Structural complexity of CSPs: the role of treewidth and its generalizations

Dániel Marx
Tel Aviv University, Israel
`dmarx@cs.bme.hu`

October 26, 2009

Dagstuhl Seminar 09441

Overview

Main message:

Tree-structured problems are easy to solve.

- If the graph/hypergraph describing the constraint of the variables has “simple structure,” then the problem is easier to solve.
- Simple structure usually means treelike structure.

Overview

- Part 1: Trees, treewidth and their algorithmic consequences
- Part 2: Treewidth and lower bounds
- Part 3: Width notions for hypergraphs

Overview

- Part 1: Trees, treewidth and their algorithmic consequences
- Part 2: Treewidth and lower bounds
- Part 3: Width notions for hypergraphs

The Party Problem

PARTY PROBLEM

Problem: Invite some colleagues for a party.

Maximize: The total fun factor of the invited people.

Constraint: Everyone should be having fun.

The Party Problem

PARTY PROBLEM

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

The Party Problem

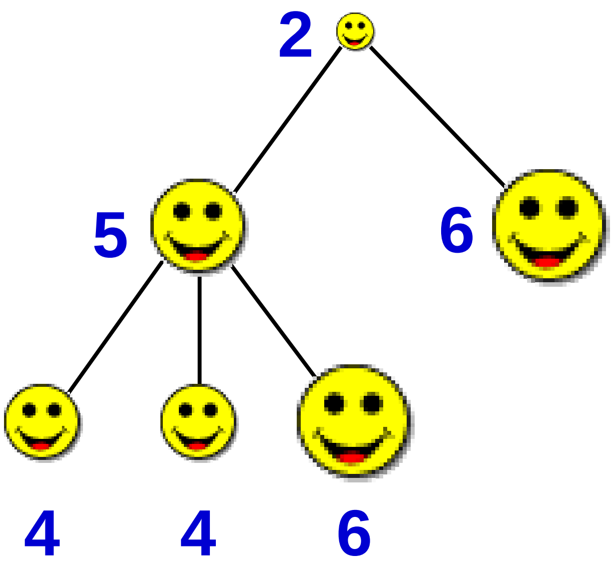
PARTY PROBLEM

Problem: Invite some colleagues for a party.

Maximize: The total fun factor of the invited people.

Constraint: Everyone should be having fun.

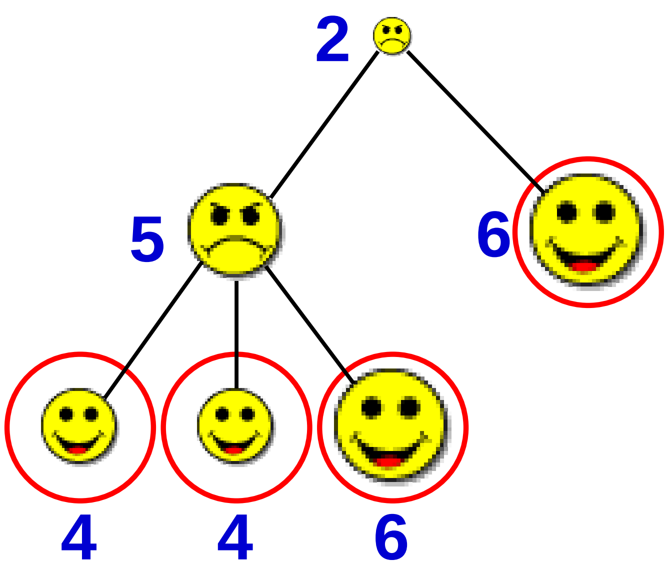
Do not invite a colleague and his direct boss at the same time!



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

The Party Problem

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



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

Solving the Party Problem

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

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

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

Solving the Party Problem

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

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

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

Method:
Assume $v_1, \ldots, v_k$ 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).

Constraint Satisfaction Problems (CSP)

A CSP instance is given by describing the

- variables,
- domain of the variables,
- constraints on the variables.

Task: Find an assignment that satisfies every constraint.

$$I = C_1(x_2, x_1, x_3) \wedge C_2(x_4, x_3) \wedge C_3(x_1, x_4, x_2)$$

Later: equivalent formulation as the homomorphism problem of relational structures.

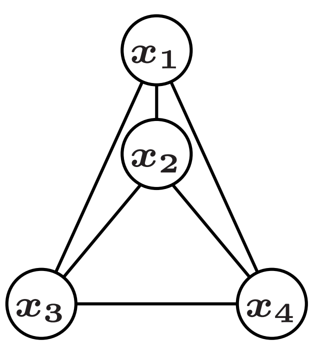
Graphs and hypergraphs related to CSP

Gaifman/primal graph: vertices are the variables, two variables are adjacent if they appear in a common constraint.

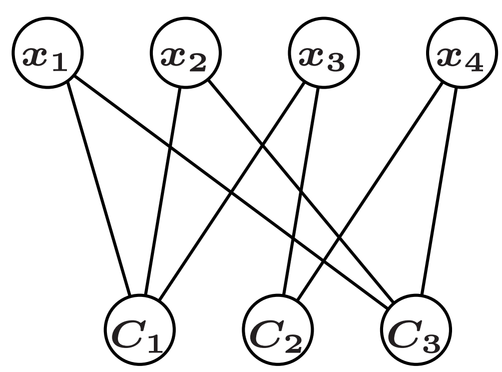
Incidence graph: bipartite graph, vertices are the variables and constraints.

Hypergraph: vertices are the variables, constraints are the hyperedges.

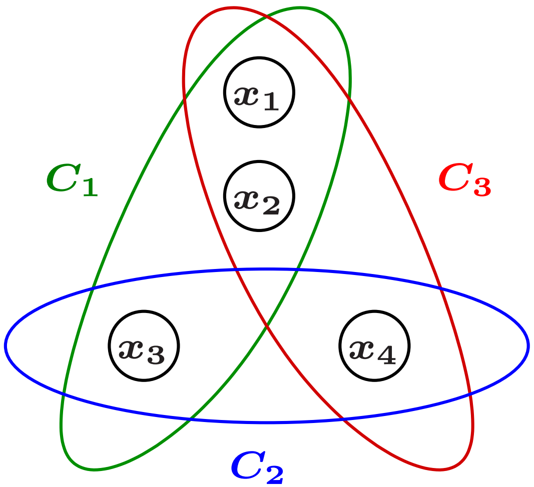
$$I = C_1(x_2, x_1, x_3) \wedge C_2(x_4, x_3) \wedge C_3(x_1, x_4, x_2)$$



Primal graph



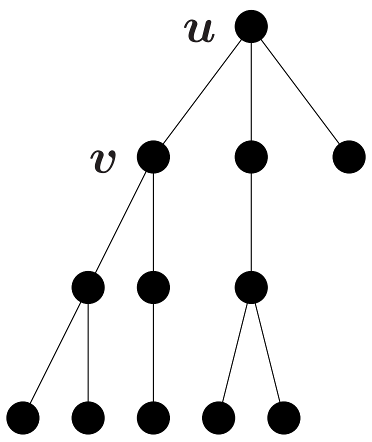
Incidence graph



Hypergraph

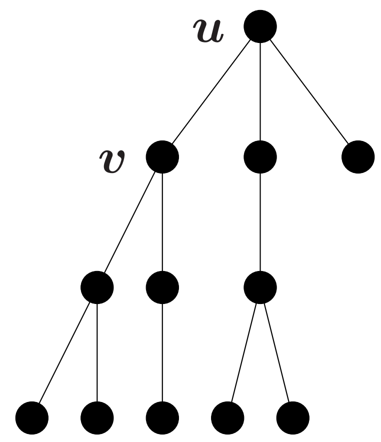
Tree-shaped CSP

Fact: Binary CSP is polynomial-time solvable restricted to instances whose primal graphs are trees.



Tree-shaped CSP

Fact: Binary CSP is polynomial-time solvable restricted to instances whose primal graphs are trees.

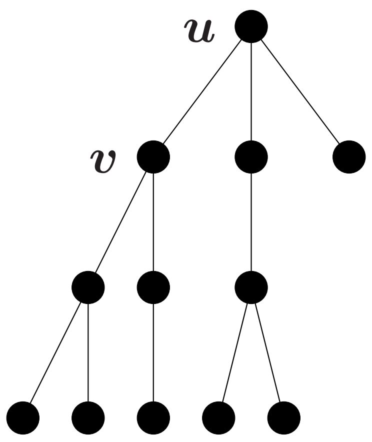


Proof 1: Dynamic programming. For $v \in V$, $d \in D$, let $x[v, d] =$ true if there is a partial solution on the subtree rooted at v such that variable v has value d.

The leaves are trivial. If the table is ready for the children of v, then computing $x[v, d]$ is easy.

Tree-shaped CSP

Fact: Binary CSP is polynomial-time solvable restricted to instances whose primal graphs are trees.

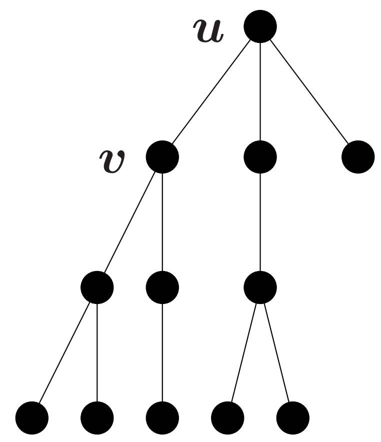


Proof 2: Arc consistency algorithm. If there is a constraint on (u, v) such that value d cannot appear on u, then remove every pair from every constraint on v that gives value d to u. Repeat.

Claim: When the algorithm stops, either every constraint is empty, or there is a solution.

Tree-shaped CSP

Fact: Binary CSP is polynomial-time solvable restricted to instances whose primal graphs are trees.



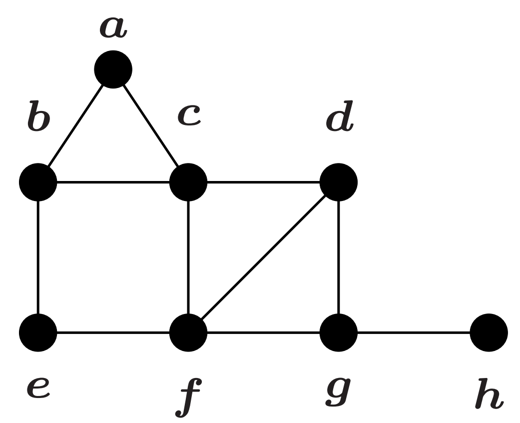
Can we generalize these ideas to wider classes of graphs?

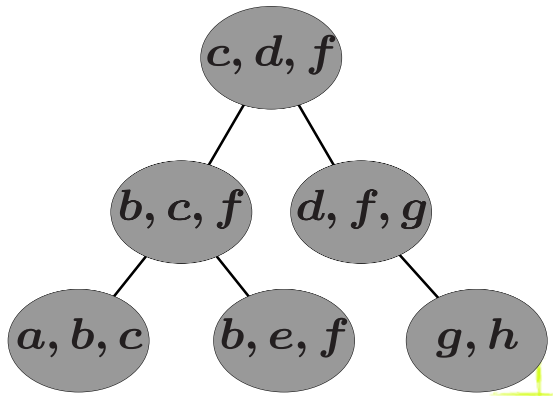
Treewidth

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

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

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



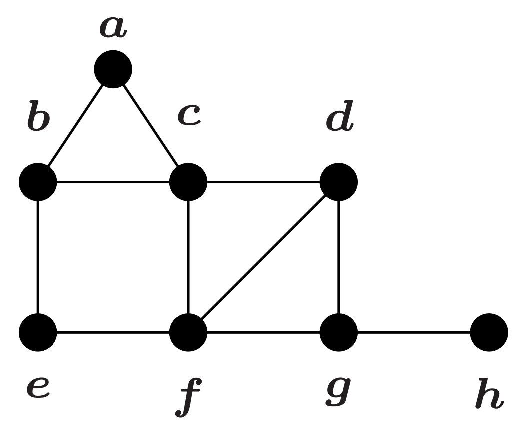


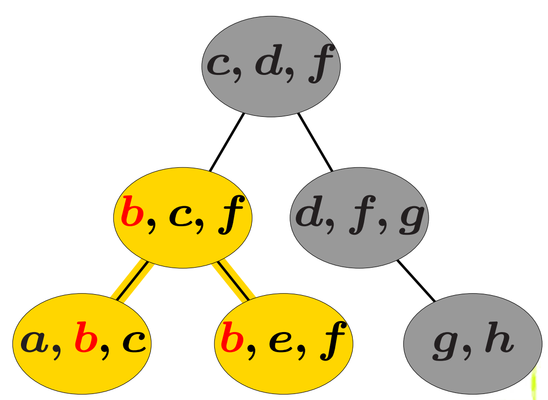
Treewidth

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

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

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



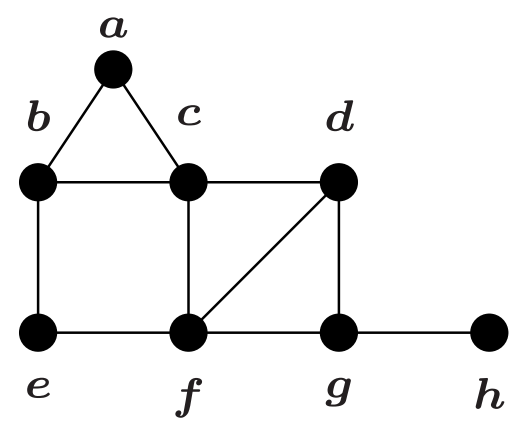


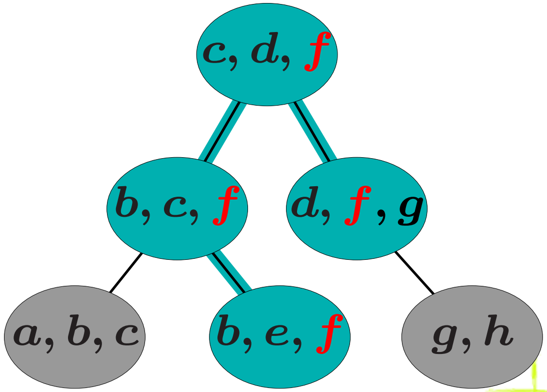
Treewidth

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

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

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



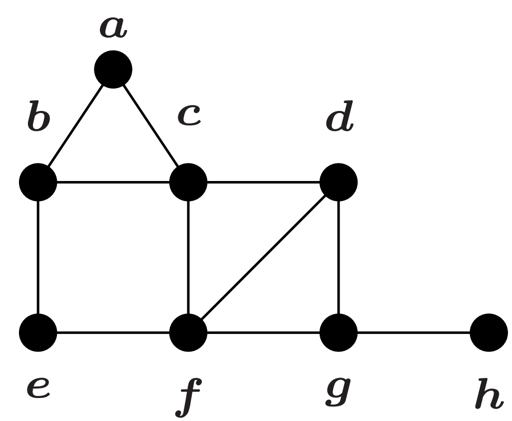


Treewidth

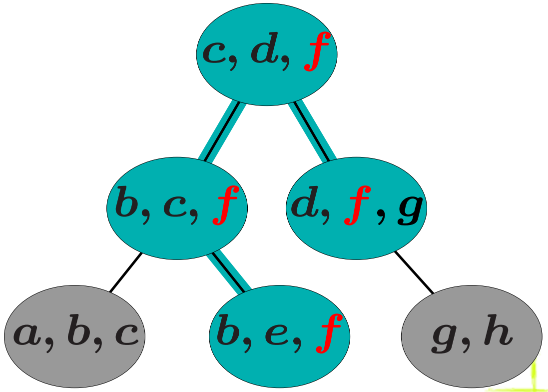
Treewidth: A measure of how "tree-like" the graph is. (Introduced by Robertson and Seymour.)

