Treewidth: Vol. 1

Dániel Marx

Lecture #7
June 19, 2020

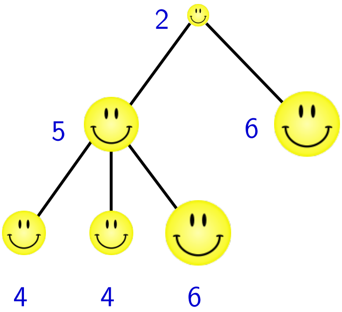
Treewidth

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

The Party Problem

Party Problem

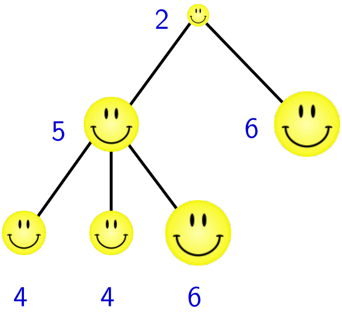
Problem: Invite some colleagues for a party.
Maximize: The total fun factor of the invited people.
Constraint: Everyone should be having fun.



The Party Problem

Party Problem

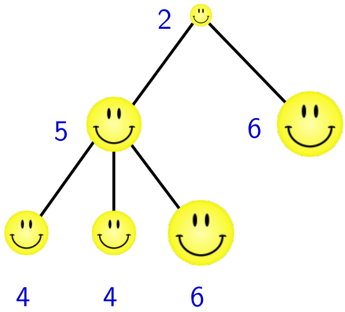
Problem: Invite some colleagues for a party.
Maximize: The total fun factor of the invited people.
Constraint: Everyone should be having fun.
Do not invite a colleague and their direct boss at the same time!



The Party Problem

Party Problem

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

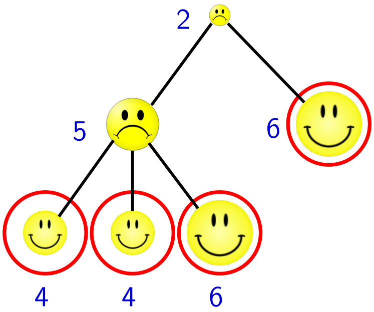


- **Input:** A tree with weights on the vertices.
- **Task:** Find an independent set of maximum weight.

The Party Problem

Party Problem

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



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

Solving the Party Problem

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

Subproblems:

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

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

Solving the Party Problem

Subproblems:

T_v: the subtree rooted at v.

$A[v]$: max. weight of an independent set in T_v

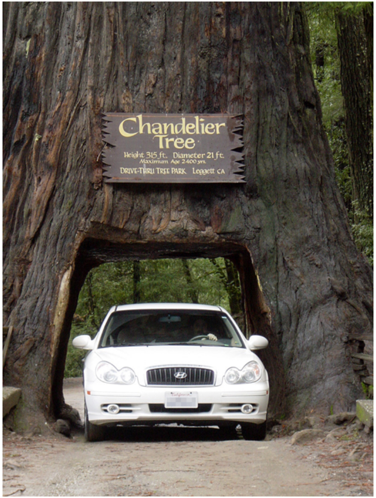
$B[v]$: max. weight of an independent set in T_v
that does not contain v

Recurrence:

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

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

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


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

Treewidth

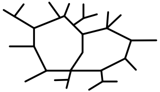
Generalizing trees

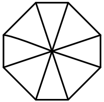
How could we define that a graph is "treelike"?

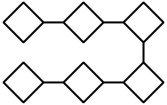
Generalizing trees

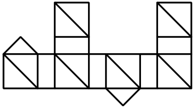
How could we define that a graph is “treelike”?

1. Number of cycles is bounded.


good


bad

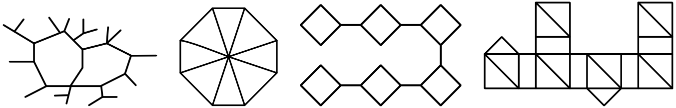

bad


bad

Generalizing trees

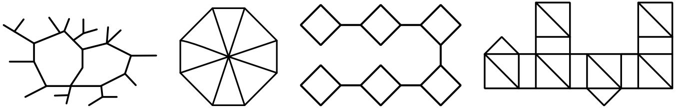
How could we define that a graph is "treelike"?

1. Number of cycles is bounded.



good bad bad bad

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

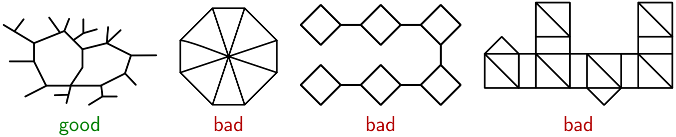


good good bad bad

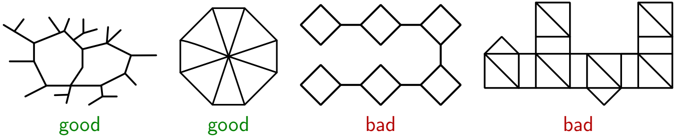
Generalizing trees

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

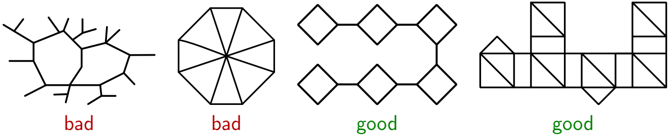
1. Number of cycles is bounded.



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



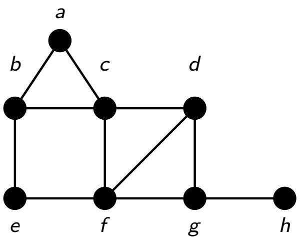
3. Bounded-size parts connected in a tree-like way.

