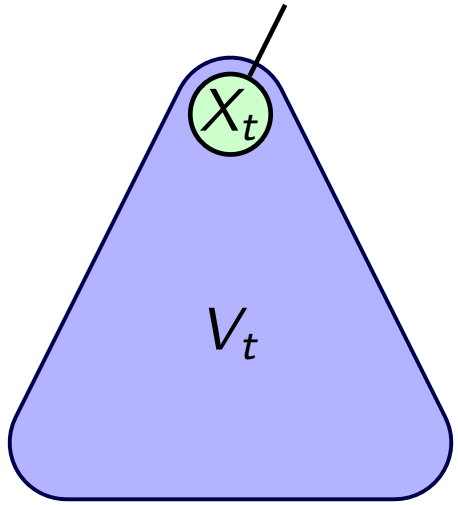
$$\Psi[i, f] = \max\{|S| : S \text{ is an independent set in } V_i \text{ and } S \cap X_i = f^{-1}(1)\}.$$

- Computation in the bottom-up manner. The answer is $\Psi[r, \emptyset]$.

Dynamic programming for Independent Set cont'd

For every node t of a tree decomposition of graph G:

- X_t = the bag at t,
- V_t = union of all bags in the subtree rooted at t.



For every $f : X_i \to \{0, 1\}$ we want to compute the following.

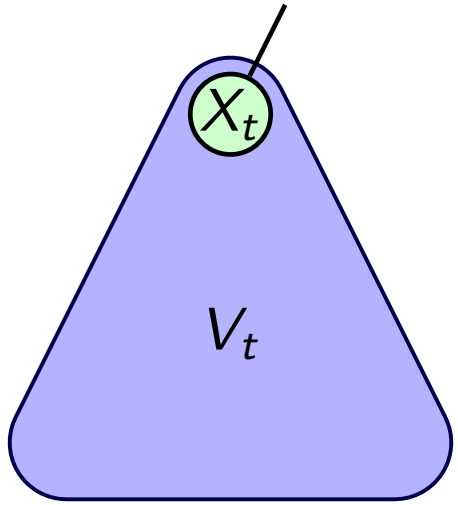
$$\Psi[i, f] = \max\{|S| : S \text{ is an independent set in } V_i \text{ and } S \cap X_i = f^{-1}(1)\}.$$

- Computation in the bottom-up manner. The answer is $\Psi[r, \emptyset]$.
- For leaves $\Psi[i, \emptyset] = 0$, as before.

Dynamic programming for Independent Set cont'd

For every node t of a tree decomposition of graph G:

- X_t = the bag at t,
- V_t = union of all bags in the subtree rooted at t.



For every $f : X_i \to \{0, 1\}$ we want to compute the following.

$$\Psi[i, f] = \max\{|S| : S \text{ is an independent set in } V_i \text{ and } S \cap X_i = f^{-1}(1)\}.$$

- Computation in the bottom-up manner. The answer is $\Psi[r, \emptyset]$.
- For leaves $\Psi[i, \emptyset] = 0$, as before.
- Introduce and forget nodes processed as before.

DP for Independent Set: join node

Join node i with children j and k.

DP for Independent Set: join node

Join node i with children j and k.

Goal: For $f : X_i \to \{0, 1\}$ compute

$$\Psi[i, f] = \max\{|S| : S \text{ is an independent set in } V_i \text{ and } S \cap X_i = f^{-1}(1)\}.$$

DP for Independent Set: join node

Join node i with children j and k.

Goal: For $f : X_i \to \{0, 1\}$ compute

$$\Psi[i, f] = \max\{|S| : S \text{ is an independent set in } V_i \text{ and } S \cap X_i = f^{-1}(1)\}.$$

We have the following:

$$\Psi[i, f] = \Psi[j, f] + \Psi[k, f] - |f^{-1}(1)|.$$

DP for Independent Set: conclusion

Theorem

Given an n-vertex graph and its tree decomposition of width t, Independent Set can be solved in time $2^t n^{O(1)} t^{O(1)}$.

- Formal proofs are tedious, but often necessary.

DP for Independent Set: conclusion

Theorem

Given an n-vertex graph and its tree decomposition of width t, Independent Set *can be solved in time $2^t n^{O(1)} t^{O(1)}$.*

- Formal proofs are tedious, but often necessary.
- There are tricks to make your life easier:

DP for Independent Set: conclusion

Theorem

Given an n-vertex graph and its tree decomposition of width t, Independent Set can be solved in time $2^t n^{O(1)} t^{O(1)}$.

- Formal proofs are tedious, but often necessary.
- There are tricks to make your life easier:
  - Solution cost does not include the cost of the bag $\Rightarrow$ No double-counting in join.

DP for Independent Set: conclusion

Theorem

Given an n-vertex graph and its tree decomposition of width t, Independent Set *can be solved in time* $2^t n^{O(1)} t^{O(1)}$.