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

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



Width of the decomposition: size of the largest bag minus 1.



treewidth: width of the best decomposition.

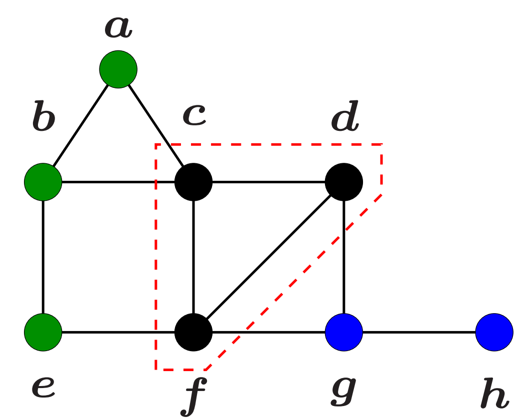
Fact: treewidth $= 1$ $\iff$ graph is a forest

Treewidth

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

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

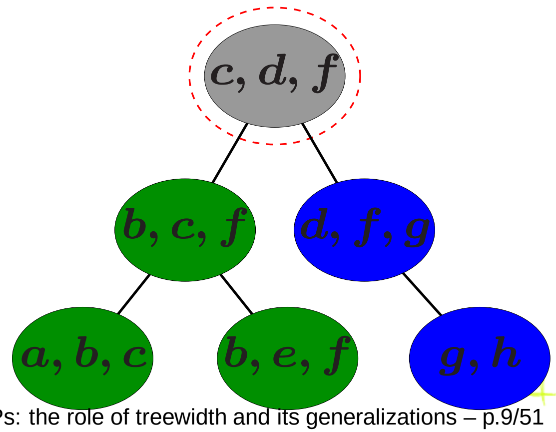
1. If u and v are neighbors, then there is a bag containing both of them.
2. For every vertex v, the bags containing v form a connected subtree.



Width of the decomposition: size of the largest bag minus 1.

treewidth: width of the best decomposition.

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

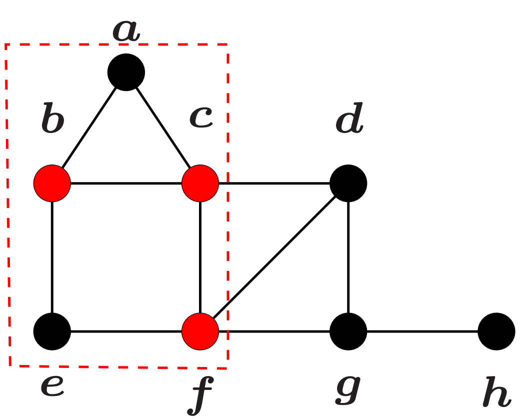


Treewidth

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

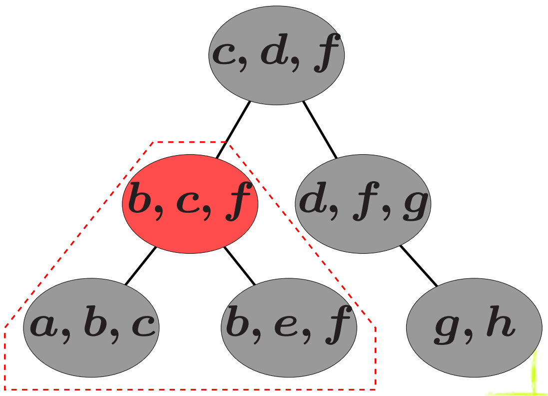
Tree decomposition: Vertices are arranged in a tree structure satisfying the following properties:

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



Width of the decomposition: size of the largest bag minus 1.

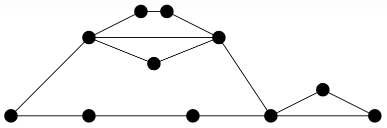
treewidth: width of the best decomposition.



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

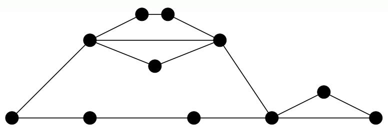
Properties of treewidth

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

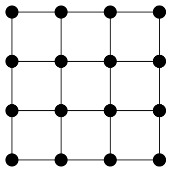


Properties of treewidth

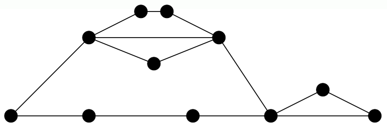
Fact: treewidth $\leq 2 \iff$ graph is subgraph of a series-parallel graph



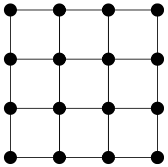
Fact: For every $k \geq 2$, the treewidth of the $k \times k$ grid is exactly k.



Properties of treewidth



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



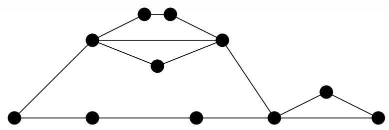
Fact: For every $k \geq 2$, the treewidth of the $k \times k$ grid is exactly k.

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

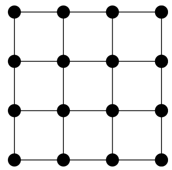
$\Rightarrow$ If F is a **minor** of G, then the treewidth of F is at most the treewidth of G.

Properties of treewidth

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



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



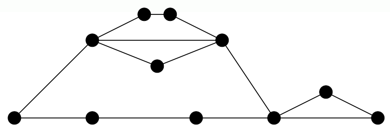
Fact: Treewidth does not increase if we delete edges, delete vertices, or contract edges.

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

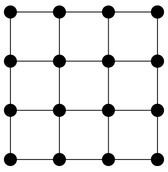
Fact: [Excluded Grid Theorem] If the treewidth of G is at least $k^{4k^2(k+2)}$, then G has a $k \times k$ grid minor.

Properties of treewidth

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



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



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

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

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

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

$\Rightarrow$ In the primal graph of a CSP instance, the scope of each constraint is a clique, hence it is fully contained in a bag.

Bounded treewidth graphs

Many problems are polynomial-time solvable for bounded treewidth graphs:

- VERTEX COLORING
- EDGE COLORING
- HAMILTONIAN CYCLE
- MAXIMUM CLIQUE
- VERTEX DISJOINT PATHS

Bounded treewidth graphs

Many problems are polynomial-time solvable for bounded treewidth graphs:

- VERTEX COLORING
- EDGE COLORING
- HAMILTONIAN CYCLE
- MAXIMUM CLIQUE
- VERTEX DISJOINT PATHS

Usually, if a problem can be solved on trees by bottom-up dynamic programming, then the same approach works for bounded treewidth graphs.

Some notable exceptions:

Fact: EDGE DISJOINT PATHS is NP-hard for graphs with treewidth 2.

Fact: LIST EDGE COLORING is NP-hard for graphs with treewidth 2.

Fact: STEINER FOREST is polynomial-time solvable for graphs with treewidth 2, but NP-hard for treewidth 3.

Treewidth and CSP

Fact: For every fixed k, CSP can be solved in polynomial time if the primal graph of the instance has treewidth at most k.

Two proofs:

- Using the k-consistency algorithm.
  Note: solves only the decision problem, does not give directly a solution.
- Using the tree decomposition.
  Note: requires a tree decomposition.

Consistency

A **partial solution** on $S \subseteq V$ is a mapping $f : S \longrightarrow D$ that satisfies every constraint whose scope is in S.

Definition: An instance is k**-consistent** if for any subsets $X \subset Y \subseteq V$ of size at most $k + 1$, every partial solution on X can be extended to Y.

Consistency

A **partial solution** on $S \subseteq V$ is a mapping $f : S \to D$ that satisfies every constraint whose scope is in S.

Definition: An instance is **k-consistent** if for any subsets $X \subset Y \subseteq V$ of size at most $k+1$, every partial solution on X can be extended to Y.

The k-Consistency algorithm generates a set of partial solutions that do no violate the consistency requirement.

k-Consistency

1. For every $S \subseteq V$ with $|S| \le k+1$, generate the list L_S of all partial solutions on S.
2. If for some $X \subseteq Y$, there is an $f \in L_X$ having no extension in L_Y, then remove f and every extension of f from the lists.
3. Repeat Step 2 until there are no further changes.

Consistency

k-Consistency

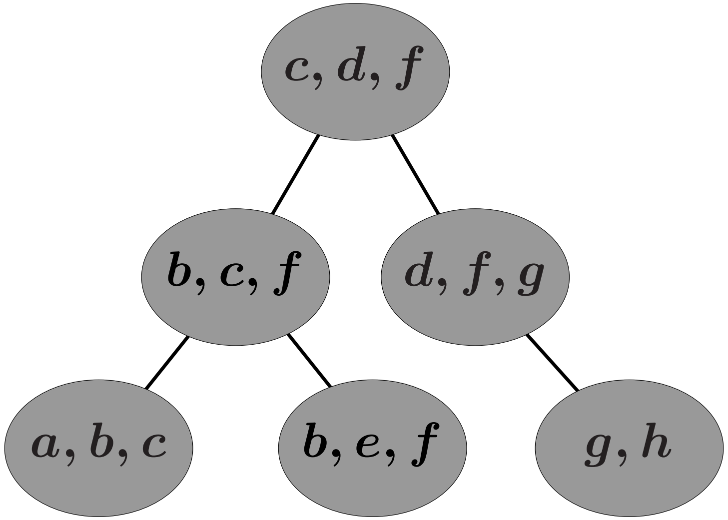
1. For every $S \subseteq V$ with $|S| \leq k+1$, generate the list L_S of all partial solutions on S.
2. If for some $X \subseteq Y$, there is an $f \in L_X$ having no extension in L_Y, then remove f and every extension of f from the lists.
3. Repeat Step 2 until there are no further changes.

Note:

- If an L_S is empty, then we can conclude that there is no solution.
- If $f \in L_Y$, then $f_{|X} \in L_X$ for every $X \subset Y$.
- The running time is polynomial for every fixed k: we manipulate subsets of size at most $k+1$ and the size of each L_S is at most $|D|^{k+1}$.

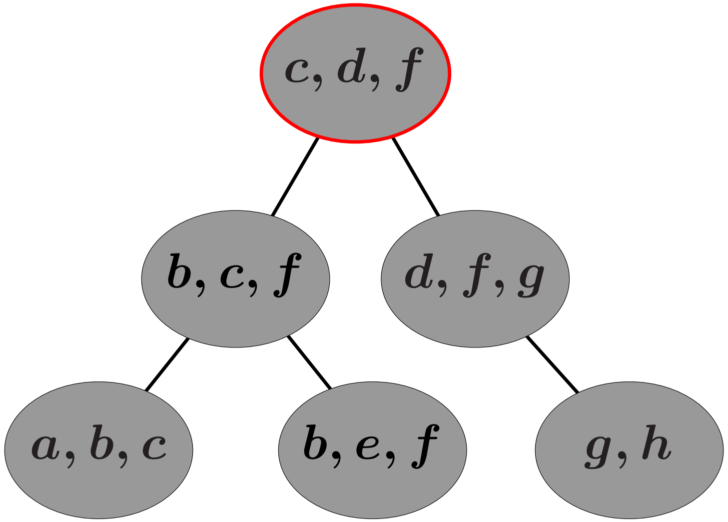
Consistency and treewidth

Fact: If the primal graph of the instance has treewidth at most k and the L_S's are not empty, then there is a solution.



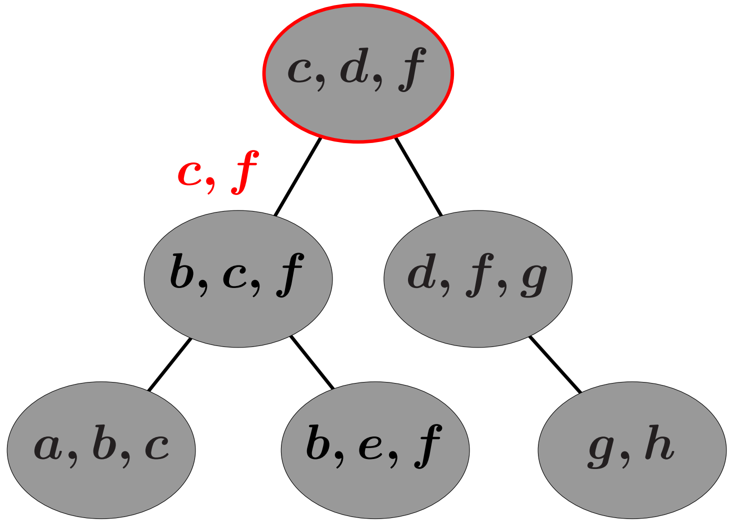
Consistency and treewidth

Fact: If the primal graph of the instance has treewidth at most k and the L_S's are not empty, then there is a solution.



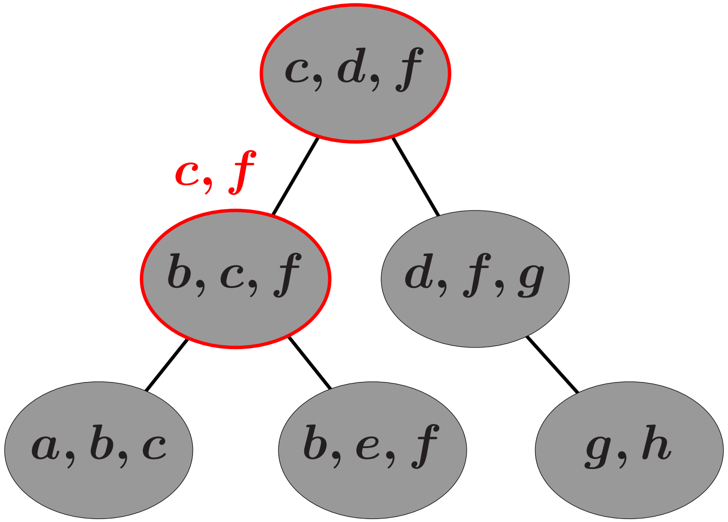
Consistency and treewidth

Fact: If the primal graph of the instance has treewidth at most k and the L_S's are not empty, then there is a solution.



