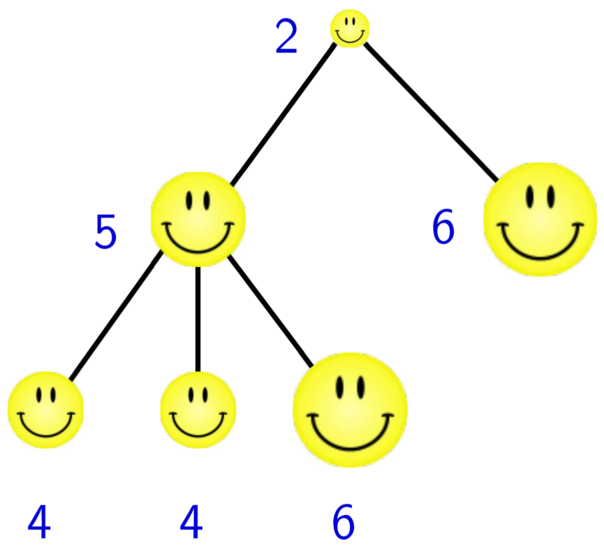
# Treewidth

- Treewidth: a notion of "treelike" graphs.
- Some combinatorial properties.
- Algorithmic results.
  - Algorithms on graphs of bounded treewidth.
  - Applications for other problems.

# The Party Problem

**Party Problem**

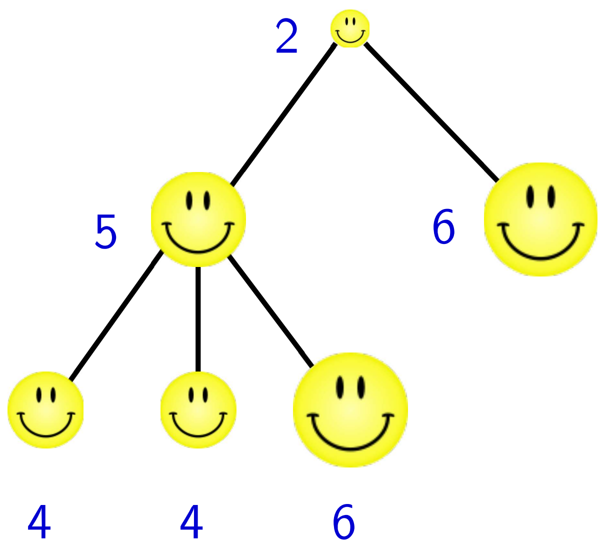
| | |
|---|---|
| **Problem:** | Invite some colleagues for a party. |
| **Maximize:** | The total fun factor of the invited people. |
| **Constraint:** | Everyone should be having fun. |



# The Party Problem

**Party Problem**

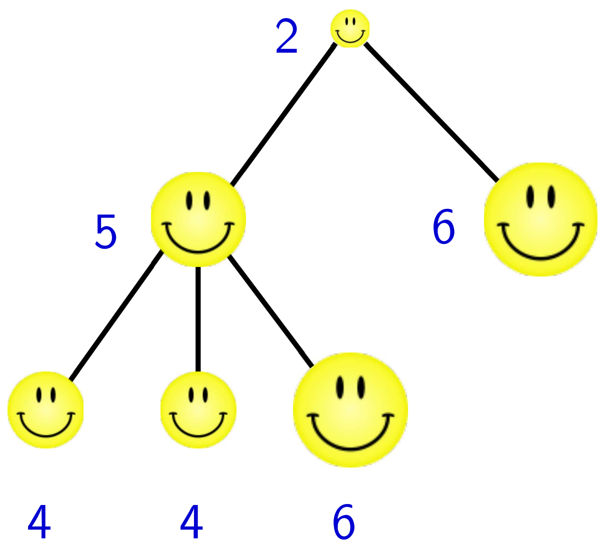
| | |
|---|---|
| **Problem:** | Invite some colleagues for a party. |
| **Maximize:** | The total fun factor of the invited people. |
| **Constraint:** | Everyone should be having fun. |
| | Do not invite a colleague and their direct boss at the same time! |



## The Party Problem

**Party Problem**

| | |
|---|---|
| **Problem:** | Invite some colleagues for a party. |
| **Maximize:** | The total fun factor of the invited people. |
| **Constraint:** | Everyone should be having fun. |
| | Do not invite a colleague and their direct boss at the same time! |

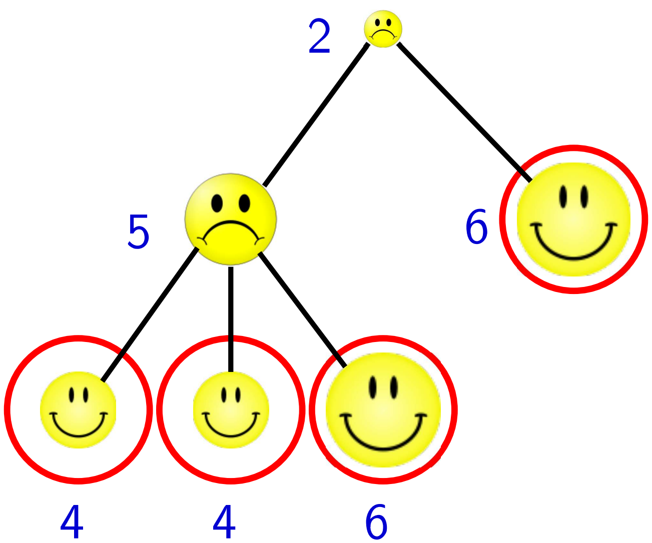



- **Input:** A tree with weights on the vertices.
- **Task:** Find an independent set of maximum weight.

# The Party Problem

PARTY PROBLEM

| | |
|---|---|
| **Problem:** | Invite some colleagues for a party. |
| **Maximize:** | The total fun factor of the invited people. |
| **Constraint:** | Everyone should be having fun.<br>Do not invite a colleague and<br>their direct boss at the same time! |




- **Input:** A tree with weights on the vertices.
- **Task:** Find an independent set of maximum weight.

# Solving the Party Problem

**Dynamic programming paradigm:**
We solve a large number of subproblems that depend on each other. The answer is a single subproblem.

**Subproblems:**

$T_v$: the subtree rooted at $v$.
$A[v]$: max. weight of an independent set in $T_v$
$B[v]$: max. weight of an independent set in $T_v$
that does not contain $v$

**Goal:** determine $A[r]$ for the root $r$.

# Solving the Party Problem

**Subproblems:**

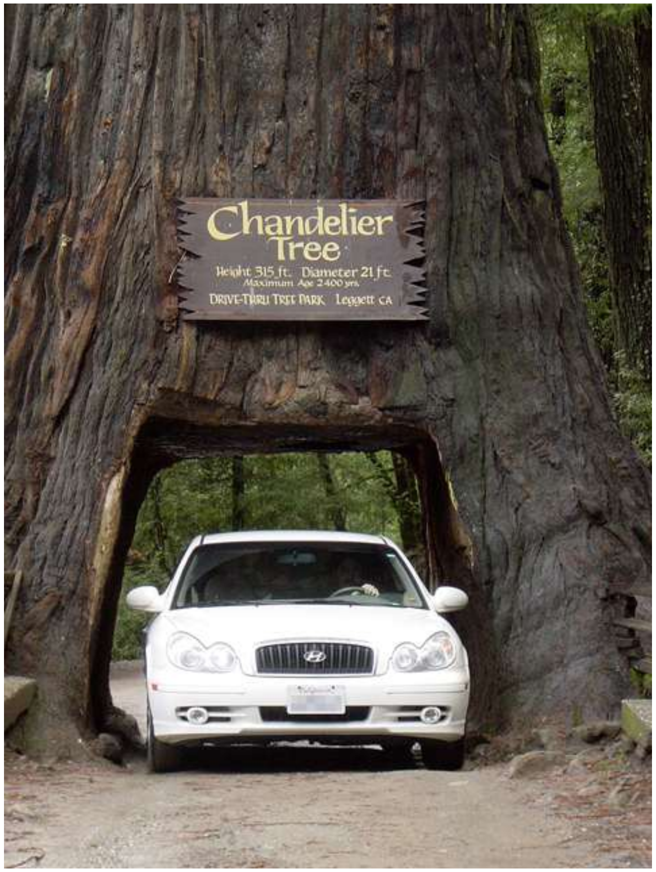
$T_v$: the subtree rooted at $v$.
$A[v]$: max. weight of an independent set in $T_v$
$B[v]$: max. weight of an independent set in $T_v$ that does not contain $v$

**Recurrence:**

Assume $v_1, \ldots, v_k$ are the children of $v$. Use the recurrence relations

$$B[v] = \sum_{i=1}^{k} A[v_i]$$
$$A[v] = \max\{B[v]\ ,\ w(v) + \sum_{i=1}^{k} B[v_i]\}$$

The values $A[v]$ and $B[v]$ can be calculated in a bottom-up order (the leaves are trivial).


Chandelier Tree
Height 315 ft. Diameter 21 ft.
Maximum Age 2400 yrs
DRIVE-THRU TREE PARK Leggett CA

# Treewidth

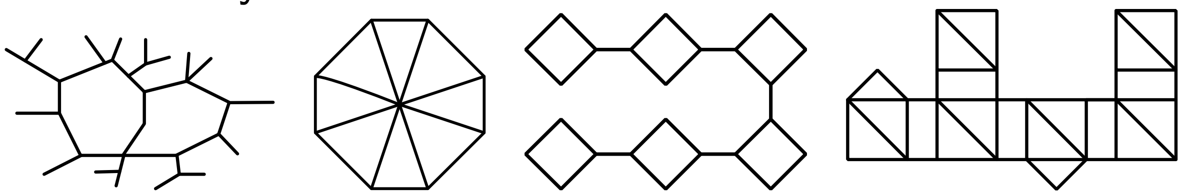
## Generalizing trees

How could we define that a graph is “treelike”?

# Generalizing trees

How could we define that a graph is "treelike"?

1. Number of cycles is bounded.

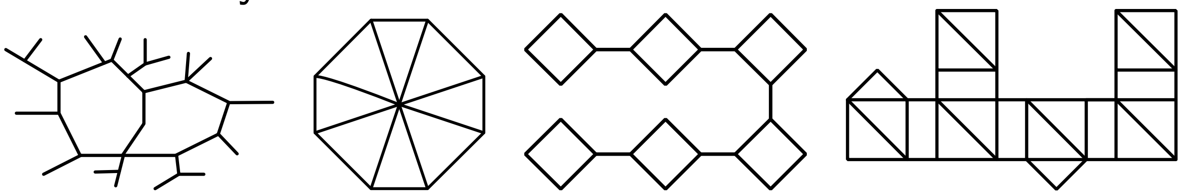


good bad bad bad

# Generalizing trees

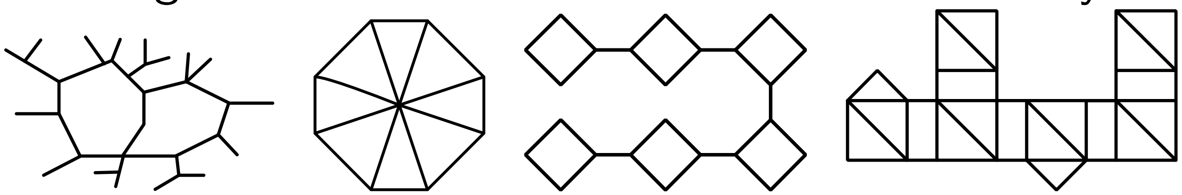
How could we define that a graph is “treelike”?

1. Number of cycles is bounded.



good bad bad bad

2. Removing a bounded number of vertices makes it acyclic.

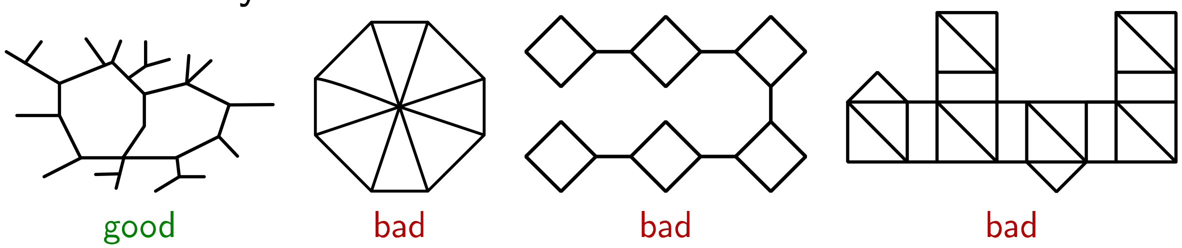


good good bad bad

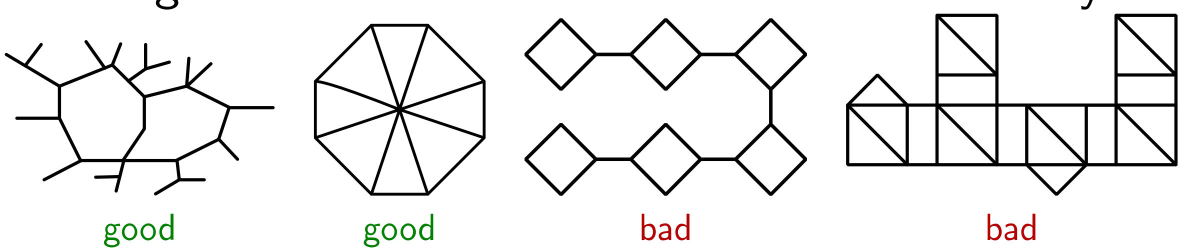
## Generalizing trees

How could we define that a graph is "treelike"?

