Updated 4/6/2022

# Topic 3: Efficient query evaluation
# Unit 2: Cyclic queries (continued)
# Lecture 20

Wolfgang Gatterbauer

CS7240 Principles of scalable data management (sp22)

https://northeastern-datalab.github.io/cs7240/sp22/

4/1/2022

# Pre-class conversations

- Current topic: reducing cycles to trees
- Keep on commenting on slides and sending me pointers (e.g. email exchange on treewidth for CSPs)

- Today:
  - Reducing cycles to trees (tree decompositions)
  - Reducing cycles in CQs to trees based on the domain or based on atoms (treewidth, query width hypertree decompositions)
  - Linear Programming Duality

# Tree decomposition example 5: the triangle

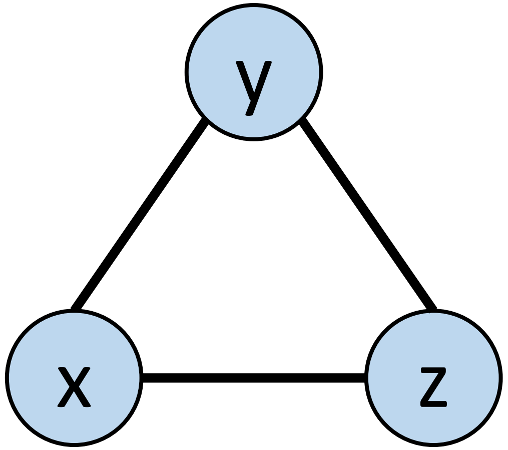
A tree decomposition of graph $G(N, E)$ is a tree $T(V, F)$ and a subset $N_v \subseteq N$ assigned to each vertex (or "supernode") $v \in V$ s.t.:

(1) Node coverage: Every vertex of $G$ is assigned at least one vertex in $T$

(2) Edge coverage: For every edge $e$ of $G$, there is a vertex in $T$ that contains both ends of $e$

(3) Coherence: The tree is "attribute-connected"

The width of a tree decomposition is the size of its largest set minus one




tree decomposition

?

# Tree decomposition example 5: the triangle

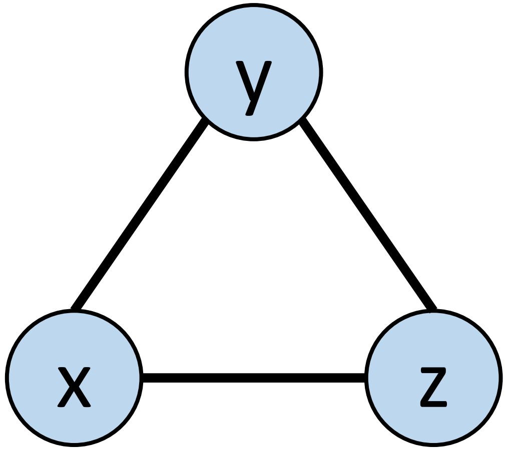
A tree decomposition of graph $G(N, E)$ is a tree $T(V, F)$ and a subset $N_v \subseteq N$ assigned to each vertex (or "supernode") $v \in V$ s.t.:

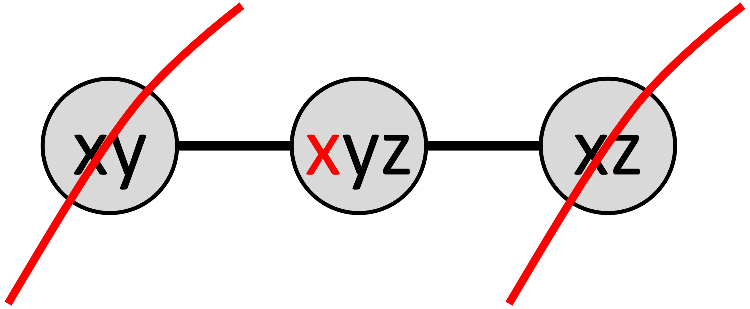
(1) Node coverage: Every vertex of $G$ is assigned at least one vertex in $T$

(2) Edge coverage: For every edge $e$ of $G$, there is a vertex in $T$ that contains both ends of $e$

(3) Coherence: The tree is "attribute-connected"

The width of a tree decomposition is the size of its largest set minus one



# Tree decomposition example 5: the triangle

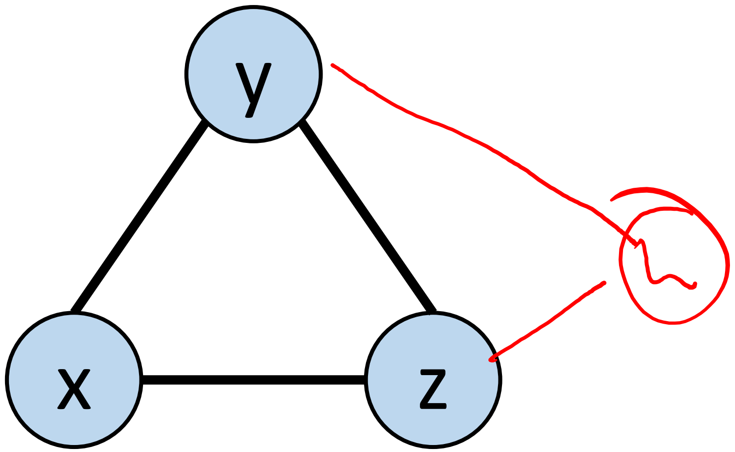
A tree decomposition of graph $G(N, E)$ is a tree $T(V, F)$ and a subset $N_v \subseteq N$ assigned to each vertex (or "supernode") $v \in V$ s.t.:

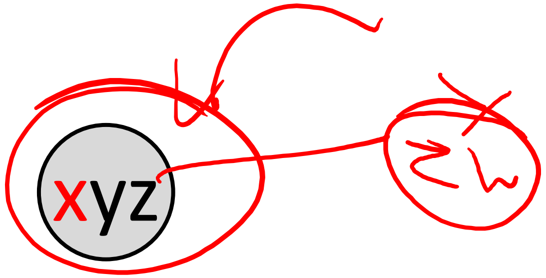
(1) Node coverage: Every vertex of $G$ is assigned at least one vertex in $T$

(2) Edge coverage: For every edge $e$ of $G$, there is a vertex in $T$ that contains both ends of $e$

(3) Coherence: The tree is "attribute-connected"

The width of a tree decomposition is the size of its largest set minus one




More generally, a $K_d$ (d-clique) has a minimal treewidth of d-1

# Tree decomposition example 6: a longer tree

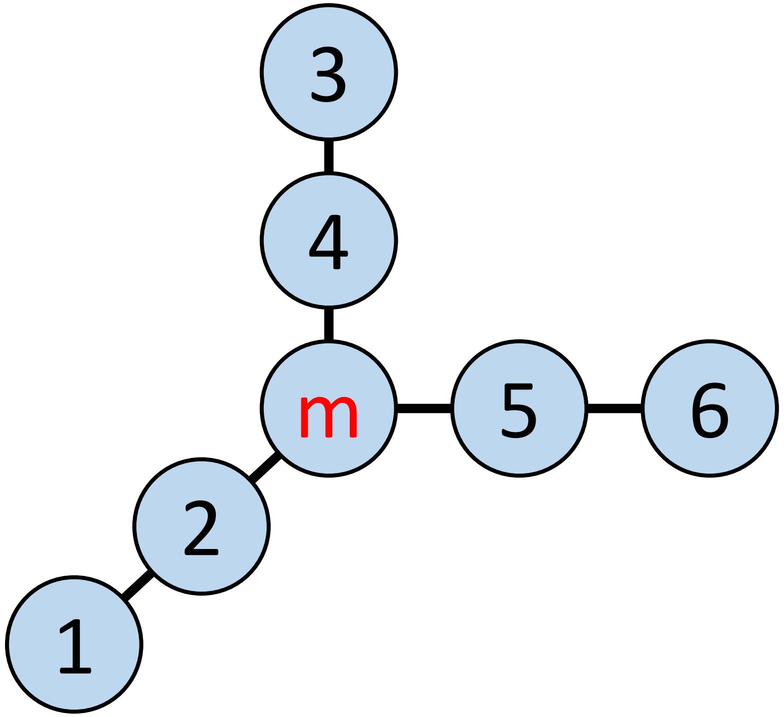
A tree decomposition of graph $G(N, E)$ is a tree $T(V, F)$ and a subset $N_v \subseteq N$ assigned to each vertex (or "supernode") $v \in V$ s.t.:

(1) Node coverage: Every vertex of $G$ is assigned at least one vertex in $T$

(2) Edge coverage: For every edge $e$ of $G$, there is a vertex in $T$ that contains both ends of $e$

(3) Coherence: The tree is "attribute-connected"

The width of a tree decomposition is the size of its largest set minus one




# Tree decomposition example 6: a longer tree

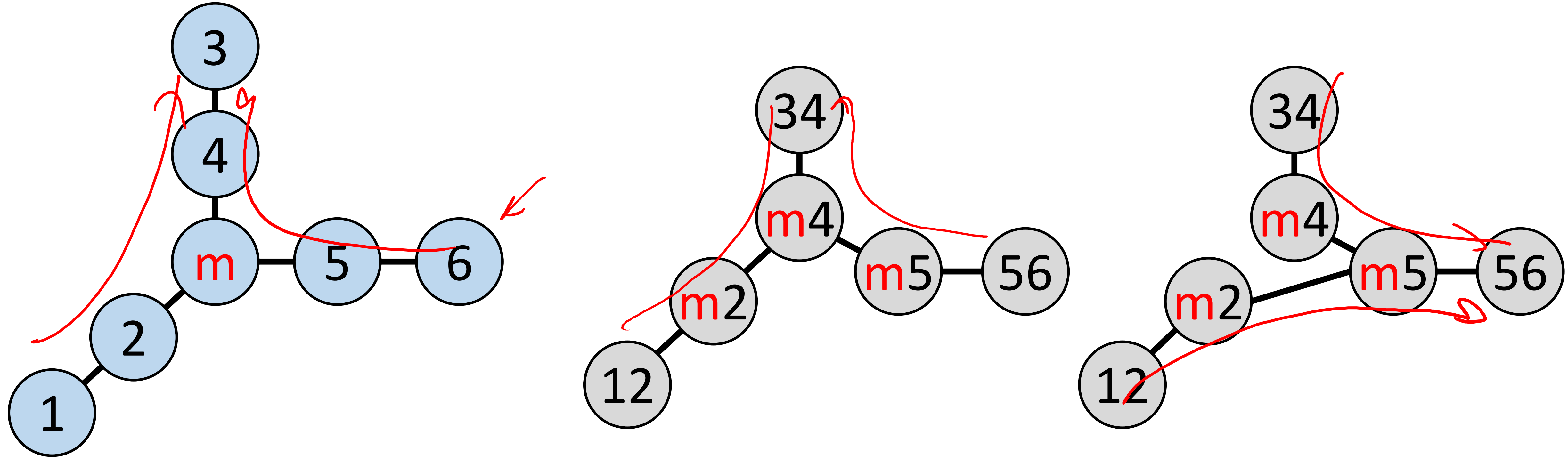
A tree decomposition of graph $\mathrm{G(N, E)}$ is a tree $\mathrm{T(V, F)}$ and a subset $\mathrm{N_v \subseteq N}$ assigned to each vertex (or "supernode") $\mathrm{v \in V}$ s.t.:

(1) Node coverage: Every vertex of $\mathrm{G}$ is assigned at least one vertex in $\mathrm{T}$

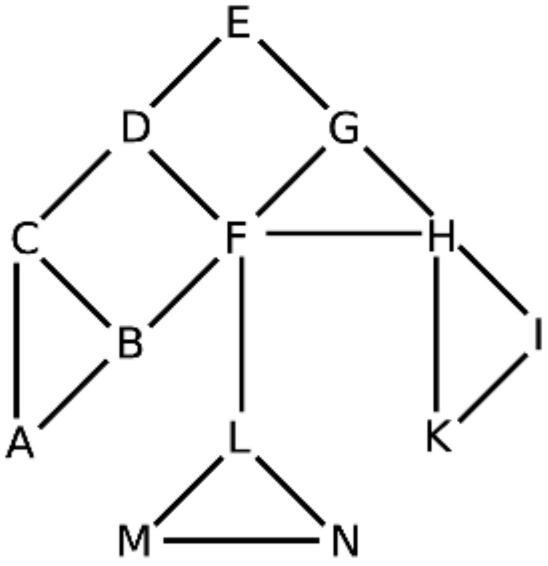
(2) Edge coverage: For every edge $\mathrm{e}$ of $\mathrm{G}$, there is a vertex in $\mathrm{T}$ that contains both ends of $\mathrm{e}$

(3) Coherence: The tree is "attribute-connected"

The width of a tree decomposition is the size of its largest set minus one

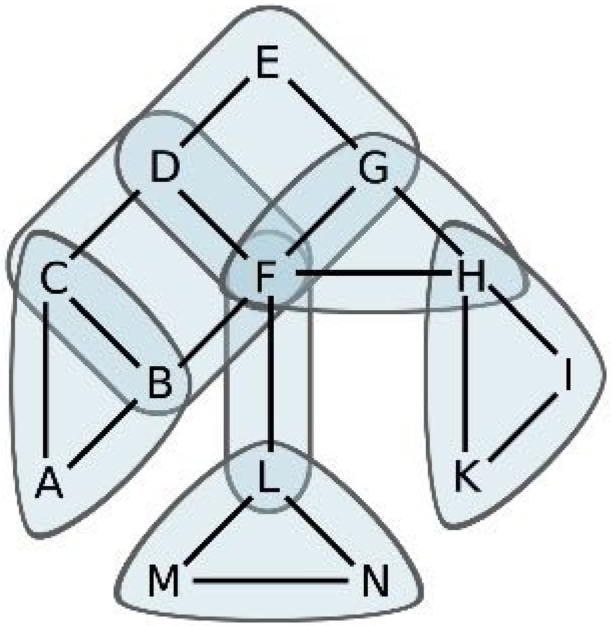


# Tree decomposition example 7

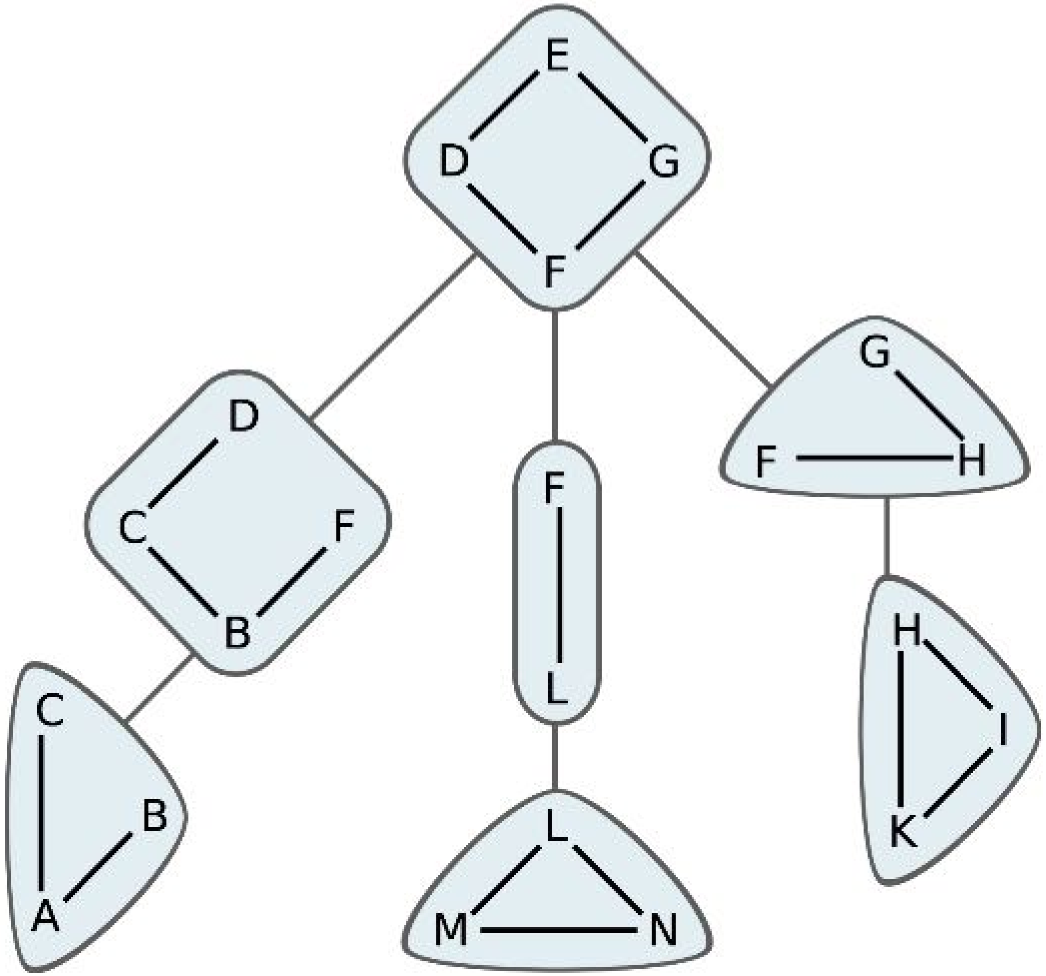



Example by: Markus Krötzsch. "Database theory: Lecture 6: Tree-like Conjunctive Queries." 2016. https://iccl.inf.tu-dresden.de/web/Database_Theory_(SS2016)/en