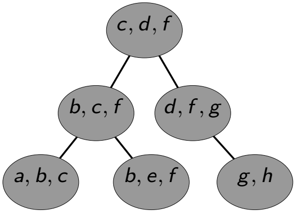


Treewidth — a measure of "tree-likeness"

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

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

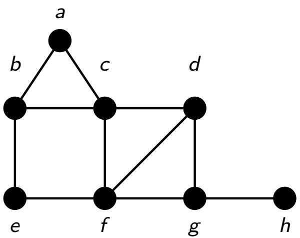


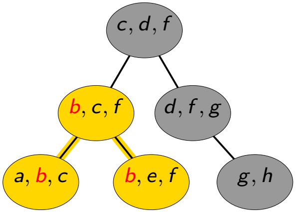


Treewidth — a measure of “tree-likeness”

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

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

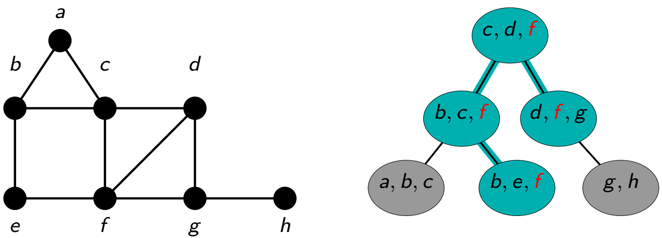




Treewidth — a measure of "tree-likeness"

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

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



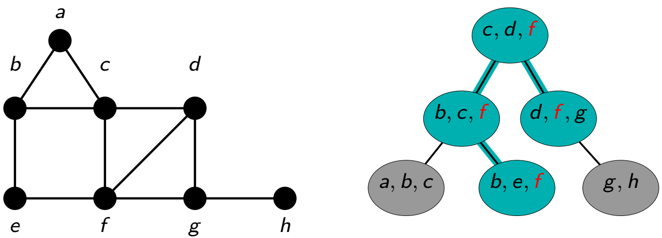
Treewidth — a measure of "tree-likeness"

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

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

Width of the decomposition: largest bag size -1.

treewidth: width of the best decomposition.



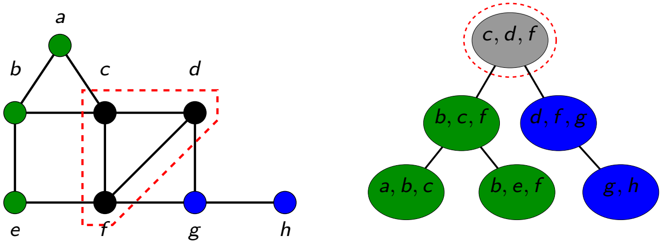
Treewidth — a measure of “tree-likeness”

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

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

Width of the decomposition: largest bag size -1.

treewidth: width of the best decomposition.



Each bag is a separator.

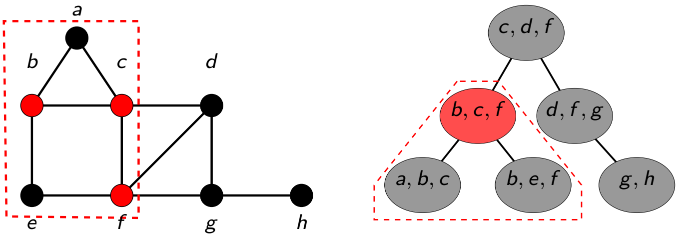
Treewidth — a measure of "tree-likeness"

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

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

Width of the decomposition: largest bag size -1.

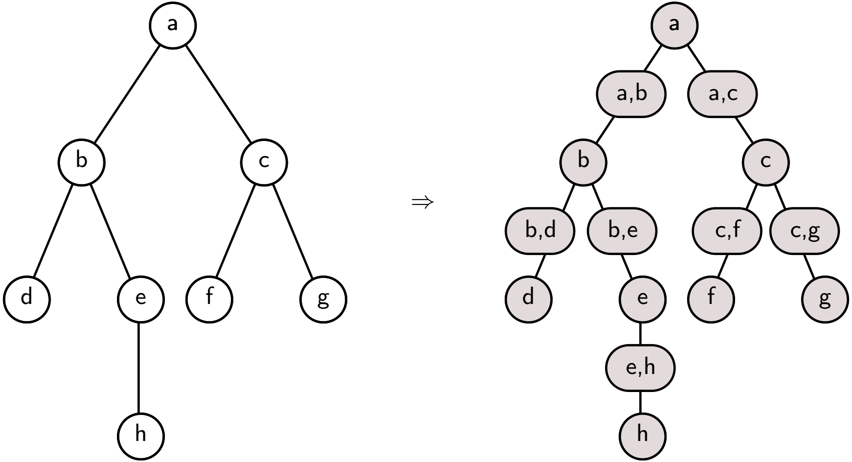
treewidth: width of the best decomposition.



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

Treewidth

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



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

Treewidth — outline

1. Basic algorithms
2. Combinatorial properties
3. Applications

Finding tree decompositions

Hardness:

Theorem [Arnborg, Corneil, Proskurowski 1987]

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

Fixed-parameter tractability:

Theorem [Bodlaender 1996]

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

Consequence:

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

Finding tree decompositions — approximately

Sometimes we can get better dependence on treewidth using approximation.