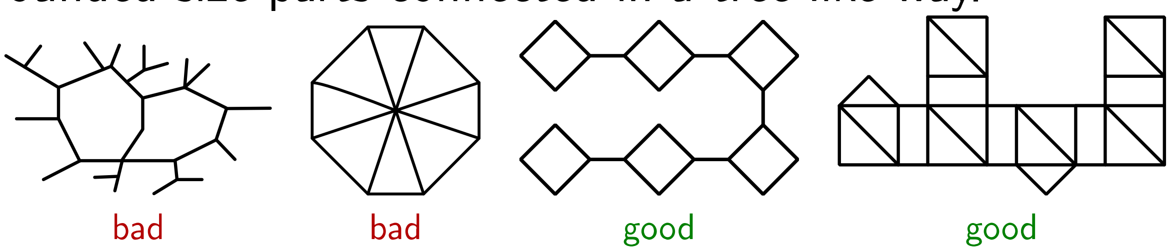
1. Number of cycles is bounded.




2. Removing a bounded number of vertices makes it acyclic.



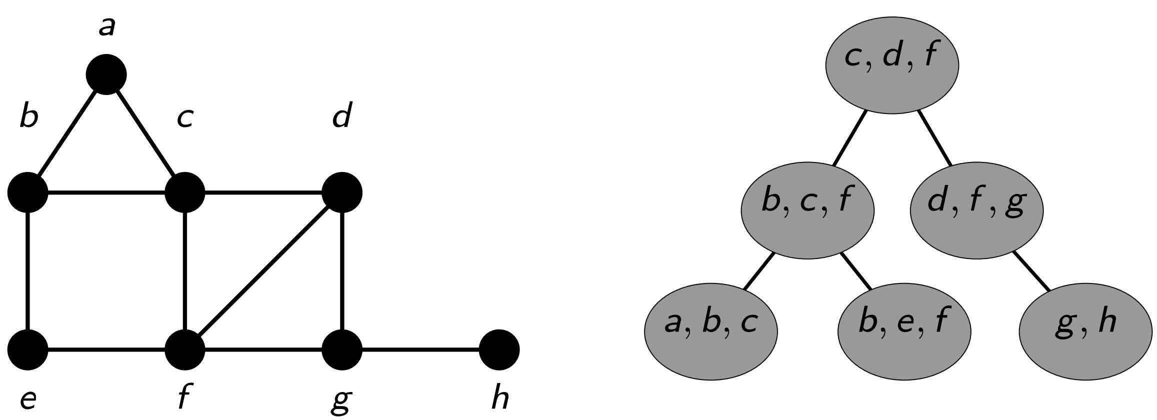

3. Bounded-size parts connected in a tree-like way.



# Treewidth — a measure of "tree-likeness"

**Tree decomposition:** Vertices are arranged in a tree structure satisfying the following properties:

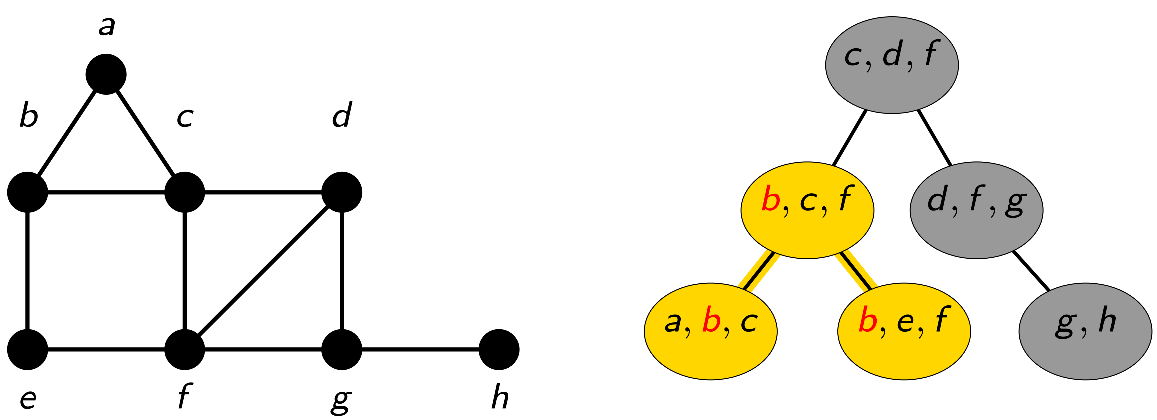
1. If $u$ and $v$ are neighbors, then there is a bag containing both of them.
2. For every $v$, the bags containing $v$ form a connected subtree.



# Treewidth — a measure of "tree-likeness"

**Tree decomposition:** Vertices are arranged in a tree structure satisfying the following properties:

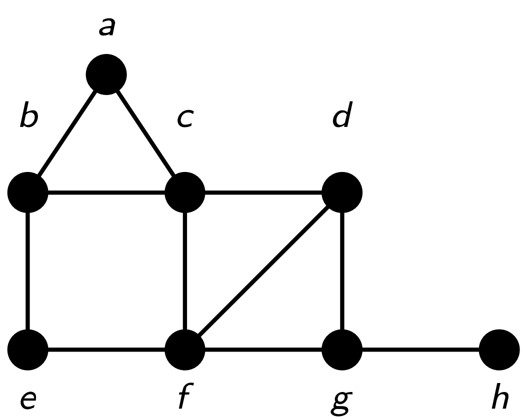
1. If $u$ and $v$ are neighbors, then there is a bag containing both of them.
2. For every $v$, the bags containing $v$ form a connected subtree.

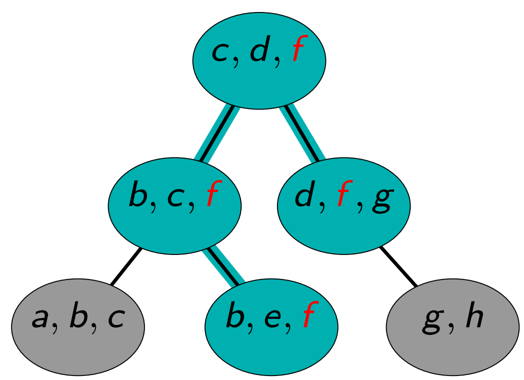


# Treewidth — a measure of "tree-likeness"

**Tree decomposition:** Vertices are arranged in a tree structure satisfying the following properties:

1. If $u$ and $v$ are neighbors, then there is a bag containing both of them.
2. For every $v$, the bags containing $v$ form a connected subtree.



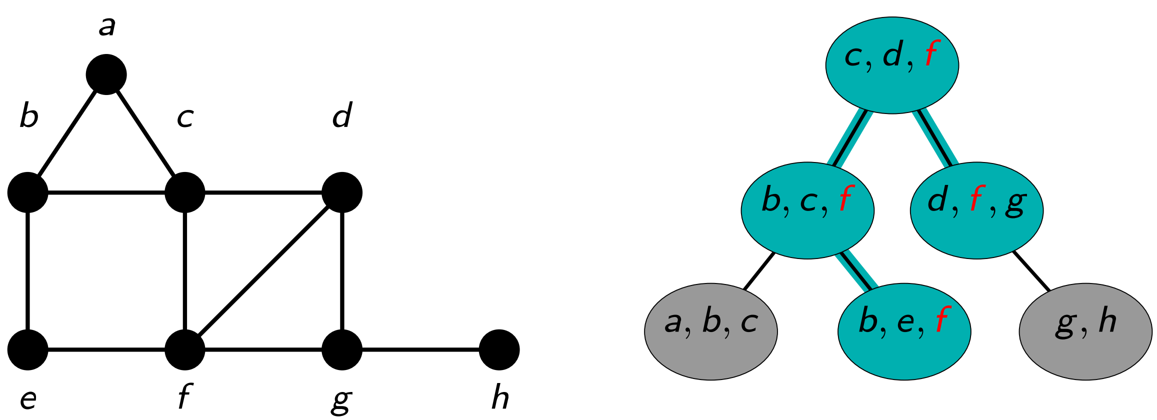
# Treewidth — a measure of "tree-likeness"

**Tree decomposition:** Vertices are arranged in a tree structure satisfying the following properties:

1. If $u$ and $v$ are neighbors, then there is a bag containing both of them.
2. For every $v$, the bags containing $v$ form a connected subtree.

**Width of the decomposition:** largest bag size $-1$.

**treewidth:** width of the best decomposition.



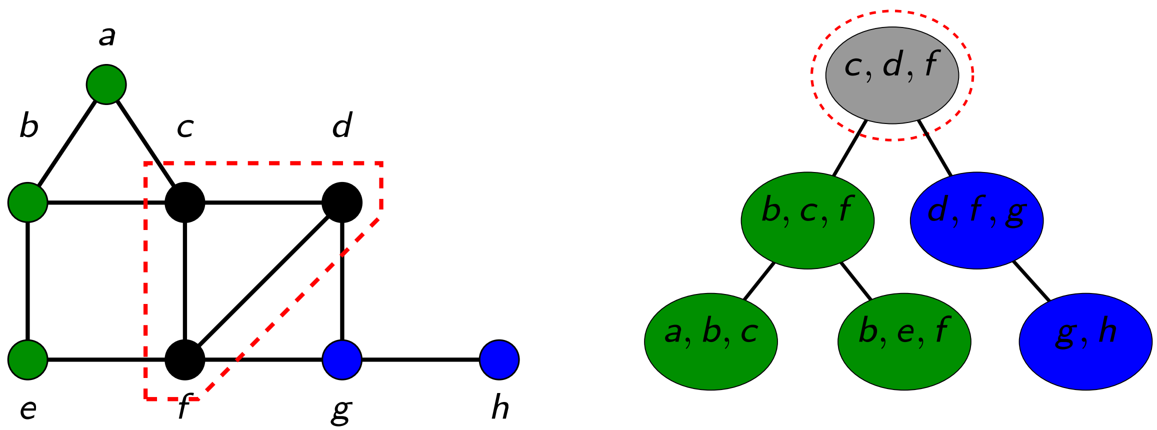
# Treewidth — a measure of "tree-likeness"

**Tree decomposition:** Vertices are arranged in a tree structure satisfying the following properties:

1. If $u$ and $v$ are neighbors, then there is a bag containing both of them.
2. For every $v$, the bags containing $v$ form a connected subtree.

**Width of the decomposition:** largest bag size $-1$.

**treewidth:** width of the best decomposition.




Each bag is a separator.

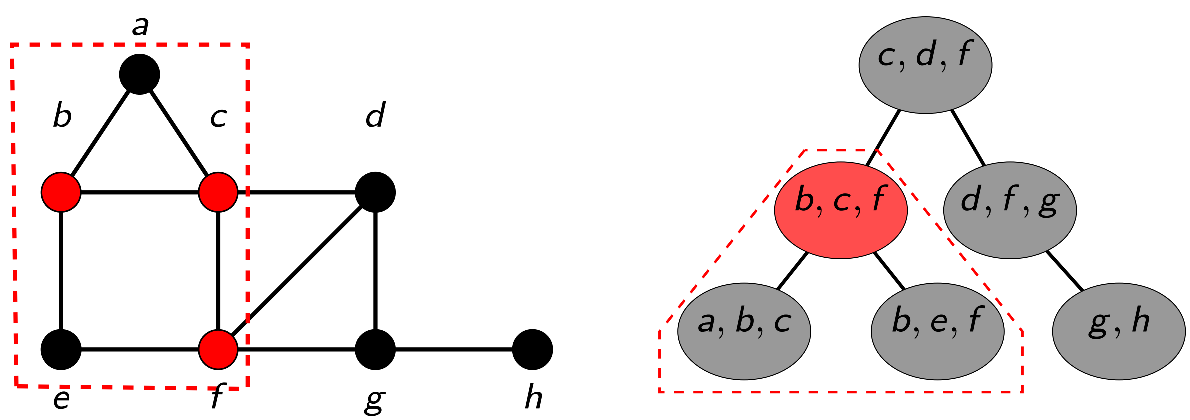
# Treewidth — a measure of "tree-likeness"

**Tree decomposition:** Vertices are arranged in a tree structure satisfying the following properties:

1. If $u$ and $v$ are neighbors, then there is a bag containing both of them.
2. For every $v$, the bags containing $v$ form a connected subtree.

