The Square Root Phenomenon in Planar Graphs

Dániel Marx

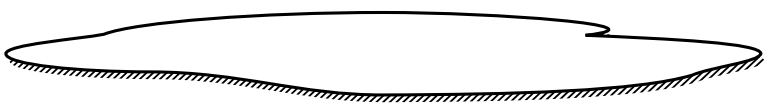
Institute for Computer Science and Control,
Hungarian Academy of Sciences (MTA SZTAKI)
Budapest, Hungary

Fine-Grained Complexity and Algorithms Course
UC Berkeley
November 12, 2015

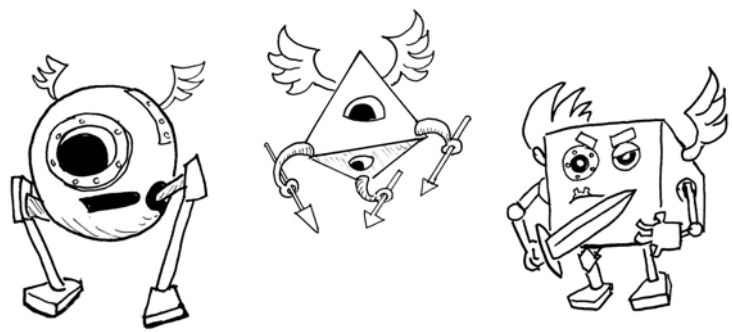
Main message

NP-hard problems become easier on planar graphs and geometric objects, and usually exactly by a square root factor.

Planar graphs



Geometric objects



Better exponential algorithms

Most NP-hard problems (e.g., 3-Coloring, Independent Set, Hamiltonian Cycle, Steiner Tree, etc.) remain NP-hard on planar graphs,[1] so what do we mean by "easier"?

[1] Notable exception: Max Cut is in P for planar graphs.

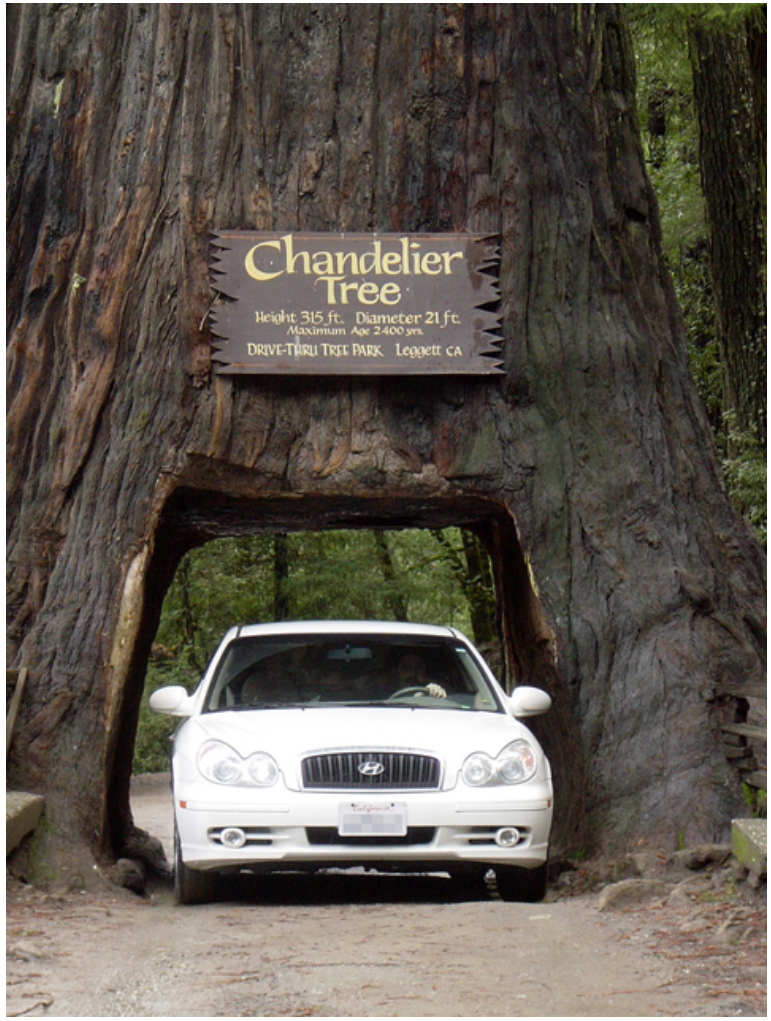
Better exponential algorithms

Most NP-hard problems (e.g., 3-Coloring, Independent Set, Hamiltonian Cycle, Steiner Tree, etc.) remain NP-hard on planar graphs,[1] so what do we mean by "easier"?

The running time is still exponential, but significantly smaller:

$$\begin{aligned} 2^{O(n)} &\Rightarrow 2^{O(\sqrt{n})} \\ n^{O(k)} &\Rightarrow n^{O(\sqrt{k})} \\ 2^{O(k)} \cdot n^{O(1)} &\Rightarrow 2^{O(\sqrt{k})} \cdot n^{O(1)} \end{aligned}$$

[1] Notable exception: Max Cut is in P for planar graphs.


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

Treewidth

Subexponential algorithms using treewidth

Treewidth is a measure of "how treelike the graph is."

We need only the following basic facts:

Treewidth

1. If a graph G has treewidth k, then many classical NP-hard problems can be solved in time $2^{O(k)} \cdot n^{O(1)}$ or $2^{O(k \log k)} \cdot n^{O(1)}$ on G.
2. A planar graph on n vertices has treewidth $O(\sqrt{n})$.

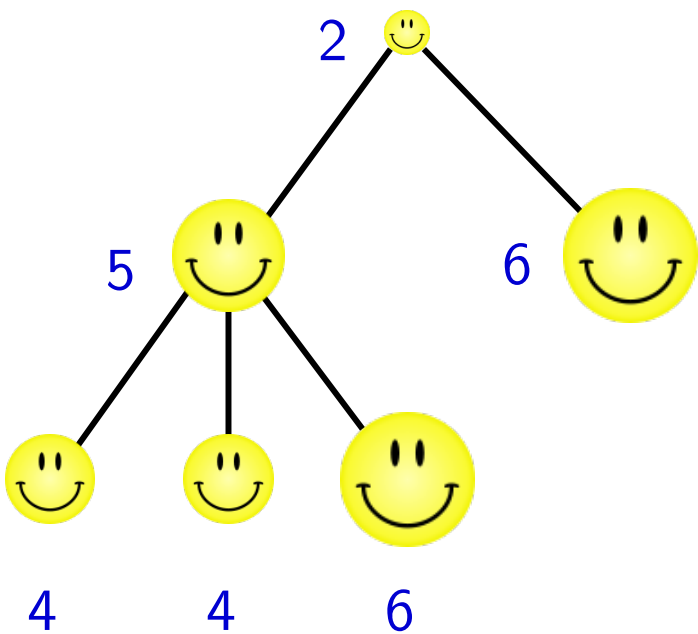
The Party Problem

Party Problem

Problem: Invite some colleagues for a party.

Maximize: The total fun factor of the invited people.

Constraint: Everyone should be having fun.



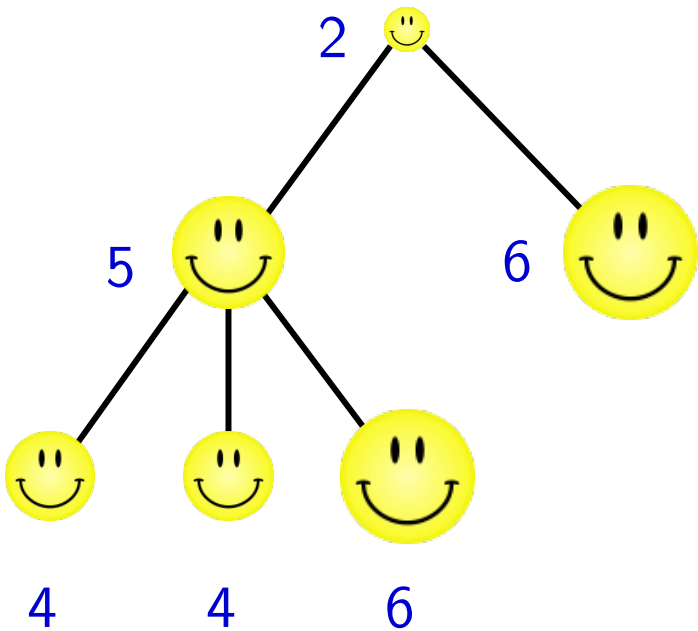
The Party Problem

Party Problem

Problem: Invite some colleagues for a party.

Maximize: The total fun factor of the invited people.

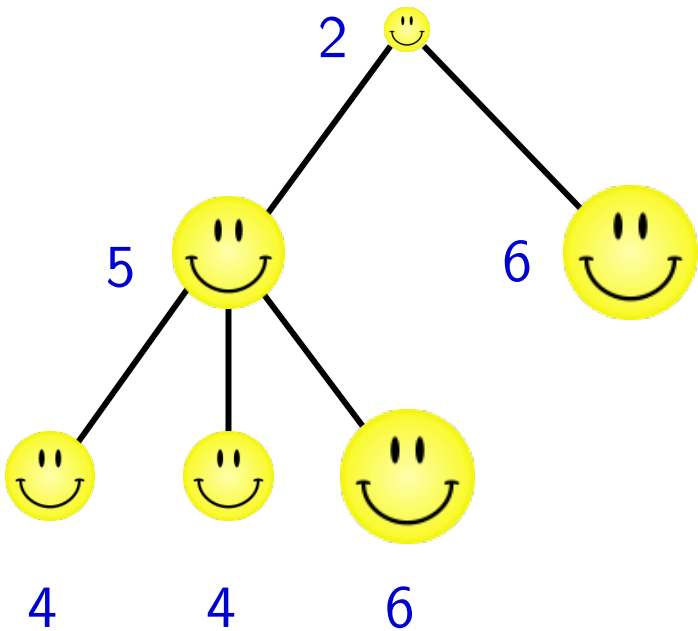
Constraint: Everyone should be having fun.
Do not invite a colleague and
his direct boss at the same time!



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!

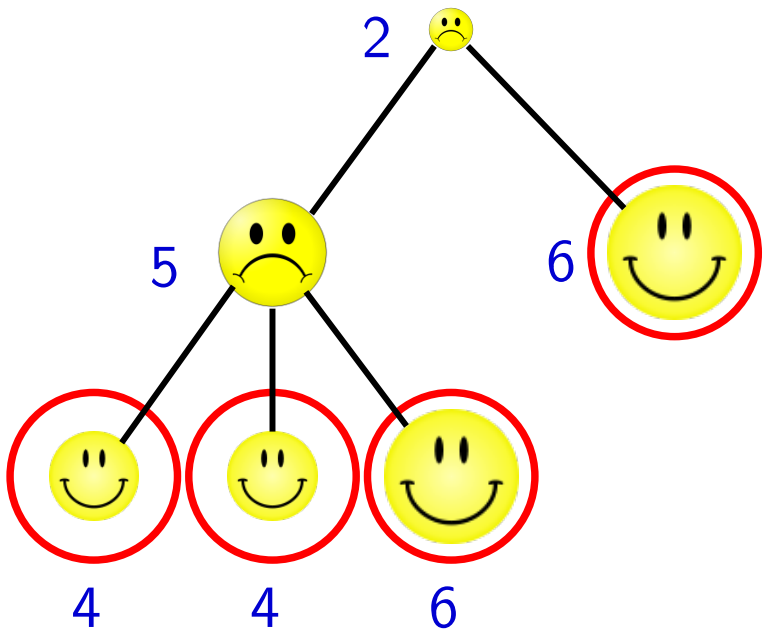


- **Input:** A tree with weights on the vertices.
- **Task:** Find an independent set of maximum weight.

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).

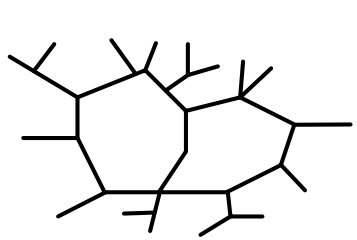
Generalizing trees

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

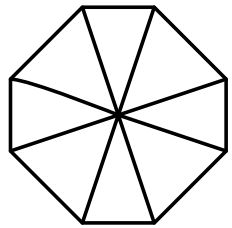
Generalizing trees

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

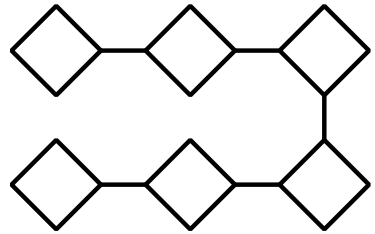
1. Number of cycles is bounded.



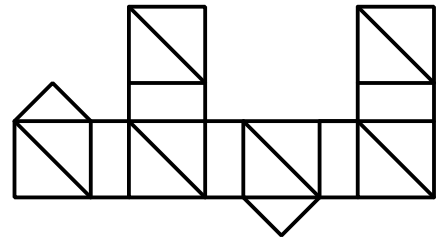
good



bad



bad

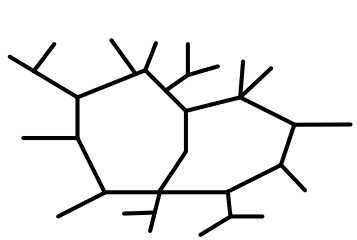


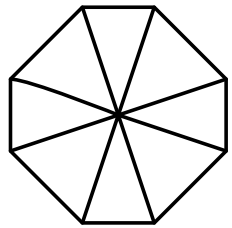
bad

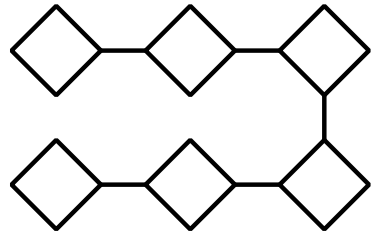
Generalizing trees

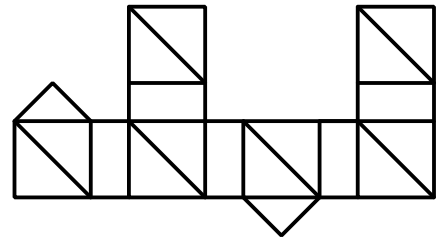
How could we define that a graph is "treelike"?