FPT approximation:

Theorem [Robertson and Seymour]

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

Polynomial-time approximation:

Theorem [Feige, Hajiaghayi, Lee 2008]

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

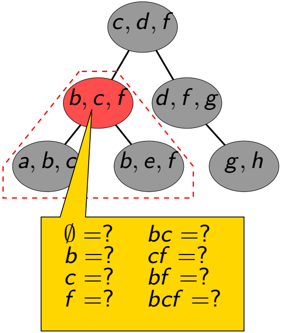
WEIGHTED MAX INDEPENDENT SET and treewidth

Theorem

Given a tree decomposition of width w, WEIGHTED MAX INDEPENDENT SET can be solved in time $O(2^w \cdot w^{O(1)} \cdot n)$.

B_x: vertices appearing in node x.

V_x: vertices appearing in the subtree rooted at x.



Generalizing our solution for trees:

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

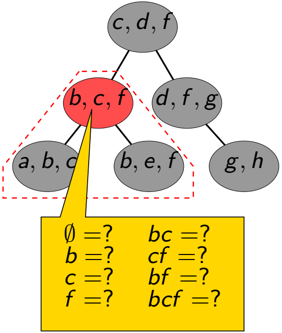
Theorem

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 $I \cap B_x = S$.

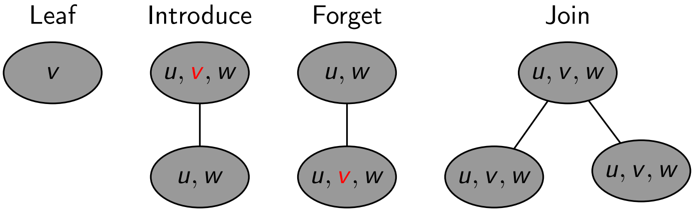
How to determine $M[x, S]$ if all the values are known for the children of x?

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

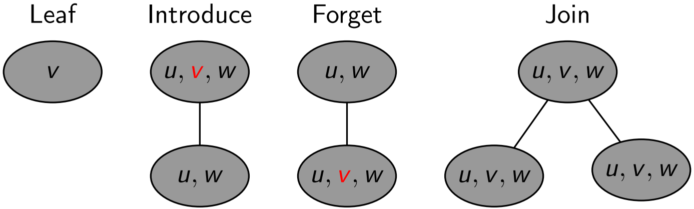
Theorem

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

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



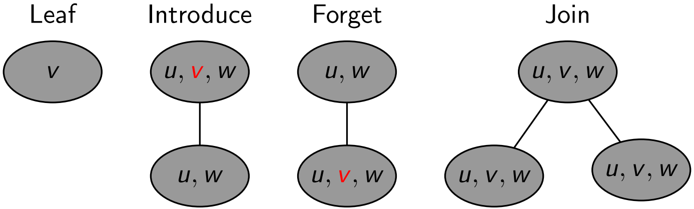
Weighted Max Independent Set
and nice tree decompositions

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

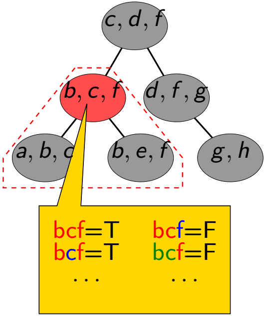
3-Coloring and tree decompositions

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.

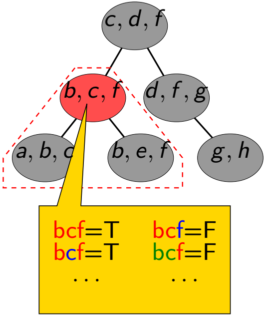


3-Coloring and tree decompositions

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.

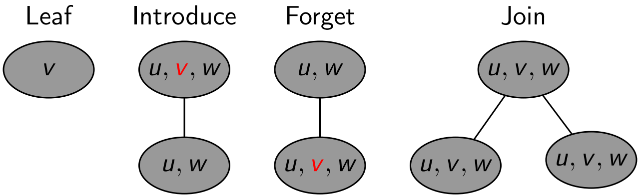


For every node x and coloring $c : B_x \to \{1,2,3\}$, we compute the Boolean value $E[x, c]$, which is true if and only if c can be extended to a proper 3-coloring of V_x.

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

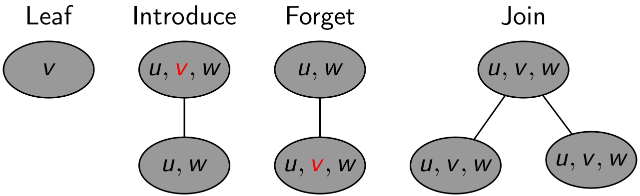
3-Coloring and nice tree decompositions

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



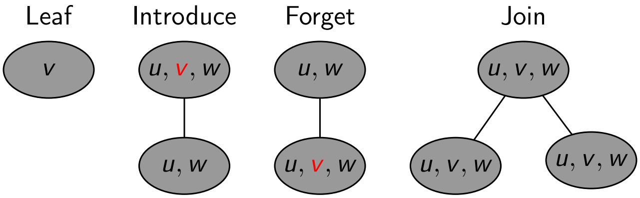
3-Coloring and nice tree decompositions

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



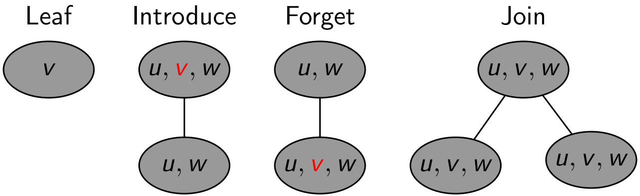
3-COLORING and nice tree decompositions

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



3-Coloring and nice tree decompositions

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



3-Coloring and nice tree decompositions

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

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

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

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

Vertex coloring

More generally:

Theorem

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

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

Theorem

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

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

Dominating Set and treewidth

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

B_x: vertices appearing in node x.