# Tree decomposition example 7

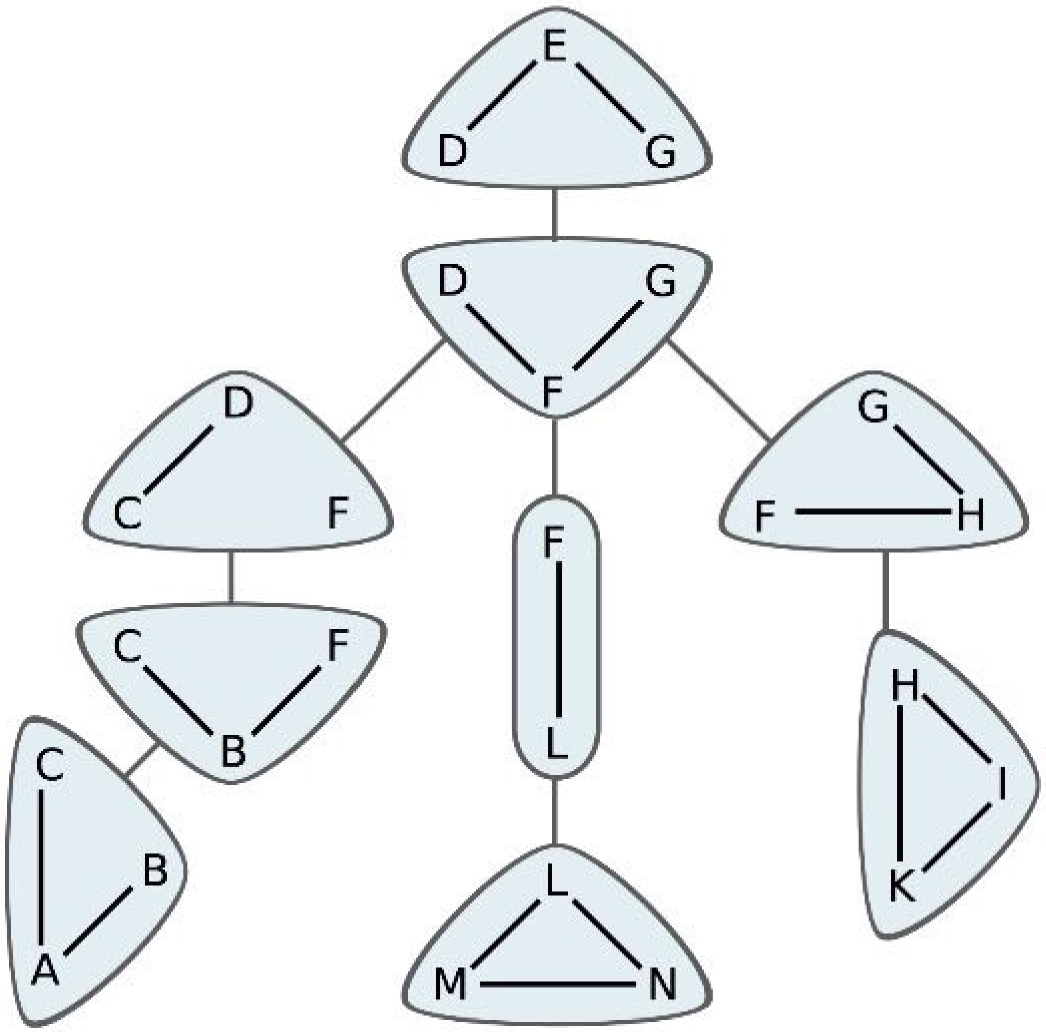


# Tree decomposition example 7



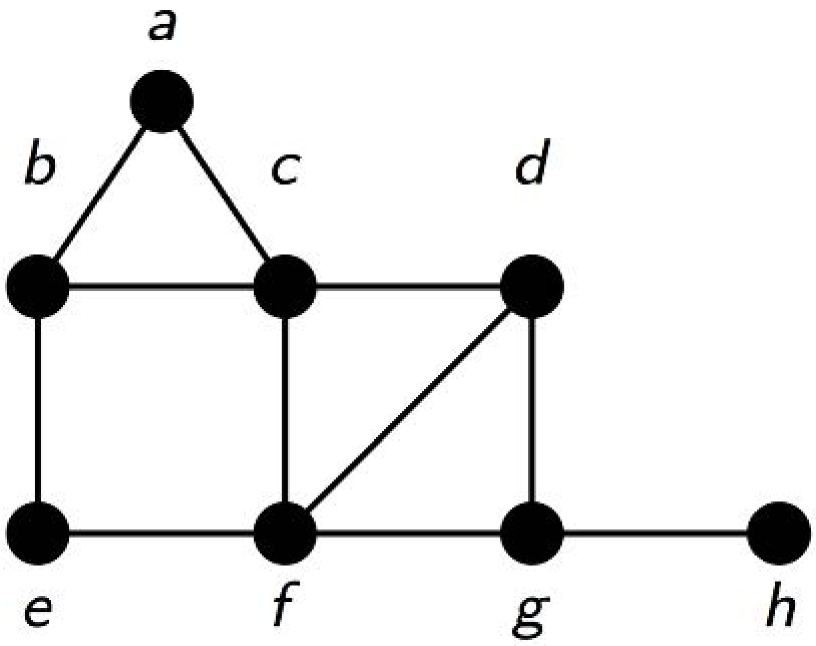

⤳ tree decomposition of width 3

# Tree decomposition example 7



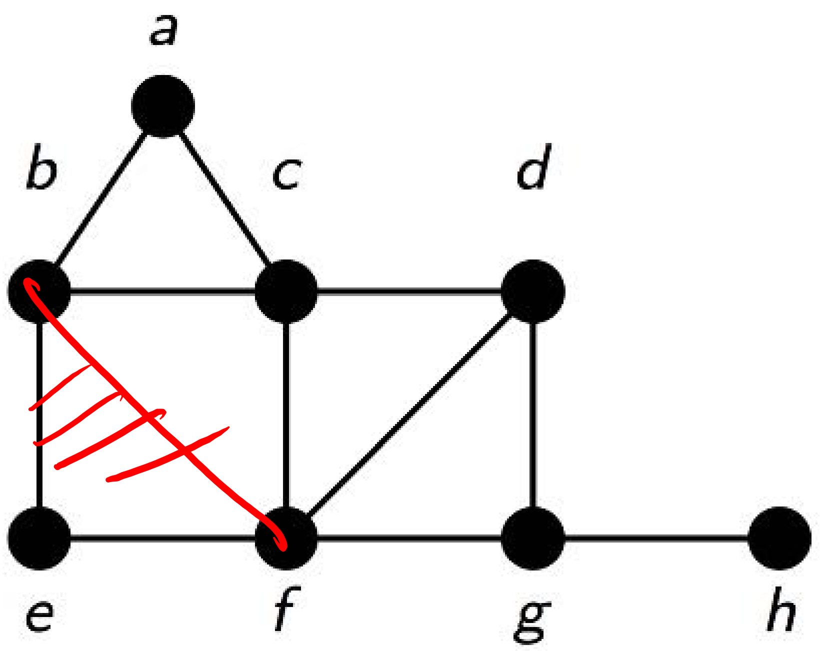

⤳ tree decomposition of width 2 = treewidth of the example graph

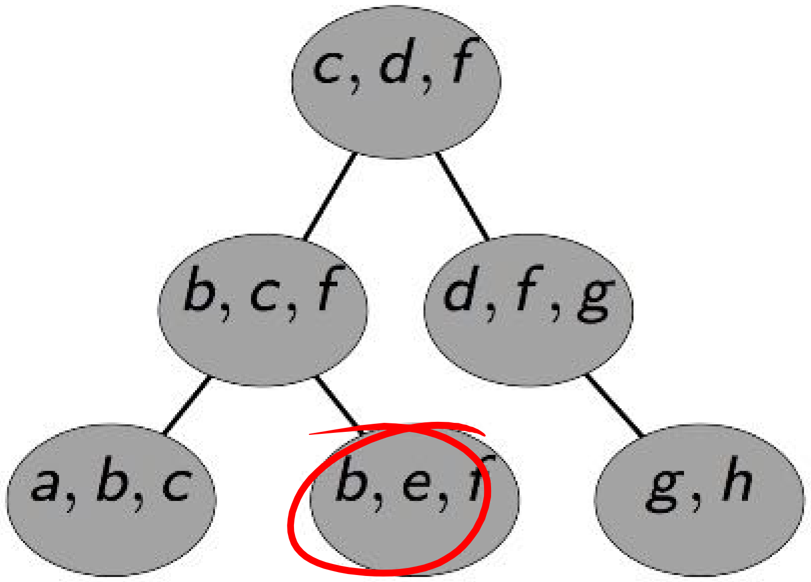
# Tree decomposition example 8



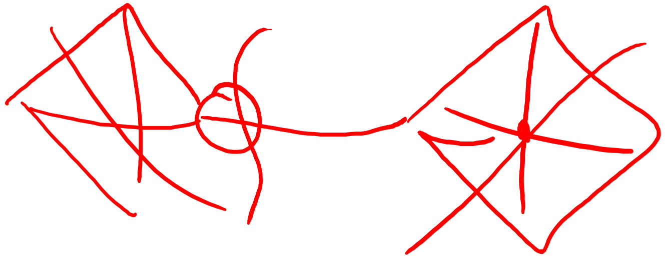

Example by: Marx. "Graphs, hypergraphs, and the complexity of conjunctive database queries", ICDT 2017. http://edbticdt2017.unive.it/marx-icdt2017-talk.pdf