**Width of the decomposition:** largest bag size $-1$.

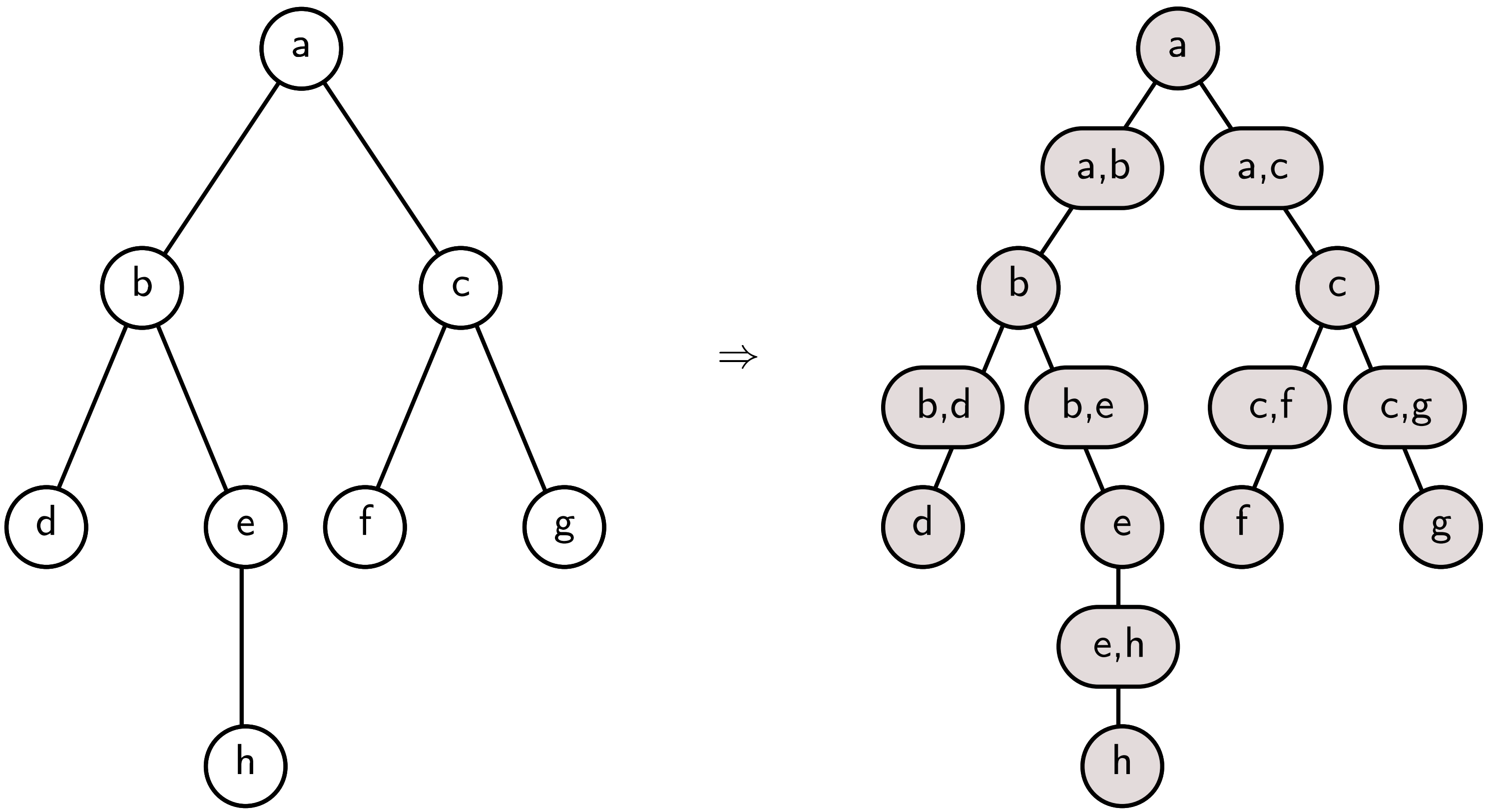
**treewidth:** width of the best decomposition.




A subtree communicates with the outside world only via the root of the subtree.

# Treewidth

**Fact:** treewidth $= 1 \iff$ graph is a forest




**Exercise:** A cycle cannot have a tree decomposition of width 1.

# Treewidth — outline

1. Basic algorithms
2. Combinatorial properties
3. Applications

# Finding tree decompositions

**Hardness:**

**Theorem** [Arnborg, Corneil, Proskurowski 1987]

It is NP-hard to determine the treewidth of a graph (given a graph $G$ and an integer $w$, decide if the treewidth of $G$ is at most $w$).

**Fixed-parameter tractability:**

**Theorem** [Bodlaender 1996]

There is a $2^{O(w^3)} \cdot n$ time algorithm that finds a tree decomposition of width $w$ (if exists).

**Consequence:**
If we want an FPT algorithm parameterized by treewidth $w$ of the input graph, then we can assume that a tree decomposition of width $w$ is available.

# Finding tree decompositions — approximately

Sometimes we can get better dependence on treewidth using approximation.

**FPT approximation:**

**Theorem** [Robertson and Seymour]

There is a $O(3^{3w} \cdot w \cdot n^2)$ time algorithm that finds a tree decomposition of width $4w + 1$, if the treewidth of the graph is at most $w$.

**Polynomial-time approximation:**

**Theorem** [Feige, Hajiaghayi, Lee 2008]

There is a polynomial-time algorithm that finds a tree decomposition of width $O(w\sqrt{\log w})$, if the treewidth of the graph is at most $w$.

# Weighted Max Independent Set and treewidth

**Theorem**

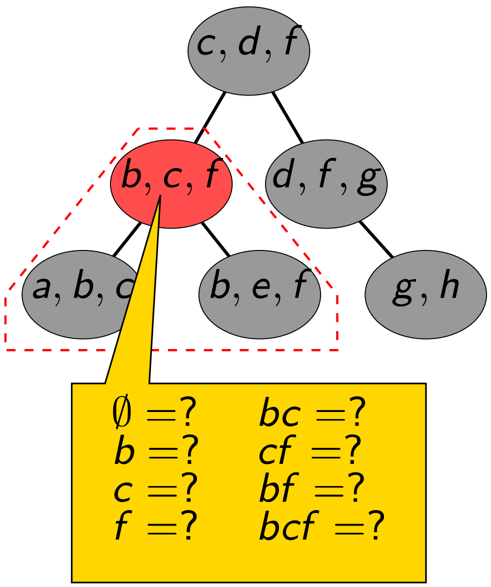
Given a tree decomposition of width $w$, Weighted Max Independent Set can be solved in time $O(2^w \cdot w^{O(1)} \cdot n)$.

$B_x$: vertices appearing in node $x$.
$V_x$: vertices appearing in the subtree rooted at $x$.

Generalizing our solution for trees:

Instead of computing 2 values $A[v]$, $B[v]$ for each **vertex** of the graph, we compute $2^{|B_x|} \le 2^{w+1}$ values for each bag $B_x$.

$M[x, S]$:
the max. weight of an independent set $I \subseteq V_x$ with $I \cap B_x = S$.



# Weighted Max Independent Set and treewidth

**Theorem**

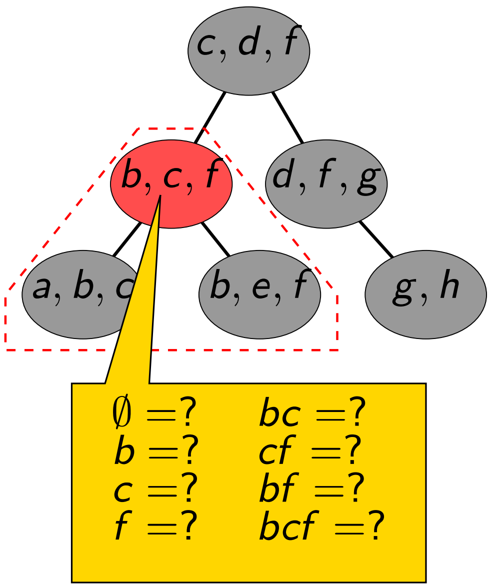
Given a tree decomposition of width $w$, Weighted Max Independent Set can be solved in time $O(2^w \cdot w^{O(1)} \cdot n)$.

$B_x$: vertices appearing in node $x$.
$V_x$: vertices appearing in the subtree rooted at $x$.

Generalizing our solution for trees:

Instead of computing 2 values $A[v]$, $B[v]$ for each **vertex** of the graph, we compute $2^{|B_x|} \le 2^{w+1}$ values for each bag $B_x$.

$M[x, S]$:
the max. weight of an independent set $I \subseteq V_x$ with $I \cap B_x = S$.



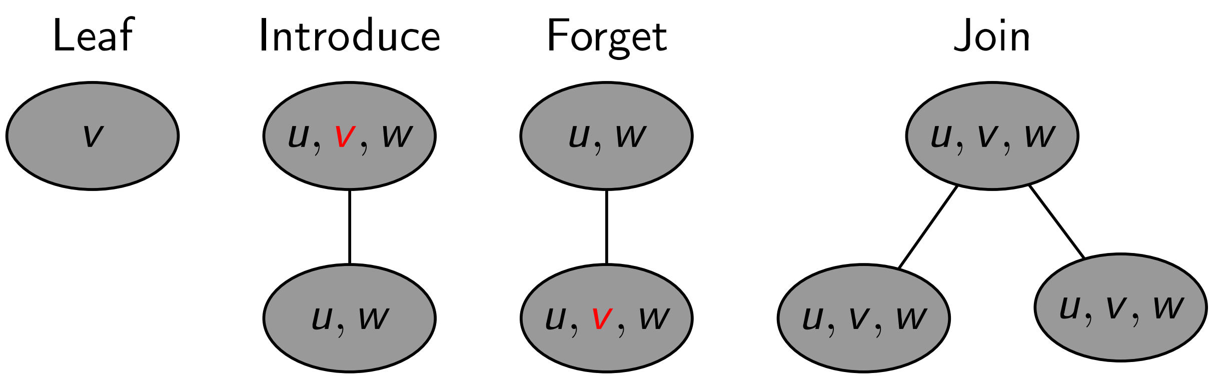

How to determine $M[x, S]$ if all the values are known for the children of $x$?

# Nice tree decompositions

**Definition**

A rooted tree decomposition is **nice** if every node $x$ is one of the following 4 types:

- **Leaf:** no children, $|B_x| = 1$
- **Introduce:** 1 child $y$ with $B_x = B_y \cup \{v\}$ for some vertex $v$
- **Forget:** 1 child $y$ with $B_x = B_y \setminus \{v\}$ for some vertex $v$
- **Join:** 2 children $y_1$, $y_2$ with $B_x = B_{y_1} = B_{y_2}$



## Nice tree decompositions

**Definition**

A rooted tree decomposition is **nice** if every node $x$ is one of the following 4 types:

- **Leaf:** no children, $|B_x| = 1$
- **Introduce:** 1 child $y$ with $B_x = B_y \cup \{v\}$ for some vertex $v$
- **Forget:** 1 child $y$ with $B_x = B_y \setminus \{v\}$ for some vertex $v$
- **Join:** 2 children $y_1$, $y_2$ with $B_x = B_{y_1} = B_{y_2}$

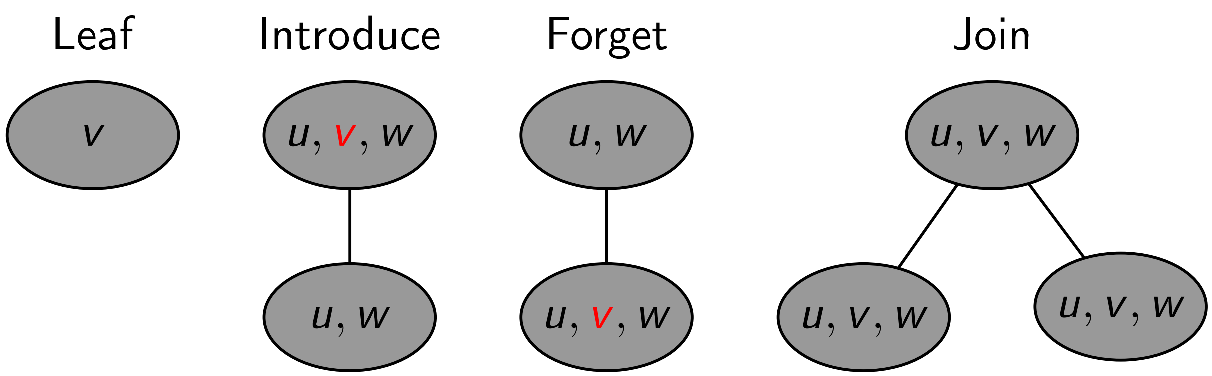
**Theorem**

A tree decomposition of width $w$ and $n$ nodes can be turned into a nice tree decomposition of width $w$ and $O(wn)$ nodes in time $O(w^2 n)$.

# Weighted Max Independent Set
## and nice tree decompositions

- **Leaf:** no children, $|B_x| = 1$
  Trivial!
- **Introduce:** 1 child $y$ with $B_x = B_y \cup \{v\}$ for some vertex $v$

$$m[x,S] = \begin{cases} M[y,S] & \text{if } v \notin S, \\ M[y, S \setminus \{v\}] + w(v) & \text{if } v \in S \text{ but } v \text{ has no neighbor in } S, \\ -\infty & \text{if } S \text{ contains } v \text{ and its neighbor.} \end{cases}$$



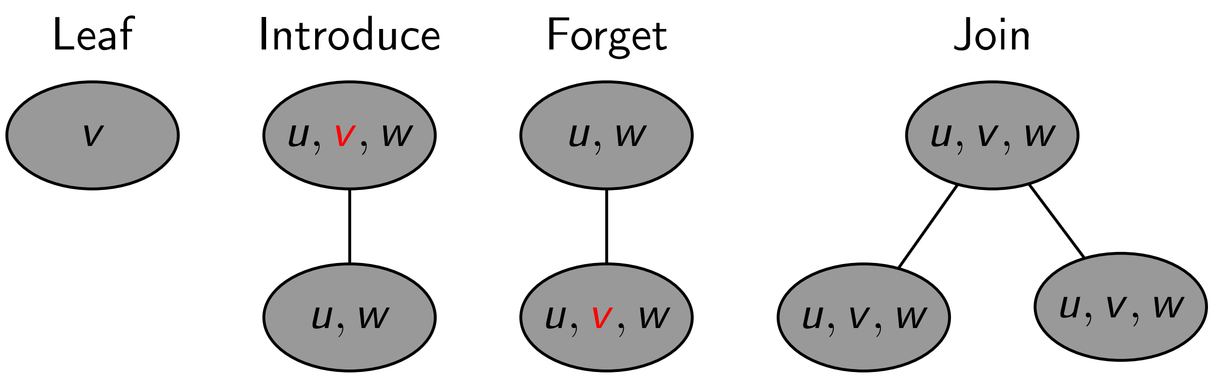
# Weighted Max Independent Set
## and nice tree decompositions

- **Forget:** 1 child $y$ with $B_x = B_y \setminus \{v\}$ for some vertex $v$

$$m[x, S] = \max\{M[y, S], M[y, S \cup \{v\}]\}$$

- **Join:** 2 children $y_1$, $y_2$ with $B_x = B_{y_1} = B_{y_2}$

$$m[x, S] = M[y_1, S] + M[y_2, S] - w(S)$$



## Weighted Max Independent Set
and nice tree decompositions

- **Forget:** 1 child $y$ with $B_x = B_y \setminus \{v\}$ for some vertex $v$

$$m[x,S] = \max\{M[y,S], M[y,S \cup \{v\}]\}$$

- **Join:** 2 children $y_1$, $y_2$ with $B_x = B_{y_1} = B_{y_2}$

$$m[x,S] = M[y_1,S] + M[y_2,S] - w(S)$$

There are at most $2^{w+1} \cdot n$ subproblems $m[x,S]$ and each subproblem can be solved in $w^{O(1)}$ time
(assuming the children are already solved).
$\Downarrow$
Running time is $O(2^w \cdot w^{O(1)} \cdot n)$.

