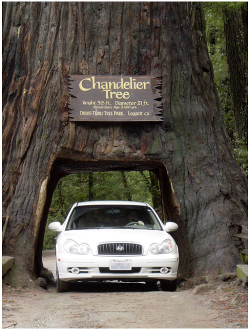
Treewidth

Dániel Marx



Recent Advances in Parameterized Complexity
Tel Aviv, Israel, December 3-7, 2017

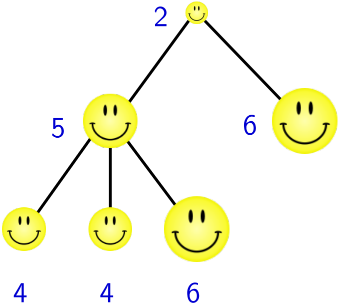
Treewidth

- 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

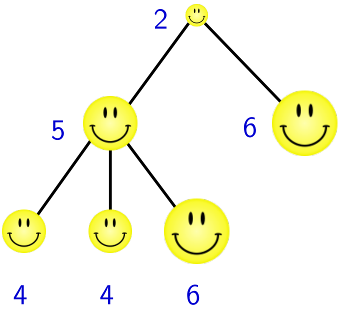
Problem: Invite some colleagues for a party.
Maximize: The total fun factor of the invited people.
Constraint: Everyone should be having fun.



The Party Problem

Party Problem

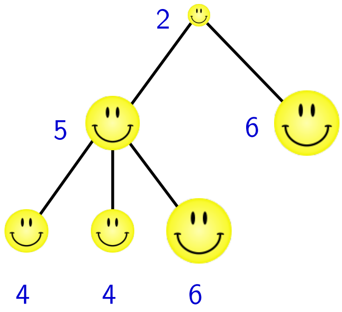
Problem:	Invite some colleagues for a party.
Maximize:	The total fun factor of the invited people.
Constraint:	Everyone should be having fun.
	Do not invite a colleague and his direct boss at the same time!



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.

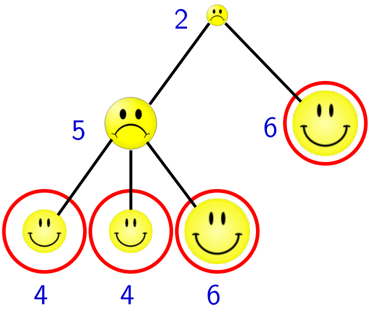
The Party Problem

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.

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).

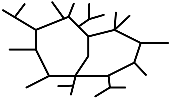
Generalizing trees

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

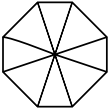
Generalizing trees

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

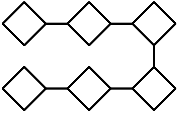
1. Number of cycles is bounded.



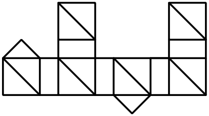
good



bad



bad

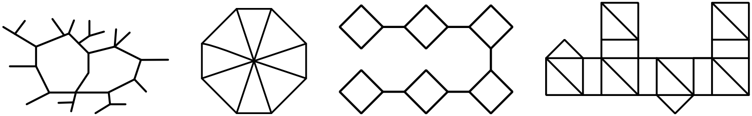


bad

Generalizing trees

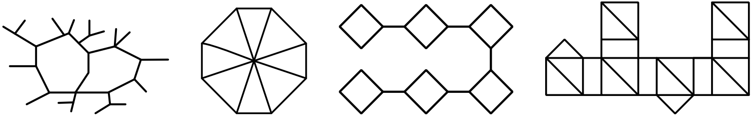
How could we define that a graph is "treelike"?

1. Number of cycles is bounded.



good bad bad bad

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

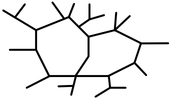


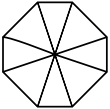
good good bad bad

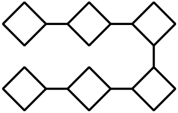
Generalizing trees

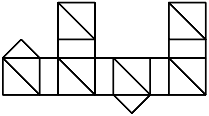
How could we define that a graph is "treelike"?

1. Number of cycles is bounded.

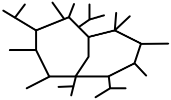

good

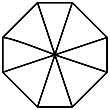

bad

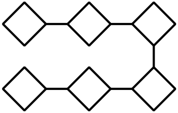

bad

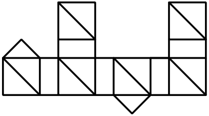

bad

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

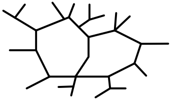

good

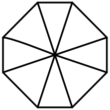

good

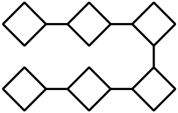

bad

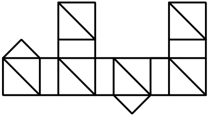

bad

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


bad


bad

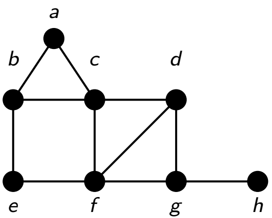

good

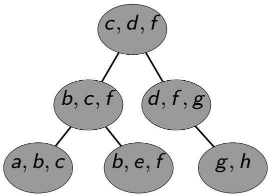

good

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.

