Fixed Parameter Algorithms

Dániel Marx

Tel Aviv University, Israel

Open lectures for PhD students in computer science

January 8, 2010, Warsaw, Poland

Recap of last lecture

- **Parameterized problem:** a parameter k is associated with each input instance.
- A parameterized problem is **fixed-parameter tractable (FPT)** if it can be solved in time $f(k) \cdot n^c$ for some function f depending only on k and constant c **not** depending on k.
- We have seen that VERTEX COVER, k-PATH, BIPARTITE DELETION, CHORDAL COMPLETION etc. are FPT parameterized by the size k of the solution.
- We would like $f(k)$ to be as slowly growing as possible (e.g., $O^*(1.2^k)$ is much better than $O^*(2^k)$).

Recap of last lecture

Techniques:

- Kernelization: construct in polynomial time an equivalent instance of size bounded by some function $f(k)$.
- Bounded depth search trees: branch into a constant number of directions, decreasing the parameter in each step.
- Iterative compression: given a solution of size $k+1$, find a solution of size k.
- Graph Minors Theory: if a property is closed under taking minors, then powerful theorems immediately imply FPT algorithms.
- Color coding: assign random colors and solve a “colorful” version of the problem.

Treewidth

- Introduction and definition
- Part I: Algorithms for bounded treewidth graphs.
- Part II: Graph-theoretic properties of treewidth.
- Part III: Applications for general graphs.

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 his direct boss at the same time!

The Party Problem

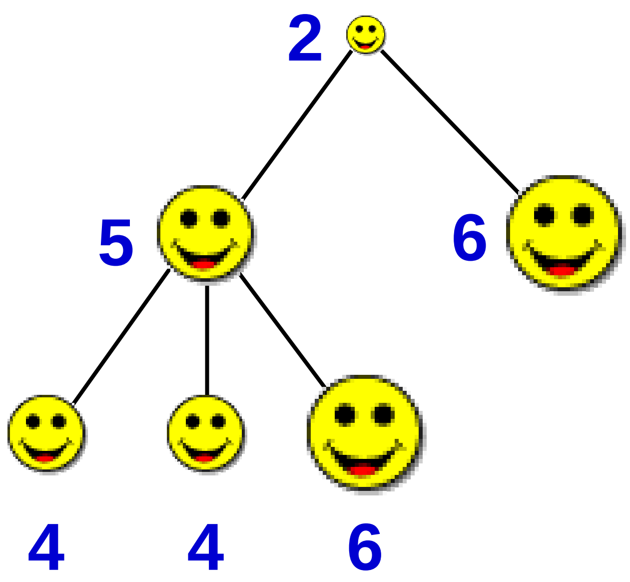
PARTY PROBLEM

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 his 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

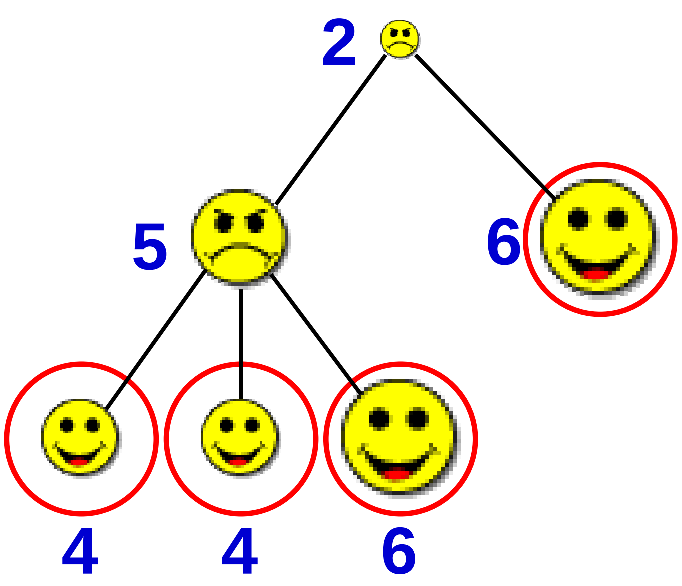
PARTY PROBLEM

Problem: Invite some colleagues for a party.

Maximize: The total fun factor of the invited people.

Constraint: Everyone should be having fun.

Do not invite a colleague and his 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.

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

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

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.

Method:

Assume $v_1, \dots, v_k$ 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).

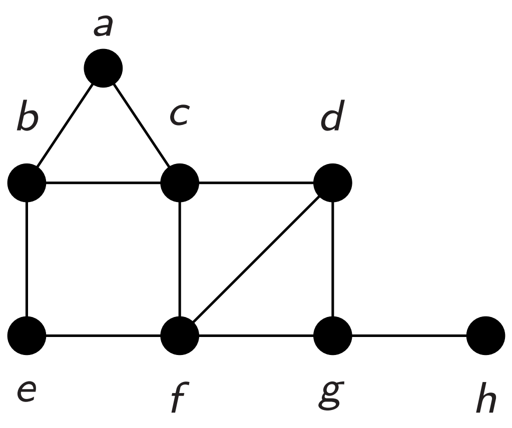
Treewidth

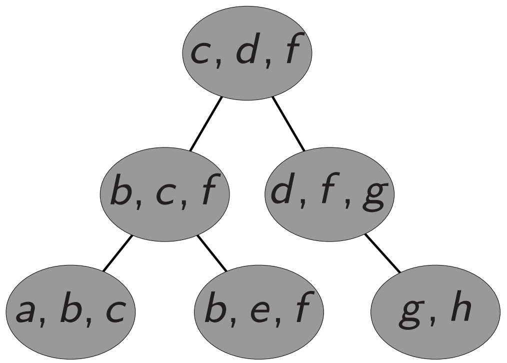
Treewidth

Treewidth: A measure of how "tree-like" the graph is. (Introduced by Robertson and Seymour.)

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 vertex v, the bags containing v form a connected subtree.



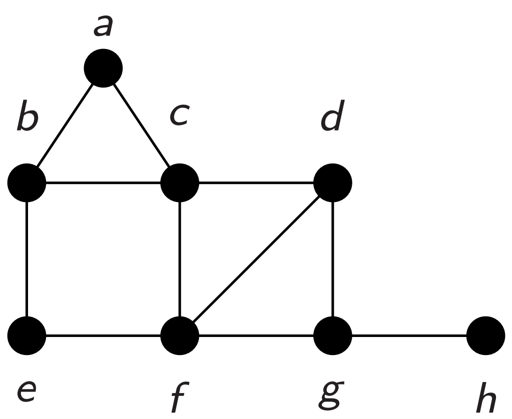


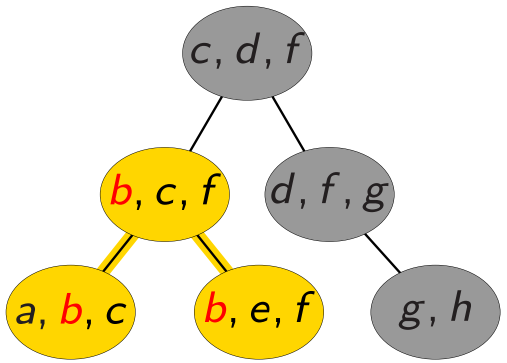
Treewidth

Treewidth: A measure of how “tree-like” the graph is. (Introduced by Robertson and Seymour.)

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 vertex v, the bags containing v form a connected subtree.



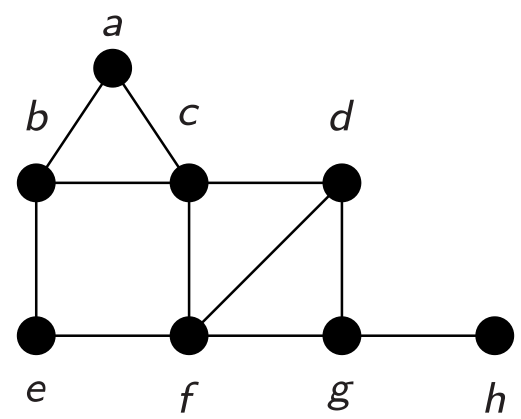


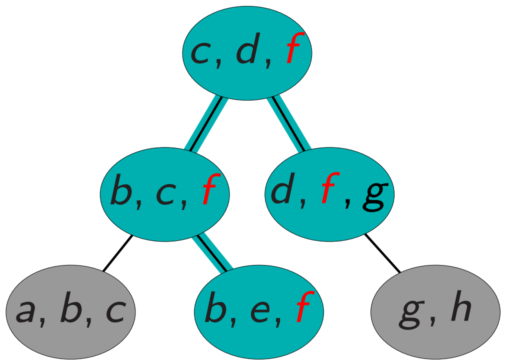
Treewidth

Treewidth: A measure of how “tree-like” the graph is. (Introduced by Robertson and Seymour.)

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

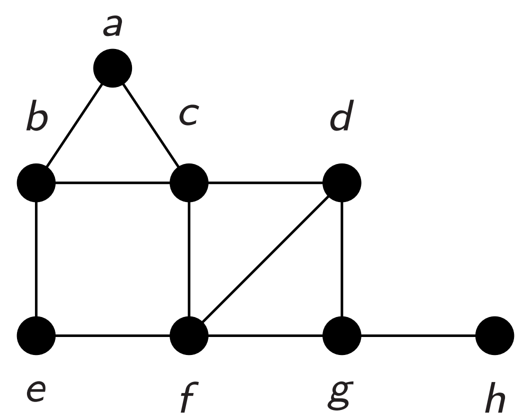
1. If u and v are neighbors, then there is a bag containing both of them.
2. For every vertex v, the bags containing v form a connected subtree.





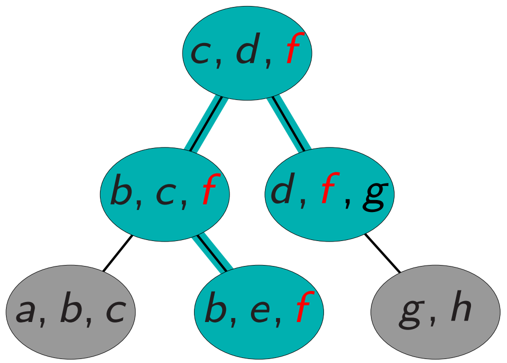
Treewidth

Treewidth: A measure of how "tree-like" the graph is. (Introduced by Robertson and Seymour.)



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 vertex v, the bags containing v form a connected subtree.



Width of the decomposition: largest bag size -1.

treewidth: width of the best decomposition.

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

Treewidth

Treewidth: A measure of how "tree-like" the graph is.
(Introduced by Robertson and Seymour.)

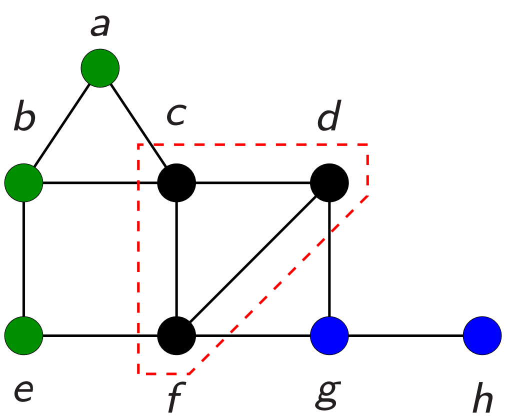
Tree decomposition: Vertices are arranged in a tree structure satisfying the following properties:

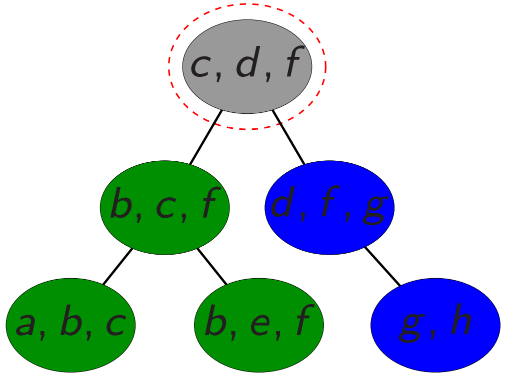
1. If u and v are neighbors, then there is a bag containing both of them.
2. For every vertex v, the bags containing v form a connected subtree.

Width of the decomposition: largest bag size -1.

treewidth: width of the best decomposition.

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





Treewidth

Treewidth: A measure of how “tree-like” the graph is. (Introduced by Robertson and Seymour.)

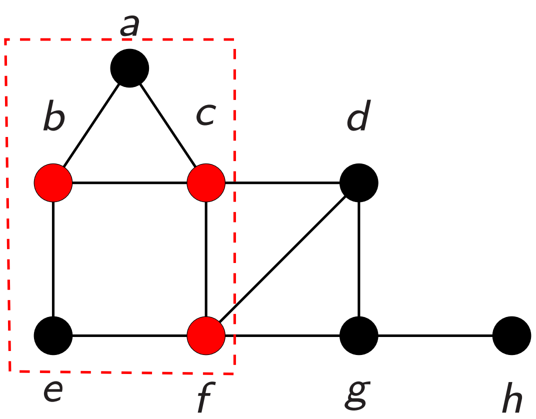
Tree decomposition: Vertices are arranged in a tree structure satisfying the following properties:

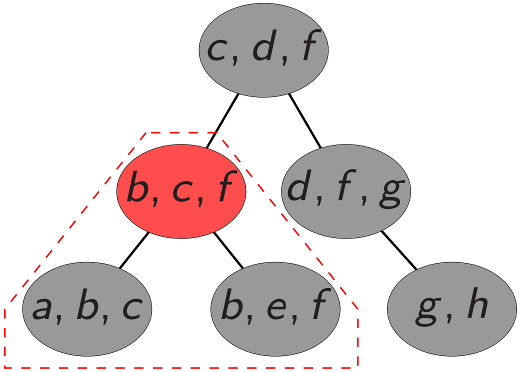
1. If u and v are neighbors, then there is a bag containing both of them.
2. For every vertex v, the bags containing v form a connected subtree.

Width of the decomposition: largest bag size -1.

treewidth: width of the best decomposition.

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





Finding tree decompositions

Fact: 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), but there is a polynomial-time algorithm for every fixed w.

Fact: [Bodlaender's Theorem] For every fixed w, there is a linear-time algorithm that finds a tree decomposition of width w (if exists).

$\Rightarrow$ Deciding if treewidth is at most w is fixed-parameter tractable.

$\Rightarrow$ 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

Fact: 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), but there is a polynomial-time algorithm for every fixed w.

Fact: [Bodlaender's Theorem] For every fixed w, there is a linear-time algorithm that finds a tree decomposition of width w (if exists).