- Formal proofs are tedious, but often necessary.
- There are tricks to make your life easier:
  - Solution cost does not include the cost of the bag $\Rightarrow$ No double-counting in join.
  - **Introduce edge** node: think of edges as introduced one by one to the graph, and not simultaneously when a vertex is introduced.

DP for Independent Set: conclusion

Theorem

Given an n-vertex graph and its tree decomposition of width t, Independent Set *can be solved in time* $2^t n^{O(1)} t^{O(1)}$.

- Formal proofs are tedious, but often necessary.
- There are tricks to make your life easier:
  - Solution cost does not include the cost of the bag $\Rightarrow$ No double-counting in join.
  - **Introduce edge** node: think of edges as introduced one by one to the graph, and not simultaneously when a vertex is introduced.
  - Introduce edges just before forgetting a vertex $\Rightarrow$ Bags do not have edges when joining.

DP for Independent Set: conclusion

Theorem

Given an n-vertex graph and its tree decomposition of width t, Independent Set *can be solved in time* $2^t n^{O(1)} t^{O(1)}$.

- Formal proofs are tedious, but often necessary.
- There are tricks to make your life easier:
  - Solution cost does not include the cost of the bag $\Rightarrow$ No double-counting in join.
  - **Introduce edge** node: think of edges as introduced one by one to the graph, and not simultaneously when a vertex is introduced.
  - Introduce edges just before forgetting a vertex $\Rightarrow$ Bags do not have edges when joining.
  - Everybody discovers his or her own tricks after writing the first five DPs on treewidth.

Treewidth: background

- Introduced in 80-s independently in several different contexts.

Treewidth: background

- Introduced in 80-s independently in several different contexts.
- Variant used here is due to Robertson and Seymour.

Treewidth: background

- Introduced in 80-s independently in several different contexts.
- Variant used here is due to Robertson and Seymour.
- Several related width parameters, characterizing the same concept;

Treewidth: background

- Introduced in 80-s independently in several different contexts.
- Variant used here is due to Robertson and Seymour.
- Several related width parameters, characterizing the same concept;
  - partial k-trees

Treewidth: background

- Introduced in 80-s independently in several different contexts.
- Variant used here is due to Robertson and Seymour.
- Several related width parameters, characterizing the same concept;
  - partial k-trees
  - branchwidth

Treewidth: background

- Introduced in 80-s independently in several different contexts.
- Variant used here is due to Robertson and Seymour.
- Several related width parameters, characterizing the same concept;
  - partial k-trees
  - branchwidth
- **Viewpoints:**

Treewidth: background

- Introduced in 80-s independently in several different contexts.
- Variant used here is due to Robertson and Seymour.
- Several related width parameters, characterizing the same concept;
  - partial k-trees
  - branchwidth
- **Viewpoints**:
  - **Algorithms**: fast FPT algorithms parameterized by treewidth, e.g. $2^t \cdot t^{\mathcal{O}(1)} \cdot n$ for IS.

Treewidth: background

- Introduced in 80-s independently in several different contexts.
- Variant used here is due to Robertson and Seymour.
- Several related width parameters, characterizing the same concept;
  - partial k-trees
  - branchwidth
- **Viewpoints**:
  - **Algorithms**: fast FPT algorithms parameterized by treewidth, e.g. $2^t \cdot t^{\mathcal{O}(1)} \cdot n$ for IS.
  - **Graph theory**: crucial tool in the Graph Minors project; treewidth and grid minors.

Treewidth: background

- Introduced in 80-s independently in several different contexts.
- Variant used here is due to Robertson and Seymour.
- Several related width parameters, characterizing the same concept;
  - partial k-trees
  - branchwidth
- **Viewpoints**:
  - **Algorithms**: fast FPT algorithms parameterized by treewidth, e.g. $2^t \cdot t^{\mathcal{O}(1)} \cdot n$ for IS.
  - **Graph theory**: crucial tool in the Graph Minors project; treewidth and grid minors.
  - **Logic**: graph classes of bounded treewidth exactly correspond to classes where $\mathbf{MSO}_2$ theory is decidable.

Treewidth: background

- Introduced in 80-s independently in several different contexts.
- Variant used here is due to Robertson and Seymour.
- Several related width parameters, characterizing the same concept;
  - partial k-trees
  - branchwidth
- **Viewpoints**:
  - **Algorithms**: fast FPT algorithms parameterized by treewidth, e.g. $2^t \cdot t^{\mathcal{O}(1)} \cdot n$ for IS.
  - **Graph theory**: crucial tool in the Graph Minors project; treewidth and grid minors.
  - **Logic**: graph classes of bounded treewidth exactly correspond to classes where $\mathbf{MSO}_2$ theory is decidable.
- **Overall**: the concept of bounded treewidth graphs presents a fundamental link between various fields of combinatorics and computer science.

Computation of treewidth

- If we want to run a dynamic programming algorithm, we need to have a decomposition of small width.

Computation of treewidth

- If we want to run a dynamic programming algorithm, we need to have a decomposition of small width.
- Computing treewidth optimally is NP-hard.