V_x: vertices appearing in the subtree rooted at x.

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

Dominating Set and treewidth

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

B_x: vertices appearing in node x.

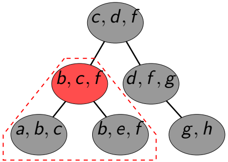
V_x: vertices appearing in the subtree rooted at x.

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

First try:

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

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



Dominating Set and treewidth

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

B_x: vertices appearing in node x.

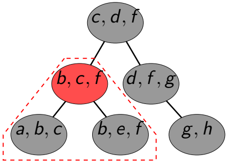
V_x: vertices appearing in the subtree rooted at x.

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

First try:

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

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



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

Dominating Set and treewidth

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

B_x: vertices appearing in node x.

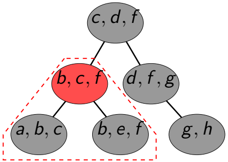
V_x: vertices appearing in the subtree rooted at x.

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

Second try:

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

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



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

DOMINATING SET and treewidth

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

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

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

Dominating Set and treewidth

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

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

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

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

Dominating Set and treewidth

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

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

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

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

DOMINATING SET and treewidth

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

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

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

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

DOMINATING SET and treewidth

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

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

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

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

Hamiltonian cycle and treewidth

Theorem

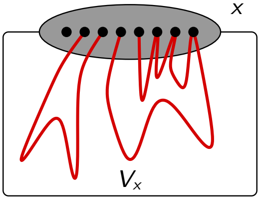
Given a tree decomposition of width w, HAMILTONIAN CYCLE can be solved in time $w^{O(w)} \cdot n$.

B_x: vertices appearing in node x.

V_x: vertices appearing in the subtree rooted at x.

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

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



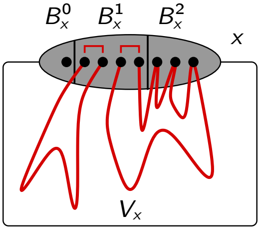
Hamiltonian cycle and treewidth

Theorem

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

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

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



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

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

No. of subproblems (B_x^0, B_x^1, B_x^2, M) for node x: at most $3^w \cdot w^w$.
For each subproblem, we have to determine if there is a set of paths with this pattern.

Other problems

There are other problems where the natural DP needs to keep track of $w^{O(w)}$ possibilties of a partition.

Theorem

Given a tree decomposition of width w, there are $w^{O(w)} \cdot n$ time algorithms for

- Hamiltonian cycle
- Steiner Tree
- Cycle Packing
- ...

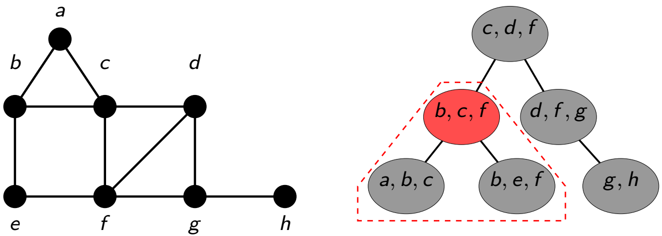
Treewidth — a measure of "tree-likeness"

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.



Monadic Second Order Logic

Extended Monadic Second Order Logic (EMSO)

A logical language on graphs consisting of the following:

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

Example:
The formula

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

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

Monadic Second Order Logic

Extended Monadic Second Order Logic (EMSO)

A logical language on graphs consisting of the following:

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

Example:
The formula

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

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

Courcelle's Theorem

Courcelle's Theorem

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

Courcelle's Theorem

Courcelle's Theorem

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

Courcelle's Theorem

Courcelle's Theorem

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

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

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

Using Courcelle's Theorem

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

Using Courcelle's Theorem

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

3-Coloring

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

Using Courcelle's Theorem

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

3-Coloring

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

Hamiltonian Cycle

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

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

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

Using Courcelle's Theorem

Three ways of using Courcelle's Theorem:

1. The problem can be described by a single formula (e.g, 3-Coloring, Hamiltonian Cycle).

   $\Rightarrow$ Problem can be solved in time $f(w) \cdot n$ for graphs of treewidth at most w, i.e., FPT parameterized by treewidth.

Using Courcelle's Theorem

Three ways of using Courcelle's Theorem:

1. The problem can be described by a single formula (e.g, 3-Coloring, Hamiltonian Cycle).

   $\Rightarrow$ Problem can be solved in time $f(w) \cdot n$ for graphs of treewidth at most w, i.e., FPT parameterized by treewidth.

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

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

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

   $\Rightarrow$ Problem can be solved in time $f(k, w) \cdot n$ for graphs of treewidth w, i.e., FPT parameterized with combined parameter k and treewidth w.

Using Courcelle's Theorem

Three ways of using Courcelle's Theorem:

1. The problem can be described by a single formula (e.g, 3-Coloring, Hamiltonian Cycle).

   $\Rightarrow$ Problem can be solved in time $f(w) \cdot n$ for graphs of treewidth at most w, i.e., FPT parameterized by treewidth.

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

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

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

   $\Rightarrow$ Problem can be solved in time $f(k, w) \cdot n$ for graphs of treewidth w, i.e., FPT parameterized with combined parameter k and treewidth w.