$\Rightarrow$ Deciding if treewidth is at most w is fixed-parameter tractable.
$\Rightarrow$ 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.

Running time is $2^{O(w^3)} \cdot n$. Sometimes it is better to use the following results instead:

Fact: 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.

Fact: 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.

Part I:

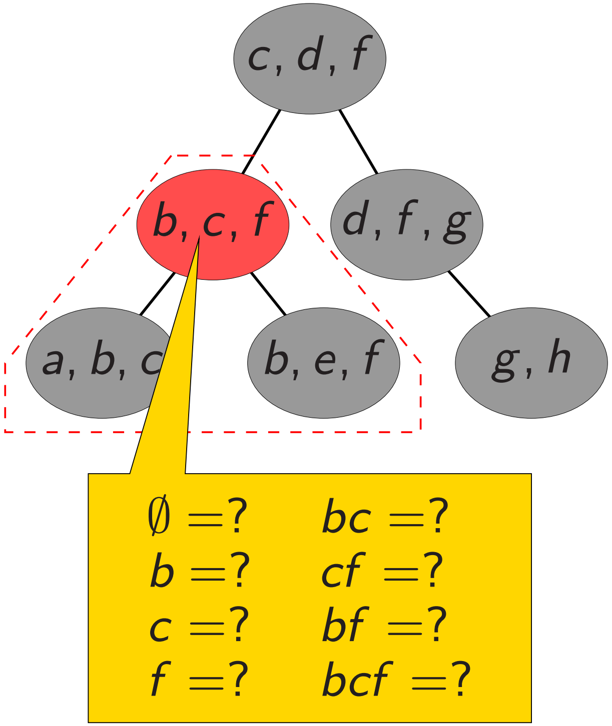
Algoritmhs for
bounded-treewidth graphs

WEIGHTED MAX INDEPENDENT SET *and tree decompositions*

Fact: Given a tree decomposition of width w, WEIGHTED MAX INDEPENDENT SET can be solved in time $O(2^w \cdot n)$.

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 $2^{|B_x|} \leq 2^{w+1}$ values for each bag B_x.

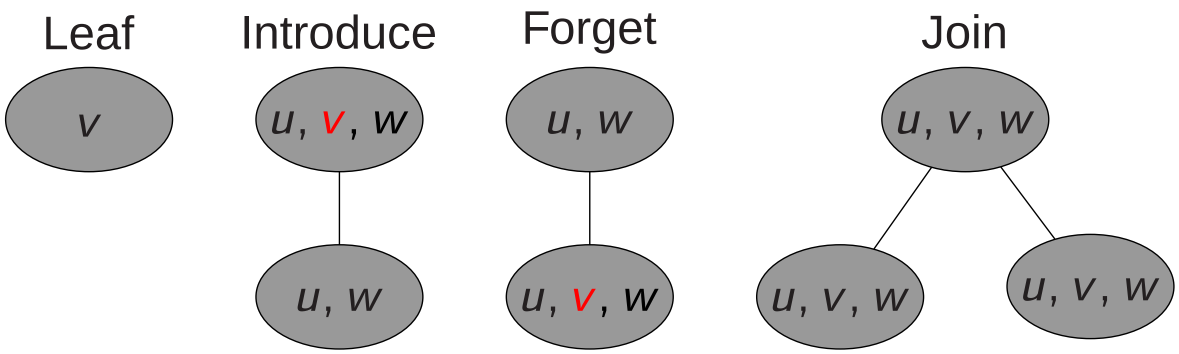
$M[x, S]$: the maximum weight of an independent set $I \subseteq V_x$ with $I \cap B_x = S$.

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, $B_x = B_y \cup \{v\}$ for some vertex v
- **Forget:** 1 child y, $B_x = B_y \setminus \{v\}$ for some vertex v
- **Join:** 2 children y_1, y_2 with $B_x = B_{y_1} = B_{y_2}$

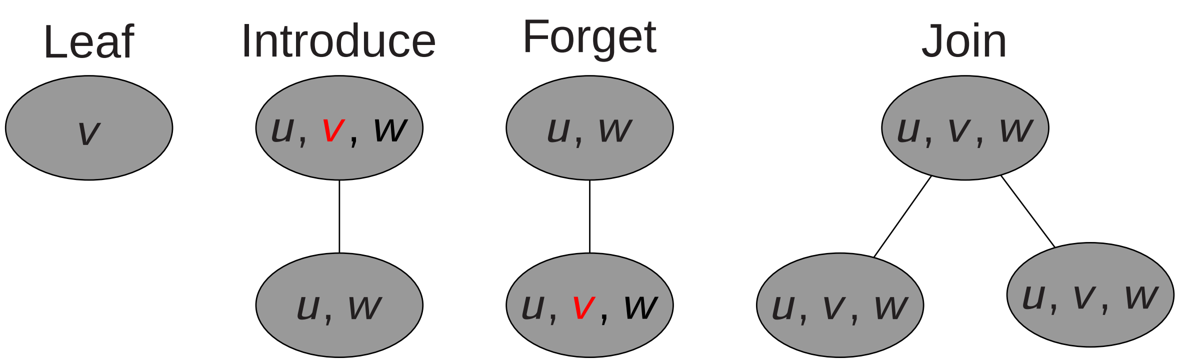


Fact: A tree decomposition of width w and n nodes can be turned into a nice tree decomposition of width w and $O(wn)$ nodes in time $O(w^2n)$.

WEIGHTED MAX INDEPENDENT SET *and nice tree decompositions*

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

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



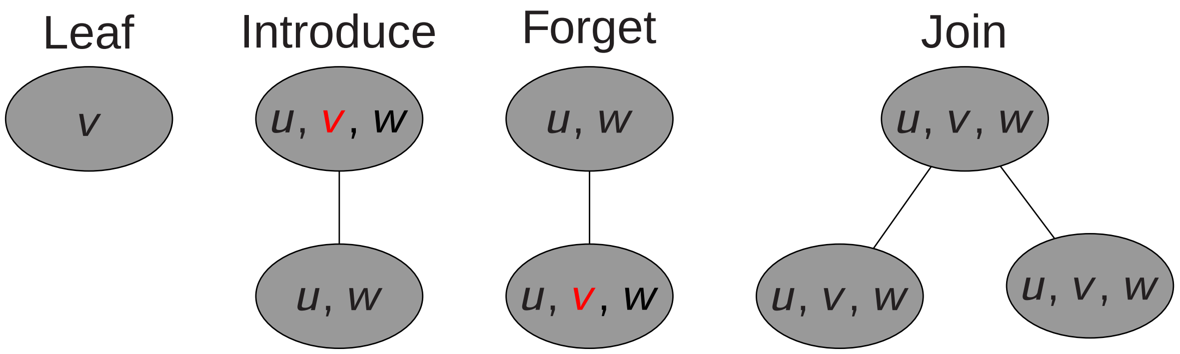
WEIGHTED MAX INDEPENDENT SET *and nice tree decompositions*

- **Forget:** 1 child y, $B_x = B_y \setminus \{v\}$ 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, $B_x = B_y \setminus \{v\}$ 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 constant time (assuming the children are already solved).

$\Rightarrow$ Running time is $O(2^w \cdot n)$.

$\Rightarrow$ WEIGHTED MAX INDEPENDENT SET is FPT parameterized by treewidth.

$\Rightarrow$ WEIGHTED MIN VERTEX COVER is FPT parameterized by treewidth.

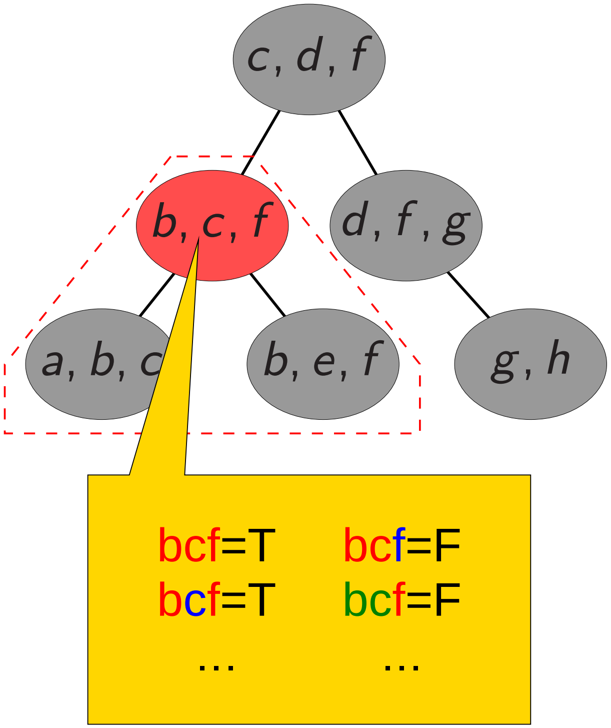
3-Coloring *and* *tree decompositions*

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

B_x: vertices appearing in node x.

V_x: vertices appearing in the subtree rooted at x.

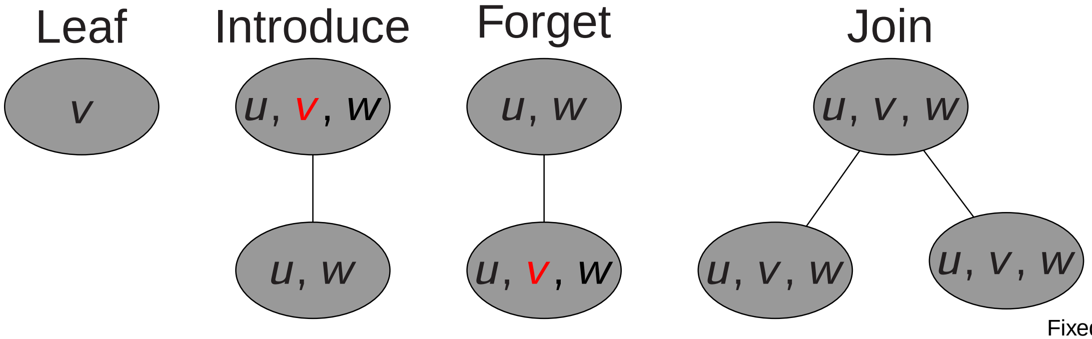
For every node x and coloring $c : B_x \longrightarrow \{1, 2, 3\}$, we compute the Boolean value $E[x, c]$, which is true if and only if c can be extended to a proper 3-coloring of V_x.



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!
- **Introduce:** 1 child y, $B_x = B_y \cup \{v\}$ 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, $B_x = B_y \setminus \{v\}$ 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, $B_x = B_y \cup \{v\}$ 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, $B_x = B_y \setminus \{v\}$ 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 constant time (assuming the children are already solved).

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

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

Vertex coloring

More generally:

Fact: Given a tree decomposition of width w, c-COLORING can be solved in $O^*(c^w)$.

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

Fact: 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.

Hamiltonian cycle and tree decompositions

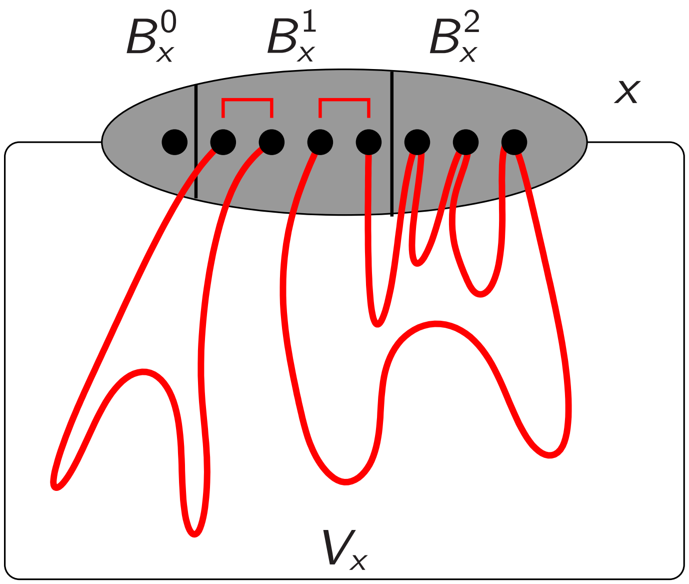
Fact: Given a tree decomposition of width w, HAMILTONIAN CYCLE can be solved in time $w^{O(w)} \cdot n$.

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

If H is a Hamiltonian cycle, then the subgraph $H[V_x]$ is a set of paths with endpoints in B_x.

What are the important properties of $H[V_x]$ "seen from the outside world"?

- The subsets B_x^0, B_x^1, B_x^2 of B_x having degree 0, 1, and 2.
- The matching M of B_x^1.



Number of subproblems (B_x^0, B_x^1, B_x^2, M) for each node x: at most $3^w \cdot w^w$.

Hamiltonian cycle and nice tree decompositions

For each subproblem (B_x^0, B_x^1, B_x^2, M), we have to determine if there is a set of paths with this pattern.

How to do this for the different types of nodes?
(Assuming that all the subproblems are solved for the children.)

Leaf: no children, $|B_x| = 1$

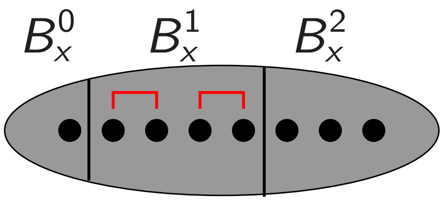
Trivial!

Hamiltonian cycle and nice tree decompositions

Solving subproblem (B_x^0, B_x^1, B_x^2, M) of node x.

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

In a solution H of (B_x^0, B_x^1, B_x^2, M), vertex v has degree 2. Thus subproblem (B_x^0, B_x^1, B_x^2, M) of x is equivalent to subproblem $(B_x^0, B_x^1, B_x^2 \cup \{v\}, M)$ of y.

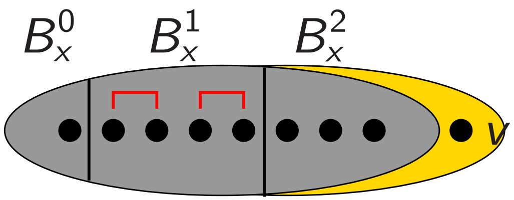


Hamiltonian cycle and nice tree decompositions

Solving subproblem (B_x^0, B_x^1, B_x^2, M) of node x.

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

In a solution H of (B_x^0, B_x^1, B_x^2, M), vertex v has degree 2. Thus subproblem (B_x^0, B_x^1, B_x^2, M) of x is equivalent to subproblem $(B_x^0, B_x^1, B_x^2 \cup \{v\}, M)$ of y.