Computation of treewidth

- If we want to run a dynamic programming algorithm, we need to have a decomposition of small width.
- Computing treewidth optimally is NP-hard.
- Moreover, constant-factor approximation is not possible under the Small Set Expansion Conjecture.

Computation of treewidth

- If we want to run a dynamic programming algorithm, we need to have a decomposition of small width.
- Computing treewidth optimally is NP-hard.
- Moreover, constant-factor approximation is not possible under the Small Set Expansion Conjecture.
- Best known poly-time approximation has apx ratio $\mathcal{O}(\sqrt{\log OPT})$.

Computation of treewidth

- If we want to run a dynamic programming algorithm, we need to have a decomposition of small width.
- Computing treewidth optimally is NP-hard.
- Moreover, constant-factor approximation is not possible under the Small Set Expansion Conjecture.
- Best known poly-time approximation has apx ratio $\mathcal{O}(\sqrt{\log OPT})$.
- **Idea**: If we run an FPT algorithm anyways, we can afford FPT running time of computing treewidth.

Algorithms for treewidth

Input: Graph G and integer k

Robertson-Seymour 4-approximation (GM XIII, '95)

Output: Either conclusion that $\mathrm{tw}(G) > k$, or a tree decomposition of width $\leq 4k + 4$

Runtime: $\mathcal{O}(8^k k^2 \cdot n^2)$

Bodlaender's algorithm ('96)

Output: Either conclusion that $\mathrm{tw}(G) > k$, or a tree decomposition of width $\leq k$

Runtime: $k^{\mathcal{O}(k^3)} \cdot n$

BDDFLP algorithm ('13)

Output: Either conclusion that $\mathrm{tw}(G) > k$, or a tree decomposition of width $\leq 5k + 4$

Runtime: $2^{\mathcal{O}(k)} \cdot n$

Graph minors

- Suppose we have two graphs H and G. Then H is a minor of G if there exists a minor model $(I_u)_{u \in V(H)}$ of H in G:

Graph minors

- Suppose we have two graphs H and G. Then H is a minor of G if there exists a minor model $(I_u)_{u \in V(H)}$ of H in G:
  - I_u are pairwise disjoint sets of vertices in G;

Graph minors

- Suppose we have two graphs H and G. Then H is a minor of G if there exists a minor model $(I_u)_{u \in V(H)}$ of H in G:
  - I_u are pairwise disjoint sets of vertices in G;
  - Each I_u induces a connected graph in G;

Graph minors

- Suppose we have two graphs H and G. Then H is a minor of G if there exists a minor model $(I_u)_{u \in V(H)}$ of H in G:
  - I_u are pairwise disjoint sets of vertices in G;
  - Each I_u induces a connected graph in G;
  - For each $uv \in E(H)$, there is an edge between some vertex of I_u and some vertex of I_v.

Graph minors

- Suppose we have two graphs H and G. Then H is a minor of G if there exists a minor model $(I_u)_{u \in V(H)}$ of H in G:
  - I_u are pairwise disjoint sets of vertices in G;
  - Each I_u induces a connected graph in G;
  - For each $uv \in E(H)$, there is an edge between some vertex of I_u and some vertex of I_v.
- Equivalently, H can be obtained from G using vertex deletions, edge deletions, and edge contractions.

Graph minors

- Suppose we have two graphs H and G. Then H is a minor of G if there exists a minor model $(I_u)_{u \in V(H)}$ of H in G:
  - I_u are pairwise disjoint sets of vertices in G;
  - Each I_u induces a connected graph in G;
  - For each $uv \in E(H)$, there is an edge between some vertex of I_u and some vertex of I_v.
- Equivalently, H can be obtained from G using vertex deletions, edge deletions, and edge contractions.
- Graph class $\mathcal{C}$ is minor-closed if it is closed under taking minors.
  Examples:

Graph minors

- Suppose we have two graphs H and G. Then H is a minor of G if there exists a minor model $(I_u)_{u \in V(H)}$ of H in G:
  - I_u are pairwise disjoint sets of vertices in G;
  - Each I_u induces a connected graph in G;
  - For each $uv \in E(H)$, there is an edge between some vertex of I_u and some vertex of I_v.
- Equivalently, H can be obtained from G using vertex deletions, edge deletions, and edge contractions.
- Graph class $\mathcal{C}$ is minor-closed if it is closed under taking minors.
  Examples:
  - Planar graphs.

Graph minors

- Suppose we have two graphs H and G. Then H is a minor of G if there exists a minor model $(I_u)_{u \in V(H)}$ of H in G:
  - I_u are pairwise disjoint sets of vertices in G;
  - Each I_u induces a connected graph in G;
  - For each $uv \in E(H)$, there is an edge between some vertex of I_u and some vertex of I_v.
- Equivalently, H can be obtained from G using vertex deletions, edge deletions, and edge contractions.
- Graph class $\mathcal{C}$ is minor-closed if it is closed under taking minors.
  Examples:
  - Planar graphs.
  - Graphs embeddable into a fixed surface.