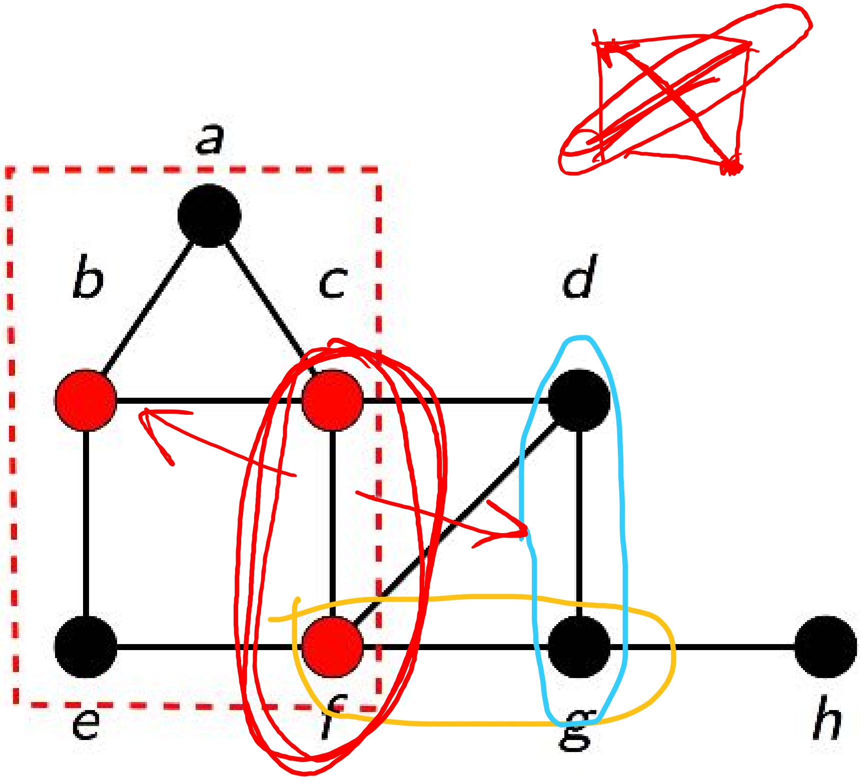
# Tree decomposition example 8

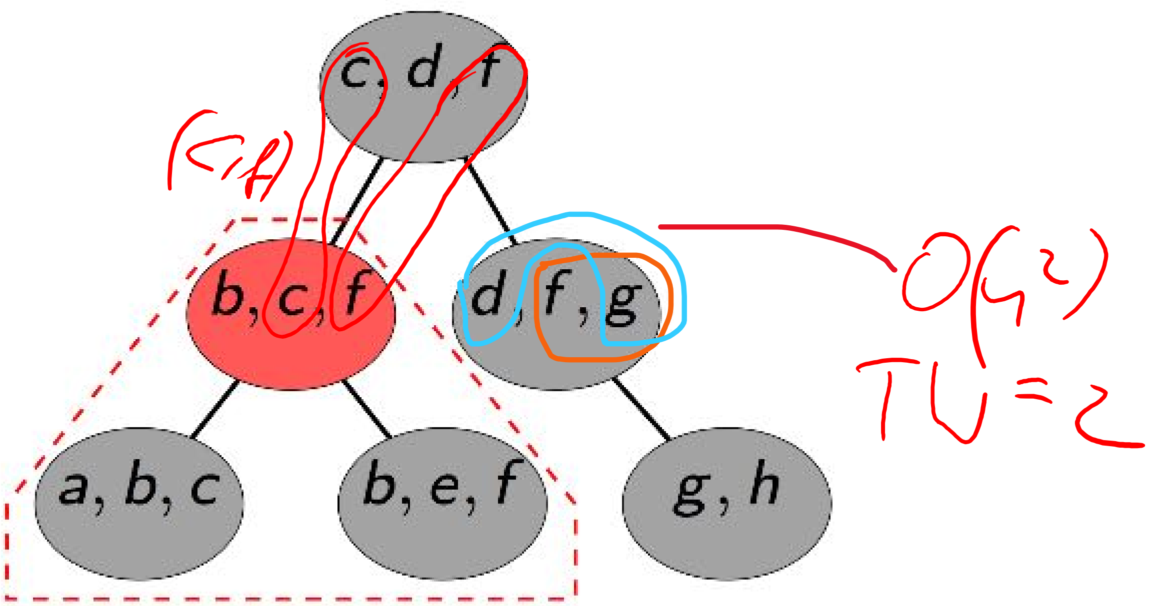


# Tree decomposition example 8



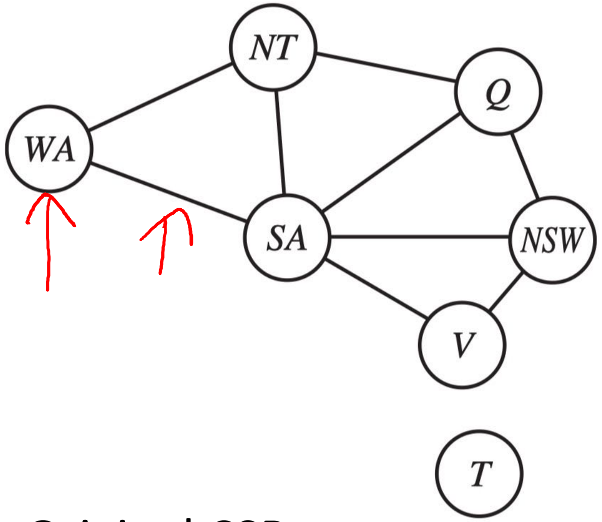


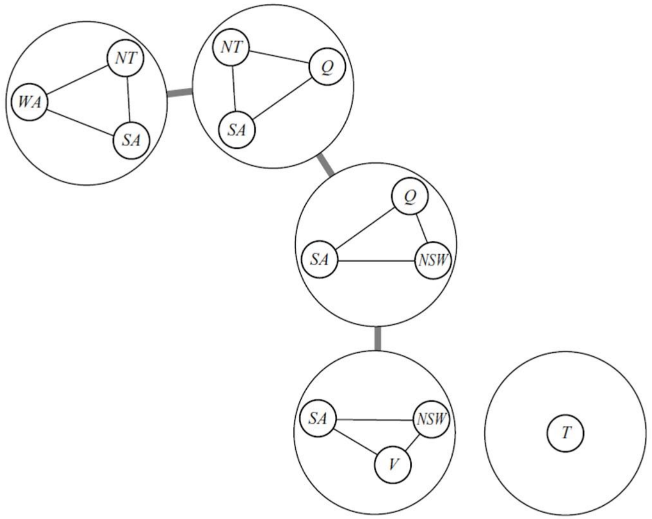

A subtree communicates with the outside world only via the root of the subtree.

# Tree Decompositions (TDs) for CSPs

Notice here each node is a variable with domain of size d (e.g. 3 colors)




Original CSP:
Map-coloring of Australia




Tree decomposition with supernodes (sets of variables)

TD:

- If two variables are connected in the original problem, they must appear together (along with the constraint) in at least one supernode
- If a variable occurs in two supernodes in the TD, it must appear in every supernode on the path that connects the two (coherence)
- The only constraints between the supernodes are that the variables take on the same values across supernodes (like semi-join messages from Yannakakis)

Translates into $O(n^{tw})$ where n is size of constraints per edge

- Solving CSP on a tree with k variables and domain size m is $O(km^2)$
- TD algorithm: find all solutions within each supernode, which is $O(m^{tw+1})$ where tw is the treewidth (= one less than size of largest supernode). Recall treewidth of tree is 1, thus complexity $O(m^2)$
- Then, use the tree-structured Yannakakis algorithm, treating the supernodes as new variables...
- Finding a tree decomposition of smallest treewidth is NP-complete, but good heuristic methods exist.

Figures: Fig 6.12 and 6.13 from Russell, Norvig. "Artificial intelligence: a modern approach". 3rd ed, 2010. https://dl.acm.org/doi/book/10.5555/1671238

# Alternative definition of Tree decomposition (TD)

A tree decomposition of graph $G(N, E)$ is a tree $T(V, F)$ and a subset $N_v \subseteq N$ assigned to each vertex $v \in V$ s.t.:

(1) Node coverage: Every vertex of $G$ is assigned at least one vertex in $T$

(2) Edge coverage: For every edge $e$ of $G$, there is a vertex in $T$ that contains both ends of $e$

(3) Coherence: The tree is "attribute-connected"

The width of a tree decomposition is the size of its largest set minus one

Alternative Definition:

A tree decomposition of graph $G(N, E)$ is a pair $\langle T, \chi \rangle$ where $T(V, F)$ is a tree, and $\chi$ is a labeling function assigning to each vertex $v \in V$ a set of vertices $\chi(v) \subseteq N$, s.t. above conditions (2) and (3) are satisfied.

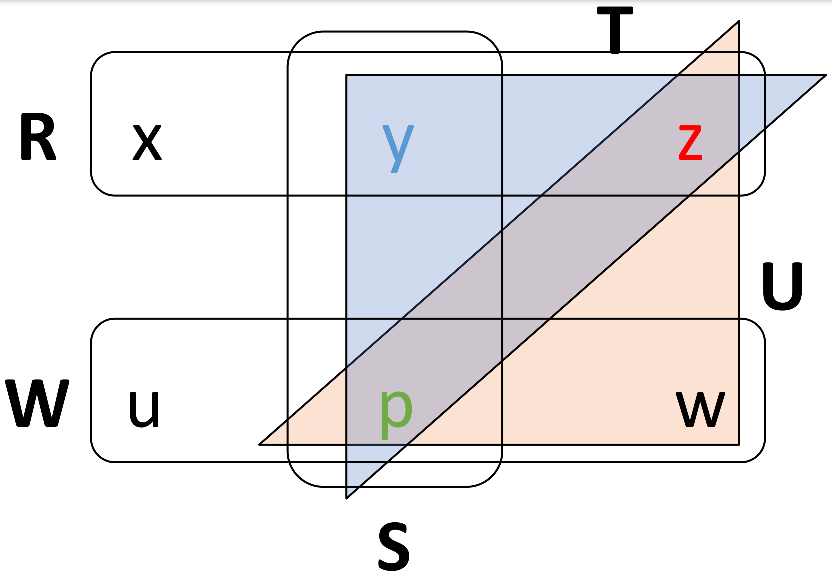
# Outline: T3-2: Cyclic conjunctive queries

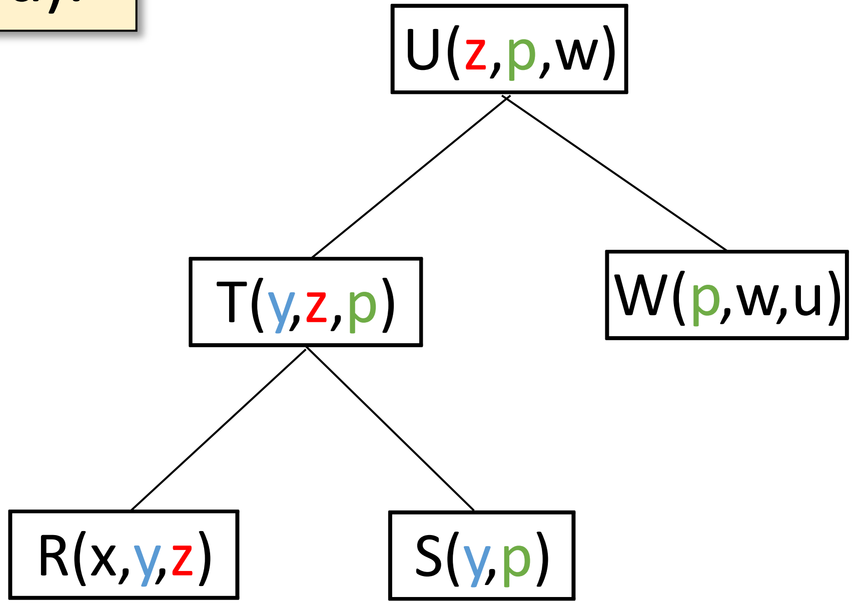
- T3-1: Acyclic conjunctive queries
- T3-2: Cyclic conjunctive queries
  - 2SAT (a detour)
  - Tree decompositions
  - Decompositions of hypertrees
  - Duality in Linear programming (a quick primer)
  - AGM bound (maximal result size for full CQs)
  - Worst-case optimal joins & the triangle query
  - Worst-case optimal joins & the 4-cycle
  - Optimal joins & the 4-cycle

# Acyclic Conjunctive Queries

- A join tree for a hypergraph H=(V,E) is a labeled tree T =(N,F,$\lambda$) such that:
  - The nodes of T are formed by the hyperedges. In other words, $\lambda$: N→E s.t. for each hyperedge e ∈ E of H, there exists n ∈ N such that e = $\lambda$(n)
  - For each node u ∈ V of H, the set {n ∈ N | u ∈ $\lambda$(n)} induces a connected subtree of T. (also called: running intersection property)

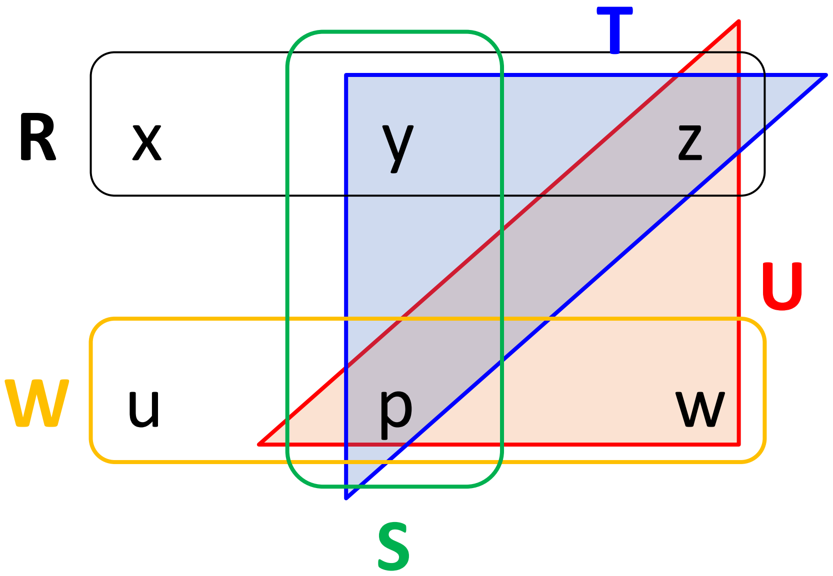
Q :- R(x,y,z), S(y,p), T(y,z,p), U(z,p,w), W(p,w,u).

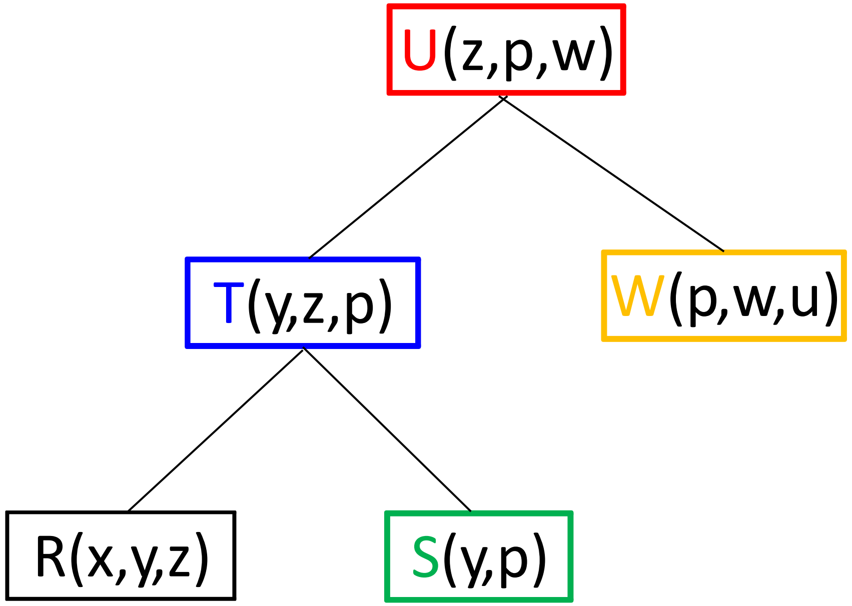


# Acyclic Conjunctive Queries

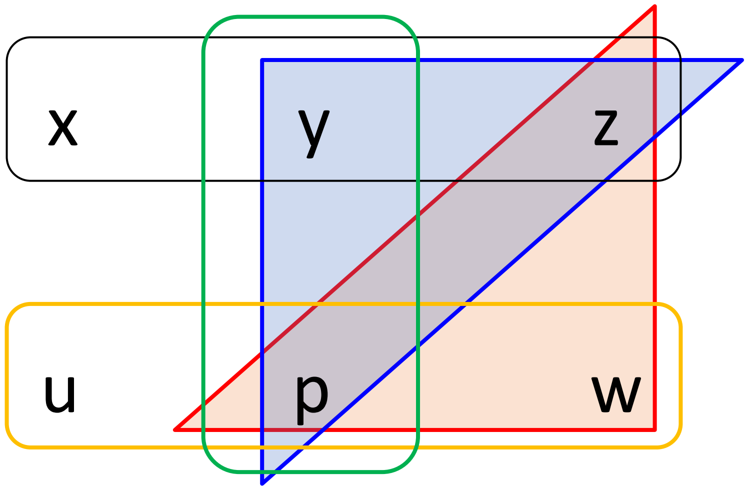
- A join tree for a hypergraph H=(V,E) is a labeled tree T =(N,F,$\lambda$) such that:
  - The nodes of T are formed by the hyperedges. In other words, $\lambda$: N→E s.t. for each hyperedge e ∈ E of H, there exists n ∈ N such that e = $\lambda$(n)
  - For each node u ∈ V of H, the set {n ∈ N | u ∈ $\lambda$(n)} induces a connected subtree of T. (also called: running intersection property)

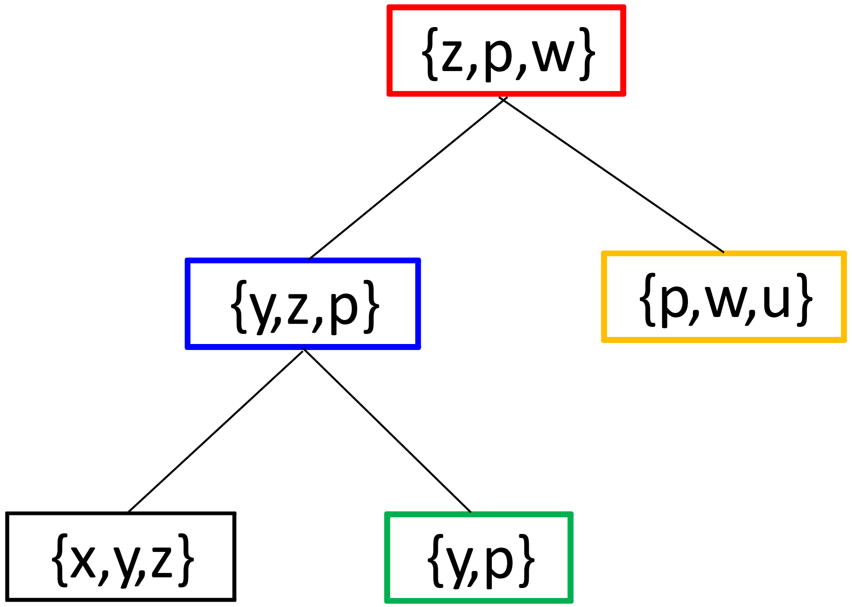


# Acyclic Conjunctive Queries

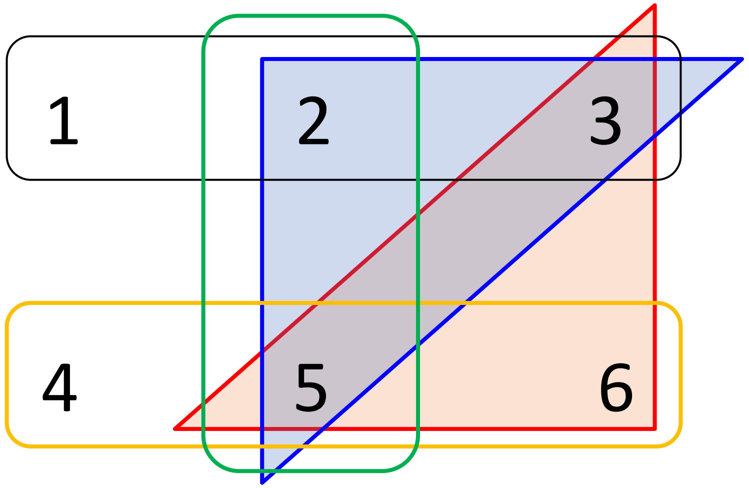
- A join tree for a hypergraph H=(V,E) is a labeled tree T =(N,F,$\lambda$) such that:
  - The nodes of T are formed by the hyperedges. In other words, $\lambda$: N→E s.t. for each hyperedge e ∈ E of H, there exists n ∈ N such that e = $\lambda$(n)
  - For each node u ∈ V of H, the set {n ∈ N | u ∈ $\lambda$(n)} induces a connected subtree of T. (also called: running intersection property)

