Updated 4/6/2024

Topic 3: Efficient query evaluation
Unit 2: Cyclic query evaluation
Lecture 22

Wolfgang Gatterbauer

CS7240 Principles of scalable data management (sp24)

https://northeastern-datalab.github.io/cs7240/sp24/

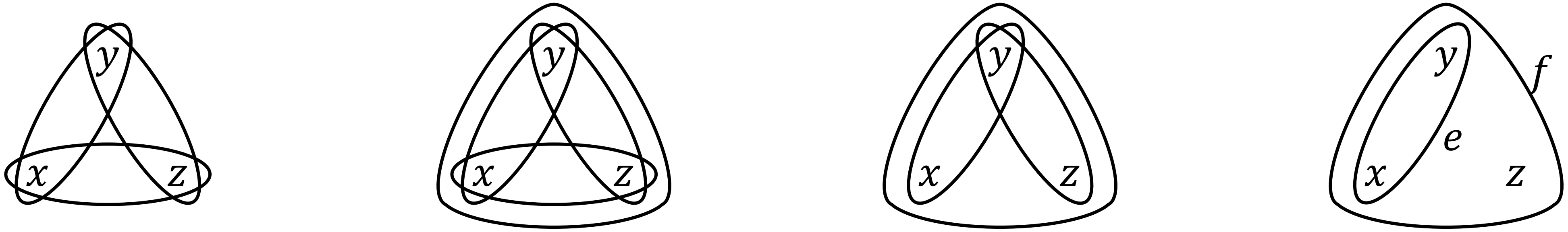
4/5/2024

Pre-class conversations

- Last class summary
- Project: I hope Feedback was usefull
  - Approach me with questions, or schedule office hours
  - Latex template, missing line numbers on first page
- Scribes: Feedback yet to come

- Today:
  - Why cycles change everything

Acyclic graphs: $\alpha \supset \beta \supset \gamma \supset$ Berge (α-acyclic graphs are $\supset$ of others)



	Triangle	**Beta triangle**	**Gamma triangle**	**Berge cycle**
alpha	cyclic	acyclic	acyclic	acyclic
beta	cyclic	cyclic	acyclic	acyclic
gamma	cyclic	cyclic	cyclic	acyclic
Berge	cyclic	cyclic	cyclic	cyclic

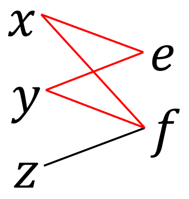
Define a <u>hypergraph</u> as a set of nonempty sets.

$\mathcal{H}_1$ is a <u>subhypergraph</u> (subset) of $\mathcal{H}_2$ if $\mathcal{H}_1 \subseteq \mathcal{H}_2$.

A hypergraph $\mathcal{H}$ is <u>beta acyclic</u> if all its subhypergraphs are alpha acyclic.

A hypergraph $\mathcal{H}$ is <u>gamma acyclic</u> if if it is beta acyclic and we cannot find x, y, z s.t. $\{\{x, y\}, \{y, z\}, \{x, y, z\}\} \subseteq \mathcal{H}[\{x, y, z\}]$, the <u>induced subhypergraph</u> on the set $\{x, y, z\}$.

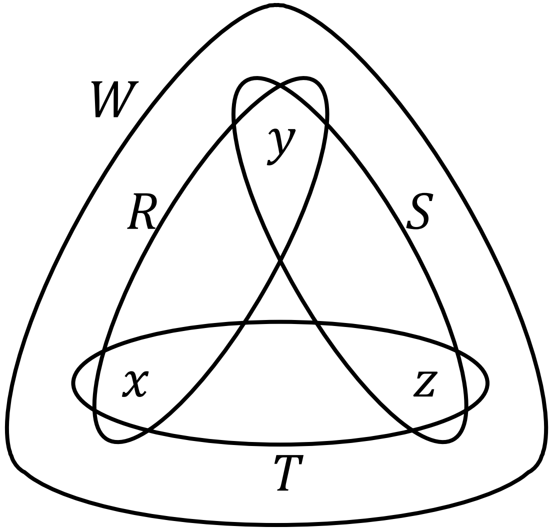
A hypergraph H is <u>Berge acyclic</u> if the incidence graph $\{\{x, e\} \mid e \in \mathcal{H} \text{ and } x \in e\}$ is acyclic.



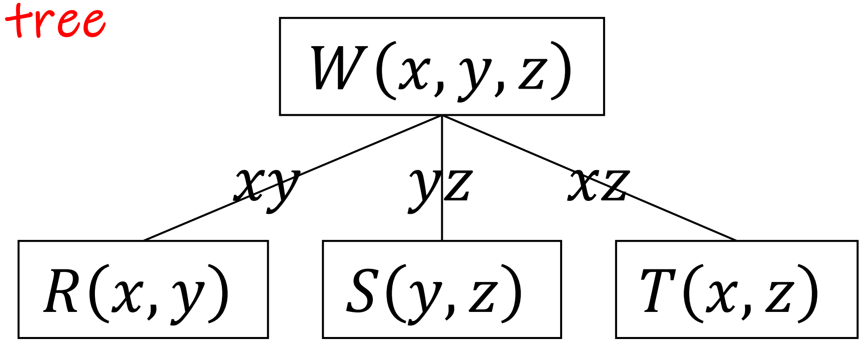
For definitions see: Brault-Baron. "Hypergraph Acyclicity Revisited". ACM Computing Surveys 2016. https://doi.org/10.1145/2983573

"beta-triangle" is alpha-acyclic, but not its dual

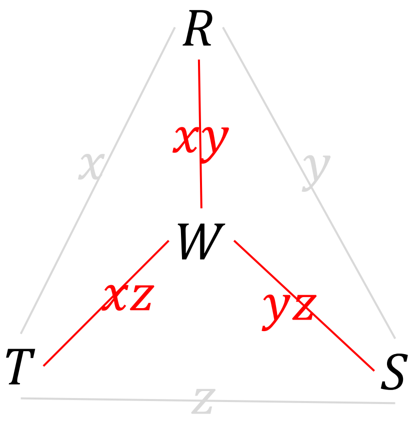
$\mathcal{H} = \{R(x,y), S(y,z), T(x,z), W(x,y,z)\}$



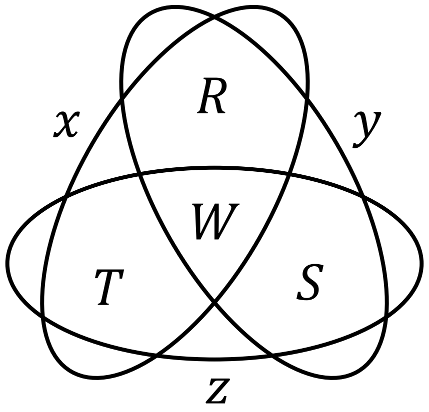
Join tree



Gaifman graph of $\mathcal{D}(\mathcal{H})$ (w/ attribute-connected spanning tree)

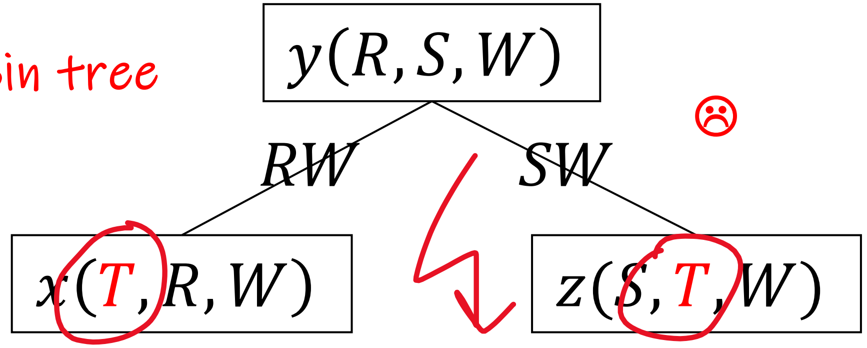


Dual $\mathcal{D}(\mathcal{H})$



$\mathcal{D}(\mathcal{H}) = \{x(T,R,W), y(R,S,W), z(S,T,W)\}$

No join tree



Outline: T3-2: Cyclic conjunctive queries

- T3-1: Acyclic conjunctive queries
- T3-2: Cyclic conjunctive queries
  - 2SAT (a detour)
  - Tree decompositions
  - Hypertrees decompositions
  - Duality in Linear programming (a not so quick primer)
  - AGM bound (maximal result size for full CQs) and Worst-case optimal joins for the triangle query
  - Worst-case optimal joins & the 4-cycle
  - Optimal joins & the 4-cycle

cycles make everything more complicated ☹

Why cyclic queries (other than social networks)

```
Likes(person, drink)
Frequents(person, bar)
Serves(bar, drink, cost)
```

2. Specify or choose a Query Supported grammar

104 Bars: Persons who frequent some bar that serves some drink they like.

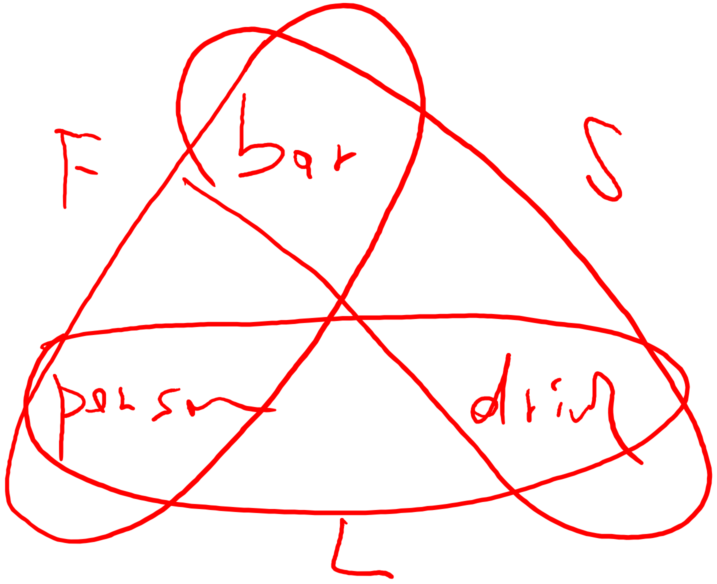
Source: http://demo.queryvis.com

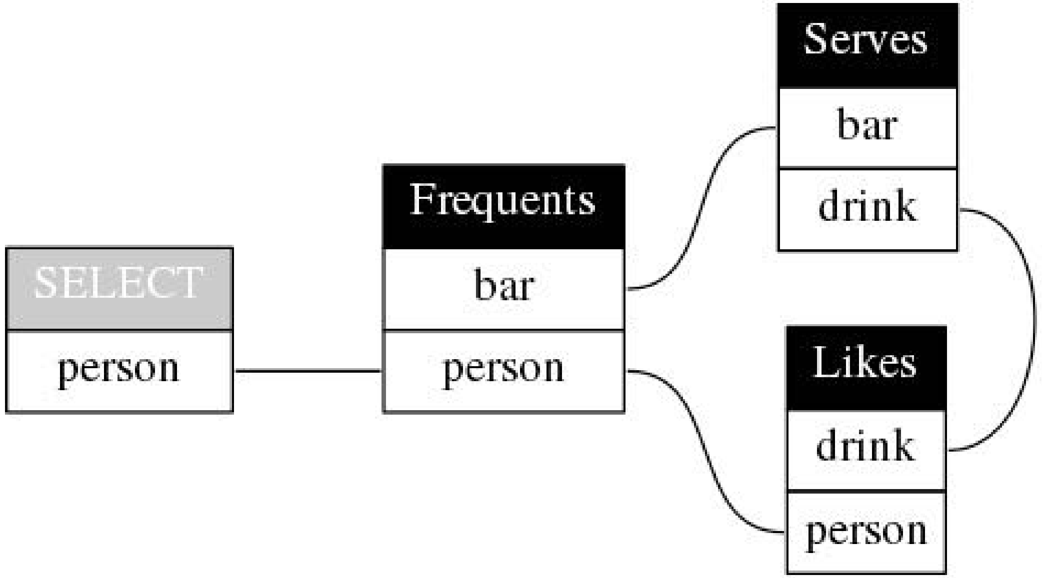
Why cyclic queries (other than social networks)

```
Likes(person, drink)
Frequents(person, bar)
Serves(bar, drink, cost)
```

2. Specify or choose a Query [Supported grammar]

104 Bars: Persons who frequent some bar that serves some drink they like.





Source: http://demo.queryvis.com

Why cyclic queries (other than social networks)

```
Likes(person, drink)
Frequents(person, bar)
Serves(bar, drink, cost)
```

2. Specify or choose a Query Supported grammar

104 Bars: Persons who frequent some bar that serves some drink they like.

```
SELECT  F1.person
FROM    Frequents F1
WHERE   exists
        (SELECT *
        FROM    Serves S2
        WHERE   S2.bar = F1.bar
        AND     exists
                (SELECT *
                FROM    Likes L3
                WHERE   L3.person = F1.person
                AND     S2.drink = L3.drink))
```

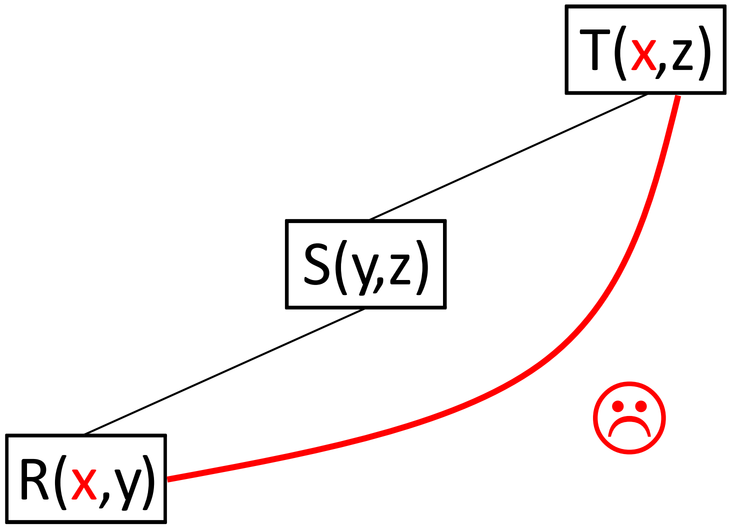
Source: http://demo.queryvis.com

Joins in databases: one-at-a-time

How can we efficiently process multi-way joins with cycles?

$Q(x,y,z) \text{ :- } R(x,y), S(y,z), T(x,z).$

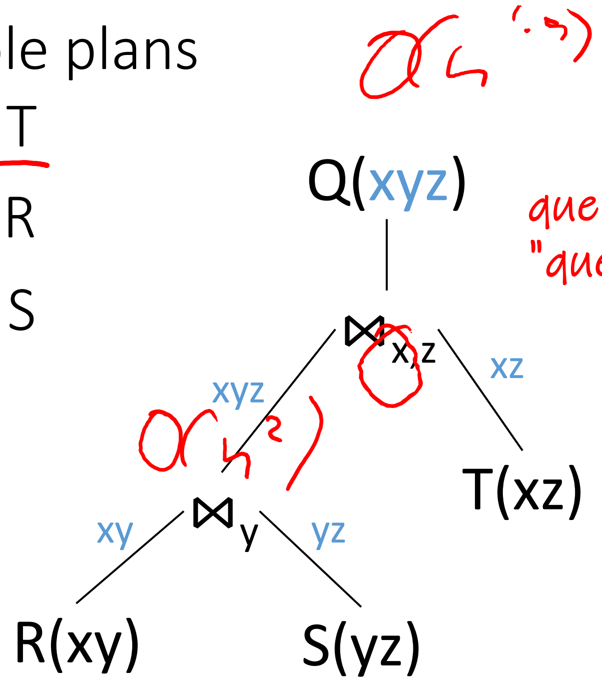
Recall:



There is no join tree! You can't fulfill the running intersection property...

Three possible plans

- $(R \bowtie S) \bowtie T$
- $(S \bowtie T) \bowtie R$
- $(T \bowtie R) \bowtie S$



query plan as "query tree"

☹

- there is no full semijoin reducer
- intermediate result size bigger than output

Can we do better for cyclic queries? ☺

Outline: T3-2: Cyclic conjunctive queries

- T3-1: Acyclic conjunctive queries
- T3-2: Cyclic conjunctive queries
  - 2SAT (a detour)
  - Tree decompositions
  - Hypertrees decompositions
  - Duality in Linear programming (a not so quick primer)
  - AGM bound (maximal result size for full CQs) and Worst-case optimal joins for the triangle query
  - Worst-case optimal joins & the 4-cycle
  - Optimal joins & the 4-cycle

2SAT

$\varphi = (x \vee y) \wedge (\neg y \vee z) \wedge (\neg x \vee \neg z) \wedge (z \vee y)$

- Instance: A 2-CNF formula φ
- Problem: To decide if φ is satisfiable

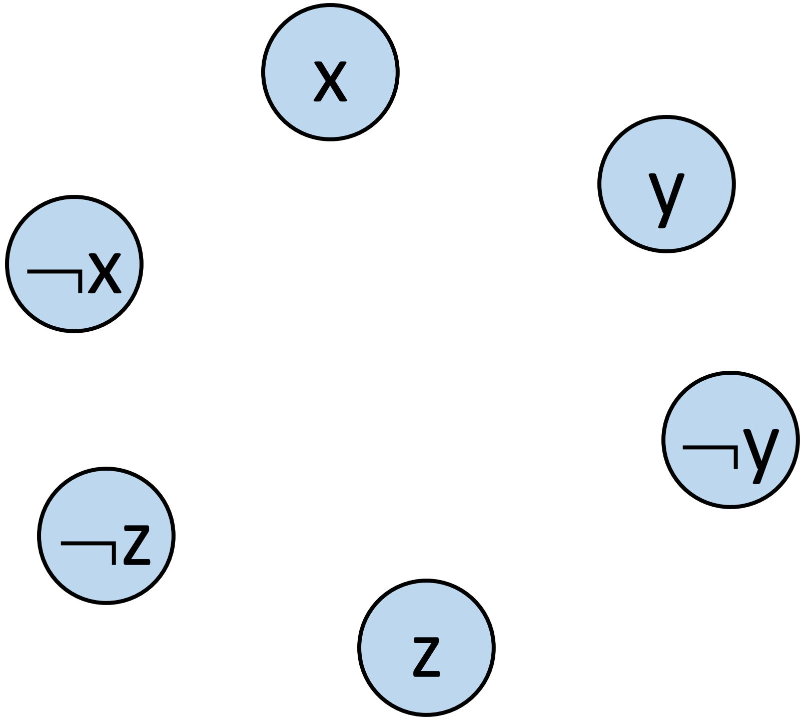
- Theorem: 2SAT is polynomial-time decidable.
  - Proof: We'll show how to solve this problem efficiently using path searches in graphs…

- Background: Given a graph G=(V,E) and two vertices $s,t \in V$, finding if there is a path from s to t in G is linear-time decidable. Use some search algorithm (DFS/BFS).

2SAT: Graph Construction

$\varphi = (x \vee y) \wedge (\neg y \vee z) \wedge (\neg x \vee \neg z) \wedge (z \vee y)$

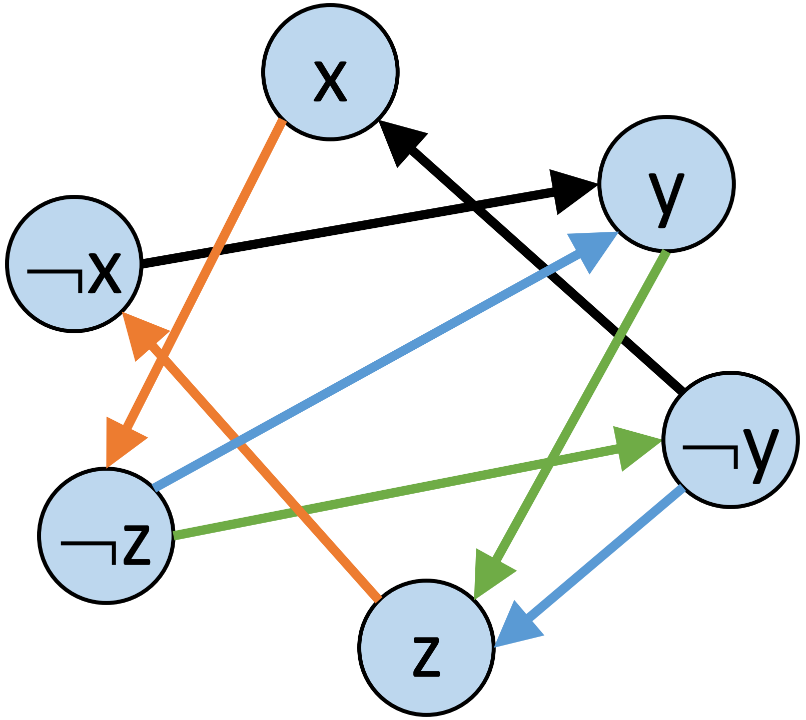
- Vertex for each variable and a negation of a variable



2SAT: Graph Construction

$\varphi = (x \lor y) \land (\neg y \lor z) \land (\neg x \lor \neg z) \land (z \lor y)$

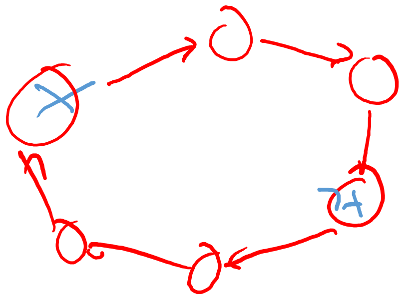
- Vertex for each variable and a negation of a variable
- Edge $(\neg x \rightarrow y)$ iff there exists a clause equivalent to $(x \lor y)$
  - Recall $(x \lor y)$ same as $(\neg x \Rightarrow y)$ and $(\neg y \Rightarrow x)$, thus also $(\neg y \rightarrow x)$

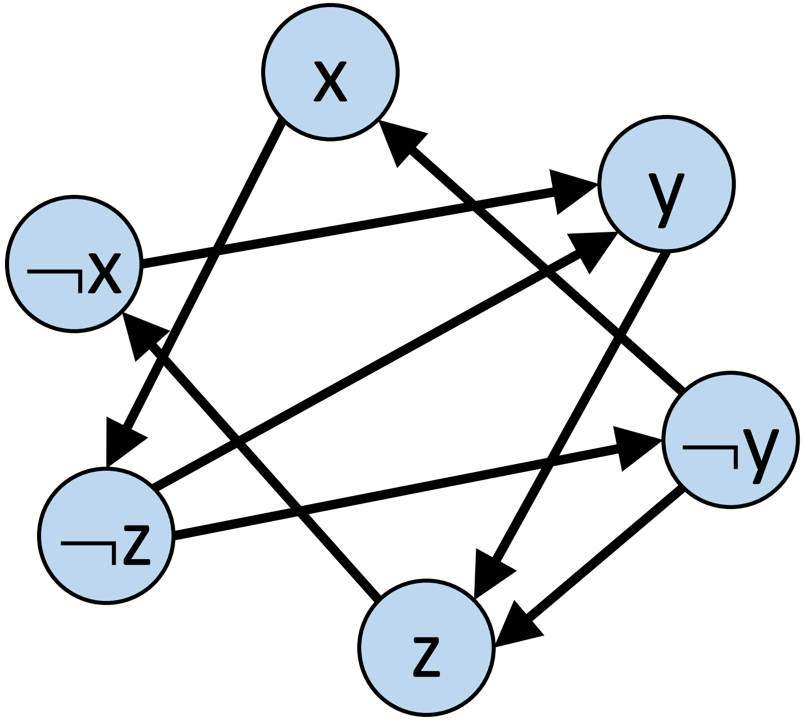


2SAT: Graph Construction

$\varphi = (x \lor y) \land (\neg y \lor z) \land (\neg x \lor \neg z) \land (z \lor y)$

- Vertex for each variable and a negation of a variable
- Edge $(\neg x \rightarrow y)$ iff there exists a clause equivalent to $(x \lor y)$
  - Recall $(x \lor y)$ same as $(\neg x \Rightarrow y)$ and $(\neg y \Rightarrow x)$, thus also $(\neg y \rightarrow x)$
- Claim: a 2-CNF formula φ is unsatisfiable iff there exists a variable x, such that:
  - there is a path from x to ¬x in the graph, and
  - there is a path from ¬x to x in the graph



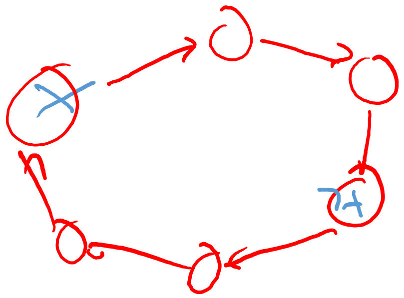


2SAT: Graph Construction

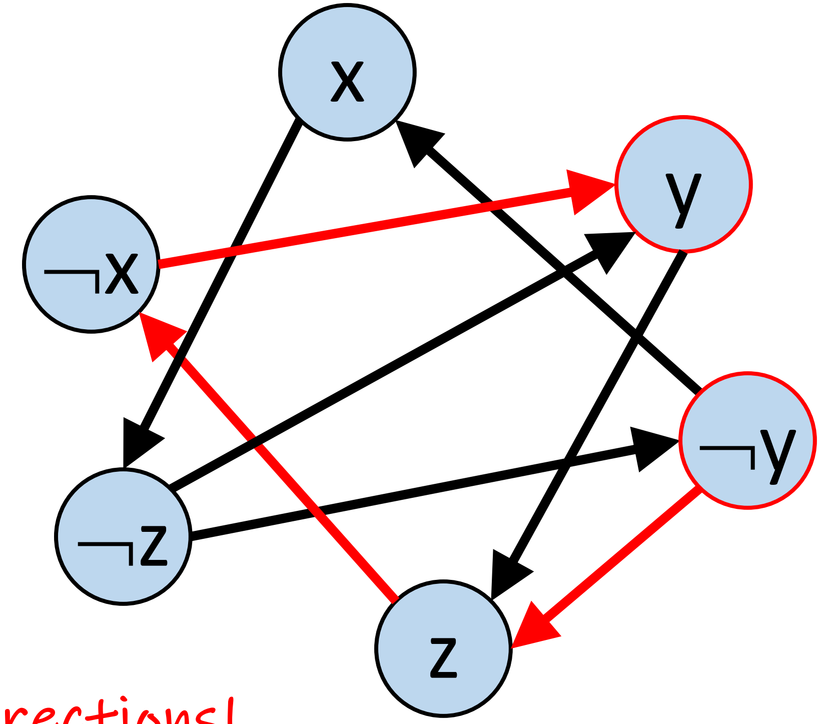
$\varphi = (x \lor y) \land (\neg y \lor z) \land (\neg x \lor \neg z) \land (z \lor y)$

- Vertex for each variable and a negation of a variable
- Edge $(\neg x \rightarrow y)$ iff there exists a clause equivalent to $(x \lor y)$
  - Recall $(x \lor y)$ same as $(\neg x \Rightarrow y)$ and $(\neg y \Rightarrow x)$, thus also $(\neg y \rightarrow x)$

- Claim: a 2-CNF formula φ is unsatisfiable iff there exists a variable x, such that:
  - there is a path from x to ¬x in the graph, and
  - there is a path from ¬x to x in the graph



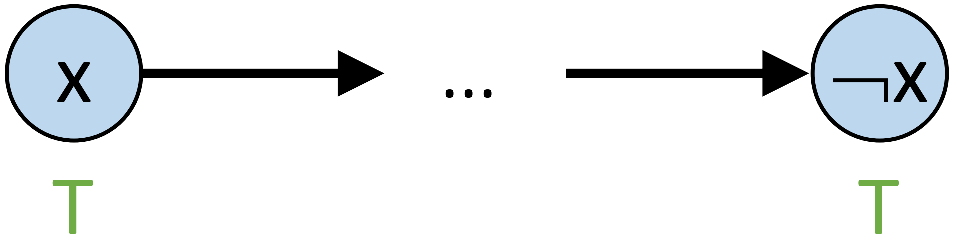
not enough,
needs both directions!