Graph minors

- Suppose we have two graphs H and G. Then H is a minor of G if there exists a minor model $(I_u)_{u \in V(H)}$ of H in G:
  - I_u are pairwise disjoint sets of vertices in G;
  - Each I_u induces a connected graph in G;
  - For each $uv \in E(H)$, there is an edge between some vertex of I_u and some vertex of I_v.
- Equivalently, H can be obtained from G using vertex deletions, edge deletions, and edge contractions.
- Graph class $\mathcal{C}$ is minor-closed if it is closed under taking minors.
  Examples:
  - Planar graphs.
  - Graphs embeddable into a fixed surface.
  - Graphs of treewidth at most k.

The Graph Minors theorem

- **Kuratowski, Wagner**: Graph is planar iff it does not contain K_5 or $K_{3,3}$ as a minor.

The Graph Minors theorem

- **Kuratowski, Wagner**: Graph is planar iff it does not contain K_5 or $K_{3,3}$ as a minor.
- **Wagner's conjecture**: Every minor-closed class is characterized by a finite number of forbidden minors.

The Graph Minors theorem

- **Kuratowski, Wagner**: Graph is planar iff it does not contain K_5 or $K_{3,3}$ as a minor.
- **Wagner's conjecture**: Every minor-closed class is characterized by a finite number of forbidden minors.
- **Robertson, Seymour**: This is indeed true.

The Graph Minors theorem

- **Kuratowski, Wagner**: Graph is planar iff it does not contain K_5 or $K_{3,3}$ as a minor.
- **Wagner's conjecture**: Every minor-closed class is characterized by a finite number of forbidden minors.
- **Robertson, Seymour**: This is indeed true.
  - The proof spans over 700 pages and has set up a new branch of graph theory.

The Graph Minors theorem

- **Kuratowski, Wagner**: Graph is planar iff it does not contain K_5 or $K_{3,3}$ as a minor.
- **Wagner's conjecture**: Every minor-closed class is characterized by a finite number of forbidden minors.
- **Robertson, Seymour**: This is indeed true.
  - The proof spans over 700 pages and has set up a new branch of graph theory.
  - For graphs of treewidth k, or of genus at most k, there is list of forbidden minors whose length depends only on k.

The Graph Minors theorem

- **Kuratowski, Wagner**: Graph is planar iff it does not contain K_5 or $K_{3,3}$ as a minor.
- **Wagner's conjecture**: Every minor-closed class is characterized by a finite number of forbidden minors.
- **Robertson, Seymour**: This is indeed true.
  - The proof spans over 700 pages and has set up a new branch of graph theory.
  - For graphs of treewidth k, or of genus at most k, there is list of forbidden minors whose length depends only on k.
- **Main technical result**: A graph that is H-minor-free is can be composed from pieces that are "almost" embeddable into a surface depending on H only.

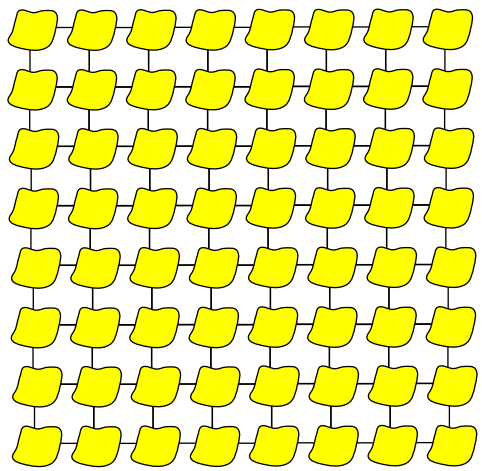
The Graph Minors theorem

- **Kuratowski, Wagner**: Graph is planar iff it does not contain K_5 or $K_{3,3}$ as a minor.
- **Wagner's conjecture**: Every minor-closed class is characterized by a finite number of forbidden minors.
- **Robertson, Seymour**: This is indeed true.
  - The proof spans over 700 pages and has set up a new branch of graph theory.
  - For graphs of treewidth k, or of genus at most k, there is list of forbidden minors whose length depends only on k.
- **Main technical result**: A graph that is H-minor-free is can be composed from pieces that are "almost" embeddable into a surface depending on H only.
- Treewidth is one of the main technical tools.

Grid Minor Theorem

Grid Minor Theorem

There exists a function $g : \mathbb{N} \to \mathbb{N}$ such that if $\mathrm{tw}(G) > g(k)$, then G contains $k \times k$ grid as a minor.