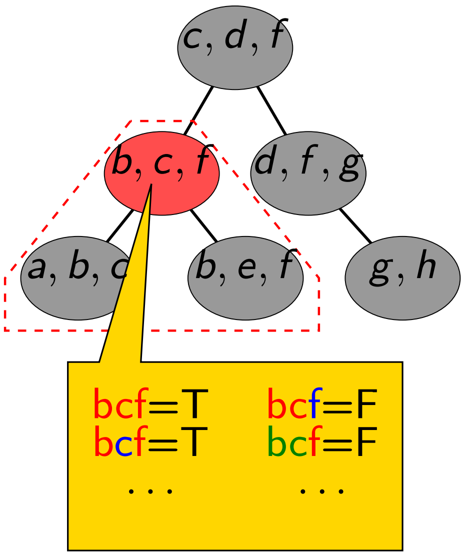
# 3-Coloring and tree decompositions

**Theorem**

Given a tree decomposition of width $w$, 3-Coloring can be solved in $O(3^w \cdot w^{O(1)} \cdot n)$.

$B_x$: vertices appearing in node $x$.
$V_x$: vertices appearing in the subtree rooted at $x$.

For every node $x$ and coloring $c : B_x \to \{1,2,3\}$, we compute the Boolean value $E[x,c]$, which is true if and only if $c$ can be extended to a proper 3-coloring of $V_x$.

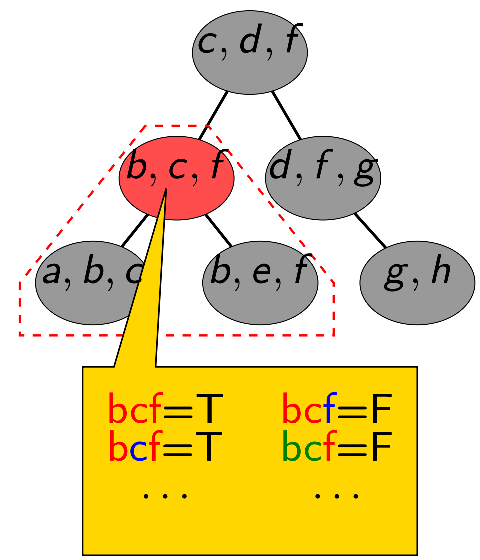


## 3-Coloring and tree decompositions

**Theorem**

Given a tree decomposition of width $w$, 3-Coloring can be solved in $O(3^w \cdot w^{O(1)} \cdot n)$.

$B_x$: vertices appearing in node $x$.
$V_x$: vertices appearing in the subtree rooted at $x$.

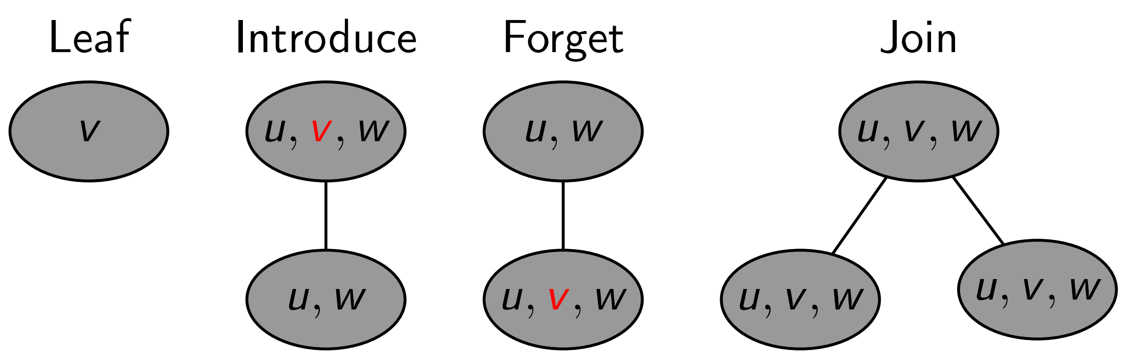



For every node $x$ and coloring $c : B_x \to \{1, 2, 3\}$, we compute the Boolean value $E[x, c]$, which is true if and only if $c$ can be extended to a proper 3-coloring of $V_x$.

How to determine $E[x, c]$ if all the values are known for the children of $x$?

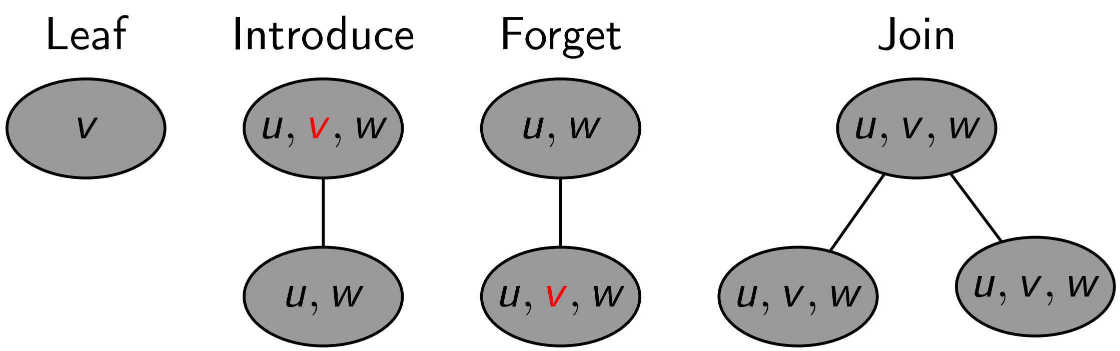
# 3-Coloring and nice tree decompositions

- **Leaf:** no children, $|B_x| = 1$
  Trivial!



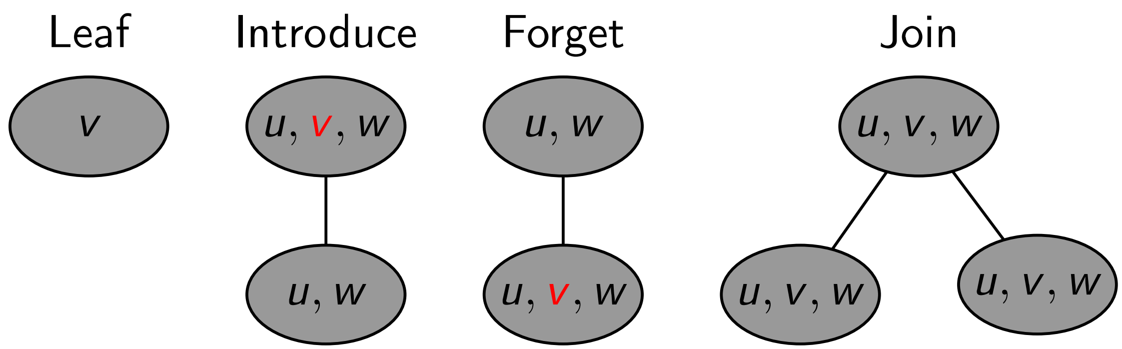
## 3-Coloring and nice tree decompositions

- **Leaf:** no children, $|B_x| = 1$
  Trivial!
- **Introduce:** 1 child $y$ with $B_x = B_y \cup \{v\}$ for some vertex $v$
  If $c(v) \neq c(u)$ for every neighbor $u$ of $v$, then $E[x, c] = E[y, c']$, where $c'$ is $c$ restricted to $B_y$.



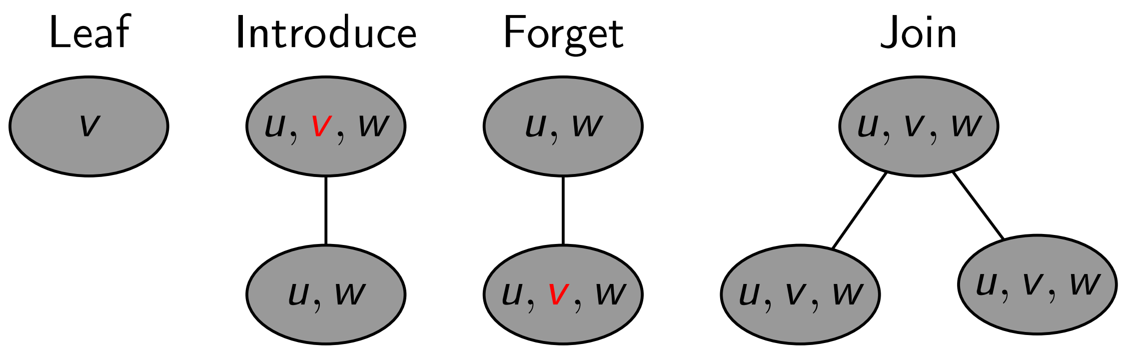
## 3-Coloring and nice tree decompositions

- **Leaf:** no children, $|B_x| = 1$
  Trivial!
- **Introduce:** 1 child $y$ with $B_x = B_y \cup \{v\}$ for some vertex $v$
  If $c(v) \neq c(u)$ for every neighbor $u$ of $v$, then $E[x, c] = E[y, c']$, where $c'$ is $c$ restricted to $B_y$.
- **Forget:** 1 child $y$ with $B_x = B_y \setminus \{v\}$ for some vertex $v$
  $E[x, c]$ is true if $E[y, c']$ is true for one of the 3 extensions of $c$ to $B_y$.



# 3-Coloring and nice tree decompositions

- **Leaf:** no children, $|B_x| = 1$
  Trivial!
- **Introduce:** 1 child $y$ with $B_x = B_y \cup \{v\}$ for some vertex $v$
  If $c(v) \neq c(u)$ for every neighbor $u$ of $v$, then $E[x, c] = E[y, c']$, where $c'$ is $c$ restricted to $B_y$.
- **Forget:** 1 child $y$ with $B_x = B_y \setminus \{v\}$ for some vertex $v$
  $E[x, c]$ is true if $E[y, c']$ is true for one of the 3 extensions of $c$ to $B_y$.
- **Join:** 2 children $y_1$, $y_2$ with $B_x = B_{y_1} = B_{y_2}$
  $E[x, c] = E[y_1, c] \wedge E[y_2, c]$



## 3-Coloring and nice tree decompositions

- **Leaf:** no children, $|B_x| = 1$
  Trivial!
- **Introduce:** 1 child $y$ with $B_x = B_y \cup \{v\}$ for some vertex $v$
  If $c(v) \neq c(u)$ for every neighbor $u$ of $v$, then $E[x, c] = E[y, c']$, where $c'$ is $c$ restricted to $B_y$.
- **Forget:** 1 child $y$ with $B_x = B_y \setminus \{v\}$ for some vertex $v$
  $E[x, c]$ is true if $E[y, c']$ is true for one of the 3 extensions of $c$ to $B_y$.
- **Join:** 2 children $y_1$, $y_2$ with $B_x = B_{y_1} = B_{y_2}$
  $E[x, c] = E[y_1, c] \wedge E[y_2, c]$

There are at most $3^{w+1} \cdot n$ subproblems $E[x, c]$ and each subproblem can be solved in $w^{O(1)}$ time (assuming the children are already solved).

$\Rightarrow$ Running time is $O(3^w \cdot w^{O(1)} \cdot n)$.

$\Rightarrow$ 3-Coloring is FPT parameterized by treewidth.

# Vertex coloring

More generally:

**Theorem**

Given a tree decomposition of width $w$, $c$-Coloring can be solved in time $c^w \cdot n^{O(1)}$.

**Exercise:** Every graph of treewidth at most $w$ can be colored with $w+1$ colors.

**Theorem**

Given a tree decomposition of width $w$, Vertex Coloring can be solved in time $O^*(w^w)$.

$\Rightarrow$ Vertex Coloring is FPT parameterized by treewidth.

## Dominating Set and treewidth

| |
|---|
| Dominating Set: Given $G$ and $k$, find a set $S$ of $k$ vertices such that every vertex of $G$ is in $S$ or has a neighbor in $S$. |

$B_x$: vertices appearing in node $x$.
$V_x$: vertices appearing in the subtree rooted at $x$.

What would be the subproblems for Dominating Set at node $x$?

# Dominating Set and treewidth

Dominating Set: Given $G$ and $k$, find a set $S$ of $k$ vertices such that every vertex of $G$ is in $S$ or has a neighbor in $S$.