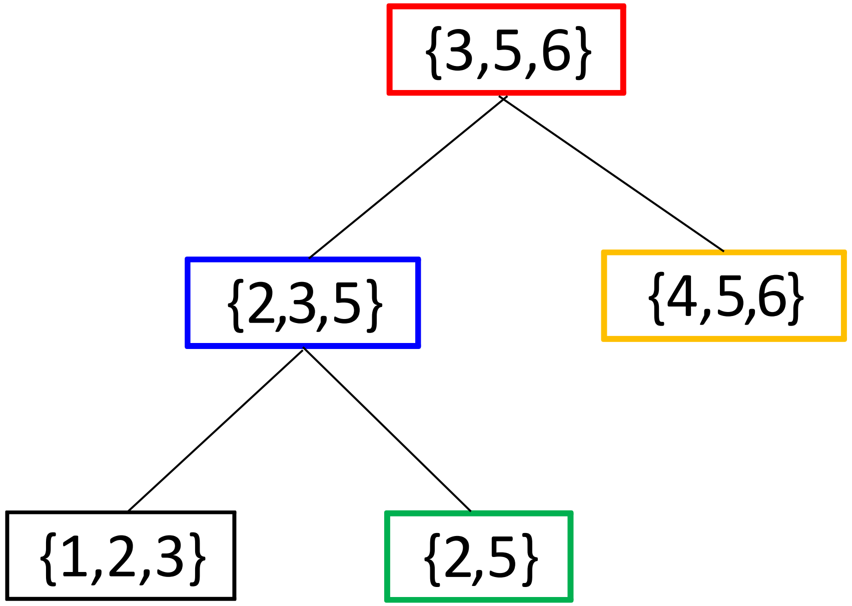


# Acyclic Conjunctive Queries

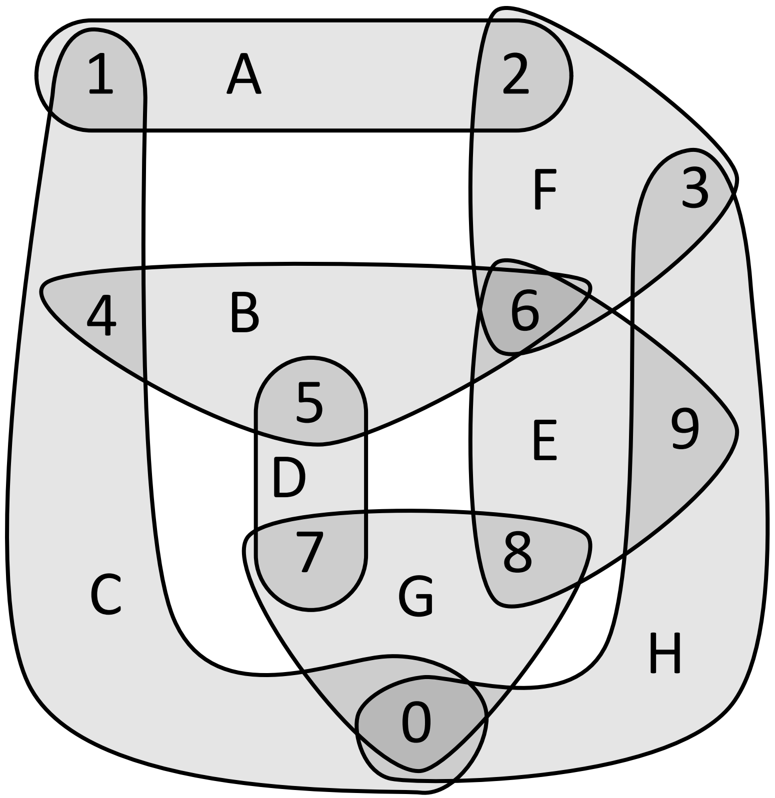
- A join tree for a hypergraph H=(V,E) is a labeled tree T =(N,F,$\lambda$) such that:
  - The nodes of T are formed by the hyperedges. In other words, $\lambda$: N→E s.t. for each hyperedge e ∈ E of H, there exists n ∈ N such that e = $\lambda$(n)
  - For each node u ∈ V of H, the set {n ∈ N | u ∈ $\lambda$(n)} induces a connected subtree of T. (also called: running intersection property)



# Cyclic Conjunctive Queries

Hypergraph




For queries that are not acyclic, what bounds can we give on the data complexity of query evaluation, considering various structural properties of the query?

We will see:

- Coherence (as in TDs) are still a key structural criterion for efficiency!
- But Treewidth by itself is not a good bound. Number of atoms needed to cover sets of variables will help ☺.
- Reason: size of database is determined by number of tuples n not domain size m

# Issues with standard Treewidth (TW) for CQs

Treewidth based on graphs.
TW of CQ is TW of its clique graph (i.e. replace each hyperedge with a clique)

a clique is a graph where where every vertex is connected to every other vertex

Q(x,y,z,w) :- R(x,y,z,w).

Hypergraph

?

Clique graph

?

Treewidth: ?

# Issues with standard Treewidth (TW) for CQs

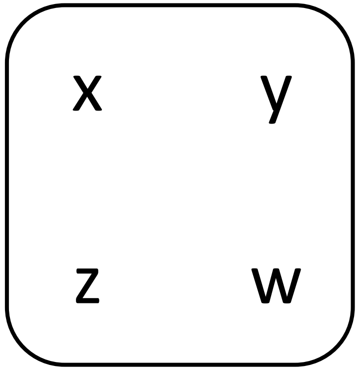
Treewidth based on graphs.

TW of CQ is TW of its clique graph (i.e. replace each hyperedge with a clique)

a clique is a graph where where every vertex is connected to every other vertex

Q(x,y,z,w) :- R(x,y,z,w).

Hypergraph




Clique graph

?

Treewidth: ?

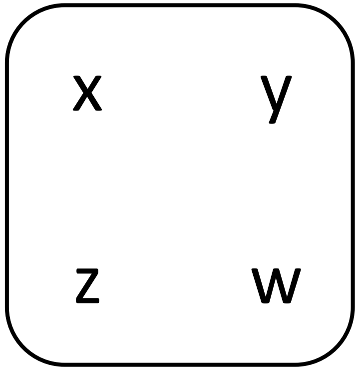
# Issues with standard Treewidth (TW) for CQs

Treewidth based on graphs.
TW of CQ is TW of its clique graph (i.e. replace each hyperedge with a clique)

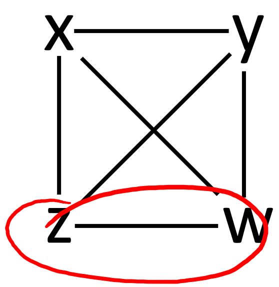
a clique is a graph where where every vertex is connected to every other vertex

Q(x,y,z,w) :- R(x,y,z,w).

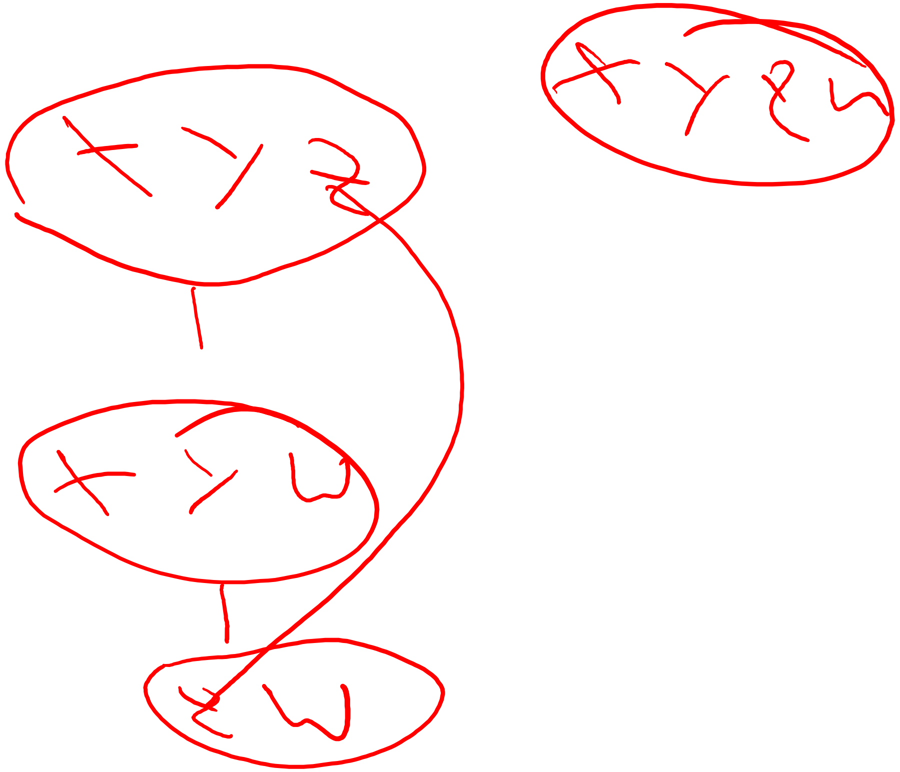
Hypergraph




Clique graph




Treewidth: ?



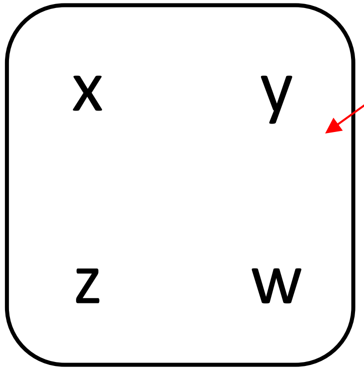
# Issues with standard Treewidth (TW) for CQs

Treewidth based on graphs.
TW of CQ is TW of its clique graph (i.e. replace each hyperedge with a clique)

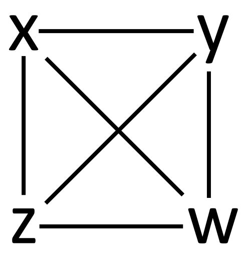
Q(x,y,z,w) :- R(x,y,z,w).

This is actually the best tree decomposition: Nodes of a clique need to appear in the same supernode

Hypertree




Clique graph




Treewidth: 3

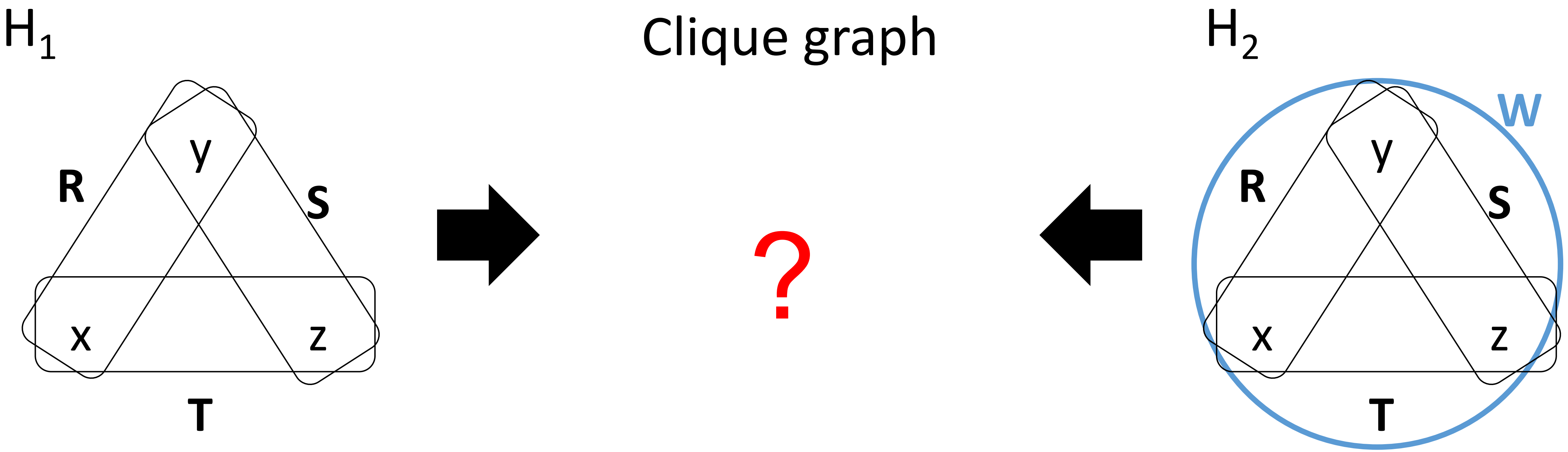
Resulting complexity bound $O(n^3)$!

That's a pretty bad bound. We know we can evaluate this query in $O(n)$.

# Issues with standard Treewidth (TW) for CQs

$Q_1(x,y,z)$ :- $R(x,y)$, $S(y,z)$, $T(x,z)$.
$Q_2(x,y,z)$ :- $R(x,y)$, $S(y,z)$, $T(x,z)$, $W(x,y,z)$.

We also know that these two queries have different maximal output sizes: $O(n^{1.5})$ vs. $O(n)$.
But TW cannot distinguish them ☹



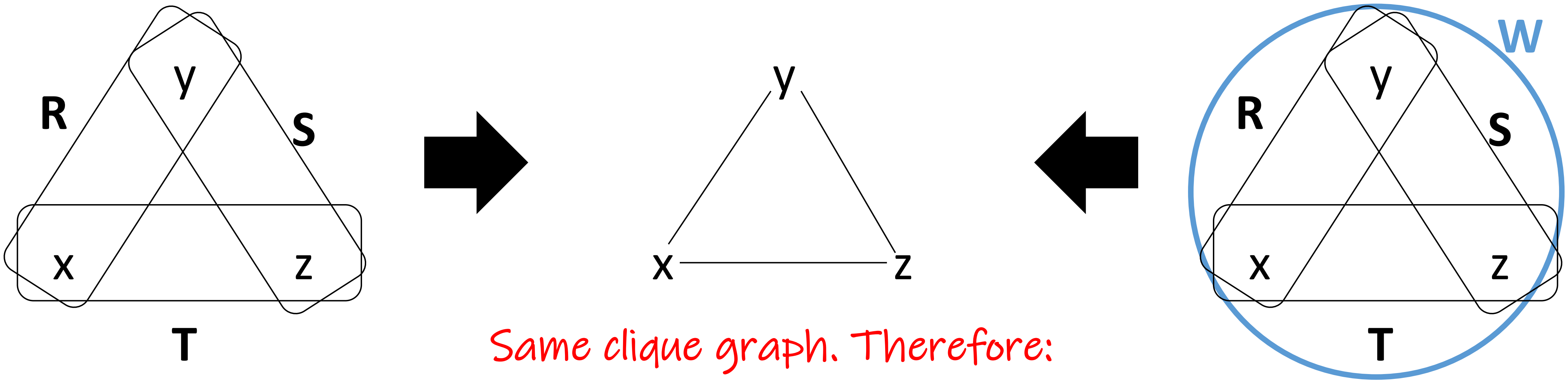
# Issues with standard Treewidth (TW) for CQs

$Q_1(x,y,z)$ :- R(x,y), S(y,z), T(x,z).
$Q_2(x,y,z)$ :- R(x,y), S(y,z), T(x,z), W(x,y,z).