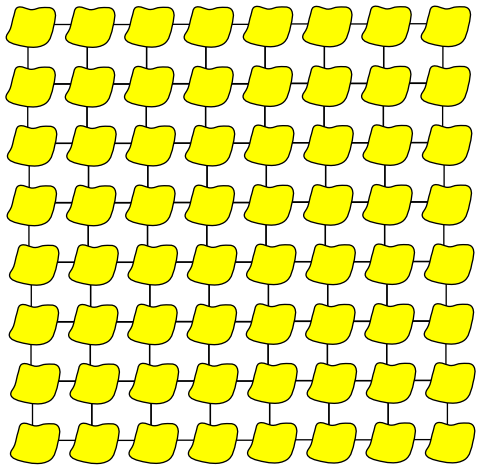


Grid Minor Theorem

Grid Minor Theorem

There exists a function $g : \mathbb{N} \to \mathbb{N}$ such that if $\mathrm{tw}(G) > g(k)$, then G contains $k \times k$ grid as a minor.

- Original proof: no elementary bound on g.

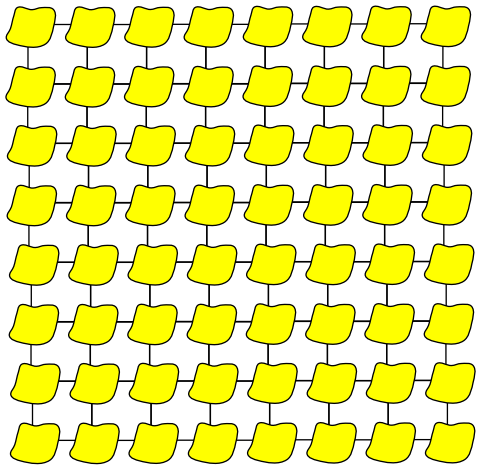


Grid Minor Theorem

Grid Minor Theorem

There exists a function $g : \mathbb{N} \to \mathbb{N}$ such that if $\mathrm{tw}(G) > g(k)$, then G contains $k \times k$ grid as a minor.

- Original proof: no elementary bound on g.
- RST: $g(k) = 2^{\mathcal{O}(k^5)}$, then a series of improvements.

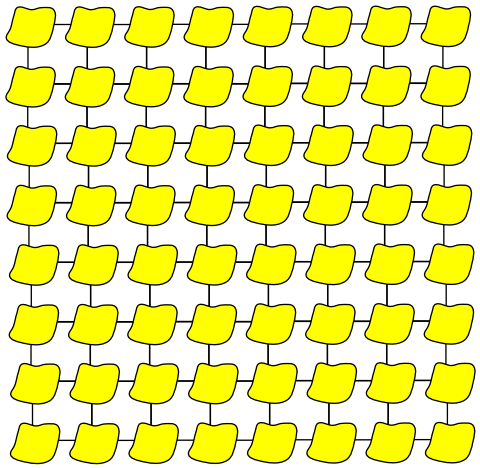


Grid Minor Theorem

Grid Minor Theorem

There exists a function $g : \mathbb{N} \to \mathbb{N}$ such that if $\mathrm{tw}(G) > g(k)$, then G contains $k \times k$ grid as a minor.

- Original proof: no elementary bound on g.
- RST: $g(k) = 2^{\mathcal{O}(k^5)}$, then a series of improvements.
- Chekuri&Chuzhoy'13: $g(k) = \mathcal{O}(k^{99})$.

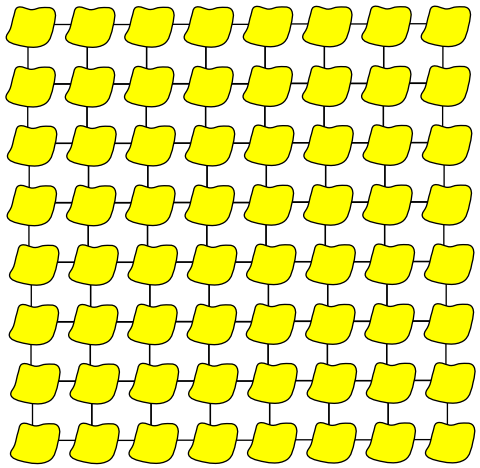


Grid Minor Theorem

Grid Minor Theorem

There exists a function $g : \mathbb{N} \to \mathbb{N}$ such that if $\mathrm{tw}(G) > g(k)$, then G contains $k \times k$ grid as a minor.

- Original proof: no elementary bound on g.
- RST: $g(k) = 2^{\mathcal{O}(k^5)}$, then a series of improvements.
- Chekuri&Chuzhoy'13: $g(k) = \mathcal{O}(k^{99})$.
- **Important**: If G is planar (or H-minor-free), then $g(k)$ is **linear**.



Implications for planar graphs

- **Corollary**: A planar graph on n vertices has treewidth $\mathcal{O}(\sqrt{n})$.

Implications for planar graphs

- **Corollary**: A planar graph on n vertices has treewidth $\mathcal{O}(\sqrt{n})$.
  - **Proof**: If it had $\lceil\sqrt{n+1}\rceil \times \lceil\sqrt{n+1}\rceil$ grid as a minor, it would have $n+1$ vertices.