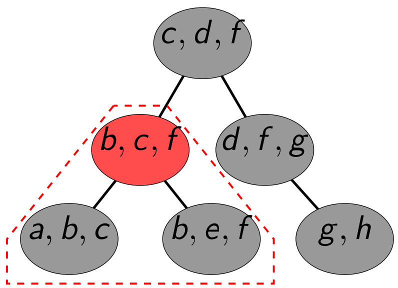
$B_x$: vertices appearing in node $x$.
$V_x$: vertices appearing in the subtree rooted at $x$.

What would be the subproblems for Dominating Set at node $x$?

First try:

$M[x, S]$: size of the smallest set $D \subseteq V_x$ such that

- Every vertex in $V_x$ is dominated by $D$.
- $D \cap B_x = S$.



# Dominating Set and treewidth

Dominating Set: Given $G$ and $k$, find a set $S$ of $k$ vertices such that every vertex of $G$ is in $S$ or has a neighbor in $S$.

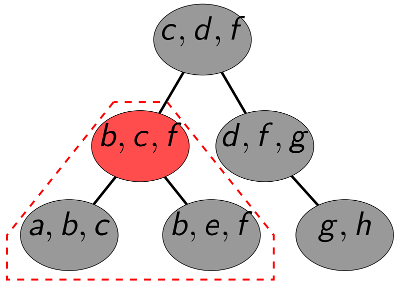
$B_x$: vertices appearing in node $x$.
$V_x$: vertices appearing in the subtree rooted at $x$.

What would be the subproblems for Dominating Set at node $x$?

First try:

$M[x, S]$: size of the smallest set $D \subseteq V_x$ such that

- Every vertex in $V_x$ is dominated by $D$.
- $D \cap B_x = S$.




**Problem:** vertices in $B_x$ can be dominated by vertices outside $V_x$.

# Dominating Set and treewidth

Dominating Set: Given $G$ and $k$, find a set $S$ of $k$ vertices such that every vertex of $G$ is in $S$ or has a neighbor in $S$.

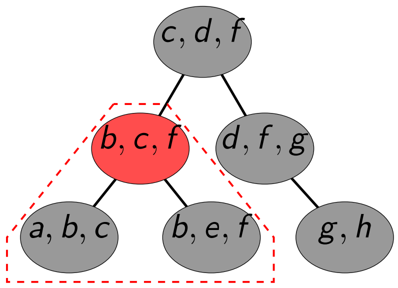
$B_x$: vertices appearing in node $x$.
$V_x$: vertices appearing in the subtree rooted at $x$.

What would be the subproblems for Dominating Set at node $x$?

Second try:

$M[x, S_1, S_2]$: size of the smallest set $D \subseteq V_x$ such that

- Every vertex in $V_x \setminus B_x$ is dominated by $D$.
- $D \cap B_x = S_1$.
- $D$ dominates every vertex of $S_2$.




$\Rightarrow 3^{w+1}$ subproblems at each node $x$.

# Dominating Set and treewidth

$M[x, S_1, S_2]$: size of the smallest set $D \subseteq V_x$ such that

- Every vertex in $V_x \setminus B_x$ is dominated by $D$.
- $S \cap B_x = S_1$.
- $D$ dominates every vertex of $S_2$.

How can we solve subproblem $M[x, S_1, S_2]$ when $x$ is a join node?

# Dominating Set and treewidth

$M[x, S_1, S_2]$: size of the smallest set $D \subseteq V_x$ such that

- Every vertex in $V_x \setminus B_x$ is dominated by $D$.
- $S \cap B_x = S_1$.
- $D$ dominates every vertex of $S_2$.

How can we solve subproblem $M[x, S_1, S_2]$ when $x$ is a join node?

- Consider $3^{|S_2|}$ cases: each vertex of $S_2$ is dominated from the left child, right child, or both $\Rightarrow O(9^w \cdot n)$ time.

# Dominating Set and treewidth

$M[x, S_1, S_2]$: size of the smallest set $D \subseteq V_x$ such that

- Every vertex in $V_x \setminus B_x$ is dominated by $D$.
- $S \cap B_x = S_1$.
- $D$ dominates every vertex of $S_2$.

How can we solve subproblem $M[x, S_1, S_2]$ when $x$ is a join node?

- Consider $3^{|S_2|}$ cases: each vertex of $S_2$ is dominated from the left child, right child, or both $\Rightarrow O(9^w \cdot n)$ time.
- Consider $5^{|B_x|}$ subproblems: in the solution/not dominated/dominated from left/dominated from right/dominated from both $\Rightarrow O(5^w \cdot n)$ time.

# Dominating Set and treewidth

$M[x, S_1, S_2]$: size of the smallest set $D \subseteq V_x$ such that

- Every vertex in $V_x \setminus B_x$ is dominated by $D$.
- $S \cap B_x = S_1$.
- $D$ dominates every vertex of $S_2$.

How can we solve subproblem $M[x, S_1, S_2]$ when $x$ is a join node?

- Consider $3^{|S_2|}$ cases: each vertex of $S_2$ is dominated from the left child, right child, or both $\Rightarrow O(9^w \cdot n)$ time.
- Consider $5^{|B_x|}$ subproblems: in the solution/not dominated/dominated from left/dominated from right/dominated from both $\Rightarrow O(5^w \cdot n)$ time.
- Renaming "not dominated" to "don't care" can improve to $O(4^w \cdot n)$ time.

# DOMINATING SET and treewidth

$M[x, S_1, S_2]$: size of the smallest set $D \subseteq V_x$ such that

- Every vertex in $V_x \setminus B_x$ is dominated by $D$.
- $S \cap B_x = S_1$.
- $D$ dominates every vertex of $S_2$.

How can we solve subproblem $M[x, S_1, S_2]$ when $x$ is a join node?

- Consider $3^{|S_2|}$ cases: each vertex of $S_2$ is dominated from the left child, right child, or both $\Rightarrow O(9^w \cdot n)$ time.
- Consider $5^{|B_x|}$ subproblems: in the solution/not dominated/dominated from left/dominated from right/dominated from both $\Rightarrow O(5^w \cdot n)$ time.
- Renaming "not dominated" to "don't care" can improve to $O(4^w \cdot n)$ time.
- Fast subset convolution: $O(3^w \cdot n)$ time.

# Monadic Second Order Logic

**Extended Monadic Second Order Logic (EMSO)**

A logical language on graphs consisting of the following:

- Logical connectives $\wedge$, $\vee$, $\rightarrow$, $\neg$, $=$, $\neq$
- quantifiers $\forall$, $\exists$ over vertex/edge variables
- predicate $\mathrm{adj}(u, v)$: vertices $u$ and $v$ are adjacent
- predicate $\mathrm{inc}(e, v)$: edge $e$ is incident to vertex $v$
- quantifiers $\forall$, $\exists$ over vertex/edge set variables
- $\in$, $\subseteq$ for vertex/edge sets

**Example:**
The formula

$$\exists C \subseteq V \forall v \in C \; \exists u_1, u_2 \in C (u_1 \neq u_2 \wedge \mathrm{adj}(u_1, v) \wedge \mathrm{adj}(u_2, v))$$

is true on graph $G$ if and only if . . .

# Monadic Second Order Logic

**Extended Monadic Second Order Logic (EMSO)**

A logical language on graphs consisting of the following:

- Logical connectives $\wedge$, $\vee$, $\rightarrow$, $\neg$, $=$, $\neq$
- quantifiers $\forall$, $\exists$ over vertex/edge variables
- predicate $\text{adj}(u, v)$: vertices $u$ and $v$ are adjacent
- predicate $\text{inc}(e, v)$: edge $e$ is incident to vertex $v$
- quantifiers $\forall$, $\exists$ over vertex/edge set variables
- $\in$, $\subseteq$ for vertex/edge sets

**Example:**
The formula

$$\exists C \subseteq V \forall v \in C \; \exists u_1, u_2 \in C (u_1 \neq u_2 \wedge \text{adj}(u_1, v) \wedge \text{adj}(u_2, v))$$

is true on graph $G$ if and only if $G$ has a cycle.

# Courcelle's Theorem

**Courcelle's Theorem**

If a graph property can be expressed in EMSO, then for every fixed $w \geq 1$, there is a linear-time algorithm for testing this property on graphs having treewidth at most $w$.

# Courcelle's Theorem

**Courcelle's Theorem**

There exists an algorithm that, given a width-$w$ tree decomposition of an $n$-vertex graph $G$ and an EMSO formula $\phi$, decides whether $G$ satisfies $\phi$ in time $f(w, |\phi|) \cdot n$.

# Courcelle's Theorem

**Courcelle's Theorem**

There exists an algorithm that, given a width-$w$ tree decomposition of an $n$-vertex graph $G$ and an EMSO formula $\phi$, decides whether $G$ satisfies $\phi$ in time $f(w, |\phi|) \cdot n$.

If we can express a property in EMSO, then we immediately get that testing this property is FPT parameterized by the treewidth $w$ of the input graph.

**Note:** The constant depending on $w$ can be very large (double, triple exponential etc.), therefore a direct dynamic programming algorithm can be more efficient.

# Using Courcelle's Theorem

Can we express 3-Coloring and Hamiltonian Cycle in EMSO?

## Using Courcelle's Theorem

Can we express 3-Coloring and Hamiltonian Cycle in EMSO?

**3-Coloring**

$\exists C_1, C_2, C_3 \subseteq V \left(\forall v \in V\, (v \in C_1 \vee v \in C_2 \vee v \in C_3)\right) \wedge \left(\forall u, v \in V\, \mathrm{adj}(u, v) \to (\neg(u \in C_1 \wedge v \in C_1) \wedge \neg(u \in C_2 \wedge v \in C_2) \wedge \neg(u \in C_3 \wedge v \in C_3))\right)$

# Using Courcelle's Theorem

Can we express 3-Coloring and Hamiltonian Cycle in EMSO?

**3-Coloring**

$\exists C_1, C_2, C_3 \subseteq V \big(\forall v \in V\, (v \in C_1 \vee v \in C_2 \vee v \in C_3)\big) \wedge \big(\forall u, v \in V\, \mathrm{adj}(u,v) \rightarrow (\neg(u \in C_1 \wedge v \in C_1) \wedge \neg(u \in C_2 \wedge v \in C_2) \wedge \neg(u \in C_3 \wedge v \in C_3)))\big)$

**Hamiltonian Cycle**

$\exists H \subseteq E\big(\mathrm{spanning}(H) \wedge (\forall v \in V\, \mathrm{degree2}(H,v))\big)$

$\mathrm{degree2}(H,v) := \exists e_1, e_2 \in H\big((e_1 \neq e_2) \wedge \mathrm{inc}(e_1,v) \wedge \mathrm{inc}(e_2,v) \wedge (\forall e_3 \in H \mathrm{inc}(e_3,v) \rightarrow (e_1 = e_3 \vee e_2 = e_3))\big)$

$\mathrm{spanning}(H) := \forall Z \subseteq V\big(((\exists v \in V : v \in Z) \wedge (\exists v \in V : v \notin Z)) \rightarrow (\exists e \in H \exists x \in V \exists y \in V : (x \in Z) \wedge (y \notin Z) \wedge \mathrm{inc}(e,x) \wedge \mathrm{inc}(e,y))\big)$

# Subgraph Isomorphism

**Subgraph Isomorphism**

Input: graphs $H$ and $G$

Find: a subgraph of $G$ isomorphic to $H$.

# Subgraph Isomorphism

**Subgraph Isomorphism**

Input: graphs $H$ and $G$
Find: a subgraph of $G$ isomorphic to $H$.

For each $H$, we can construct a formula $\phi_H$ that expresses "$G$ has a subgraph isomorphic to $H$" (similarly to the $k$-cycle on the previous slide).

$\Rightarrow$ By Courcelle's Theorem, Subgraph Isomorphism can be solved in time $f(H, w) \cdot n$ if $G$ has treewidth at most $w$.

# Subgraph Isomorphism

**Subgraph Isomorphism**

Input: graphs $H$ and $G$
Find: a subgraph of $G$ isomorphic to $H$.

Since there is only a finite number of simple graphs on $k$ vertices, Subgraph Isomorphism can be solved in time $f(k, w) \cdot n$ if $H$ has $k$ vertices and $G$ has treewidth at most $w$.

**Theorem**

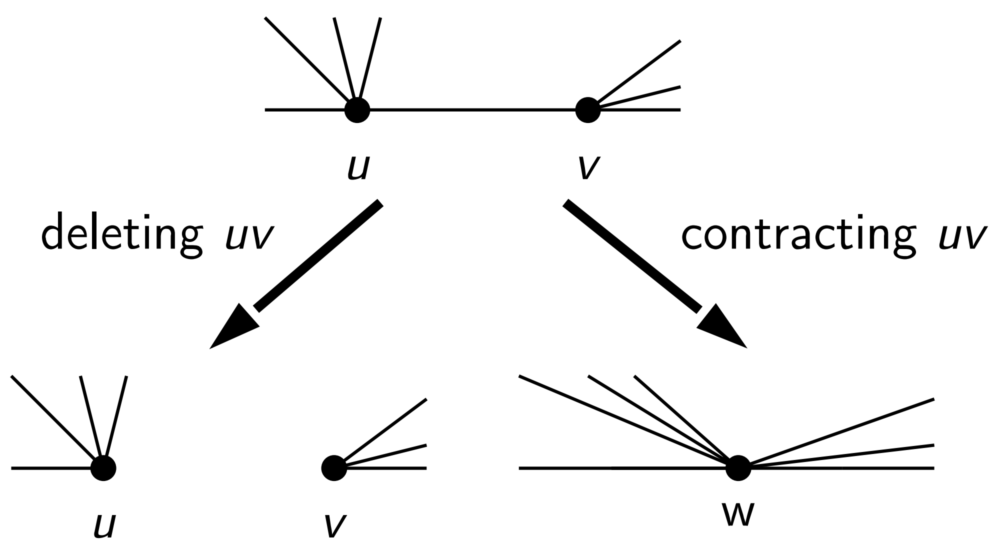
Subgraph Isomorphism is FPT parameterized by combined parameter $k := |V(H)|$ and the treewidth $w$ of $G$.