3. Optimization version: find largest set X such that...

Subgraph Isomorphism

Subgraph Isomorphism

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

Subgraph Isomorphism

Subgraph Isomorphism

Input: graphs H and G

Find: a subgraph of G isomorphic to H.

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

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

Subgraph Isomorphism

Subgraph Isomorphism

Input: graphs H and G

Find: a subgraph of G isomorphic to H.

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

Theorem

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

MSO on words

Theorem [Büchi, Elgot, Trakhtenbrot 1960]

If a language $L \subseteq \Sigma^*$ can be defined by an MSO formula ϕ using the relation $<$, then L is regular.

Example: a^*bc^* is defined by

$$\exists x : P_b(x) \wedge (\forall y : (y < x) \to P_a(y)) \wedge (\forall y : (x < y) \to P_c(y)).$$

MSO on words

Theorem [Büchi, Elgot, Trakhtenbrot 1960]

If a language $L \subseteq \Sigma^*$ can be defined by an MSO formula ϕ using the relation $<$, then L is regular.

Example: a^*bc^* is defined by

$$\exists x : P_b(x) \wedge (\forall y : (y < x) \rightarrow P_a(y)) \wedge (\forall y : (x < y) \rightarrow P_c(y)).$$

We prove a more general statement for formulas $\phi(w, X_1, \ldots, X_k)$ and words over $\Sigma \cup \{0,1\}^k$, where X_i is a subset of positions of w.

Induction over the structure of ϕ:

- FSM for $\neg\phi(w)$, given FSM for $\phi(w)$.
- FSM for $\phi_1(w) \wedge \phi_2(w)$, given FSMs for $\phi_1(w)$ and $\phi_2(w)$.
- FSM for $\exists X \phi(w, X)$, given FSM for $\phi(w, X)$.
- etc.

MSO on words

Theorem [Büchi, Elgot, Trakhtenbrot 1960]

If a language $L \subseteq \Sigma^*$ can be defined by an MSO formula ϕ using the relation $<$, then L is regular.

Proving Courcelle's Theorem:

- Generalize from words to trees.
- A width-w tree decomposition can be interpreted as a tree over an alphabet of size $f(w)$.
- Formula $\Rightarrow$ tree automata.

Running times

We have seen:

- Independent Set: 2^w
- Vertex Cover: 2^w
- Dominating Set: 3^w
- 3-Coloring: 3^w
- Vertex Coloring: $2^{O(w \log w)}$
- Hamiltonian Cycle: $2^{O(w \log w)}$

Can we improve on any of these running times?

Running times

We have seen:

- Independent Set: 2^w
- Vertex Cover: 2^w
- Dominating Set: 3^w
- 3-Coloring: 3^w
- Vertex Coloring: $2^{O(w \log w)}$
- Hamiltonian Cycle: $2^{O(w \log w)}$

Can we improve on any of these running times?

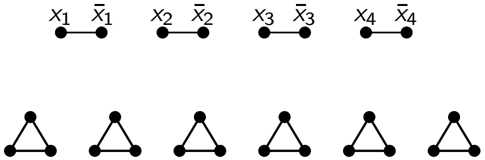
Hamiltonian Cycle can be improved to $2^{O(w)}$, but lower bounds show that the other algorithms are essentially optimal.

Lower bounds based on ETH

Exponential Time Hypothesis (ETH) + Sparsification Lemma

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

The textbook reduction from 3SAT to INDEPENDENT SET:

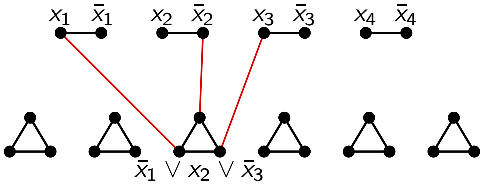


Lower bounds based on ETH

Exponential Time Hypothesis (ETH) + Sparsification Lemma

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

The textbook reduction from 3SAT to Independent Set:



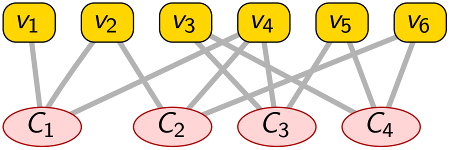
formula is satisfiable $\Leftrightarrow$ there is an independent set of size $n + 2m$

Lower bounds based on ETH

Exponential Time Hypothesis (ETH) + Sparsification Lemma

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

3SAT formula ϕ		Graph G
n variables	$\Rightarrow$	$O(n+m)$ vertices
m clauses		$O(n+m)$ edges

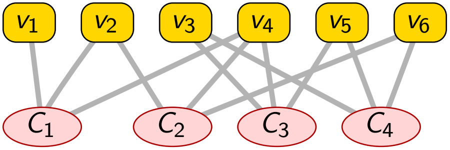


Lower bounds based on ETH

Exponential Time Hypothesis (ETH) + Sparsification Lemma

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

3SAT formula ϕ		Graph G
n variables	$\Rightarrow$	$O(n+m)$ vertices
m clauses		$O(n+m)$ edges



Corollary

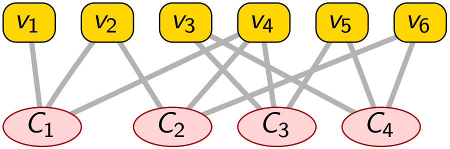
Assuming ETH, there is no $2^{o(n)}$ algorithm for Independent Set on an n-vertex graph.

Lower bounds based on ETH

Exponential Time Hypothesis (ETH) + Sparsification Lemma

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

3SAT formula ϕ		Graph G
n variables	$\Rightarrow$	$O(n+m)$ vertices
m clauses		$O(n+m)$ edges



Corollary

Assuming ETH, there is no $2^{o(w)} \cdot n^{O(1)}$ algorithm for Independent Set on graphs of treewidth w.