We also know that these two queries have different maximal output sizes: $O(n^{1.5})$ vs. $O(n)$.
But TW cannot distinguish them ☹

R,S | x y z

$H_1$ Clique graph $H_2$




Same clique graph. Therefore:
→ same TW 2.
→ same complexity bound $O(m^2)$

# "Query decomposition" [Chekuri, Rajaraman'97]

QUERY DECOMPOSITION
Tree decomposition with coherence conditions on both: 1) variables and 2) atoms.
Query width: max # of atoms in a supernode

A *query decomposition* of $Q$ is a tree $T=(I,F)$, with a set $X(i)$ of subgoals and arguments associated with each vertex $i \in I$, such that the following conditions are satisfied:

- For each subgoal $s$ of $Q$, there is an $i \in I$ such that $s \in X(i)$.
- For each subgoal $s$ of $Q$, the set $\{i \in I \mid s \in X(i)\}$ induces a (connected) subtree of $T$.
- For each argument $A$ of $Q$, the set

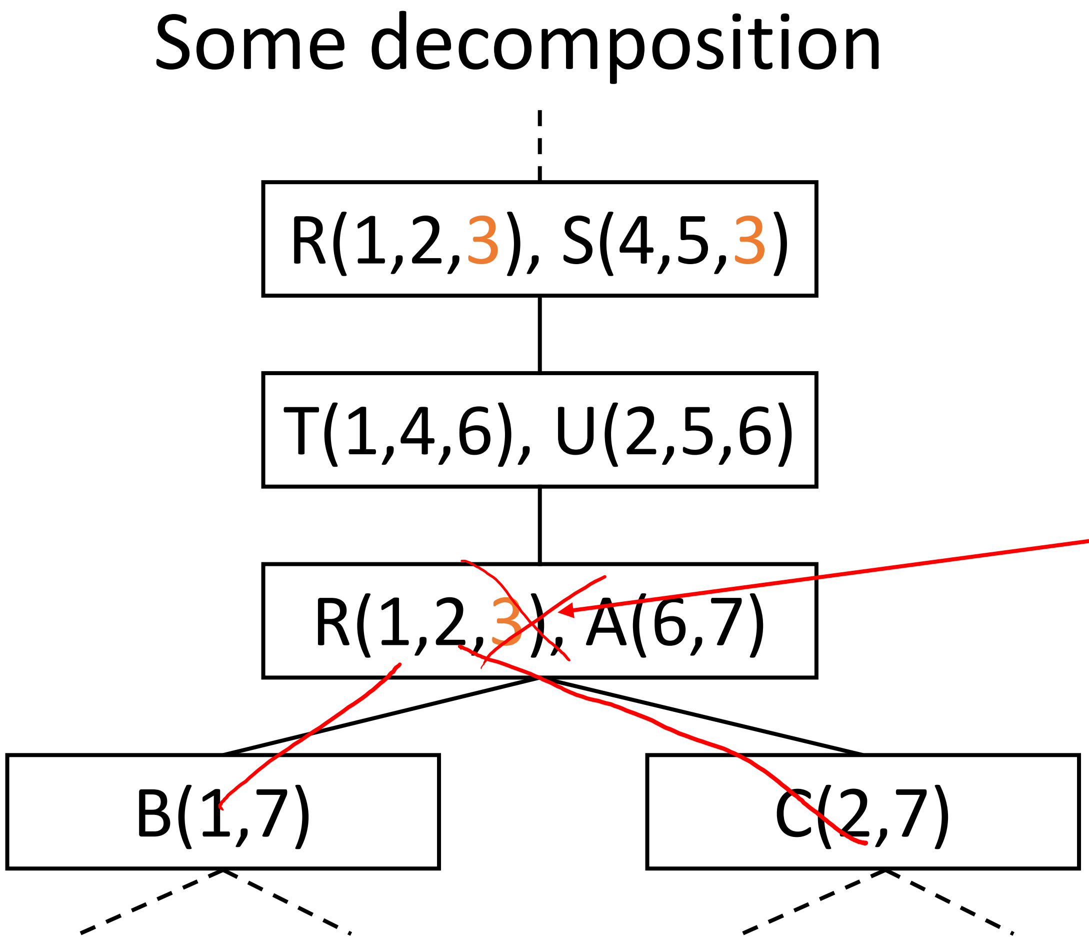
$$\{i \in I \mid A \in X(i)\} \cup \{i \in I \mid A \text{ appears in a subgoal } s \text{ such that } s \in X(i)\}$$

induces a (connected) subtree of $T$.

The *width* of the query decomposition is $\max_{i \in I} |X(i)|$. The *query width* of $Q$ is the minimum width over all its query decompositions.

Chekuri, Rajaraman. "Conjunctive query containment revisited", TCS 2000. https://doi.org/10.1016/S0304-3975(99)00220-0 (ICDT'97 conference paper, ICDT'16 test-of-time award)

# Important Observation 1



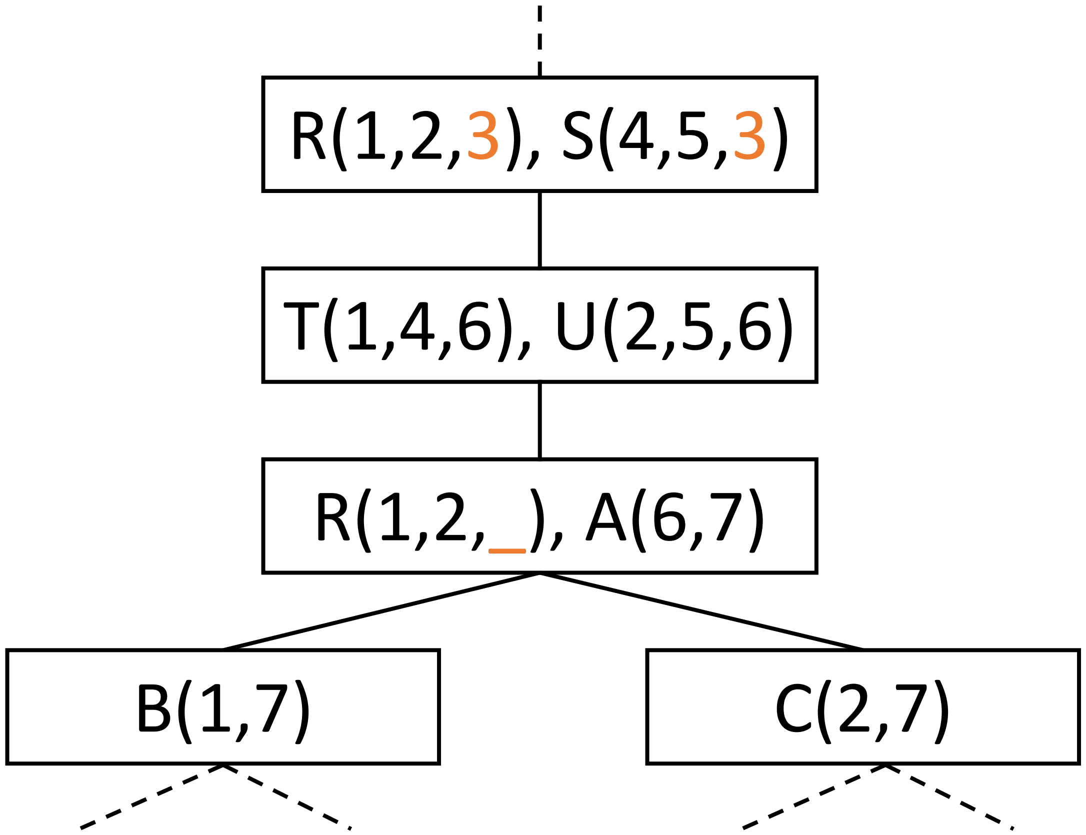

"Query decomposition" as defined by [Chekuri, Rajaraman'97] is too strict about atoms needing to be connected and atoms not allowing projections

This decomposition would not possible for original "query decomposition"

Adopted from an example by Georg Gottlob

# Important Observation 1

Some decomposition




Here the reuse of R(1,2,3) is harmless: we could have added an atom R(1,2,_) here without changing the query.

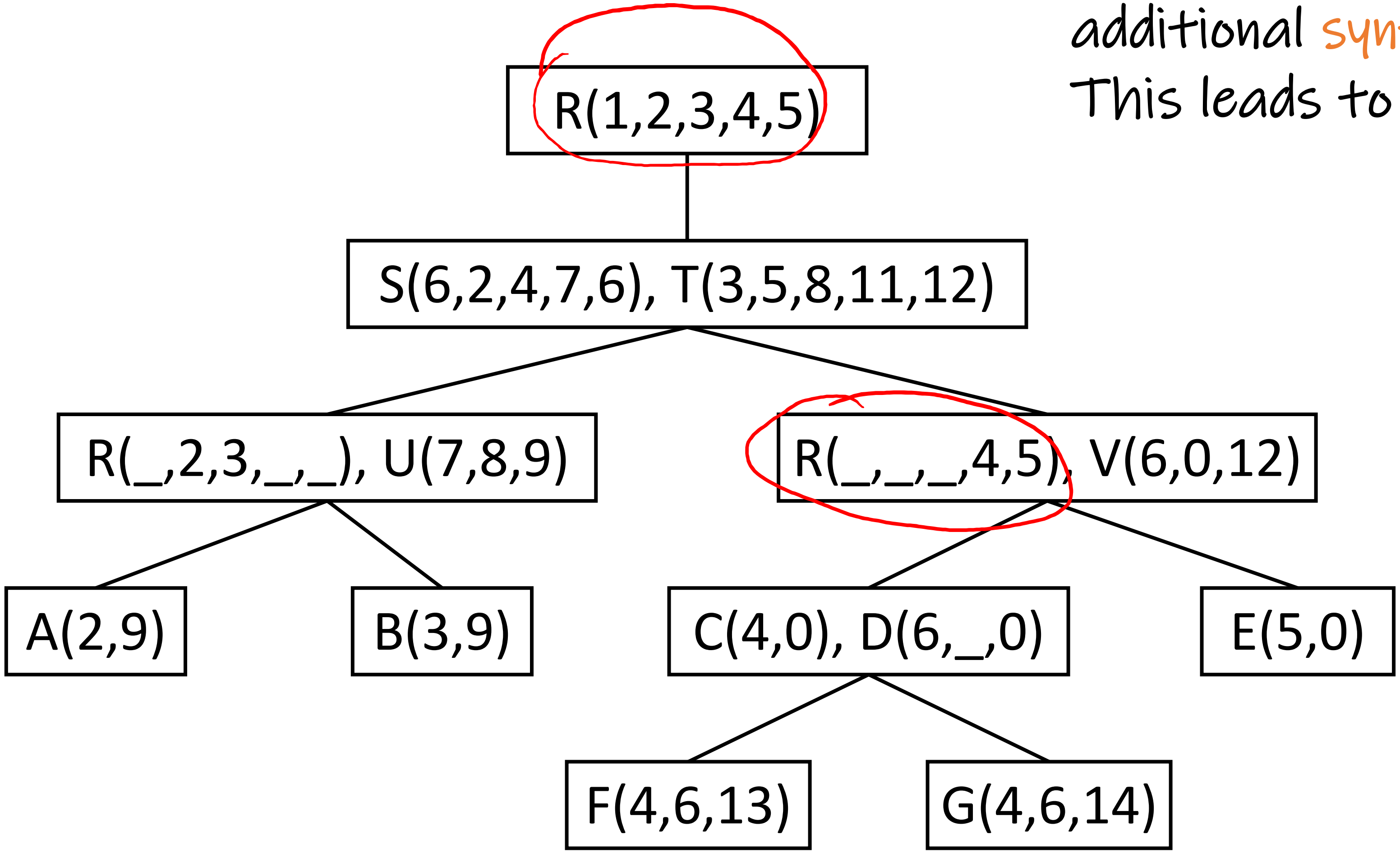
Idea: allow query atoms to be reused partially (with projections) as long as the full atom appears somewhere else.

This leads to "generalized hypertree decompositions" which define coherence only based on variables, not atoms. More liberal than "query decomposition", and thus can give tighter bounds.

Adopted from an example by Georg Gottlob

# Important Observation 2

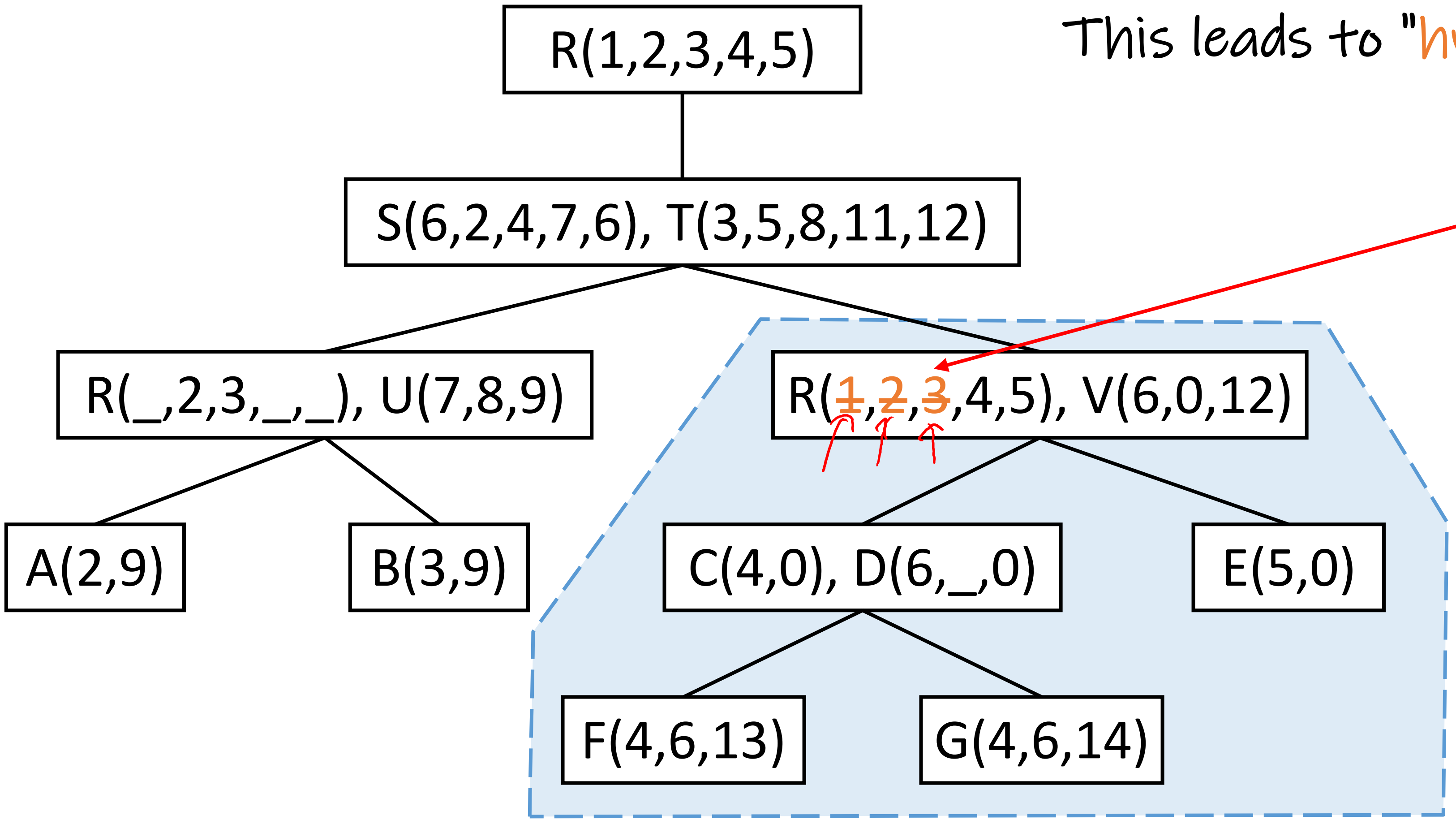
One can avoid NP-hardness of finding a minimal size decomposition by adding an additional syntactic "descendant condition".
This leads to "hypertree decompositions"