# Minor

An operation similar to taking subgraphs:

**Definition**

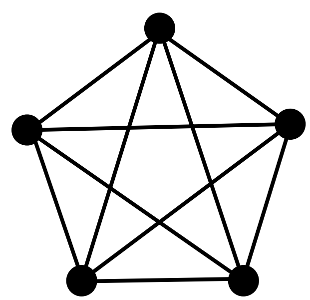
Graph $H$ is a **minor** of $G$ ($H \leq G$) if $H$ can be obtained from $G$ by deleting edges, deleting vertices, and contracting edges.



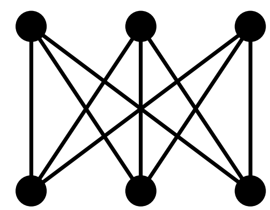
## A classical result

**Theorem** [Kuratowski 1930]

A graph $G$ is planar if and only if $G$ does not contain a subdivision of $K_5$ or $K_{3,3}$.



$K_5$



$K_{3,3}$

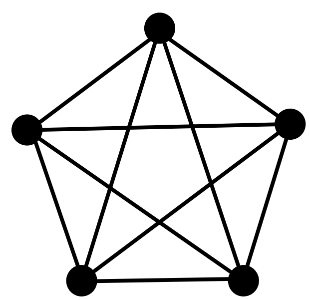
## A classical result

**Theorem** [Kuratowski 1930]

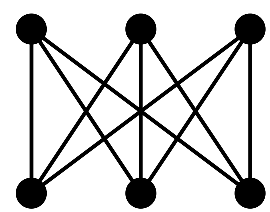
A graph $G$ is planar if and only if $G$ does not contain a subdivision of $K_5$ or $K_{3,3}$.

**Theorem** [Wagner 1937]

A graph $G$ is planar if and only if $G$ does not contain $K_5$ or $K_{3,3}$ as minor.



$K_5$



$K_{3,3}$

# Graph Minors Theory

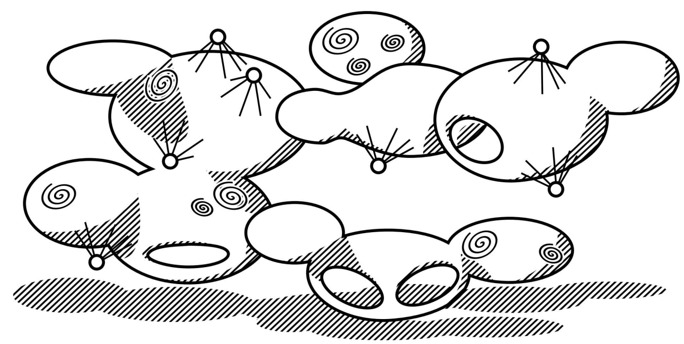
Neil Robertson

Paul Seymour

Theory of graph minors developed in the monumental series

Graph Minors I–XXIII.
J. Combin. Theory, Ser. B
1983–2012

- Structure theory of graphs excluding minors (and much more).
- Galactic combinatorial bounds and running times.
- Important early influence for parameterized algorithms.



# Properties of treewidth

**Fact:** Treewidth does not increase if we delete edges, delete vertices, or contract edges.

$\Rightarrow$ If $F$ is a **minor** of $G$, then the treewidth of $F$ is at most the treewidth of $G$.

# Properties of treewidth

**Fact:** Treewidth does not increase if we delete edges, delete vertices, or contract edges.

$\Rightarrow$ If $F$ is a **minor** of $G$, then the treewidth of $F$ is at most the treewidth of $G$.

**Fact:** For every clique $K$, there is a bag $B$ with $K \subseteq B$.

**Fact:** The treewidth of the $k$-clique is $k-1$.

# Properties of treewidth

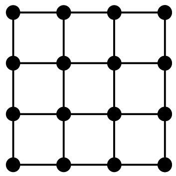
**Fact:** Treewidth does not increase if we delete edges, delete vertices, or contract edges.

$\Rightarrow$ If $F$ is a **minor** of $G$, then the treewidth of $F$ is at most the treewidth of $G$.

**Fact:** For every clique $K$, there is a bag $B$ with $K \subseteq B$.

**Fact:** The treewidth of the $k$-clique is $k-1$.

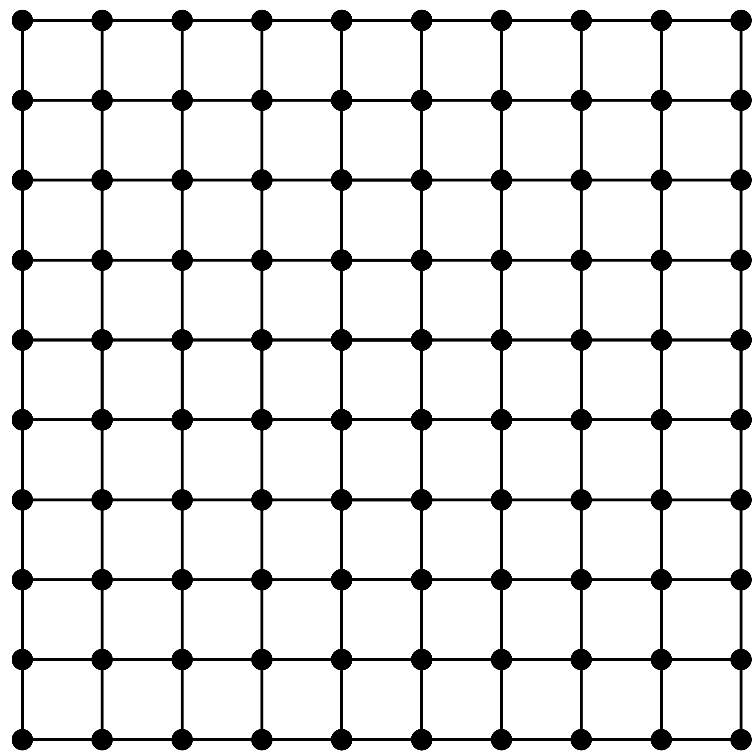
**Fact:** For every $k \geq 2$, the treewidth of the $k \times k$ grid is exactly $k$.



# Excluded Grid Theorem

**Excluded Grid Theorem**

If the treewidth of $G$ is $\Omega(k^9 \log k)$, then $G$ has a $k \times k$ grid minor.



# Excluded Grid Theorem

**Excluded Grid Theorem**

If the treewidth of $G$ is $\Omega(k^9 \log k)$, then $G$ has a $k \times k$ grid minor.

A large grid minor is a "witness" that treewidth is large, but the relation is approximate:

No $k \times k$ grid minor $\Longrightarrow$ tree decomposition of width $O(k^9 \log k)$

tree decomposition of width $< k$ $\Longrightarrow$ no $k \times k$ grid minor

## Excluded Grid Theorem

**Excluded Grid Theorem**

If the treewidth of $G$ is $\Omega(k^9 \log k)$, then $G$ has a $k \times k$ grid minor.

**Observation:** Every planar graph is the minor of a sufficiently large grid.

**Consequence**

If $H$ is planar, then every $H$-minor free graph has treewidth at most $f(H)$.

# Planar Excluded Grid Theorem

For planar graphs, we get linear instead of exponential dependence:

**Theorem**

Every **planar graph** with treewidth at least $5k$ has a $k \times k$ grid minor.

| | | |
|---|---|---|
| No $k \times k$ grid minor | $\Longrightarrow$ | tree decomposition of width $O(k)$ |
| tree decomposition of width $< k$ | $\Longrightarrow$ | no $k \times k$ grid minor |

## Planar Excluded Grid Theorem

For planar graphs, we get linear instead of exponential dependence:

**Theorem**

Every **planar graph** with treewidth at least $5k$ has a $k \times k$ grid minor.

**Theorem**

An $n$-vertex planar graph has treewidth $O(\sqrt{n})$.

# Vertex Cover

**Theorem**

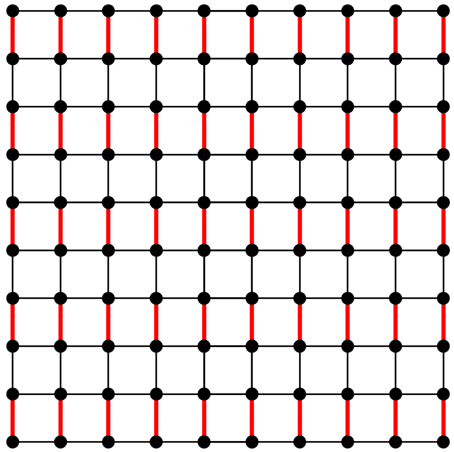
Vertex Cover can be solved in time $2^{O(\sqrt{k})} \cdot n^{O(1)}$ in planar graphs.

We need two facts:

- Removing an edge, removing a vertex, contracting an edge cannot increase the vertex cover number.
- Vertex Cover can be solved in time $2^{w} \cdot n^{O(1)}$ if a tree decomposition of width $w$ is given.

# Vertex Cover

**Observation:** If the treewidth of a planar graph $G$ is at least $5\sqrt{2k}$
$\Rightarrow$ It has a $\sqrt{2k} \times \sqrt{2k}$ grid minor (Planar Excluded Grid Theorem)
$\Rightarrow$ The grid has a matching of size $k$
$\Rightarrow$ Vertex cover size is at least $k$ in the grid.
$\Rightarrow$ Vertex cover size is at least $k$ in $G$.

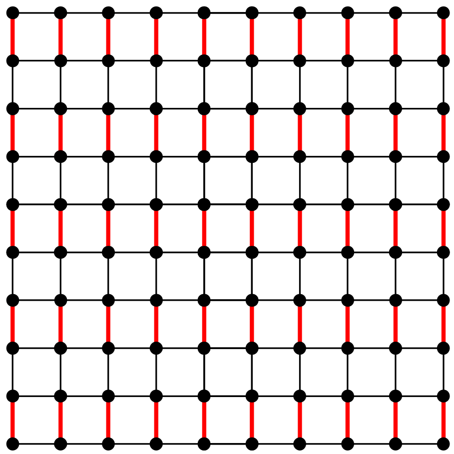


# Vertex Cover

**Observation:** If the treewidth of a planar graph $G$ is at least $5\sqrt{2k}$
$\Rightarrow$ It has a $\sqrt{2k} \times \sqrt{2k}$ grid minor (Planar Excluded Grid Theorem)
$\Rightarrow$ The grid has a matching of size $k$
$\Rightarrow$ Vertex cover size is at least $k$ in the grid.
$\Rightarrow$ Vertex cover size is at least $k$ in $G$.

We use this observation to solve Vertex Cover on planar graphs:

- If treewidth is at least $5\sqrt{2k}$: we answer "vertex cover is $\geq k$."
- If treewidth is less than $5\sqrt{2k}$, then we can solve the problem in time $2^{O(5\sqrt{2k})} \cdot n^{O(1)} = 2^{O(\sqrt{k})} \cdot n^{O(1)}$.

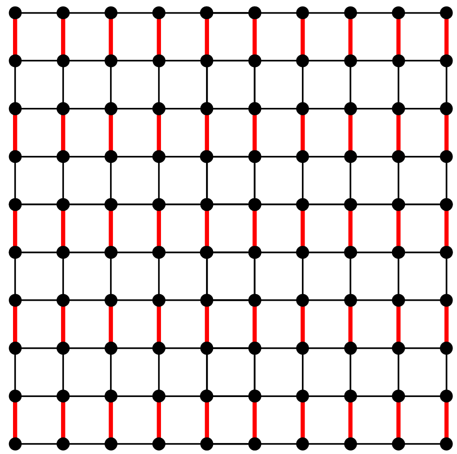


# Vertex Cover

**Observation:** If the treewidth of a planar graph $G$ is at least $5\sqrt{2k}$
$\Rightarrow$ It has a $\sqrt{2k} \times \sqrt{2k}$ grid minor (Planar Excluded Grid Theorem)
$\Rightarrow$ The grid has a matching of size $k$
$\Rightarrow$ Vertex cover size is at least $k$ in the grid.
$\Rightarrow$ Vertex cover size is at least $k$ in $G$.

We use this observation to solve Vertex Cover on planar graphs:

- Set $w := 5\sqrt{2k}$.
- Find a 4-approximate tree decomposition.
  - If treewidth is at least $w$: we answer "vertex cover is $\geq k$."
  - If we get a tree decomposition of width $4w$, then we can solve the problem in time $2^{O(w)} \cdot n^{O(1)} = 2^{O(\sqrt{k})} \cdot n^{O(1)}$.



# Bidimensionality

A powerful framework for efficient algorithms on planar graphs.

**Setup:**

- Let $x(G)$ be some graph invariant (i.e., an integer associated with each graph).
- Given $G$ and $k$, we want to decide if $x(G) \le k$ (or $x(G) \ge k$).
- Typical examples:
  - Maximum independent set size.
  - Minimum vertex cover size.
  - Length of the longest path.
  - Minimum dominating set size.
  - Minimum feedback vertex set size.