Lower bounds for treewidth

Similarly, assuming ETH, there is no $2^{o(w)} \cdot n^{O(1)}$ time algorithm for

- Independent Set
- Dominating Set
- Odd Cycle Transversal
- Hamiltonian Cycle
- ...

It is possible to show that there is no $2^{o(w \log w)} \cdot n^{O(1)}$ time algorithms for Vertex Coloring, Cycle Packing, and for some other problems.

Lower bounds for treewidth

Similarly, assuming ETH, there is no $2^{o(w)} \cdot n^{O(1)}$ time algorithm for

- Independent Set
- Dominating Set
- Odd Cycle Transversal
- Hamiltonian Cycle
- ...

It is possible to show that there is no $2^{o(w \log w)} \cdot n^{O(1)}$ time algorithms for Vertex Coloring, Cycle Packing, and for some other problems.

But can we show that there is no $(2-\epsilon)^w \cdot n^{O(1)}$ algorithm for Indpendent Set?

Lower bounds for treewidth

Similarly, assuming ETH, there is no $2^{o(w)} \cdot n^{O(1)}$ time algorithm for

- Independent Set
- Dominating Set
- Odd Cycle Transversal
- Hamiltonian Cycle
- ...

It is possible to show that there is no $2^{o(w \log w)} \cdot n^{O(1)}$ time algorithms for Vertex Coloring, Cycle Packing, and for some other problems.

But can we show that there is no $(2-\epsilon)^w \cdot n^{O(1)}$ algorithm for Indpendent Set?

ETH seems to be too weak for this:
2^w vs. $\sqrt{2}^w$ is just a polynomial difference!

ETH and SETH

Let $s_d = \inf\{c : d\text{-SAT}$ has a 2^{cn} algorithm$\}$

Let $s_\infty = \lim_{d \to \infty} s_d$.

ETH: $s_3 > 0$

SETH: $s_\infty = 1$.

ETH and SETH

Let $s_d = \inf\{c : d\text{-SAT has a } 2^{cn} \text{ algorithm}\}$

Let $s_\infty = \lim_{d\to\infty} s_d$.

ETH: $s_3 > 0$ SETH: $s_\infty = 1$.

In other words:

Strong Exponential-Time Hypothesis (SETH)

There is no $\epsilon > 0$ such that d-SAT for every d can be solved in time $(2-\epsilon)^n$.

ETH and SETH

Let $s_d = \inf\{c : d\text{-SAT}$ has a 2^{cn} algorithm$\}$

Let $s_\infty = \lim_{d\to\infty} s_d$.

ETH: $s_3 > 0$ SETH: $s_\infty = 1$.

In other words:

Strong Exponential-Time Hypothesis (SETH)

There is no $\epsilon > 0$ such that d-SAT for every d can be solved in time $(2-\epsilon)^n$.

Consequence of SETH

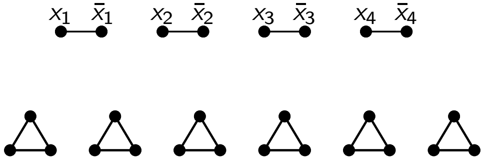
There is no $(2-\epsilon)^n \cdot m^{O(1)}$ time algorithm for SAT (with clauses of aribtrary length).

Lower bounds based on SETH

Strong Exponential-Time Hypothesis (SETH)

There is no $\epsilon > 0$ such that d-SAT for every d can be solved in time $(2-\epsilon)^n$.

The textbook reduction from 3SAT to Independent Set:

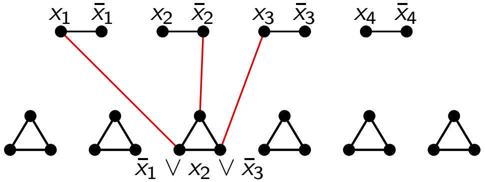


Lower bounds based on SETH

Strong Exponential-Time Hypothesis (SETH)

There is no $\epsilon > 0$ such that d-SAT for every d can be solved in time $(2-\epsilon)^n$.

The textbook reduction from 3SAT to Independent Set:



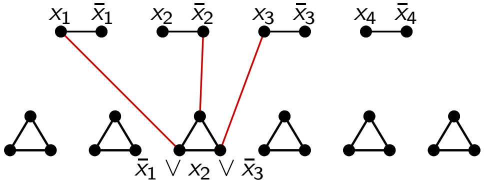
formula is satisfiable $\Leftrightarrow$ there is an independent set of size $n+m$

Lower bounds based on SETH

Strong Exponential-Time Hypothesis (SETH)

There is no $\epsilon > 0$ such that d-SAT for every d can be solved in time $(2-\epsilon)^n$.

The textbook reduction from 3SAT to INDEPENDENT SET:



Treewidth of the constructed graph is at most $2n+3$.

Lower bounds based on SETH

Strong Exponential-Time Hypothesis (SETH)

There is no $\epsilon > 0$ such that d-SAT for every d can be solved in time $(2-\epsilon)^n$.

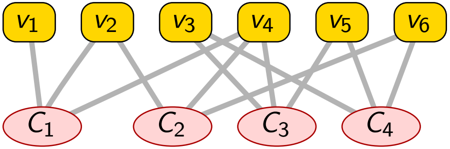
3SAT formula ϕ n variables m clauses	$\Rightarrow$	Graph G treewidth $2n+3$

Lower bounds based on SETH

Strong Exponential-Time Hypothesis (SETH)

There is no $\epsilon > 0$ such that d-SAT for every d can be solved in time $(2-\epsilon)^n$.