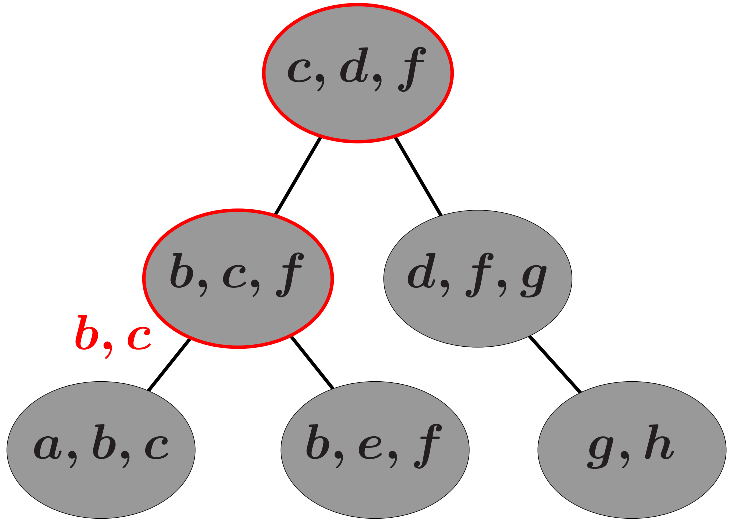
Consistency and treewidth

Fact: If the primal graph of the instance has treewidth at most k and the L_S's are not empty, then there is a solution.



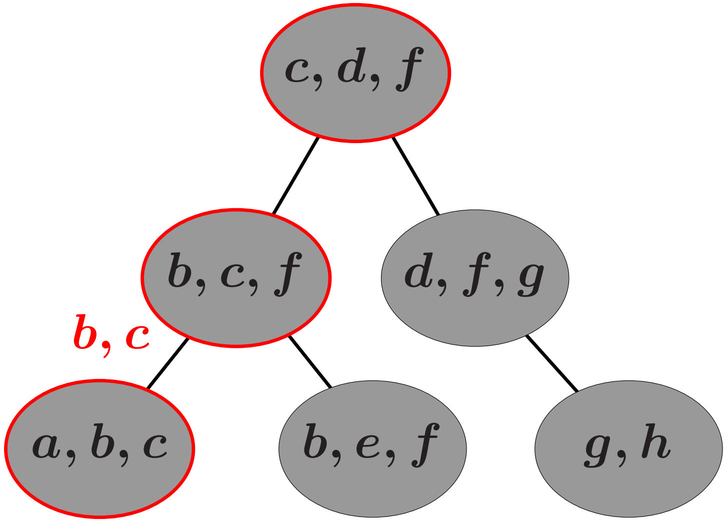
Consistency and treewidth

Fact: If the primal graph of the instance has treewidth at most k and the L_S's are not empty, then there is a solution.



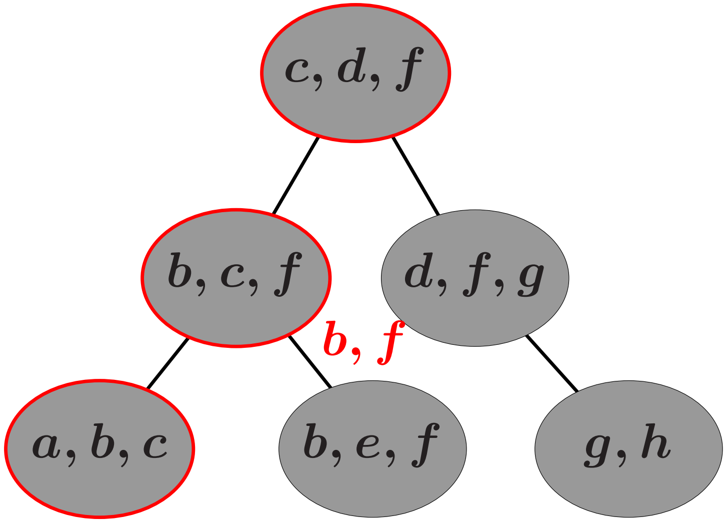
Consistency and treewidth

Fact: If the primal graph of the instance has treewidth at most k and the L_S's are not empty, then there is a solution.



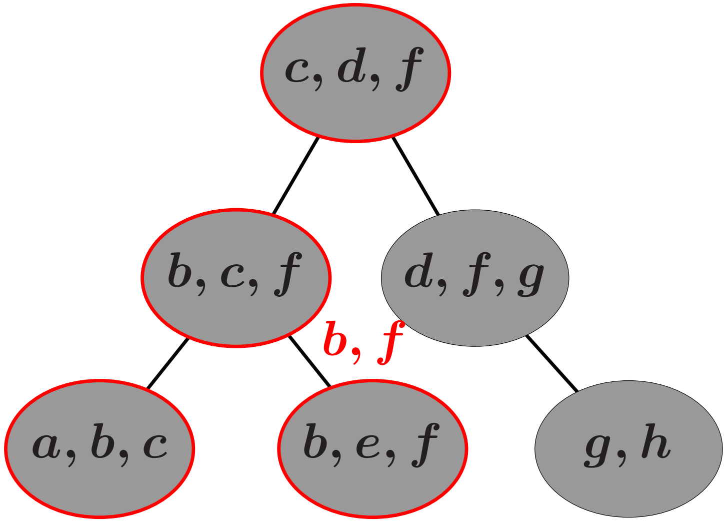
Consistency and treewidth

Fact: If the primal graph of the instance has treewidth at most k and the L_S's are not empty, then there is a solution.



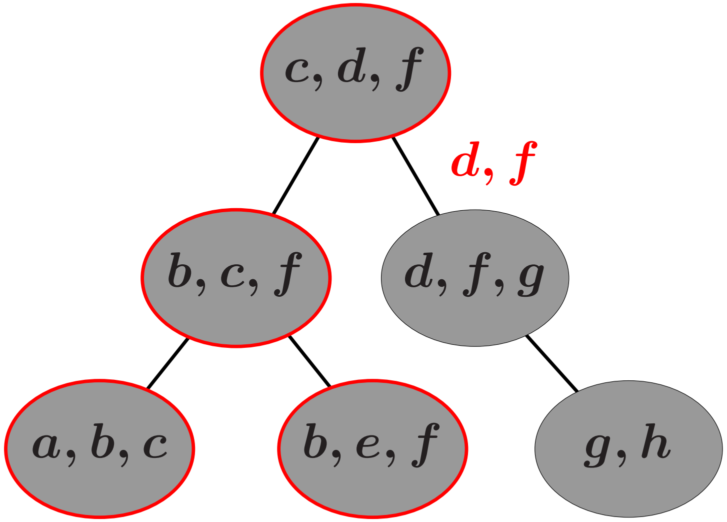
Consistency and treewidth

Fact: If the primal graph of the instance has treewidth at most k and the L_S's are not empty, then there is a solution.



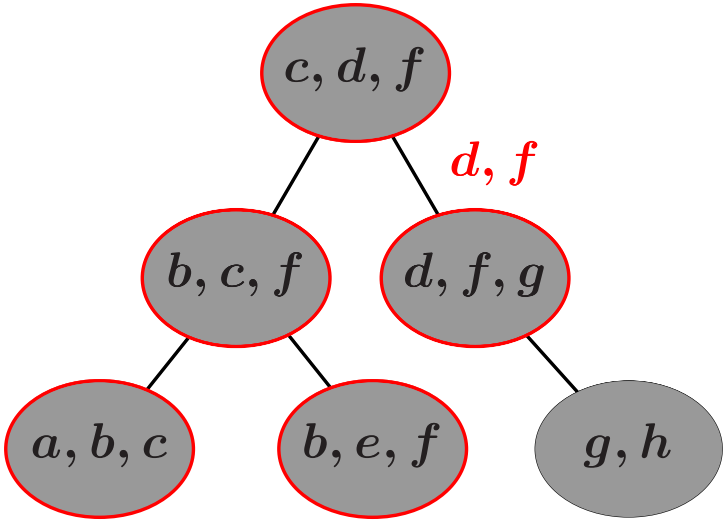
Consistency and treewidth

Fact: If the primal graph of the instance has treewidth at most k and the L_S's are not empty, then there is a solution.



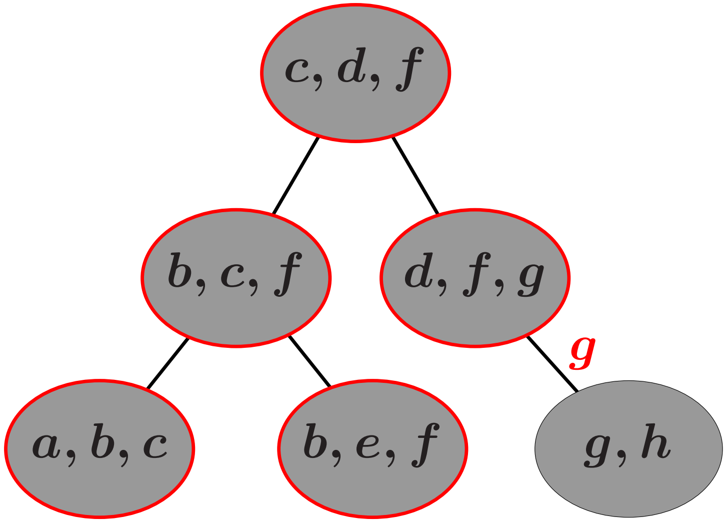
Consistency and treewidth

Fact: If the primal graph of the instance has treewidth at most k and the L_S's are not empty, then there is a solution.



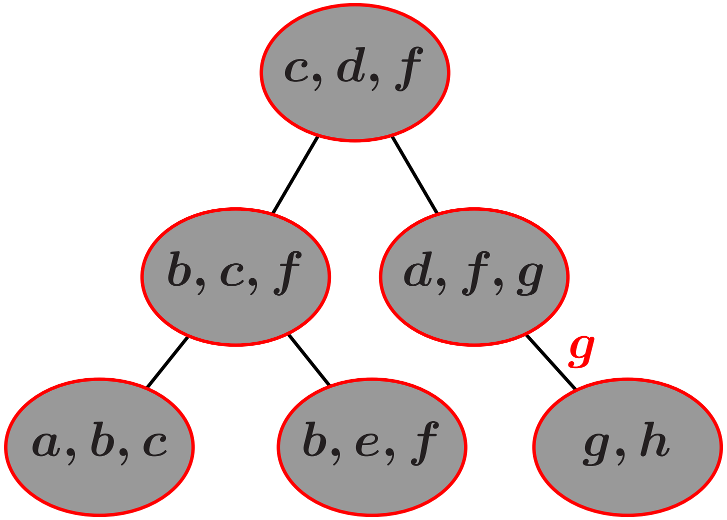
Consistency and treewidth

Fact: If the primal graph of the instance has treewidth at most k and the L_S's are not empty, then there is a solution.



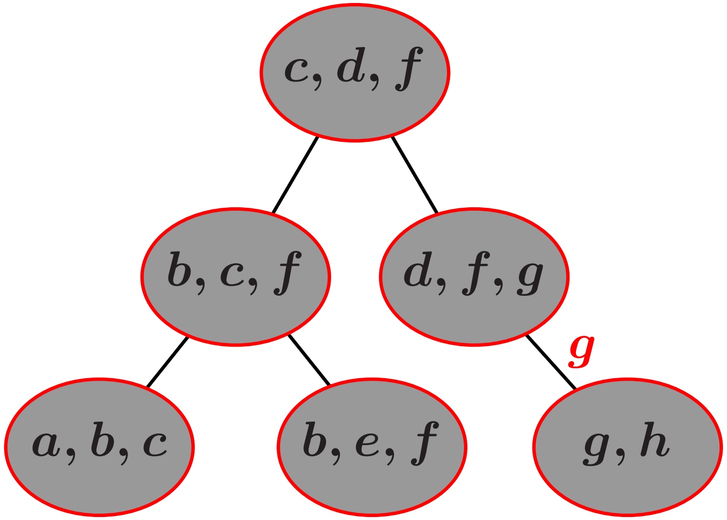
Consistency and treewidth

Fact: If the primal graph of the instance has treewidth at most k and the L_S's are not empty, then there is a solution.



Consistency and treewidth

Fact: If the primal graph of the instance has treewidth at most k and the L_S's are not empty, then there is a solution.



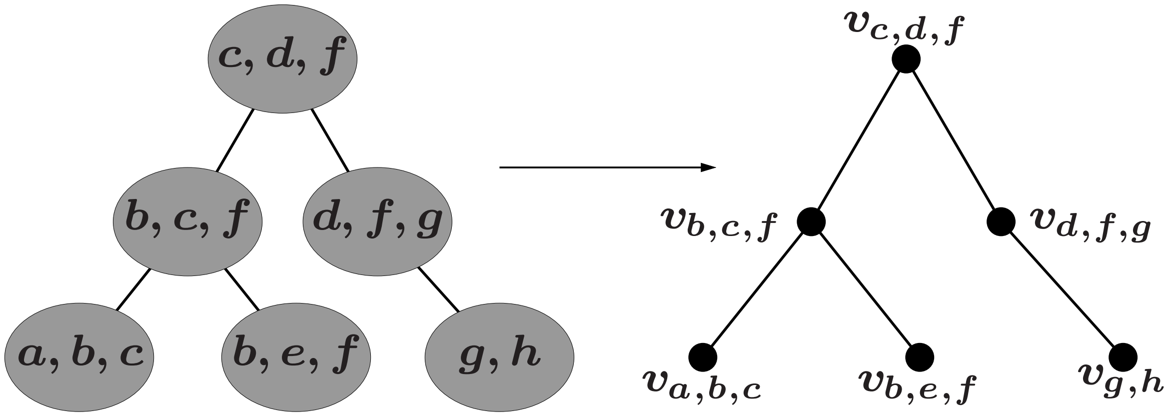
The properties of the tree decomposition ensure that

- each variable gets only a single value (connectedness property) and
- every constraint is satisfied (every clique appears in a bag).

Note: proof shows that a solution exists, but (unless we have a tree decomposition), does not show how to find one.

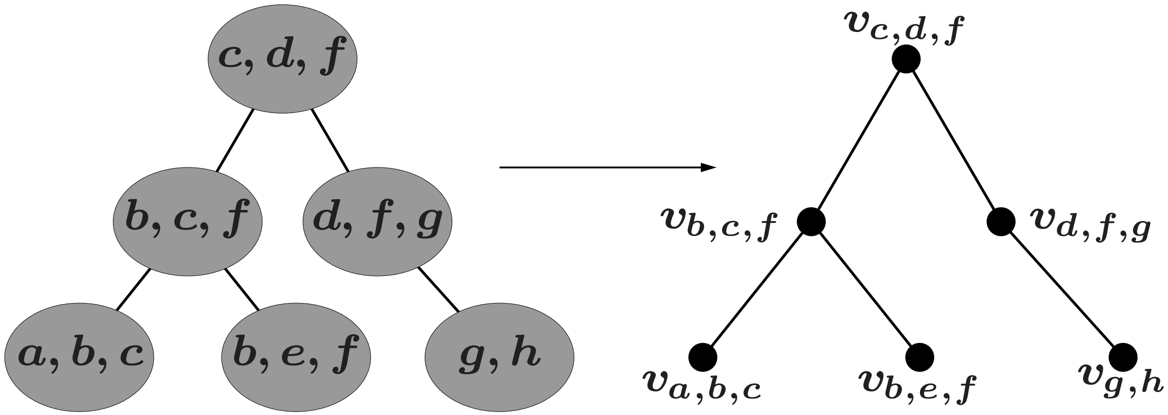
Solving CSP using a tree decompositions

Suppose that a tree decomposition of width k is given. We build an equivalent tree CSP where every variable represents a bag of the decomposition.