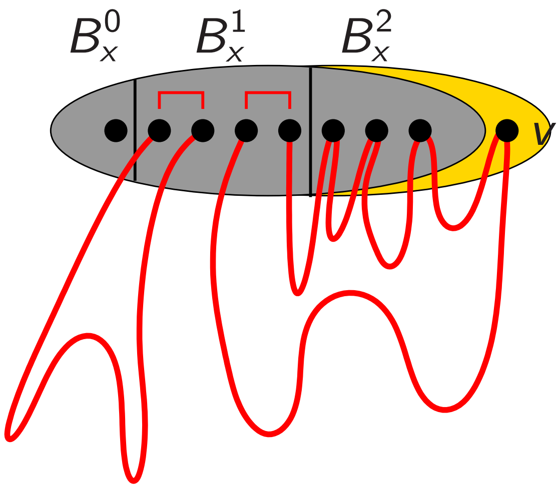


Hamiltonian cycle and nice tree decompositions

Solving subproblem (B_x^0, B_x^1, B_x^2, M) of node x.

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

In a solution H of (B_x^0, B_x^1, B_x^2, M), vertex v has degree 2. Thus subproblem (B_x^0, B_x^1, B_x^2, M) of x is equivalent to subproblem $(B_x^0, B_x^1, B_x^2 \cup \{v\}, M)$ of y.

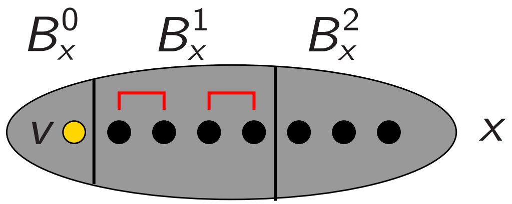


Hamiltonian cycle and nice tree decompositions

Solving subproblem (B^0_x, B^1_x, B^2_x, M) of node x.

Introduce: 1 child y, $B_x = B_y \cup \{v\}$ for some vertex v

Case 1: $v \in B^0_x$. Subproblem is equivalent with $(B^0_x \setminus \{v\}, B^1_x, B^2_x, M)$ for node y.



Hamiltonian cycle and nice tree decompositions

Solving subproblem (B_x^0, B_x^1, B_x^2, M) of node x.

Introduce: 1 child y, $B_x = B_y \cup \{v\}$ for some vertex v

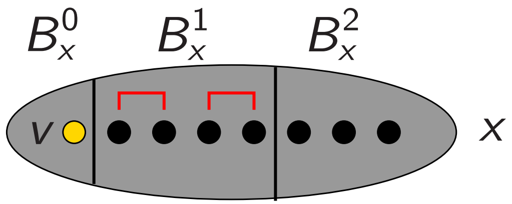
Case 1: $v \in B_x^0$. Subproblem is equivalent with $(B_x^0 \setminus \{v\}, B_x^1, B_x^2, M)$ for node y.

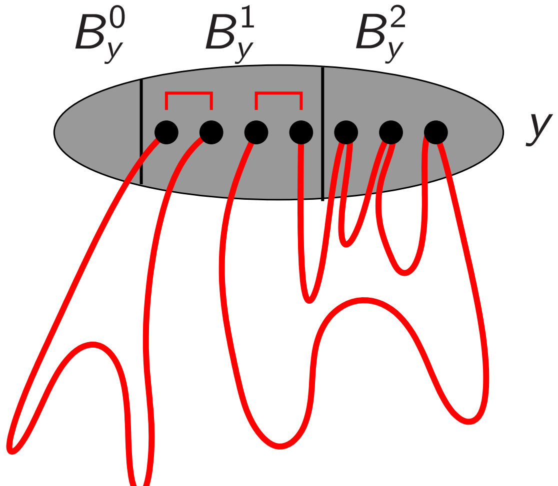
Hamiltonian cycle and nice tree decompositions

Solving subproblem (B_x^0, B_x^1, B_x^2, M) of node x.

Introduce: 1 child y, $B_x = B_y \cup \{v\}$ for some vertex v

Case 1: $v \in B_x^0$. Subproblem is equivalent with $(B_x^0 \setminus \{v\}, B_x^1, B_x^2, M)$ for node y.



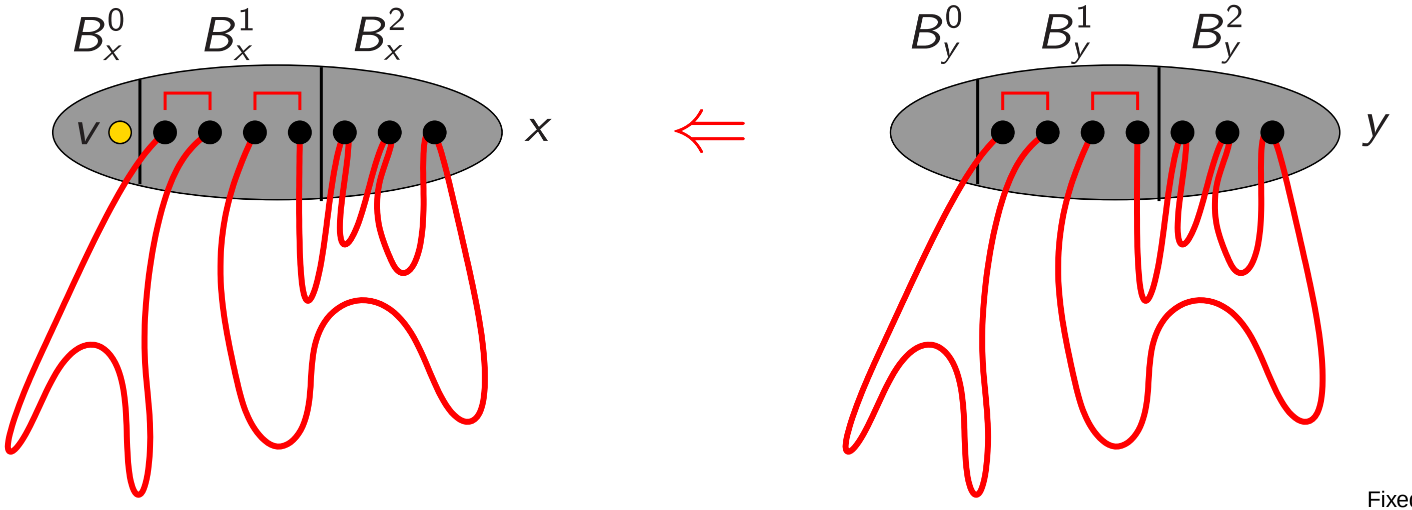


Hamiltonian cycle and nice tree decompositions

Solving subproblem (B_x^0, B_x^1, B_x^2, M) of node x.

Introduce: 1 child y, $B_x = B_y \cup \{v\}$ for some vertex v

Case 1: $v \in B_x^0$. Subproblem is equivalent with $(B_x^0 \setminus \{v\}, B_x^1, B_x^2, M)$ for node y.



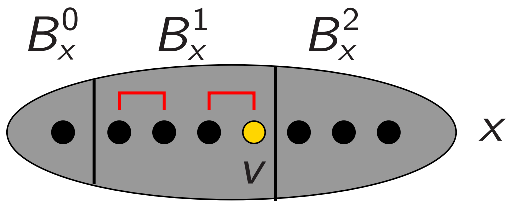
Hamiltonian cycle and nice tree decompositions

Solving subproblem (B_x^0, B_x^1, B_x^2, M) of node x.

Introduce: 1 child y, $B_x = B_y \cup \{v\}$ for some vertex v

Case 2: $v \in B_x^1$. Every neighbor of v in V_x is in B_x. Thus v has to be adjacent with one other vertex of B_x.

Is there a subproblem $(B_y^0, B_y^1, B_y^2, M')$ of node y such that making a vertex of B_y adjacent to v makes it a solution for subproblem (B_x^0, B_x^1, B_x^2, M) of node x?



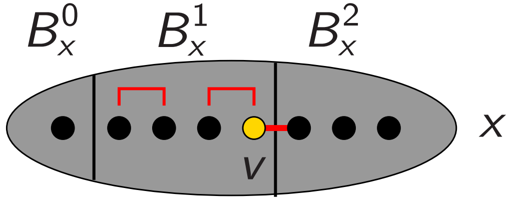
Hamiltonian cycle and nice tree decompositions

Solving subproblem (B_x^0, B_x^1, B_x^2, M) of node x.

Introduce: 1 child y, $B_x = B_y \cup \{v\}$ for some vertex v

Case 2: $v \in B_x^1$. Every neighbor of v in V_x is in B_x. Thus v has to be adjacent with one other vertex of B_x.

Is there a subproblem $(B_y^0, B_y^1, B_y^2, M')$ of node y such that making a vertex of B_y adjacent to v makes it a solution for subproblem (B_x^0, B_x^1, B_x^2, M) of node x?



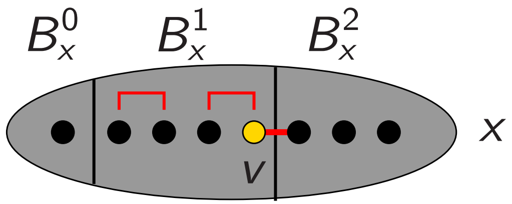
Hamiltonian cycle and nice tree decompositions

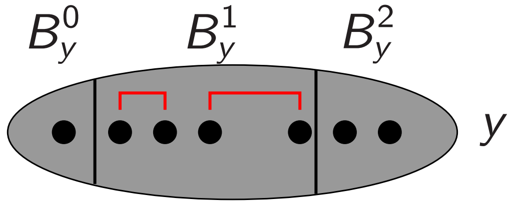
Solving subproblem (B_x^0, B_x^1, B_x^2, M) of node x.

Introduce: 1 child y, $B_x = B_y \cup \{v\}$ for some vertex v

Case 2: $v \in B_x^1$. Every neighbor of v in V_x is in B_x. Thus v has to be adjacent with one other vertex of B_x.

Is there a subproblem $(B_y^0, B_y^1, B_y^2, M')$ of node y such that making a vertex of B_y adjacent to v makes it a solution for subproblem (B_x^0, B_x^1, B_x^2, M) of node x?





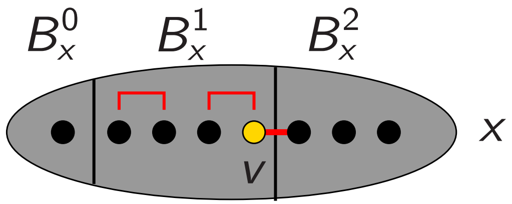
Hamiltonian cycle and nice tree decompositions

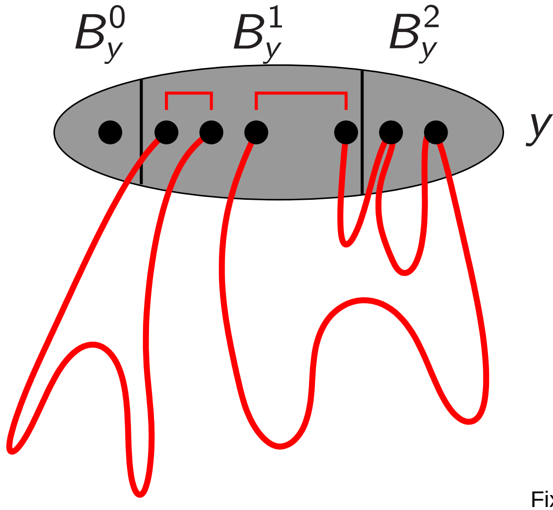
Solving subproblem (B_x^0, B_x^1, B_x^2, M) of node x.

Introduce: 1 child y, $B_x = B_y \cup \{v\}$ for some vertex v

Case 2: $v \in B_x^1$. Every neighbor of v in V_x is in B_x. Thus v has to be adjacent with one other vertex of B_x.

Is there a subproblem $(B_y^0, B_y^1, B_y^2, M')$ of node y such that making a vertex of B_y adjacent to v makes it a solution for subproblem (B_x^0, B_x^1, B_x^2, M) of node x?





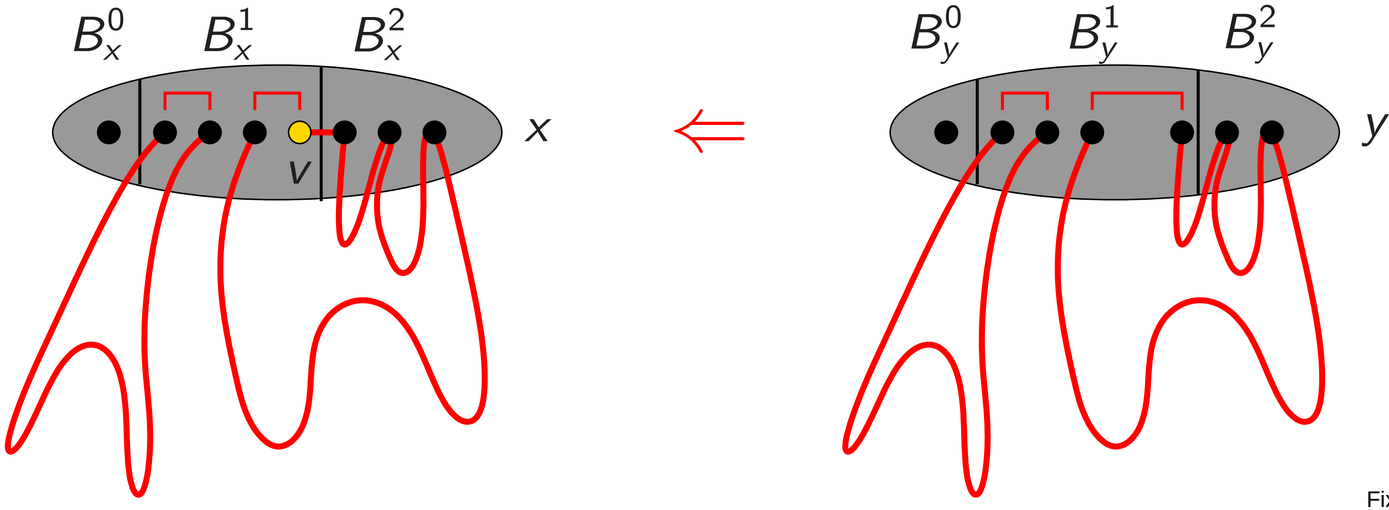
Hamiltonian cycle and nice tree decompositions

Solving subproblem (B_x^0, B_x^1, B_x^2, M) of node x.