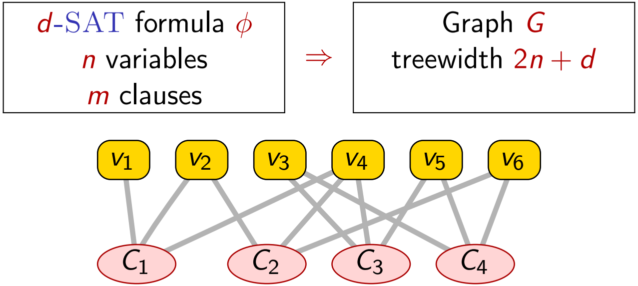
d-SAT formula ϕ n variables m clauses	$\Rightarrow$	Graph G treewidth $2n + d$



Lower bounds based on SETH

Strong Exponential-Time Hypothesis (SETH)

There is no $\epsilon > 0$ such that d-SAT for every d can be solved in time $(2-\epsilon)^n$.



Corollary

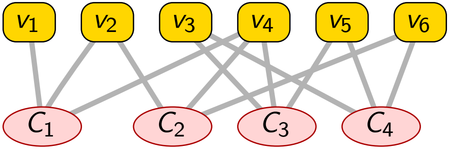
Assuming SETH, there is no $(2-\epsilon)^{w/2} \cdot n^{O(1)}$ algorithm for Independent Set for any $\epsilon > 0$.

Lower bounds based on SETH

Strong Exponential-Time Hypothesis (SETH)

There is no $\epsilon > 0$ such that d-SAT for every d can be solved in time $(2-\epsilon)^n$.

d-SAT formula ϕ n variables m clauses	$\Rightarrow$	Graph G treewidth $2n+d$



Corollary

Assuming SETH, there is no $(1.41-\epsilon)^w \cdot n^{O(1)}$ algorithm for Independent Set for any $\epsilon > 0$.

Better lower bound

We need a reduction of the following form for every d:

d-SAT formula ϕ n variables m clauses	$\Rightarrow$	Graph G treewidth $n + O_d(1)$

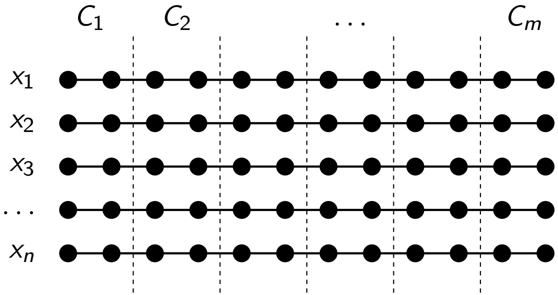
This would show:

Theorem

Assuming SETH, there is no $(2-\epsilon)^w \cdot n^{O(1)}$ algorithm for Independent Set for any $\epsilon > 0$.

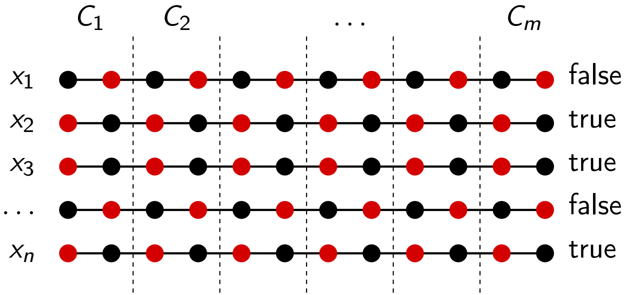
Better lower bound

d-SAT with n variables and m clauses $\Rightarrow$ n paths of $2m$ vertices.



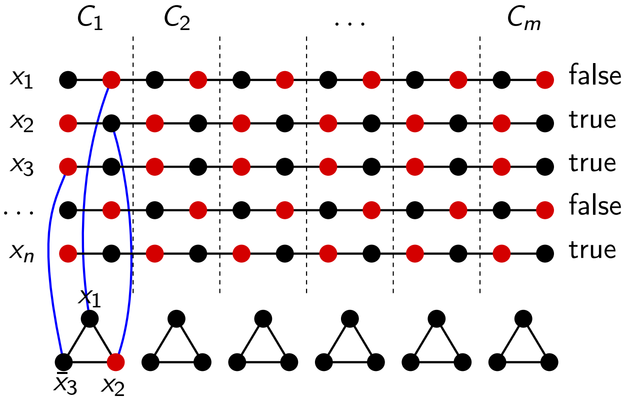
Better lower bound

d-SAT with n variables and m clauses $\Rightarrow$ n paths of $2m$ vertices.



Better lower bound

d-SAT with n variables and m clauses $\Rightarrow$ n paths of $2m$ vertices.

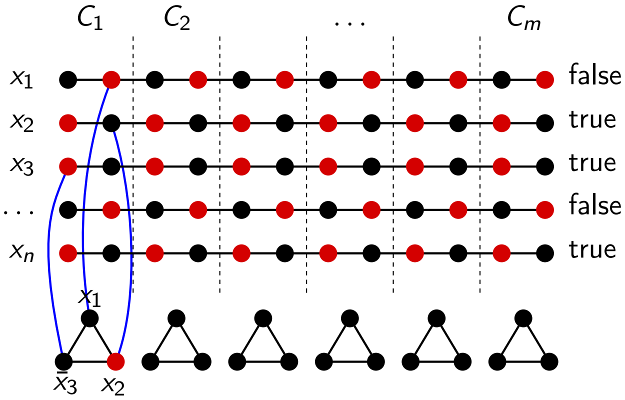


$(x_1 \vee x_2 \vee \bar{x}_3)$

Independent set of size $nm + m$ $\iff$ formula is satisfiable

Better lower bound

d-SAT with n variables and m clauses $\Rightarrow$ n paths of $2m$ vertices.