Solving CSP using a tree decompositions

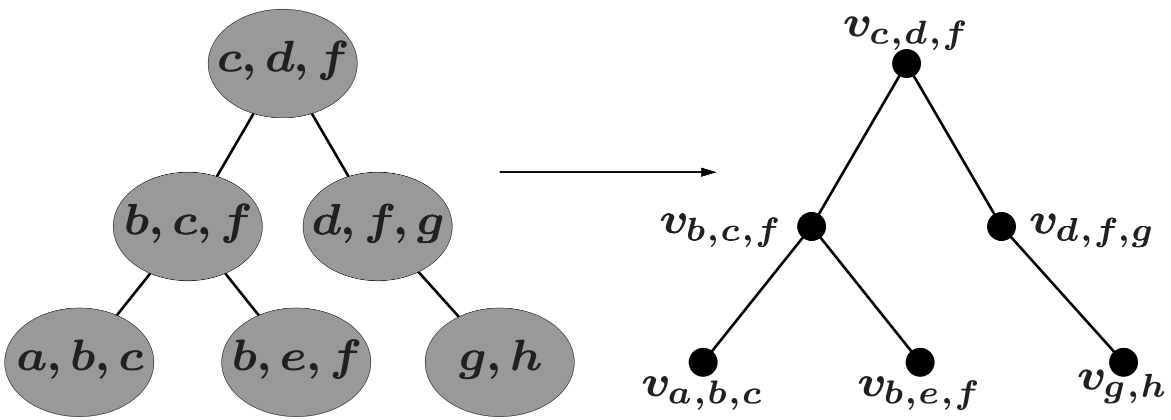
Suppose that a tree decomposition of width k is given. We build an equivalent tree CSP where every variable represents a bag of the decomposition.



The domain of variable $v_{a,b,c}$ is the set of all partial solutions on $\{a, b, c\}$.
Binary constraint between $v_{a,b,c}$ and $v_{b,c,f}$ require that the two partial solutions agree on the intersection $\{b, c\}$.

Solving CSP using a tree decompositions

Suppose that a tree decomposition of width k is given. We build an equivalent tree CSP where every variable represents a bag of the decomposition.



The domain of variable $v_{a,b,c}$ is the set of all partial solutions on $\{a, b, c\}$.
Binary constraint between $v_{a,b,c}$ and $v_{b,c,f}$ require that the two partial solutions agree on the intersection $\{b, c\}$.

- Instance has polynomial size for fixed k: domain size $\leq |D|^{k+1}$.
- There are no conflicts between the partial assignments.
- Every original constraint is satisfied by one of the partial solutions.

CSP and tree decompositions

Fact: If we are given a tree decomposition of the primal graph of instance I together with a list (having length $\leq C$) of all partial solutions for each bag B, then I can be solved in time polynomial in $\|I\|$ and C.

CSP and tree decompositions

Fact: If we are given a tree decomposition of the primal graph of instance I together with a list (having length $\leq C$) of all partial solutions for each bag B, then I can be solved in time polynomial in $\|I\|$ and C.

Can be made a little stronger:

Fact: If we are given a tree decomposition of the primal graph of instance I together with a list (having length $\leq C$) of all solutions of the **projection** to B for each bag B, then I can be solved in time polynomial in $\|I\|$ and C.

The **projection** of instance I to $B \subseteq V$ is an instance on B such that for every constraint c of I with scope S such that $S \cap B \neq \emptyset$, there is a constraint on $S \cap B$ that is satisfied if it can be extended to a satisfying tuple of c.

CSP and tree decompositions

Fact: If we are given a tree decomposition of the primal graph of instance I together with a list (having length $\leq C$) of all partial solutions for each bag B, then I can be solved in time polynomial in $\|I\|$ and C.

Can be made a little stronger:

Fact: If we are given a tree decomposition of the primal graph of instance I together with a list (having length $\leq C$) of all solutions of the **projection** to B for each bag B, then I can be solved in time polynomial in $\|I\|$ and C.

The **projection** of instance I to $B \subseteq V$ is an instance on B such that for every constraint c of I with scope S such that $S \cap B \neq \emptyset$, there is a constraint on $S \cap B$ that is satisfied if it can be extended to a satisfying tuple of c.

Example:
Projection to $\{v_1, v_2, v_3\}$

$$s = (v_2, v_3, v_4, v_5),\ R = \begin{array}{c}(1,4,1,1)\\(2,3,2,4)\\(2,3,5,1)\\(5,5,1,1)\\(5,5,2,5)\end{array} \Rightarrow \quad s = (v_2, v_3),\ R = \begin{array}{c}(1,4)\\(2,3)\\(5,5)\end{array}$$

Finding tree decompositions

Fact: It is NP-hard to determine the treewidth of a graph (given a graph G and a integer k, decide if the treewidth of G is at most k), but there is a polynomial-time algorithm for every fixed k.

Fact: [Bodlaender's Theorem] For every fixed k, there is a linear-time algorithm that finds a tree decomposition of width k (if exists).
$\Rightarrow$ Treewidth is fixed-parameter tractable

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

Finding tree decompositions

Fact: It is NP-hard to determine the treewidth of a graph (given a graph G and a integer k, decide if the treewidth of G is at most k), but there is a polynomial-time algorithm for every fixed k.

Fact: [Bodlaender's Theorem] For every fixed k, there is a linear-time algorithm that finds a tree decomposition of width k (if exists).
$\Rightarrow$Treewidth is fixed-parameter tractable

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

Two main approaches:

- Game-theoretic characterization.
- Separator-based approach.

The Robber and Cops game

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

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

Fact:

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

The Robber and Cops game

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

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

Fact:

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

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

$$\Downarrow$$

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

The Robber and Cops game

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

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

Fact:

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

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

$$\Downarrow$$

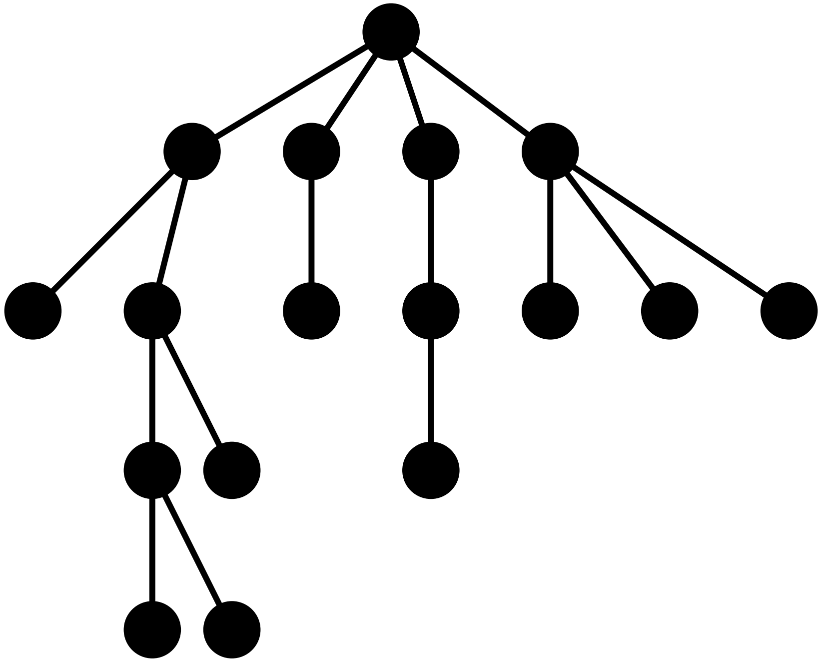
For every fixed k, it can be checked in polynomial-time if treewidth is at most k.

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

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

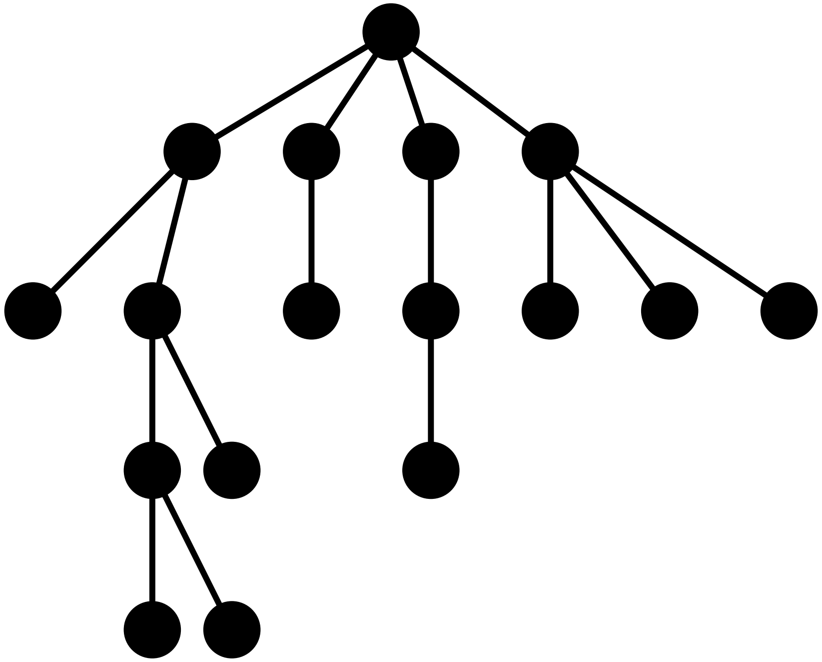
The Robber and Cops game (cont.)

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



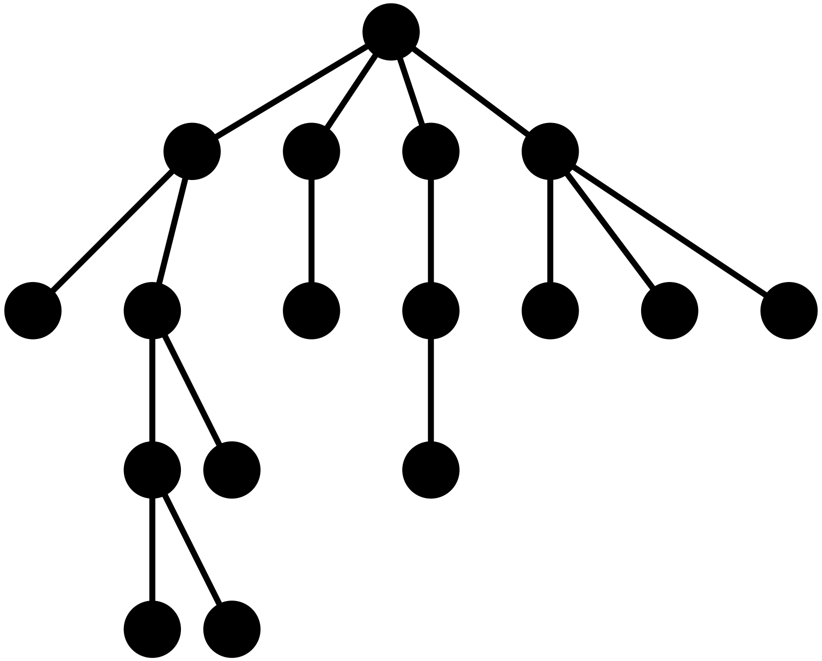
The Robber and Cops game (cont.)

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



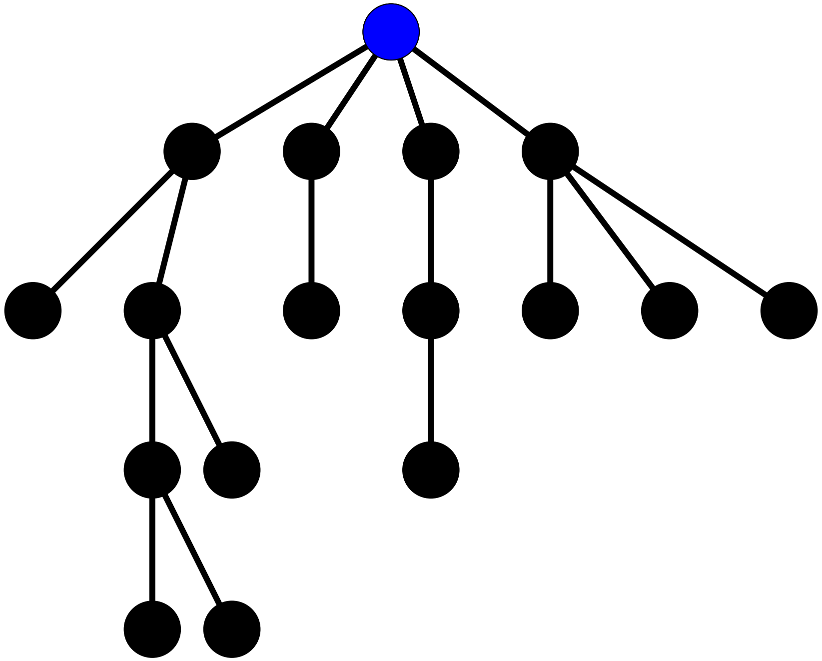
The Robber and Cops game (cont.)

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



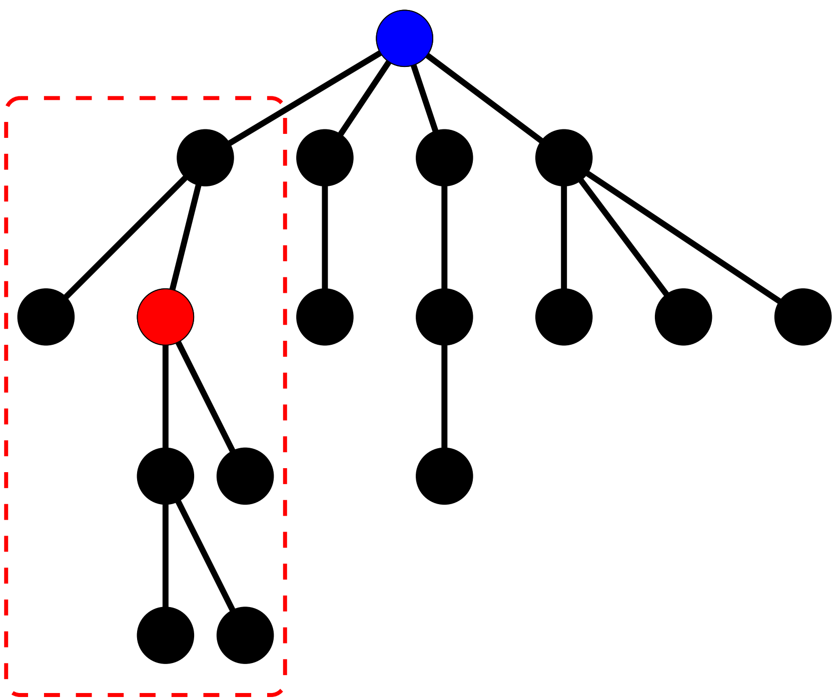
The Robber and Cops game (cont.)