Introduce: 1 child y, $B_x = B_y \cup \{v\}$ for some vertex v

Case 2: $v \in B_x^1$. Every neighbor of v in V_x is in B_x. Thus v has to be adjacent with one other vertex of B_x.

Is there a subproblem $(B_y^0, B_y^1, B_y^2, M')$ of node y such that making a vertex of B_y adjacent to v makes it a solution for subproblem (B_x^0, B_x^1, B_x^2, M) of node x?

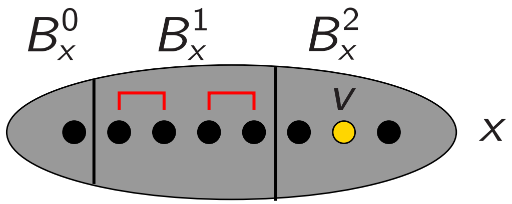


Hamiltonian cycle and nice tree decompositions

Solving subproblem (B_x^0, B_x^1, B_x^2, M) of node x.

Introduce: 1 child y, $B_x = B_y \cup \{v\}$ for some vertex v

Case 3: $v \in B_x^1$. Similar to Case 2, but 2 vertices of B_y are adjacent with v.

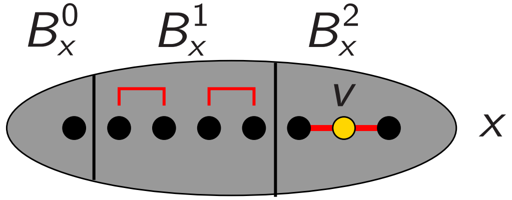


Hamiltonian cycle and nice tree decompositions

Solving subproblem (B_x^0, B_x^1, B_x^2, M) of node x.

Introduce: 1 child y, $B_x = B_y \cup \{v\}$ for some vertex v

Case 3: $v \in B_x^1$. Similar to Case 2, but 2 vertices of B_y are adjacent with v.



Hamiltonian cycle and nice tree decompositions

Solving subproblem (B_x^0, B_x^1, B_x^2, M) of node x.

Introduce: 1 child y, $B_x = B_y \cup \{v\}$ for some vertex v

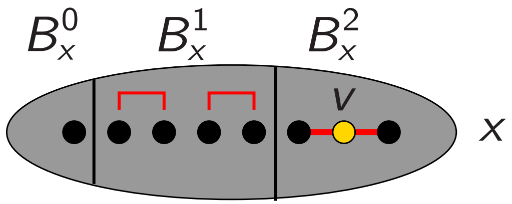
Case 3: $v \in B_x^1$. Similar to Case 2, but 2 vertices of B_y are adjacent with v.

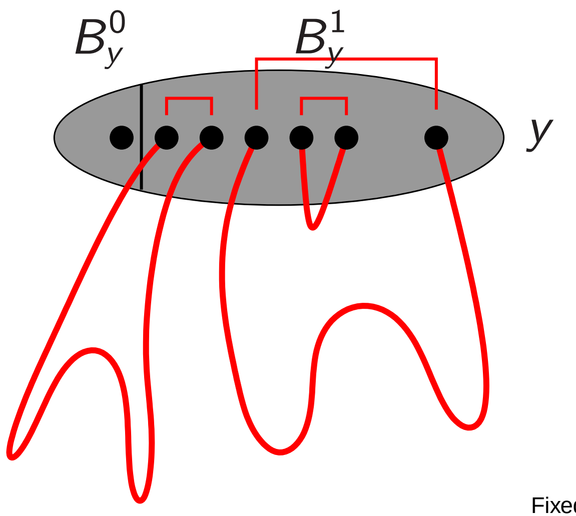
Hamiltonian cycle and nice tree decompositions

Solving subproblem (B_x^0, B_x^1, B_x^2, M) of node x.

Introduce: 1 child y, $B_x = B_y \cup \{v\}$ for some vertex v

Case 3: $v \in B_x^1$. Similar to Case 2, but 2 vertices of B_y are adjacent with v.



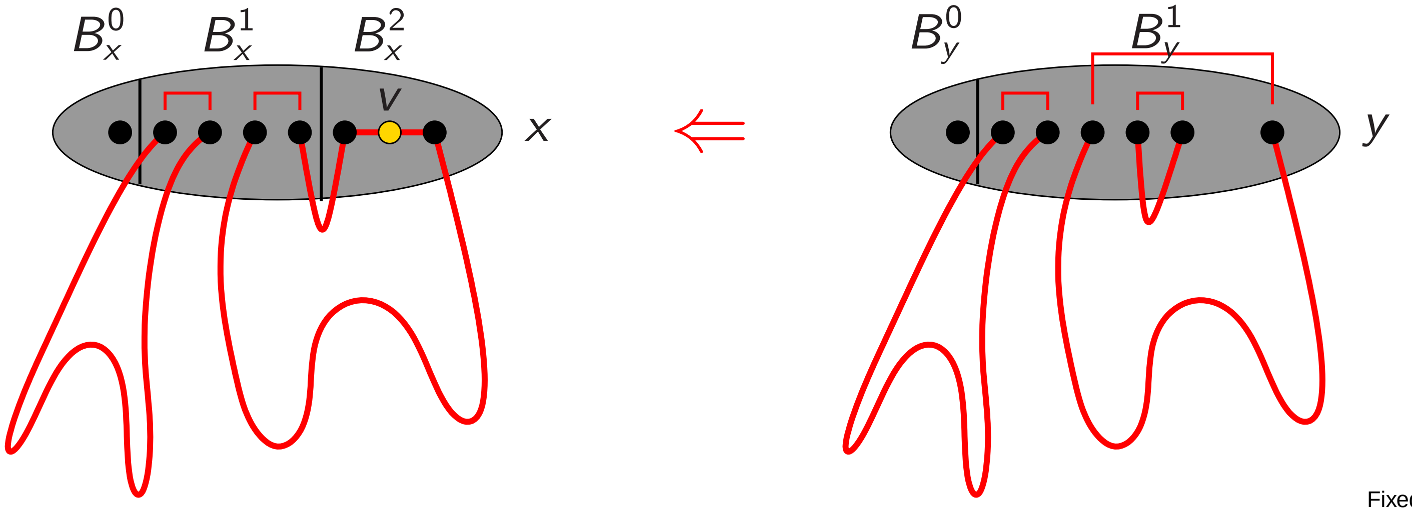


Hamiltonian cycle and nice tree decompositions

Solving subproblem (B_x^0, B_x^1, B_x^2, M) of node x.

Introduce: 1 child y, $B_x = B_y \cup \{v\}$ for some vertex v

Case 3: $v \in B_x^1$. Similar to Case 2, but 2 vertices of B_y are adjacent with v.



Hamiltonian cycle and nice tree decompositions

Solving subproblem (B_x^0, B_x^1, B_x^2, M) of node x.

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

A solution H is the union of a subgraph $H_1 \subseteq G[V_{y_1}]$ and a subgraph $H_2 \subseteq G[V_{y_2}]$.

If H_1 is a solution for $(B_{y_1}^0, B_{y_1}^1, B_{y_1}^2, M_1)$ of node y_1 and H_2 is a solution for $(B_{y_2}^0, B_{y_2}^1, B_{y_2}^2, M_2)$ of node y_2, then we can check if $H_1 \cup H_2$ is a solution for (B_x^0, B_x^1, B_x^2, M) of node x.

For any two subproblems of y_1 and y_2, we check if they have solutions and if their union is a solution for (B_x^0, B_x^1, B_x^2, M) of node x.

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 adj(u, v): vertices u and v are adjacent
- predicate 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 ...

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 if graph $G(V, E)$ 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.

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

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.

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

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.

Can we express 3-COLORING and HAMILTONIAN CYCLE in EMSO?

Using Courcelle's Theorem

3-COLORING

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

Using Courcelle's Theorem

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

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

$$\mathrm{degree0}(H, v) := \neg\exists e \in H\, \mathrm{inc}(e, v)$$
$$\mathrm{degree1}(H, v) := \neg\mathrm{degree0}(H, v) \wedge \big(\neg\exists e_1, e_2 \in H\, (e_1 \neq e_2 \wedge \mathrm{inc}(e_1, v) \wedge \mathrm{inc}(e_2, v))\big)$$
$$\mathrm{degree2}(H, v) := \neg\mathrm{degree0}(H, v) \wedge \neg\mathrm{degree1}(H, v) \wedge \big(\neg\exists e_1, e_2, e_3 \in H\, (e_1 \neq e_2 \wedge e_2 \neq e_3 \wedge e_1 \neq e_3 \wedge \mathrm{inc}(e_1, v) \wedge \mathrm{inc}(e_2, v) \wedge \mathrm{inc}(e_3, v))\big)$$
$$\mathrm{spanning}(H) := \forall u, v \in V\, \exists P \subseteq H\, \forall x \in V\, \big(((x = u \vee x = v) \wedge \mathrm{degree1}(P, x)) \vee (x \neq u \wedge x \neq v \wedge (\mathrm{degree0}(P, x) \vee \mathrm{degree2}(P, x)))\big)$$

Using Courcelle's Theorem

Two ways of using Courcelle's Theorem:

1. The problem can be described by a single formula (e.g, 3-COLORING, HAMILTONIAN CYCLE).

$\Rightarrow$ Problem can be solved in time $f(w) \cdot n$ for graphs of treewidth at most w.

$\Rightarrow$ Problem is FPT parameterized by the treewidth w of the input graph.

Using Courcelle's Theorem

Two ways of using Courcelle's Theorem:

1. The problem can be described by a single formula (e.g, 3-COLORING, HAMILTONIAN CYCLE).

⇒ Problem can be solved in time $f(w) \cdot n$ for graphs of treewidth at most w.

⇒ Problem is FPT parameterized by the treewidth w of the input graph.

2. The problem can be described by a formula for each value of the parameter k.

Example: For each k, having a cycle of length exactly k can be expressed as

$$\exists v_1, \ldots, v_k \in V\,(\text{adj}(v_1, v_2) \wedge \text{adj}(v_2, v_3) \wedge \cdots \wedge \text{adj}(v_{k-1}, v_k) \wedge \text{adj}(v_k, v_1)).$$

⇒ Problem can be solved in time $f(k, w) \cdot n$ for graphs of treewidth w.

⇒ Problem is FPT parameterized with combined parameter k and treewidth w.

SUBGRAPH ISOMORPHISM

SUBGRAPH ISOMORPHISM: given graphs H and G, find a copy of H in G as subgraph. Parameter $k := |V(H)|$ (size of the small graph).

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

SUBGRAPH ISOMORPHISM

SUBGRAPH ISOMORPHISM: given graphs H and G, find a copy of H in G as subgraph. Parameter $k := |V(H)|$ (size of the small graph).

For each H, we can construct a formula ϕ_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: given graphs H and G, find a copy of H in G as subgraph. Parameter $k := |V(H)|$ (size of the small graph).

For each H, we can construct a formula ϕ_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.

$\Rightarrow$ 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.

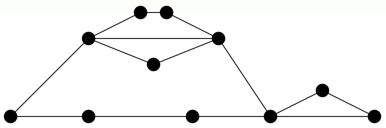
$\Rightarrow$ SUBGRAPH ISOMORPHISM is FPT parameterized by combined parameter $k := |V(H)|$ and the treewidth w of G.

Part II:

Graph-theoretical properties of treewidth

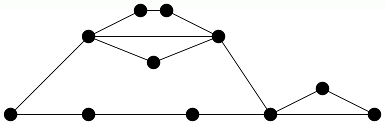
Properties of treewidth

Fact: treewidth ≤ 2 $\Longleftrightarrow$ graph is subgraph of a series-parallel graph

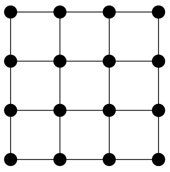


Properties of treewidth

Fact: treewidth $\leq 2 \iff$ graph is subgraph of a series-parallel graph

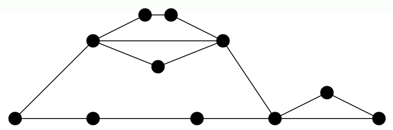


Fact: For every $k \geq 2$, the treewidth of the $k \times k$ grid is exactly k.

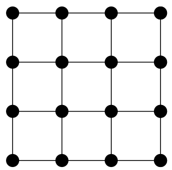


Properties of treewidth

Fact: treewidth $\leq 2 \iff$ graph is subgraph of a series-parallel graph



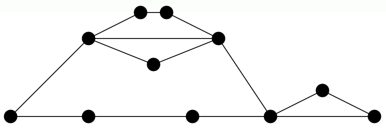
Fact: For every $k \geq 2$, the treewidth of the $k \times k$ grid is exactly k.



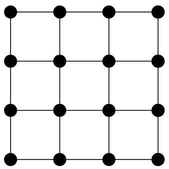
Fact: Treewidth does not increase if we delete edges, delete vertices, or contract edges.

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

Properties of treewidth



Fact: treewidth $\leq 2 \iff$ graph is subgraph of a series-parallel graph



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

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.

The treewidth of the k-clique is $k-1$. Follows from:

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

Excluded Grid Theorem

Fact: [Excluded Grid Theorem] If the treewidth of G is at least $k^{4k^2(k+2)}$, then G has a $k \times k$ grid minor.

A large grid minor is a “witness” that treewidth is large.

Fact: Every **planar graph** with treewidth at least $4k$ has $k \times k$ grid minor.

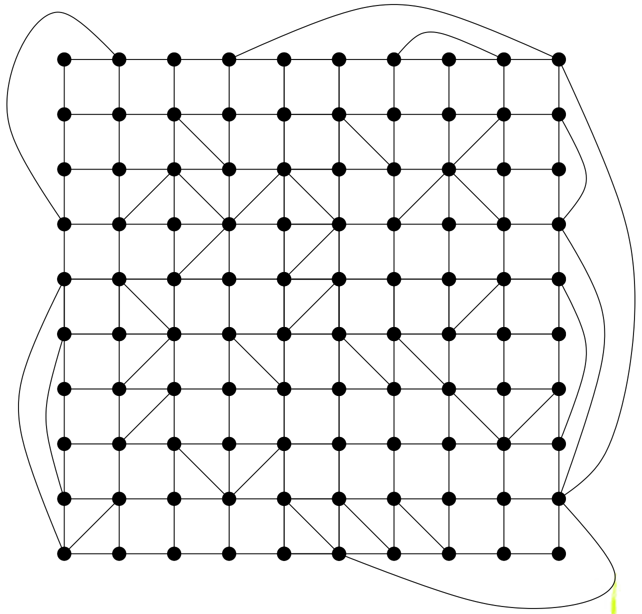
Excluded Grid Theorem

Fact: [Excluded Grid Theorem] If the treewidth of G is at least $k^{4k^2(k+2)}$, then G has a $k \times k$ grid minor.