1. Number of cycles is bounded.

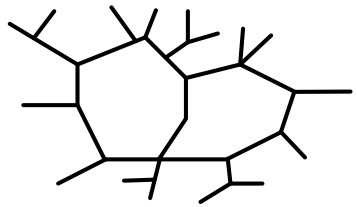

good

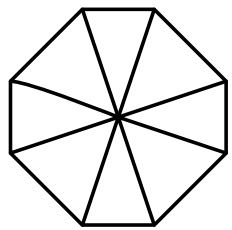

bad

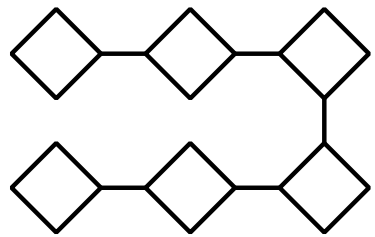

bad

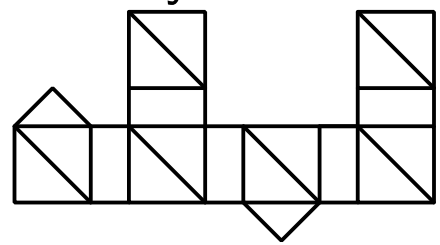

bad

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


good


good

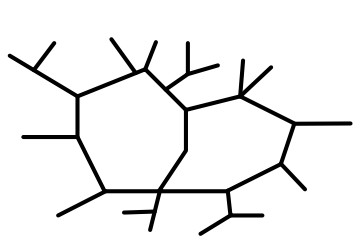

bad

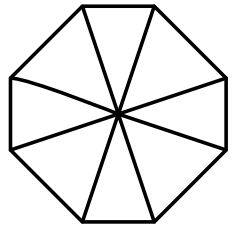

bad

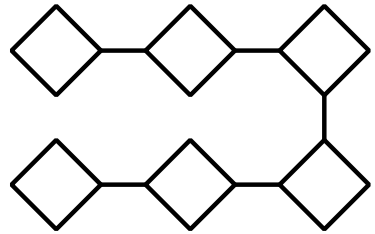
Generalizing trees

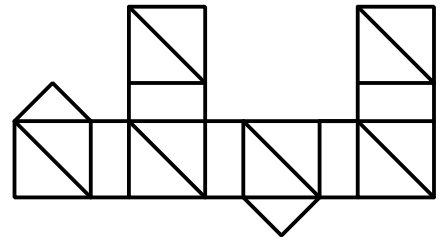
How could we define that a graph is "treelike"?

1. Number of cycles is bounded.

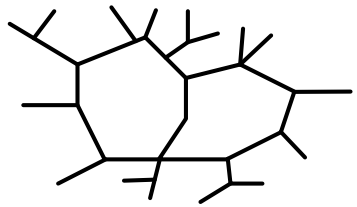

good

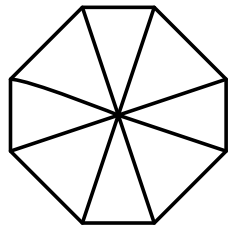

bad

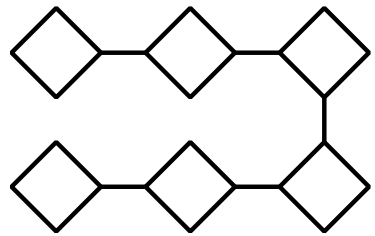

bad

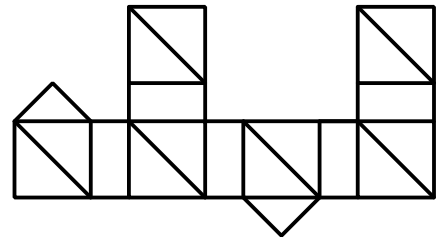

bad

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

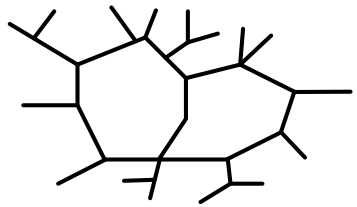

good

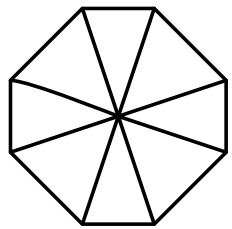

good

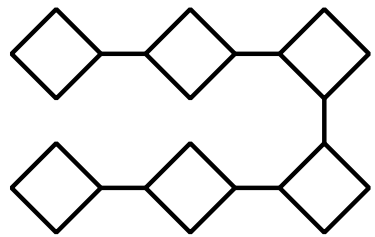

bad

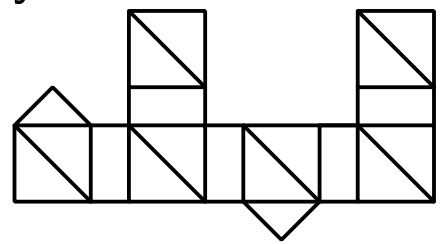

bad

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


bad


bad

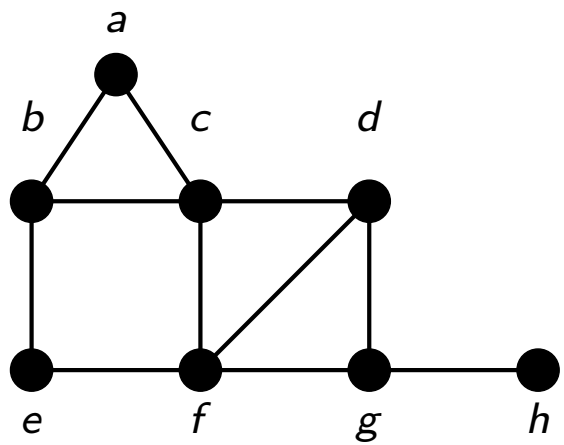

good

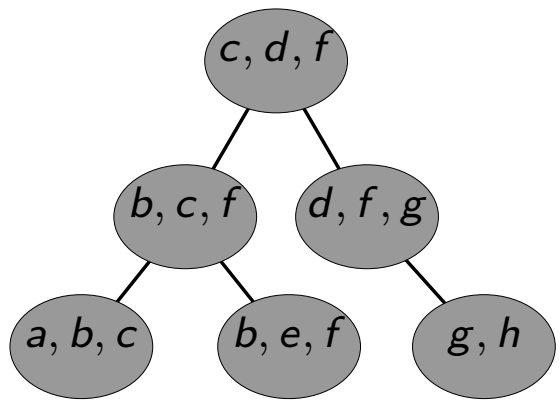

good

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.

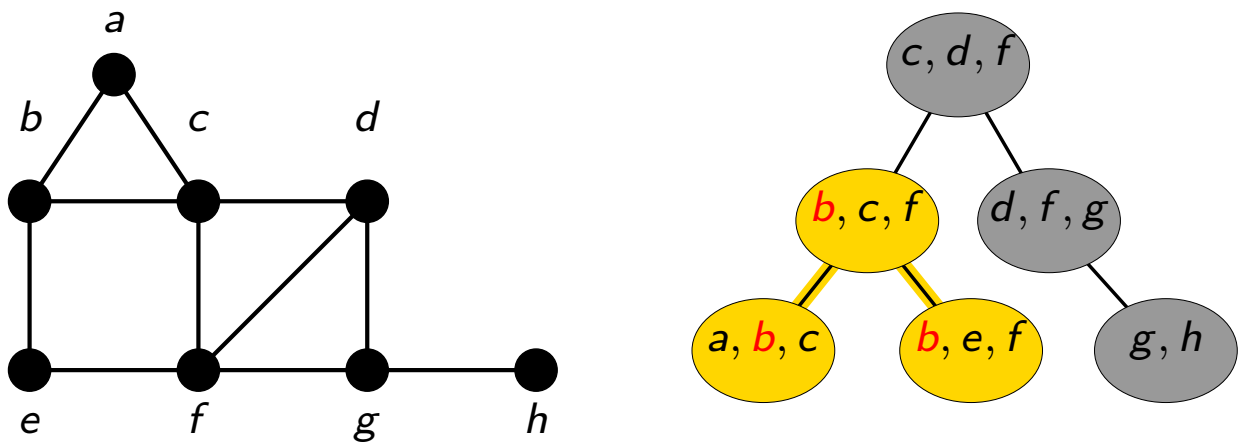




Treewidth — a measure of "tree-likeness"

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

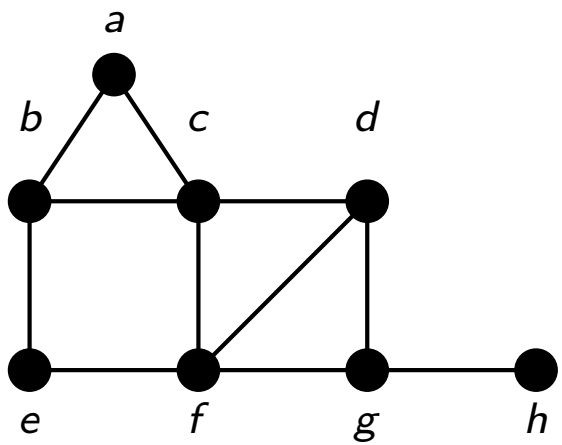
1. If u and v are neighbors, then there is a bag containing both of them.
2. For every v, the bags containing v form a connected subtree.

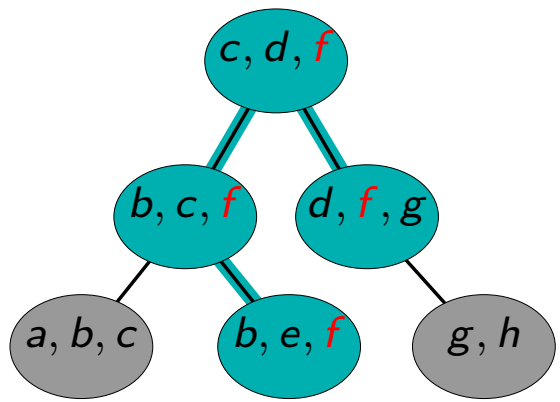


Treewidth — a measure of "tree-likeness"

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.





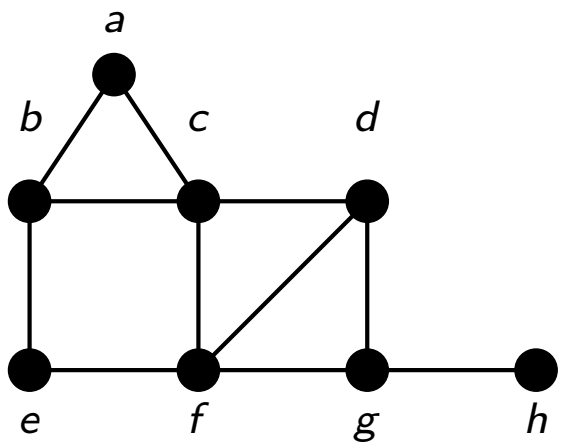
Treewidth — a measure of "tree-likeness"

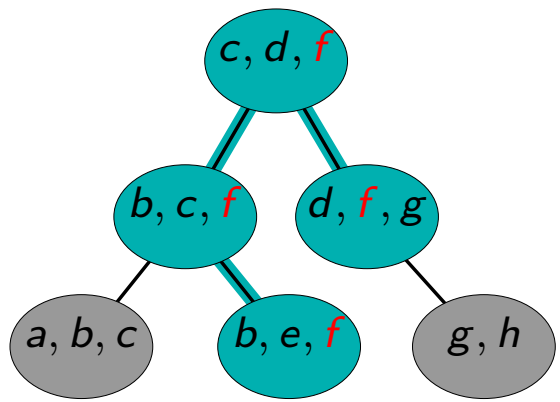
Tree decomposition: Vertices are arranged in a tree structure satisfying the following properties:

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.





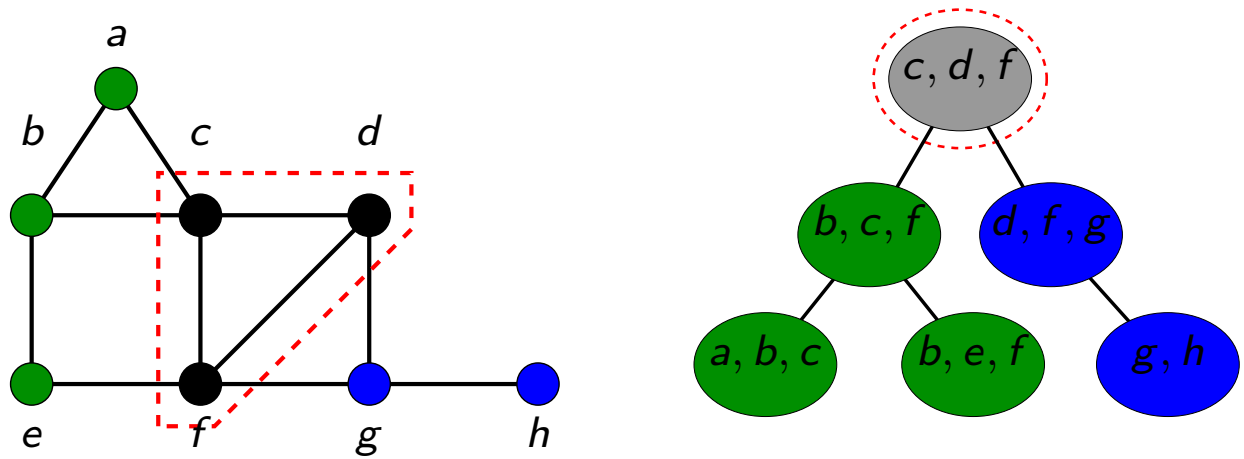
Treewidth — a measure of "tree-likeness"

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.

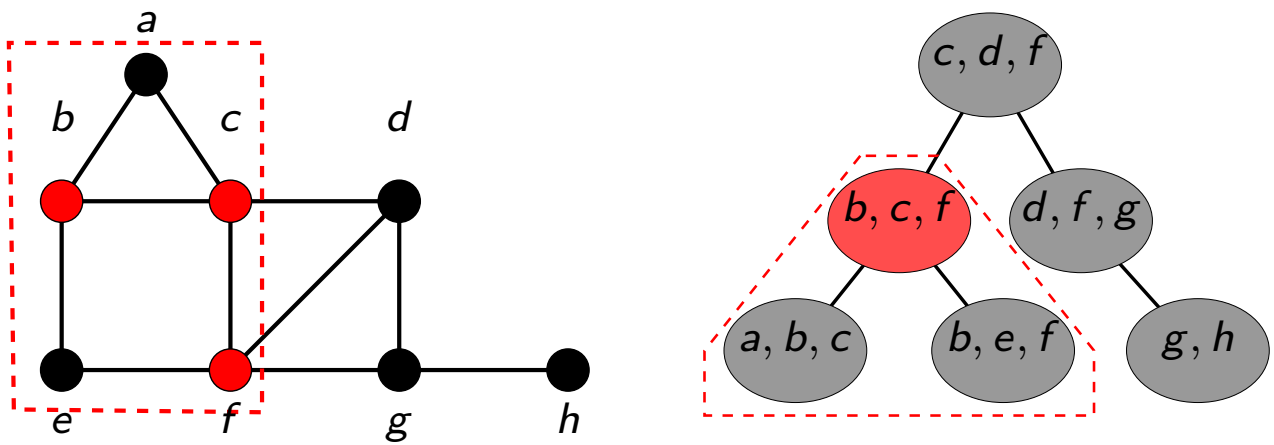
Treewidth — a measure of "tree-likeness"

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.