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



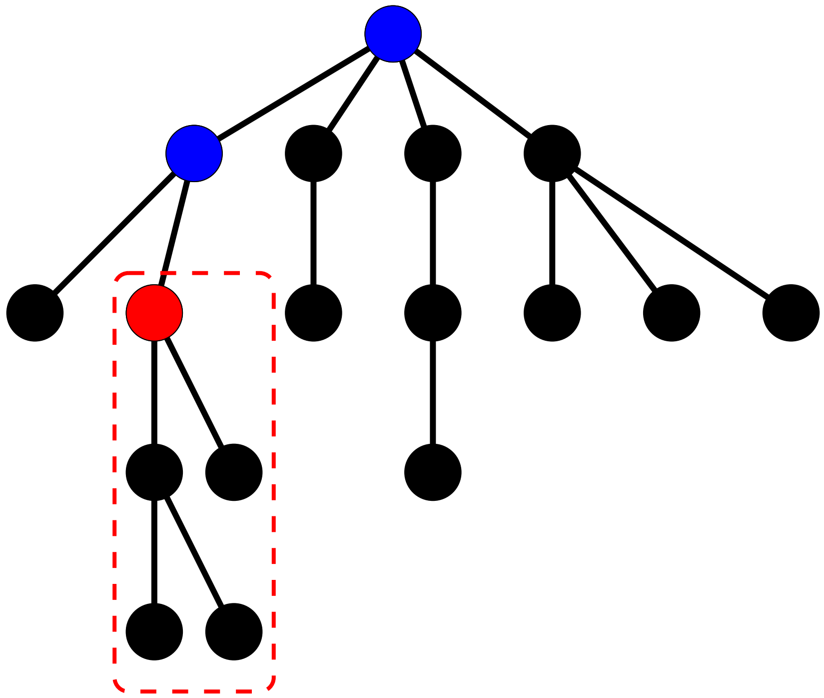
The Robber and Cops game (cont.)

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



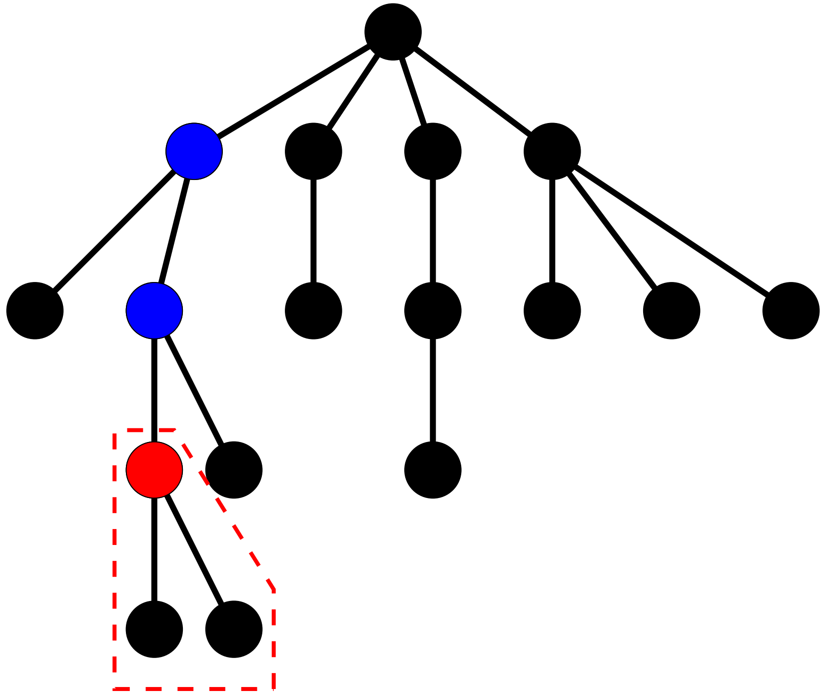
The Robber and Cops game (cont.)

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



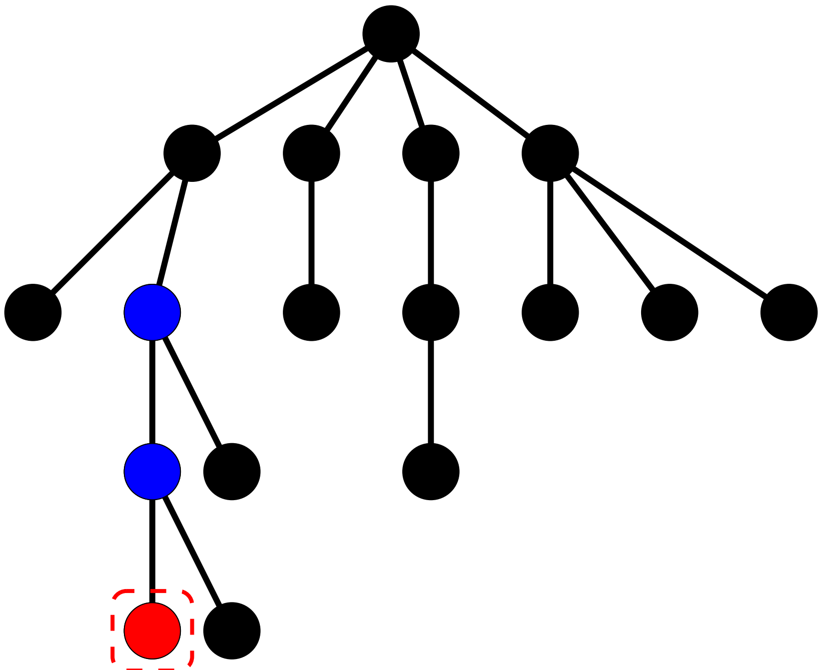
The Robber and Cops game (cont.)

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



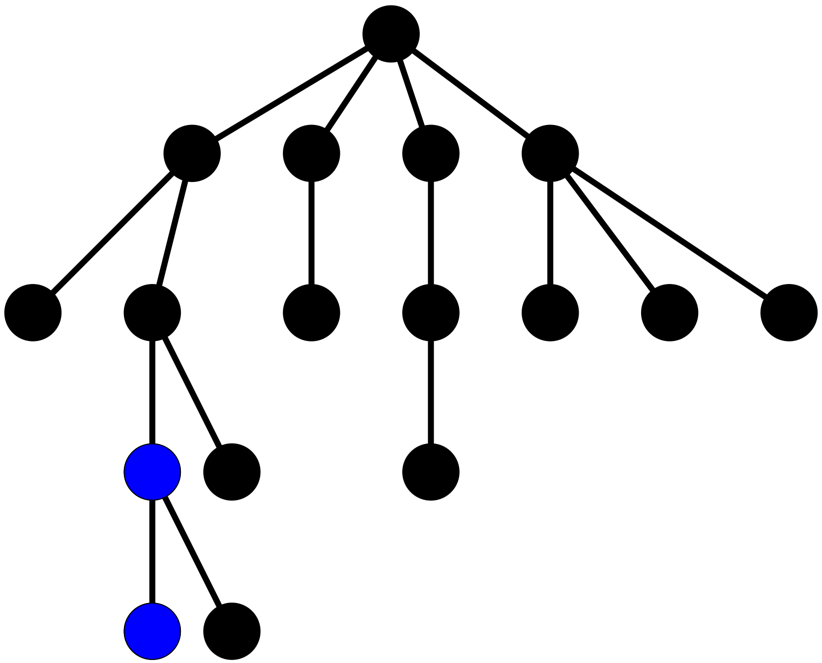
The Robber and Cops game (cont.)

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



The Robber and Cops game (cont.)

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



Separator-based approach

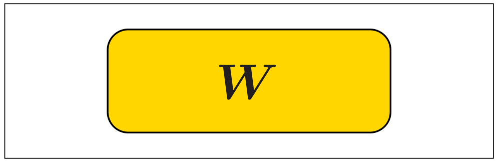
Solve the more general problem: Build a tree decomposition with the additional restriction that some bag contains a given set W (of appropriate size).

Main step: Find a separator S that splits W in an appropriate way and recurse on the two parts of the graph.

Separator-based approach

Solve the more general problem: Build a tree decomposition with the additional restriction that some bag contains a given set W (of appropriate size).

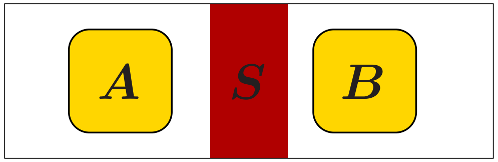
Main step: Find a separator S that splits W in an appropriate way and recurse on the two parts of the graph.



Separator-based approach

Solve the more general problem: Build a tree decomposition with the additional restriction that some bag contains a given set W (of appropriate size).

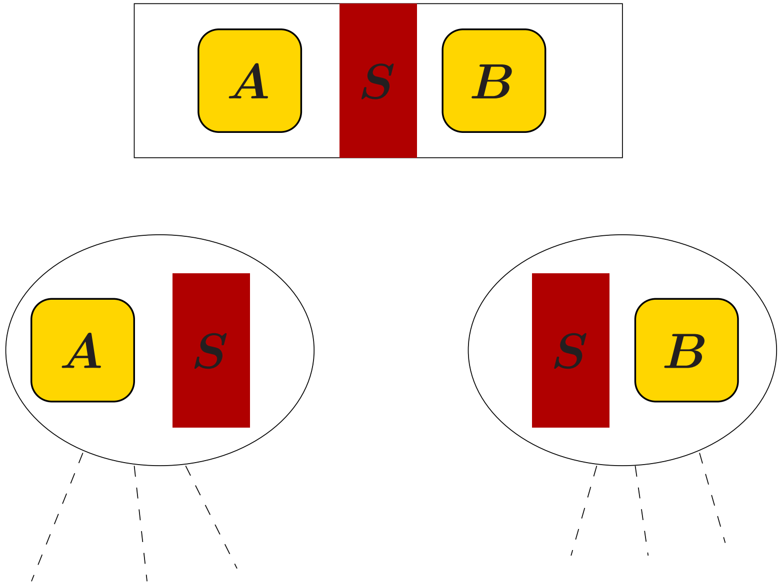
Main step: Find a separator S that splits W in an appropriate way and recurse on the two parts of the graph.



Separator-based approach

Solve the more general problem: Build a tree decomposition with the additional restriction that some bag contains a given set W (of appropriate size).

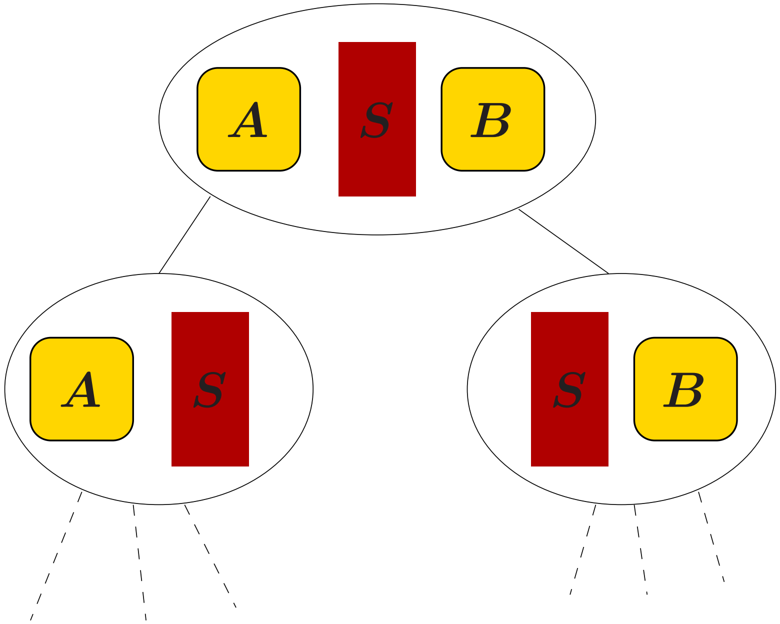
Main step: Find a separator S that splits W in an appropriate way and recurse on the two parts of the graph.



Separator-based approach

Solve the more general problem: Build a tree decomposition with the additional restriction that some bag contains a given set W (of appropriate size).

Main step: Find a separator S that splits W in an appropriate way and recurse on the two parts of the graph.



If no suitable separator S exists, then we can argue that treewidth is large.

Overview

- Part 1: Trees, treewidth and their algorithmic consequences
- Part 2: Treewidth and lower bounds
- Part 3: Width notions for hypergraphs

Lower bounds

We know:

Bounded treewidth instances are “easy.”

Question:

Is there some other graph-theoretic property that makes CSP easy to solve?

Formal setting

Note: CSP is polynomial-time solvable for every fixed graph G, as the number of variables is a constant. Therefore, we want to find **classes** of graphs where CSP is easy.

Definition: Given a (possibly infinite) set $\mathcal{G}$ of graphs, CSP$(\mathcal{G})$ is the CSP restricted to instances whose primal graph is in $\mathcal{G}$.

Definition: CSP$(\mathcal{G})$ is **polynomial-time solvable** if there is an algorithm solving every instance I of CSP$(\mathcal{G})$ in time $\|I\|^c$ for some constant c.

Formal setting

Note: CSP is polynomial-time solvable for every fixed graph G, as the number of variables is a constant. Therefore, we want to find **classes** of graphs where CSP is easy.

Definition: Given a (possibly infinite) set $\mathcal{G}$ of graphs, CSP$(\mathcal{G})$ is the CSP restricted to instances whose primal graph is in $\mathcal{G}$.

Definition: CSP$(\mathcal{G})$ is **polynomial-time solvable** if there is an algorithm solving every instance I of CSP$(\mathcal{G})$ in time $\|I\|^c$ for some constant c.

Definition: CSP$(\mathcal{G})$ is **fixed-parameter tractable (FPT)** if there is an algorithm solving every instance I of CSP$(\mathcal{G})$ in time $f(G)\|I\|^c$ for some function f depending only on the primal graph G and a constant c.

Note: The definition does not change if we replace $f(G)$ with a function $g(k)$ depending on the number of variables.

Dichotomy for binary CSP

We know that CSP($\mathcal{G}$) is polynomial-time solvable if $\mathcal{G}$ has bounded treewidth.
Are there other polynomial cases?

Dichotomy for binary CSP

We know that CSP($\mathcal{G}$) is polynomial-time solvable if $\mathcal{G}$ has bounded treewidth. Are there other polynomial cases?

Fact: Let $\mathcal{G}$ be a recursively enumerable class of graphs. Assuming FPT $\neq$ W[1], the following are equivalent:

- Binary CSP($\mathcal{G}$) is polynomial-time solvable.
- Binary CSP($\mathcal{G}$) is FPT.
- $\mathcal{G}$ has bounded treewidth.

Dichotomy for binary CSP

We know that CSP($\mathcal{G}$) is polynomial-time solvable if $\mathcal{G}$ has bounded treewidth. Are there other polynomial cases?