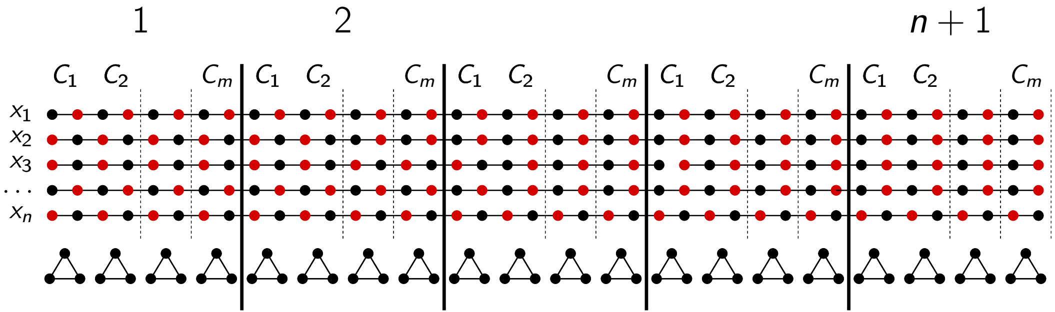
$(x_1 \vee x_2 \vee \bar{x}_3)$

Not difficult to show: treewidth is at most $n + d$

A problem

A path may start as "true" and switch to "false".

Simple solution: repeat the instance $n+1$ times.

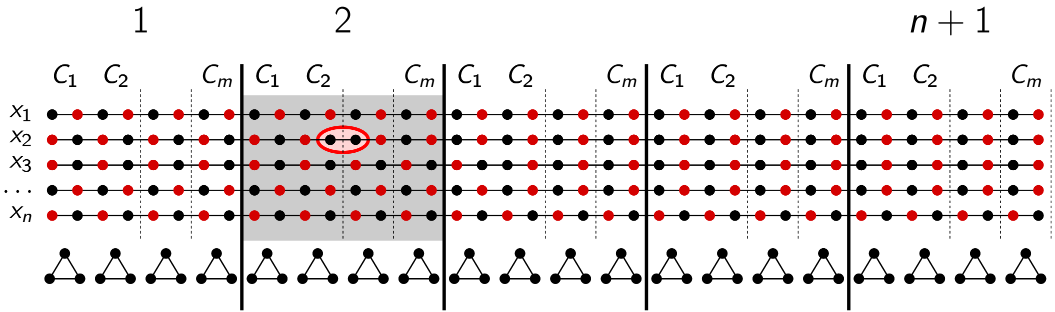


By the Pigeonhole Principle, there is a repetition where no switch occurs.

A problem

A path may start as "true" and switch to "false".

Simple solution: repeat the instance $n+1$ times.

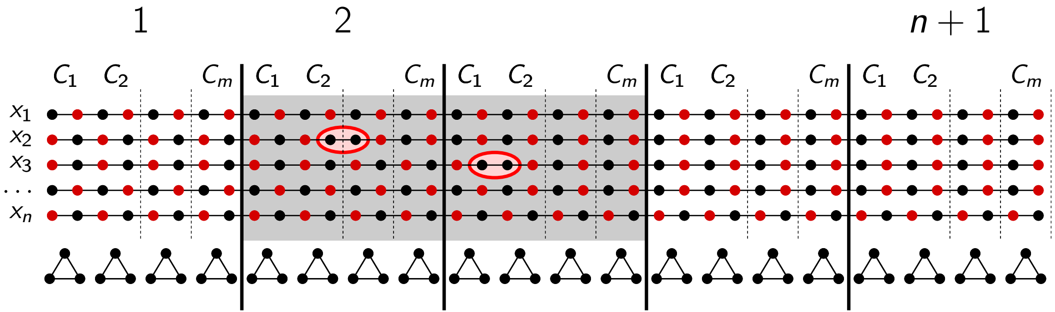


By the Pigeonhole Principle, there is a repetition where no switch occurs.

A problem

A path may start as "true" and switch to "false".

Simple solution: repeat the instance $n+1$ times.



By the Pigeonhole Principle, there is a repetition where no switch occurs.

Lower bound for INDEPENDENT SET

We have shown: Reduction from n-variable d-SAT to INDEPENDENT SET in a graph with treewidth $w = n + d$.

$$(2-\epsilon)^w \cdot n^{O(1)} \text{ algorithm for INDEPENDENT SET}$$
$$\Downarrow$$
$$(2-\epsilon)^n \cdot n^{O(1)} \text{ algorithm for } d\text{-SAT}$$

As this is true for any d, having such an algorithm for INDEPENDENT SET would violate SETH.

Theorem

Assuming SETH, there is no $(2-\epsilon)^w \cdot n^{O(1)}$ algorithm for INDEPENDENT SET for any $\epsilon > 0$.

Algorithms — overview

- Algorithms exploit the fact that a subtree communicates with the rest of the graph via a single bag.
- Key point: defining the subproblems.
- Courcelle's Theorem makes this process automatic for many problems.
- Lower bounds based on SETH can show the optimality of algorithms.
- There are notable problems that are easy for trees, but hard for bounded-treewidth graphs.

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.