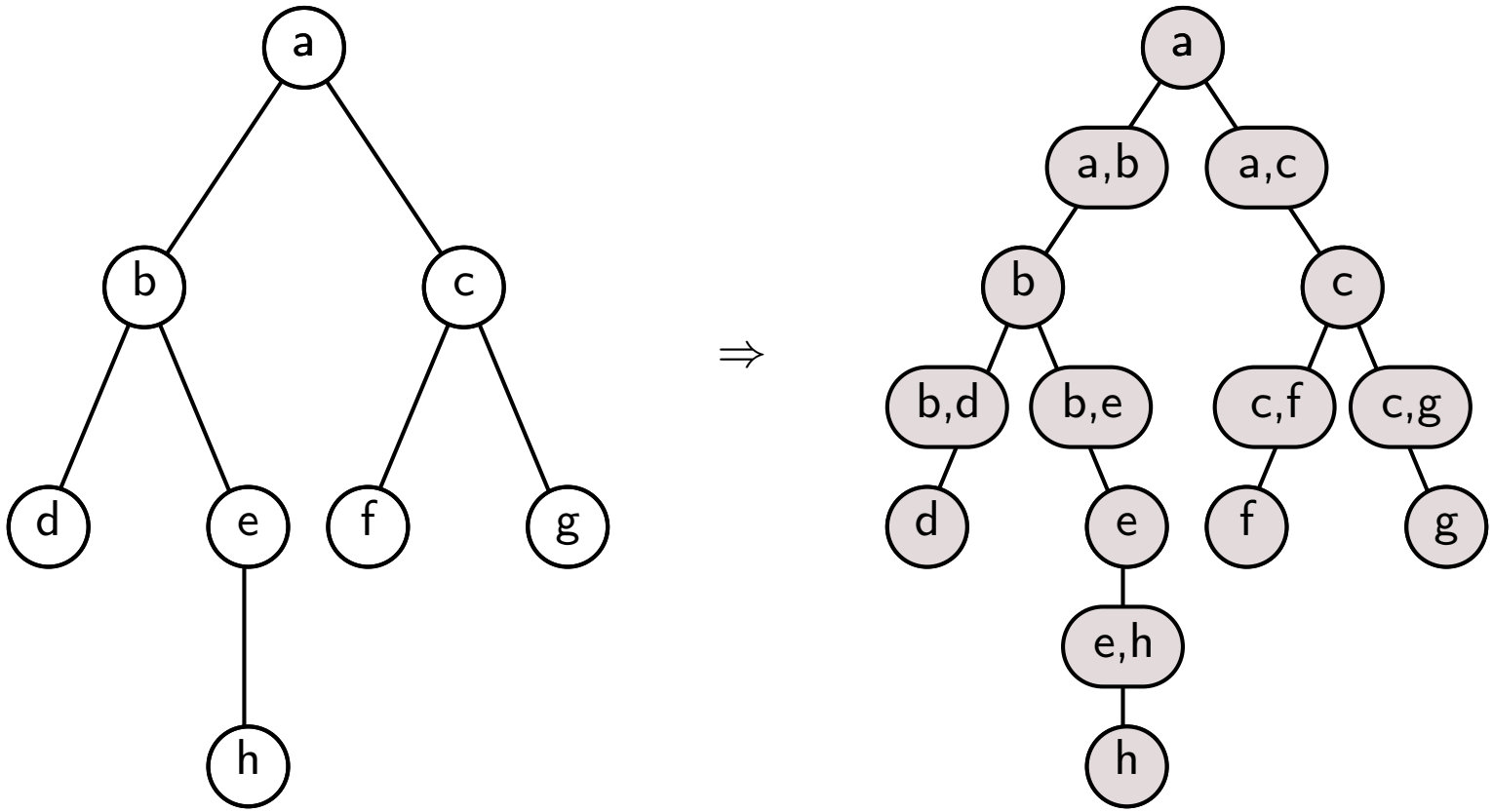
treewidth: width of the best decomposition.



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.

Finding tree decompositions

Various algorithms for finding optimal or approximate tree decompositions if treewidth is w:

- optimal decomposition in time $2^{O(w^3)} \cdot n$ [Bodlaender 1996].
- 4-approximate decomposition in time $2^{O(w)} \cdot n^2$ [Robertson and Seymour].
- 5-approximate decomposition in time $2^{O(w)} \cdot n$ [Bodlaender et al. 2013].
- $O(\sqrt{\log w})$-approximation in polynomial time [Feige, Hajiaghayi, Lee 2008].

As we are mostly interested in algorithms with running time $2^{O(w)} \cdot n^{O(1)}$, we may assume that we have a decomposition.

Weighted Max Independent Set and treewidth

Theorem

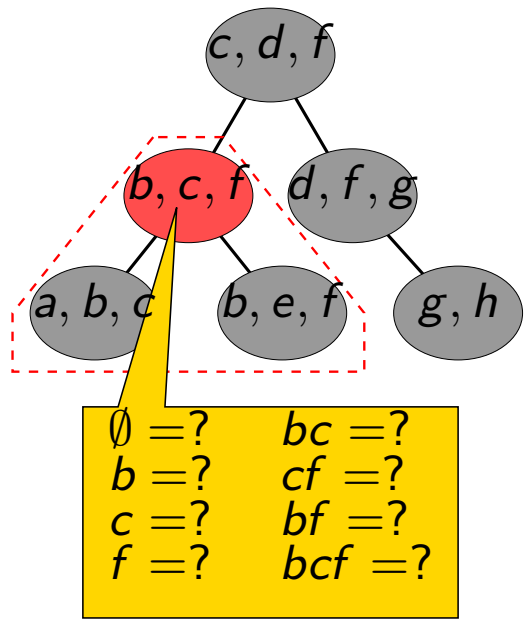
Given a tree decomposition of width w, Weighted Max Independent Set can be solved in time $O(2^w \cdot w^{O(1)} \cdot n)$.

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

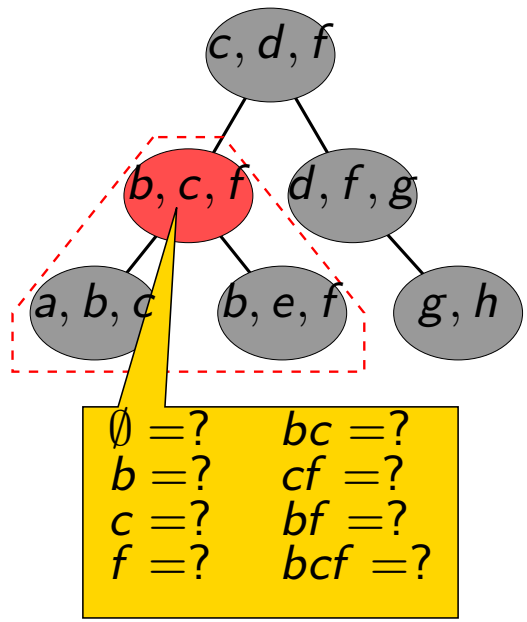
Theorem

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 $I \cap B_x = S$.

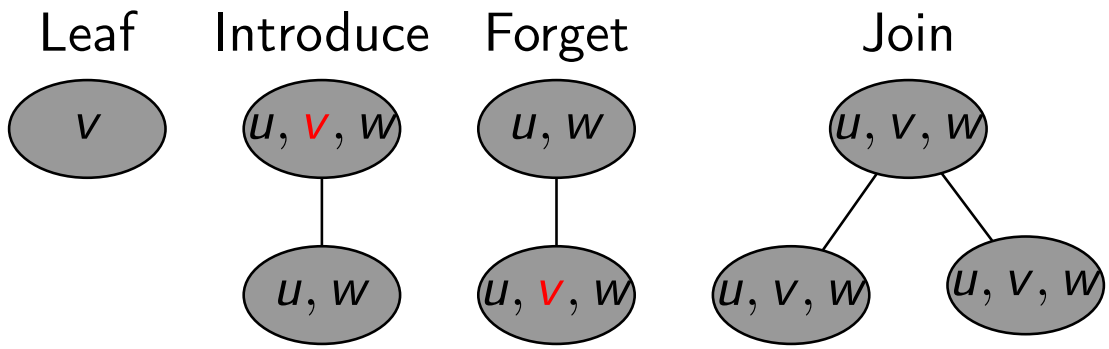
How to determine $M[x, S]$ if all the values are known for the children of x?

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

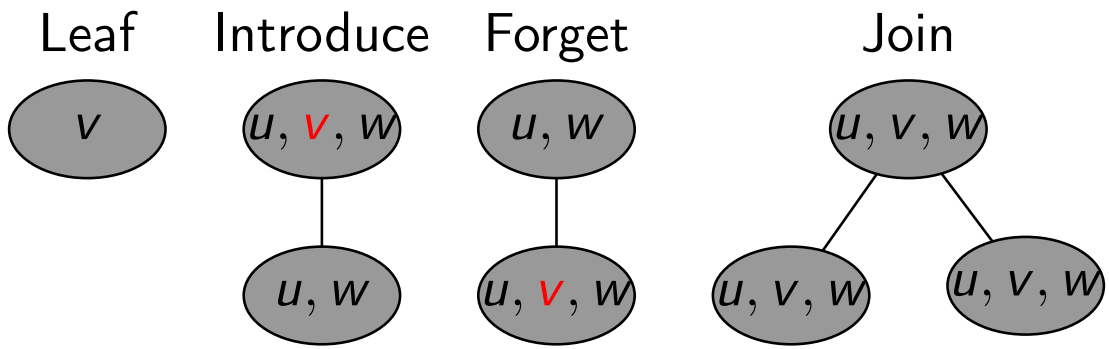
Theorem

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 $O(w^2 n)$.

Weighted Max Independent Set and nice tree decompositions

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

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



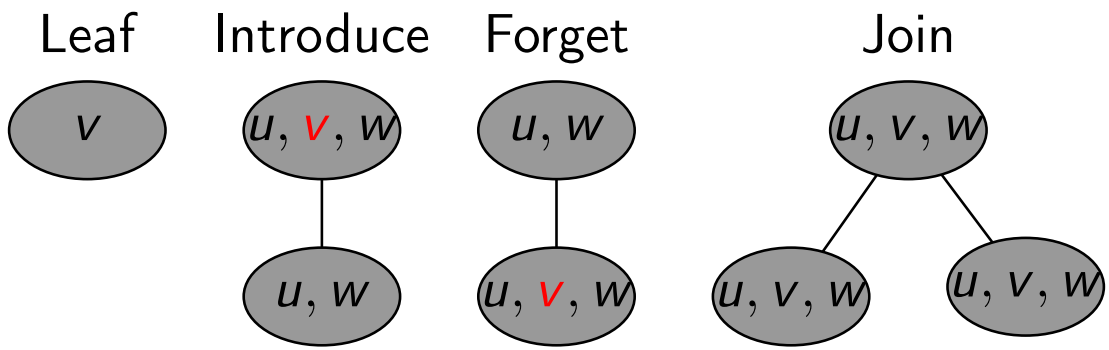
Weighted Max Independent Set and nice tree decompositions

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

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

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

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



Weighted Max Independent Set and nice tree decompositions

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

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

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

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

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

$\Downarrow$

Running time is $O(2^w \cdot w^{O(1)} \cdot n)$.

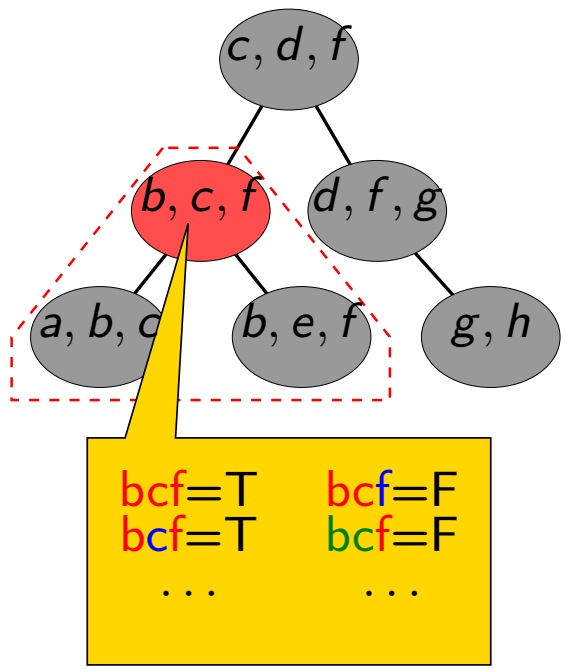
3-Coloring and tree decompositions

Theorem

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

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

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



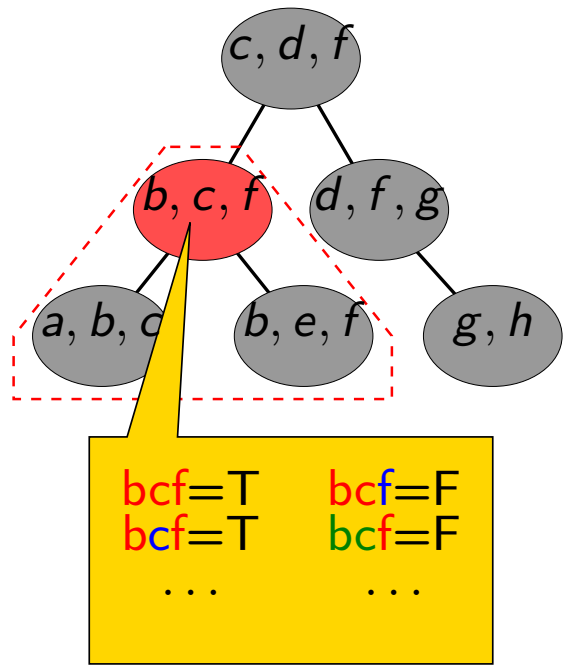
3-Coloring and tree decompositions

Theorem

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

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

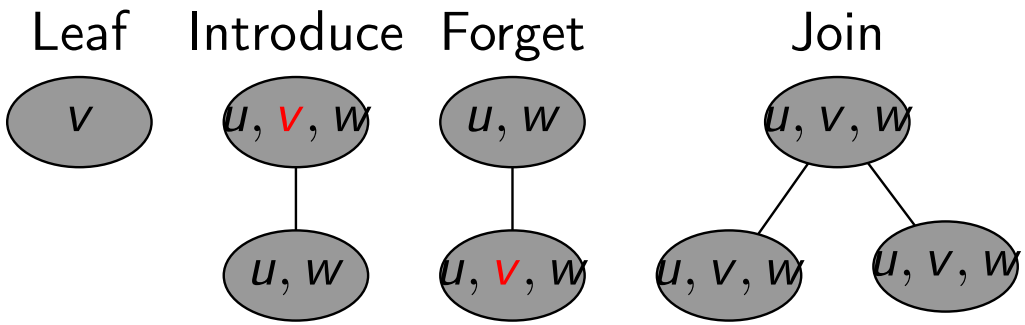
For every node x and coloring $c : B_x \to \{1, 2, 3\}$, we compute the Boolean value $E[x, c]$, which is true if and only if c can be extended to a proper 3-coloring of V_x.



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

3-Coloring and nice tree decompositions

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

Subexponential algorithm for 3-Coloring

Theorem [textbook dynamic programming]

3-Coloring can be solved in time $2^{O(w)} \cdot n^{O(1)}$ on graphs of treewidth w.

$+$

Theorem [Robertson and Seymour]

A planar graph on n vertices has treewidth $O(\sqrt{n})$.

Subexponential algorithm for 3-Coloring

Theorem [textbook dynamic programming]

3-Coloring can be solved in time $2^{O(w)} \cdot n^{O(1)}$ on graphs of treewidth w.

$+$

Theorem [Robertson and Seymour]

A planar graph on n vertices has treewidth $O(\sqrt{n})$.