Correctness (1)

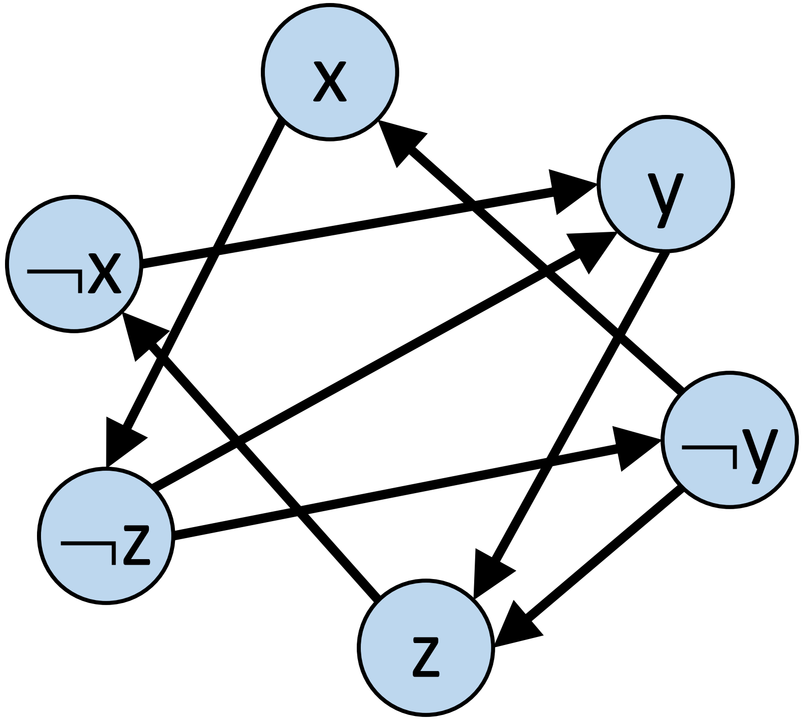
$\varphi = (x \vee y) \wedge (\neg y \vee z) \wedge (\neg x \vee \neg z) \wedge (z \vee y)$

- Suppose there are paths $x..\neg x$ and $\neg x..x$ for some variable x, but there's also a satisfying assignment ρ.
  - If $\rho(x)=T$:



  - Similarly for $\rho(x)=F$...

recall, needs to hold in both directions!



Correctness (2)

$\varphi = (x \lor y) \land (\neg y \lor z) \land (\neg x \lor \neg z) \land (z \lor y)$

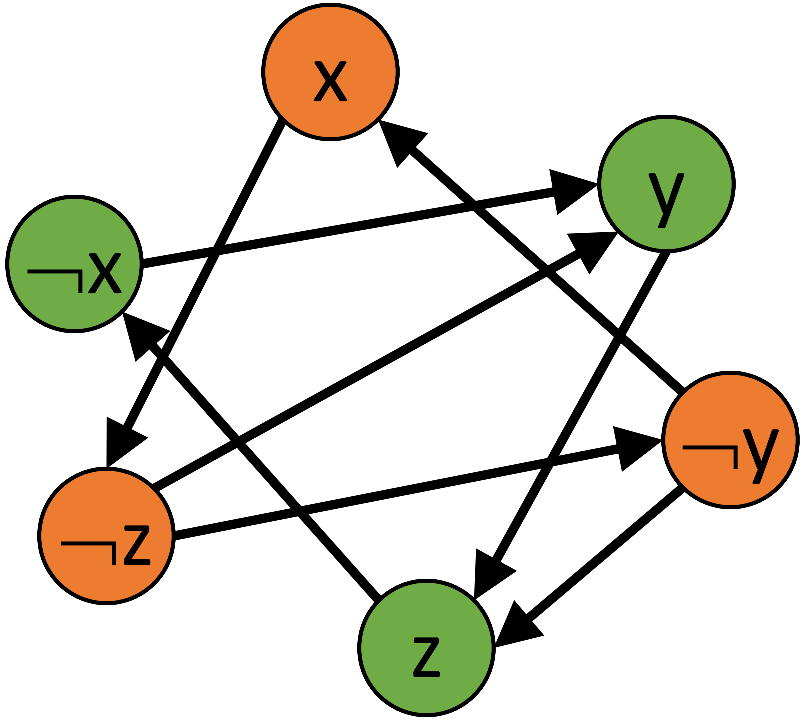
- Suppose there are no variables with such paths.
- Construct an assignment as follows:

1. pick an unassigned literal α, with no path from α to $\neg\alpha$, and assign it T

2. assign T to all reachable vertices

3. assign F to their negations

4. Repeat until all vertices are assigned



2SAT is in P

We get the following PTIME algorithm for 2SAT:

- For each variable x find if there is a path from x to ¬x and vice-versa.
- Reject if any of these tests succeeded.
- Accept otherwise.

$\Rightarrow$ 2SAT $\in$ P. ■

Outline: T3-2: Cyclic conjunctive queries

- T3-1: Acyclic conjunctive queries
- T3-2: Cyclic conjunctive queries
  - 2SAT (a detour)
  - Tree decompositions
  - Hypertrees decompositions
  - Duality in Linear programming (a not so quick primer)
  - AGM bound (maximal result size for full CQs) and Worst-case optimal joins for the triangle query
  - Worst-case optimal joins & the 4-cycle
  - Optimal joins & the 4-cycle

Join Processing: two approaches

1. Cardinality-based
   - binary joins, consider the sizes of input relations as to reduce the intermediate sizes
   - commercial DBMSs: series of pairwise joins, system R (Selinger), join size estimation
2. Structural approaches (next)
   - acylicity: Yannakakis, GYO algorithm, join tree
   - bounded "width": query width, hypertree width (hw), generalized hw (ghw). All go back to notion of treewidth (work by Robertson & Seymour on graph minors)

AGM: fractional hw (fhw):

- consider both statistics on relations and query structure

Tree decomposition

In graph theory, a **tree decomposition** is a mapping of a graph into a tree that can be used to define the treewidth of the graph and speed up solving certain computational problems on the graph.

Tree decompositions are also called **junction trees**, **clique trees**, or **join trees**. They play an important role in problems like probabilistic inference, constraint satisfaction, query optimization, [*citation needed*] and matrix decomposition.

The concept of tree decomposition was originally introduced by Rudolf Halin (1976). Later it was rediscovered by Neil Robertson and Paul Seymour (1984) and has since been studied by many other authors.[1]

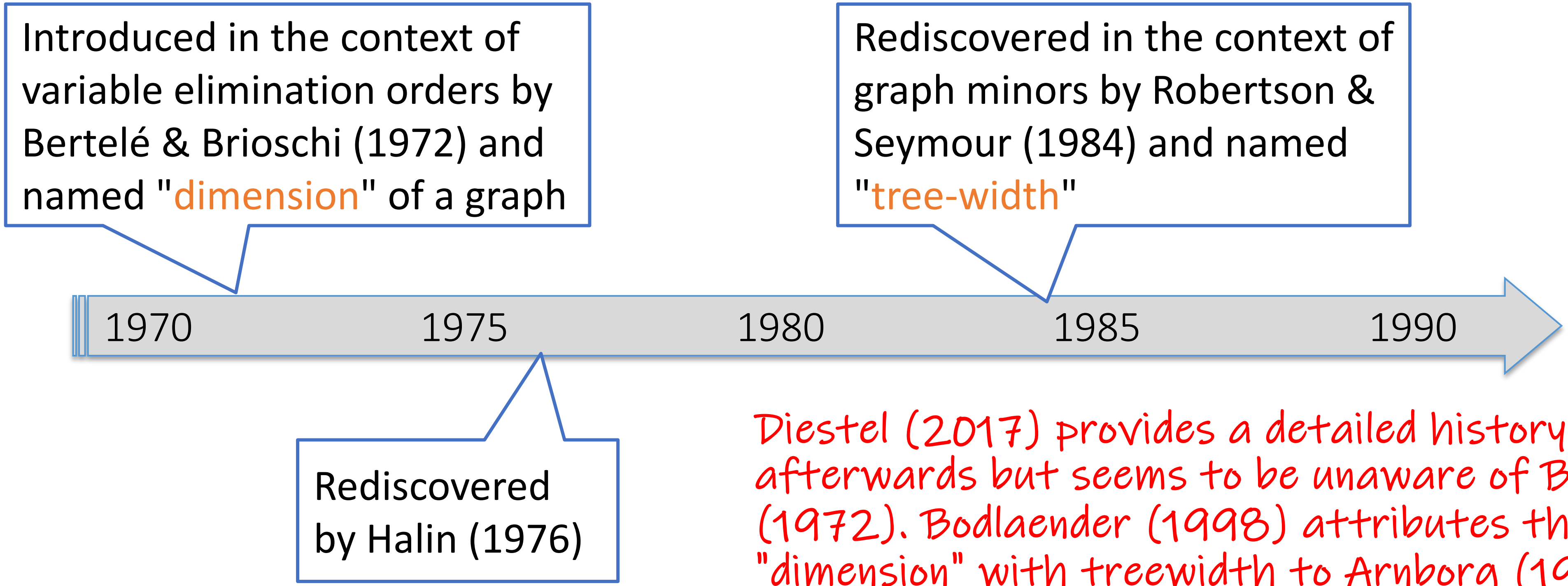
Dynamic programming [edit]

At the beginning of the 1970s, it was observed that a large class of combinatorial optimization problems defined on graphs could be efficiently solved by non-serial dynamic programming as long as the graph had a bounded *dimension*,[5] a parameter related to treewidth. Later, several authors independently observed, at the end of the 1980s,[6] that many algorithmic problems that are NP-complete for arbitrary graphs may be solved efficiently by dynamic programming for graphs of bounded treewidth, using the tree-decompositions of these graphs.

- Robertson, Neil; Seymour, Paul D. (1984), "Graph minors III: Planar tree-width", *Journal of Combinatorial Theory*, Series B, **36** (1): 49–64, doi:10.1016/0095-8956(84)90013-3.

Very incomplete history of treewidth

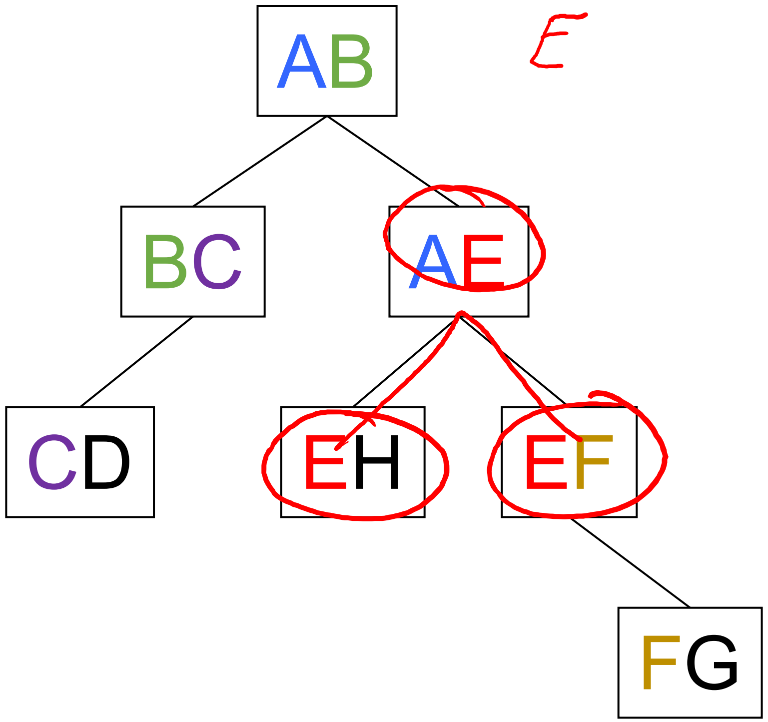
The treewidth of a graph is an important graph complexity parameter that determines the runtime of practical algorithms. Intuitively measures how close a graph is to being a tree.



Diestel (2017) provides a detailed history of what happened afterwards but seems to be unaware of Bertelé & Brioschi (1972). Bodlaender (1998) attributes the connection of "dimension" with treewidth to Arnborg (1985) who actually never uses the word "treewidth" nor references R&S (1984)...

Bertelè, Brioschi. Nonserial Dynamic Programming, 1972 (definition 2.7.8). https://dl.acm.org/doi/10.5555/578817 , Halin. S-functions for graphs, Journal of Geometry, 1976. https://doi.org/10.1007%2FBF01917434 , Robertson, Seymour. Graph minors III: Planar tree-width, Journal of Combinatorial Theory, 1984 https://doi.org/10.1016%2F0095-8956%2884%2990013-3 , Diestel. Graph theory, 5th ed, 2017 (section 12). https://doi.org/10.1007/978-3-662-53622-3 , Bodlaender. A partial k-arboretum of graphs with bounded treewidth (tutorial), Theoretical Computer Science, 1998. https://doi.org/10.1016/S0304-3975(97)00228-4 , Arnborg. Efficient algorithms for combinatorial problems on graphs with bounded decomposability -- a survey, BIT, 1985. https://dl.acm.org/doi/abs/10.5555/3765.3773

Definition of an attribute-connected tree



DEFINITION: A tree is attribute-connected if the subtree induced by each attribute is connected

Same as the running intersection property from join trees (also known as junction tree)

Also called "coherence"

Tree decomposition

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 1: a tree

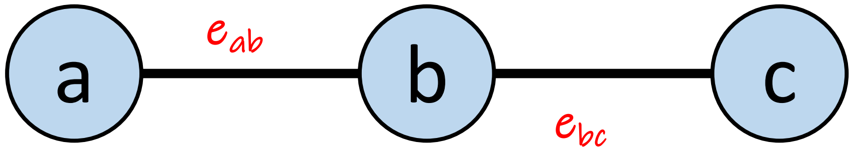
A tree decomposition of graph $\mathrm{G(N, E)}$ is a tree $\mathrm{T(V, F)}$ and a subset $\mathrm{N_v \subseteq N}$ assigned to each vertex (or "supernode") $\mathrm{v \in V}$ s.t.:

(1) Node coverage: Every vertex of 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 1: a tree

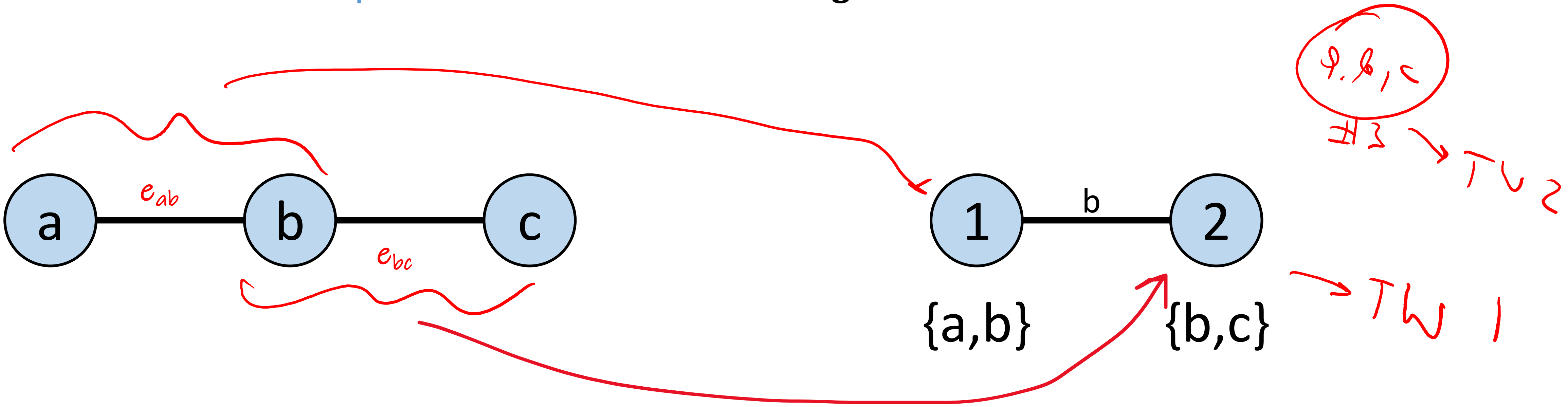
A tree decomposition of graph $G(N, E)$ is a tree $T(V, F)$ and a subset $N_v \subseteq N$ assigned to each vertex (or "supernode") $v \in V$ s.t.:

(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



That's why treewidth defined as max cardinality - 1

Tree decomposition example 2

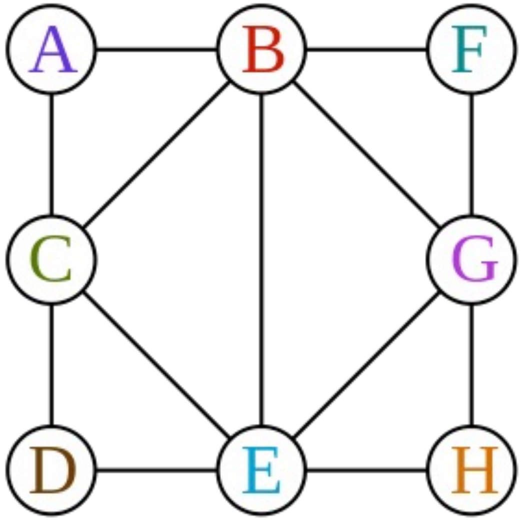
A tree decomposition of graph $G(N, E)$ is a tree $T(V, F)$ and a subset $N_v \subseteq N$ assigned to each vertex (or "supernode") $v \in V$ s.t.:

(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 2

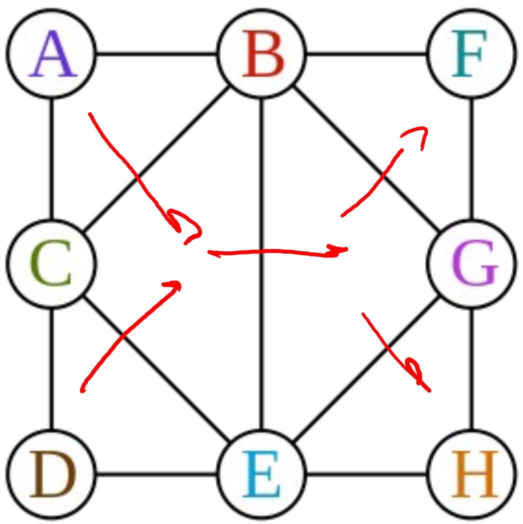
A tree decomposition of graph $G(N, E)$ is a tree $T(V, F)$ and a subset $N_v \subseteq N$ assigned to each vertex (or "supernode") $v \in V$ s.t.:

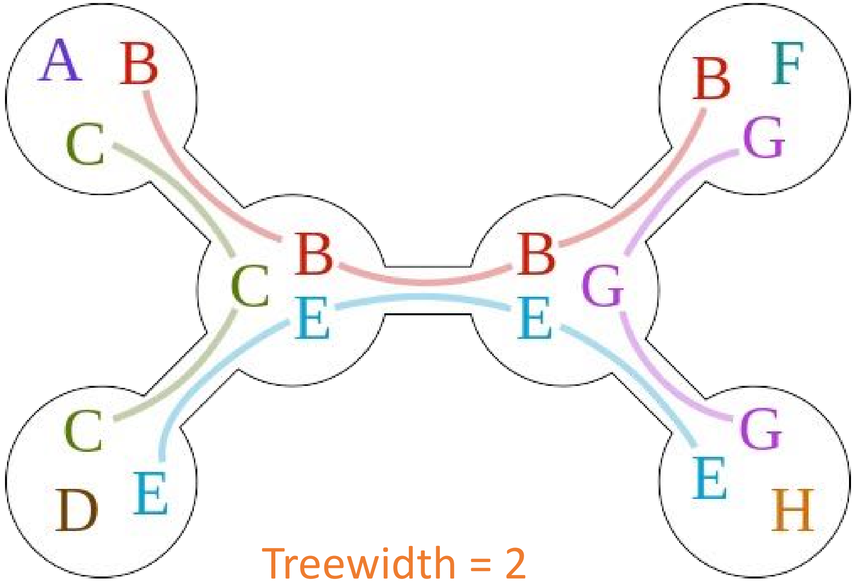
(1) Node coverage: Every vertex of G is assigned at least one vertex in T