Fact: Let $\mathcal{G}$ be a recursively enumerable class of graphs. Assuming FPT $\neq$ W[1], the following are equivalent:

- Binary CSP($\mathcal{G}$) is polynomial-time solvable.
- Binary CSP($\mathcal{G}$) is FPT.
- $\mathcal{G}$ has bounded treewidth.

Note: FPT $\neq$ W[1] is the standard assumption of parameterized complexity.

Note: Fixed-parameter tractability does not give us more power here than polynomial-time solvability.

Note: We cannot hope a P vs. NP-complete dichotomy: there are classes $\mathcal{G}$ for which the problem is equivalent to LOGCLIQUE.

Proof outline

Suppose that $\mathcal{G}$ has unbounded treewidth, but CSP($\mathcal{G}$) is FPT.

- Assuming FPT $\neq$ W[1], there is no $f(k)n^c$ time algorithm for k-CLIQUE. But we can solve k-CLIQUE the following way:
- Formulate k-CLIQUE as a binary CSP instance on the $k \times k$ grid.
- Find a $G_k \in \mathcal{G}$ containing a $k \times k$ minor (there is such a G_k by the Excluded Grid Theorem).
- Reduce CSP on the $k \times k$ grid to CSP with graph G_k, which is an instance of CSP($\mathcal{G}$).
- Use the assumed algorithm for CSP($\mathcal{G}$).
- The running time is $f(k)n^c$: the nonpolynomial factors in the running time depend only on k (finding G_k, size of G_k, solving CSP($\mathcal{G}$))
  $\Rightarrow$ k-CLIQUE is FPT, contradicting the hypothesis FPT $\neq$ W[1].

Can you beat treewidth?

If $\mathcal{G}$ has unbounded treewidth, then there is no polynomial algorithm for binary CSP($\mathcal{G}$), but it can be solved in time $\|I\|^{O(k)}$, where k is the treewidth of the primal graph.

Is there a class $\mathcal{G}$ where we can do much better, for example, there is a $\|I\|^{O(\sqrt{k})}$ or even $\|I\|^{O(\log \log \log k)}$ algorithm CSP($\mathcal{G}$)?

Can you beat treewidth?

If $\mathcal{G}$ has unbounded treewidth, then there is no polynomial algorithm for binary CSP($\mathcal{G}$), but it can be solved in time $\|I\|^{O(k)}$, where k is the treewidth of the primal graph.

Is there a class $\mathcal{G}$ where we can do much better, for example, there is a $\|I\|^{O(\sqrt{k})}$ or even $\|I\|^{O(\log\log\log k)}$ algorithm CSP($\mathcal{G}$)?

Fact: [M. 2007] If $\mathcal{G}$ is a recursively enumerable class of graphs such that CSP($\mathcal{G}$) can be solved in time $f(G) \cdot \|I\|^{o(k/\log k)}$ (where G is the primal graph and $k = \text{tw}(G)$), then the Exponential Time Hypothesis fails.

Exponential Time Hypothesis (ETH): There is no $2^{o(n)}$ time algorithm for n-variable 3SAT (known to be equivalent with "There is no $2^{o(m)}$ time algorithm for m-clause 3SAT").

Embeddings

The previous proof is based on embedding the k-CLIQUE problem into a CSP instance using the grid whose existence is guaranteed by the Excluded Grid Theorem. However, this theorem is very weak: a $k \times k$ grid minor exists, if treewidth is exponentially large in k.

Embeddings

The previous proof is based on embedding the k-CLIQUE problem into a CSP instance using the grid whose existence is guaranteed by the Excluded Grid Theorem. However, this theorem is very weak: a $k \times k$ grid minor exists, if treewidth is exponentially large in k.

Definition: A q**-embedding** ϕ of graph F in graph G maps a subset of $V(G)$ to each vertex of F such that

- For every $v \in V(F)$, $\phi(v)$ is connected.
- If $u, v \in V(F)$ are adjacent in F, then $\phi(u)$ and $\phi(v)$ touch: either they intersect or there is an edge connecting them.
- Every $w \in V(G)$ appears in the images of at most q vertices of F.

Note: F is a minor of $G \iff$ there is a 1-embedding from F to G.

An embedding result

Fact: [M. 2007] If $m = |E(F)|$ is sufficiently large and $k = \text{tw}(G)$, then there is a q-embedding of F in G for $q = O(m \log k / k)$.

Note: A q-embedding for $q = O(m)$ is trivial, thus treewidth k means that we can gain a factor of $\Omega(k/\log k)$ compared to the trivial embedding.

Main ingredients of the proof:

- characterization of treewidth by sets having no balanced separators,
- results from combinatorial optimization that show that certain flows exist if there is no balanced separator,
- the q-embedding is constructed using the paths appearing in the flows.

Proof

Fact: [M. 2007] If $\mathcal{G}$ is a recursively enumerable class of graphs such that binary CSP($\mathcal{G}$) can be solved in time $f(G) \cdot \|I\|^{o(k/\log k)}$ (where G is the primal graph and $k = \text{tw}(G)$), then the Exponential Time Hypothesis fails.

Proof outline:

- Given a 3SAT instance with m clauses and n variables, we turn it into a CSP instance I_1 with $3m$ binary constraints.
- We use the embedding result to find a q-embedding of the primal graph of I_1 into some $G_k \in \mathcal{G}$ (chosen appropriately).
- We simulate I_1 by an instance I_2 whose primal graph is G_k: each variable of I_2 simulates at most q variables of I_1.
- Now the 3SAT problem can be solved by solving I_2. Calculation of the running time shows that that a "too fast" algorithm for CSP($\mathcal{G}$) would give a $2^{o(m)}$ algorithm for m-clause 3SAT, violating ETH.

Constraint of higher arity

How are the constraints represented in the input?

- full truth table
- listing the satisfying tuples
- formula/circuit
- oracle

Constraint of higher arity

How are the constraints represented in the input?

- full truth table
- listing the satisfying tuples
- formula/circuit
- oracle

If the arity of every constraint is bounded by a constant, then the representations are polynomially equivalent, but if there is no bound there can be exponential difference between different representations.

The choice of representation changes the length of the input, thus can change the complexity of the problem.

Constraint of higher arity

How are the constraints represented in the input?

- full truth table
- listing the satisfying tuples
- formula/circuit
- oracle

In this talk: Each constraint is given by listing all the tuples that satisfy it.

Motivation: Applications in database theory (Conjunctive Query Evaluation, Conjunctive Query Containment)

Constraints are known databases, “satisfying” means “appears in the database.”

Characterization with higher arities

What are the tractable graph classes $\mathcal{G}$ for **not necessarily binary** CSP?

Characterization with higher arities

What are the tractable graph classes $\mathcal{G}$ for **not necessarily binary** CSP?

The answer does not change:

Fact: Let $\mathcal{G}$ be a recursively enumerable class of graphs. Assuming FPT $\neq$ W[1], the following are equivalent:

- CSP($\mathcal{G}$) is polynomial-time solvable.
- CSP($\mathcal{G}$) is FPT.
- $\mathcal{G}$ has bounded treewidth.

$\mathcal{G}$ has bounded treewidth: same algorithm works.
$\mathcal{G}$ has unbounded treewidth: problem was hard already in the binary case.

Hypergraphs

Considering the hypergraph instead of the primal graph makes the complexity analysis more precise.

$$I_1 = C(x_1, x_2, \dots, x_n) \text{ vs.}$$

$$I_2 = C(x_1, x_2) \wedge C(x_1, x_3) \wedge \cdots \wedge C(x_{n-1}, x_n)$$

I_1, I_2 have the same primal graph (n-clique), but I_1 is always easy, I_2 can be hard.

Goal: Characterize classes $\mathcal{H}$ of hypergraphs that make CSP$(\mathcal{H})$ easy.

Definition: In the **primal graph** of a hypergraph two vertices are adjacent if they appear together in some hyperedge.

Definition: The treewidth of a hypergraph is the treewidth of its primal graph.

Bounded arity hypergraphs

Fact: Let $\mathcal{H}$ be a recursively enumerable class of hypergraphs of **bounded arity**. Assuming FPT $\neq$ W[1], the following are equivalent:

- CSP($\mathcal{H}$) is polynomial-time solvable.
- CSP($\mathcal{H}$) is FPT.
- $\mathcal{H}$ has bounded treewidth.

Bounded arity hypergraphs

Fact: Let $\mathcal{H}$ be a recursively enumerable class of hypergraphs of **bounded arity**. Assuming FPT $\neq$ W[1], the following are equivalent:

- CSP($\mathcal{H}$) is polynomial-time solvable.
- CSP($\mathcal{H}$) is FPT.
- $\mathcal{H}$ has bounded treewidth.

For unbounded-arity classes, this characterization is not correct:

Example: Let $\mathcal{H}$ contain every hypergraph having only a single edge (of arbitrary size). $\mathcal{H}$ has unbounded treewidth, but CSP($\mathcal{H}$) is trivial (since there is only a single constraint).

Bounded arity hypergraphs

Fact: Let $\mathcal{H}$ be a recursively enumerable class of hypergraphs of **bounded arity**. Assuming FPT $\neq$ W[1], the following are equivalent:

- CSP($\mathcal{H}$) is polynomial-time solvable.
- CSP($\mathcal{H}$) is FPT.
- $\mathcal{H}$ has bounded treewidth.

Before entering the world of unbounded arities, let us make a short detour to relational structures.

Homomorphisms of relational structures

Relational structure: a set of relations over the same universe.

$$\mathbb{A} = (R_1^{\mathbb{A}}(x_1, x_2, x_3), R_2^{\mathbb{A}}(x_1), R_3^{\mathbb{A}}(x_1, x_2, x_3))$$

Homomorphisms of relational structures

Relational structure: a set of relations over the same universe.

$$\mathbb{A} = (R_1^{\mathbb{A}}(x_1, x_2, x_3), R_2^{\mathbb{A}}(x_1), R_3^{\mathbb{A}}(x_1, x_2, x_3))$$
$$\mathbb{B} = (R_1^{\mathbb{B}}(x_1, x_2, x_3), R_2^{\mathbb{B}}(x_1), R_3^{\mathbb{B}}(x_1, x_2, x_3))$$

Homomorphism of relational structures: If A is the universe of $\mathbb{A}$ and B is the universe of $\mathbb{B}$, then a homomorphism from $\mathbb{A}$ to $\mathbb{B}$ is a mapping $f : A \to B$ such that for every relation

$$(a_1, a_2, a_3) \in R_1^{\mathbb{A}} \Rightarrow (f(a_1), f(a_2), f(a_3)) \in R_1^{\mathbb{B}}.$$

Homomorphism problem HOM$(\mathbb{A}, \mathbb{B})$**:** Is there a homomorphism from $\mathbb{A}$ to $\mathbb{B}$?

Equivalent formulation of constraint satisfaction problems:

- $R_1^{\mathbb{A}}$ lists which tuples of variables have a constraint R_1 imposed on them.
- $R_1^{\mathbb{B}}$ lists the tuples that satisfy constraint R_1.

Restricting the left hand side

If $\mathcal{A}$ is a class of relational structures, then HOM$(\mathcal{A}, -)$ is the homomorphism problem restricted to instances where the left side is in $\mathcal{A}$.

Goal: Characterize classes $\mathcal{A}$ for which HOM$(\mathcal{A}, -)$ is in PTIME.

Restricting the left hand side

If $\mathcal{A}$ is a class of relational structures, then HOM$(\mathcal{A}, -)$ is the homomorphism problem restricted to instances where the left side is in $\mathcal{A}$.

Goal: Characterize classes $\mathcal{A}$ for which HOM$(\mathcal{A}, -)$ is in PTIME.

Fact: Let $\mathcal{A}$ be a recursively enumerable class of relational structures of **bounded arity**. Assuming FPT $\neq$ W[1], the following are equivalent:

- The decision problem HOM$(\mathcal{A}, -)$ is polynomial-time solvable.
- The decision problem HOM$(\mathcal{A}, -)$ is FPT.
- The **cores** of the structures in $\mathcal{A}$ have bounded treewidth.

Core of $\mathbb{A}$: minimum subset A' of the universe A such that there is a homomorphism $\mathbb{A} \longrightarrow \mathbb{A}[A']$ (unique up to isomorphism)

If the treewidth of $\mathcal{A}$ is k, then the k-consistency algorithm decides HOM$(\mathcal{A}, -)$, but does not produce a solution!

Overview

- Part 1: Trees, treewidth and their algorithmic consequences
- Part 2: Treewidth and lower bounds
- Part 3: Width notions for hypergraphs

Unbounded arities

Another example showing that unbounded treewidth of a class does not imply that the problem is hard:

Example: Let $\mathcal{H}_d$ contain every hypergraph where there is a subset $e_1, \ldots, e_d$ of edges that cover every vertex. Then CSP$(\mathcal{H}_d)$ is polynomial-time solvable for every fixed d: try every combination of tuples for the d constraints corresponding to the d edges (at most $||I||^d$ combinations).

This gives us an idea: try to use tree decompositions where the bags are not necessarily small, but can be covered by a small number of edges.

Tree decompositions of hypergraphs

Tree decomposition of a hypergraph is a tree decomposition of its primal graph.
Equivalently:

Bags of vertices are arranged in a tree structure satisfying the following properties:

1. For every hyperedge e, there is a bag containing every vertex of e.
2. For every vertex v, the bags containing v form a connected subtree.

Tree decompositions of hypergraphs

Tree decomposition of a hypergraph is a tree decomposition of its primal graph. Equivalently:

Bags of vertices are arranged in a tree structure satisfying the following properties:

1. For every hyperedge e, there is a bag containing every vertex of e.
2. For every vertex v, the bags containing v form a connected subtree.

The **generalized hypertree width** of a decomposition is the minimum integer k such that every bag can be covered by k edges. Generalized hypertree width ghw(H) of H is the minimum width over all possible decompositions.

Tree decompositions of hypergraphs

Tree decomposition of a hypergraph is a tree decomposition of its primal graph. Equivalently:

Bags of vertices are arranged in a tree structure satisfying the following properties:

1. For every hyperedge e, there is a bag containing every vertex of e.
2. For every vertex v, the bags containing v form a connected subtree.

The **generalized hypertree width** of a decomposition is the minimum integer k such that every bag can be covered by k edges. Generalized hypertree width ghw(H) of H is the minimum width over all possible decompositions.

The original definition of **hypertree width** hw(H) adds a third technical requirement (monotonicity condition) to the definition of tree decomposition.

Fact: ghw$(H) \leq$ hw$(H) \leq 3$ghw(H).
$\Rightarrow$ $\mathcal{H}$ has bounded hypertree width if and only if it has bounded generalized hypertree width.

Using hypertree width

Recall: If we are given a tree decomposition of the primal graph of instance I together with a list (having length $\leq C$) of all solutions of the **projection** to B for each bag B, then I can be solved in time polynomial in $\|I\|$ and C.

Using hypertree width

Recall: If we are given a tree decomposition of the primal graph of instance I together with a list (having length $\leq C$) of all solutions of the **projection** to B for each bag B, then I can be solved in time polynomial in $\|I\|$ and C.