$\Downarrow$

Corollary

3-Coloring can be solved in time $2^{O(\sqrt{n})}$ on planar graphs.

textbook algorithm + combinatorial bound

$\Downarrow$

subexponential algorithm

Lower bounds

Corollary

3-Coloring can be solved in time $2^{O(\sqrt{n})}$ on planar graphs.

Two natural questions:

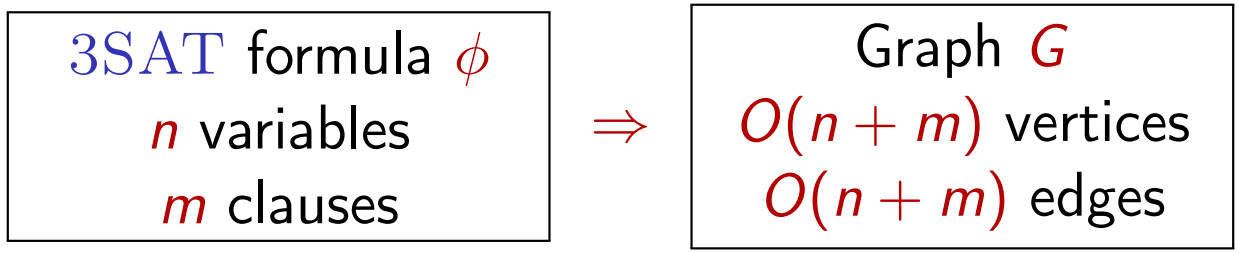
- Can we achieve this running time on general graphs?
- Can we achieve even better running time (e.g., $2^{O(\sqrt[3]{n})}$) on planar graphs?

Lower bounds based on ETH

ETH + Sparsification Lemma

There is no $2^{o(m)}$-time algorithm for m-clause 3SAT.

The textbook reduction from 3SAT to 3-Coloring:



Corollary

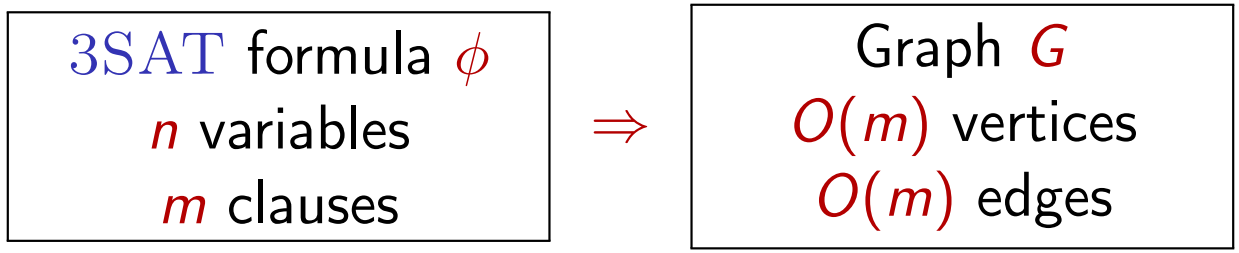
Assuming ETH, there is no $2^{o(n)}$ algorithm for 3-Coloring on an n-vertex graph G.

Lower bounds based on ETH

ETH + Sparsification Lemma

There is no $2^{o(m)}$-time algorithm for m-clause 3SAT.

The textbook reduction from 3SAT to 3-Coloring:



Corollary

Assuming ETH, there is no $2^{o(n)}$ algorithm for 3-Coloring on an n-vertex graph G.

Transfering bounds

There are polynomial-time reductions from, say, 3-Coloring to many other problems such that the reduction increases the number of vertices by at most a constant factor.

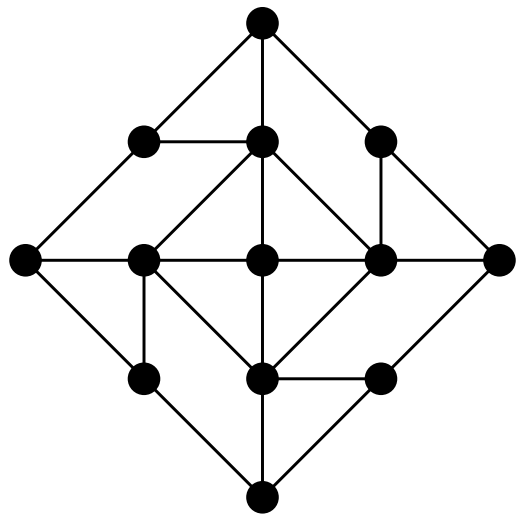
Consequence: Assuming ETH, there is no $2^{o(n)}$ time algorithm on n-vertex graphs for

- Independent Set
- Clique
- Dominating Set
- Vertex Cover
- Hamiltonian Path
- Feedback Vertex Set
- ...

Lower bounds based on ETH

What about 3-Coloring on planar graphs?

The textbook reduction from 3-Coloring to Planar 3-Coloring uses a "crossover gadget" with 4 external connectors:

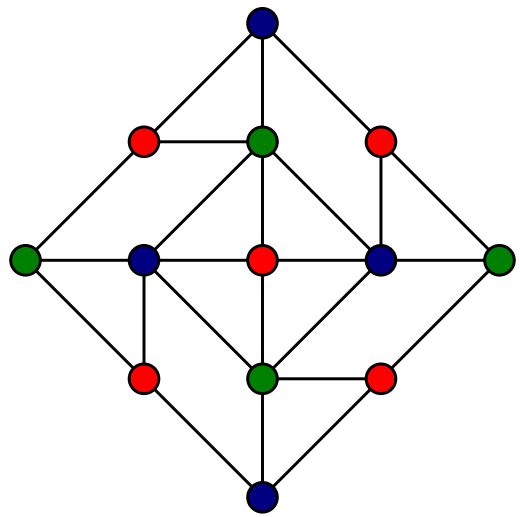


- In every 3-coloring of the gadget, opposite external connectors have the same color.
- Every coloring of the external connectors where the opposite vertices have the same color can be extended to the whole gadget.
- If two edges cross, replace them with a crossover gadget.

Lower bounds based on ETH

What about 3-Coloring on planar graphs?

The textbook reduction from 3-Coloring to Planar 3-Coloring uses a "crossover gadget" with 4 external connectors:

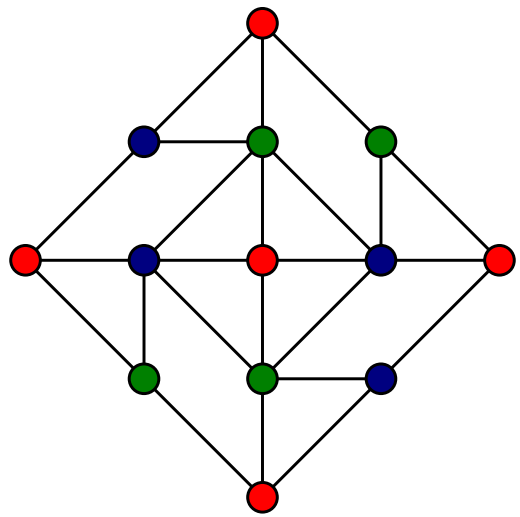


- In every 3-coloring of the gadget, opposite external connectors have the same color.
- Every coloring of the external connectors where the opposite vertices have the same color can be extended to the whole gadget.
- If two edges cross, replace them with a crossover gadget.

Lower bounds based on ETH

What about 3-Coloring on planar graphs?

The textbook reduction from 3-Coloring to Planar 3-Coloring uses a "crossover gadget" with 4 external connectors:



- In every 3-coloring of the gadget, opposite external connectors have the same color.
- Every coloring of the external connectors where the opposite vertices have the same color can be extended to the whole gadget.
- If two edges cross, replace them with a crossover gadget.

Lower bounds based on ETH

- The reduction from 3-Coloring to Planar 3-Coloring introduces $O(1)$ new edges/vertices for each crossing.
- A graph with m edges can be drawn with $O(m^2)$ crossings.

3SAT formula ϕ n variables m clauses	$\Rightarrow$	Graph G $O(m)$ vertices $O(m)$ edges	$\Rightarrow$	Planar graph G' $O(m^2)$ vertices $O(m^2)$ edges

Corollary

Assuming ETH, there is no $2^{o(\sqrt{n})}$ algorithm for 3-Coloring on an n-vertex planar graph G.

(Essentially observed by [Cai and Juedes 2001])

Lower bounds for planar problems

Consequence: Assuming ETH, there is no $2^{o(\sqrt{n})}$ time algorithm on n-vertex **planar graphs** for

- Independent Set
- Dominating Set
- Vertex Cover
- Hamiltonian Path
- Feedback Vertex Set
- ...

Summary so far

Streamlined way of obtaining tight upper and lower bounds for planar problems.

- **Upper bound:**
  Standard bounded-treewidth algorithm + treewidth bound on planar graphs give $2^{O(\sqrt{n})}$ time subexponential algorithms.
- **Lower bound:**
  Textbook NP-hardness proof with quadratic blow up + ETH rule out $2^{o(\sqrt{n})}$ algorithms.

Works for Hamiltonian Cycle, Vertex Cover, Independent Set, Feedback Vertex Set, Dominating Set, Steiner Tree, ...

Parameterized problems

Main idea

Instead of expressing the running time as a function $T(n)$ of n, we express it as a function $T(n, k)$ of the input size n and some parameter k of the input.

In other words: we do not want to be efficient on all inputs of size n, only for those where k is small.

Parameterized problems

Main idea

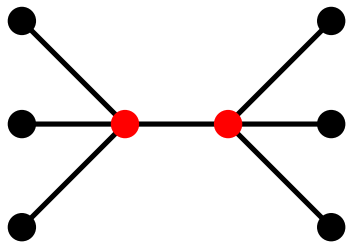
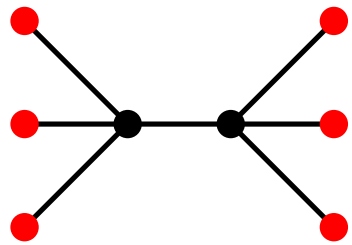
Instead of expressing the running time as a function $T(n)$ of n, we express it as a function $T(n, k)$ of the input size n and some parameter k of the input.

In other words: we do not want to be efficient on all inputs of size n, only for those where k is small.

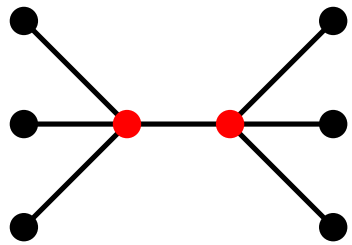
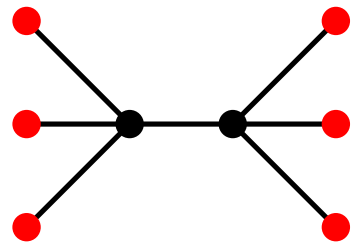
What can be the parameter k?

- The size k of the solution we are looking for.
- The maximum degree of the input graph.
- The treewidth of the input graph.
- The dimension of the point set in the input.
- The length of the strings in the input.
- ...

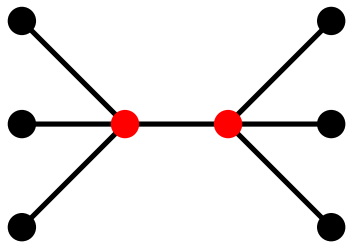
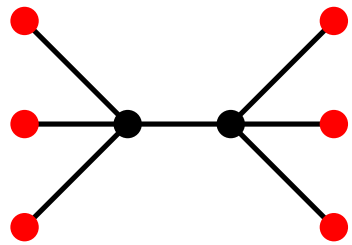
Parameterized complexity

	Vertex Cover	Independent Set
Problem:		
Input:	Graph G, integer k	Graph G, integer k
Question:	Is it possible to cover the edges with k vertices?	Is it possible to find k independent vertices?
		
Complexity:	NP-complete	NP-complete

Parameterized complexity

	Vertex Cover	Independent Set
Problem:		
Input:	Graph G, integer k	Graph G, integer k
Question:	Is it possible to cover the edges with k vertices?	Is it possible to find k independent vertices?
		
Complexity:	NP-complete	NP-complete
Brute force:	$O(n^k)$ possibilities	$O(n^k)$ possibilities

Parameterized complexity

	Vertex Cover	Independent Set
Problem:		
Input:	Graph G, integer k	Graph G, integer k
Question:	Is it possible to cover the edges with k vertices?	Is it possible to find k independent vertices?
		
Complexity:	NP-complete	NP-complete
Brute force:	$O(n^k)$ possibilities	$O(n^k)$ possibilities
	$O(2^k n^2)$ algorithm exists  **exists**	**No $n^{o(k)}$ algorithm known** 

Bounded search tree method

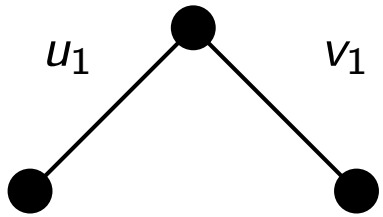
Algorithm for Vertex Cover:

$e_1 = u_1 v_1$

Bounded search tree method

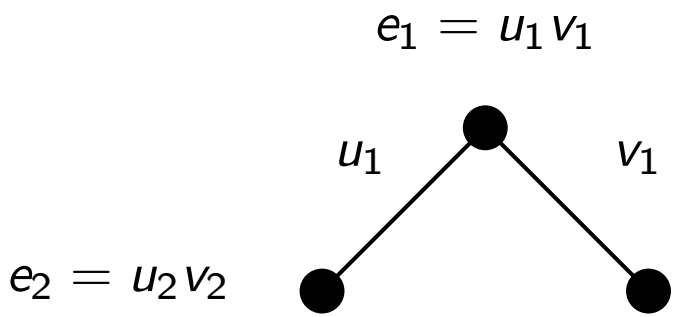
Algorithm for Vertex Cover:

$e_1 = u_1 v_1$



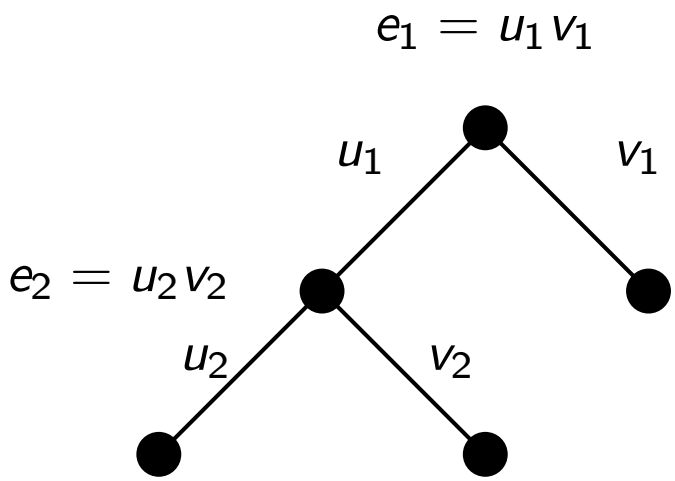
Bounded search tree method

Algorithm for Vertex Cover:



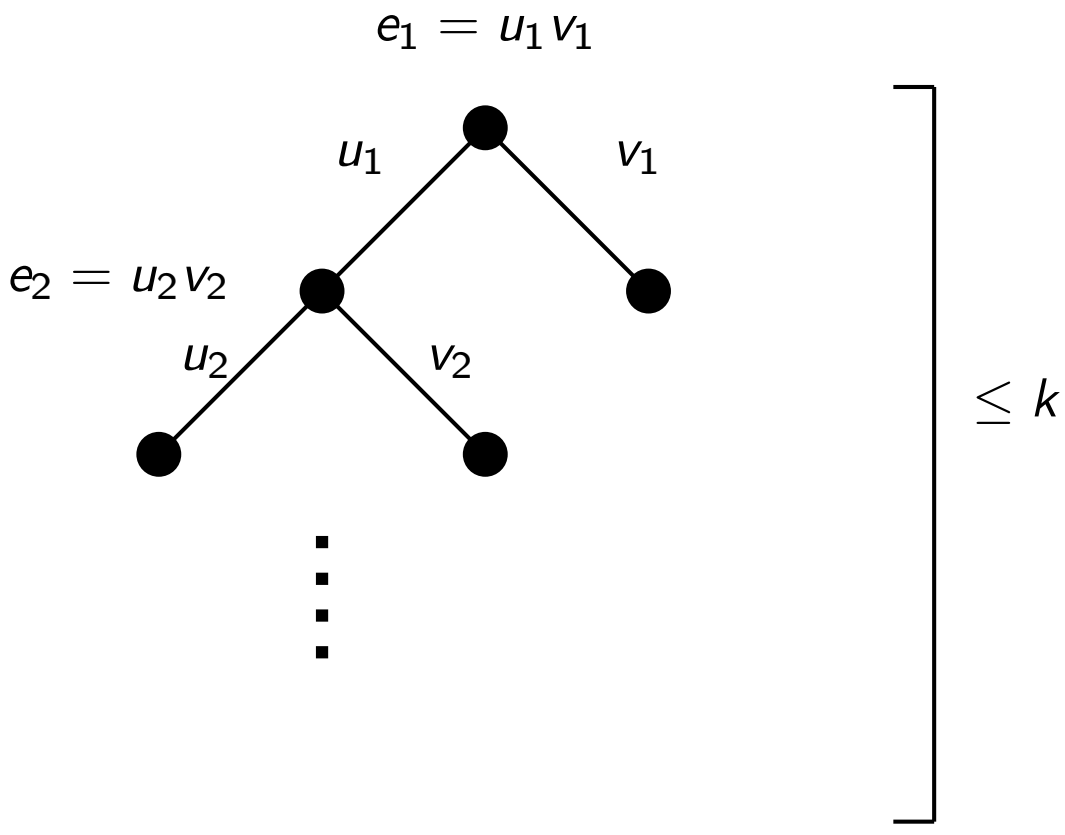
Bounded search tree method

Algorithm for Vertex Cover:



Bounded search tree method

Algorithm for VERTEX COVER:



Height of the search tree $\leq k \Rightarrow$ at most 2^k leaves $\Rightarrow 2^k \cdot n^{O(1)}$ time algorithm.

Fixed-parameter tractability

Main definition

A parameterized problem is **fixed-parameter tractable (FPT)** if there is an $f(k)n^c$ time algorithm for some constant c.

Fixed-parameter tractability

Main definition

A parameterized problem is **fixed-parameter tractable (FPT)** if there is an $f(k)n^c$ time algorithm for some constant c.

Examples of NP-hard problems that are FPT:

- Finding a vertex cover of size k.
- Finding a path of length k.
- Finding k disjoint triangles.
- Drawing the graph in the plane with k edge crossings.
- Finding disjoint paths that connect k pairs of points.
- ...

Lower bounds for planar problems

Consequence: Assuming ETH, there is no $2^{o(\sqrt{n})}$ time algorithm on n-vertex **planar graphs** for

- Independent Set
- Dominating Set
- Vertex Cover
- Hamiltonian Path
- Feedback Vertex Set
- ...

Lower bounds for planar problems

Consequence: Assuming ETH, there is no $2^{o(\sqrt{k})} \cdot n^{O(1)}$ time algorithm on **planar graphs** for

- k-Independent Set
- k-Dominating Set
- k-Vertex Cover
- k-Path
- k-Feedback Vertex Set
- ...

Lower bounds for planar problems

Consequence: Assuming ETH, there is no $2^{o(\sqrt{k})} \cdot n^{O(1)}$ time algorithm on **planar graphs** for

- k-Independent Set
- k-Dominating Set
- k-Vertex Cover
- k-Path
- k-Feedback Vertex Set
- ...

What about matching upper bounds?

Do we have $2^{O(\sqrt{k})} \cdot n^{O(1)}$ time parameterized algorithms for planar problems?

Bidimensionality

Bidimensionality theory [Demaine, Fomin, Hajiaghayi, Thilikos 2005] gives very elegant subexponential algorithms on planar graphs for parameterized problems such as

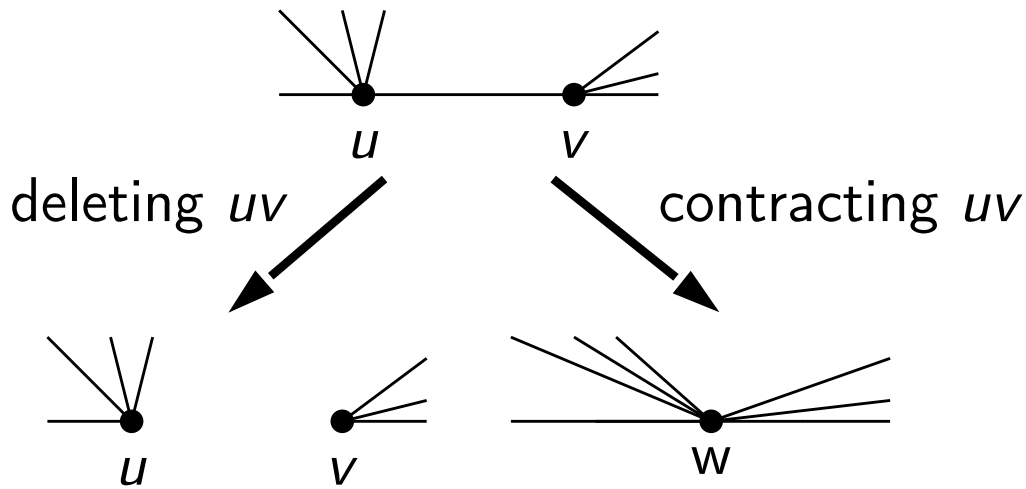
- k-Path
- Vertex Cover
- Feedback Vertex Set
- Independent Set
- Dominating Set

We already know that (assuming ETH), there are no $2^{o(\sqrt{k})} \cdot n^{O(1)}$ time algorithms for these problems.

Minors

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.

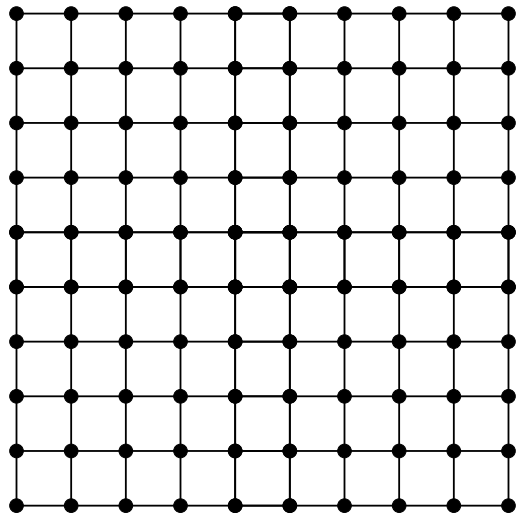


Note: length of the longest path in H is at most the length of the longest path in G.

Planar Excluded Grid Theorem

Theorem [Robertson, Seymour, Thomas 1994]

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

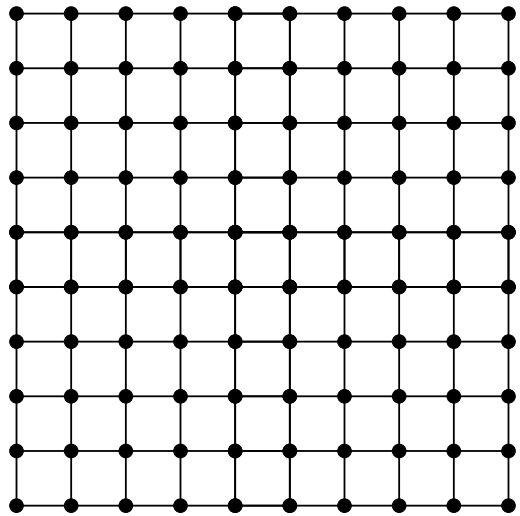


Note: for general graphs, treewidth at least k^{100} or so guarantees a $k \times k$ grid minor [Chekuri and Chuzhoy 2013]!

Planar Excluded Grid Theorem

Theorem [Robertson, Seymour, Thomas 1994]

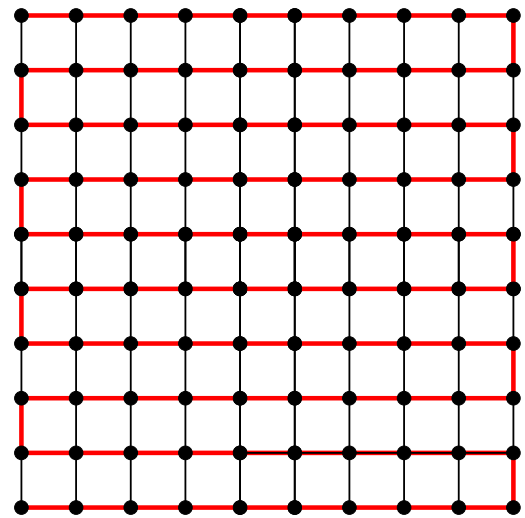
Every planar graph with treewidth at least $5k$ has a $k \times k$ grid minor.



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

Bidimensionality for k-PATH

Observation: If the treewidth of a planar graph G is at least $5\sqrt{k}$
$\Rightarrow$ It has a $\sqrt{k} \times \sqrt{k}$ grid minor (Planar Excluded Grid Theorem)
$\Rightarrow$ The grid has a path of length at least k.
$\Rightarrow$ G has a path of length at least k.

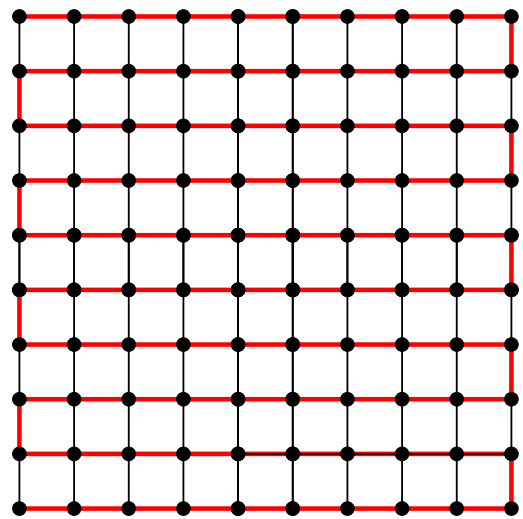


Bidimensionality for k-PATH

Observation: If the treewidth of a planar graph G is at least $5\sqrt{k}$
$\Rightarrow$ It has a $\sqrt{k} \times \sqrt{k}$ grid minor (Planar Excluded Grid Theorem)
$\Rightarrow$ The grid has a path of length at least k.
$\Rightarrow$ G has a path of length at least k.

We use this observation to find a path of length at least k on planar graphs:

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

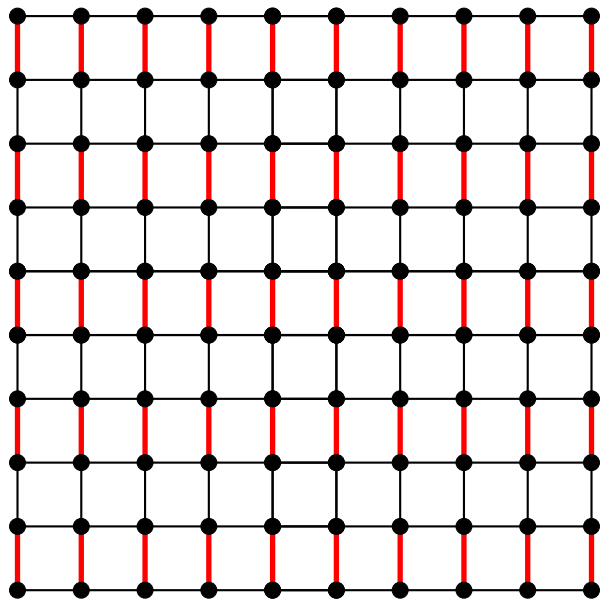


Bidimensionality

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 $c > 0$).



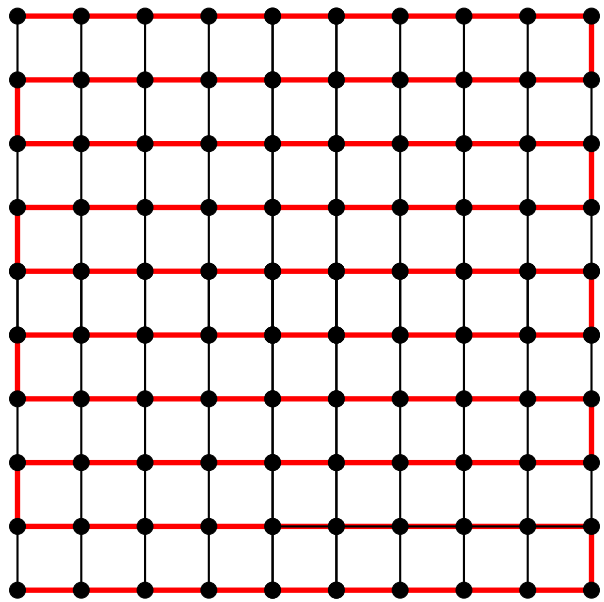
Examples: minimum vertex cover, length of the longest path, feedback vertex set are minor-bidimensional.

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 $c > 0$).



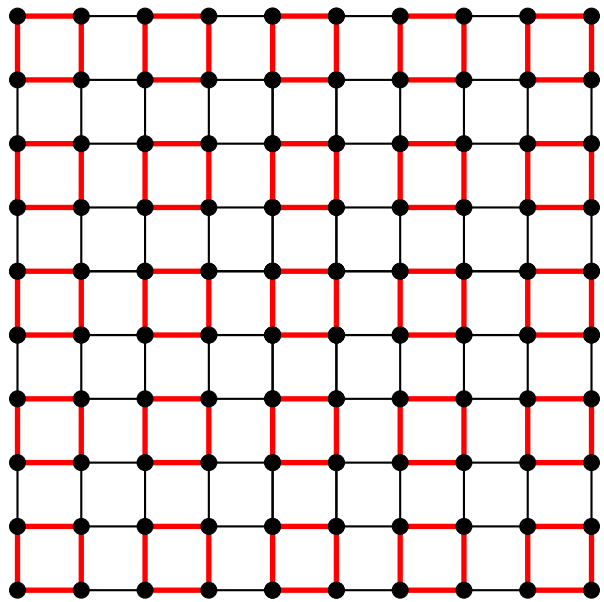
Examples: minimum vertex cover, length of the longest path, feedback vertex set are minor-bidimensional.