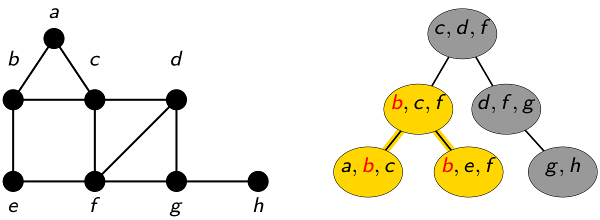




Treewidth — a measure of “tree-likeness”

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

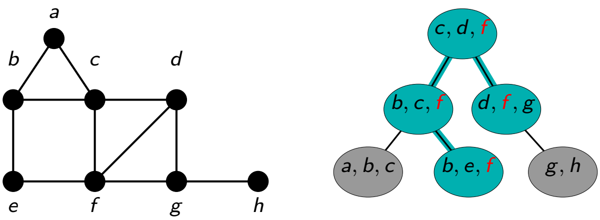
1. If u and v are neighbors, then there is a bag containing both of them.
2. For every v, the bags containing v form a connected subtree.



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.



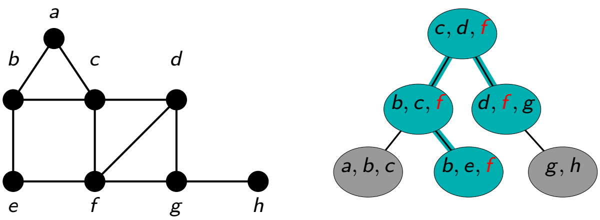
Treewidth — a measure of "tree-likeness"

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.



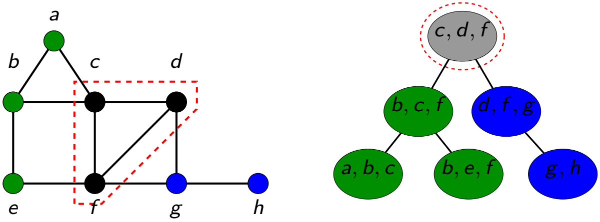
Treewidth — a measure of "tree-likeness"

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.

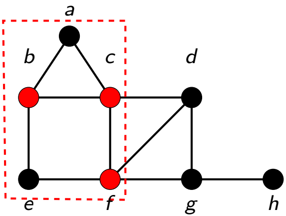
Treewidth — a measure of "tree-likeness"

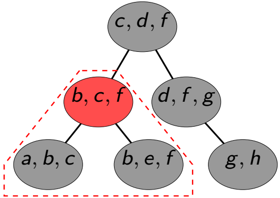
Tree decomposition: Vertices are arranged in a tree structure satisfying the following properties:

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.

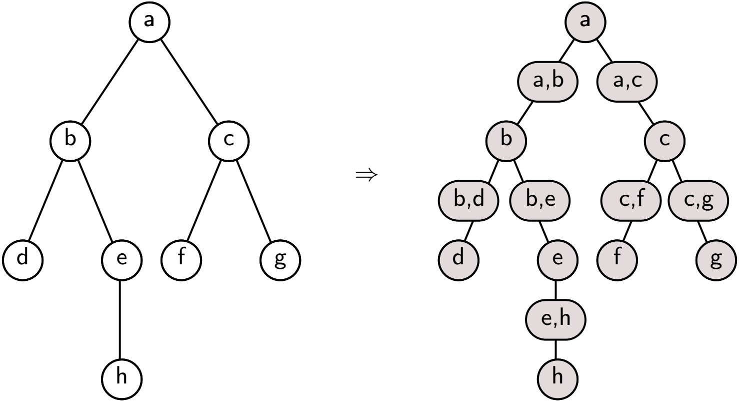




A subtree communicates with the outside world only via the root of the subtree.

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

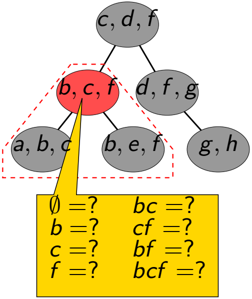
Given a tree decomposition of width w, Weighted Max Independent Set can be solved in time $O(2^w \cdot w^{O(1)} \cdot n)$.