(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





Treewidth = 2

Notice running intersection property

Example from: https://en.wikipedia.org/wiki/Tree_decomposition

Tree decomposition example 3

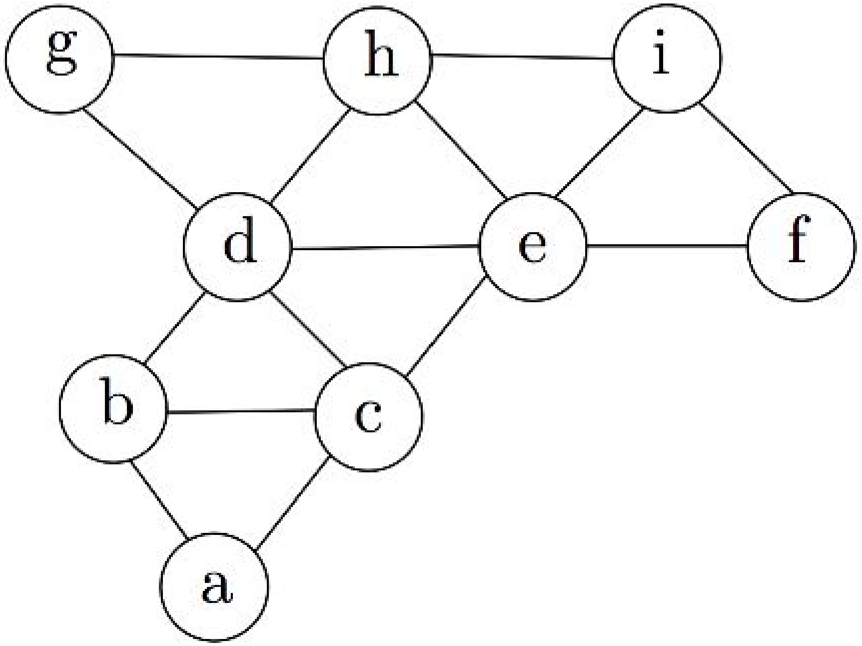
A tree decomposition of graph $G(N, E)$ is a tree $T(V, F)$ and a subset $N_v \subseteq N$ assigned to each vertex (or "supernode") $v \in V$ s.t.:

(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 from: https://www.mi.fu-berlin.de/en/inf/groups/abi/teaching/lectures/lectures_past/WS0910/V____Discrete_Mathematics_for_Bioinformatics__P1/material/scripts/treedecomposition1.pdf

Tree decomposition example 3

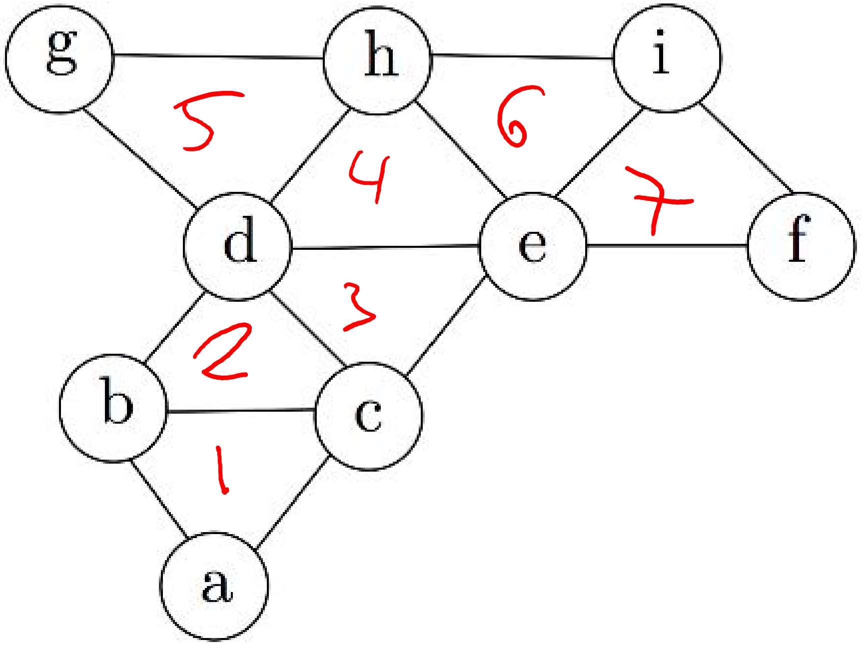
A tree decomposition of graph $G(N, E)$ is a tree $T(V, F)$ and a subset $N_v \subseteq N$ assigned to each vertex (or "supernode") $v \in V$ s.t.:

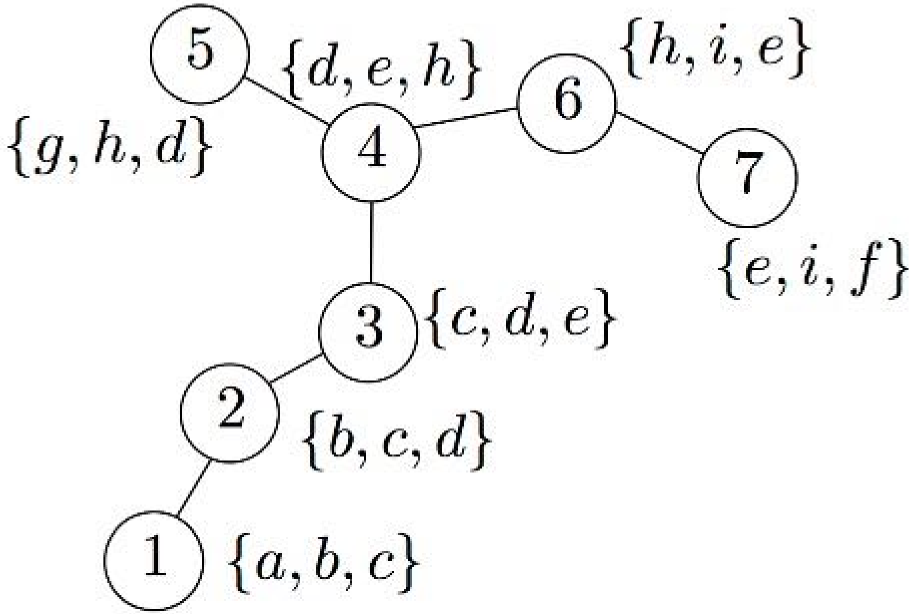
(1) Node coverage: Every vertex of G is assigned at least one vertex in T

(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 4: a cycle

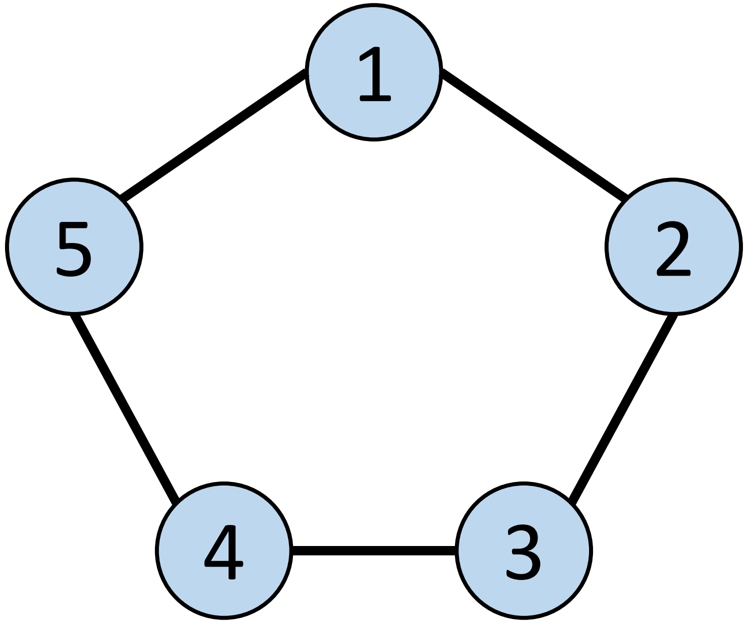
A tree decomposition of graph $G(N, E)$ is a tree $T(V, F)$ and a subset $N_v \subseteq N$ assigned to each vertex (or "supernode") $v \in V$ s.t.:

(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 4: a cycle

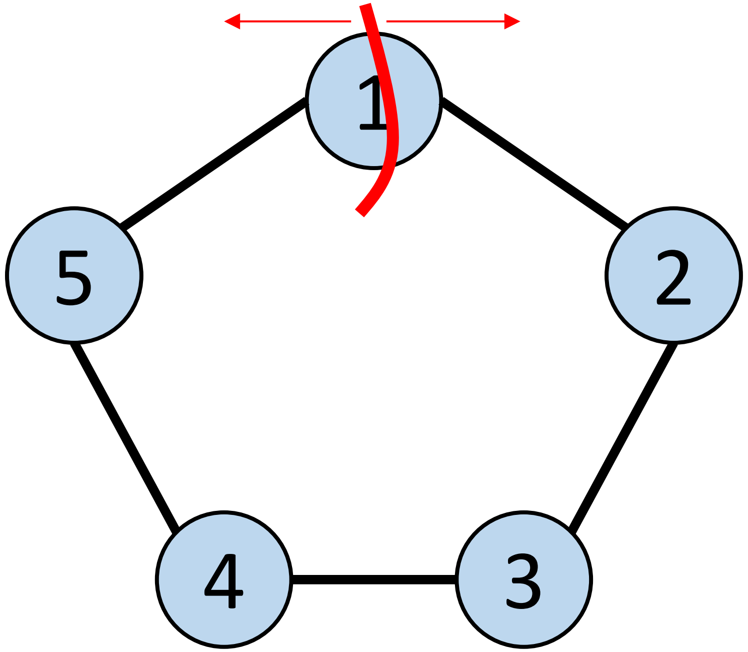
A tree decomposition of graph $G(N, E)$ is a tree $T(V, F)$ and a subset $N_v \subseteq N$ assigned to each vertex (or "supernode") $v \in V$ s.t.:

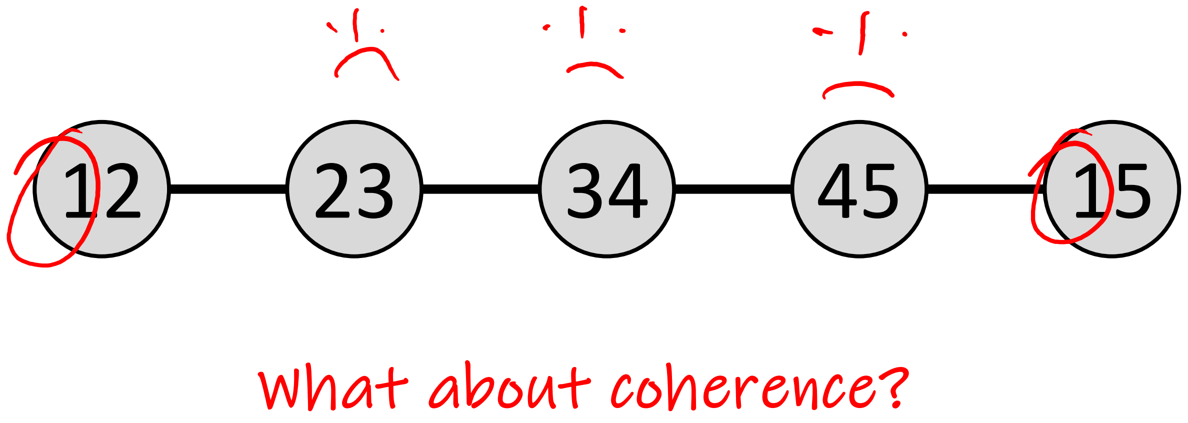
(1) Node coverage: Every vertex of G is assigned at least one vertex in T

(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 4: a cycle

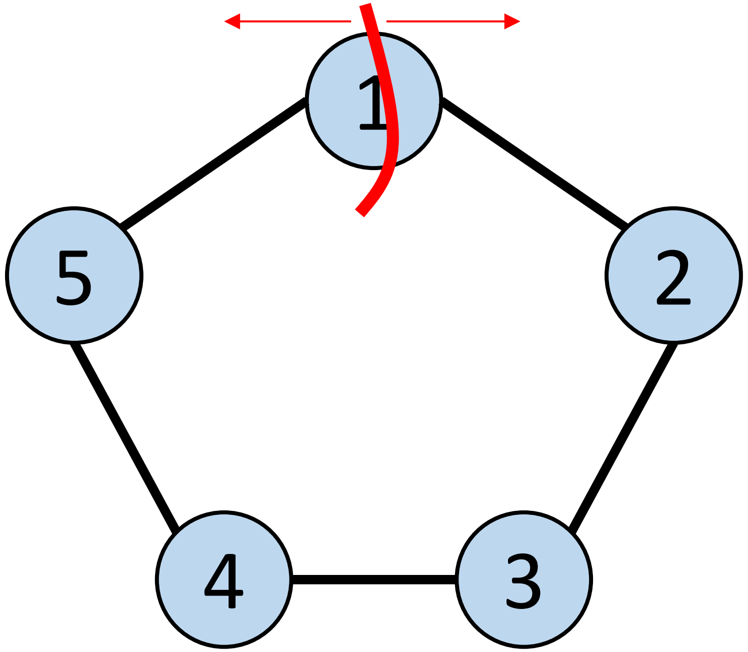
A tree decomposition of graph $G(N, E)$ is a tree $T(V, F)$ and a subset $N_v \subseteq N$ assigned to each vertex (or "supernode") $v \in V$ s.t.:

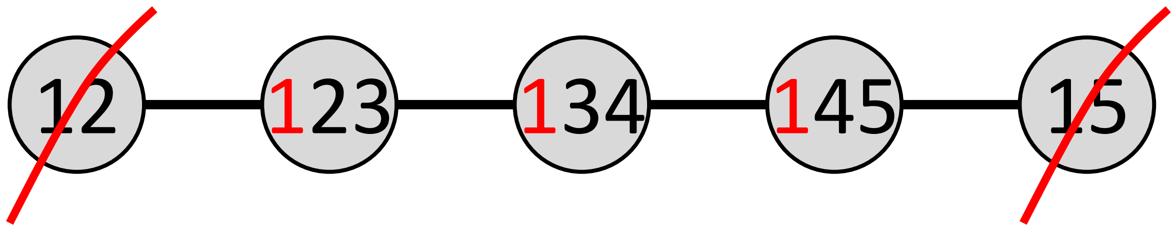
(1) Node coverage: Every vertex of G is assigned at least one vertex in T

(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 4: a cycle

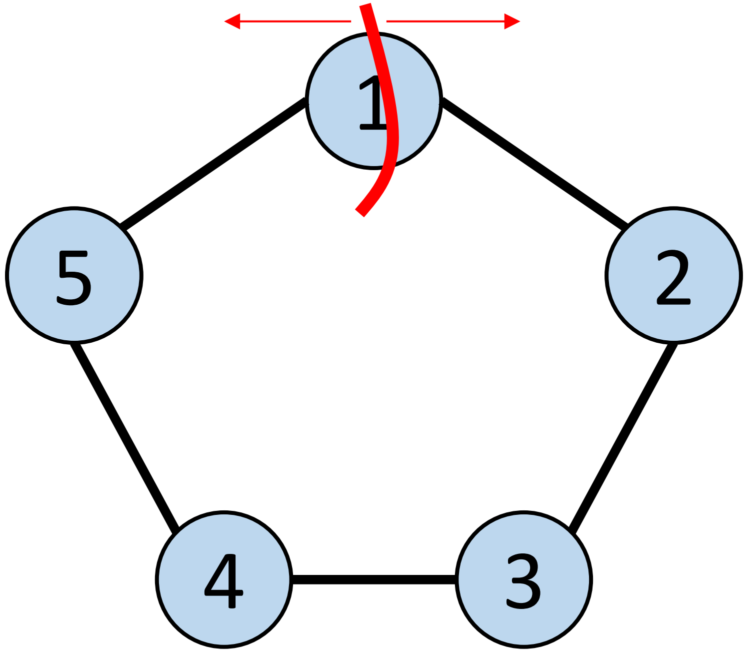
A tree decomposition of graph $G(N, E)$ is a tree $T(V, F)$ and a subset $N_v \subseteq N$ assigned to each vertex (or "supernode") $v \in V$ s.t.:

(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 4: a cycle

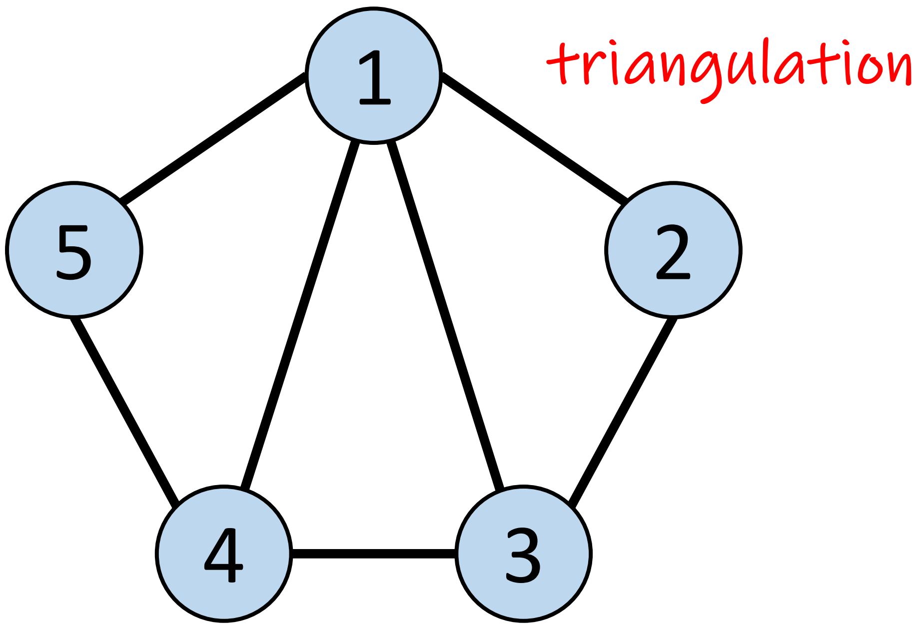
A tree decomposition of graph $G(N, E)$ is a tree $T(V, F)$ and a subset $N_v \subseteq N$ assigned to each vertex (or "supernode") $v \in V$ s.t.:

(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

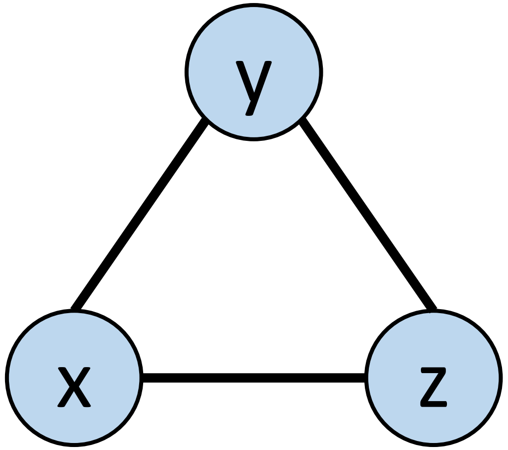
A tree decomposition of graph $G(N, E)$ is a tree $T(V, F)$ and a subset $N_v \subseteq N$ assigned to each vertex (or "supernode") $v \in V$ s.t.:

(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



tree decomposition

?

Tree decomposition example 5: the triangle

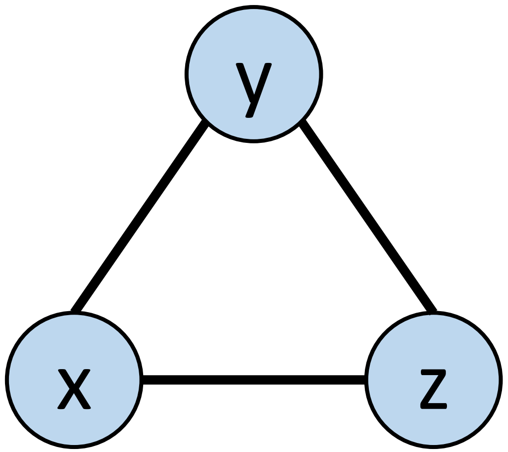
A tree decomposition of graph $\mathrm{G}(\mathrm{N}, \mathrm{E})$ is a tree $\mathrm{T}(\mathrm{V}, \mathrm{F})$ and a subset $\mathrm{N}_\mathrm{v} \subseteq \mathrm{N}$ assigned to each vertex (or "supernode") $\mathrm{v} \in \mathrm{V}$ s.t.:

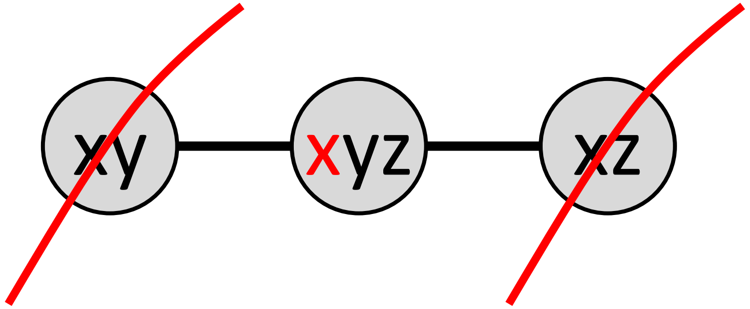
(1) Node coverage: Every vertex of G is assigned at least one vertex in T

(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

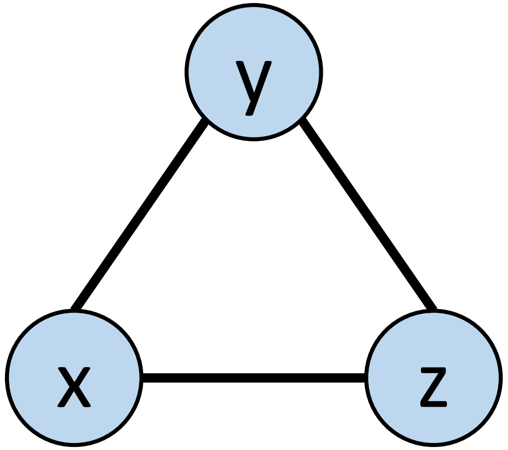
A tree decomposition of graph $G(N, E)$ is a tree $T(V, F)$ and a subset $N_v \subseteq N$ assigned to each vertex (or "supernode") $v \in V$ s.t.:

(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 5: the triangle

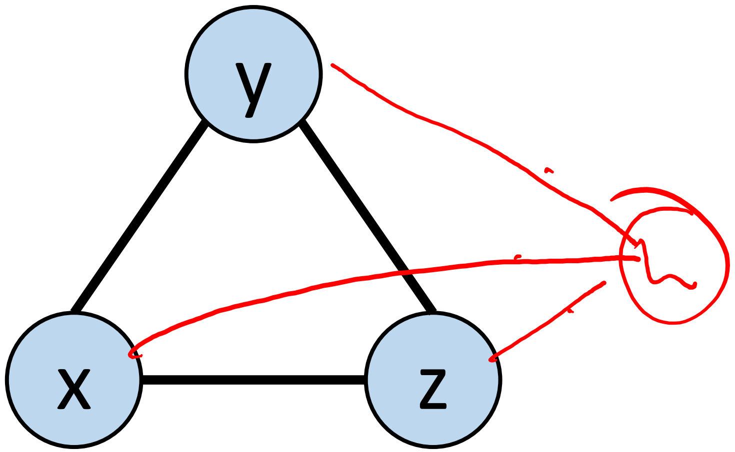
A tree decomposition of graph $G(N, E)$ is a tree $T(V, F)$ and a subset $N_v \subseteq N$ assigned to each vertex (or "supernode") $v \in V$ s.t.:

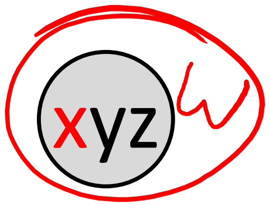
(1) Node coverage: Every vertex of G is assigned at least one vertex in T

(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

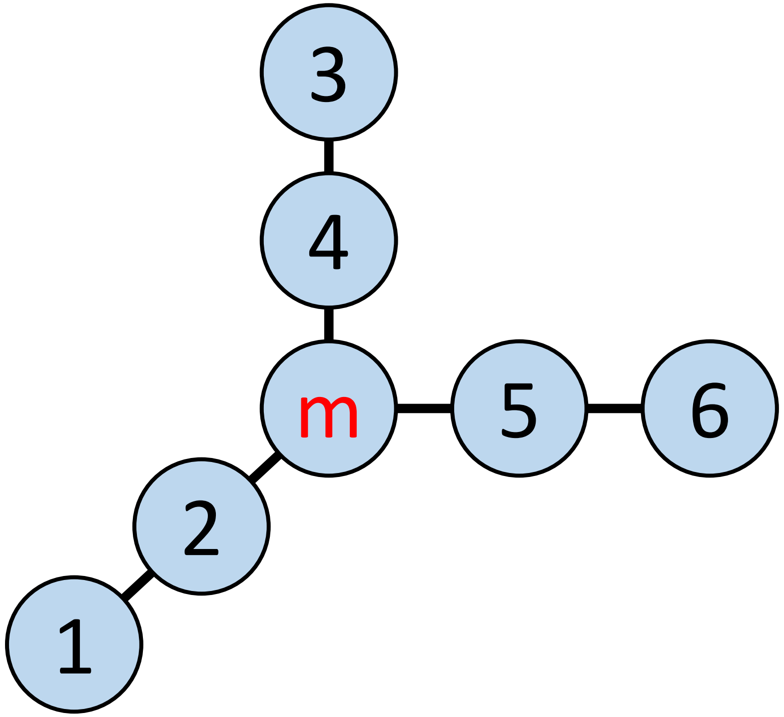
A tree decomposition of graph $G(N, E)$ is a tree $T(V, F)$ and a subset $N_v \subseteq N$ assigned to each vertex (or "supernode") $v \in V$ s.t.:

(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

?