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.

Summary of bidimensionality

Tight bounds for minor-bidimensional planar problems.

- **Upper bound:**
  Standard bounded-treewidth algorithm + planar excluded grid theorem give $2^{O(\sqrt{k})} \cdot n^{O(1)}$ time FPT algorithms.
- **Lower bound:**
  Textbook NP-hardness proof with quadratic blow up + ETH rule out $2^{o(\sqrt{n})}$ time algorithms $\Rightarrow$ no $2^{o(\sqrt{k})} \cdot n^{O(1)}$ time algorithm.

Variant of theory works for **contraction-bidimensional** problems, e.g., Independent Set, Dominating Set.

Hard parameterized problems

There are very natural parameterized problems that are unlikely to be fixed-parameter tractable.

Theorem [Chen et al. 2004]

Assuming ETH, there is no $f(k)n^{o(k)}$ time algorithm for Clique for any computable function f.

Hard parameterized problems

There are very natural parameterized problems that are unlikely to be fixed-parameter tractable.

Theorem [Chen et al. 2004]

Assuming ETH, there is no $f(k)n^{o(k)}$ time algorithm for Clique for any computable function f.

We want to obtain similar results for other natural probelms **without** going through the same proof all over again.

We need a notion of reduction to transfer this result to other problems!

Polynomial-time reductions

Polynomial-time reduction from problem P to problem Q: a function ϕ with the following properties:

- $\phi(x)$ can be computed in time $|x|^{O(1)}$,
- $\phi(x)$ is a yes-instance of Q if and only if x is a yes-instance of P.

Polynomial-time reductions

Polynomial-time reduction from problem P to problem Q: a function ϕ with the following properties:

- $\phi(x)$ can be computed in time $|x|^{O(1)}$,
- $\phi(x)$ is a yes-instance of Q if and only if x is a yes-instance of P.

Polynomial-time reductions are not good for our purposes.

Example: Graph G has an independent set k if and only if it has a vertex cover of size $n - k$.

$\Rightarrow$ Transforming an Independent Set instance (G, k) into a Vertex Cover instance $(G, n - k)$ is a correct polynomial-time reduction.

However, Vertex Cover is FPT, but Independent Set is not known to be FPT.

Parameterized reduction

Definition

Parameterized reduction from problem P to problem Q: a function ϕ with the following properties:

- $\phi(x)$ can be computed in time $f(k) \cdot |x|^{O(1)}$, where k is the parameter of x,
- $\phi(x)$ is a yes-instance of $Q \iff x$ is a yes-instance of P.
- If k is the parameter of x and k' is the parameter of $\phi(x)$, then $k' \leq g(k)$ for some function g.

Fact: If there is a parameterized reduction from problem P to problem Q and Q is FPT, then P is also FPT.

Parameterized reduction

Definition

Parameterized reduction from problem P to problem Q: a function ϕ with the following properties:

- $\phi(x)$ can be computed in time $f(k) \cdot |x|^{O(1)}$, where k is the parameter of x,
- $\phi(x)$ is a yes-instance of $Q \iff x$ is a yes-instance of P.
- If k is the parameter of x and k' is the parameter of $\phi(x)$, then $k' \leq g(k)$ for some function g.

Fact: If there is a parameterized reduction from problem P to problem Q and Q is FPT, then P is also FPT.

Non-example: Transforming an Independent Set instance (G, k) into a Vertex Cover instance $(G, n - k)$ is **not** a parameterized reduction.

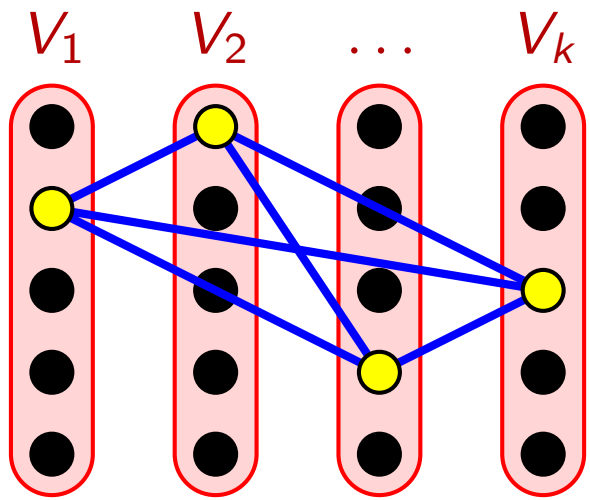
Example: Transforming an Independent Set instance (G, k) into a Clique instance $(\overline{G}, k)$ **is** a parameterized reduction.

Multicolored Clique

A useful variant of Clique:

> Multicolored Clique: The vertices of the input graph G are colored with k colors and we have to find a clique containing one vertex from each color.

(or Partitioned Clique)



Theorem

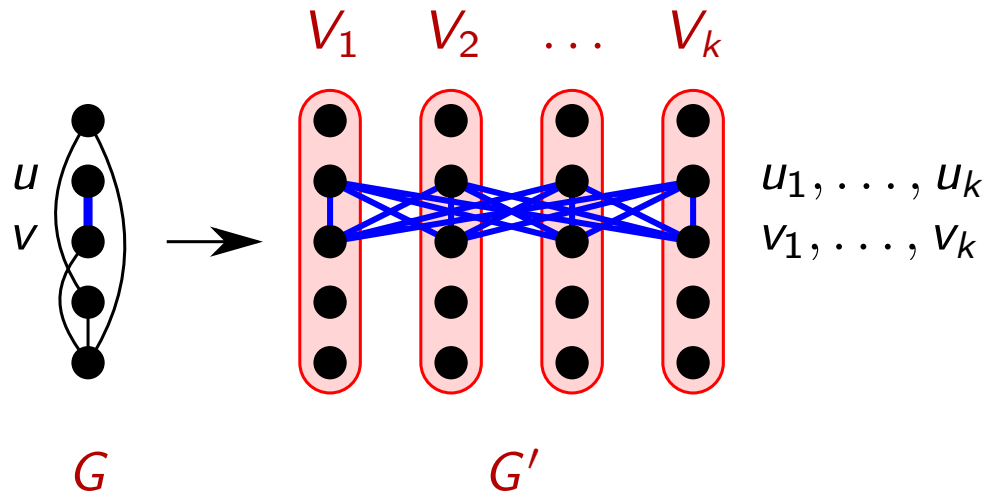
There is a parameterized reduction from Clique to Multicolored Clique.

Multicolored Clique

Theorem

There is a parameterized reduction from Clique to Multicolored Clique.

Create G' by replacing each vertex v with k vertices, one in each color class. If u and v are adjacent in the original graph, connect all copies of u with all copies of v.



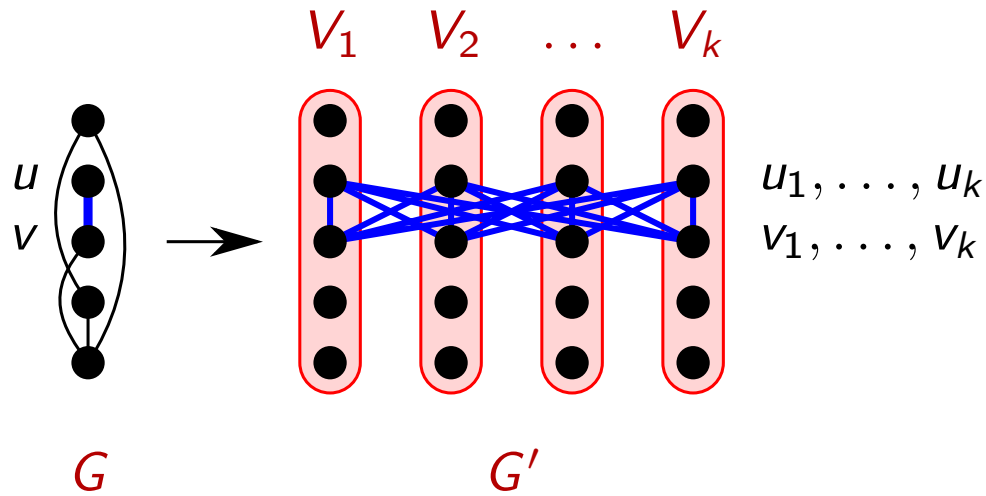
k-clique in G $\iff$ multicolored k-clique in G'.

Multicolored Clique

Theorem

There is a parameterized reduction from Clique to Multicolored Clique.

Create G' by replacing each vertex v with k vertices, one in each color class. If u and v are adjacent in the original graph, connect all copies of u with all copies of v.



k-clique in G $\iff$ multicolored k-clique in G'.

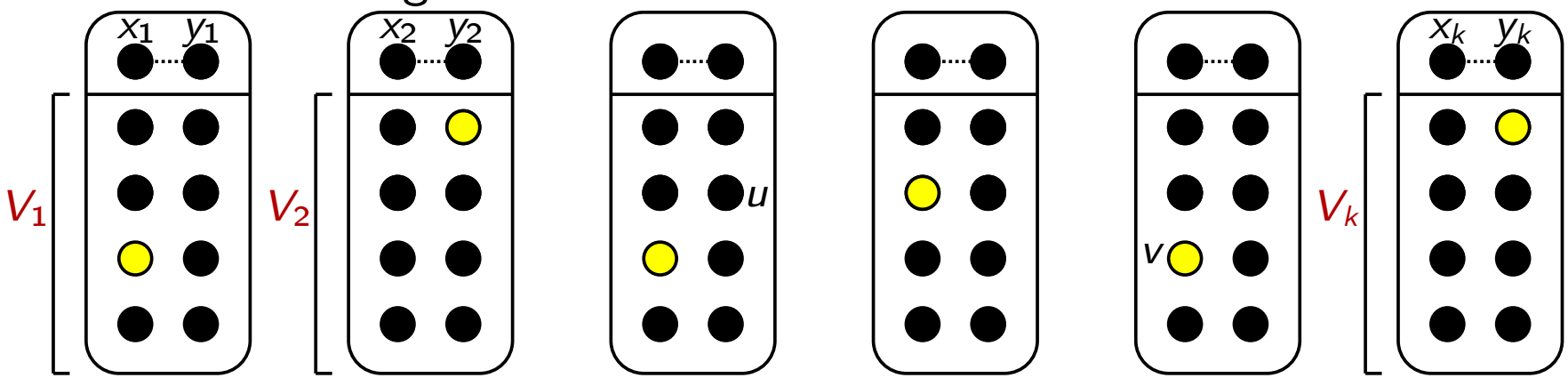
Similarly: reduction to Multicolored Independent Set.

Dominating Set

Theorem

There is a parameterized reduction from Multicolored Independent Set to Dominating Set.

Proof: Let G be a graph with color classes $V_1, \ldots, V_k$. We construct a graph H such that G has a multicolored k-clique iff H has a dominating set of size k.



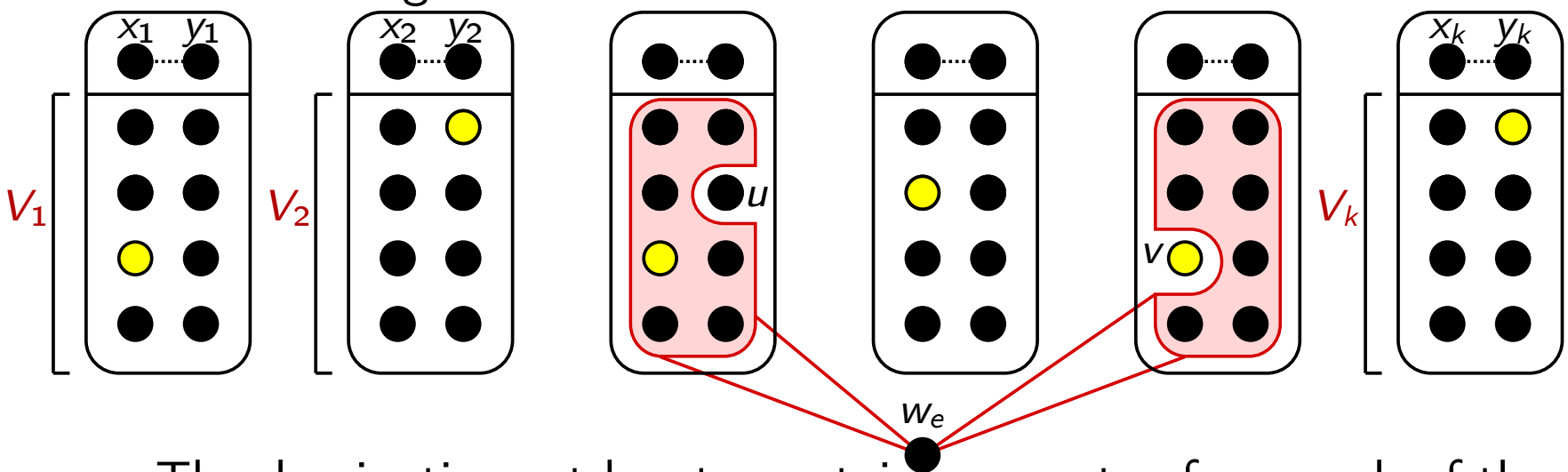
- The dominating set has to contain one vertex from each of the k cliques $V_1, \ldots, V_k$ to dominate every x_i and y_i.

Dominating Set

Theorem

There is a parameterized reduction from Multicolored Independent Set to Dominating Set.

Proof: Let G be a graph with color classes $V_1, \ldots, V_k$. We construct a graph H such that G has a multicolored k-clique iff H has a dominating set of size k.



- The dominating set has to contain one vertex from each of the k cliques $V_1, \ldots, V_k$ to dominate every x_i and y_i.
- For every edge $e = uv$, an additional vertex w_e ensures that these selections describe an independent set.

Variants of Dominating Set

- Dominating Set: Given a graph, find k vertices that dominate every vertex.
- Red-Blue Dominating Set: Given a bipartite graph, find k vertices on the red side that dominate the blue side.
- Set Cover: Given a set system, find k sets whose union covers the universe.
- Hitting Set: Given a set system, find k elements that intersect every set in the system.

All of these problems are equivalent under parameterized reductions, hence at least as hard as Clique.

W[1]-hard problems

- There parameterized reductions from Clique to hundreds of parameterized problems.
- If there is a parameterized reduction from Clique to a parameterized problem P, then P is **W[1]-hard.**
- The reduction we have seen are linear in the parameter, hence it follows that there are no $f(k)n^{o(k)}$ time algorithms for the target problems.
- For planar and geometric problems, we have natural examples where we have weaker lower bounds and $f(k)n^{o(k)}$ time algorithms.