**Bidimensionality**

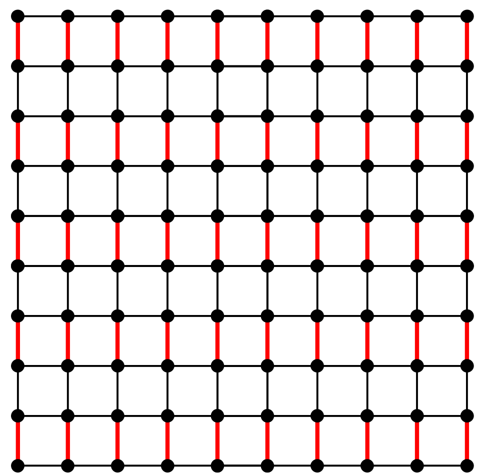
For many natural invariants, we can do this in time $2^{O(\sqrt{k})} \cdot n^{O(1)}$ on planar graphs.

# Bidimensionality

## Definition

A graph invariant $x(G)$ is **minor-bidimensional** if

- $x(G') \leq x(G)$ for every minor $G'$ of $G$, and
- If $G_k$ is the $k \times k$ grid, then $x(G_k) \geq ck^2$ (for some constant $c > 0$).



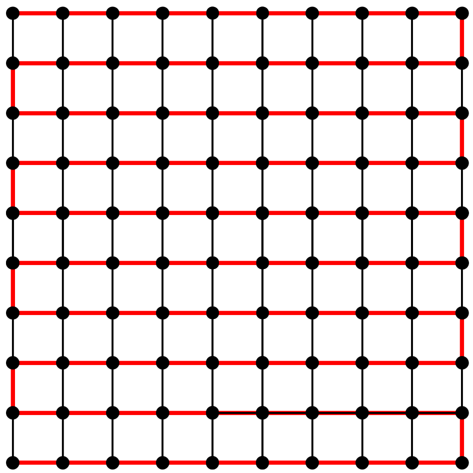
**Examples:** minimum vertex cover, length of the longest path, feedback vertex set are minor-bidimensional.

# Bidimensionality

**Definition**

A graph invariant $x(G)$ is **minor-bidimensional** if

- $x(G') \leq x(G)$ for every minor $G'$ of $G$, and
- If $G_k$ is the $k \times k$ grid, then $x(G_k) \geq ck^2$ (for some constant $c > 0$).



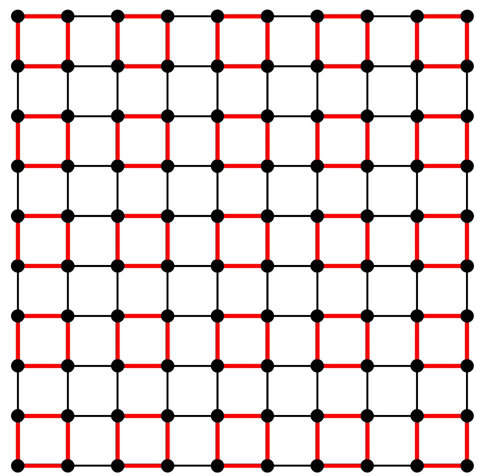
**Examples:** minimum vertex cover, length of the longest path, feedback vertex set are minor-bidimensional.

# Bidimensionality

## Definition

A graph invariant $x(G)$ is **minor-bidimensional** if

- $x(G') \leq x(G)$ for every minor $G'$ of $G$, and
- If $G_k$ is the $k \times k$ grid, then $x(G_k) \geq ck^2$ (for some constant $c > 0$).



**Examples:** minimum vertex cover, length of the longest path, feedback vertex set are minor-bidimensional.

## Bidimensionality (cont.)

We can answer "$x(G) \geq k$?" for a minor-bidimensional invariant the following way:

- Set $w := c\sqrt{k}$ for an appropriate constant $c$.
- Use the 4-approximation tree decomposition algorithm.
  - If treewidth is at least $w$: $x(G)$ is at least $k$.
  - If we get a tree decomposition of width $4w$, then we can solve the problem using dynamic programming on the tree decomposition.

Running time:

- If we can solve the problem on tree decomposition of width $w$ in time $2^{O(w)} \cdot n^{O(1)}$, then the running time is $2^{O(\sqrt{k})} \cdot n^{O(1)}$.
- If we can solve the problem on tree decomposition of width $w$ in time $w^{O(w)} \cdot n^{O(1)}$, then the running time is $2^{O(\sqrt{k} \log k)} \cdot n^{O(1)}$.

# Contraction bidimensionality

**Definition**

A graph invariant $x(G)$ is **minor-bidimensional** if

- $x(G') \leq x(G)$ for every minor $G'$ of $G$, and
- If $G_k$ is the $k \times k$ grid, then $x(G_k) \geq ck^2$ (for some constant $c > 0$).

**Exercise:** DOMINATING SET is **not** minor-bidimensional.

# Contraction bidimensionality

**Definition**

A graph invariant $x(G)$ is **minor-bidimensional** if

- $x(G') \leq x(G)$ for every minor $G'$ of $G$, and
- If $G_k$ is the $k \times k$ grid, then $x(G_k) \geq ck^2$ (for some constant $c > 0$).

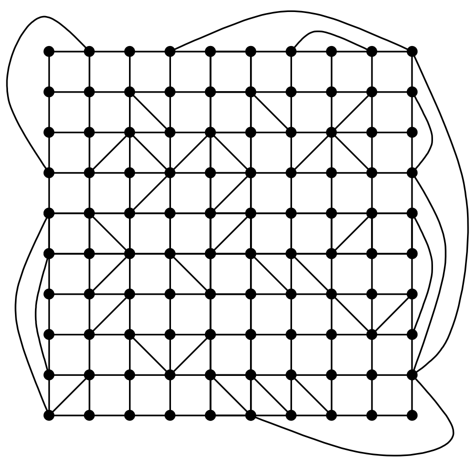
**Exercise:** Dominating Set is **not** minor-bidimensional.

We fix the problem by allowing only contractions but not edge/vertex deletions.

## Contraction bidimensionality

**Theorem**

Every **planar graph** with treewidth at least $5k$ can be contracted to a **partially triangulated** $k \times k$ grid.

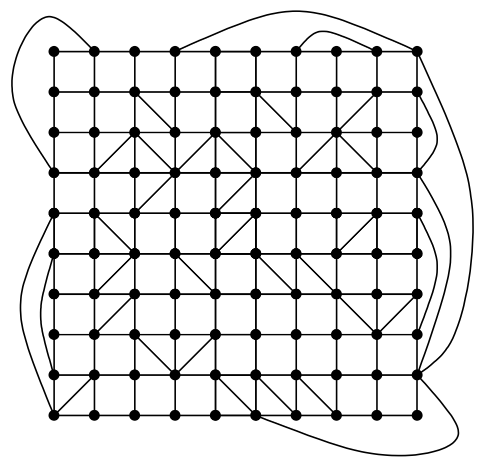


## Contraction bidimensionality

**Definition**

A graph invariant $x(G)$ is **contraction-bidimensional** if

- $x(G') \leq x(G)$ for every contraction $G'$ of $G$, and
- If $G_k$ is a $k \times k$ partially triangulated grid, then $x(G_k) \geq ck^2$ (for some $c > 0$).

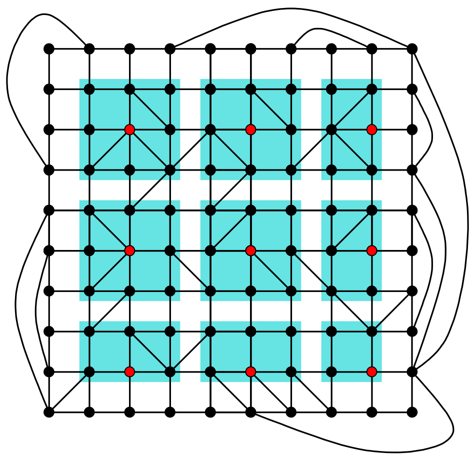


# Contraction bidimensionality

## Definition

A graph invariant $x(G)$ is **contraction-bidimensional** if

- $x(G') \leq x(G)$ for every contraction $G'$ of $G$, and
- If $G_k$ is a $k \times k$ partially triangulated grid, then $x(G_k) \geq ck^2$ (for some $c > 0$).



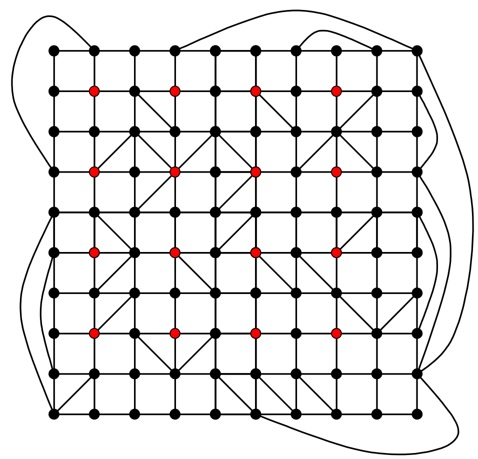
**Example:** minimum dominating set, maximum independent set are contraction-bidimensional.

# Contraction bidimensionality

## Definition

A graph invariant $x(G)$ is **contraction-bidimensional** if

- $x(G') \leq x(G)$ for every contraction $G'$ of $G$, and
- If $G_k$ is a $k \times k$ partially triangulated grid, then $x(G_k) \geq ck^2$ (for some $c > 0$).



**Example:** minimum dominating set, maximum independent set are contraction-bidimensional.

# Bidimensionality for Dominating Set

The size of a minimum dominating set is a **contraction bidimensional** invariant: we need at least $(k-2)^2/9$ vertices to dominate all the internal vertices of a partially triangulated $k \times k$ grid (since a vertex can dominate at most 9 internal vertices).

**Theorem**

Given a tree decomposition of width $w$, Dominating Set can be solved in time $3^w \cdot w^{O(1)} \cdot n^{O(1)}$.

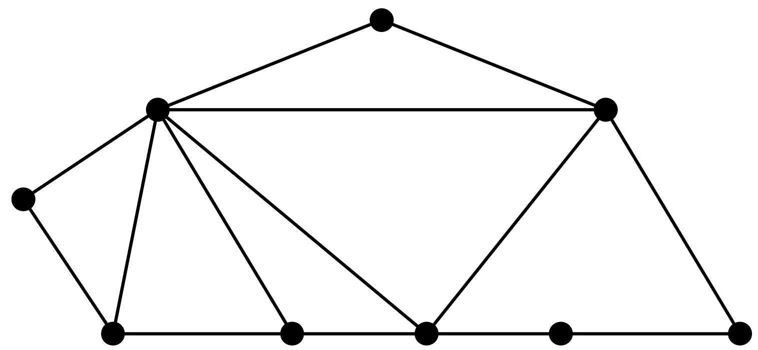
Solving Dominating Set on planar graphs:

- Set $w := 5(3\sqrt{k}+2)$.
- Use the 4-approximation tree decomposition algorithm.
  - If treewidth is at least $w$: we answer 'dominating set is $\geq k$'.
  - If we get a tree decomposition of width $4w$, then we can solve the problem in time $3^w \cdot n^{O(1)} = 2^{O(\sqrt{k})} \cdot n^{O(1)}$.

# Outerplanar graphs

**Definition**

A planar graph is **outerplanar** if it has a planar embedding where every vertex is on the infinite face.



**Fact**

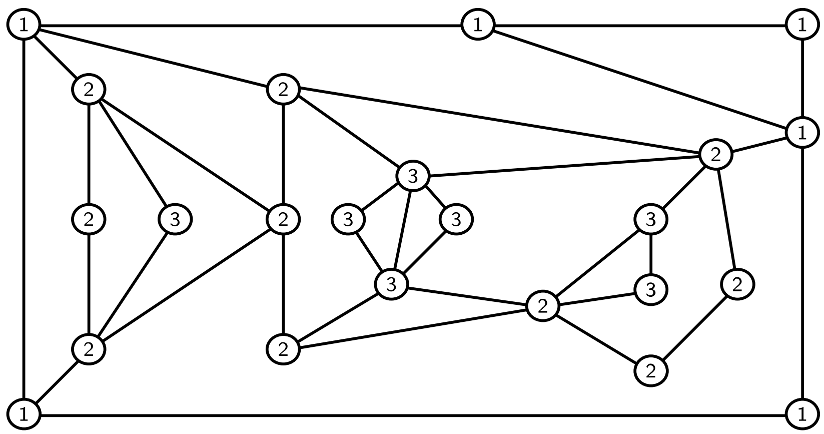
Every outerplanar graph has treewidth at most 2.

## $k$-outerplanar graphs

Given a planar embedding, we can define **layers** by iteratively removing the vertices on the infinite face.

**Definition**

A planar graph is $k$-**outerplanar** if it has a planar embedding having at most $k$ layers.



**Fact**

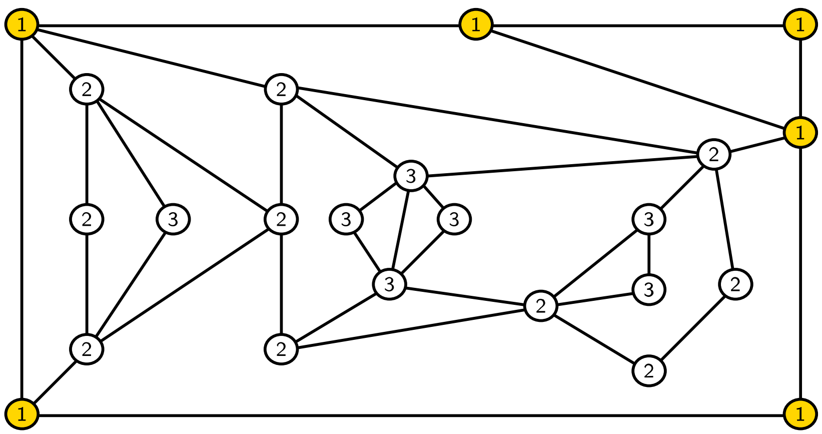
Every $k$-outerplanar graph has treewidth at most $3k + 1$.