Tree decomposition example 6: a longer tree

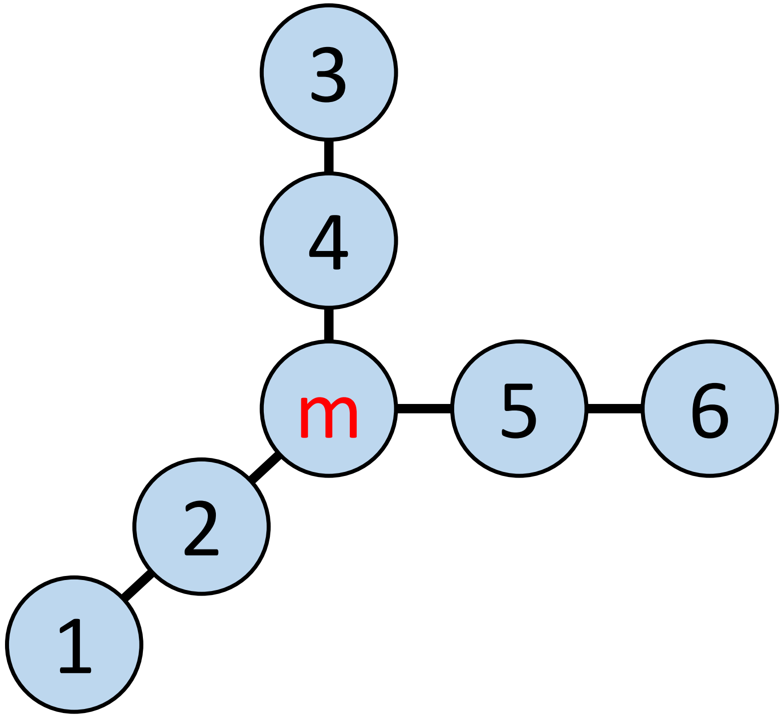
A tree decomposition of graph $G(N, E)$ is a tree $T(V, F)$ and a subset $N_v \subseteq N$ assigned to each vertex (or "supernode") $v \in V$ s.t.:

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

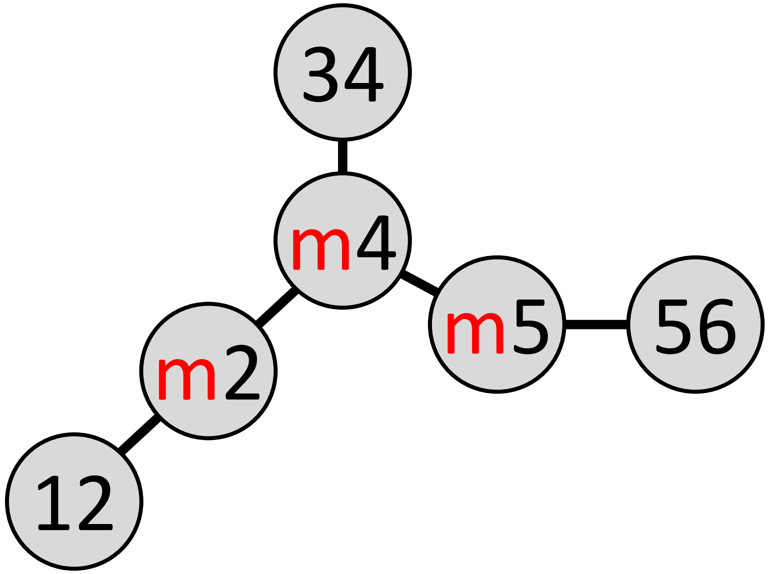
(2) Edge coverage: For every edge e of G, there is a vertex in T that contains both ends of e

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

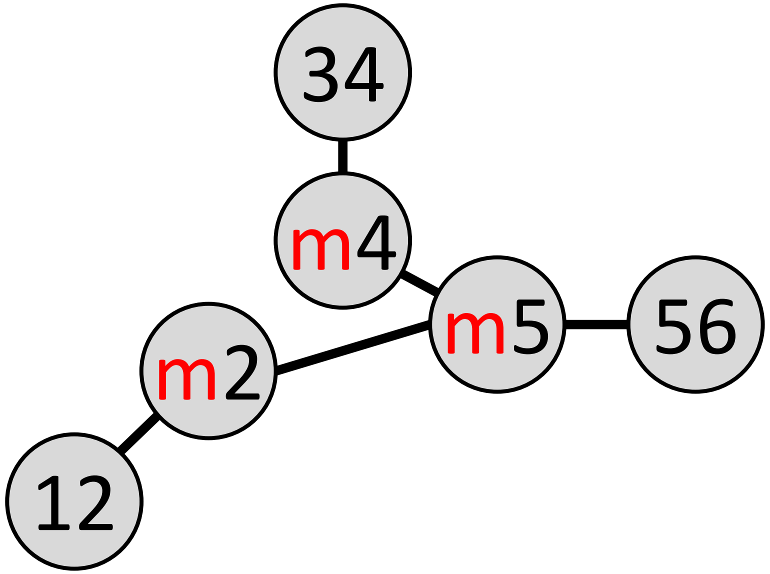
The width of a tree decomposition is the size of its largest set minus one



Tree decomposition



Another tree decomposition



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.:

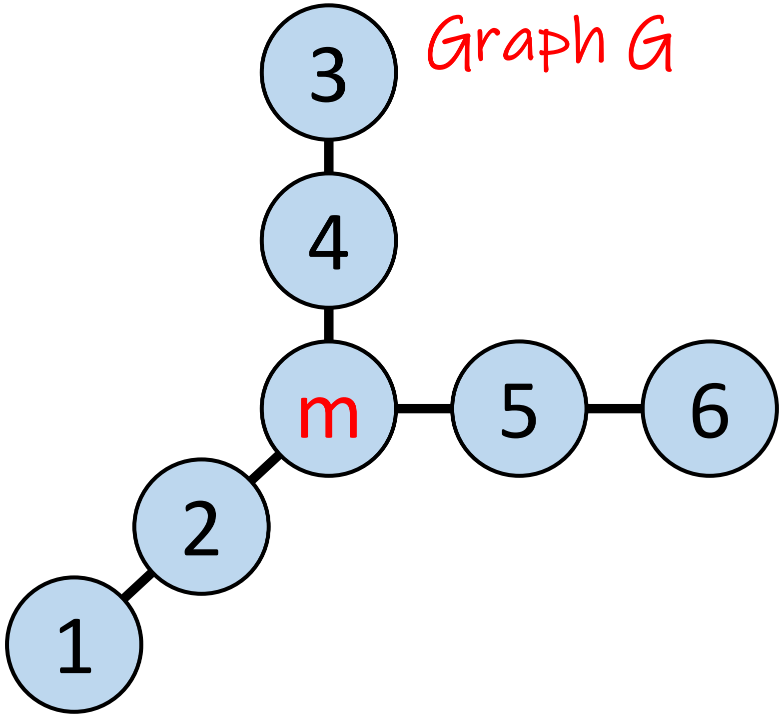
(1) Node coverage: Every vertex of G is assigned at least one vertex in T

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

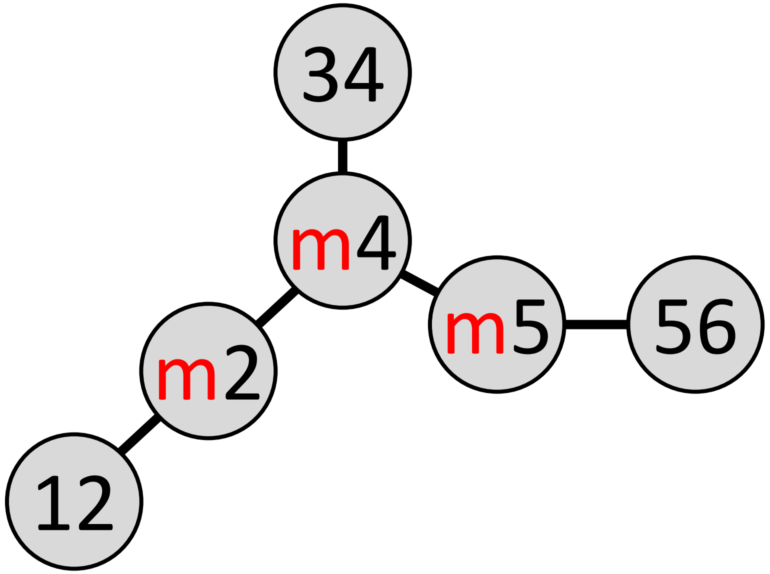
(3) Coherence: The tree is "attribute-connected"

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

Graph G



Tree decomposition



Line graph L(G) ?

- Nodes of L(G) are edges of G
- Edges of L(G) are drawn between nodes that share common endpoints in G

Tree decomposition example 6: a longer tree

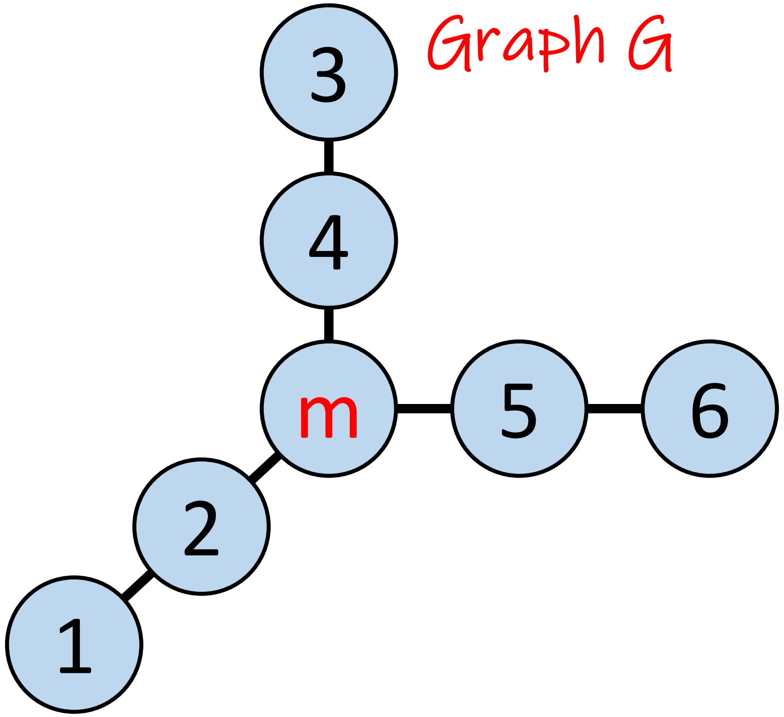
A tree decomposition of graph $\mathrm{G(N, E)}$ is a tree $\mathrm{T(V, F)}$ and a subset $\mathrm{N_v \subseteq N}$ assigned to each vertex (or "supernode") $\mathrm{v \in V}$ s.t.:

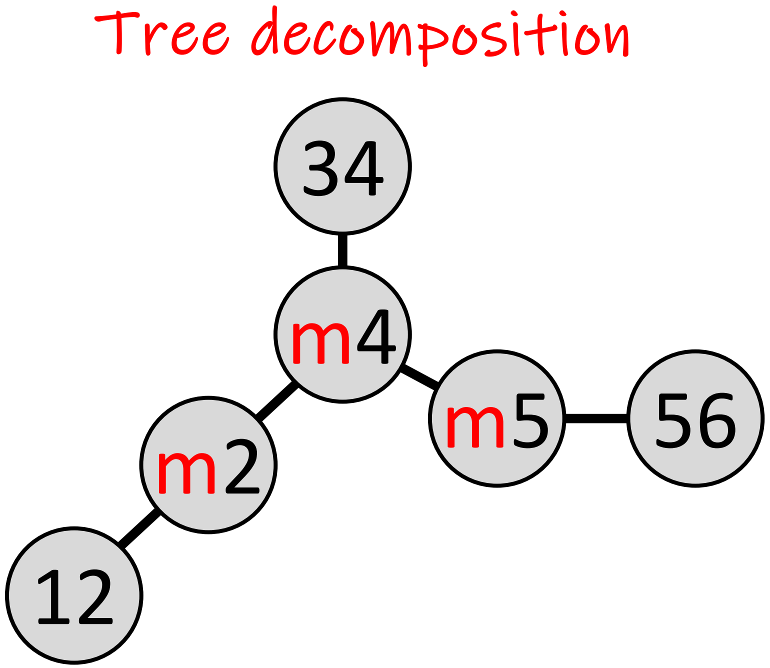
(1) Node coverage: Every vertex of G is assigned at least one vertex in T

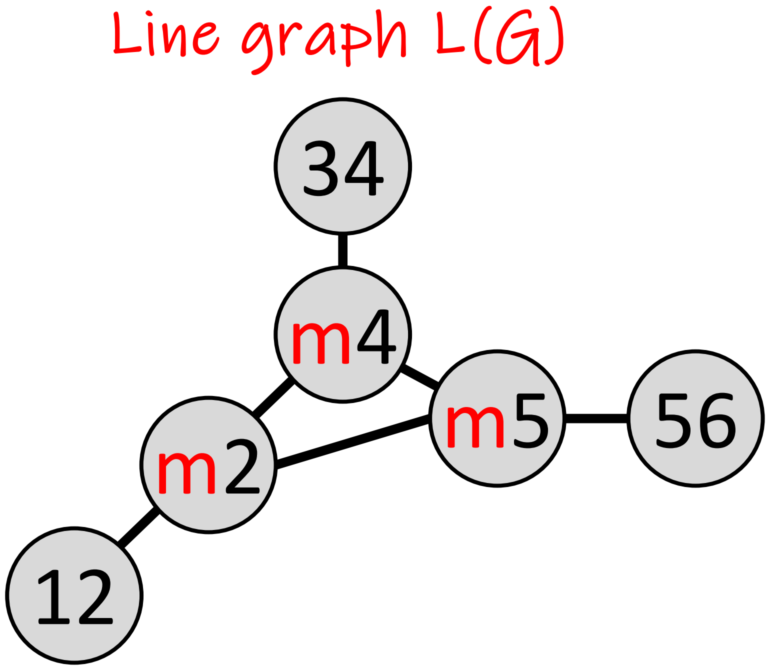
(2) Edge coverage: For every edge e of G, there is a vertex in T that contains both ends of e

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

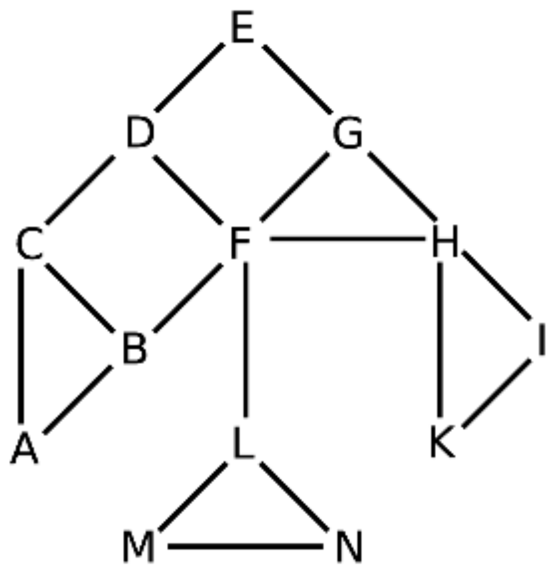
The width of a tree decomposition is the size of its largest set minus one



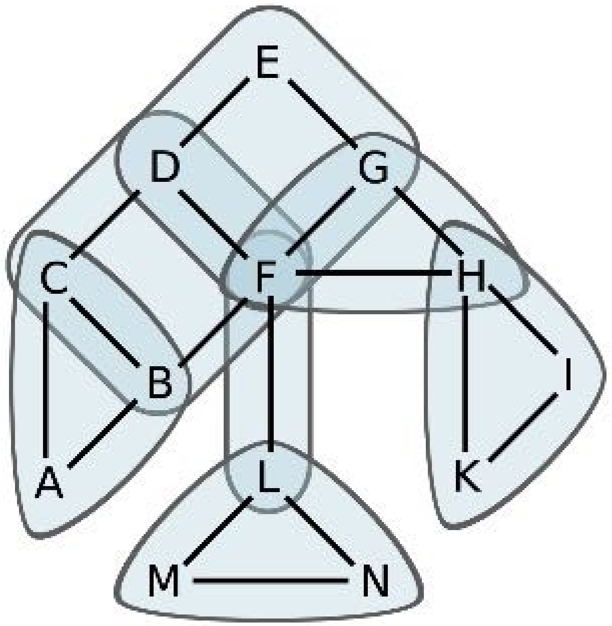




Tree decomposition example 7

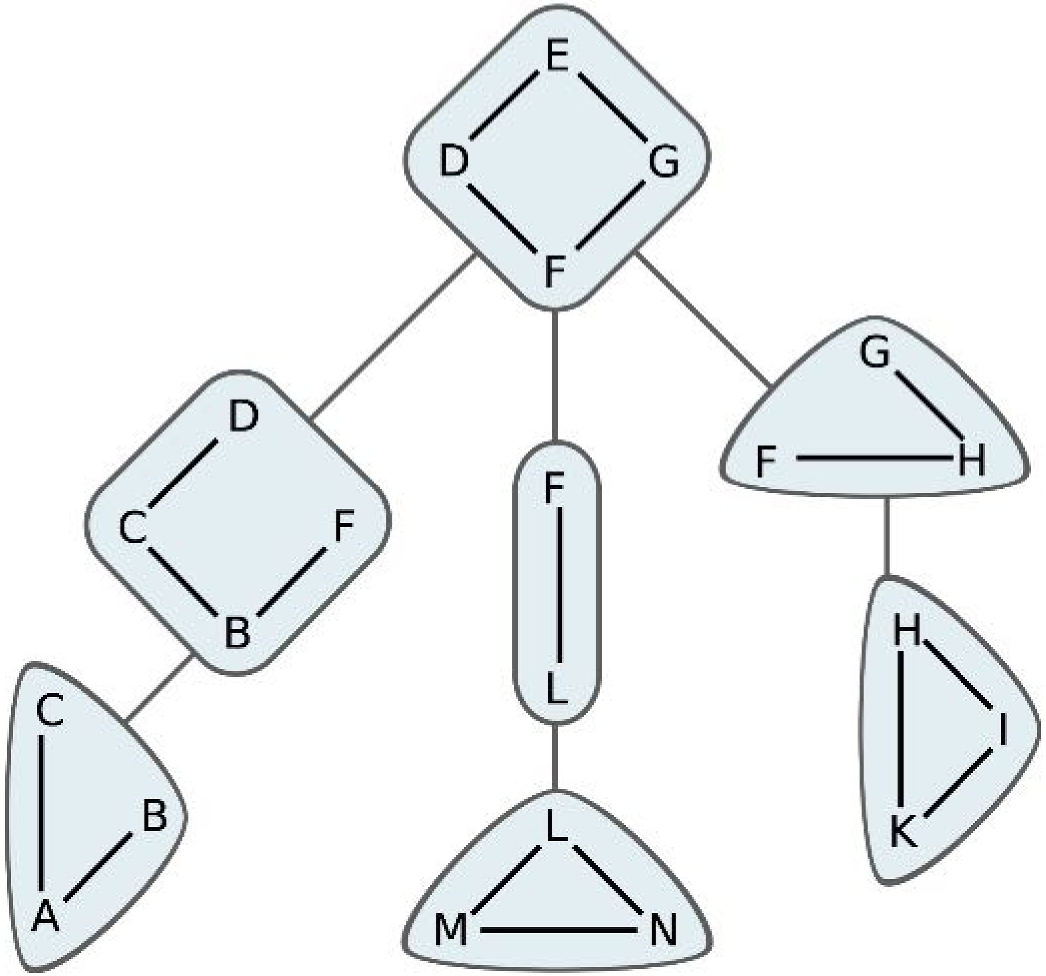


Tree decomposition example 7



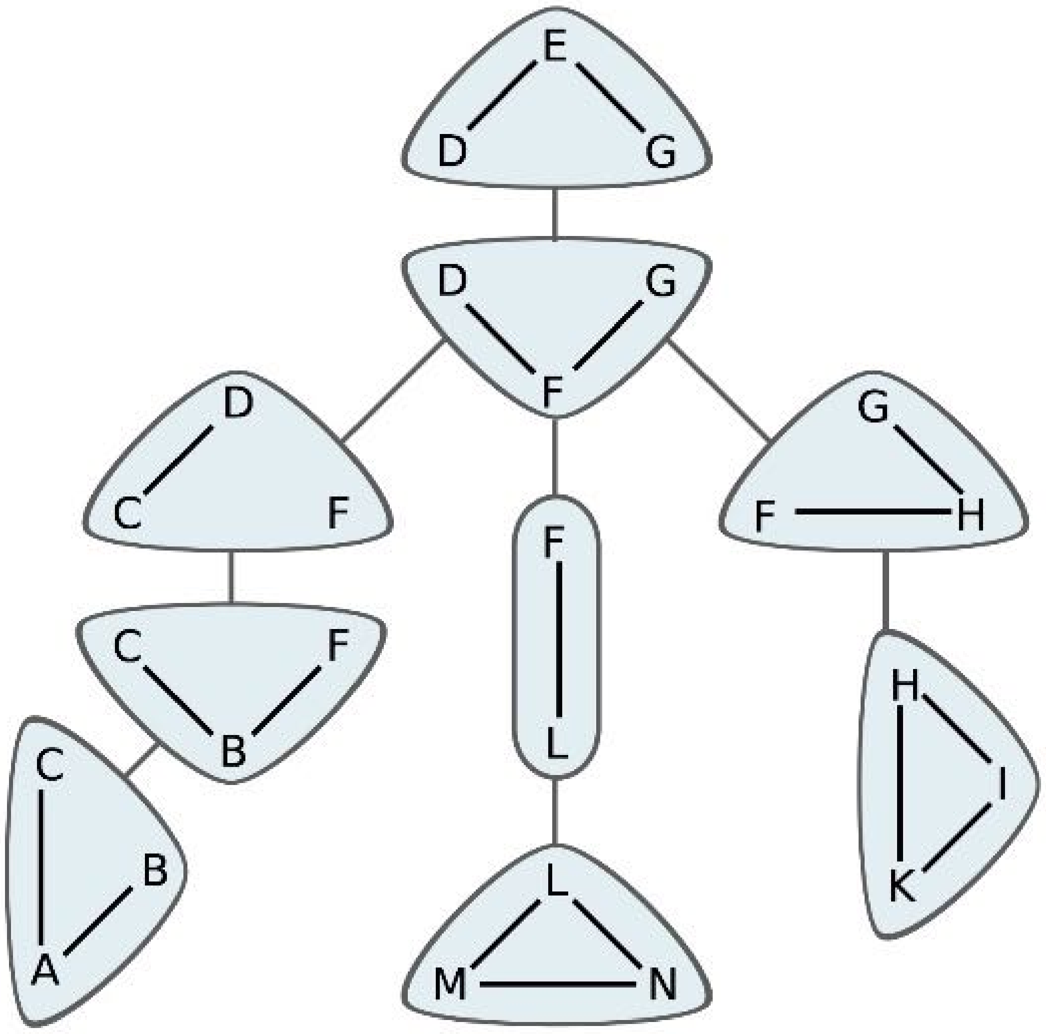
Example by: Markus Krötzsch. "Database theory: Lecture 6: Tree-like Conjunctive Queries." 2016. https://iccl.inf.tu-dresden.de/web/Database_Theory_(SS2016)/en
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Tree decomposition example 7