Independent Set for unit disks

Theorem [Alber and Fiala 2004]

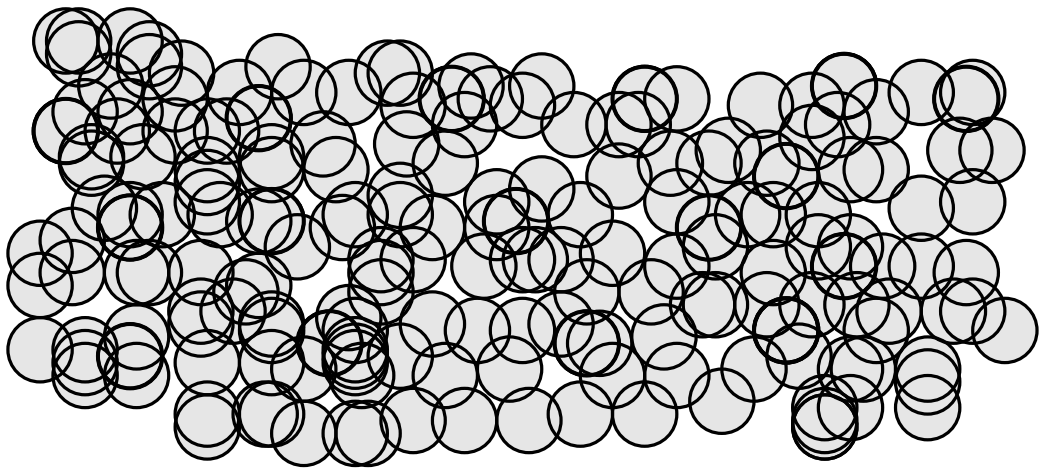
The INDEPENDENT SET problem for unit (diameter) disks can be solved in time $n^{O(\sqrt{k})}$.

Complicated proof using a geometric separator theorem, simple proof by shifting.

Independent Set for unit disks

Theorem [Alber and Fiala 2004]

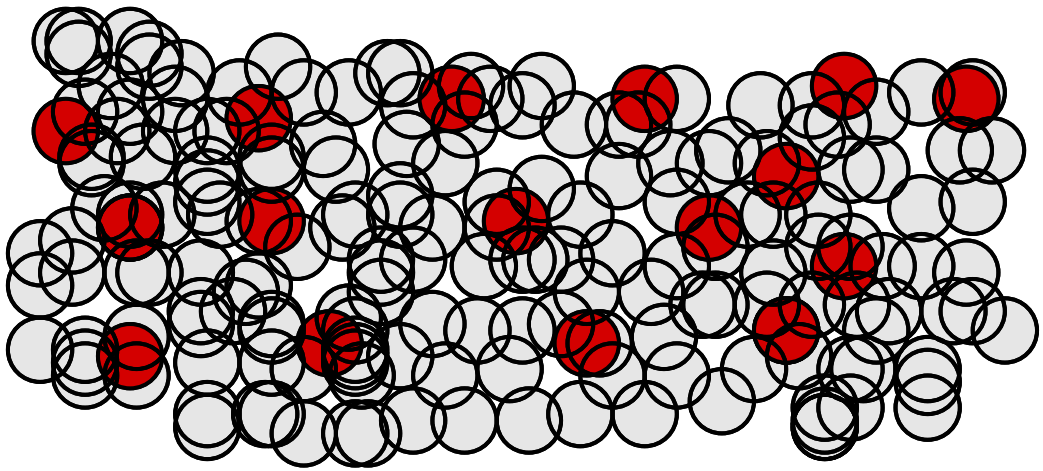
The INDEPENDENT SET problem for unit (diameter) disks can be solved in time $n^{O(\sqrt{k})}$.



Independent Set for unit disks

Theorem [Alber and Fiala 2004]

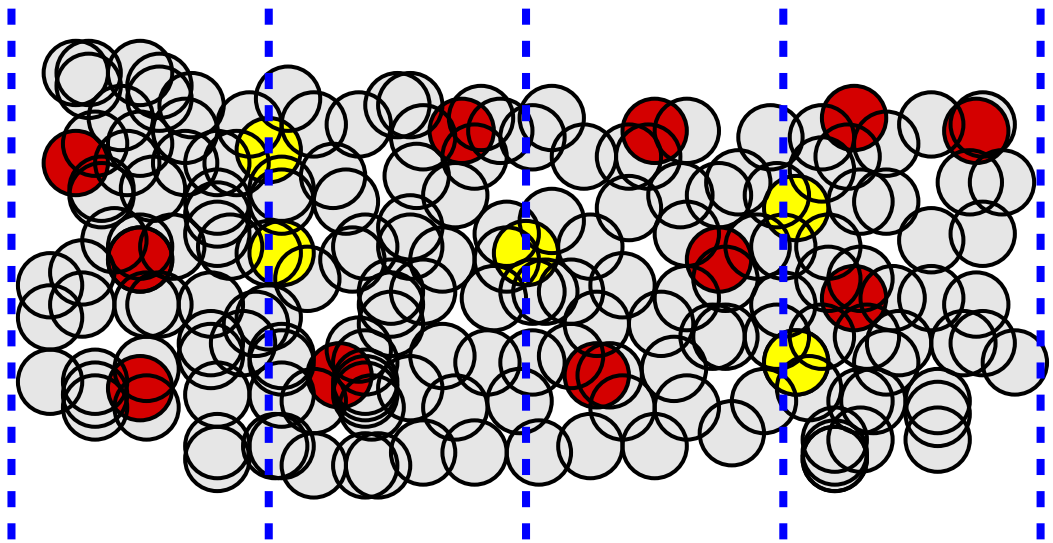
The Independent Set problem for unit (diameter) disks can be solved in time $n^{O(\sqrt{k})}$.



Independent Set for unit disks

Theorem [Alber and Fiala 2004]

The INDEPENDENT SET problem for unit (diameter) disks can be solved in time $n^{O(\sqrt{k})}$.



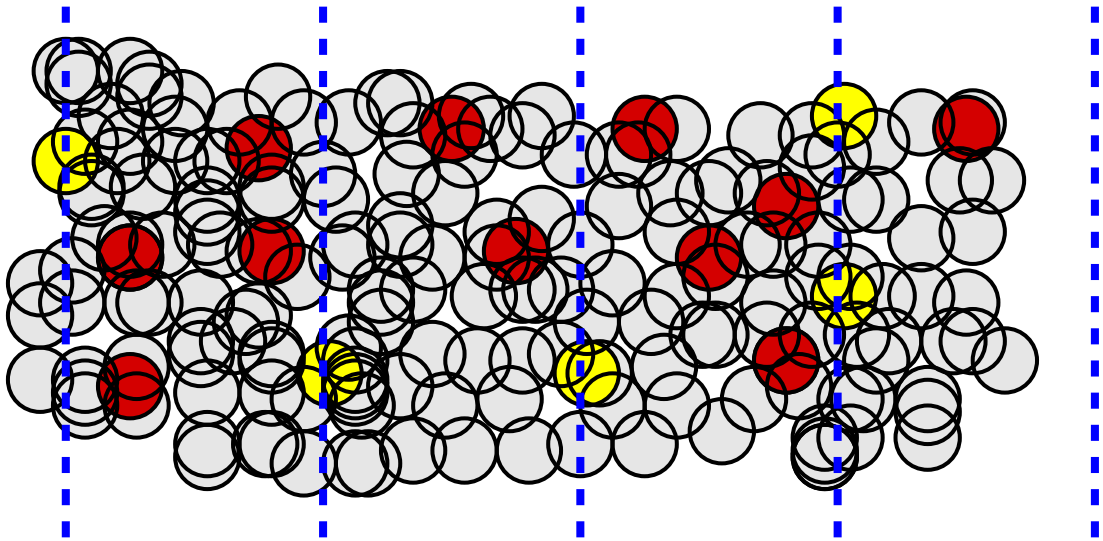
Consider a family of vertical lines at distance $\lfloor\sqrt{k}\rfloor$ from each other, going through $(i, 0)$ for some integer $0 \leq i < \lfloor\sqrt{k}\rfloor$.

Claim: Exists i such that the lines hit $O(\sqrt{k})$ disks of the solution.

Independent Set for unit disks

Theorem [Alber and Fiala 2004]

The INDEPENDENT SET problem for unit (diameter) disks can be solved in time $n^{O(\sqrt{k})}$.



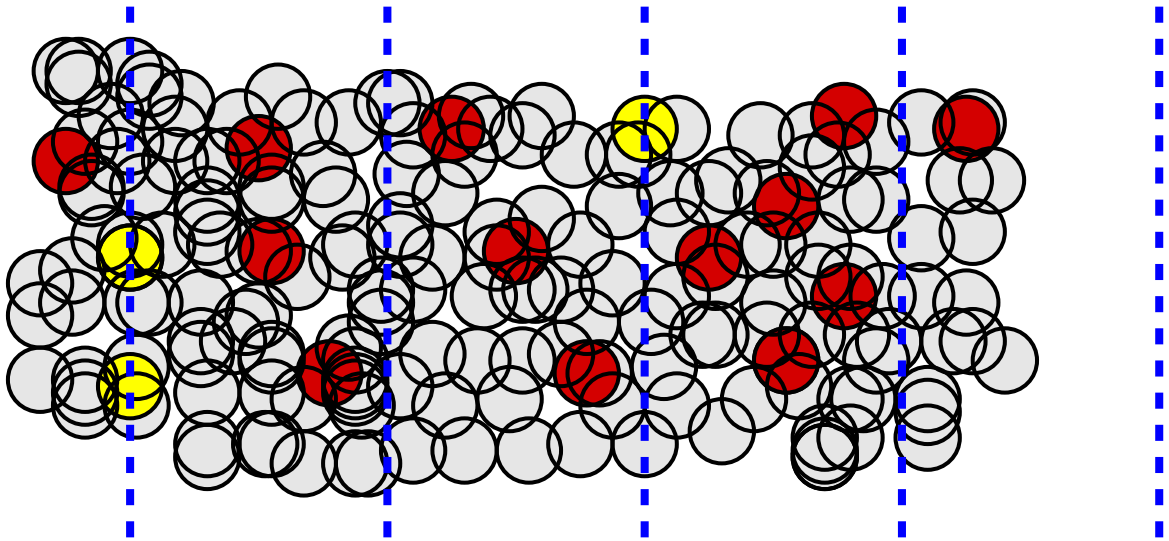
Consider a family of vertical lines at distance $\lfloor\sqrt{k}\rfloor$ from each other, going through $(i, 0)$ for some integer $0 \leq i < \lfloor\sqrt{k}\rfloor$.

Claim: Exists i such that the lines hit $O(\sqrt{k})$ disks of the solution.

Independent Set for unit disks

Theorem [Alber and Fiala 2004]

The INDEPENDENT SET problem for unit (diameter) disks can be solved in time $n^{O(\sqrt{k})}$.



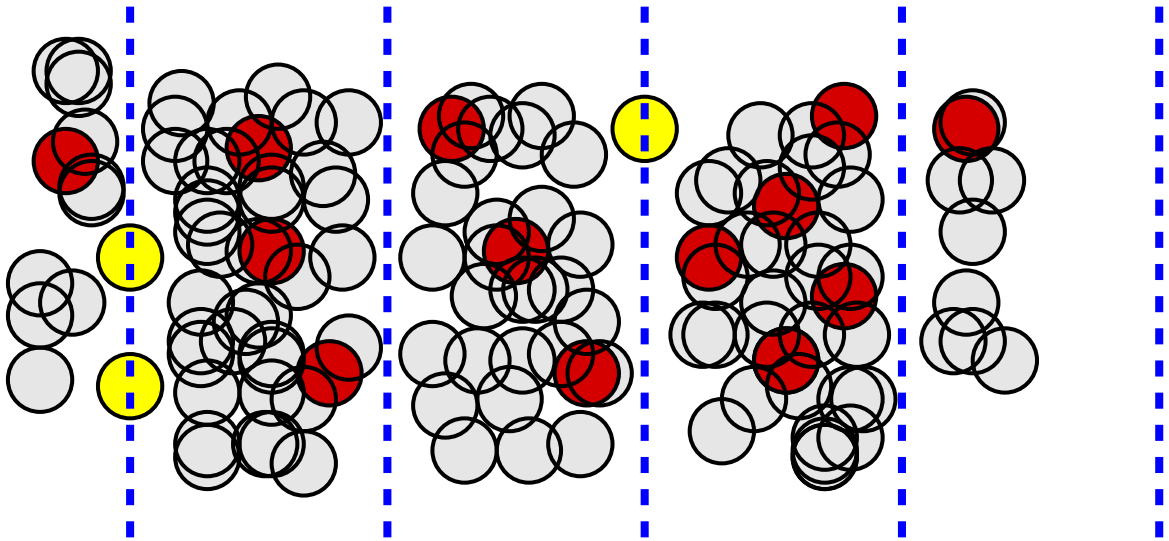
Consider a family of vertical lines at distance $\lfloor\sqrt{k}\rfloor$ from each other, going through $(i, 0)$ for some integer $0 \leq i < \lfloor\sqrt{k}\rfloor$.

Claim: Exists i such that the lines hit $O(\sqrt{k})$ disks of the solution.

Independent Set for unit disks

Theorem [Alber and Fiala 2004]

The INDEPENDENT SET problem for unit (diameter) disks can be solved in time $n^{O(\sqrt{k})}$.



Consider a family of vertical lines at distance $\lfloor\sqrt{k}\rfloor$ from each other, going through $(i, 0)$ for some integer $0 \leq i < \lfloor\sqrt{k}\rfloor$.

Algorithm: Guess i and the $O(\sqrt{k})$ disks hit by the lines $\Rightarrow$ Remove every disk intersected by the lines or disks $\Rightarrow$ Problem falls apart into strips of height $O(\sqrt{k})$; can be solved optimally in time $n^{O(\sqrt{k})}$.

Independent Set for unit disks

Theorem [Alber and Fiala 2004]

The Independent Set problem for unit (diameter) disks can be solved in time $n^{O(\sqrt{k})}$.

Matching lower bound:

Theorem

Independent Set for unit disks is

- is W[1]-hard, and
- cannot be solved in time $f(k)n^{o(\sqrt{k})}$ for any function f.

Key technique for the hardness proof: the Grid Tiling problem.

Grid Tiling

Grid Tiling

Input: A $k \times k$ matrix and a set of pairs $S_{i,j} \subseteq [D] \times [D]$ for each cell.

Find: A pair $s_{i,j} \in S_{i,j}$ for each cell such that

- Vertical neighbors agree in the 1st coordinate.
- Horizontal neighbors agree in the 2nd coordinate.

(1,1) (3,1) (2,4)	(5,1) (1,4) (5,3)	(1,1) (2,4) (3,3)
(2,2) (1,4)	(3,1) (1,2)	(2,2) (2,3)
(1,3) (2,3) (3,3)	(1,1) (1,3)	(2,3) (5,3)

$k = 3,\ D = 5$

Grid Tiling

Grid Tiling

Input: A $k \times k$ matrix and a set of pairs $S_{i,j} \subseteq [D] \times [D]$ for each cell.

Find: A pair $s_{i,j} \in S_{i,j}$ for each cell such that

- Vertical neighbors agree in the 1st coordinate.
- Horizontal neighbors agree in the 2nd coordinate.

(1,1) (3,1) (2,4)	(5,1) (1,4) (5,3)	(1,1) (2,4) (3,3)
(2,2) (1,4)	(3,1) (1,2)	(2,2) (2,3)
(1,3) (2,3) (3,3)	(1,1) (1,3)	(2,3) (5,3)

$k = 3$, $D = 5$

Grid Tiling

Grid Tiling

Input: A $k \times k$ matrix and a set of pairs $S_{i,j} \subseteq [D] \times [D]$ for each cell.