## $k$-outerplanar graphs

Given a planar embedding, we can define **layers** by iteratively removing the vertices on the infinite face.

### Definition

A planar graph is $k$-**outerplanar** if it has a planar embedding having at most $k$ layers.




### Fact

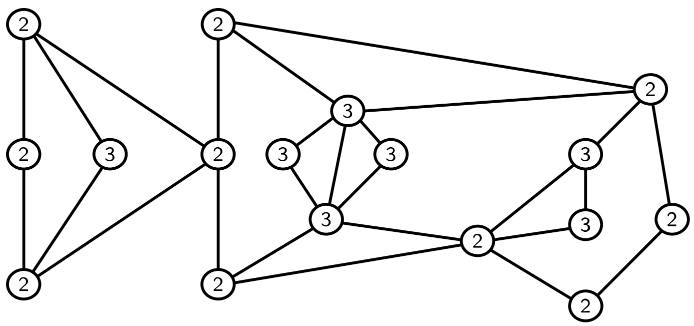
Every $k$-outerplanar graph has treewidth at most $3k + 1$.

## $k$-outerplanar graphs

Given a planar embedding, we can define **layers** by iteratively removing the vertices on the infinite face.

### Definition

A planar graph is $k$-**outerplanar** if it has a planar embedding having at most $k$ layers.




### Fact

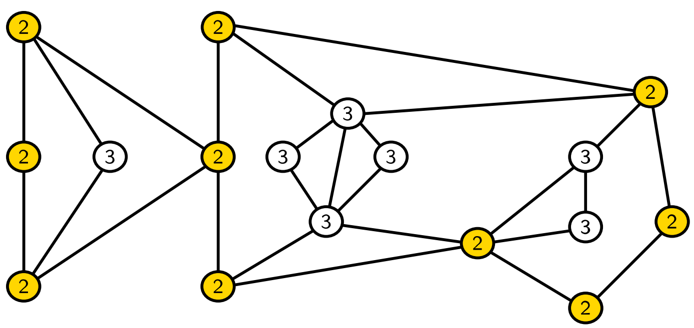
Every $k$-outerplanar graph has treewidth at most $3k + 1$.

## $k$-outerplanar graphs

Given a planar embedding, we can define **layers** by iteratively removing the vertices on the infinite face.

### Definition

A planar graph is $k$-**outerplanar** if it has a planar embedding having at most $k$ layers.




### Fact

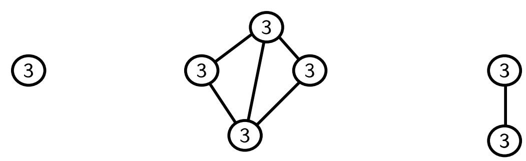
Every $k$-outerplanar graph has treewidth at most $3k + 1$.

## $k$-outerplanar graphs

Given a planar embedding, we can define **layers** by iteratively removing the vertices on the infinite face.

### Definition

A planar graph is $k$-**outerplanar** if it has a planar embedding having at most $k$ layers.



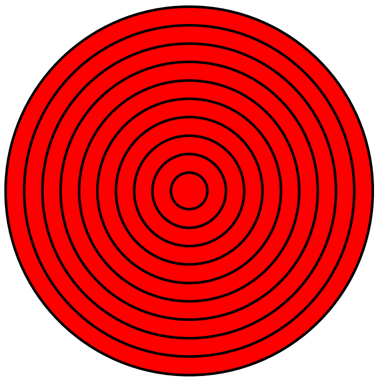

### Fact

Every $k$-outerplanar graph has treewidth at most $3k+1$.

# Baker's shifting strategy for FPT

**Subgraph Isomorphism**

Input: graphs $H$ and $G$
Find: a subgraph $G$ isomorphic to $H$.

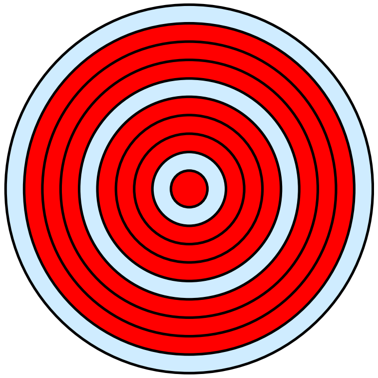


# Baker's shifting strategy for FPT

**Subgraph Isomorphism**

Input: graphs $H$ and $G$

Find: a subgraph $G$ isomorphic to $H$.



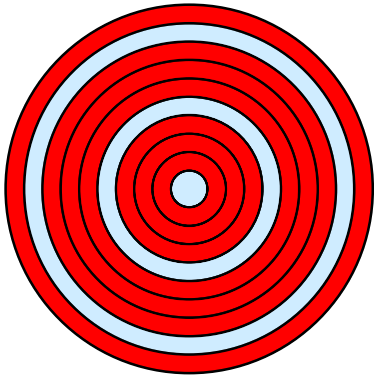
- For a fixed $0 \leq s < k+1$, delete every layer $L_i$ with $i = s \pmod{k+1}$

# Baker's shifting strategy for FPT

**Subgraph Isomorphism**

Input: graphs $H$ and $G$

Find: a subgraph $G$ isomorphic to $H$.

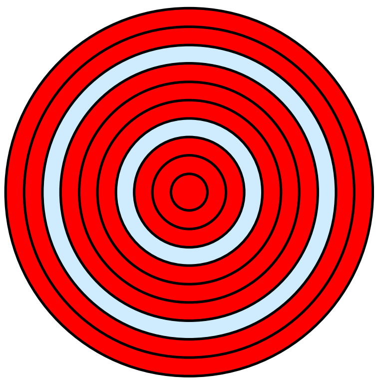


- For a fixed $0 \leq s < k+1$, delete every layer $L_i$ with $i = s \pmod{k+1}$

# Baker's shifting strategy for FPT

**SUBGRAPH ISOMORPHISM**

Input: graphs $H$ and $G$
Find: a subgraph $G$ isomorphic to $H$.



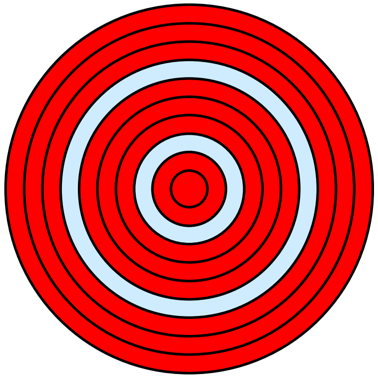
- For a fixed $0 \leq s < k+1$, delete every layer $L_i$ with $i = s \pmod{k+1}$

# Baker's shifting strategy for FPT

**Subgraph Isomorphism**

Input: graphs $H$ and $G$

Find: a subgraph $G$ isomorphic to $H$.

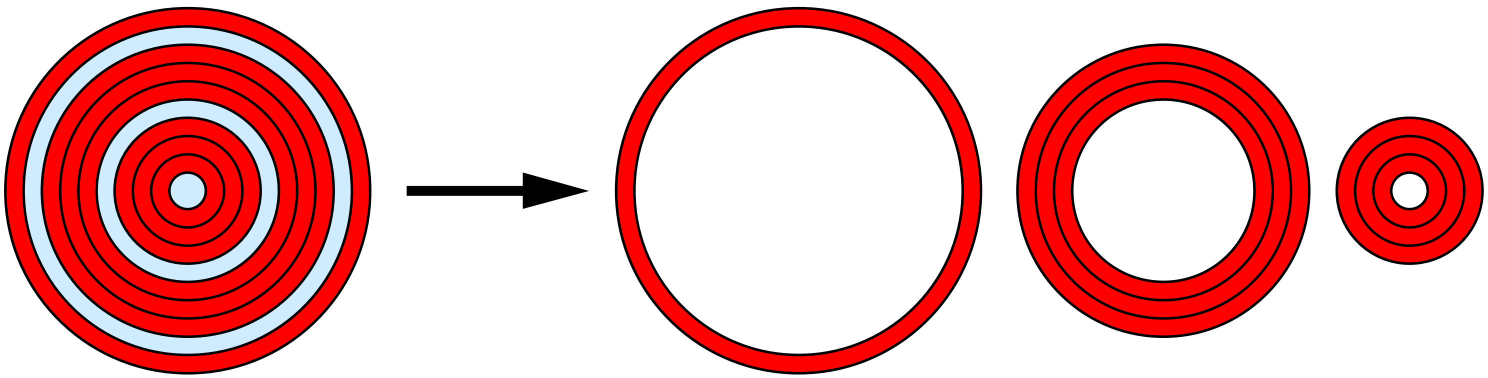


- For a fixed $0 \leq s < k+1$, delete every layer $L_i$ with $i = s \pmod{k+1}$

# Baker's shifting strategy for FPT

**SUBGRAPH ISOMORPHISM**

Input: graphs $H$ and $G$
Find: a subgraph $G$ isomorphic to $H$.



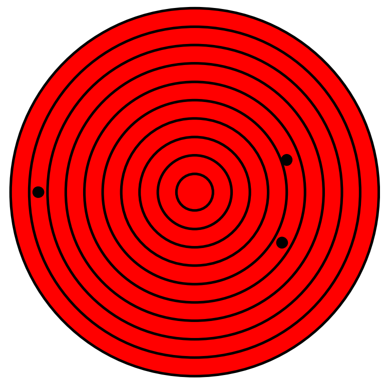
- For a fixed $0 \leq s < k+1$, delete every layer $L_i$ with $i = s \pmod{k+1}$
- The resulting graph is $k$-outerplanar, hence it has treewidth at most $3k+1$.
- Using the $f(k, \mathrm{tw}) \cdot n$ time algorithm for SUBGRAPH ISOMORPHISM, the problem can be solved in time $f(k, 3k+1) \cdot n$.

# Baker's shifting strategy for FPT

**Subgraph Isomorphism**

Input: graphs $H$ and $G$

Find: a subgraph $G$ isomorphic to $H$.



We do this for every $0 \leq s < k+1$:
for at least one value of $s$, we do not delete any of the $k$ vertices of the solution

$\Downarrow$

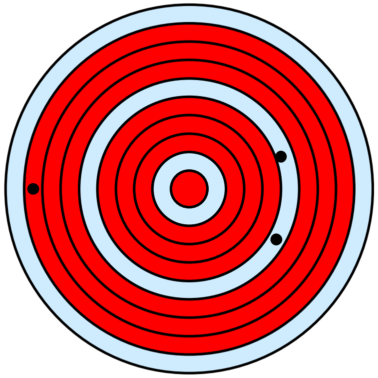
We find a copy of $H$ in $G$ if there is one.

# Baker's shifting strategy for FPT

**Subgraph Isomorphism**

Input: graphs $H$ and $G$

Find: a subgraph $G$ isomorphic to $H$.



We do this for every $0 \leq s < k+1$:
for at least one value of $s$, we do not delete
any of the $k$ vertices of the solution

$\Downarrow$

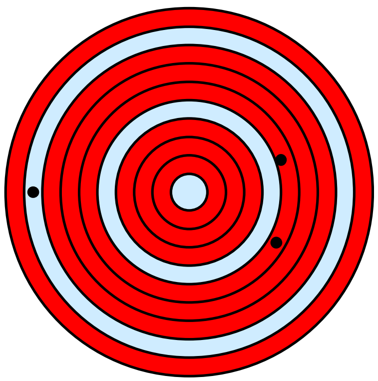
We find a copy of $H$ in $G$ if there is one.

# Baker's shifting strategy for FPT

**SUBGRAPH ISOMORPHISM**

Input: graphs $H$ and $G$

Find: a subgraph $G$ isomorphic to $H$.



We do this for every $0 \leq s < k + 1$:
for at least one value of $s$, we do not delete
any of the $k$ vertices of the solution

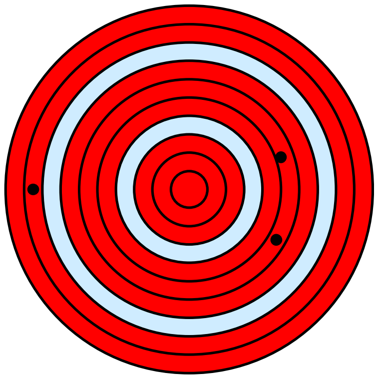
$\Downarrow$

We find a copy of $H$ in $G$ if there is one.

# Baker's shifting strategy for FPT

**Subgraph Isomorphism**

Input: graphs $H$ and $G$
Find: a subgraph $G$ isomorphic to $H$.



We do this for every $0 \leq s < k+1$:
for at least one value of $s$, we do not delete
any of the $k$ vertices of the solution

$\Downarrow$

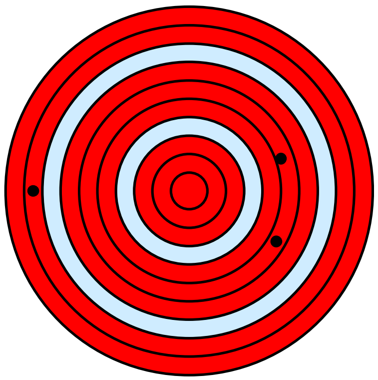
We find a copy of $H$ in $G$ if there is one.

## Baker's shifting strategy for FPT

**Subgraph Isomorphism**

Input: graphs $H$ and $G$

Find: a subgraph $G$ isomorphic to $H$.



**Theorem**

Subgraph Isomorphism for planar graphs is FPT parameterized by $k := |V(H)|$.