⤳ tree decomposition of width 3

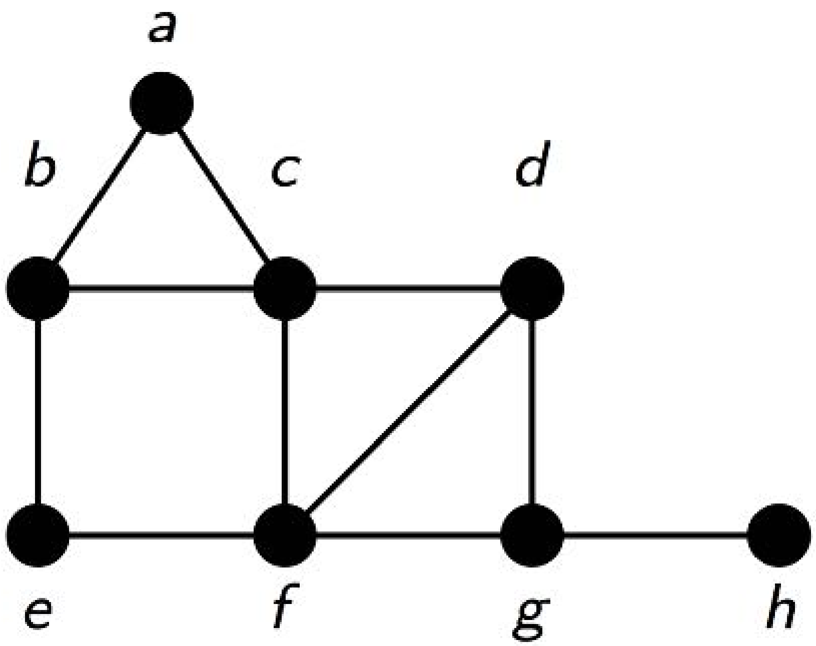
Tree decomposition example 7



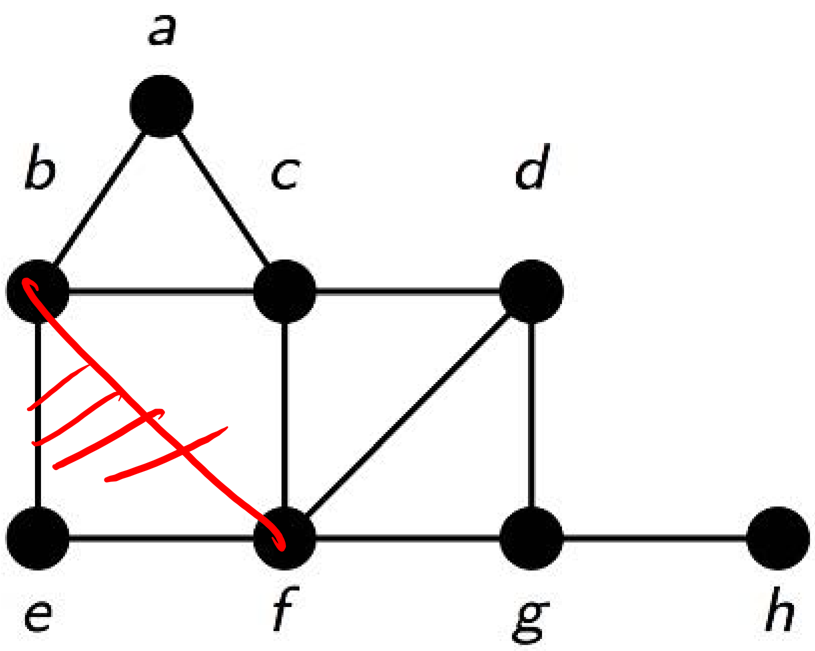
⤳ tree decomposition of width 2 = treewidth of the example graph

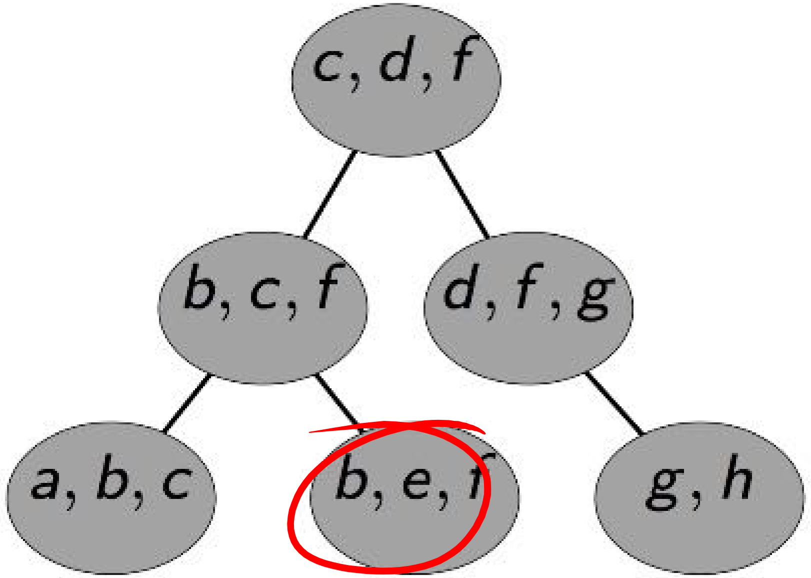
Example by: Markus Krötzsch. "Database theory: Lecture 6: Tree-like Conjunctive Queries." 2016. https://iccl.inf.tu-dresden.de/web/Database_Theory_(SS2016)/en

Tree decomposition example 8



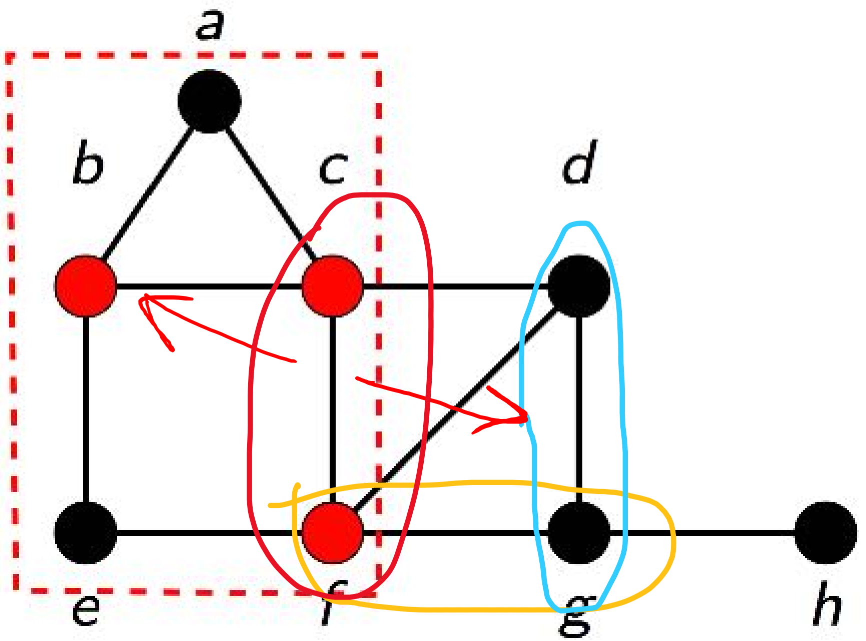
Tree decomposition example 8

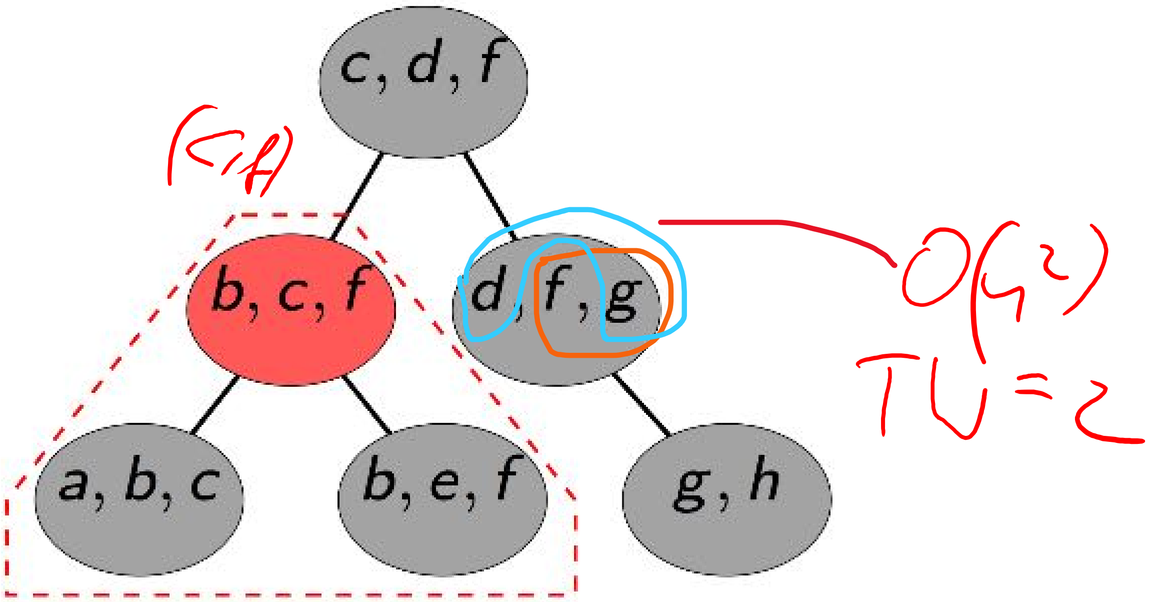




Example by: Marx. "Graphs, hypergraphs, and the complexity of conjunctive database queries", ICDT 2017. http://edbticdt2017.unive.it/marx-icdt2017-talk.pdf

Tree decomposition example 8

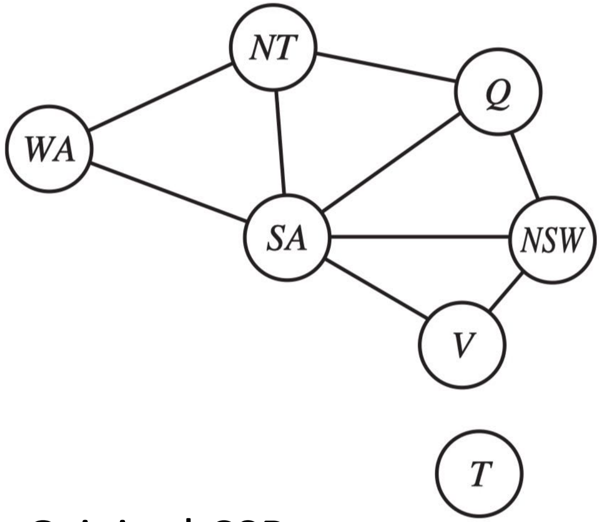




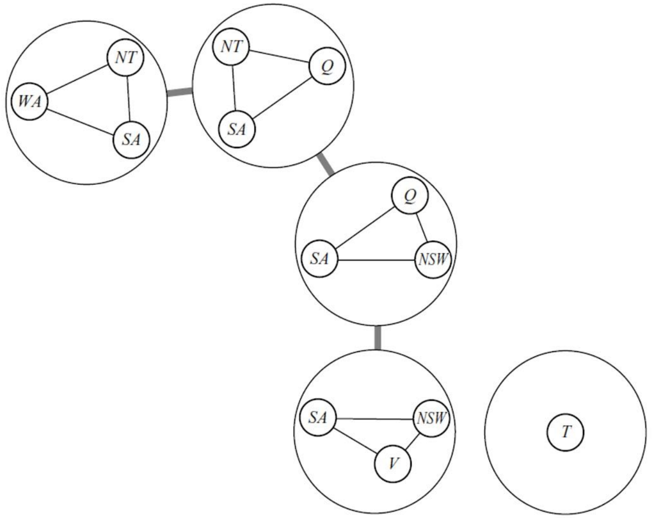
A subtree communicates with the outside world only via the root of the subtree.

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 $\mathrm{G(N, E)}$ is a tree $\mathrm{T(V, F)}$ and a subset $\mathrm{N_v} \subseteq \mathrm{N}$ assigned to each vertex (or "supernode") $\mathrm{v} \in \mathrm{V}$ 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 χ 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.

Source of alternative definition: Gottlob, Leone, Scarcello. Hypertree decompositions: a survey. MFCS 2001. https://doi.org/10.1007/3-540-44683-4_5

Small decompositions allow to "compress" the search space

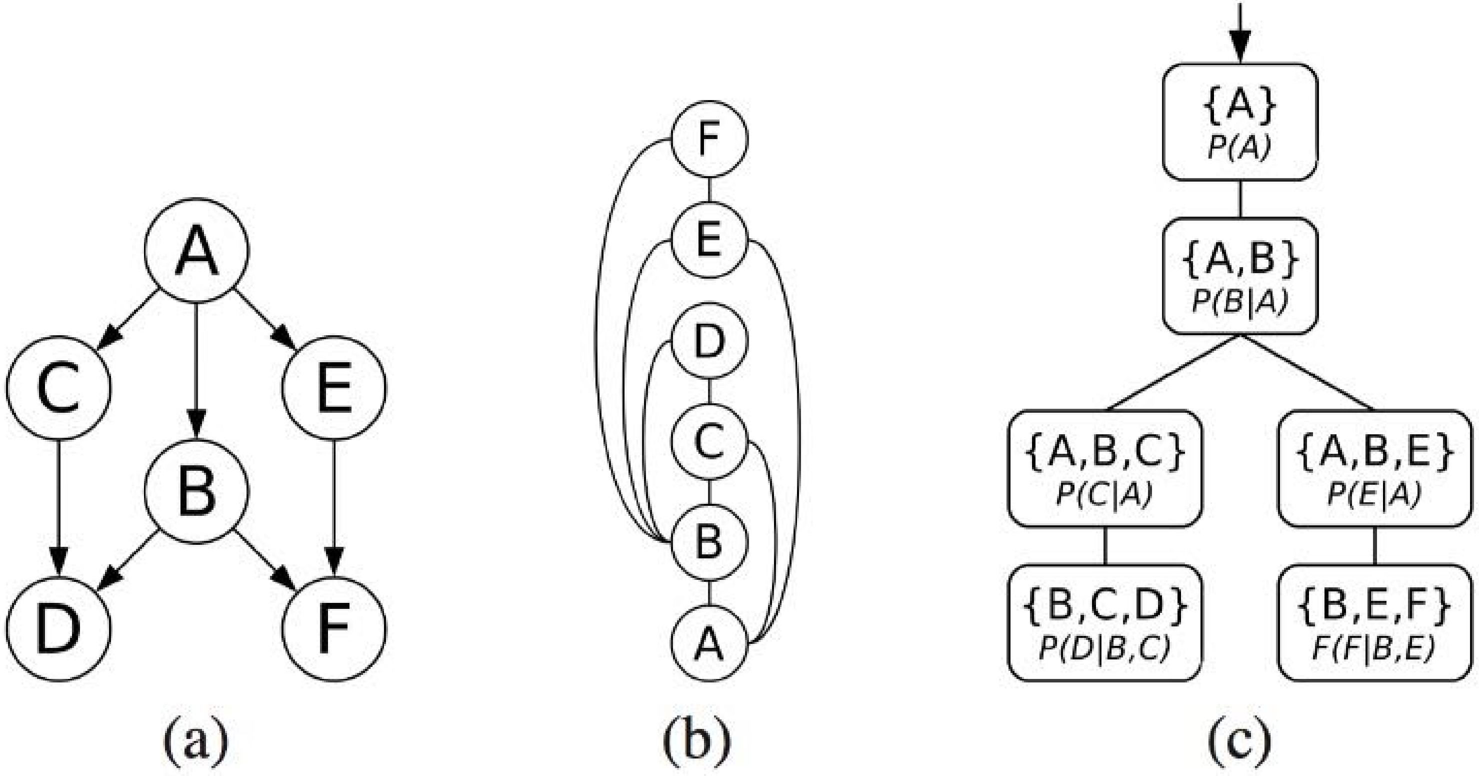


Figure 1: Example belief network, its triangulated primal graph along ordering $d = A, B, C, D, E, F$, and the corresponding bucket tree decomposition.

Figure from: Otten, Dechter. Bounding Search Space Size via (Hyper)tree Decompositions. UAI 2008. https://arxiv.org/abs/1206.3284

Explaining Treewidth with cops & robbers

Pursuit-evasion games

- Pursuit-evasion (sometimes called "cops and robber") is a family of problems in which one group (cops) attempts to track down members of another group (robbers) in some structured environment, usually graphs.
- Related to pebble games and Ehrenfeucht–Fraïssé games

- Next: A variations of "Cops and Robber" can be used to describe the treewidth of a graph

For more details see: https://en.wikipedia.org/wiki/Pursuit%E2%80%93evasion, https://en.wikipedia.org/wiki/Pebble_game, https://en.wikipedia.org/wiki/Ehrenfeucht%E2%80%93Fra%C3%AFss%C3%A9_game, https://en.wikipedia.org/wiki/Cop_number#Special_classes_of_graphs

Treewidth with Cops and robber

k cops and 1 robber move on vertices of a graph. The robber can move quickly along paths that are not blocked by cops. Cops can fly via helicopters to new nodes. You control the cops and want to catch the robber (catch = occupy the same node). A single move consists of:

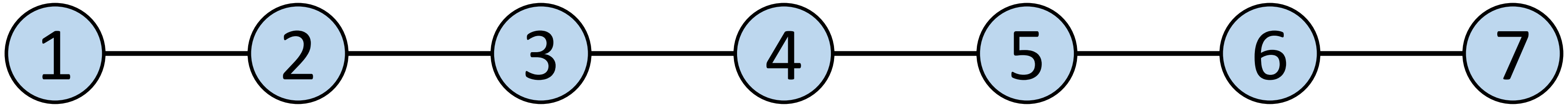
(1) A **cop** flies off the graph in a helicopter and announces a new landing vertex.

(2) While the cop flies, the **robber** can move quickly along the edges and escape.

(3) Then the **cop** lands.

can also take multiple steps

> THEOREM [Seymour & Thomas (1993)]
>
> You have a winning strategy with k cops iff the tree-width of the graph is at most k−1.



Seymour, Thomas. Graph searching and a min-max theorem for tree-width, Journal of Combinatorial Theory, Series B, 1993. https://doi.org/10.1006/jctb.1993.1027

Treewidth with Cops and robber

k cops and 1 robber move on vertices of a graph. The robber can move quickly along paths that are not blocked by cops. Cops can fly via helicopters to new nodes. You control the cops and want to catch the robber (catch = occupy the same node). A single move consists of:

(1) A cop flies off the graph in a helicopter and announces a new landing vertex.

(2) While the cop flies, the robber can move quickly along the edges and escape.

(3) Then the cop lands.

Seymour, Thomas. Graph searching and a min-max theorem for tree-width, Journal of Combinatorial Theory, Series B, 1993. https://doi.org/10.1006/jctb.1993.1027
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

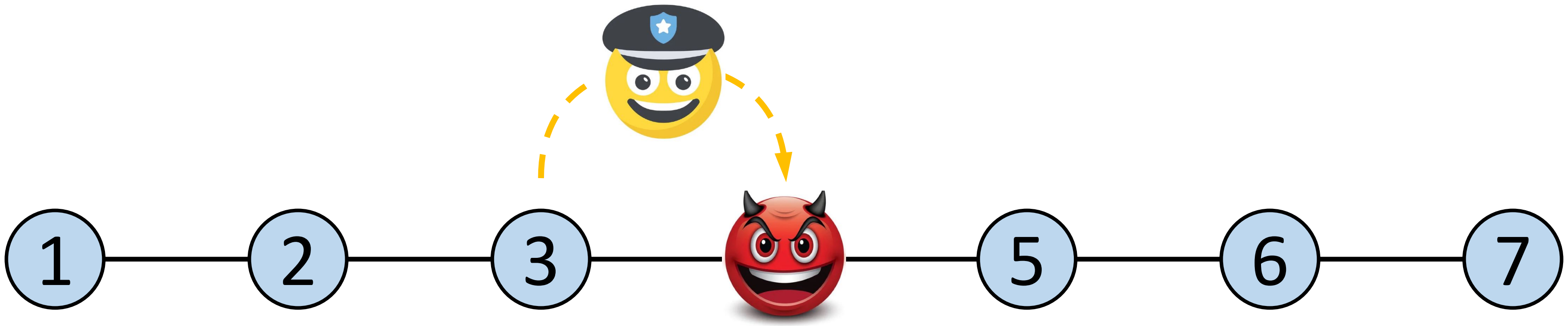
Treewidth with Cops and robber

k cops and 1 robber move on vertices of a graph. The robber can move quickly along paths that are not blocked by cops. Cops can fly via helicopters to new nodes. You control the cops and want to catch the robber (catch = occupy the same node). A single move consists of:

(1) A cop flies off the graph in a helicopter and announces a new landing vertex.

(2) While the cop flies, the robber can move quickly along the edges and escape.

(3) Then the cop lands.



Seymour, Thomas. Graph searching and a min-max theorem for tree-width, Journal of Combinatorial Theory, Series B, 1993. https://doi.org/10.1006/jctb.1993.1027

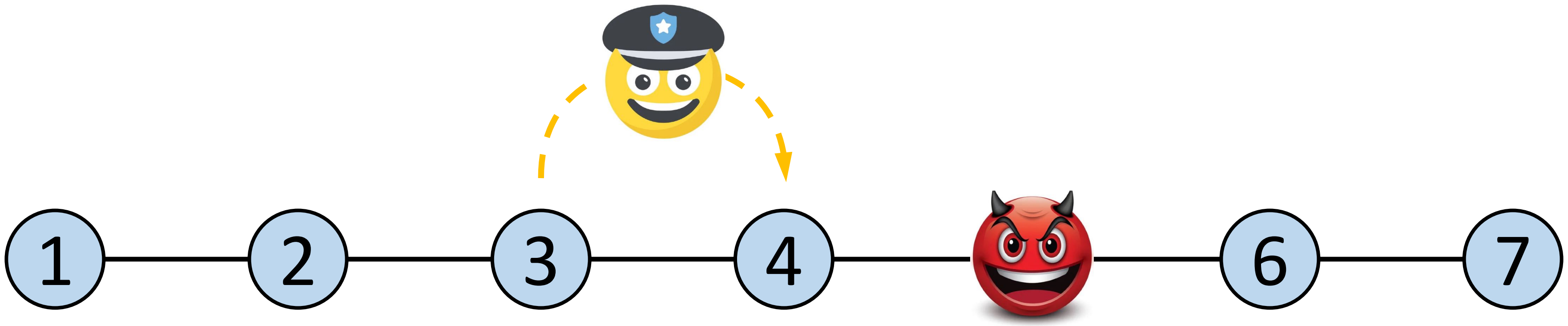
Treewidth with Cops and robber

k cops and 1 robber move on vertices of a graph. The robber can move quickly along paths that are not blocked by cops. Cops can fly via helicopters to new nodes. You control the cops and want to catch the robber (catch = occupy the same node). A single move consists of:

(1) A cop flies off the graph in a helicopter and announces a new landing vertex.

(2) While the cop flies, the robber can move quickly along the edges and escape.

(3) Then the cop lands.



Seymour, Thomas. Graph searching and a min-max theorem for tree-width, Journal of Combinatorial Theory, Series B, 1993. https://doi.org/10.1006/jctb.1993.1027
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Treewidth with Cops and robber

k cops and 1 robber move on vertices of a graph. The robber can move quickly along paths that are not blocked by cops. Cops can fly via helicopters to new nodes. You control the cops and want to catch the robber (catch = occupy the same node). A single move consists of:

(1) A cop flies off the graph in a helicopter and announces a new landing vertex.

(2) While the cop flies, the robber can move quickly along the edges and escape.

(3) Then the cop lands.

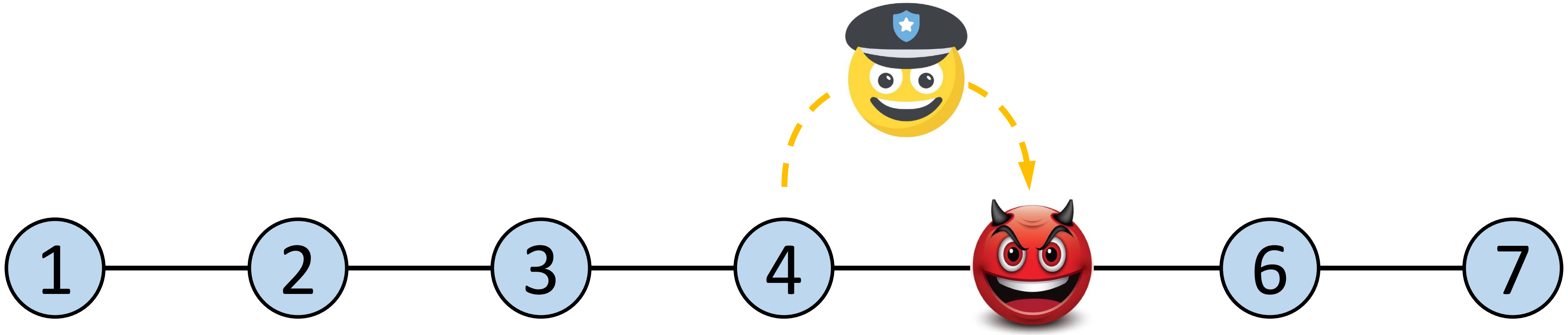
Treewidth with Cops and robber

k cops and 1 robber move on vertices of a graph. The robber can move quickly along paths that are not blocked by cops. Cops can fly via helicopters to new nodes. You control the cops and want to catch the robber (catch = occupy the same node). A single move consists of:

(1) A cop flies off the graph in a helicopter and announces a new landing vertex.