A large grid minor is a “witness” that treewidth is large.

Fact: Every **planar graph** with treewidth at least $4k$ has $k \times k$ grid minor.

Fact: Every **planar graph** with treewidth at least $4k$ can be contracted to a **partially triangulated** $k \times k$ grid.



The Robber and Cops game

Game: k cops try to capture a robber in the graph.

- In each step, the cops can move from vertex to vertex arbitrarily with helicopters.
- The robber moves infinitely fast on the edges, and sees where the cops will land.

Fact:

$k+1$ cops can win the game $\iff$ the treewidth of the graph is at most k.

The Robber and Cops game

Game: k cops try to capture a robber in the graph.

- In each step, the cops can move from vertex to vertex arbitrarily with helicopters.
- The robber moves infinitely fast on the edges, and sees where the cops will land.

Fact:

$k + 1$ cops can win the game $\iff$ the treewidth of the graph is at most k.

The winner of the game can be determined in time $n^{O(k)}$ using standard techniques (there are at most n^k positions for the cops)

$$\Downarrow$$

For every fixed k, it can be checked in polynomial-time if treewidth is at most k.

The Robber and Cops game

Game: k cops try to capture a robber in the graph.

- In each step, the cops can move from vertex to vertex arbitrarily with helicopters.
- The robber moves infinitely fast on the edges, and sees where the cops will land.

Fact:

$k+1$ cops can win the game $\iff$ the treewidth of the graph is at most k.

The winner of the game can be determined in time $n^{O(k)}$ using standard techniques (there are at most n^k positions for the cops)

$$\Downarrow$$

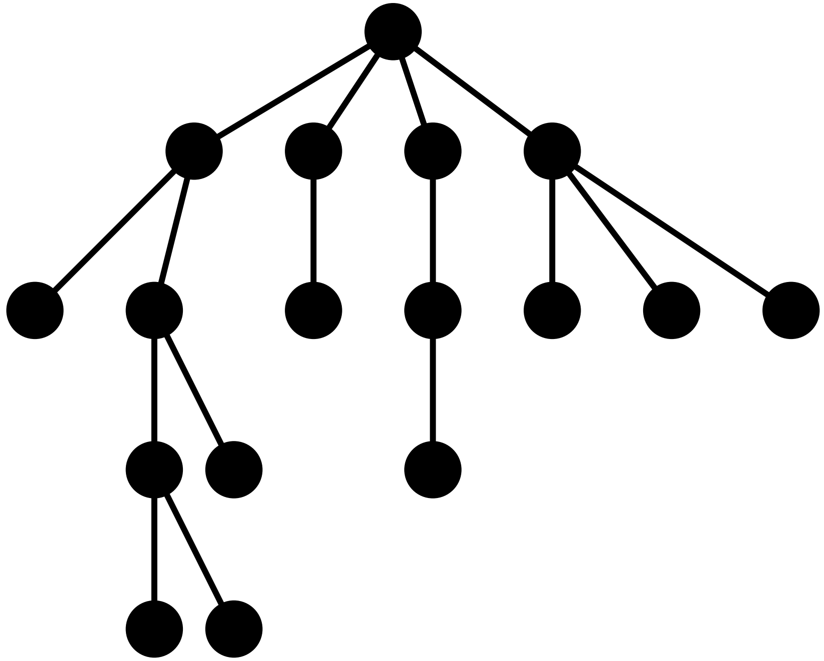
For every fixed k, it can be checked in polynomial-time if treewidth is at most k.

Exercise 1: Show that the treewidth of the $k \times k$ grid is at least $k-1$.

Exercise 2: Show that the treewidth of the $k \times k$ grid is at least k.

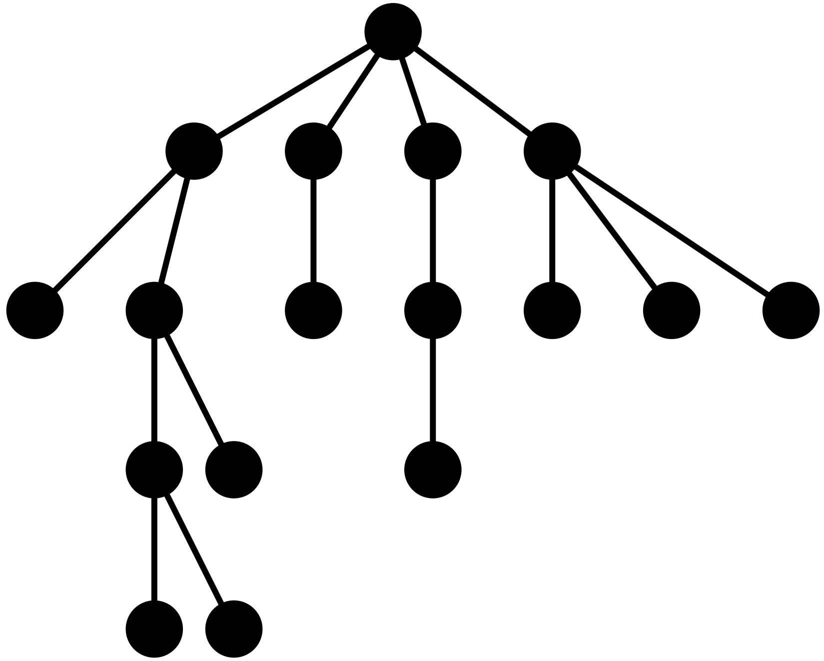
The Robber and Cops game (cont.)

Example: 2 cops have a winning strategy in a tree.



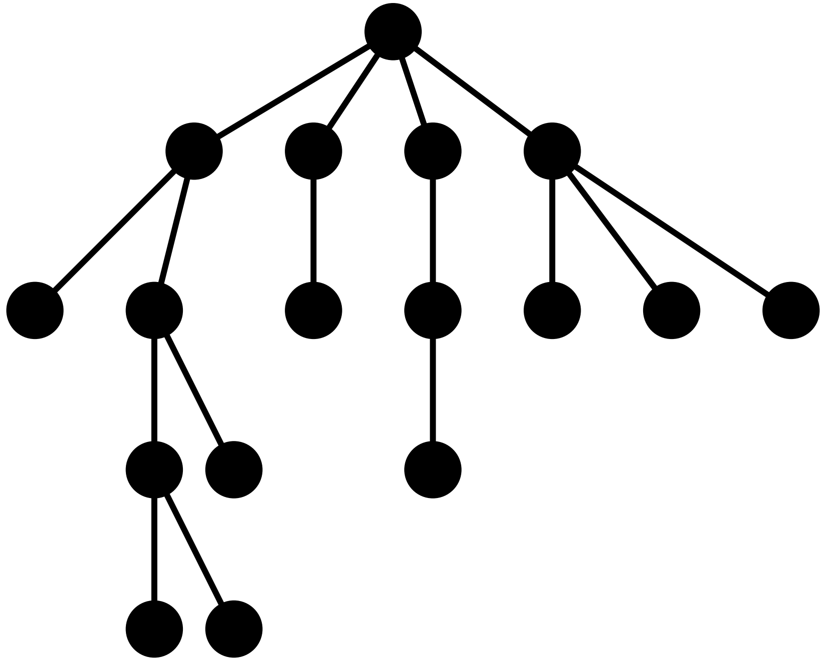
The Robber and Cops game (cont.)

Example: 2 cops have a winning strategy in a tree.



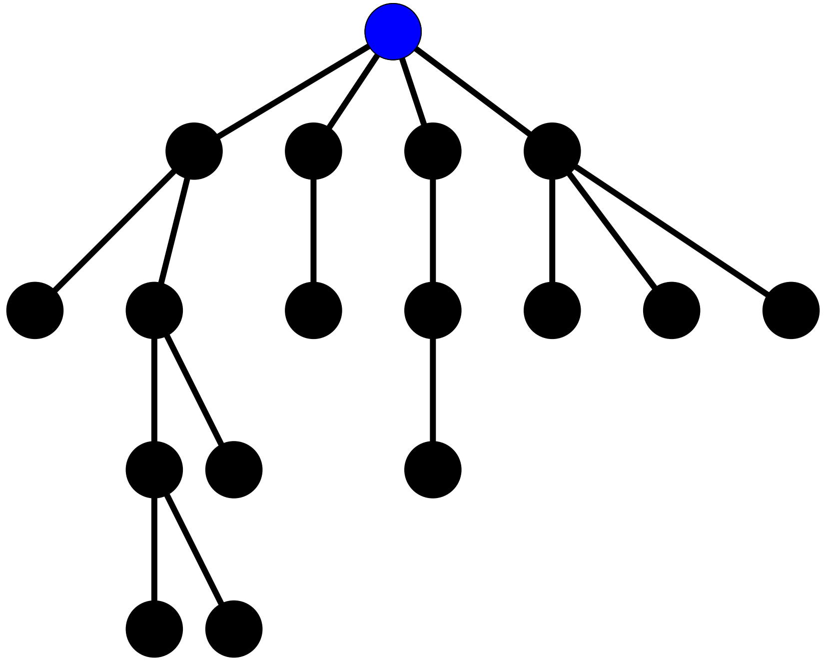
The Robber and Cops game (cont.)

Example: 2 cops have a winning strategy in a tree.



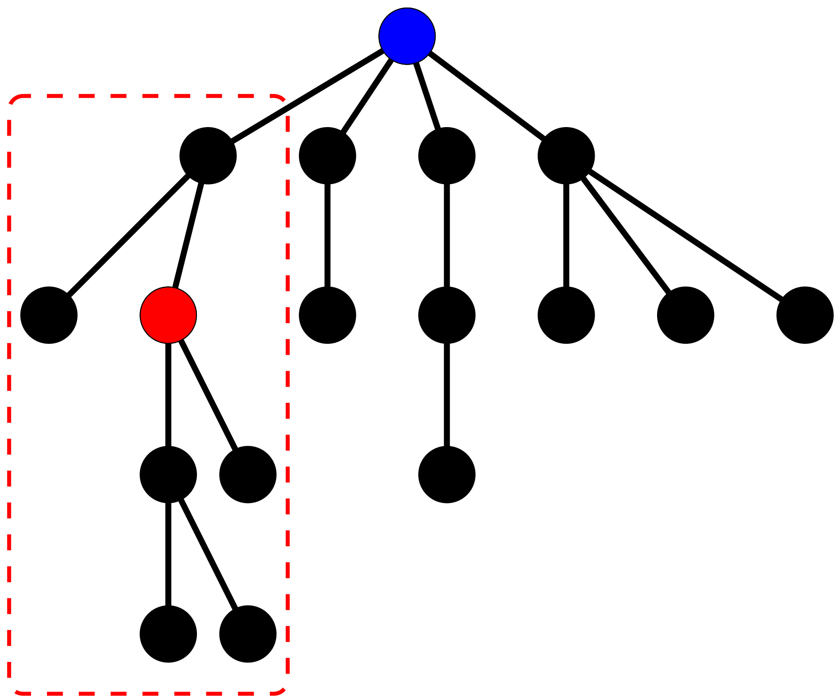
The Robber and Cops game (cont.)

Example: 2 cops have a winning strategy in a tree.



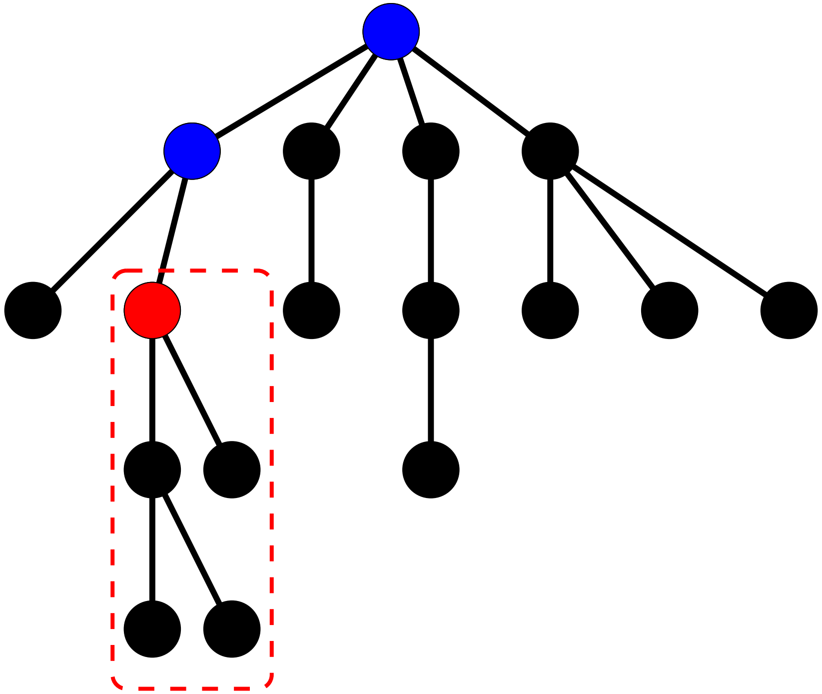
The Robber and Cops game (cont.)

Example: 2 cops have a winning strategy in a tree.



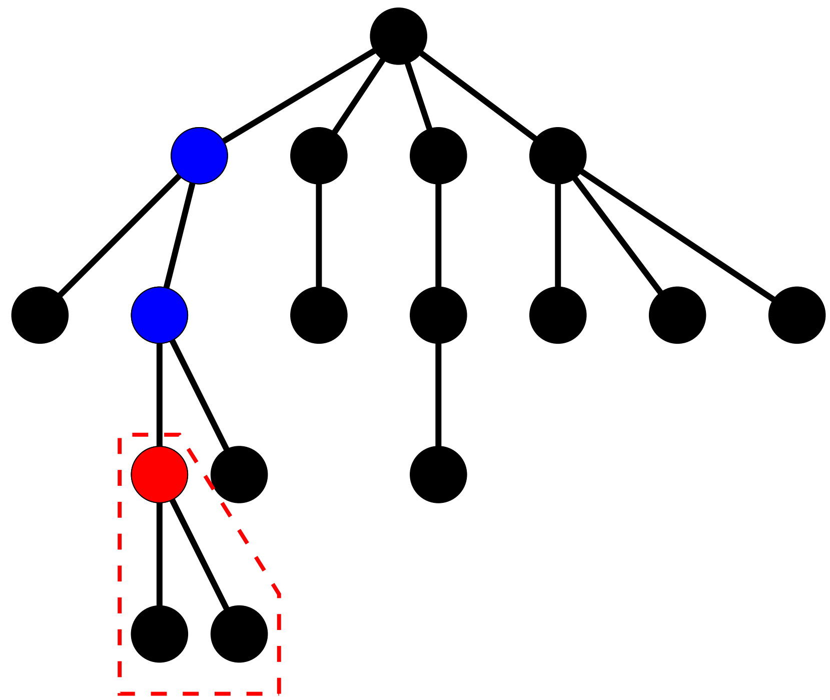
The Robber and Cops game (cont.)