Adopted from an example by Georg Gottlob

# Important Observation 2

One can avoid NP-hardness of finding a minimal size decomposition by adding an additional syntactic "descendant condition". This leads to "hypertree decompositions"




Each variable that disappears at some node, does not reappear in the subtree rooted at that node

Adopted from an example by Georg Gottlob

# HYPERTREE DECOMPOSITIONS AND TRACTABLE QUERIES *


Georg Gottlob
Inst. für Informationssysteme
Technische Universität Wien
A-1040 Vienna, Austria
gottlob@dbai.tuwien.ac.at

Nicola Leone
Inst. für Informationssysteme
Technische Universität Wien
A-1040 Vienna, Austria
leone@dbai.tuwien.ac.at

Francesco Scarcello
ISI-CNR
Via P. Bucci 41/C
I-87030 Rende, Italy
scarcello@si.deis.unical.it



**Abstract**

Several important decision problems on conjunctive queries (CQs) are NP-complete in general but become tractable, and actually highly parallelizable, if restricted to acyclic or nearly acyclic queries. Examples are the evaluation of Boolean CQs and query containment. These problems were shown tractable for conjunctive queries of bounded treewidth [9], and of bounded degree of cyclicity [24, 23]. The so far most general concept of nearly acyclic queries was the notion of queries of bounded query-width introduced by Chekuri and Rajaraman [9]. While CQs of bounded query-width are tractable, it remained unclear whether such queries are efficiently recognizable. Chekuri and Rajaraman [9] stated as an open problem whether for each constant $k$ it can be determined in polynomial time if a query has query width $\leq k$. We give a negative answer by proving this problem NP-complete (specifically, for $k = 4$). In order to circumvent this difficulty, we introduce the new concept of hypertree decomposition of a query and the corresponding notion of hypertree width. We prove: (a) for each $k$, the class of queries with query width bounded by $k$ is properly contained in the class of queries whose hypertree width is bounded by $k$; (b) unlike query width, constant hypertree-width is efficiently recognizable; (c) Boolean queries of constant hypertree-width can be efficiently evaluated.


descendent condition

**Definition 3.1** A *hypertree decomposition* of a conjunctive query $Q$ is a hypertree $\langle T, \chi, \lambda \rangle$ for $Q$ which satisfies all the following conditions:

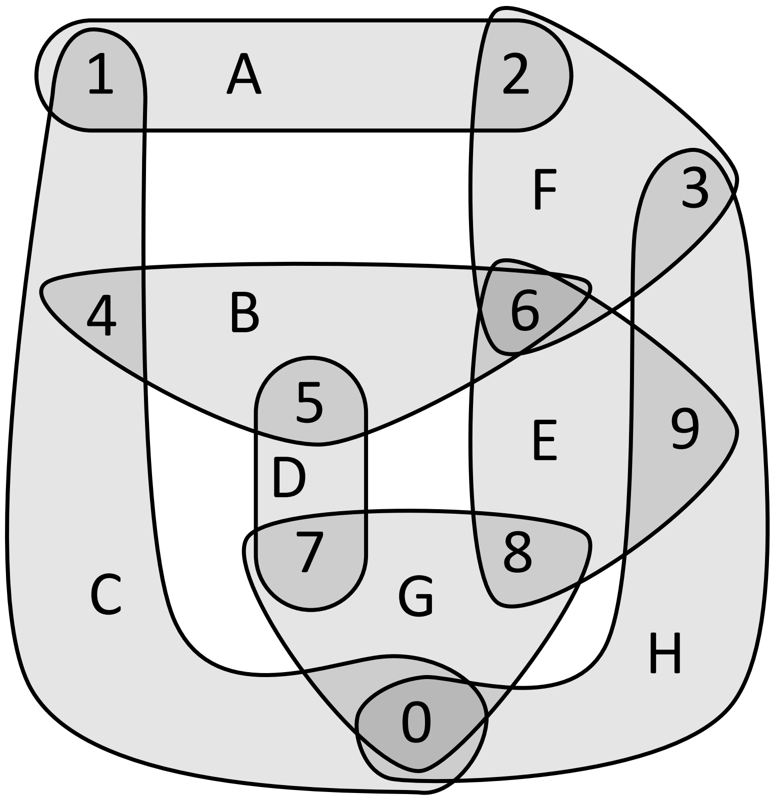
1. for each atom $A \in atoms(Q)$, there exists $p \in vertices(T)$ such that $var(A) \subseteq \chi(p)$;
2. for each variable $Y \in var(Q)$, the set $\{p \in vertices(T)$ s.t. $Y \in \chi(p)\}$ induces a (connected) subtree of $T$;
3. for each vertex $p \in vertices(T)$, $\chi(p) \subseteq var(\lambda(p))$;
4. for each vertex $p \in vertices(T)$, $var(\lambda(p)) \cap \chi(T_p) \subseteq \chi(p)$.

A hypertree decomposition $\langle T, \chi, \lambda \rangle$ of $Q$ is a *complete decomposition* of $Q$ if, for each atom $A \in atoms(Q)$, there exists $p \in vertices(T)$ such that $var(A) \subseteq \chi(p)$ and $A \in \lambda(p)$.

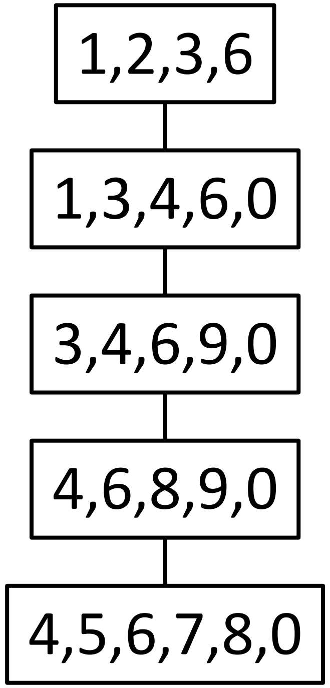
The *width* of the hypertree decomposition $\langle T, \chi, \lambda \rangle$ is $max_{p \in vertices(T)} |\lambda(p)|$. The *hypertree width* $hw(Q)$ of $Q$ is the minimum width over all its hypertree decompositions.

# Hypertree decomposition: full example

Hypergraph




Tree decomposition



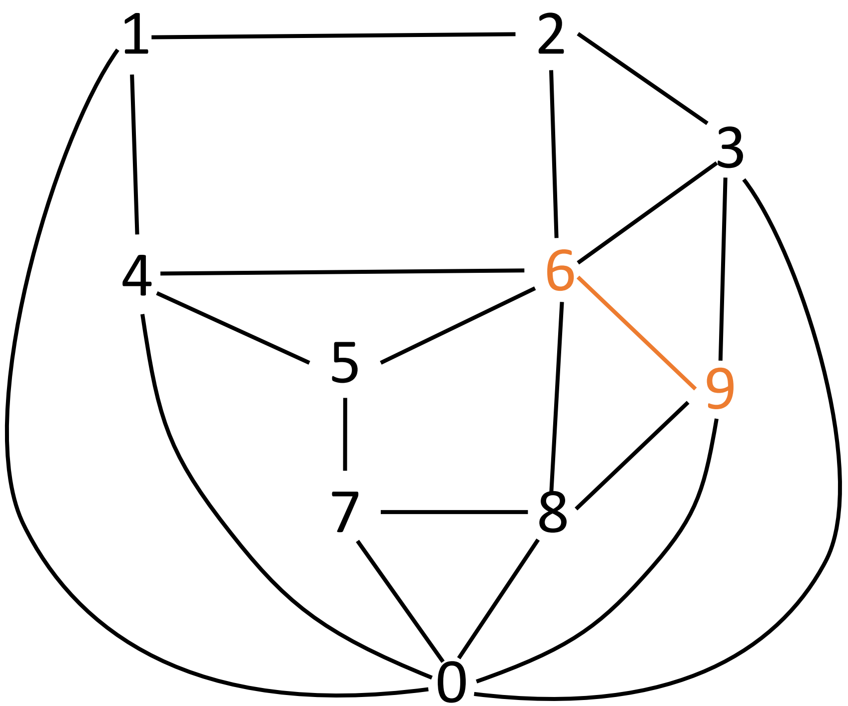

How to check that this is a valid tree decomposition?

?

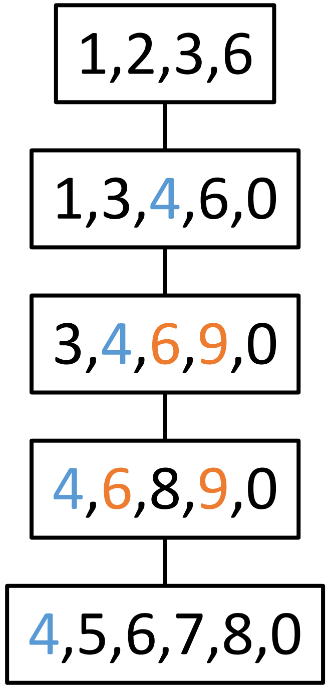
Example adopted from: Markus Krötzsch. "Database theory: Lecture 6: Tree-like Conjunctive Queries." 2016. https://iccl.inf.tu-dresden.de/web/Database_Theory_(SS2016)/en

# Hypertree decomposition: full example

Clique graph of Hypergraph
(also primal or Gaifman graph)




Tree decomposition

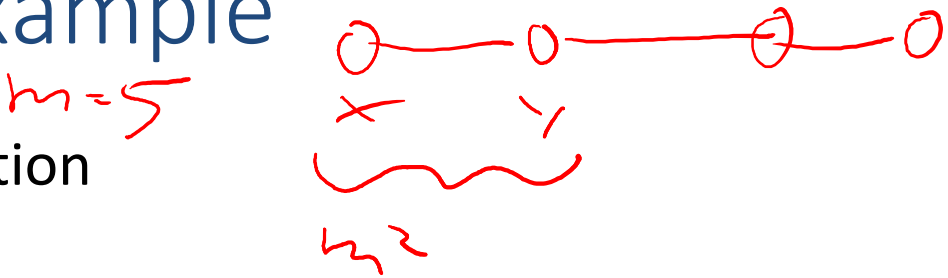



TREE DECOMPOSITION

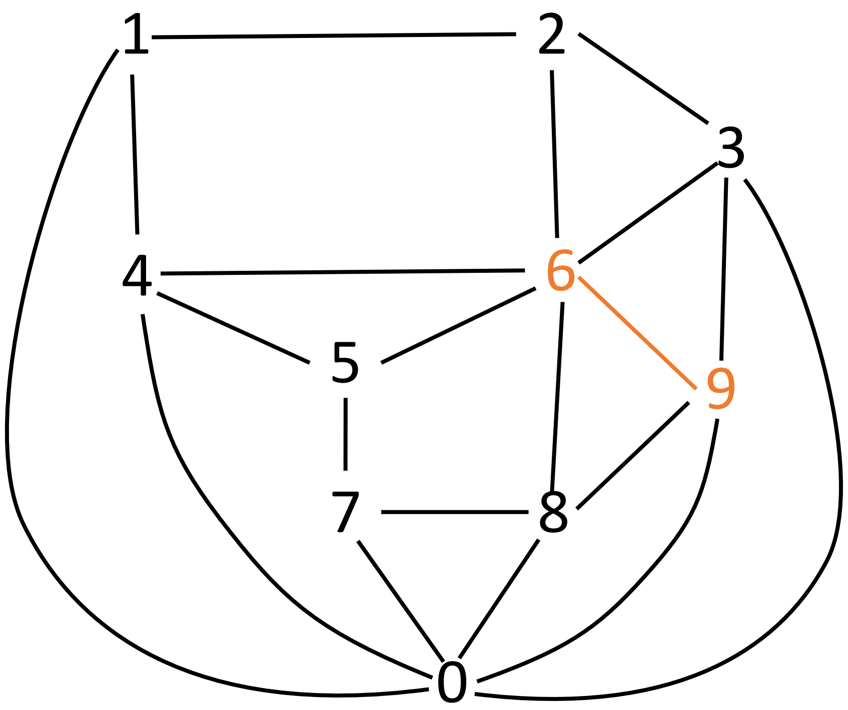
1. Edge coverage: For every edge e of G, there is a vertex in T that contains both ends of e
2. Coherence

What is its width ?