(2) While the cop flies, the robber can move quickly along the edges and escape.

(3) Then the cop lands.



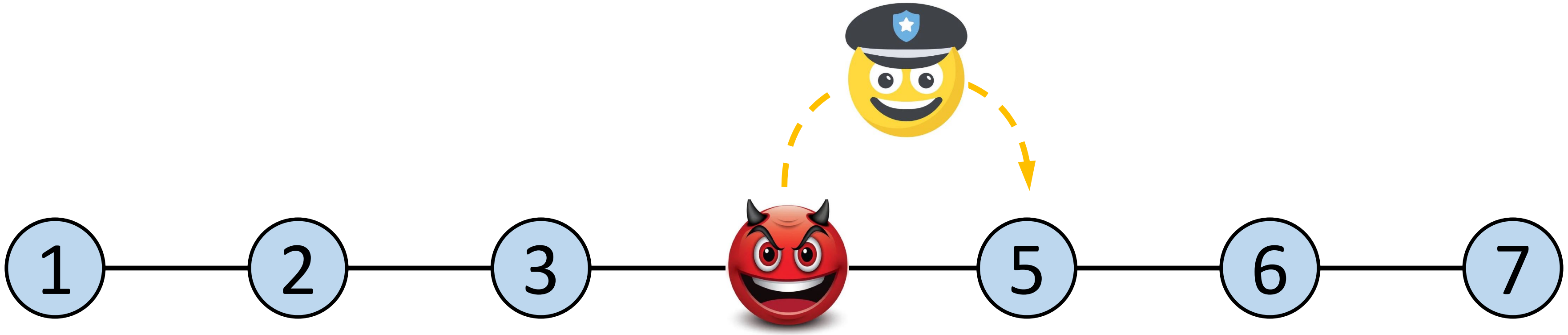
Treewidth with Cops and robber

k cops and 1 robber move on vertices of a graph. The robber can move quickly along paths that are not blocked by cops. Cops can fly via helicopters to new nodes. You control the cops and want to catch the robber (catch = occupy the same node). A single move consists of:

(1) A cop flies off the graph in a helicopter and announces a new landing vertex.

(2) While the cop flies, the robber can move quickly along the edges and escape.

(3) Then the cop lands.



Treewidth with Cops and robber

k cops and 1 robber move on vertices of a graph. The robber can move quickly along paths that are not blocked by cops. Cops can fly via helicopters to new nodes. You control the cops and want to catch the robber (catch = occupy the same node). A single move consists of:

(1) A cop flies off the graph in a helicopter and announces a new landing vertex.

(2) While the cop flies, the robber can move quickly along the edges and escape.

(3) Then the cop lands.

You can never catch the robber with only one cop ☹

Treewidth with Cops and robber

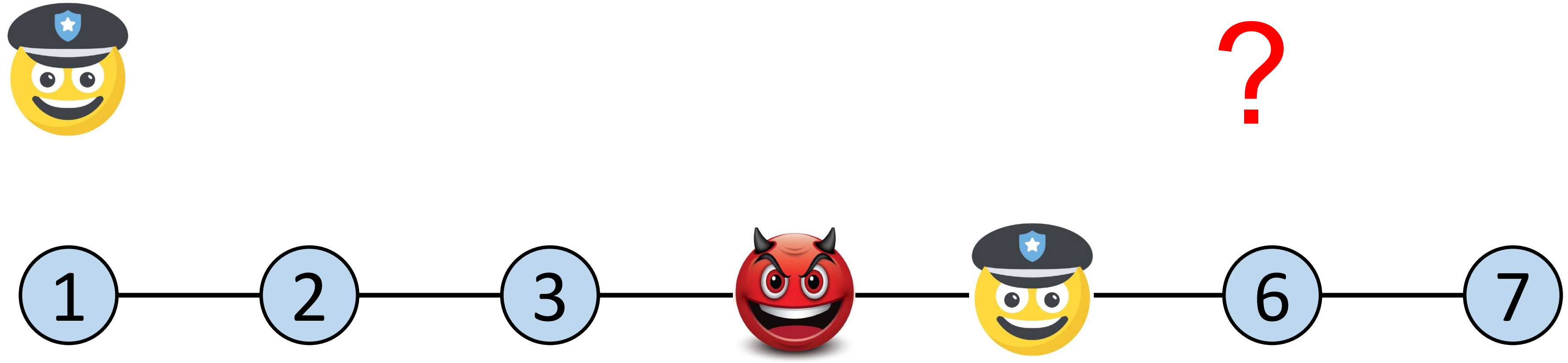
k cops and 1 robber move on vertices of a graph. The robber can move quickly along paths that are not blocked by cops. Cops can fly via helicopters to new nodes. You control the cops and want to catch the robber (catch = occupy the same node). A single move consists of:

(1) A cop flies off the graph in a helicopter and announces a new landing vertex.

(2) While the cop flies, the robber can move quickly along the edges and escape.

(3) Then the cop lands.

What is the best move with a 2nd cop



Treewidth with Cops and robber

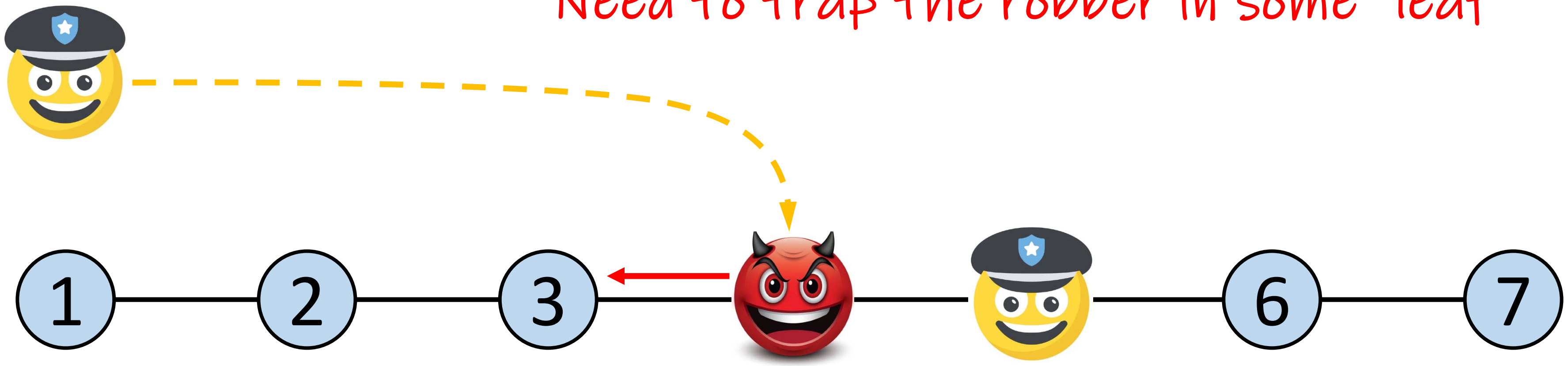
k cops and 1 robber move on vertices of a graph. The robber can move quickly along paths that are not blocked by cops. Cops can fly via helicopters to new nodes. You control the cops and want to catch the robber (catch = occupy the same node). A single move consists of:

(1) A cop flies off the graph in a helicopter and announces a new landing vertex.

(2) While the cop flies, the robber can move quickly along the edges and escape.

(3) Then the cop lands.

One cop moves in on the robber, while others block escape.
Need to trap the robber in some "leaf"



Seymour, Thomas. Graph searching and a min-max theorem for tree-width, Journal of Combinatorial Theory, Series B, 1993. https://doi.org/10.1006/jctb.1993.1027

Treewidth with Cops and robber

k cops and 1 robber move on vertices of a graph. The robber can move quickly along paths that are not blocked by cops. Cops can fly via helicopters to new nodes. You control the cops and want to catch the robber (catch = occupy the same node). A single move consists of:

(1) A cop flies off the graph in a helicopter and announces a new landing vertex.

(2) While the cop flies, the robber can move quickly along the edges and escape.

(3) Then the cop lands.

Seymour, Thomas. Graph searching and a min-max theorem for tree-width, Journal of Combinatorial Theory, Series B, 1993. https://doi.org/10.1006/jctb.1993.1027

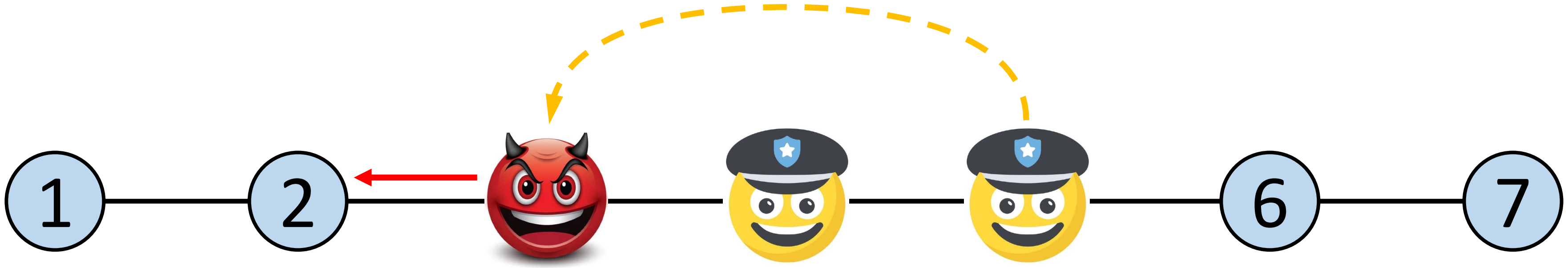
Treewidth with Cops and robber

k cops and 1 robber move on vertices of a graph. The robber can move quickly along paths that are not blocked by cops. Cops can fly via helicopters to new nodes. You control the cops and want to catch the robber (catch = occupy the same node). A single move consists of:

(1) A cop flies off the graph in a helicopter and announces a new landing vertex.

(2) While the cop flies, the robber can move quickly along the edges and escape.

(3) Then the cop lands.



Seymour, Thomas. Graph searching and a min-max theorem for tree-width, Journal of Combinatorial Theory, Series B, 1993. https://doi.org/10.1006/jctb.1993.1027

Treewidth with Cops and robber

k cops and 1 robber move on vertices of a graph. The robber can move quickly along paths that are not blocked by cops. Cops can fly via helicopters to new nodes. You control the cops and want to catch the robber (catch = occupy the same node). A single move consists of:

(1) A cop flies off the graph in a helicopter and announces a new landing vertex.

(2) While the cop flies, the robber can move quickly along the edges and escape.

(3) Then the cop lands.

Seymour, Thomas. Graph searching and a min-max theorem for tree-width, Journal of Combinatorial Theory, Series B, 1993. https://doi.org/10.1006/jctb.1993.1027

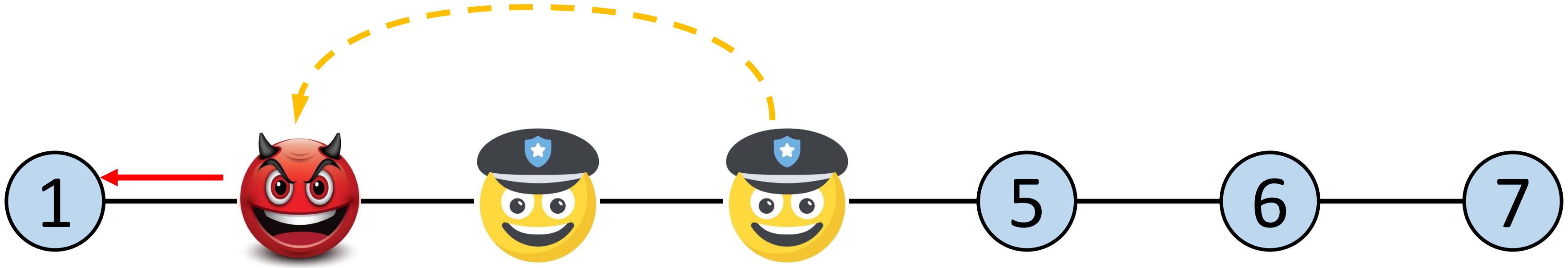
Treewidth with Cops and robber

k cops and 1 robber move on vertices of a graph. The robber can move quickly along paths that are not blocked by cops. Cops can fly via helicopters to new nodes. You control the cops and want to catch the robber (catch = occupy the same node). A single move consists of:

(1) A cop flies off the graph in a helicopter and announces a new landing vertex.

(2) While the cop flies, the robber can move quickly along the edges and escape.

(3) Then the cop lands.



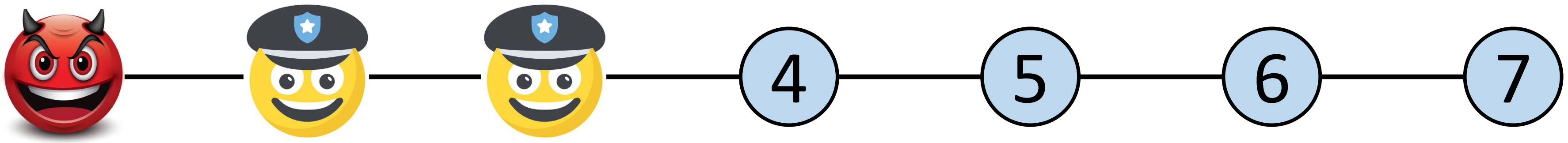
Treewidth with Cops and robber

k cops and 1 robber move on vertices of a graph. The robber can move quickly along paths that are not blocked by cops. Cops can fly via helicopters to new nodes. You control the cops and want to catch the robber (catch = occupy the same node). A single move consists of:

(1) A cop flies off the graph in a helicopter and announces a new landing vertex.

(2) While the cop flies, the robber can move quickly along the edges and escape.

(3) Then the cop lands.



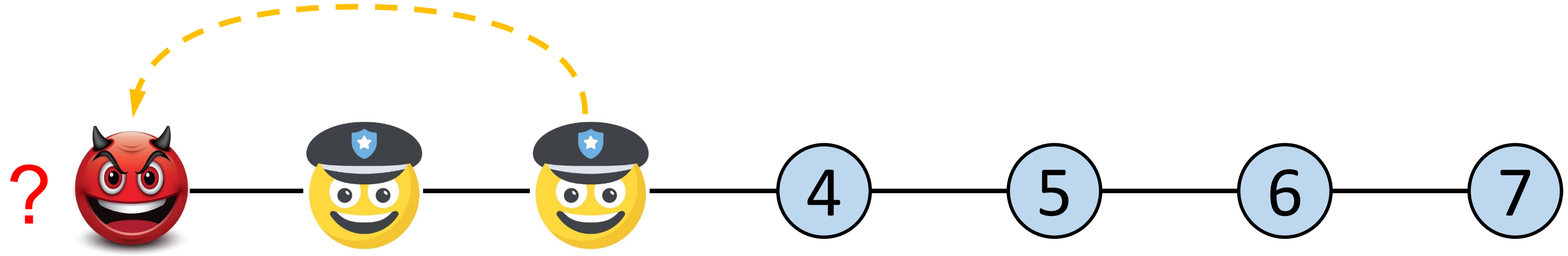
Treewidth with Cops and robber

k cops and 1 robber move on vertices of a graph. The robber can move quickly along paths that are not blocked by cops. Cops can fly via helicopters to new nodes. You control the cops and want to catch the robber (catch = occupy the same node). A single move consists of:

(1) A cop flies off the graph in a helicopter and announces a new landing vertex.

(2) While the cop flies, the robber can move quickly along the edges and escape.

(3) Then the cop lands.



Seymour, Thomas. Graph searching and a min-max theorem for tree-width, Journal of Combinatorial Theory, Series B, 1993. https://doi.org/10.1006/jctb.1993.1027

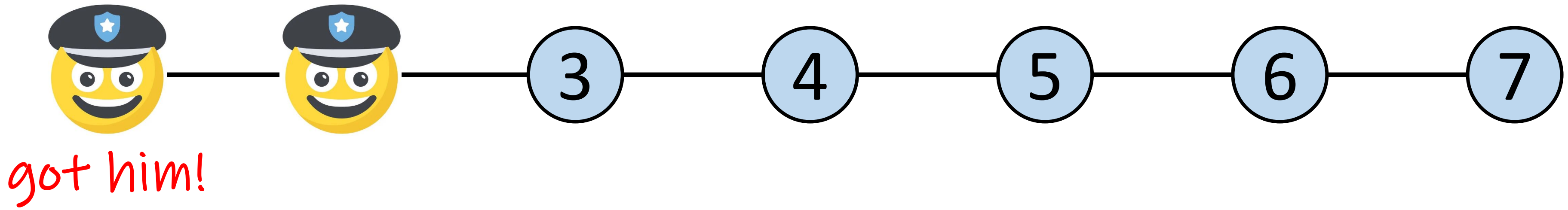
Treewidth with Cops and robber

k cops and 1 robber move on vertices of a graph. The robber can move quickly along paths that are not blocked by cops. Cops can fly via helicopters to new nodes. You control the cops and want to catch the robber (catch = occupy the same node). A single move consists of:

(1) A cop flies off the graph in a helicopter and announces a new landing vertex.

(2) While the cop flies, the robber can move quickly along the edges and escape.

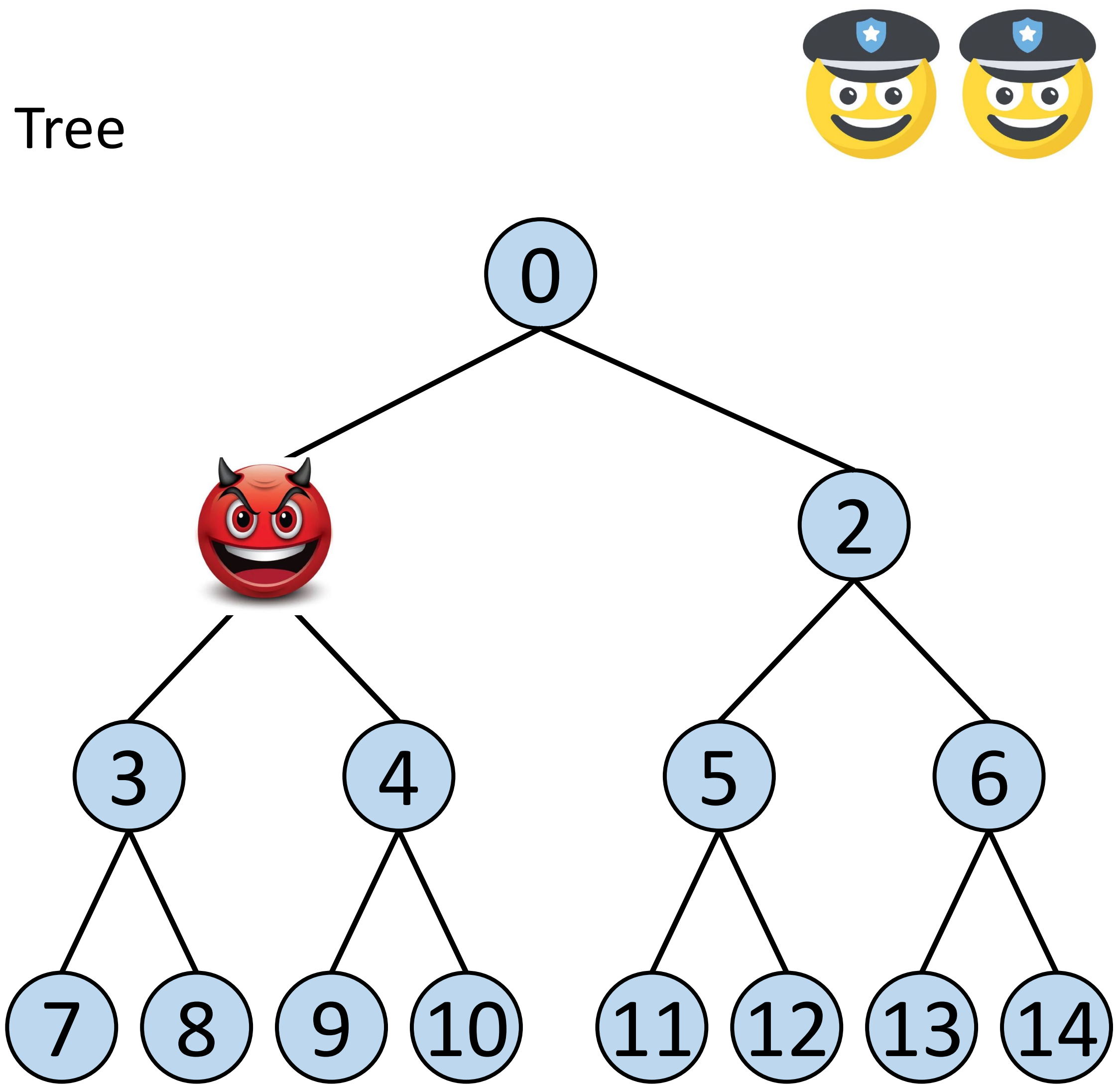
(3) Then the cop lands.