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 $2^{|B_x|} \leq 2^{w+1}$ 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

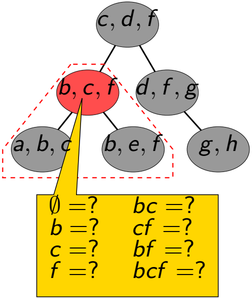
Theorem

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



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

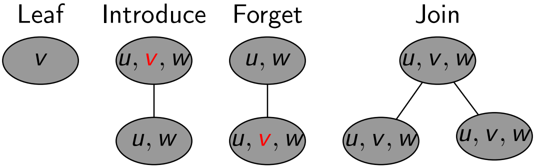
How to determine $M[x, S]$ if all the values are known for the children of x?

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 $B_x = B_{y_1} = B_{y_2}$

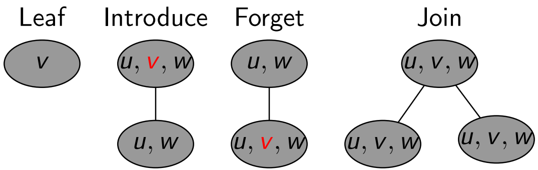
Theorem

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 $O(w^2n)$.

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

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



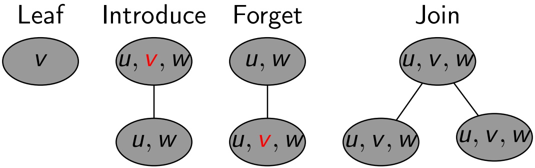
Weighted Max Independent Set
and nice tree decompositions

- **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 $O(2^w \cdot w^{O(1)} \cdot n)$.

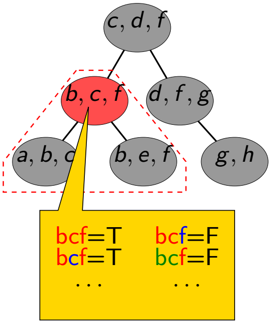
3-COLORING and tree decompositions

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.



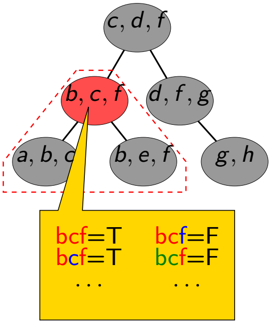
3-Coloring and tree decompositions

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.

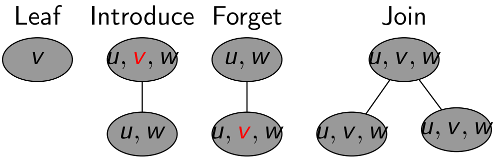
For every node x and coloring $c : B_x \to \{1, 2, 3\}$, we compute the Boolean value $E[x, c]$, which is true if and only if c can be extended to a proper 3-coloring of V_x.



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!
- **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.

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

$$\exists C \subseteq V \exists v_0 \in C \forall v \in C \; \exists u_1, u_2 \in C (u_1 \neq u_2 \wedge \text{adj}(u_1, v) \wedge \text{adj}(u_2, v))$$

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

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

$$\exists C \subseteq V \exists v_0 \in C \forall v \in C \; \exists u_1, u_2 \in C (u_1 \neq u_2 \wedge \text{adj}(u_1, v) \wedge \text{adj}(u_2, v))$$

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.

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

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

Using Courcelle's Theorem

3-Coloring

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

Hamiltonian Cycle

$\exists H \subseteq E \left(\text{spanning}(H) \wedge (\forall v \in V \,\text{degree2}(H, v))\right)$

$\text{degree0}(H, v) := \neg \exists e \in H \,\text{inc}(e, v)$

$\text{degree1}(H, v) := \neg\text{degree0}(H, v) \wedge \left(\neg \exists e_1, e_2 \in H \,(e_1 \neq e_2 \wedge \text{inc}(e_1, v) \wedge \text{inc}(e_2, v))\right)$

$\text{degree2}(H, v) := \neg\text{degree0}(H, v) \wedge \neg\text{degree1}(H, v) \wedge \left(\neg \exists e_1, e_2, e_3 \in H \,(e_1 \neq e_2 \wedge e_2 \neq e_3 \wedge e_1 \neq e_3 \wedge \text{inc}(e_1, v) \wedge \text{inc}(e_2, v) \wedge \text{inc}(e_3, v))\right)$

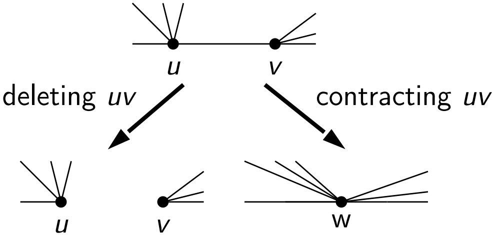
$\text{spanning}(H) := \forall u, v \in V \,\exists P \subseteq H \,\forall x \in V \left(((x = u \vee x = v) \wedge \text{degree1}(P, x)) \vee (x \neq u \wedge x \neq v \wedge (\text{degree0}(P, x) \vee \text{degree2}(P, x)))\right)$

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.