Example: 2 cops have a winning strategy in a tree.



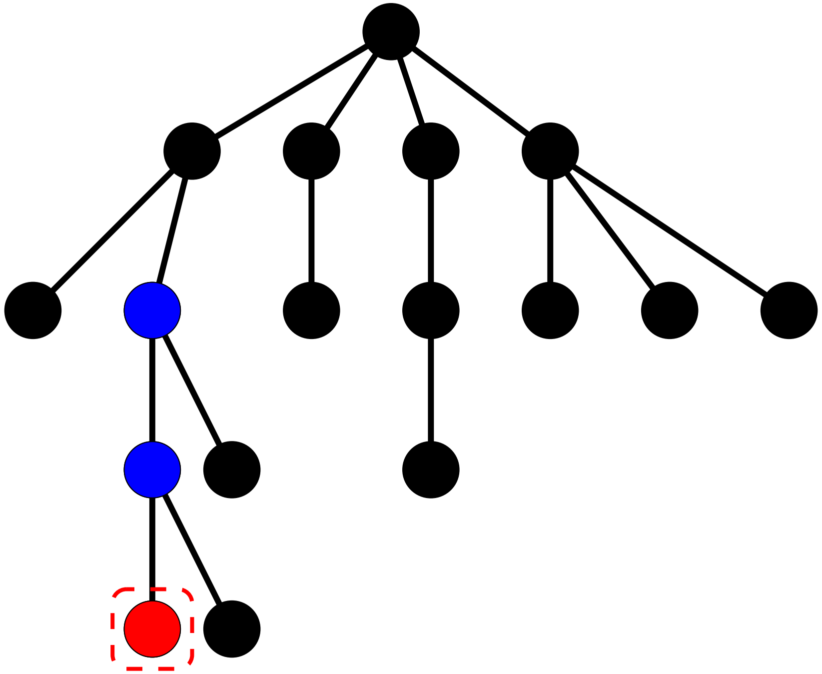
The Robber and Cops game (cont.)

Example: 2 cops have a winning strategy in a tree.



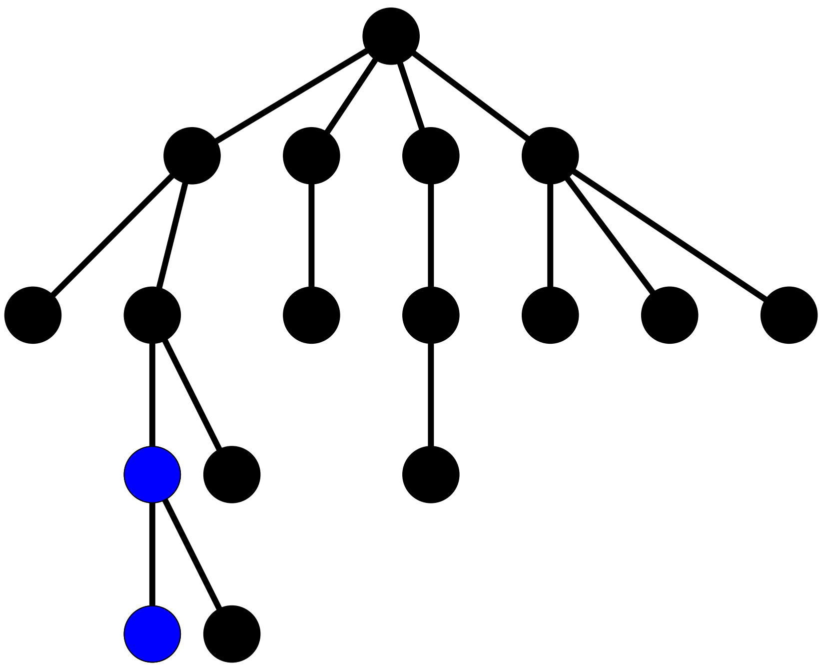
The Robber and Cops game (cont.)

Example: 2 cops have a winning strategy in a tree.



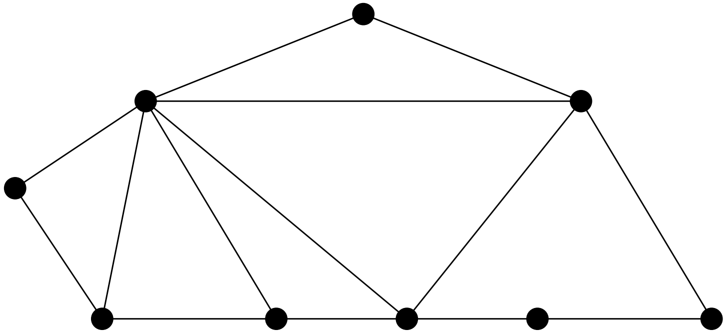
The Robber and Cops game (cont.)

Example: 2 cops have a winning strategy in a tree.



Outerplanar graphs

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



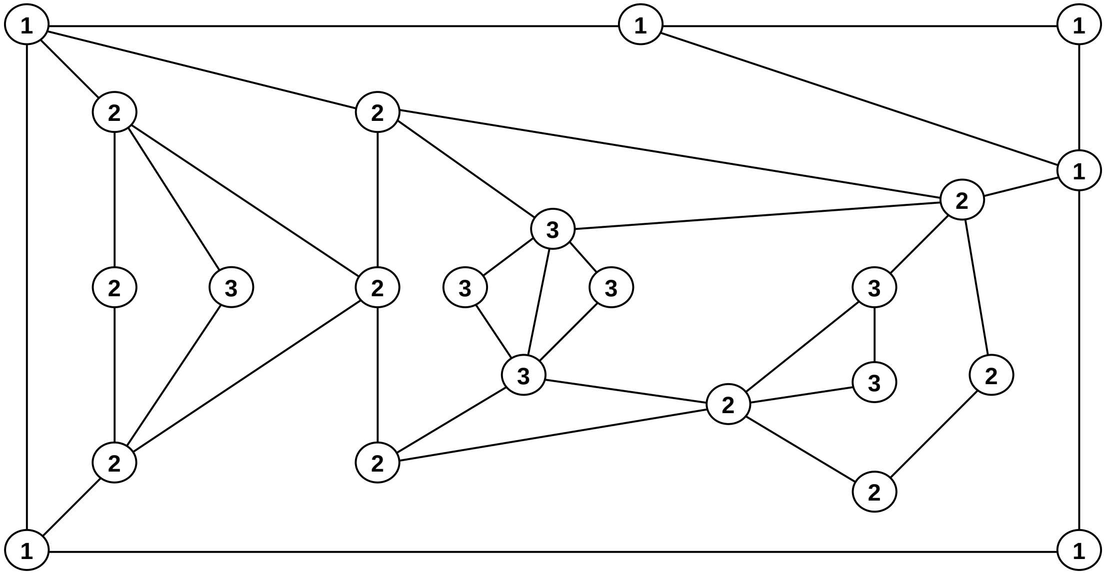
Fact: Every outerplanar graph has treewidth at most 2.

$\Rightarrow$ Every outerplanar graph is series-parallel.

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.

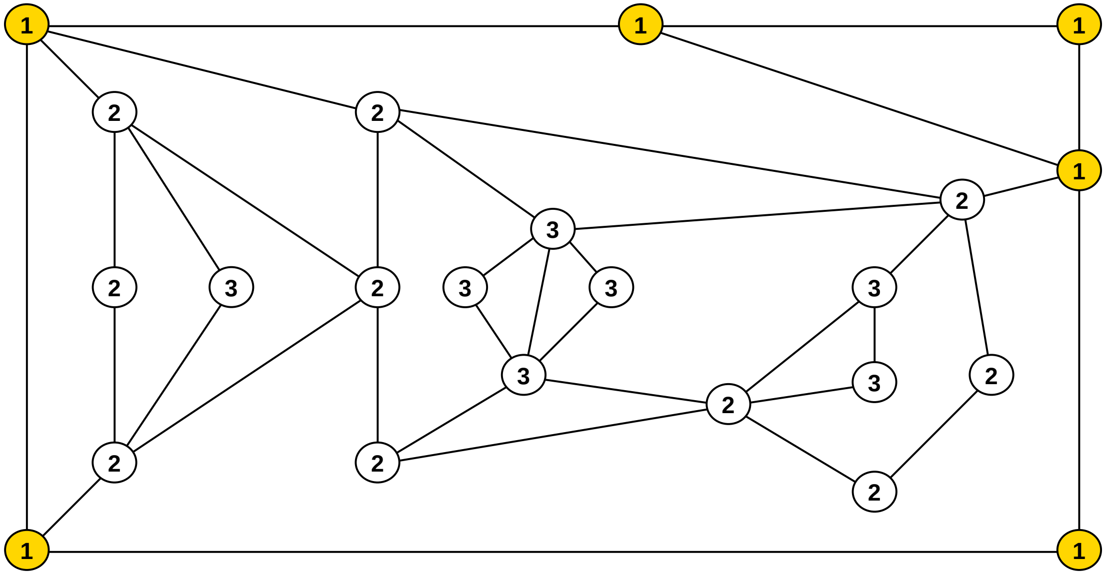


Fact: Every k-outerplanar graph has treewidth at most $3k + 1$.

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.

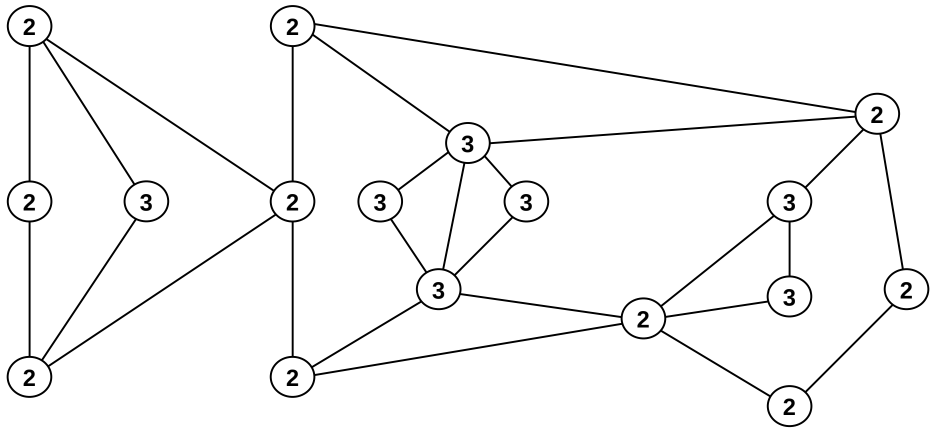


Fact: Every k-outerplanar graph has treewidth at most $3k + 1$.

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.

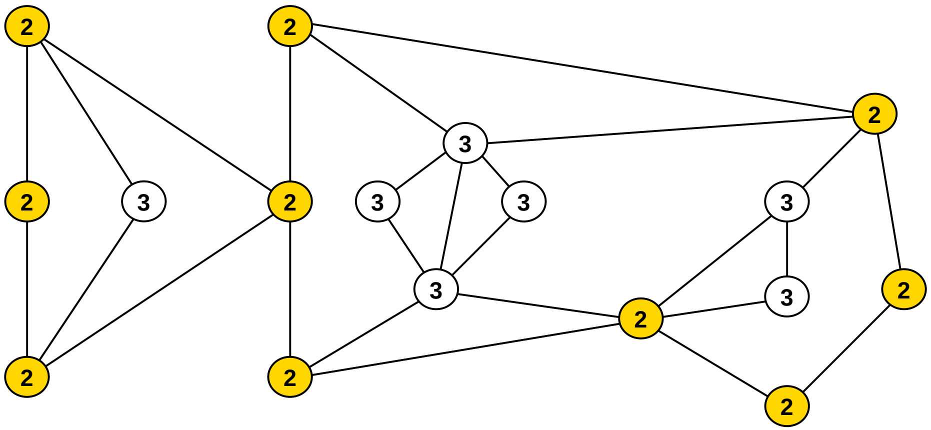


Fact: Every k-outerplanar graph has treewidth at most $3k + 1$.

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.

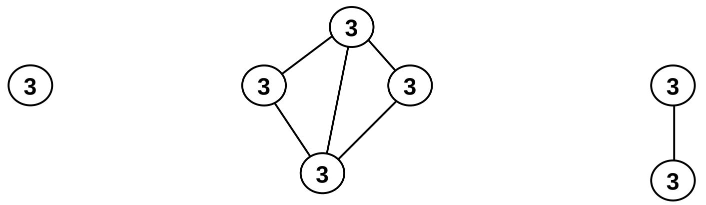


Fact: Every k-outerplanar graph has treewidth at most $3k + 1$.

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 $3k + 1$.

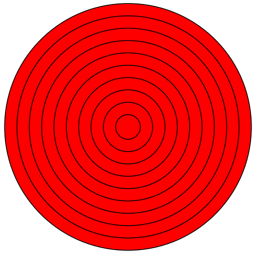
Part III:

Applications

Baker's shifting strategy

SUBGRAPH ISOMORPHISM for planar graphs: given planar graphs H and G, find a copy of H in G as subgraph. Parameter $k := |V(H)|$.

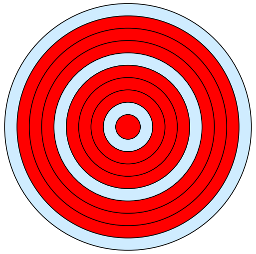
Layers of the planar graph: (as in the definition of k-outerplanar):



Baker's shifting strategy

SUBGRAPH ISOMORPHISM for planar graphs: given planar graphs H and G, find a copy of H in G as subgraph. Parameter $k := |V(H)|$.