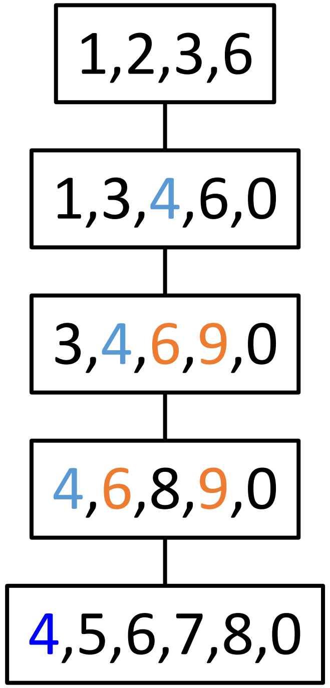
# Hypertree decomposition: full example




Clique graph of Hypergraph
(also primal or Gaifman graph)




Tree decomposition



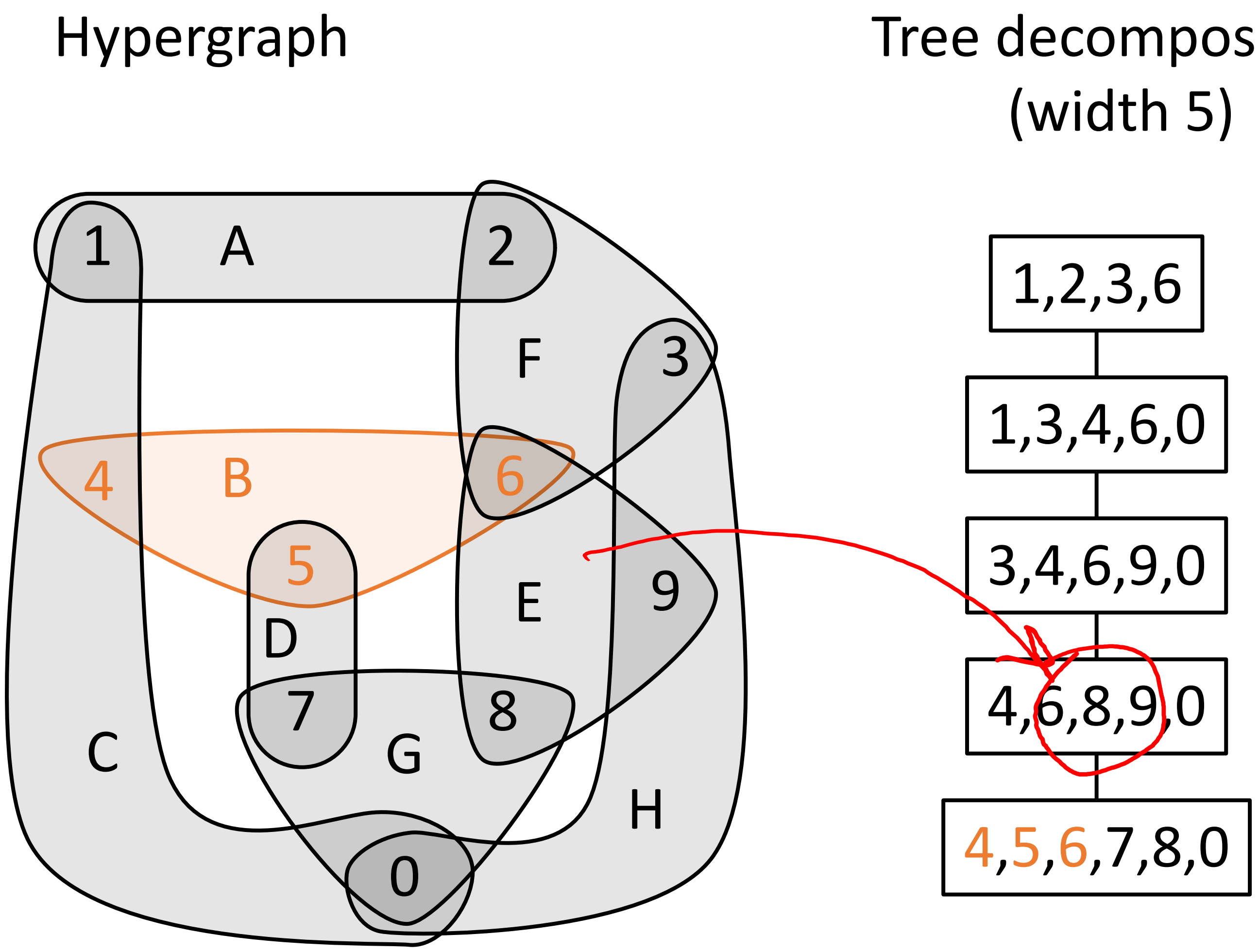

TREE DECOMPOSITION

1. Edge coverage: For every edge e of G, there is a vertex in T that contains both ends of e
2. Coherence

guarantees evaluation in $O(m^6)$
where m is the domain size or $O(n^5)$
where n is size of largest relation

tree width = 5:
= size of largest supernode - 1

# Hypertree decomposition: full example



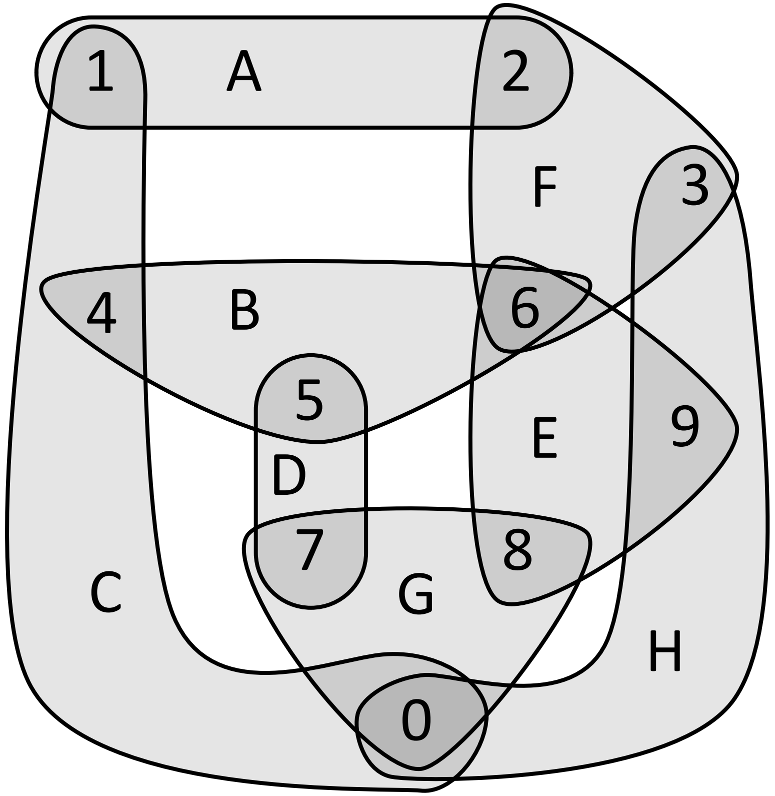

**TREE DECOMPOSITION (ALTERNATIVE)**

1. Hyperedge coverage: For every hyperedge h of H, there is a vertex in T that contains all its variables
2. Coherence

identical definition, because:
- hyperedge = clique in clique graph
- each clique needs to be contained in one supernode of the TD

# Hypertree decomposition: full example

Hypergraph




Tree decomposition
(width 5)

- 1,2,3,6
- 1,3,4,6,0
- 3,4,6,9,0
- 4,6,8,9,0
- 4,5,6,7,8,0

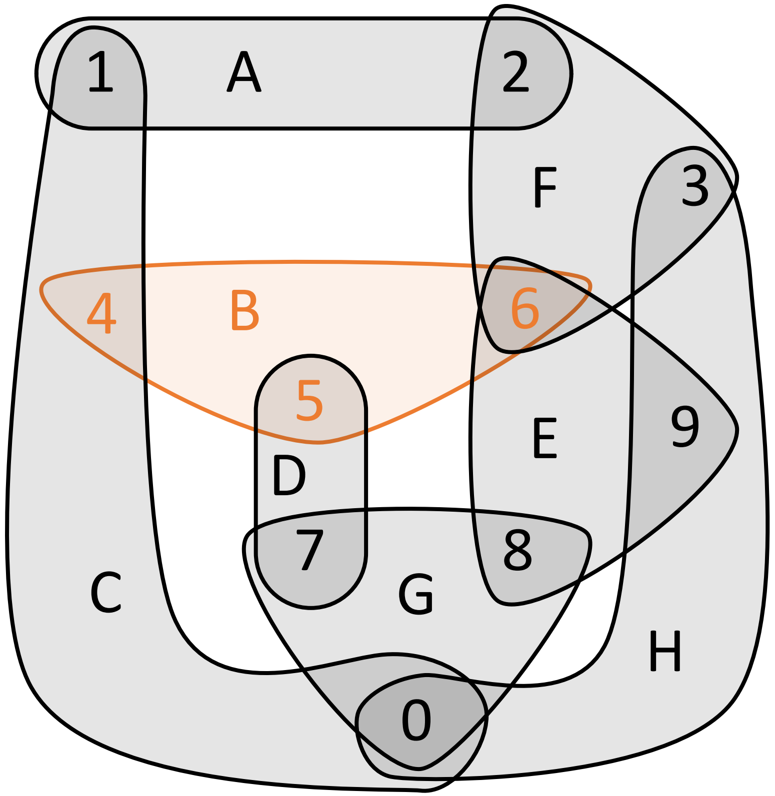
Generalized hypertree decomp.
(width 2)

- A{1,2}, F{2,3,6}
- C{1,4,0}, F{2,3,~~6~~}
- B{4,~~5~~,6}, H{3,9,0}
- C{~~1~~,4,0}, E{6,8,9}
- B{4,5,6}, G{7,8,0}

Why is this a valid "general. hypertree decomposition" ?

# Hypertree decomposition: full example

Hypergraph




Tree decomposition (width 5)

Generalized hypertree decomp. (width 2)

GENERALIZED HT DECOMP.

1. Hyperedge coverage: For every hyperedge h of H, there is a vertex in T that contains all its variables
2. Coherence

A{1,2}, F{2,3,6}

C{1,4,0}, F{2,3,~~6~~}

B{4,~~5~~,6}, H{3,9,0}

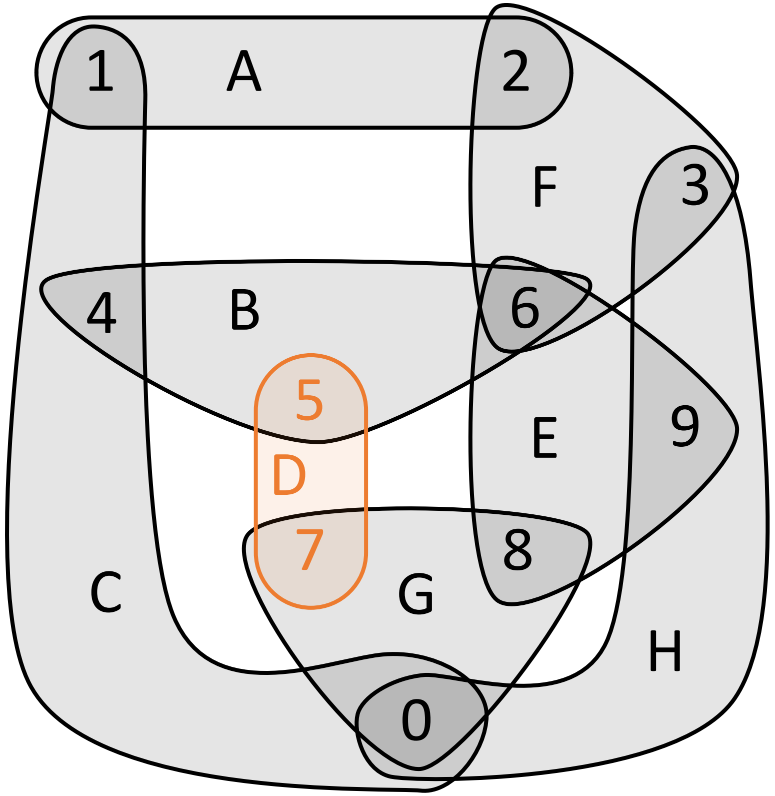
C{~~1~~,4,0}, E{6,8,9}

B{4,5,6}, G{7,8,0}

Basically identical to tree decomposition.
Just the width measure is different!

# Hypertree decomposition: full example

Hypergraph




Tree decomposition
(width 5)

**GENERALIZED HT DECOMP.**

1. Hyperedge coverage: For every hyperedge h of H, there is a vertex in T that contains all its variables
2. Coherence

Generalized hypertree decomp.
(width 2)

A{1,2}, F{2,3,6}

C{1,4,0}, F{2,3,~~6~~}

B{4,~~5~~,6}, H{3,9,0}

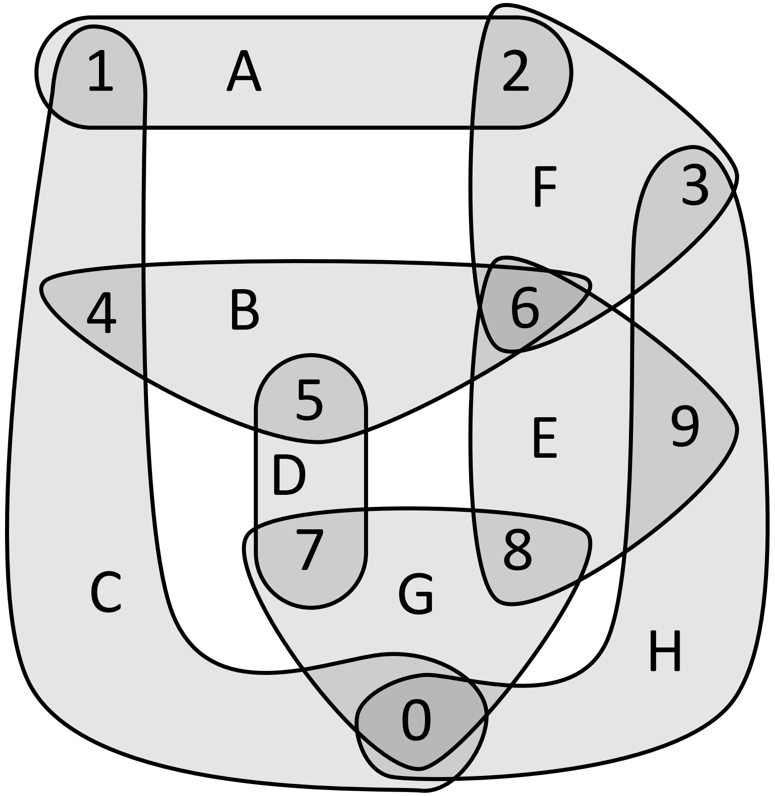
C{~~1~~,4,0}, E{6,8,9}

B{4,5,6}, G{7,8,0}

Basically identical to tree decomposition.
Just the width measure is different!

# Hypertree decomposition: full example

Hypergraph




**GENERALIZED HT DECOMP.**

1. Hyperedge coverage: For every hyperedge h of H, there is a vertex in T that contains all its variables
2. Coherence

Generalized hypertree decomp. (width 2)

A{1,2}, F{2,3,6}

C{1,4,0}, F{2,3,~~6~~}

B{4,~~5~~,6}, H{3,9,0}

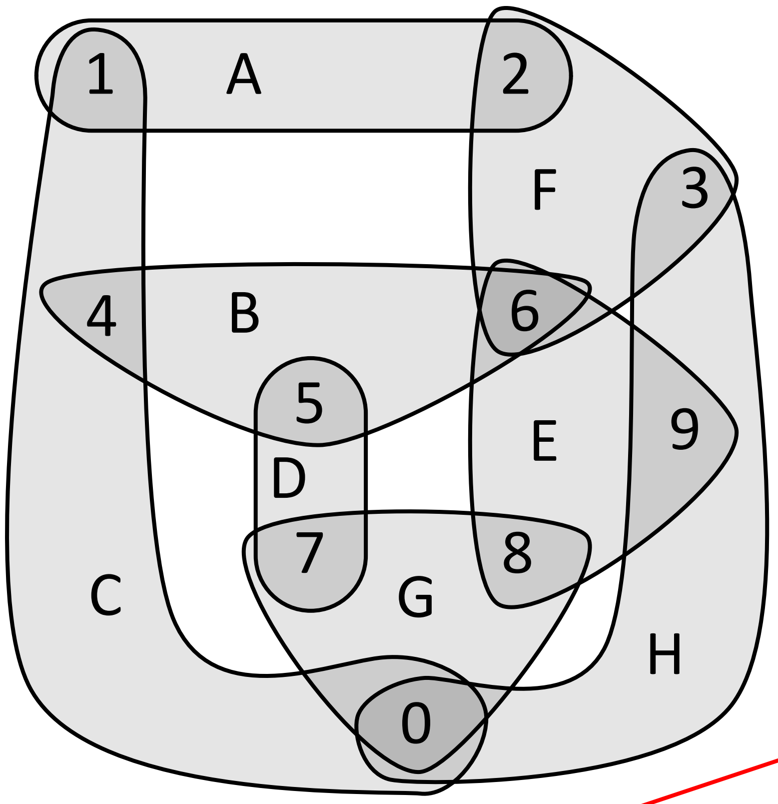
C{~~1~~,4,0}, E{6,8,9}

B{4,5,6}, G{7,8,0}

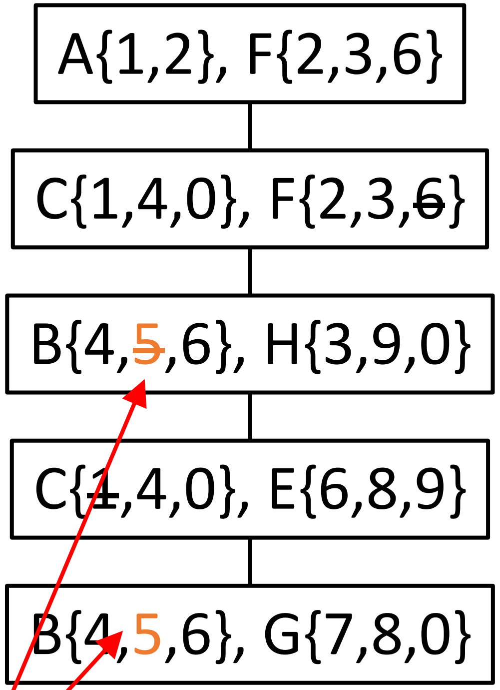
Is this a valid "hypertree decomposition" ?