Implications for planar graphs

- **Corollary**: A planar graph on n vertices has treewidth $\mathcal{O}(\sqrt{n})$.
  - **Proof**: If it had $\lceil\sqrt{n+1}\rceil \times \lceil\sqrt{n+1}\rceil$ grid as a minor, it would have $n+1$ vertices.
  - **Note**: Alternative proof via Lipton-Tarjan planar separator theorem.

Implications for planar graphs

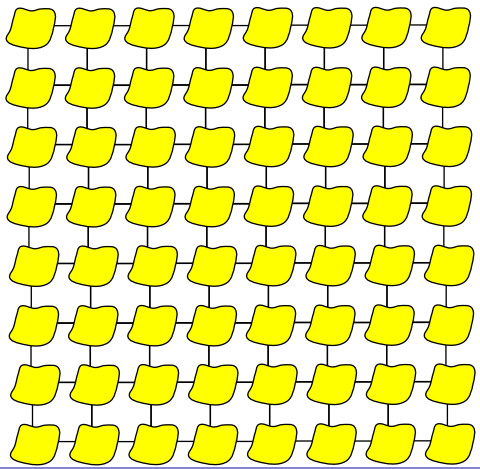
- **Corollary**: A planar graph on n vertices has treewidth $\mathcal{O}(\sqrt{n})$.
  - **Proof**: If it had $\lceil\sqrt{n+1}\rceil \times \lceil\sqrt{n+1}\rceil$ grid as a minor, it would have $n+1$ vertices.
  - **Note**: Alternative proof via Lipton-Tarjan planar separator theorem.
  - **Note**: There are efficient algorithms to compute such a tree decomposition.

Implications for planar graphs

- **Corollary**: A planar graph on n vertices has treewidth $\mathcal{O}(\sqrt{n})$.
  - **Proof**: If it had $\lceil\sqrt{n+1}\rceil \times \lceil\sqrt{n+1}\rceil$ grid as a minor, it would have $n+1$ vertices.
  - **Note**: Alternative proof via Lipton-Tarjan planar separator theorem.
  - **Note**: There are efficient algorithms to compute such a tree decomposition.
- **Corollary**: Independent Set can be solved in time $2^{\mathcal{O}(\sqrt{n})}$ on planar graphs.

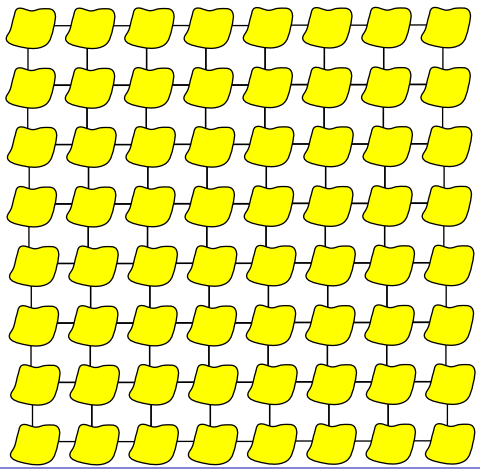
Bidimensionality: basics

- Suppose G has a $3\lceil\sqrt{k}\rceil \times 3\lceil\sqrt{k}\rceil$ grid minor.



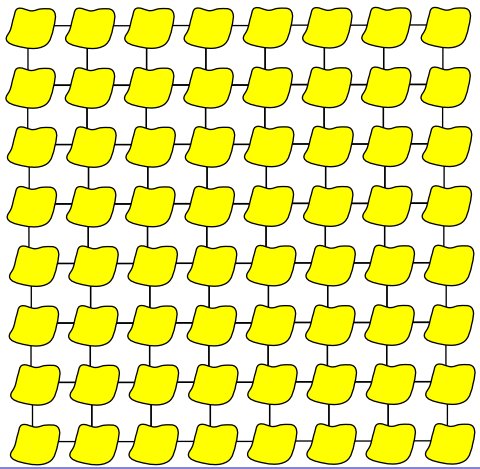
Bidimensionality: basics

- Suppose G has a $3\lceil\sqrt{k}\rceil \times 3\lceil\sqrt{k}\rceil$ grid minor.
- Then picking one vertex from each middle branch set of a 3×3 basic square forms an independent set of size $\geq k$.



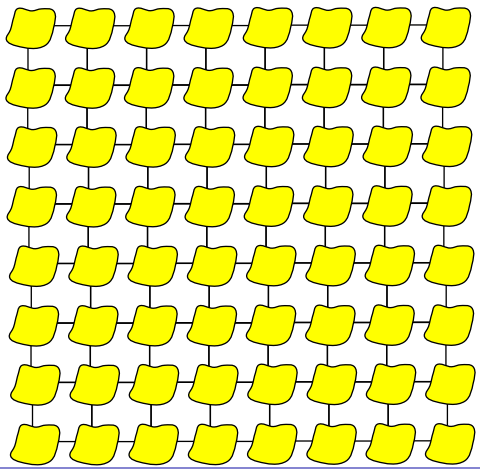
Bidimensionality: basics

- Suppose G has a $3\lceil\sqrt{k}\rceil \times 3\lceil\sqrt{k}\rceil$ grid minor.
- Then picking one vertex from each middle branch set of a 3×3 basic square forms an independent set of size $\geq k$.
- Ergo, $\mathrm{tw}(G) > c\sqrt{k} \Rightarrow \mathrm{YES}$, for some constant c.