In a tree decomposition with (generalized) hypertree width k every bag can be covered by k hyperedges.

$\Rightarrow$ There are at most $\|I\|^k$ satisfying assignments for the projection to a bag.

$\Rightarrow$ Polynomial-time algorithm for every fixed k.

Using hypertree width

Recall: If we are given a tree decomposition of the primal graph of instance I together with a list (having length $\leq C$) of all solutions of the **projection** to B for each bag B, then I can be solved in time polynomial in $\|I\|$ and C.

In a tree decomposition with (generalized) hypertree width k every bag can be covered by k hyperedges.
$\Rightarrow$ There are at most $\|I\|^k$ satisfying assignments for the projection to a bag.
$\Rightarrow$ Polynomial-time algorithm for every fixed k.

If $\mathcal{H}$ has bounded (generalized) hypertree width, then CSP$(\mathcal{H})$ is **fixed-parameter tractable.**

For polynomial-time solvability, we need to be able to find decompositions with small hypertree width.

Finding a hypertree decompositions

Fact: It is NP-hard to decide if ghw$(H) \leq 3$ for a given hypergraph H.

The Robber and Cops game characterized treewidth, the **Robber and Marshals** game characterizes hypertree width.

Fact: For every fixed k, there is a polynomial-time algorithm that finds a tree decomposition with hypertree width at most k, if exists.

$\Rightarrow$ **Fact:** For every fixed k, there is a polynomial-time algorithm that either finds a generalized hypertree decomposition of width at most $3k$, or correctly concludes that ghw$(H) > k$.

$\Rightarrow$ **Fact:** If $\mathcal{H}$ has bounded (generalized) hypertree width, then CSP$(\mathcal{H})$ is polynomial-time solvable.

Beyond hypertree width

Is there some hypergraph property more general than bounded hypertree width that guarantees polynomial-time solvability?

We need to understand what hypergraph properties can guarantee that the number of solutions in a bag is small.

This is an interesting and deep question on its own right.

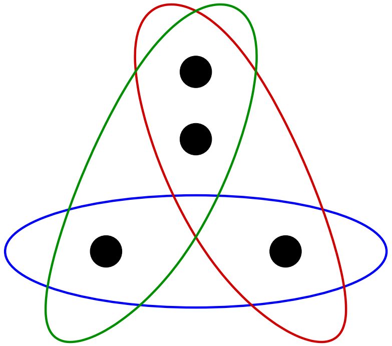
(Fractional) edge covering

An **edge cover** of a hypergraph is a subset of the edges such that every vertex is covered by at least one edge.

$\varrho(H)$: size of the smallest edge cover.

A **fractional edge cover** is a weight assignment to the edges such that every vertex is covered by total weight at least 1.

$\varrho^*(H)$: smallest total weight of a fractional edge cover.



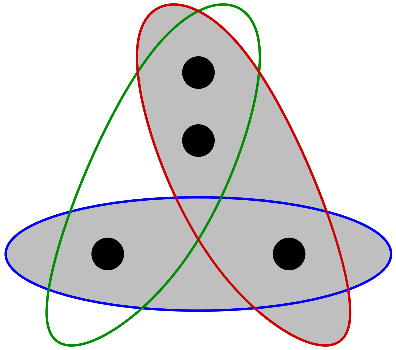
(Fractional) edge covering

An **edge cover** of a hypergraph is a subset of the edges such that every vertex is covered by at least one edge.

$\varrho(H)$: size of the smallest edge cover.

A **fractional edge cover** is a weight assignment to the edges such that every vertex is covered by total weight at least 1.

$\varrho^*(H)$: smallest total weight of a fractional edge cover.

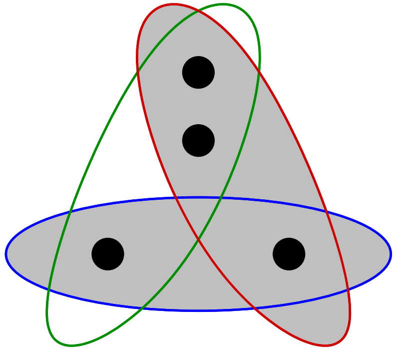


$$\varrho(H) = 2$$

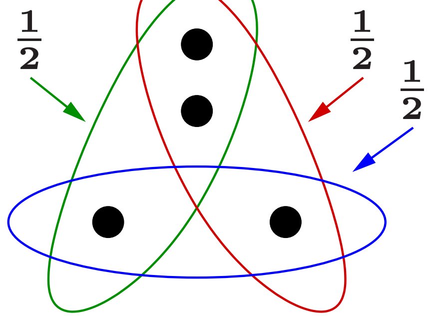
(Fractional) edge covering

An **edge cover** of a hypergraph is a subset of the edges such that every vertex is covered by at least one edge.
$\varrho(H)$: size of the smallest edge cover.

A **fractional edge cover** is a weight assignment to the edges such that every vertex is covered by total weight at least 1.
$\varrho^*(H)$: smallest total weight of a fractional edge cover.



$$\varrho(H) = 2$$

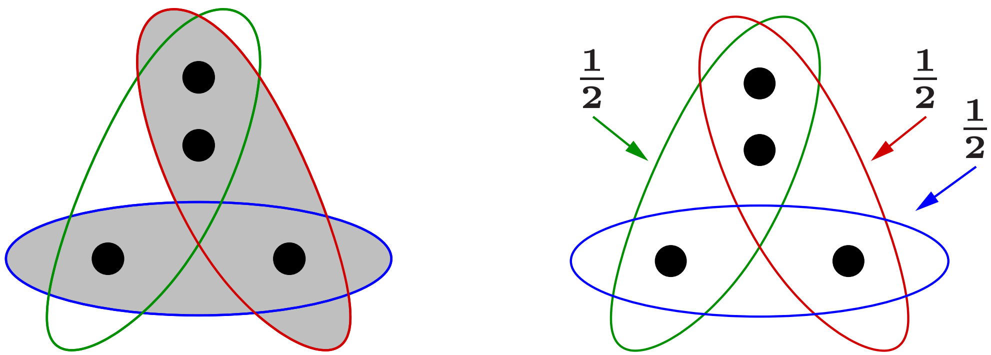


$$\varrho^*(H) = 1.5$$

(Fractional) edge covering

An **edge cover** of a hypergraph is a subset of the edges such that every vertex is covered by at least one edge.
$\varrho(H)$: size of the smallest edge cover.

A **fractional edge cover** is a weight assignment to the edges such that every vertex is covered by total weight at least 1.
$\varrho^*(H)$: smallest total weight of a fractional edge cover.



(Fractional) edge cover of a set of vertices is defined analogously.

Edge covers vs. fractional edge covers

Fact: It is NP-hard to determine the edge cover number $\varrho(H)$.

Fact: The fractional edge cover number $\varrho^*(H)$ can be determined in polynomial time using linear programming.

The gap between $\varrho(H)$ and $\varrho^*(H)$ can be arbitrarily large.

Edge covers vs. fractional edge covers

Fact: It is NP-hard to determine the edge cover number $\varrho(H)$.
Fact: The fractional edge cover number $\varrho^*(H)$ can be determined in polynomial time using linear programming.

The gap between $\varrho(H)$ and $\varrho^*(H)$ can be arbitrarily large.

Example:
$\binom{2k}{k}$ vertices: all the possible strings with k 0's and k 1's.
$2k$ hyperedges: edge E_i contains the vertices with 1 at the i-th position.

Edge covers vs. fractional edge covers

Fact: It is NP-hard to determine the edge cover number $\varrho(H)$.
Fact: The fractional edge cover number $\varrho^*(H)$ can be determined in polynomial time using linear programming.

The gap between $\varrho(H)$ and $\varrho^*(H)$ can be arbitrarily large.

Example:
$\binom{2k}{k}$ vertices: all the possible strings with k 0's and k 1's.
$2k$ hyperedges: edge E_i contains the vertices with 1 at the i-th position.

Edge cover: if only k edges are selected, then there is a vertex that contains 1's only at the remaining k positions, hence not covered $\Rightarrow \varrho(H) \geq k + 1$.

Fractional edge cover: assign weight $1/k$ to each edge, each vertex is covered by exactly k edges $\Rightarrow \varrho^*(H) \leq 2k \cdot 1/k = 2$.

CSP and fractional edge covering

Fact: [easy] If the hypergraph of instance I has edge cover number w, then there are at most $\|I\|^w$ satisfying assignments.
Proof: Assume that $C_1, \ldots, C_w$ cover the instance. Fixing a satisfying assignment for each C_i determines all the variables.

Fact: [Grohe and M. 2006] If the hypergraph of instance I has fractional edge cover number w, then there are at most $\|I\|^w$ satisfying assignments (and they can be enumerated in polynomial time).
Proof: By Shearer's Lemma.

Corollary: CSP($\mathcal{H}$) is polynomial-time solvable if $\mathcal{H}$ has bounded fractional edge cover number.

Shearer's Lemma

Shearer's Lemma: Assume we have the following random variables:

- $X_1, \ldots, X_n$,
- $Y_1, \ldots, Y_m$, where each $Y_i = (X_{i_1}, \ldots, X_{i_k})$ is a combination of some X_i's,
- $X = (X_1, \ldots, X_n)$.

If each X_j appears in at least q of the Y_i's, then $H(X) \leq \frac{1}{q} \sum H(Y_i)$.

Entropy: "information content"
$H(X) = -\sum_x P(X = x) \log_2 P(X = x)$

Bounding the number of solutions

Lemma: If the hypergraph of instance I has fractional edge cover number w, then there are at most $\|I\|^w$ satisfying assignments.

Example: Let $C_1(x_1, x_2) \wedge C_2(x_2, x_3) \wedge C_3(x_1, x_3)$ be an instance where each constraint is satisfied by at most n pairs.

Fractional edge cover number: $3/2 \Rightarrow$ we have to show that there are at most $n^{3/2}$ solutions.

Bounding the number of solutions

Lemma: If the hypergraph of instance I has fractional edge cover number w, then there are at most $\|I\|^w$ satisfying assignments.

Example: Let $C_1(x_1, x_2) \wedge C_2(x_2, x_3) \wedge C_3(x_1, x_3)$ be an instance where each constraint is satisfied by at most n pairs.

Fractional edge cover number: $3/2 \Rightarrow$ we have to show that there are at most $n^{3/2}$ solutions.

Let $X = (x_1, x_2, x_3)$ be a random variable with uniform distribution over the **satisfying assignments** of the instance.

$Y_1 = (x_1, x_2)$ $Y_2 = (x_2, x_3)$ $Y_3 = (x_1, x_3)$
$H(Y_i) \leq \log_2 n$ (has at most n different values)
$H(X) \leq \frac{1}{2}(H(Y_1) + H(Y_2) + H(Y_3)) \leq \frac{3}{2} \log_2 n$

X has uniform distribution, hence it has $2^{H(X)} = 2^{\frac{3}{2}\log_2 n} = n^{3/2}$ different values.

Fractional hypertree width

The **fractional hypertree width** of a tree decomposition is the minimum value w such that every bag has a fractional cover of weight w.

Fractional cover of a bag B: a weight assignment on the edges such that for each $v \in B$, the total weight of the edges containing v is at least 1. It can be checked in polynomial time if such an assignment of weight at most w exists.

Fractional hypertree width fhw(H): width of the best decomposition.

Note: fractional hypertree width $\leq$ generalized hypertree width

Each bag is essentially an instance with bounded fractional cover number, hence there at most $\|I\|^w$ solutions in the projection to a bag.

Fractional hypertree width

The **fractional hypertree width** of a tree decomposition is the minimum value w such that every bag has a fractional cover of weight w.

Fractional cover of a bag B: a weight assignment on the edges such that for each $v \in B$, the total weight of the edges containing v is at least 1. It can be checked in polynomial time if such an assignment of weight at most w exists.

Fractional hypertree width fhw(H): width of the best decomposition.

Note: fractional hypertree width $\leq$ generalized hypertree width

Each bag is essentially an instance with bounded fractional cover number, hence there at most $\|I\|^w$ solutions in the projection to a bag.

Fact: For every w, CSP can be solved in polynomial time if a fractional hypertree decomposition of width w is given in the input.
$\Rightarrow$ If $\mathcal{H}$ has bounded fractional hypertree width, then CSP$(\mathcal{H})$ is FPT.

Approximating fractional hypertree width

To claim polynomial-time solvability, we need a way of finding tree decompositions whose fractional hypertree width is (approximately) fhw(H).

Fact: [M. 2009] For every fixed w, there is a polynomial-time algorithm that either finds a decomposition of fractional hypertree width at most $O(w^3)$, or correctly concludes that fhw$(H) > w$.

$\Rightarrow$ If $\mathcal{H}$ has bounded fractional hypertree width, then CSP$(\mathcal{H})$ is polynomial-time solvable.

The decomposition algorithm uses the separator-based approach.

Key task: find a set S having fractional edge cover number at most w that separates A and B. Surprisingly tricky!

Beyond fractional hypertree width

If a tree decomposition has width/hypertree width/fractional hypertree at most w, then in every bag “we have to consider” at most $\|I\|^w$ satisfying assignments.

Formally, for every bag B, the projection of the instance to B has at most $\|I\|^w$ solutions.

Is there a measure smaller than fractional hypertree width that can be used to bound the number of solutions in the bags of a tree decomposition?

Beyond fractional hypertree width

If a tree decomposition has width/hypertree width/fractional hypertree at most w, then in every bag "we have to consider" at most $\|I\|^w$ satisfying assignments.

Formally, for every bag B, the projection of the instance to B has at most $\|I\|^w$ solutions.

Is there a measure smaller than fractional hypertree width that can be used to bound the number of solutions in the bags of a tree decomposition?

No. If the fractional hypertree width of a decomposition is at least w, then there are (arbitrarily large) instances I where the projection to some bag has $\|I\|^{\Omega(w)}$ solutions.

Beyond fractional hypertree width

It seems that there is no width measure better than fractional hypertree width. We can get around this “optimality” using the following ideas:

- **Idea #1:** The decomposition can depend not only on the hypergraph of the instance, but on the actual constraint relations.
- **Idea #2:** We can branch on adding further restrictions, and apply different tree decompositions to each resulting instance.