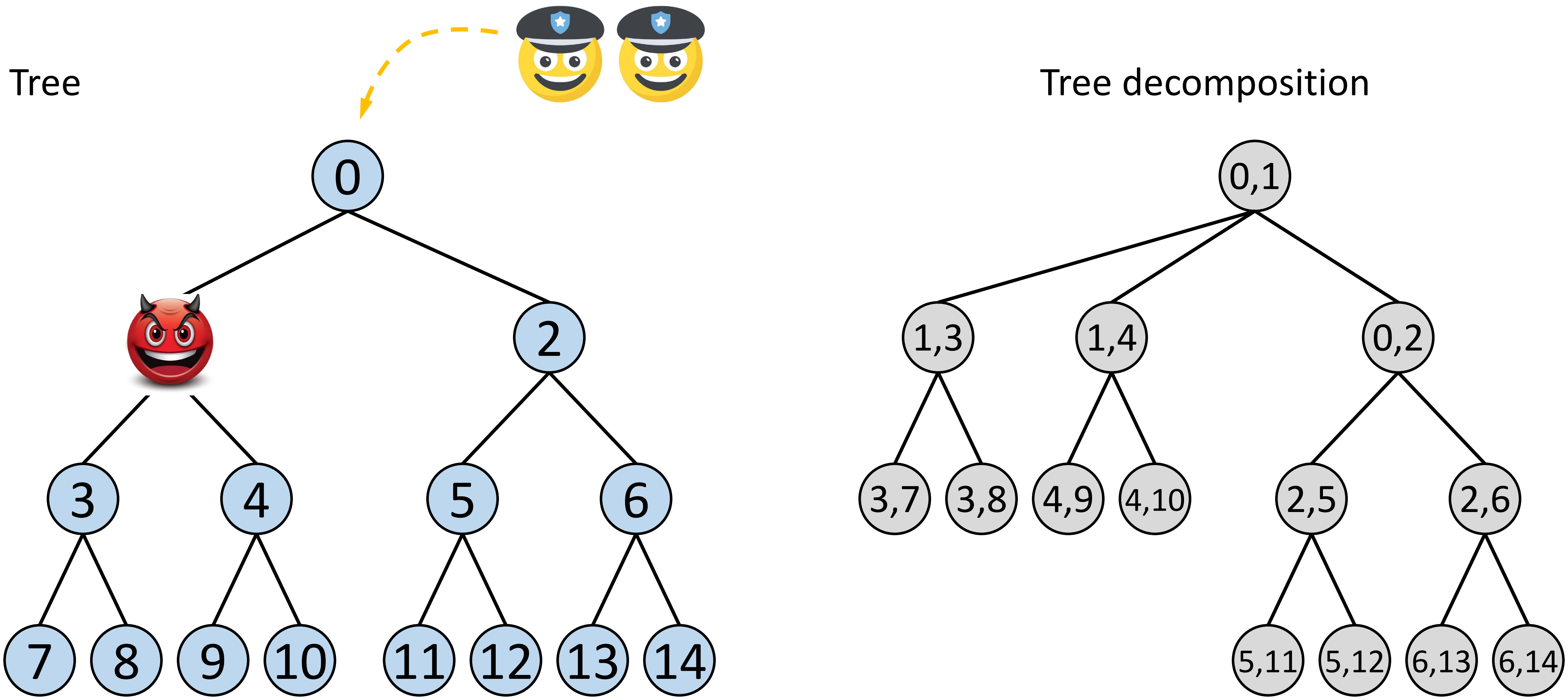
Seymour, Thomas. Graph searching and a min-max theorem for tree-width, Journal of Combinatorial Theory, Series B, 1993. https://doi.org/10.1006/jctb.1993.1027

Robbers cannot escape on trees with 2 cops

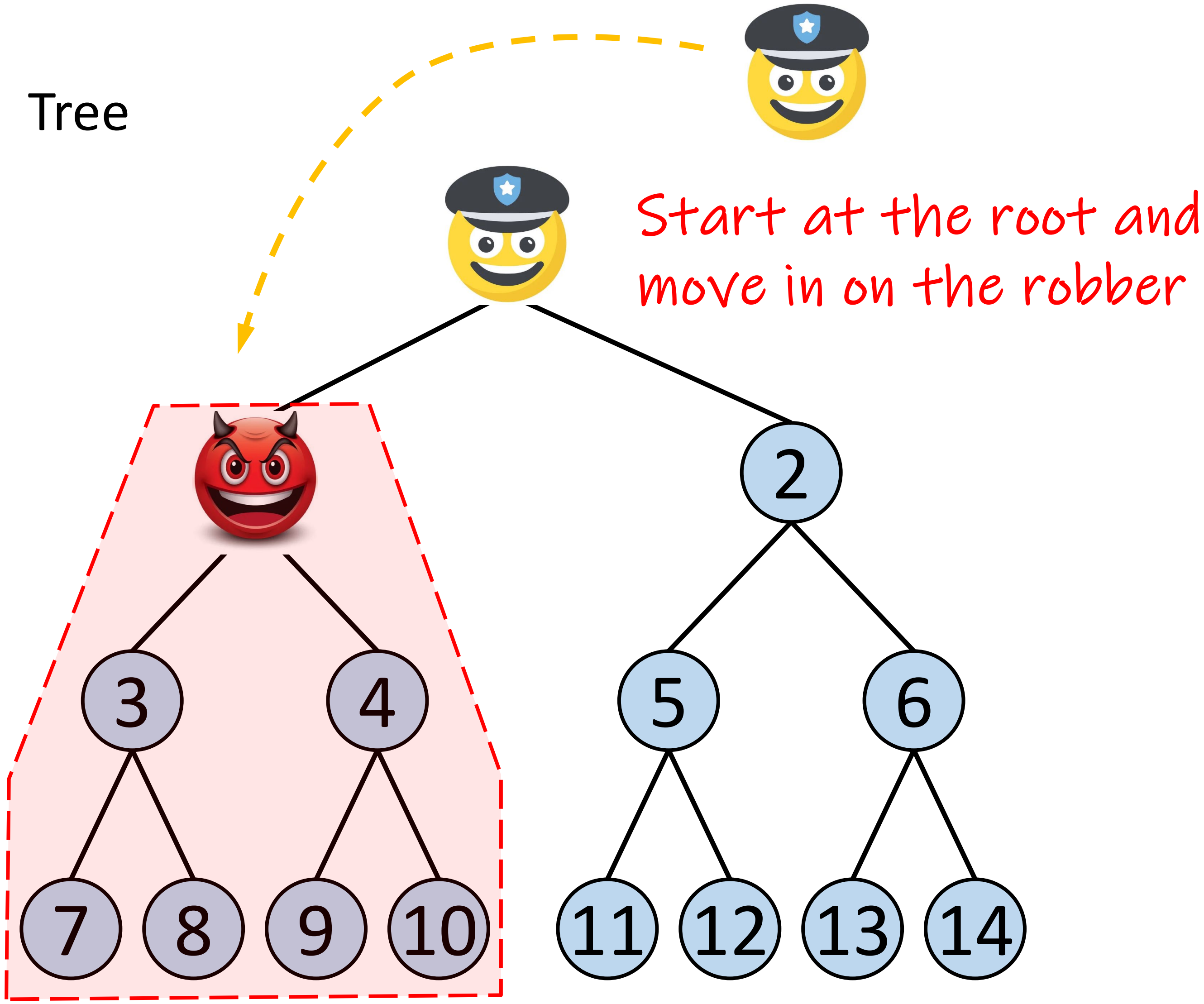
Tree

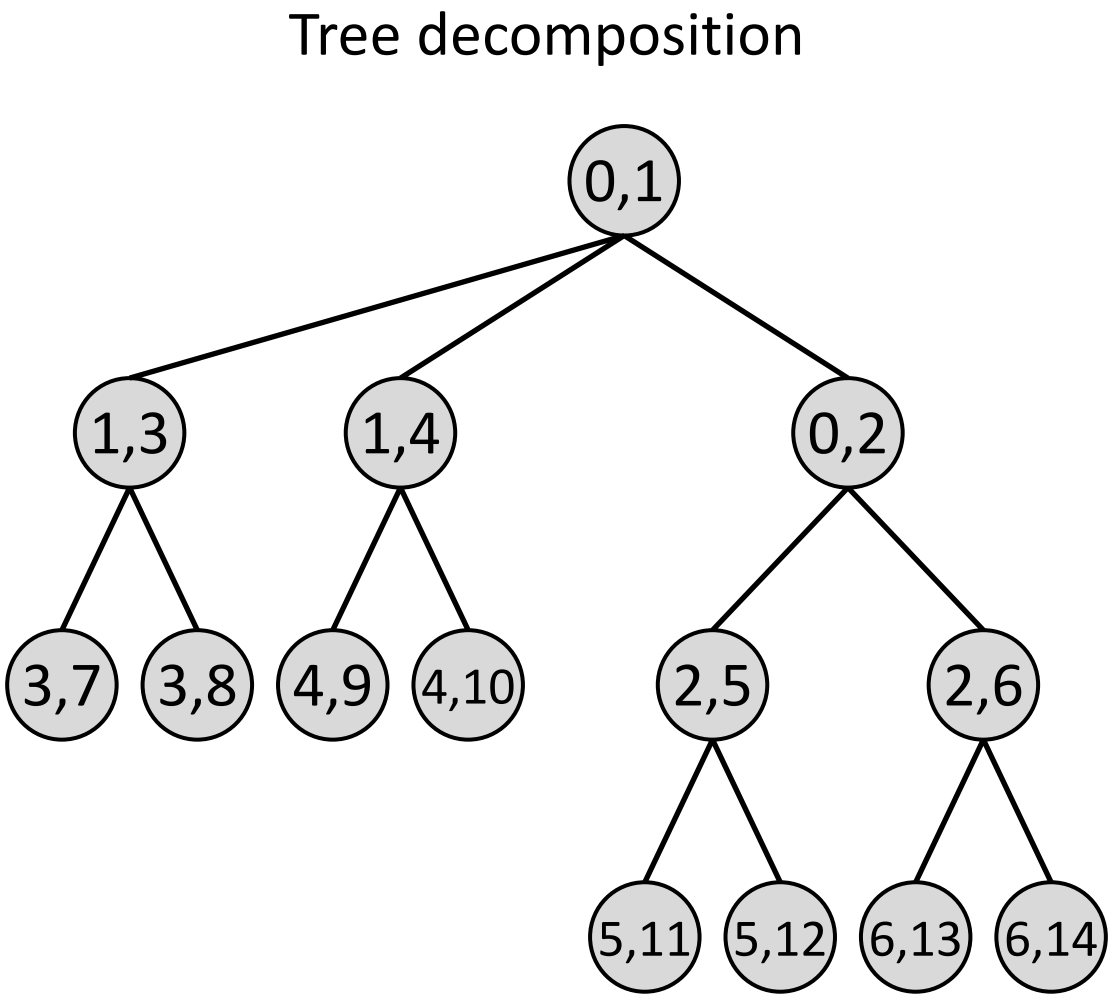


Robbers cannot escape on trees with 2 cops



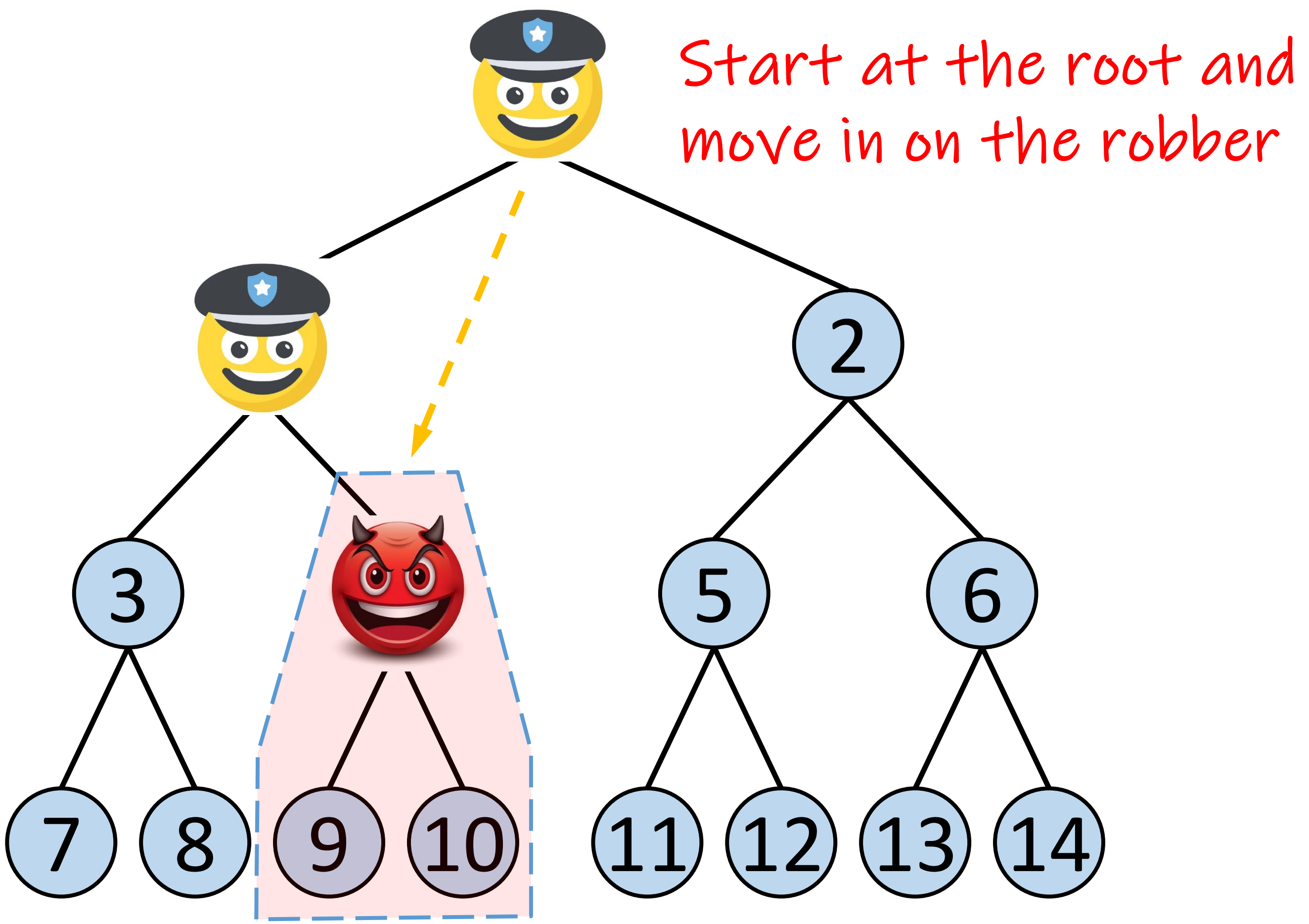
Robbers cannot escape on trees with 2 cops