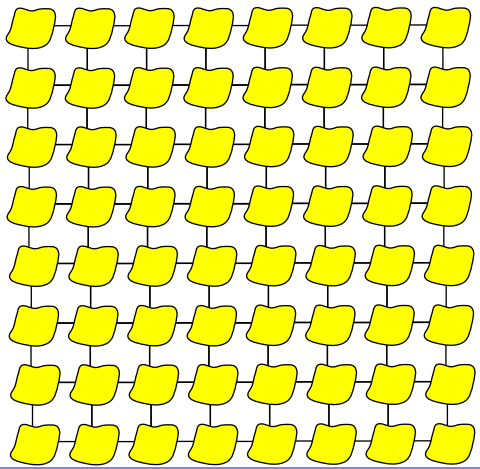
Bidimensionality: basics

- Suppose G has a $3\lceil\sqrt{k}\rceil \times 3\lceil\sqrt{k}\rceil$ grid minor.
- Then picking one vertex from each middle branch set of a 3×3 basic square forms an independent set of size $\geq k$.
- Ergo, $\mathrm{tw}(G) > c\sqrt{k} \Rightarrow \mathrm{YES}$, for some constant c.
- Algorithm:



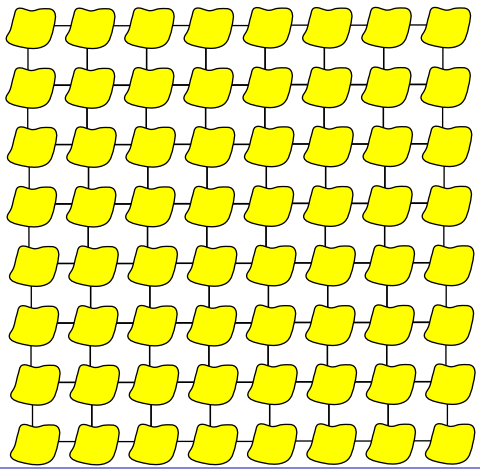
Bidimensionality: basics

- Suppose G has a $3\lceil\sqrt{k}\rceil \times 3\lceil\sqrt{k}\rceil$ grid minor.
- Then picking one vertex from each middle branch set of a 3×3 basic square forms an independent set of size $\geq k$.
- Ergo, $\mathrm{tw}(G) > c\sqrt{k} \Rightarrow \mathrm{YES}$, for some constant c.
- Algorithm:
  - Approximate the treewidth of the graph.



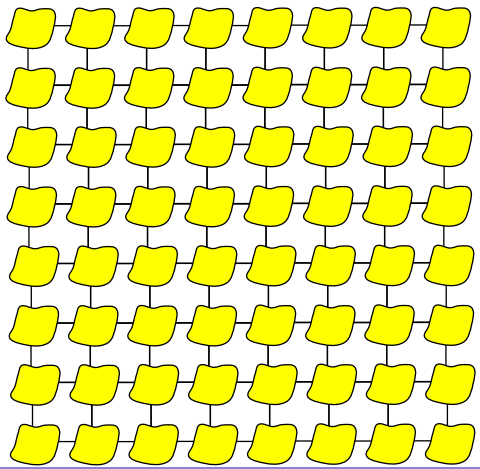
Bidimensionality: basics

- Suppose G has a $3\lceil\sqrt{k}\rceil \times 3\lceil\sqrt{k}\rceil$ grid minor.
- Then picking one vertex from each middle branch set of a 3×3 basic square forms an independent set of size $\geq k$.
- Ergo, $\mathrm{tw}(G) > c\sqrt{k} \Rightarrow \mathrm{YES}$, for some constant c.
- Algorithm:
  - Approximate the treewidth of the graph.
  - If $\mathrm{tw}(G) > c\sqrt{k}$, then answer YES.



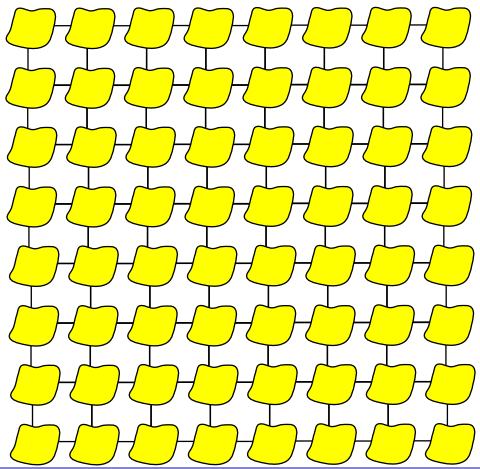
Bidimensionality: basics

- Suppose G has a $3\lceil\sqrt{k}\rceil \times 3\lceil\sqrt{k}\rceil$ grid minor.
- Then picking one vertex from each middle branch set of a 3×3 basic square forms an independent set of size $\geq k$.
- Ergo, $\mathrm{tw}(G) > c\sqrt{k} \Rightarrow \mathrm{YES}$, for some constant c.
- Algorithm:
  - Approximate the treewidth of the graph.
  - If $\mathrm{tw}(G) > c\sqrt{k}$, then answer YES.
  - Otherwise, run a DP on the obtained tree decomposition.