Layers of the planar graph:
(as in the definition of
k-outerplanar):

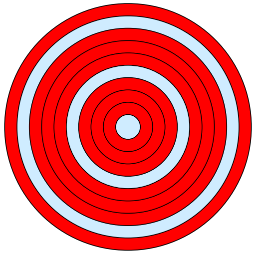


- For a fixed $0 \leq s < k + 1$, delete every layer L_i with $i = s \pmod{k+1}$

Baker's shifting strategy

SUBGRAPH ISOMORPHISM for planar graphs: given planar graphs H and G, find a copy of H in G as subgraph. Parameter $k := |V(H)|$.

Layers of the planar graph:
(as in the definition of
k-outerplanar):

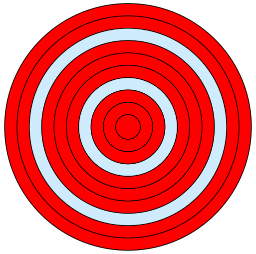


- For a fixed $0 \leq s < k+1$, delete every layer L_i with $i = s \pmod{k+1}$

Baker's shifting strategy

SUBGRAPH ISOMORPHISM for planar graphs: given planar graphs H and G, find a copy of H in G as subgraph. Parameter $k := |V(H)|$.

Layers of the planar graph: (as in the definition of k-outerplanar):

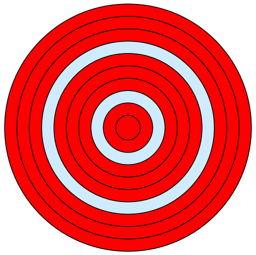


- For a fixed $0 \leq s < k+1$, delete every layer L_i with $i = s \pmod{k+1}$

Baker's shifting strategy

SUBGRAPH ISOMORPHISM for planar graphs: given planar graphs H and G, find a copy of H in G as subgraph. Parameter $k := |V(H)|$.

Layers of the planar graph:
(as in the definition of
k-outerplanar):

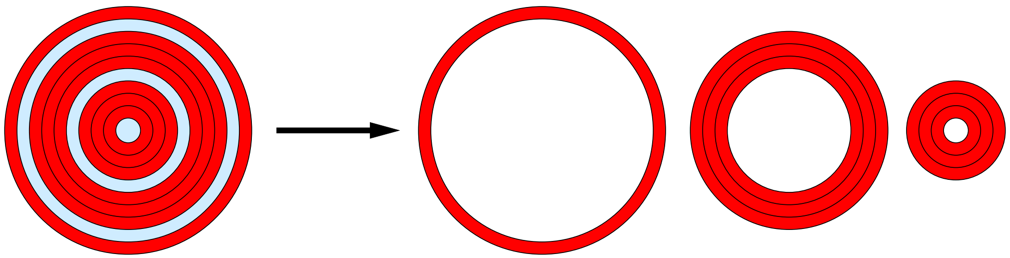


- For a fixed $0 \leq s < k+1$, delete every layer L_i with $i = s \pmod{k+1}$

Baker's shifting strategy

SUBGRAPH ISOMORPHISM for planar graphs: given planar graphs H and G, find a copy of H in G as subgraph. Parameter $k := |V(H)|$.

Layers of the planar graph: (as in the definition of k-outerplanar):

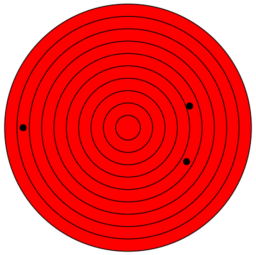


- For a fixed $0 \leq s < k+1$, delete every layer L_i with $i = s \pmod{k+1}$
- The resulting graph is k-outerplanar, hence it has treewidth at most $3k+1$.
- Using the $f(k,w) \cdot n$ time algorithm for SUBGRAPH ISOMORPHISM, the problem can be solved in time $f(k, 3k+1) \cdot n$.

Baker's shifting strategy

SUBGRAPH ISOMORPHISM for planar graphs: given planar graphs H and G, find a copy of H in G as subgraph. Parameter $k := |V(H)|$.

Layers of the planar graph: (as in the definition of k-outerplanar):

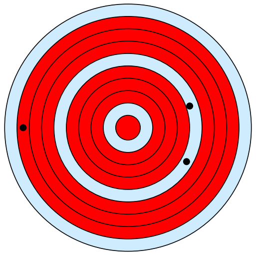


- For a fixed $0 \leq s < k+1$, delete every layer L_i with $i = s \pmod{k+1}$
- The resulting graph is k-outerplanar, hence it has treewidth at most $3k+1$.
- Using the $f(k, w) \cdot n$ time algorithm for SUBGRAPH ISOMORPHISM, the problem can be solved in time $f(k, 3k+1) \cdot n$.
- We do this for 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 $\Rightarrow$ we find a copy of H in G if there is one.
- SUBGRAPH ISOMORPHISM for planar graphs is FPT parameterized by $k := |V(H)|$.

Baker's shifting strategy

SUBGRAPH ISOMORPHISM for planar graphs: given planar graphs H and G, find a copy of H in G as subgraph. Parameter $k := |V(H)|$.

Layers of the planar graph:
(as in the definition of
k-outerplanar):

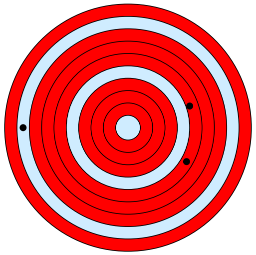


- For a fixed $0 \le s < k+1$, delete every layer L_i with $i = s \pmod{k+1}$
- The resulting graph is k-outerplanar, hence it has treewidth at most $3k+1$.
- Using the $f(k,w) \cdot n$ time algorithm for SUBGRAPH ISOMORPHISM, the problem can be solved in time $f(k, 3k+1) \cdot n$.
- We do this for every $0 \le s < k+1$: for at least one value of s, we do not delete any of the k vertices of the solution $\Rightarrow$ we find a copy of H in G if there is one.
- SUBGRAPH ISOMORPHISM for planar graphs is FPT parameterized by $k := |V(H)|$.

Baker's shifting strategy

SUBGRAPH ISOMORPHISM for planar graphs: given planar graphs H and G, find a copy of H in G as subgraph. Parameter $k := |V(H)|$.

Layers of the planar graph:
(as in the definition of
k-outerplanar):

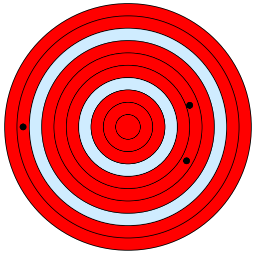


- For a fixed $0 \le s < k+1$, delete every layer L_i with $i = s \pmod{k+1}$
- The resulting graph is k-outerplanar, hence it has treewidth at most $3k+1$.
- Using the $f(k,w) \cdot n$ time algorithm for SUBGRAPH ISOMORPHISM, the problem can be solved in time $f(k, 3k+1) \cdot n$.
- We do this for every $0 \le s < k+1$: for at least one value of s, we do not delete any of the k vertices of the solution $\Rightarrow$ we find a copy of H in G if there is one.
- SUBGRAPH ISOMORPHISM for planar graphs is FPT parameterized by $k := |V(H)|$.

Baker's shifting strategy

SUBGRAPH ISOMORPHISM for planar graphs: given planar graphs H and G, find a copy of H in G as subgraph. Parameter $k := |V(H)|$.

Layers of the planar graph:
(as in the definition of
k-outerplanar):



- For a fixed $0 \le s < k+1$, 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 $f(k, w) \cdot n$ time algorithm for SUBGRAPH ISOMORPHISM, the problem can be solved in time $f(k, 3k+1) \cdot n$.
- We do this for every $0 \le s < k+1$: for at least one value of s, we do not delete any of the k vertices of the solution $\Rightarrow$ we find a copy of H in G if there is one.
- SUBGRAPH ISOMORPHISM for planar graphs is FPT parameterized by $k := |V(H)|$.

Detour to approximation...

Detour to approximation algorithms

Definition: A c**-approximation** algorithm for a maximization problem is a polynomial-time algorithm that finds a solution of cost at least OPT/c.

Definition: A c**-approximation** algorithm for a minimization problem is a polynomial-time algorithm that finds a solution of cost at most $\mathrm{OPT} \cdot c$.

There are well-known approximation algorithms for NP-hard problems: $\frac{3}{2}$-approximation for METRIC TSP, 2-approximation for VERTEX COVER, $\frac{8}{7}$-approximation for MAX 3SAT, etc.

Detour to approximation algorithms

Definition: A c**-approximation** algorithm for a maximization problem is a polynomial-time algorithm that finds a solution of cost at least OPT$/c$.

Definition: A c**-approximation** algorithm for a minimization problem is a polynomial-time algorithm that finds a solution of cost at most OPT $\cdot\, c$.

There are well-known approximation algorithms for NP-hard problems: $\frac{3}{2}$-approximation for METRIC TSP, 2-approximation for VERTEX COVER, $\frac{8}{7}$-approximation for MAX 3SAT, etc.

- For some problems, we have lower bounds: there is no $(2-\epsilon)$-approximation for VERTEX COVER or $(\frac{8}{7}-\epsilon)$-approximation for MAX 3SAT (under suitable complexity assumptions).
- For some other problems, arbitrarily good approximation is possible in polynomial time: for any $c > 1$ (say, $c = 1.000001$), there is a polynomial-time c-approximation algorithm!

Approximation schemes

Definition: A **polynomial-time approximation scheme (PTAS)** for a problem *P* is an algorithm that takes an instance of *P* and a rational number $\epsilon > 0$,

- always finds a $(1+\epsilon)$-approximate solution,
- the running time is polynomial in *n* for every fixed $\epsilon > 0$.

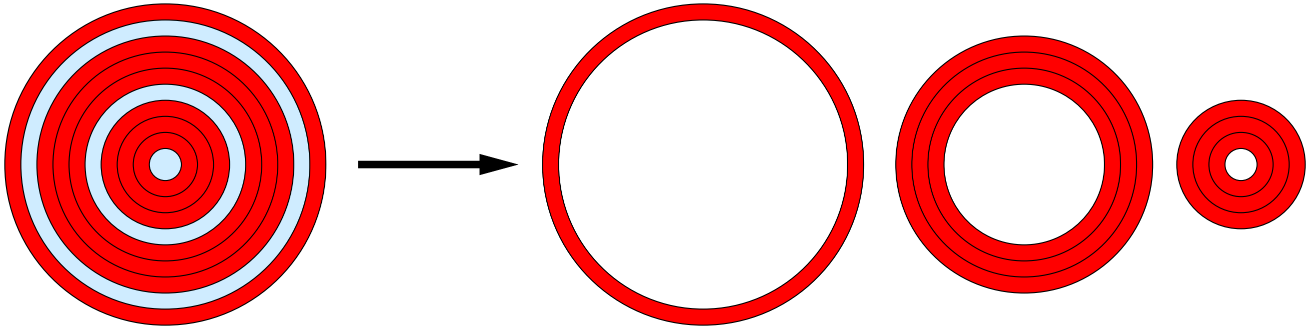
Typical running times: $2^{1/\epsilon} \cdot n$, $n^{1/\epsilon}$, $(n/\epsilon)^2$, n^{1/ϵ^2}.

Some classical problems that have a PTAS:

- INDEPENDENT SET for planar graphs
- TSP in the Euclidean plane
- STEINER TREE in planar graphs
- KNAPSACK

Baker's shifting strategy for EPTAS

Fact: There is a $2^{O(1/\epsilon)} \cdot n$ time PTAS for INDEPENDENT SET for planar graphs.



- Let $D := 1/\epsilon$. For a fixed $0 \leq s < D$, delete every layer L_i with $i = s \pmod D$
- The resulting graph is D-outerplanar, hence it has treewidth at most $3D + 1 = O(1/\epsilon)$.
- Using the $O(2^w \cdot n)$ time algorithm for INDEPENDENT SET, the problem can be solved in time $2^{O(1/\epsilon)} \cdot n$.
- We do this for every $0 \leq s < D$: for at least one value of s, we delete at most $1/D = \epsilon$ fraction of the solution $\Rightarrow$ we get a $(1 + \epsilon)$-approximate solution.

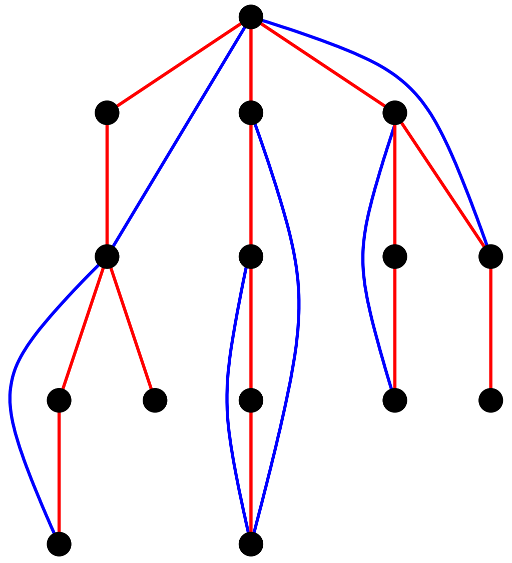
Back to FPT...

Depth-first search (DFS)

Fact: Finding a cycle of length **at least** k in a graph is FPT parameterized by k.

Let us start a depth-first search from an arbitrary vertex v. There are two types of edges: tree edges and back edges.

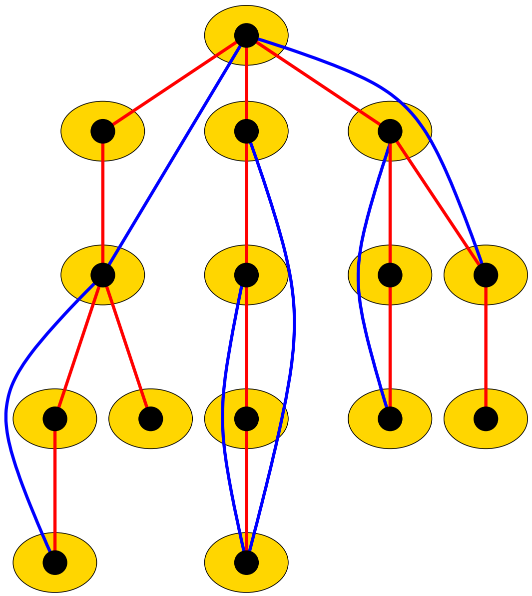
- If there is a back edge whose endpoints differ by at least $k-1$ levels $\Rightarrow$ there is a cycle of length at least k.
- Otherwise, the graph has treewidth at most $k-2$ and we can solve the problem by applying Courcelle's Theorem.