Beyond fractional hypertree width

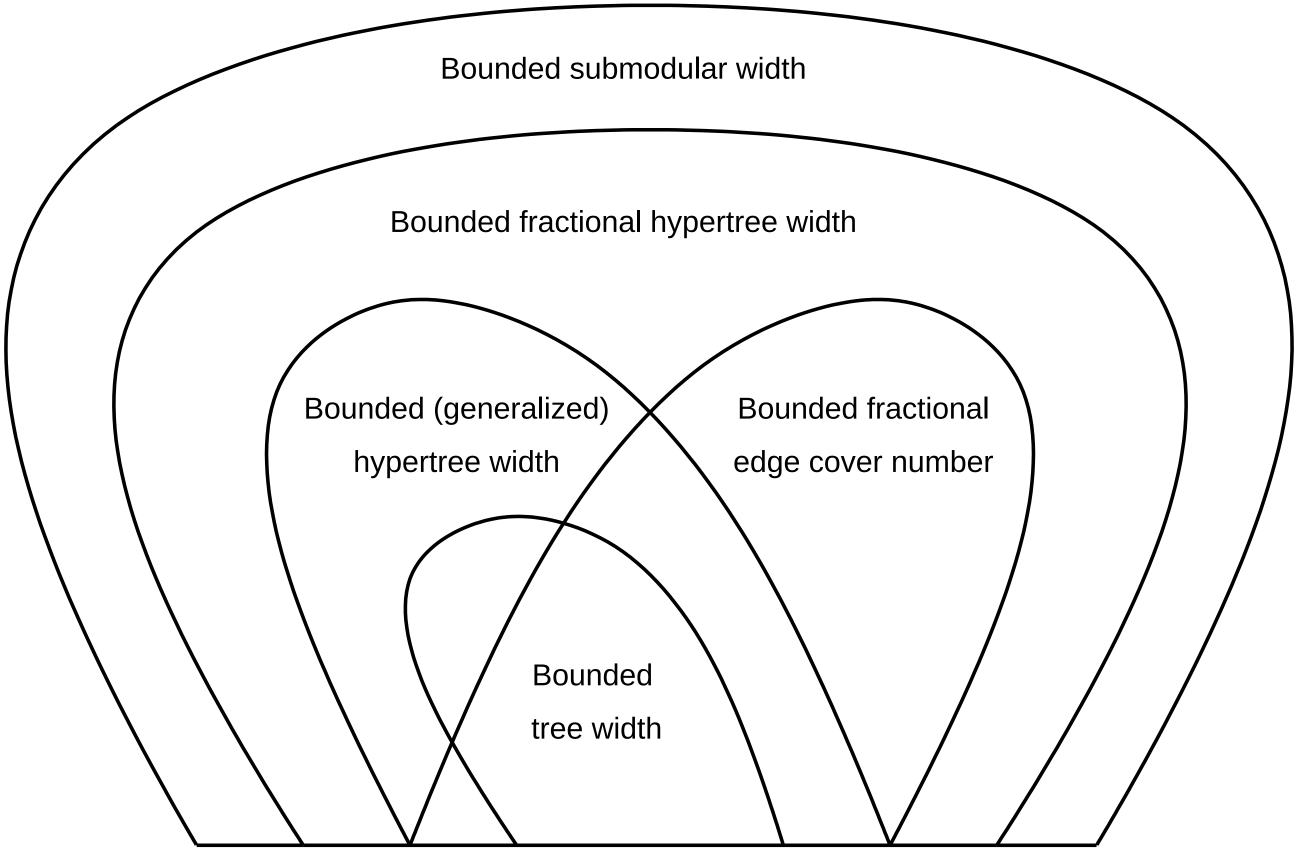
It seems that there is no width measure better than fractional hypertree width. We can get around this “optimality” using the following ideas:

- **Idea #1:** The decomposition can depend not only on the hypergraph of the instance, but on the actual constraint relations.
- **Idea #2:** We can branch on adding further restrictions, and apply different tree decompositions to each resulting instance.

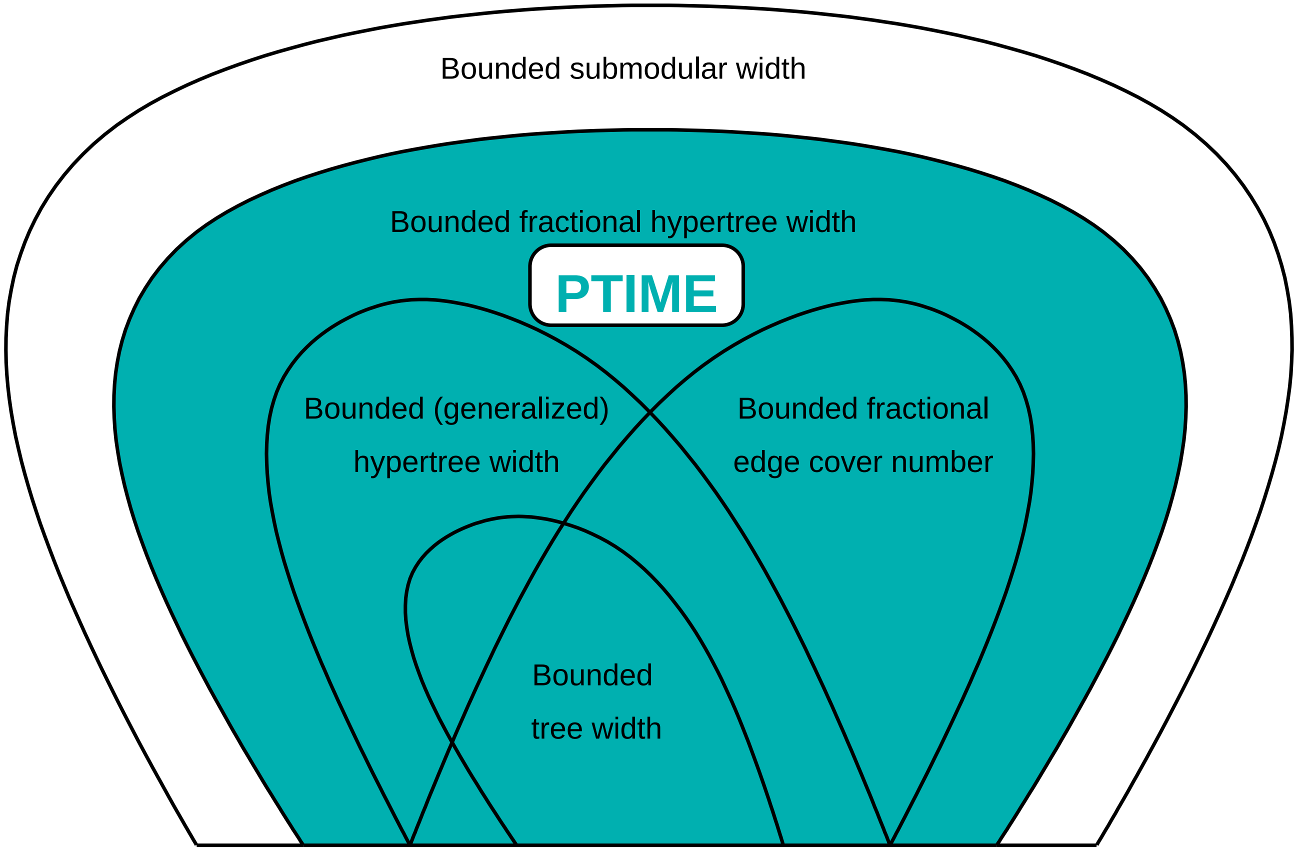
Submodular width: a new width measure that is not larger than fractional hypertree width.

Fact: [M. 2009] Assuming ETH, if $\mathcal{H}$ is a recursively enumerable class of hypergraphs, then CSP$(\mathcal{H})$ is FPT if and only if $\mathcal{H}$ has bounded submodular width.

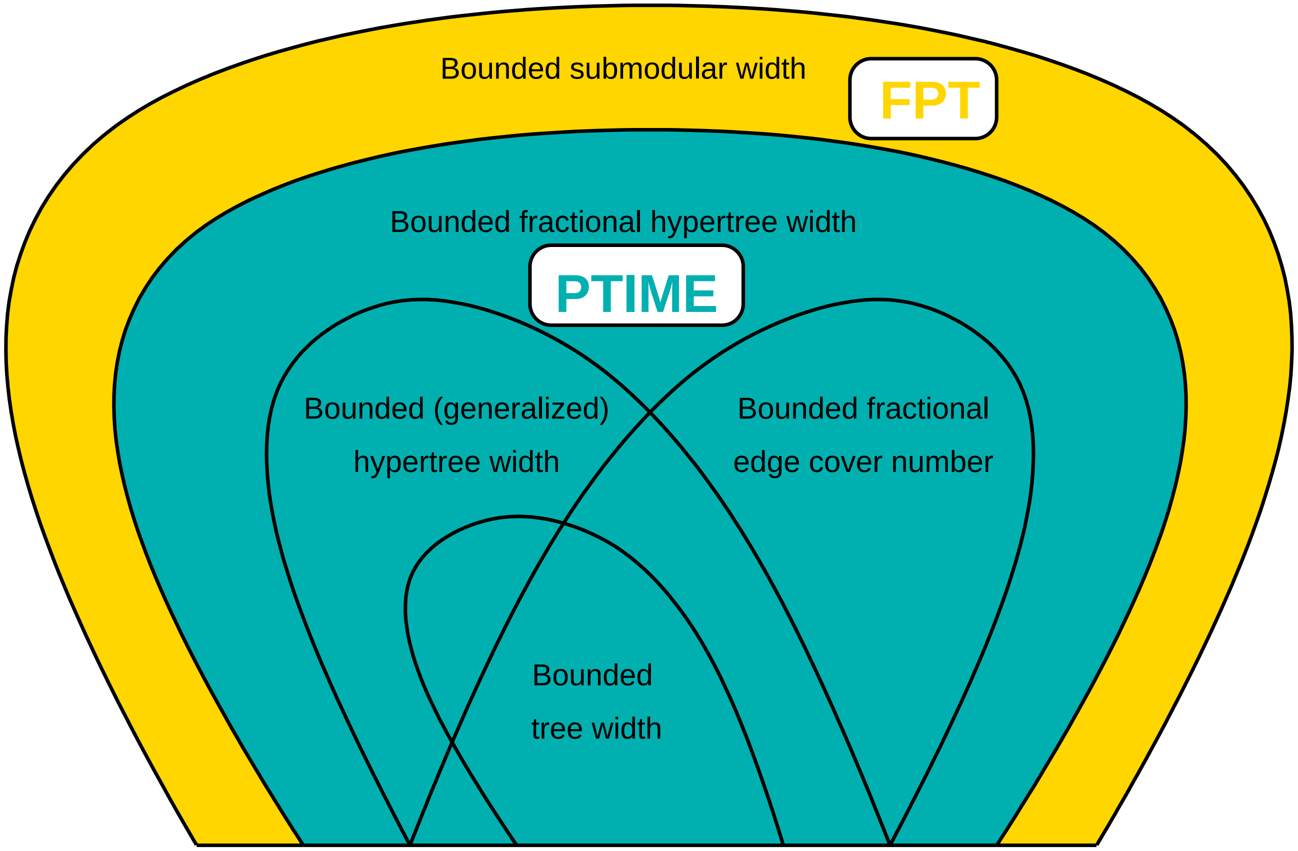
Overview


Bounded submodular width
Bounded fractional hypertree width
Bounded (generalized) hypertree width
Bounded fractional edge cover number
Bounded tree width

Overview

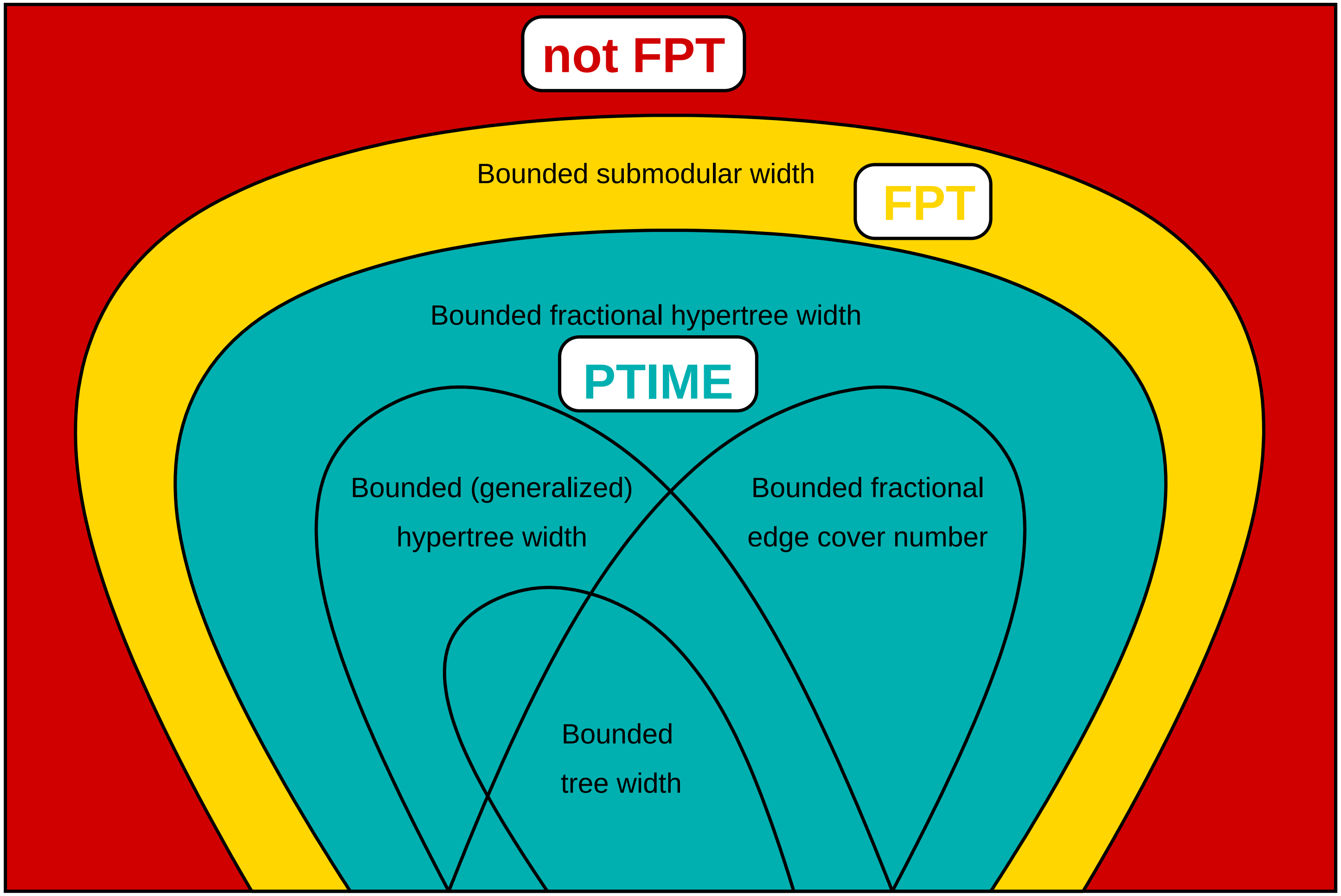

Bounded submodular width
Bounded fractional hypertree width
PTIME
Bounded (generalized)
hypertree width
Bounded fractional
edge cover number
Bounded
tree width

Overview



Bounded submodular width
FPT
Bounded fractional hypertree width
PTIME
Bounded (generalized)
hypertree width
Bounded fractional
edge cover number
Bounded
tree width

Overview


not FPT
Bounded submodular width
FPT
Bounded fractional hypertree width
PTIME
Bounded (generalized)
hypertree width
Bounded fractional
edge cover number
Bounded
tree width