# Hypertree decomposition: full example

Hypergraph




Generalized hypertree decomp. (width 2)




HT DECOMP.

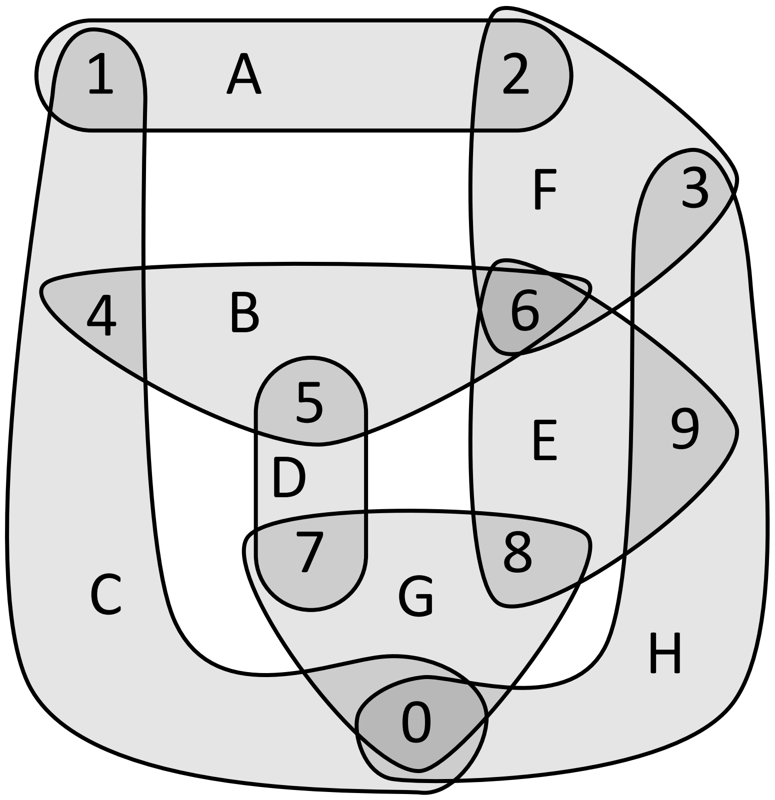
1. Hyperedge coverage: For every hyperedge h of H, there is a vertex in T that contains all its variables
2. Coherence
3. Descendant condition: Variables projected away from a hyperedge can not reappear in the subtree below

A condition to limit the search space of valid HD decompositions

5 got projected away, but reappears below

# Hypertree decomposition: full example

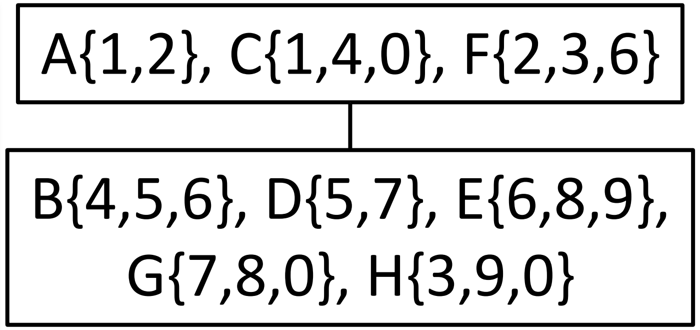
Hypergraph




HT DECOMP.

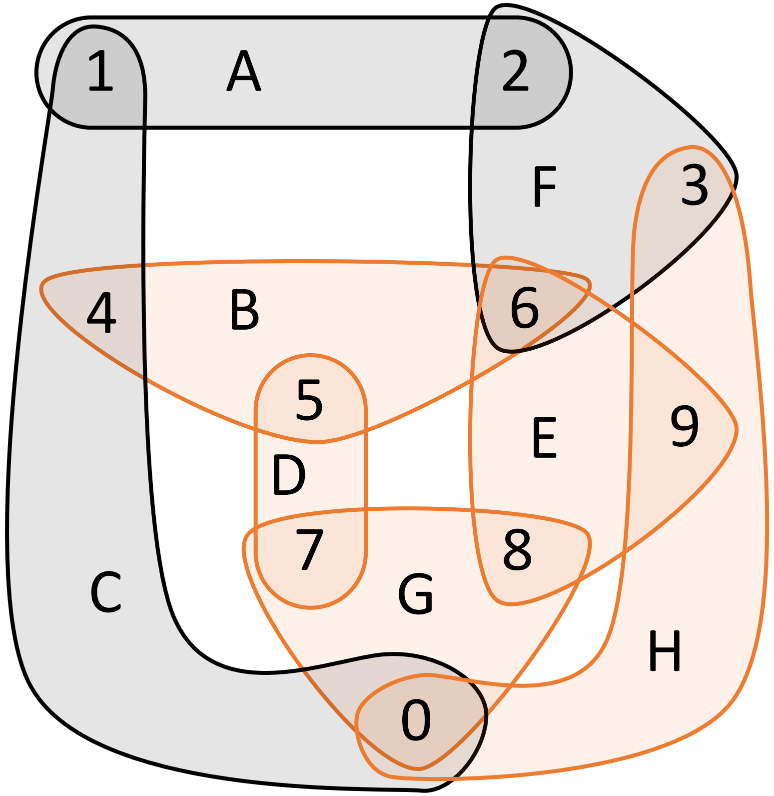
1. Hyperedge coverage: For every hyperedge h of H, there is a vertex in T that contains all its variables
2. Coherence
3. Descendant condition: Variables projected away from a hyperedge can not reappear in the subtree below

Hypertree decomposition

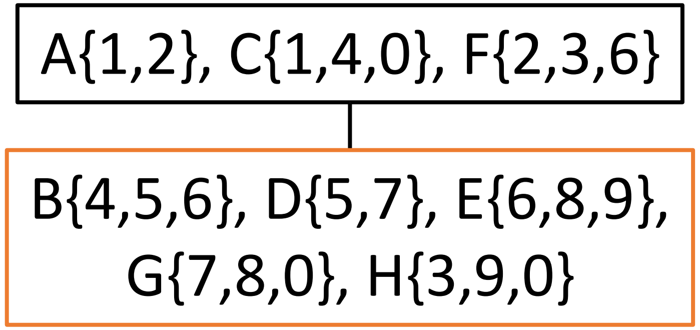


# Hypertree decomposition: full example

Hypergraph



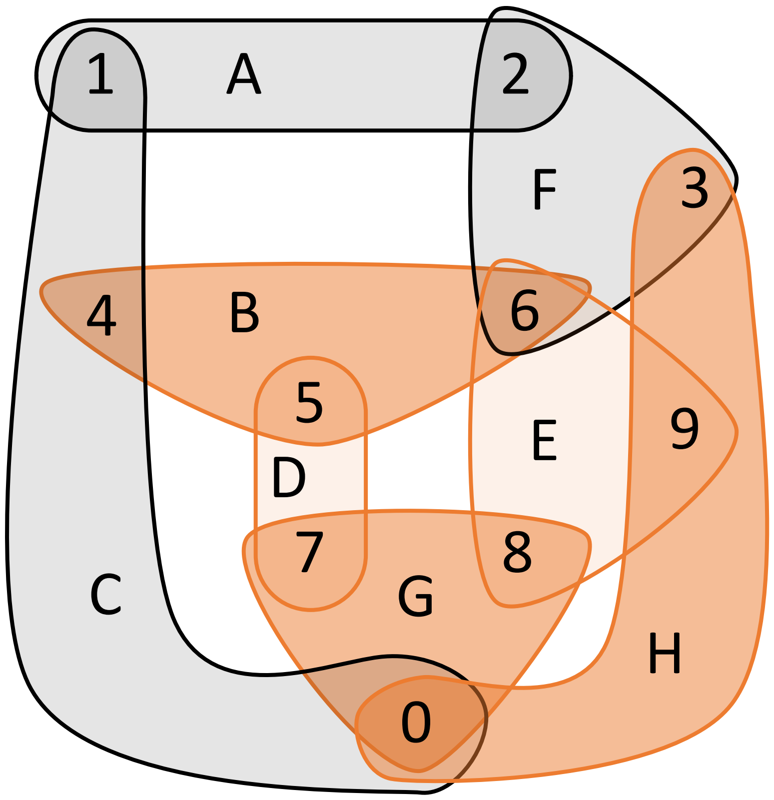

Hypertree decomposition (width ???)



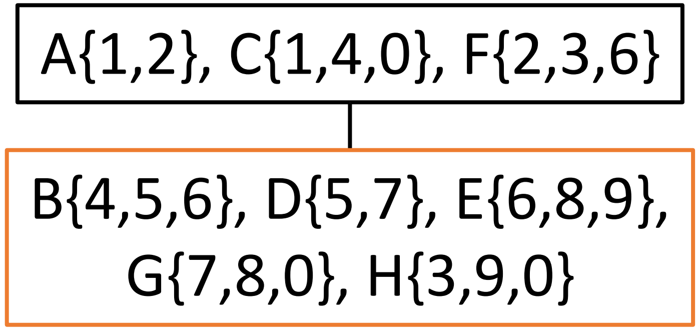

What should be the "width" of this HTD, i.e. what is the complexity of materializing this last supernode ?

# Hypertree decomposition: full example

Hypergraph




Hypertree decomposition
(width ???)




B(4,5,6)⋈G(7,8,0)⋈H(3,9,0)

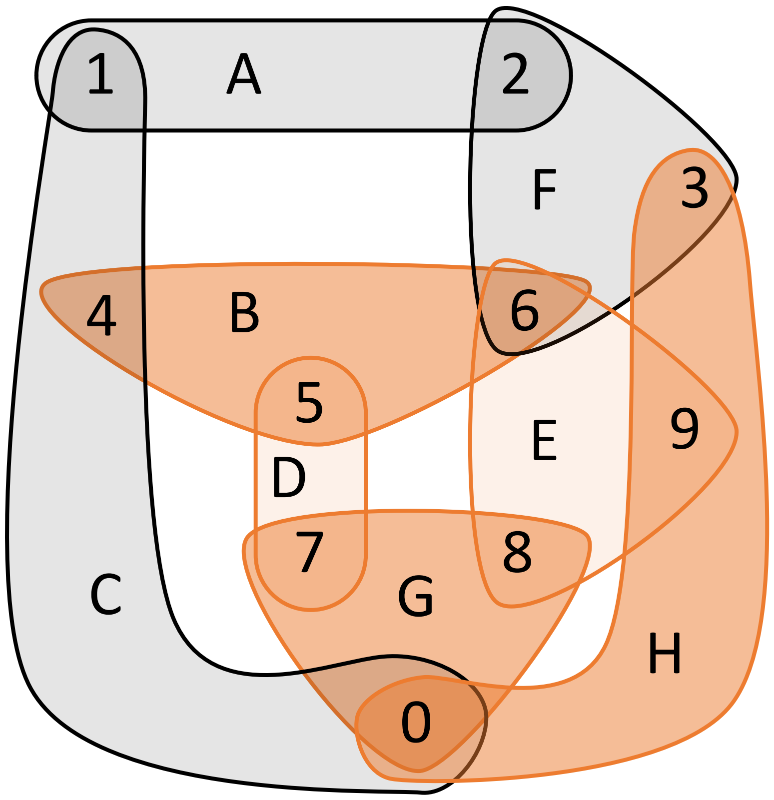
Notice that 3 relations alone "cover" all the variables.
The join can only be a subset of those tuples.

([(B(4,5,6) ⋈ G(7,8,0)) ⋈ H(3,9,0)] ← O(n³)
⋉D(5,7)) ⋉E(6,8,9)

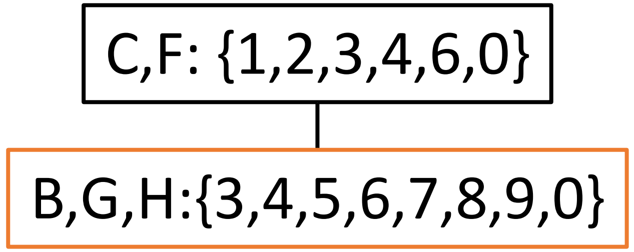
n... maximal size of relations

# Hypertree decomposition: full example

Hypergraph




Hypertree decomposition
(width 3)




B⋈G⋈H

With of HTD = maximal width of any super node.
With of supernode = minimal number of relations to cover all variables. Here covered by B⋈G⋈H

Results in a modified database and modified acyclic query. Then perform Yannakakis: $O(n^3)$

# Hypertree Decompositions: A Survey

Georg Gottlob[1], Nicola Leone[2], and Francesco Scarcello[3]

descendent condition

generalized. For instance, let us define the concept of *generalized hypertree decomposition* by just dropping condition 4 from the definition of hypertree decomposition (Def. 11). Correspondingly, we can introduce the concept of *generalized hypertree width* $ghw(\mathcal{H})$ of a hypergraph $\mathcal{H}$. We know that all classes of Boolean queries having bounded *ghw* can be answered in polynomial time. But we currently do not know whether these classes of queries are polynomially recognizable. This recognition problem is related to the mysterious *hypergraph*

Source: Gottlob, Leone, Scarcello. "Hypertree decompositions: a survey." MFCS 2001. https://dl.acm.org/doi/10.5555/645730.668191

# Hypertree width and related hypergraph invariants

Isolde Adler[a], Georg Gottlob[b], Martin Grohe[c]

European Journal of Combinatorics 28 (2007) 2167–2181

$$\mathrm{ghw}(H) \leq \mathrm{hw}(H) \leq \mathrm{tw}(H) + 1.$$

$$\mathrm{hw}(H) \leq 3 \cdot \mathrm{ghw}(H) + 1$$

# Generalized Hypertree Decompositions: NP-Hardness and Tractable Variants

Georg Gottlob
University of Oxford
Computing Laboratory
georg.gottlob@
comlab.ox.ac.uk

Zoltán Miklós
University of Oxford and
Technische Universität Wien
zoltan.miklos@
comlab.ox.ac.uk

Thomas Schwentick
Universität Dortmund
Lehrstuhl Informatik I
thomas.schwentick@
udo.edu

## ABSTRACT

The generalized hypertree width $GHW(H)$ of a hypergraph $H$ is a measure of its cyclicity. Classes of conjunctive queries or constraint satisfaction problems whose associated hypergraphs have bounded $GHW$ are known to be solvable in polynomial time. However, it has been an open problem for several years if for a fixed constant $k$ and input hypergraph $H$ it can be determined in polynomial time whether $GHW(H) \leq k$. Here, this problem is settled by proving that even for $k = 3$ the problem is already NP-hard. On