Depth-first search (DFS)

Fact: Finding a cycle of length **at least** k in a graph is FPT parameterized by k.

Let us start a depth-first search from an arbitrary vertex v. There are two types of edges: tree edges and back edges.



- If there is a back edge whose endpoints differ by at least $k-1$ levels $\Rightarrow$ there is a cycle of length at least k.
- Otherwise, the graph has treewidth at most $k-2$ and we can solve the problem by applying Courcelle's Theorem.

In the second case, a tree decomposition can be easily found: the decomposition has the same structure as the DFS spanning tree and each bag contains the vertex and its $k-2$ ancestors.

Bidimensionality

A powerful framework to obtain efficient algorithms on planar graphs.

Let $x(G)$ be some graph invariant (i.e., an integer associated with each graph). Some typical examples:

- Maximum independent set size.
- Minimum vertex cover size.
- Length of the longest path.
- Minimum dominating set size
- Minimum feedback vertex set size.

Given G and k, we want to decide if $x(G) \leq k$ (or $x(G) \geq k$).

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

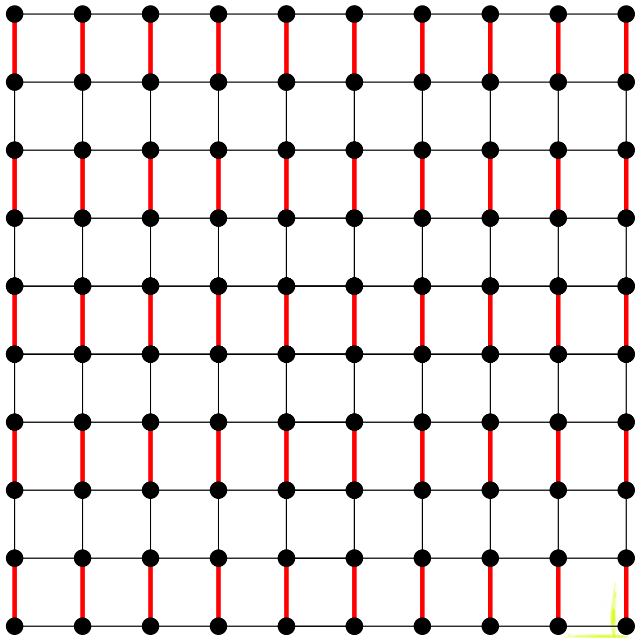
Bidimensionality for VERTEX COVER

Observation: If the treewidth of a planar graph G is at least $4\sqrt{2k}$

$\Rightarrow$ It contains a $\sqrt{2k} \times \sqrt{2k}$ grid minor (Excluded Grid Theorem for planar graphs)

$\Rightarrow$ The vertex cover size of the grid is at least k in the grid

$\Rightarrow$ Vertex cover size is at least k in G.



Bidimensionality for VERTEX COVER

Observation: If the treewidth of a planar graph G is at least $4\sqrt{2k}$

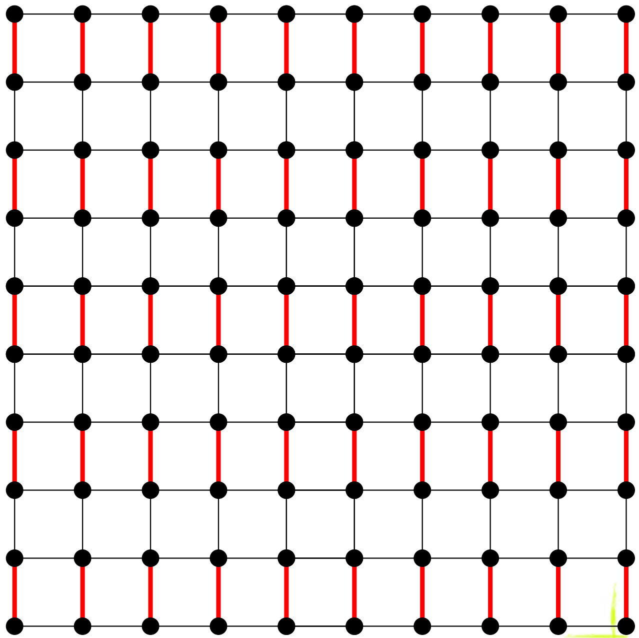
$\Rightarrow$ It contains a $\sqrt{2k} \times \sqrt{2k}$ grid minor (Excluded Grid Theorem for planar graphs)

$\Rightarrow$ The vertex cover size of the grid 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 as follows:

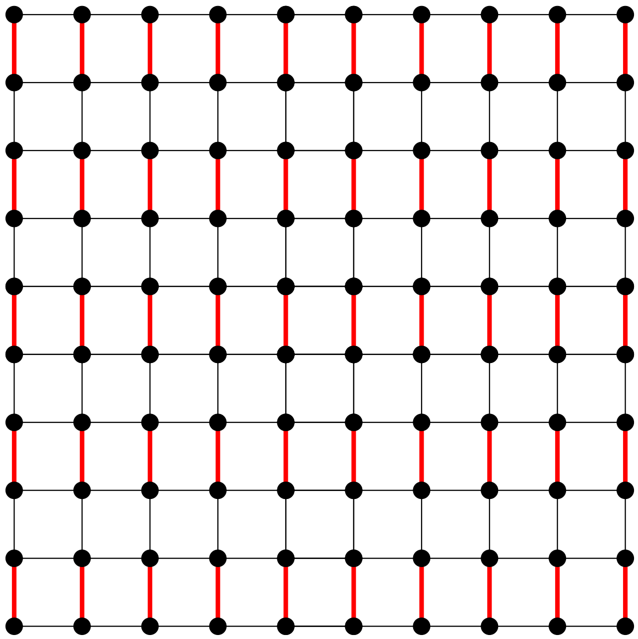
- Set $w := 4\sqrt{2k}$.
- Use the 4-approximation tree decomposition algorithm ($2^{O(w)} \cdot n^{O(1)} = 2^{O(\sqrt{k})} \cdot n^{O(1)}$ time).
  - If treewidth is at least w: we answer 'vertex cover is $\geq k$'.
  - If we get a tree decomposition of width $4w$, then we can solve the problem in time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



Bidimensionality (cont.)

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 $c > 0$).

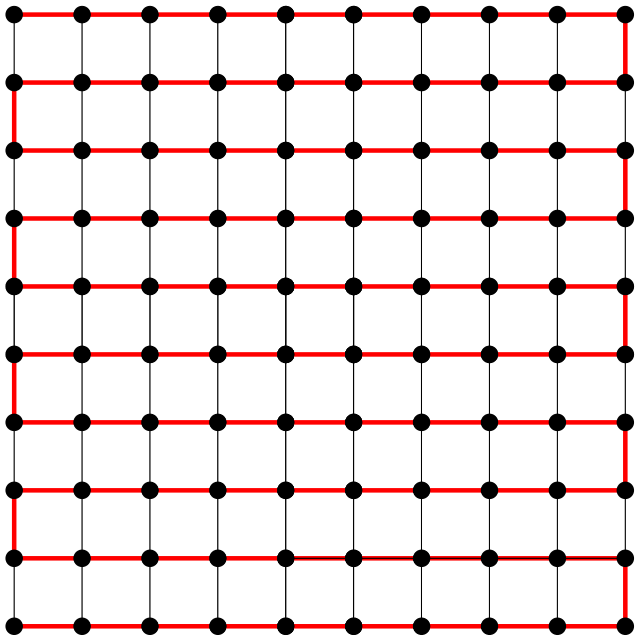


Examples: minimum vertex cover, length of the longest path, feedback vertex set are minor-bidimensional.

Bidimensionality (cont.)

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 $c > 0$).

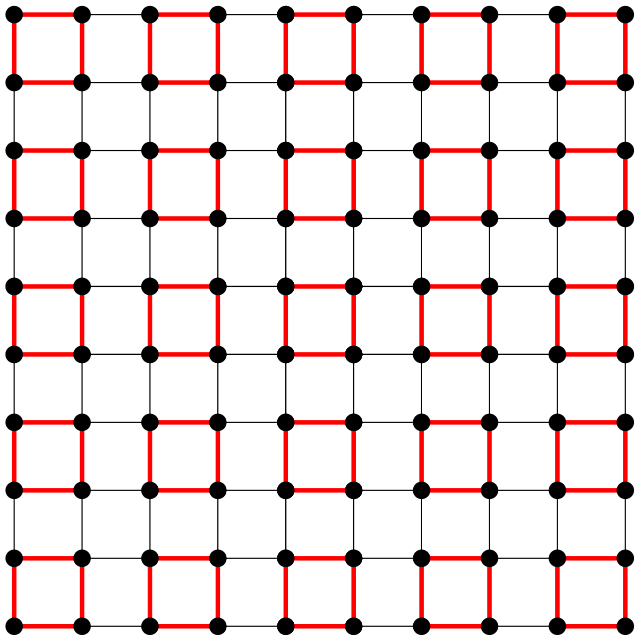


Examples: minimum vertex cover, length of the longest path, feedback vertex set are minor-bidimensional.

Bidimensionality (cont.)

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 parameter 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 using a width w tree decomposition 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 using a width w tree decomposition 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

Problem: DOMINATING SET is **not** minor-bidimensional (why?).

Contraction bidimensionality

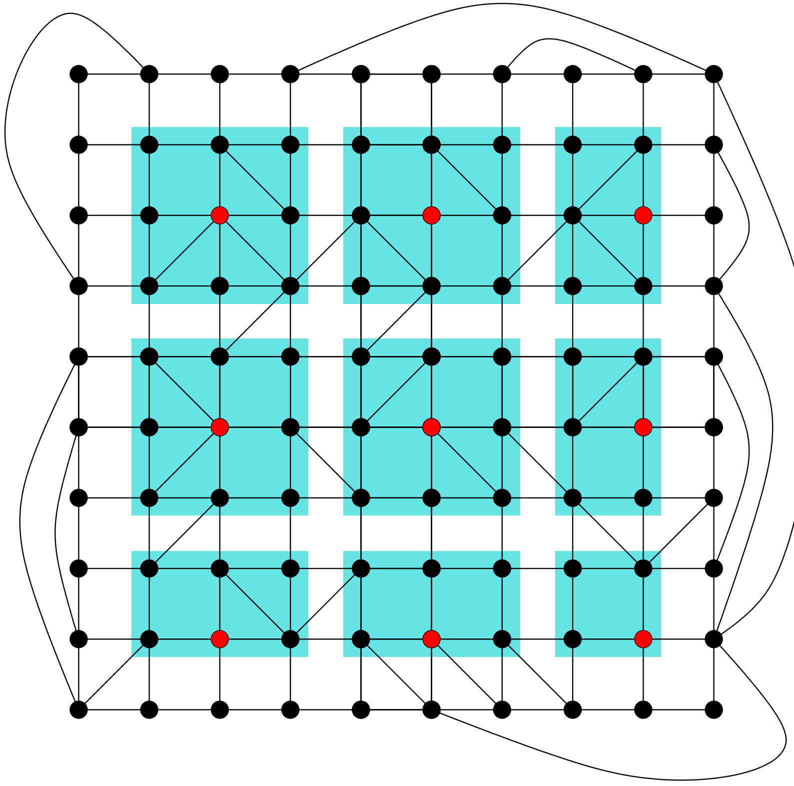
Problem: DOMINATING SET is **not** minor-bidimensional (why?).

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

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 constant $c > 0$).

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



Contraction bidimensionality

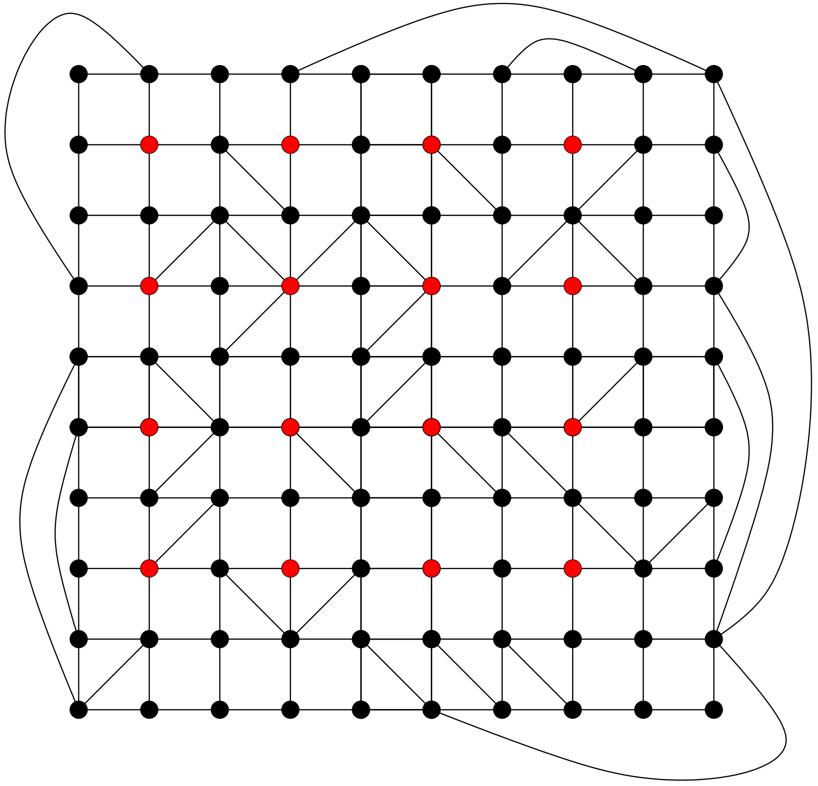
Problem: DOMINATING SET is **not** minor-bidimensional (why?).

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

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 constant $c > 0$).

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).

We use this observation to solve DOMINATING SET on planar graphs as follows:

- Set $w := 3\sqrt{k} + 2$.
- 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 $3^w \cdot n^{O(1)} = 2^{O(\sqrt{k})} \cdot n^{O(1)}$.

Fact: Given a tree decomposition of width w, DOMINATING SET can be solved in time $O^*(3^w)$.

Exercise: Given a tree decomposition of width w, DOMINATING SET can be solved in time $O^*(4^w)$.

Summary

- Notion of treewidth allows us to generalize dynamic programming on trees to more general graphs.
- Standard techniques for designing algorithms on bounded treewidth graphs: dynamic programming and Courcelle's Theorem.
- Surprising uses of treewidth in other contexts (planar graphs).

Tomorrow: Bad news. Complexity results. Which problems are not FPT?