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

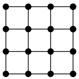
Fact: Treewidth does not increase if we delete edges, delete vertices, or contract edges.

$\Rightarrow$ 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 $k - 1$.

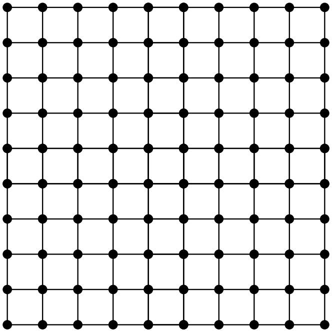
Fact: For every $k \geq 2$, the treewidth of the $k \times k$ grid is exactly k.



Excluded Grid Theorem

Excluded Grid Theorem [Diestel et al. 1999]

If the treewidth of G is at least $k^{4k^2(k+2)}$, then G has a $k \times k$ grid minor.



(A $k^{O(1)}$ bound was achieved recently [Chekuri and Chuznoy 2014]!)

Excluded Grid Theorem

Excluded Grid Theorem [Diestel et al. 1999]

If the treewidth of G is at least $k^{4k^2(k+2)}$, 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 $f(H)$.

Excluded Grid Theorem

Excluded Grid Theorem [Diestel et al. 1999]

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, but the relation is approximate:

No $k \times k$ grid minor $\Longrightarrow$ tree decomposition of width $< f(k)$

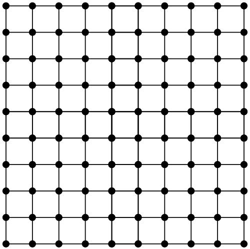
tree decomposition of width $< f(k)$ $\Longrightarrow$ no $f(k) \times f(k)$ grid minor

Planar Excluded Grid Theorem

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

Theorem [Robertson, Seymour, Thomas 1994]

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



Bidimensionality

A powerful framework for efficient algorithms on planar graphs.

Setup:

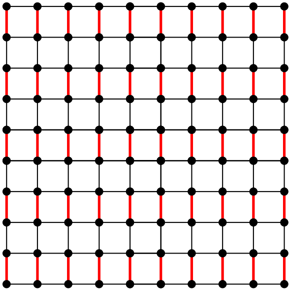
- Let $x(G)$ be some graph invariant (i.e., an integer associated with each graph).
- Given G and k, we want to decide if $x(G) \le k$ (or $x(G) \ge k$).
- Typical examples:
  - Maximum independent set size.
  - Minimum vertex cover size.
  - Length of the longest path.
  - Minimum dominating set size.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

Bidimensionality [Demaine, Fomin, Hajiaghayi, Thilikos 2005]

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

Bidimensionality for 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.

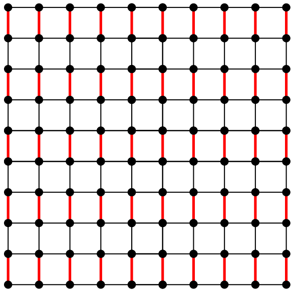


Bidimensionality for VERTEX COVER

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 $2^{O(w)} \cdot n^{O(1)} = 2^{O(\sqrt{k})} \cdot n^{O(1)}$.

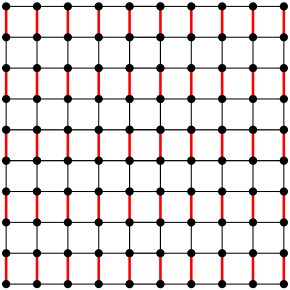


Bidimensionality

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 $c > 0$).



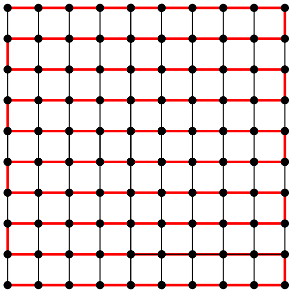
Examples: minimum vertex cover, length of the longest path, feedback vertex set are minor-bidimensional.

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 $c > 0$).



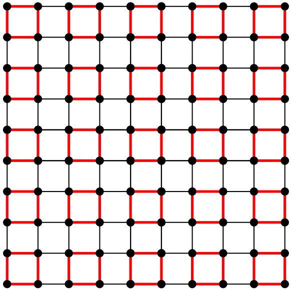
Examples: minimum vertex cover, length of the longest path, feedback vertex set are minor-bidimensional.

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 $2^{O(\sqrt{k} \log k)} \cdot n^{O(1)}$.

Treewidth

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.