Bidimensionality: basics

- Suppose G has a $3\lceil\sqrt{k}\rceil \times 3\lceil\sqrt{k}\rceil$ grid minor.
- Then picking one vertex from each middle branch set of a 3×3 basic square forms an independent set of size $\geq k$.
- Ergo, $\mathrm{tw}(G) > c\sqrt{k} \Rightarrow \mathrm{YES}$, for some constant c.
- Algorithm:
  - Approximate the treewidth of the graph.
  - If $\mathrm{tw}(G) > c\sqrt{k}$, then answer YES.
  - Otherwise, run a DP on the obtained tree decomposition.
  - Running time: $2^{\mathcal{O}(\sqrt{k})} \cdot n$.



Bidimensionality: general picture

- Works for a number of problems, where there is a correspondence between the size of the largest grid minor and the optimum size of the solution.

Bidimensionality: general picture

- Works for a number of problems, where there is a correspondence between the size of the largest grid minor and the optimum size of the solution.
- Can be applied both to minimization and to maximization problems.

Bidimensionality: general picture

- Works for a number of problems, where there is a correspondence between the size of the largest grid minor and the optimum size of the solution.
- Can be applied both to minimization and to maximization problems.
- **Examples**: Dominating Set, Feedback Vertex Set, Cycle Packing, k-Path.

Bidimensionality: general picture

- Works for a number of problems, where there is a correspondence between the size of the largest grid minor and the optimum size of the solution.
- Can be applied both to minimization and to maximization problems.
- **Examples**: Dominating Set, Feedback Vertex Set, Cycle Packing, k-Path.
- Can be extended to bounded genus and (to some extent) to H-minor-free graphs.

Bidimensionality: general picture

- Works for a number of problems, where there is a correspondence between the size of the largest grid minor and the optimum size of the solution.
- Can be applied both to minimization and to maximization problems.
- **Examples**: Dominating Set, Feedback Vertex Set, Cycle Packing, k-Path.
- Can be extended to bounded genus and (to some extent) to H-minor-free graphs.
- **Note**: Independent Set is W[1]-hard in general graphs, but it can be solved in subexponential parameterized time on planar graphs.

Bidimensionality: general picture

- Works for a number of problems, where there is a correspondence between the size of the largest grid minor and the optimum size of the solution.
- Can be applied both to minimization and to maximization problems.
- **Examples**: Dominating Set, Feedback Vertex Set, Cycle Packing, k-Path.
- Can be extended to bounded genus and (to some extent) to H-minor-free graphs.
- **Note**: Independent Set is W[1]-hard in general graphs, but it can be solved in subexponential parameterized time on planar graphs.
- **Limitations**:

Bidimensionality: general picture

- Works for a number of problems, where there is a correspondence between the size of the largest grid minor and the optimum size of the solution.
- Can be applied both to minimization and to maximization problems.
- **Examples**: Dominating Set, Feedback Vertex Set, Cycle Packing, k-Path.
- Can be extended to bounded genus and (to some extent) to H-minor-free graphs.
- **Note**: Independent Set is W[1]-hard in general graphs, but it can be solved in subexponential parameterized time on planar graphs.
- **Limitations**:
  - Subset problems, like Steiner Tree or Subset TSP.

Bidimensionality: general picture

- Works for a number of problems, where there is a correspondence between the size of the largest grid minor and the optimum size of the solution.
- Can be applied both to minimization and to maximization problems.
- **Examples**: Dominating Set, Feedback Vertex Set, Cycle Packing, k-Path.
- Can be extended to bounded genus and (to some extent) to H-minor-free graphs.
- **Note**: Independent Set is W[1]-hard in general graphs, but it can be solved in subexponential parameterized time on planar graphs.
- **Limitations**:
  - Subset problems, like Steiner Tree or Subset TSP.
  - Directed problems, like Directed k-Path.

Bidimensionality: general picture

- Works for a number of problems, where there is a correspondence between the size of the largest grid minor and the optimum size of the solution.
- Can be applied both to minimization and to maximization problems.
- **Examples**: Dominating Set, Feedback Vertex Set, Cycle Packing, k-Path.
- Can be extended to bounded genus and (to some extent) to H-minor-free graphs.
- **Note**: Independent Set is W[1]-hard in general graphs, but it can be solved in subexponential parameterized time on planar graphs.
- **Limitations**:
  - Subset problems, like Steiner Tree or Subset TSP.
  - Directed problems, like Directed k-Path.
  - Weighted problems.