Find: A pair $s_{i,j} \in S_{i,j}$ for each cell such that

- Vertical neighbors agree in the 1st coordinate.
- Horizontal neighbors agree in the 2nd coordinate.

Simple proof:

Fact

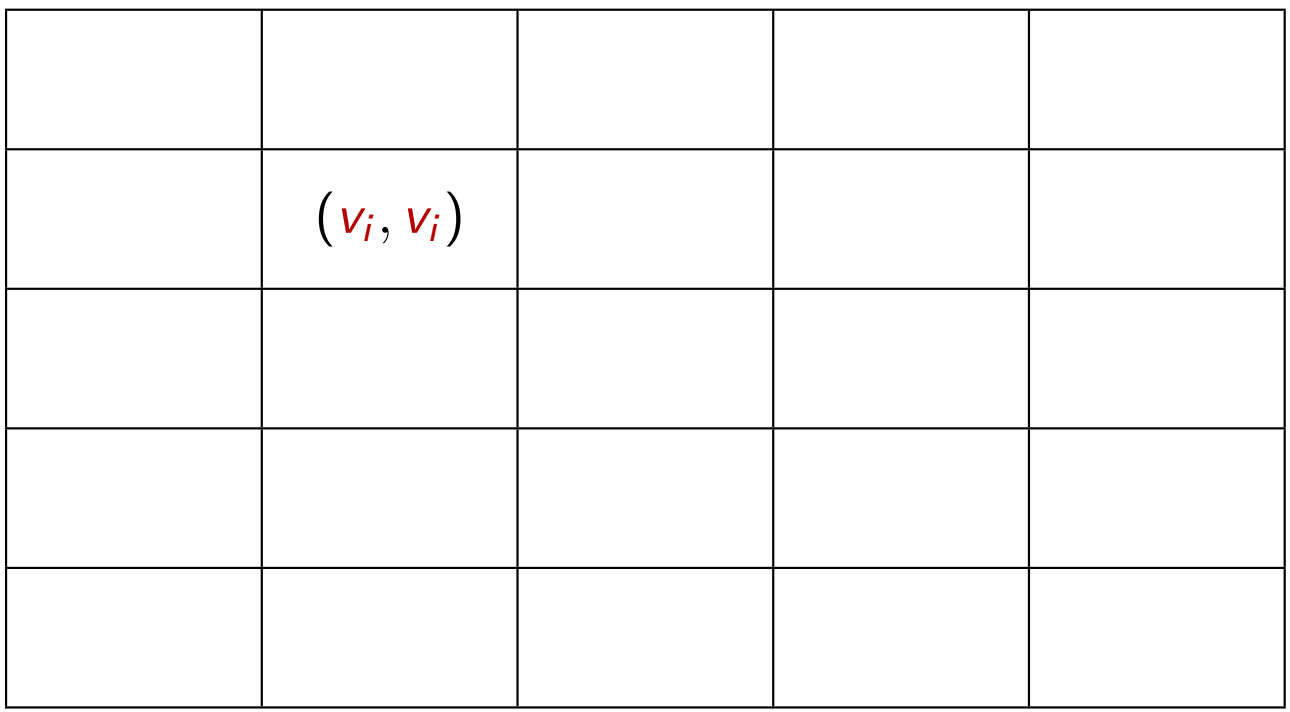
There is a parameterized reduction from k-Clique to $k \times k$ Grid Tiling.

Grid Tiling is W[1]-hard

Reduction from k-Clique

Definition of the sets:

- For $i = j$: $(x, y) \in S_{i,j} \iff x = y$
- For $i \neq j$: $(x, y) \in S_{i,j} \iff x$ and y are adjacent.



Each diagonal cell defines a value v_i ...

Grid Tiling is W[1]-hard

Reduction from k-Clique

Definition of the sets:

- For $i = j$: $(x, y) \in S_{i,j} \iff x = y$
- For $i \neq j$: $(x, y) \in S_{i,j} \iff x$ and y are adjacent.

	$(v_i, .)$			
$(., v_i)$	(v_i, v_i)	$(., v_i)$	$(., v_i)$	$(., v_i)$
	$(v_i, .)$			
	$(v_i .,)$			
	$(v_i, .)$			

. . . which appears on a "cross"

Grid Tiling is W[1]-hard

Reduction from k-Clique

Definition of the sets:

- For $i = j$: $(x, y) \in S_{i,j} \iff x = y$
- For $i \neq j$: $(x, y) \in S_{i,j} \iff x$ and y are adjacent.

	$(v_i, .)$			
$(., v_i)$	(v_i, v_i)	$(., v_i)$	$(., v_i)$	$(., v_i)$
	$(v_i, .)$			
	$(v_i, .)$		(v_j, v_j)	
	$(v_i, .)$			

v_i and v_j are adjacent for every $1 \leq i < j \leq k$.

Grid Tiling is W[1]-hard

Reduction from k-Clique

Definition of the sets:

- For $i = j$: $(x, y) \in S_{i,j} \iff x = y$
- For $i \neq j$: $(x, y) \in S_{i,j} \iff x$ and y are adjacent.

	$(v_i, .)$		$(v_j, .)$	
$(., v_i)$	(v_i, v_i)	$(., v_i)$	(v_j, v_i)	$(., v_i)$
	$(v_i, .)$		$(v_j, .)$	
$(., v_j)$	(v_i, v_j)	$(., v_j)$	(v_j, v_j)	$(., v_j)$
	$(v_i, .)$		$(v_j, .)$	

v_i and v_j are adjacent for every $1 \leq i < j \leq k$.

Grid Tiling and planar problems

Theorem

$k \times k$ Grid Tiling is W[1]-hard and, assuming ETH, cannot be solved in time $f(k)n^{o(k)}$ for any function f.

This lower bound is the key for proving hardness results for planar graphs.

Grid Tiling with $\leq$

Grid Tiling with $\leq$

Input: A $k \times k$ matrix and a set of pairs $S_{i,j} \subseteq [D] \times [D]$ for each cell.

Find: A pair $s_{i,j} \in S_{i,j}$ for each cell such that

- 1st coordinate of $s_{i,j} \leq$ 1st coordinate of $s_{i+1,j}$.
- 2nd coordinate of $s_{i,j} \leq$ 2nd coordinate of $s_{i,j+1}$.

(5,1) (1,2) (3,3)	(4,3) (3,2)	(2,3) (2,5)
(2,1) (5,5) (3,5)	(4,2) (5,3)	(5,1) (3,2)
(5,1) (2,2) (5,3)	(2,1) (4,2)	(3,1) (3,2) (3,3)

$k = 3$, $D = 5$

Grid Tiling with $\leq$

Grid Tiling with $\leq$

Input: A $k \times k$ matrix and a set of pairs $S_{i,j} \subseteq [D] \times [D]$ for each cell.

Find: A pair $s_{i,j} \in S_{i,j}$ for each cell such that

- 1st coordinate of $s_{i,j} \leq$ 1st coordinate of $s_{i+1,j}$.
- 2nd coordinate of $s_{i,j} \leq$ 2nd coordinate of $s_{i,j+1}$.

Variant of the previous proof:

Theorem

There is a parameterized reduction from $k \times k$-Grid Tiling to $O(k) \times O(k)$ Grid Tiling with $\leq$.

Very useful starting point for geometric problems!

Independent Set for unit disks

Theorem [Alber and Fiala 2004]

The Independent Set problem for unit (diameter) disks can be solved in time $n^{O(\sqrt{k})}$.

Matching lower bound:

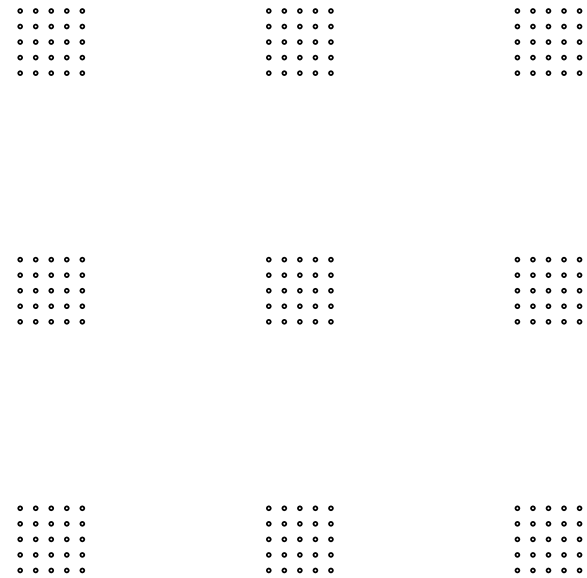
Theorem

There is a reduction from $k \times k$ Grid Tiling with $\leq$ to k^2-Independent Set for unit disks. Consequently, Independent Set for unit disks is

- is W[1]-hard, and
- cannot be solved in time $f(k)n^{o(\sqrt{k})}$ for any function f.

Reduction to unit disks

(5,1) (1,2) (3,3)	(4,3) (3,2)	(2,3) (2,5)
(2,1) (5,5) (3,5)	(4,2) (5,3)	(5,1) (3,2)
(5,1) (2,2) (5,3)	(2,1) (4,2)	(3,1) (3,2) (3,3)

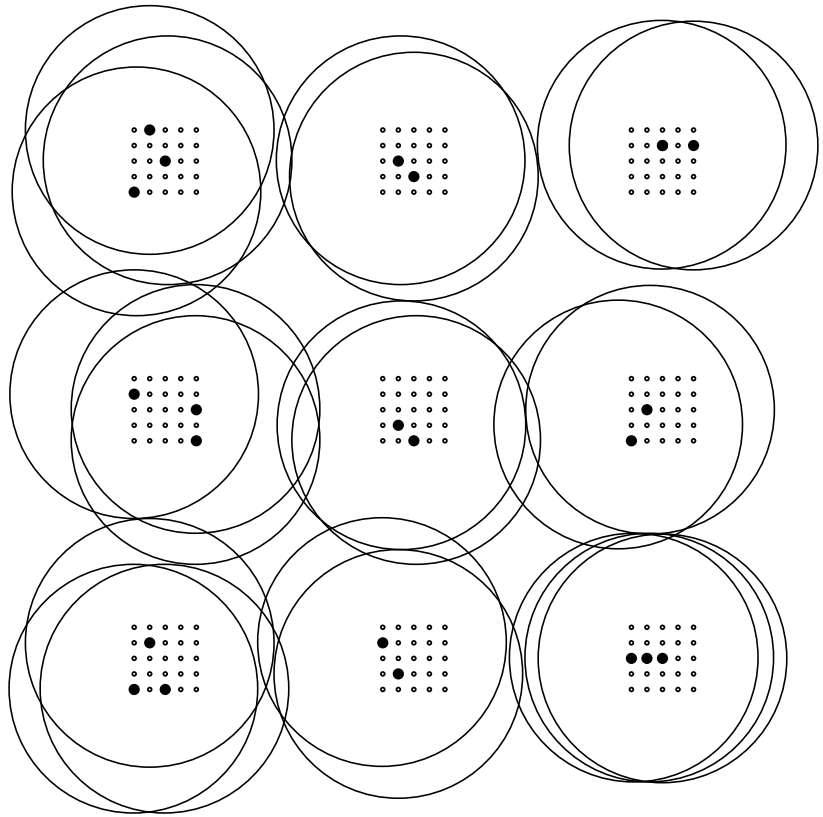


Every pair is represented by a unit disk in the plane.

$\leq$ relation between coordinates $\iff$ disks do not intersect.

Reduction to unit disks

(5,1) (1,2) (3,3)	(4,3) (3,2)	(2,3) (2,5)
(2,1) (5,5) (3,5)	(4,2) (5,3)	(5,1) (3,2)
(5,1) (2,2) (5,3)	(2,1) (4,2)	(3,1) (3,2) (3,3)

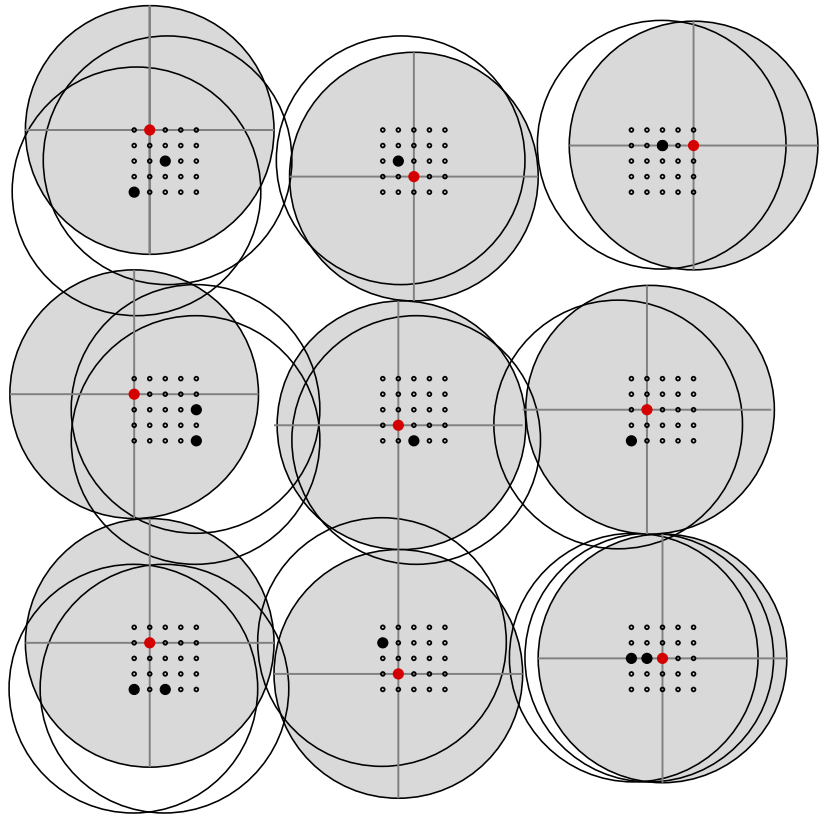


Every pair is represented by a unit disk in the plane.

$\leq$ relation between coordinates $\iff$ disks do not intersect.

Reduction to unit disks

(5,1) (1,2) (3,3)	(4,3) (3,2)	(2,3) (2,5)
(2,1) (5,5) (3,5)	(4,2) (5,3)	(5,1) (3,2)
(5,1) (2,2) (5,3)	(2,1) (4,2)	(3,1) (3,2) (3,3)



Every pair is represented by a unit disk in the plane.

$\leq$ relation between coordinates $\iff$ disks do not intersect.

Conclusions

- A robust understanding of why certain problems can be solved in time $2^{O(\sqrt{n})}$ etc. on planar graphs and why the square root is best possible.
- Lower bounds:
  - ETH assumption
  - Planar NP-hardness proofs have quadratic blow up.
  - Parameterized reductions for W[1]-hard problems.
  - Grid Tiling
- Upper bounds:
  - Treewidth, dynamic programming on tree decompositions.
  - Planar graphs have treewidth $O(\sqrt{n})$.
  - Planar Excluded Grid Theorem.
  - Shifting strategy for geometric problems.