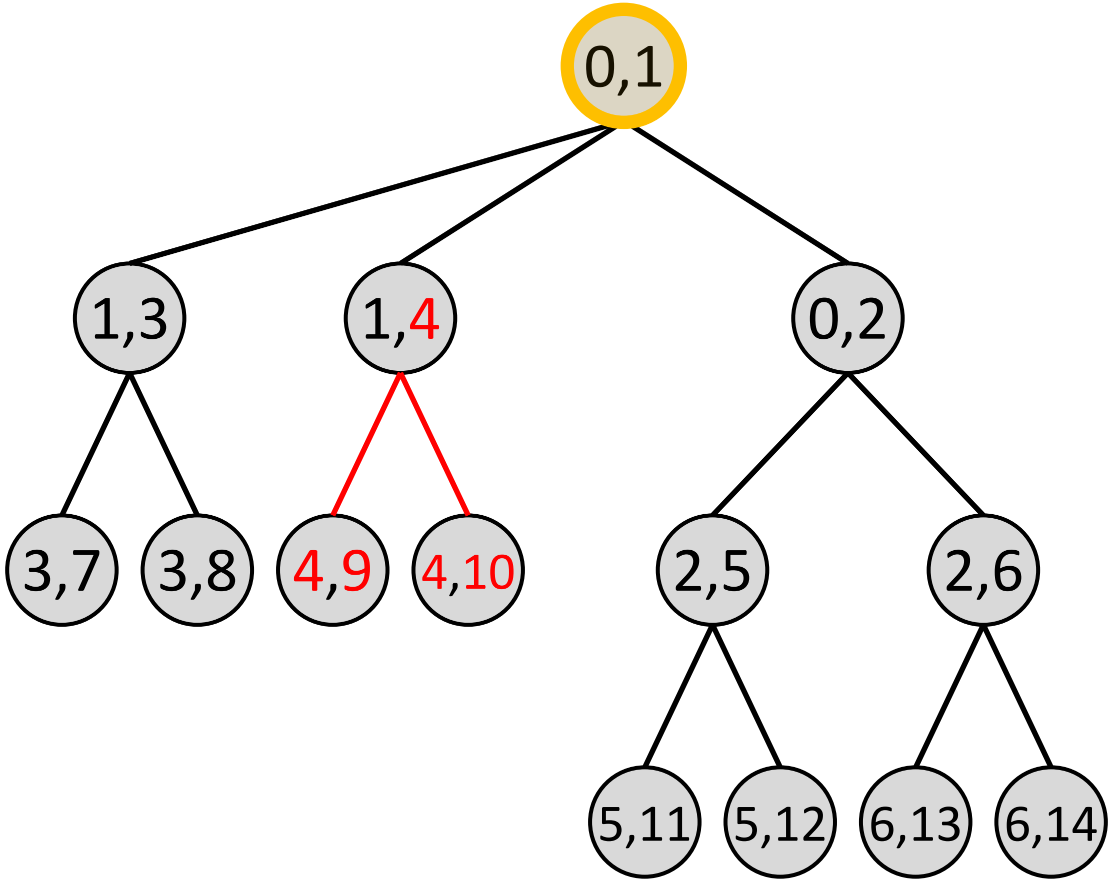


Robbers cannot escape on trees with 2 cops

Tree

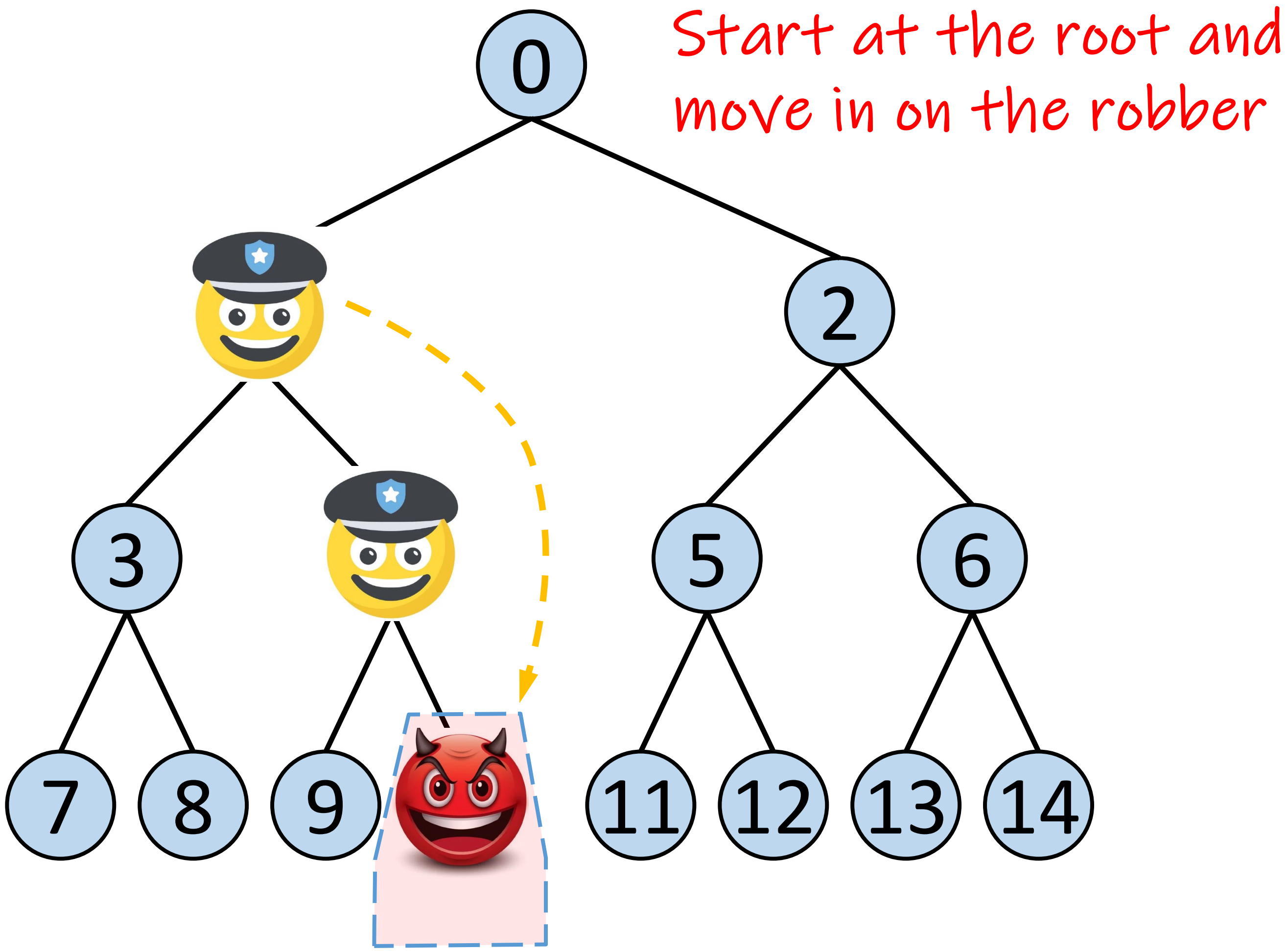


Tree decomposition

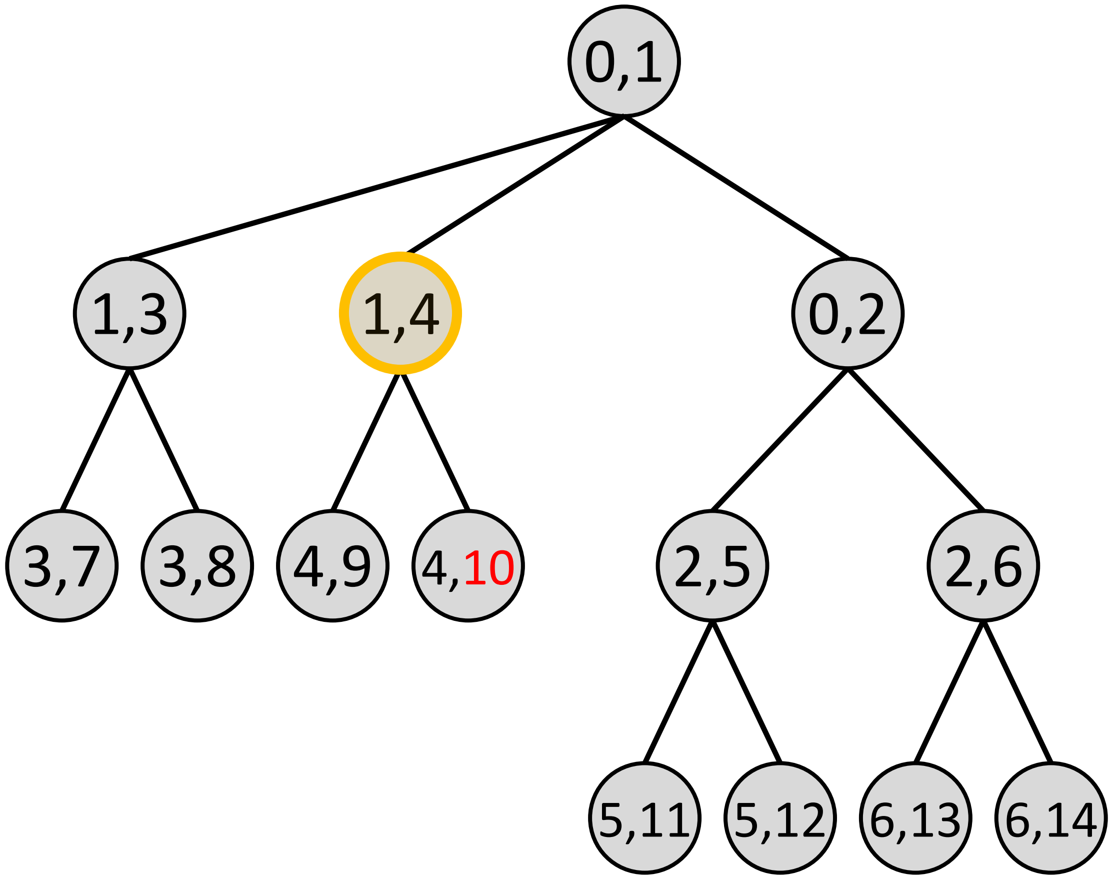


Robbers cannot escape on trees with 2 cops

Tree



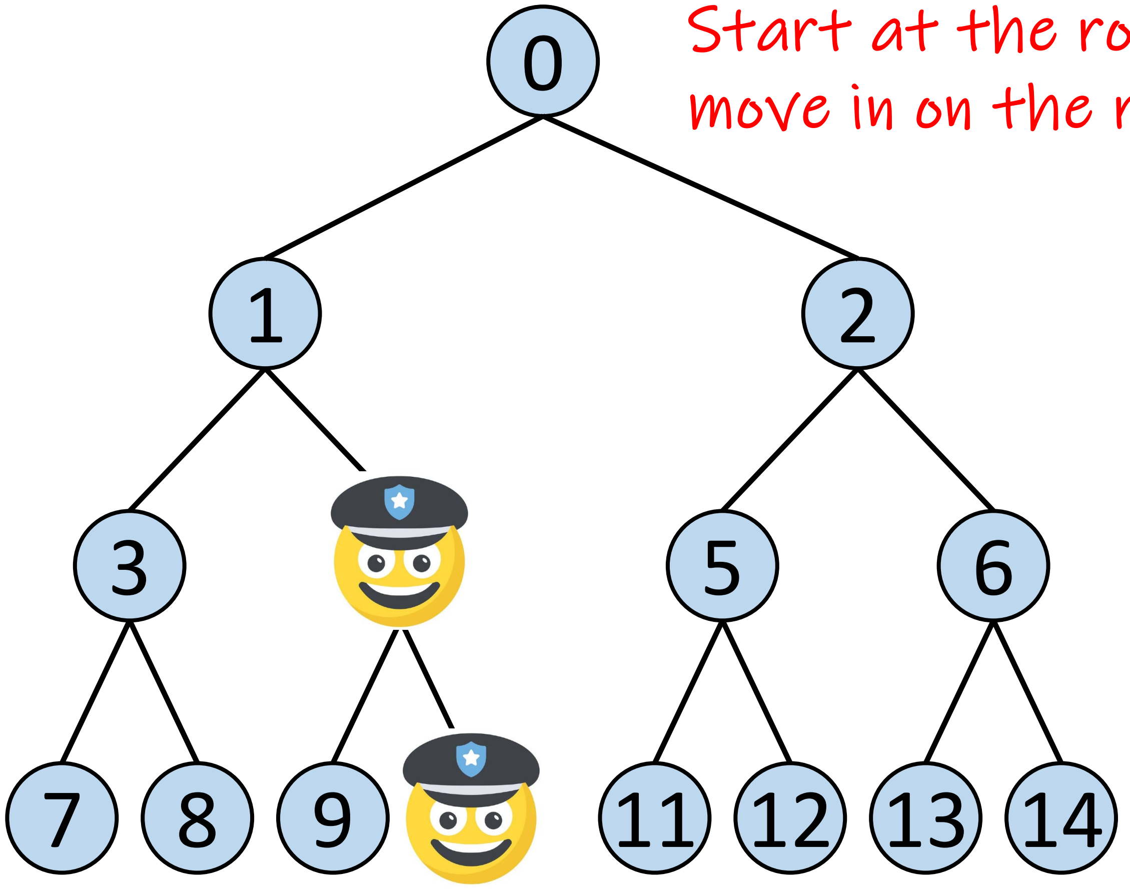
Tree decomposition



Robbers cannot escape on trees with 2 cops

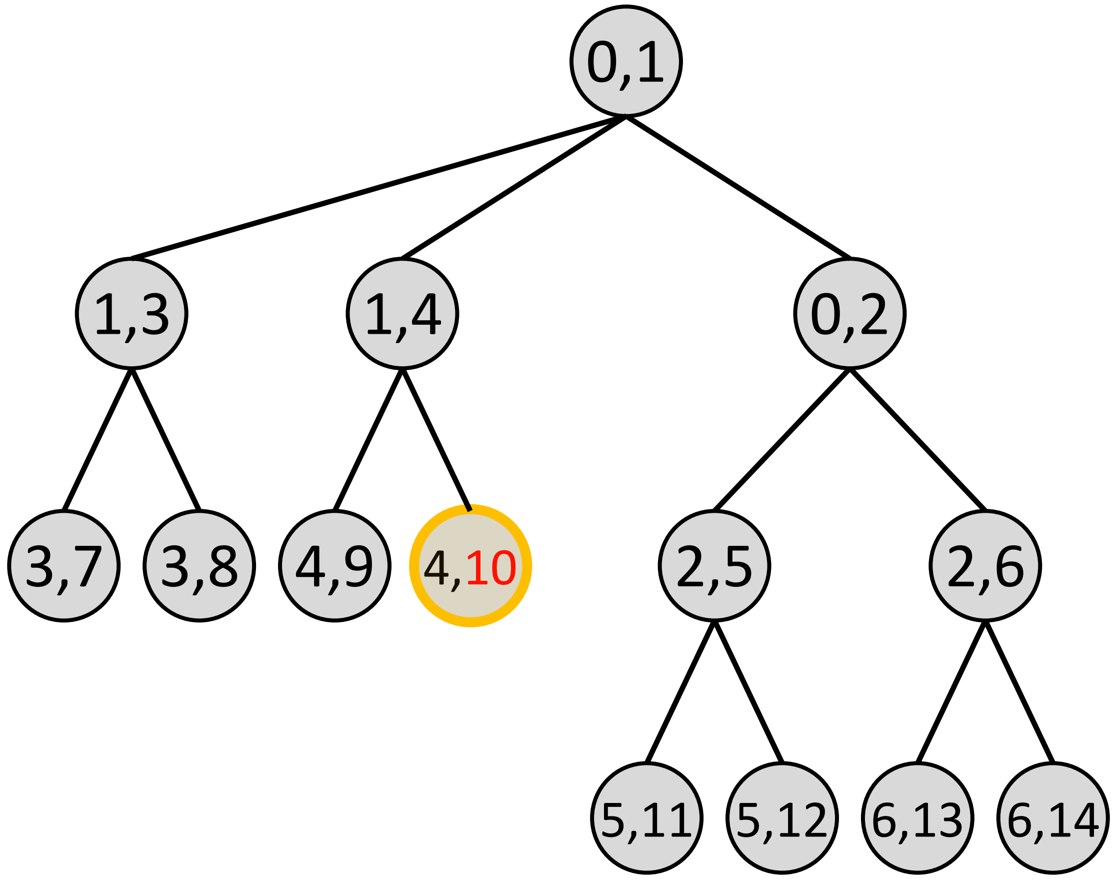
Tree

Start at the root and move in on the robber



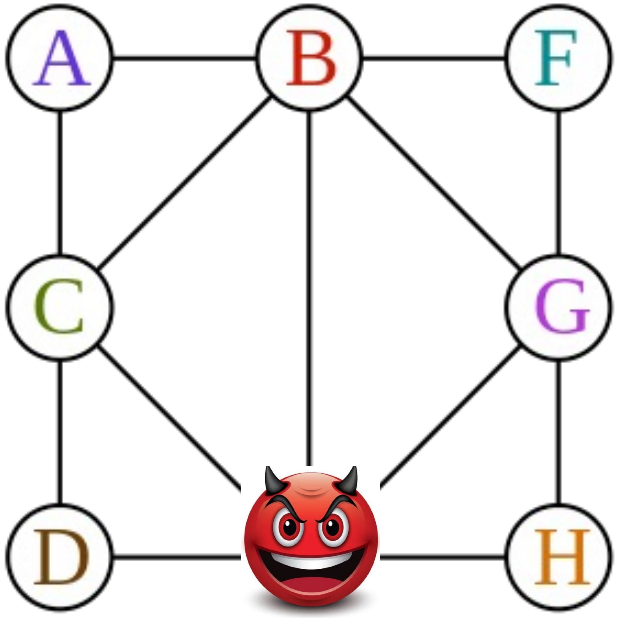
got him!

Tree decomposition



Robbers cannot hide from k=3 cops on graph with treewidth=2

Graph with treewidth = 2

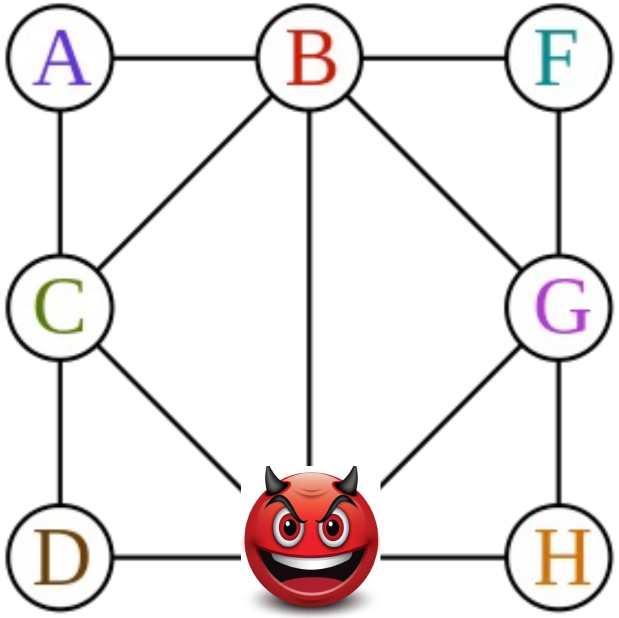


Graph and tree decomposition from: https://en.wikipedia.org/wiki/Tree_decomposition
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

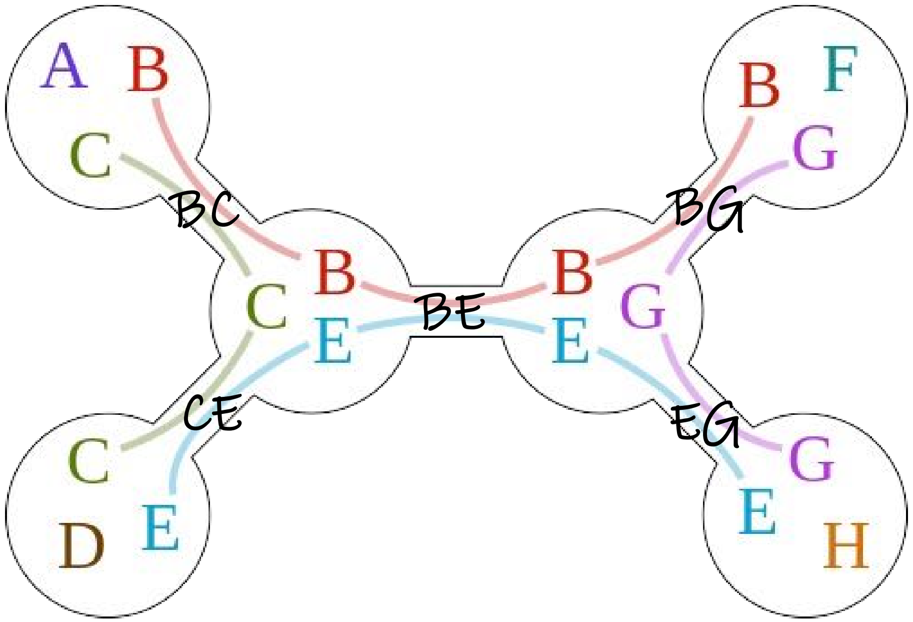
Robbers cannot hide from k=3 cops on graph with treewidth=2

Graph with treewidth = 2

You will need 3 cops



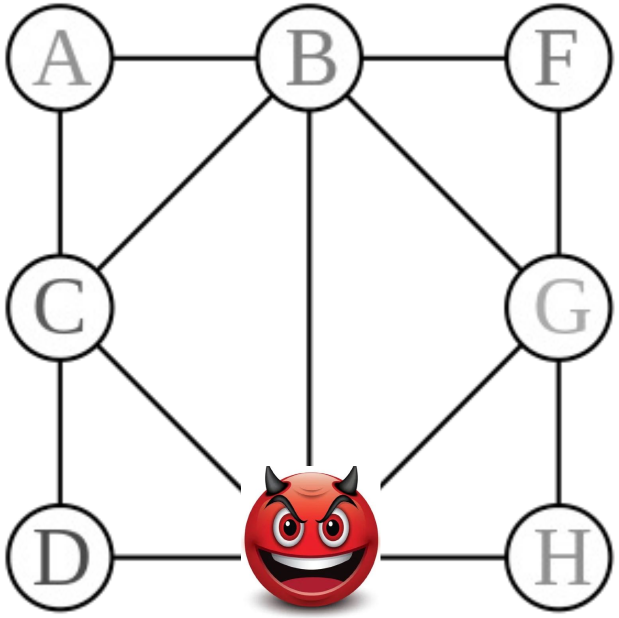
Tree decomposition



Robbers cannot hide from k=3 cops on graph with treewidth=2

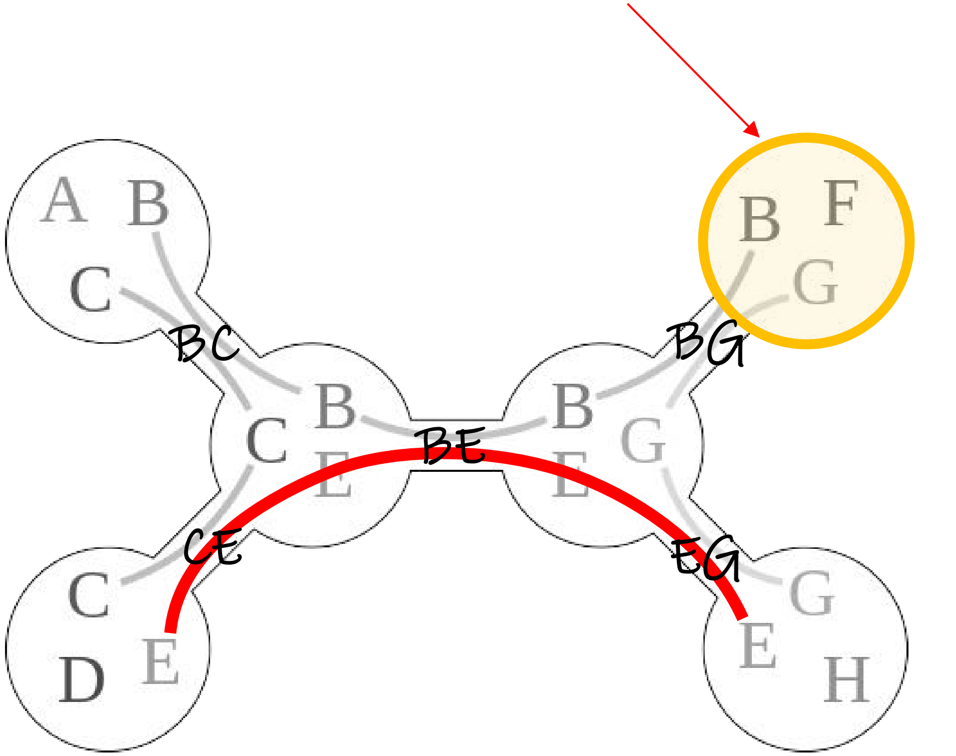
Graph with treewidth = 2

You will need 3 cops



Tree decomposition

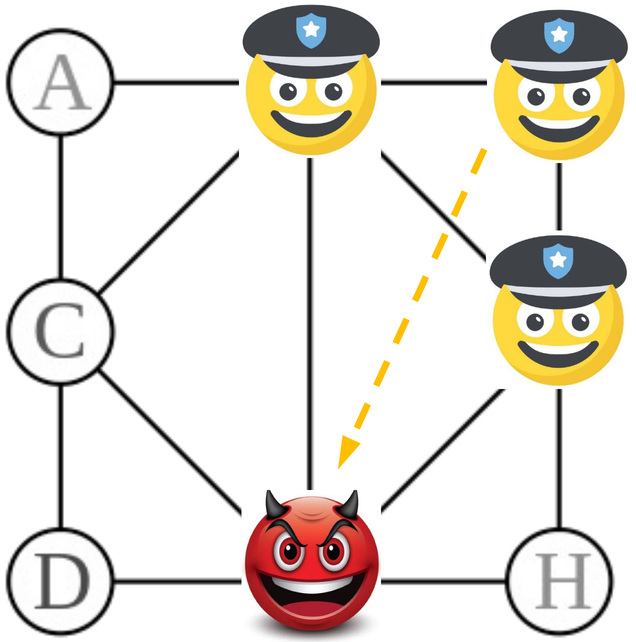
Pick some root



Robbers cannot hide from k=3 cops on graph with treewidth=2

Graph with treewidth = 2

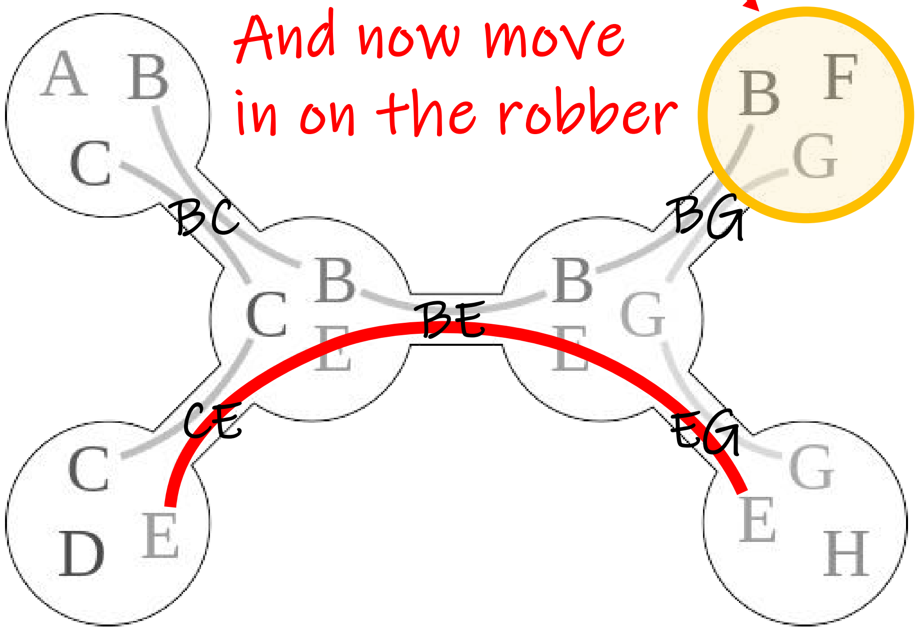
You will need 3 cops



Tree decomposition

Pick some root

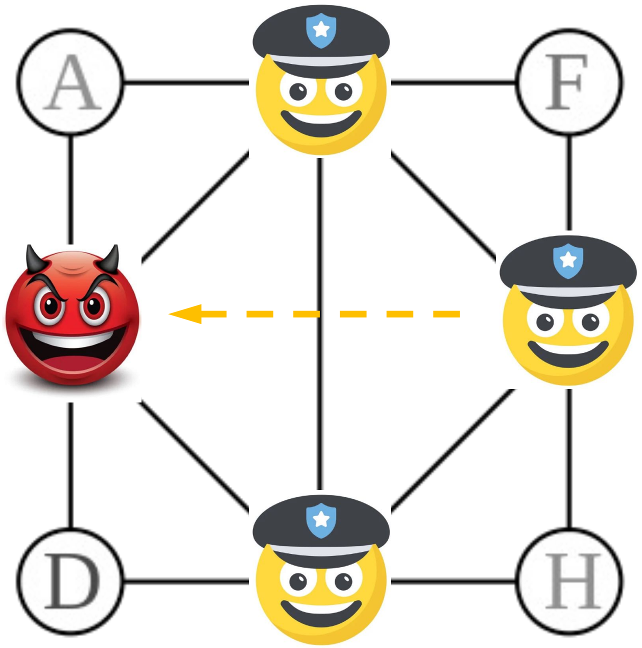
And now move in on the robber



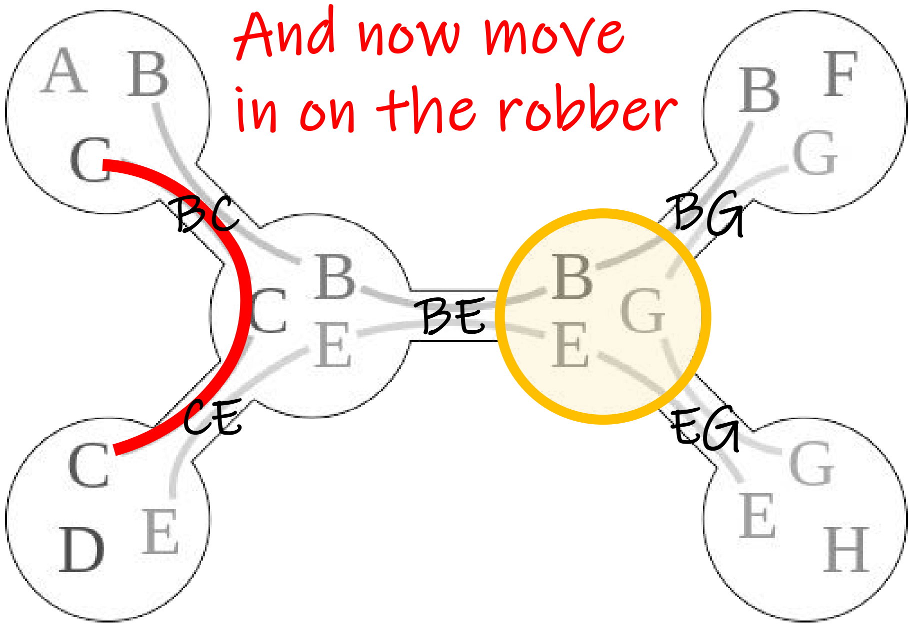
Robbers cannot hide from k=3 cops on graph with treewidth=2

Graph with treewidth = 2

You will need 3 cops



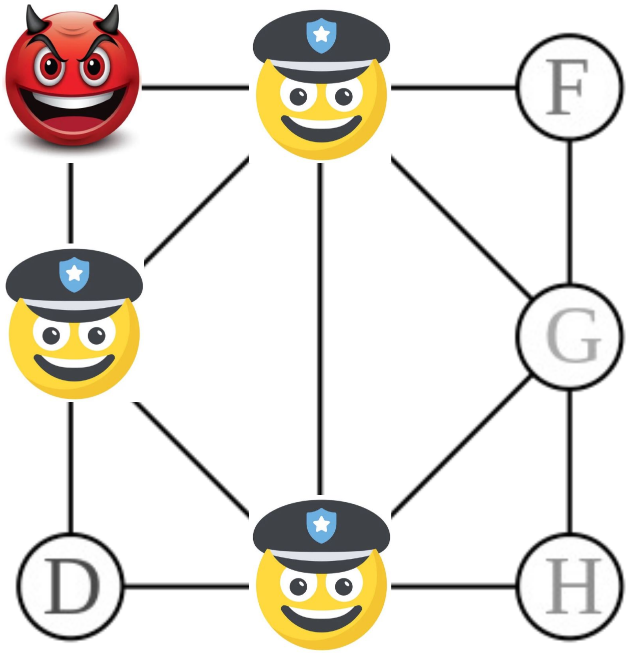
Tree decomposition



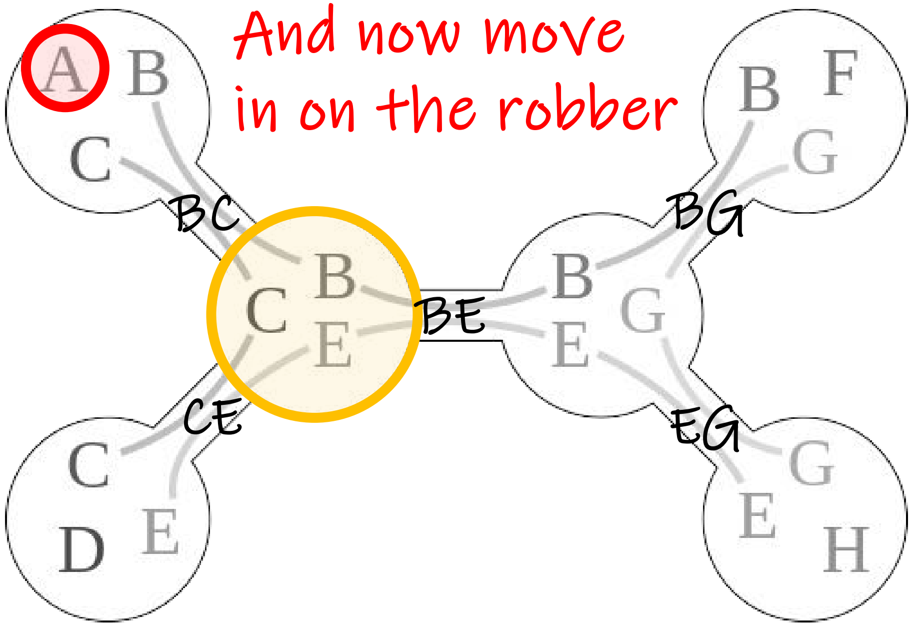
Robbers cannot hide from k=3 cops on graph with treewidth=2

Graph with treewidth = 2

You will need 3 cops



Tree decomposition



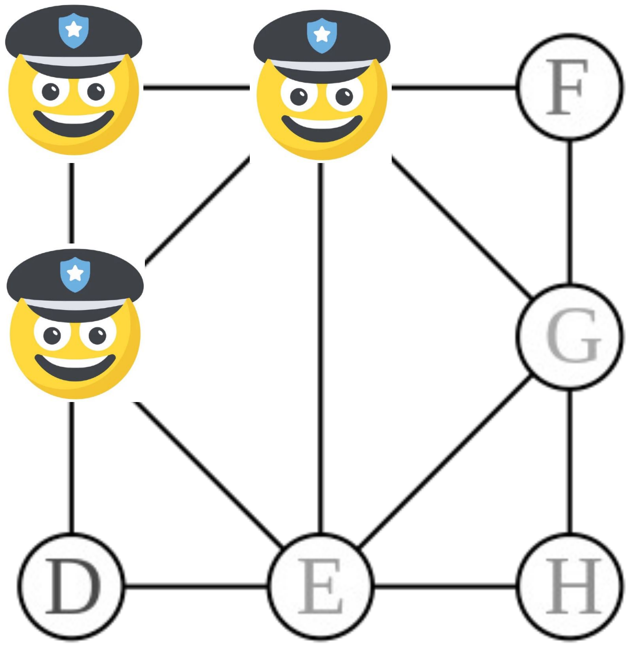
Graph and tree decomposition from: https://en.wikipedia.org/wiki/Tree_decomposition

Robbers cannot hide from k=3 cops on graph with treewidth=2

Graph with treewidth = 2

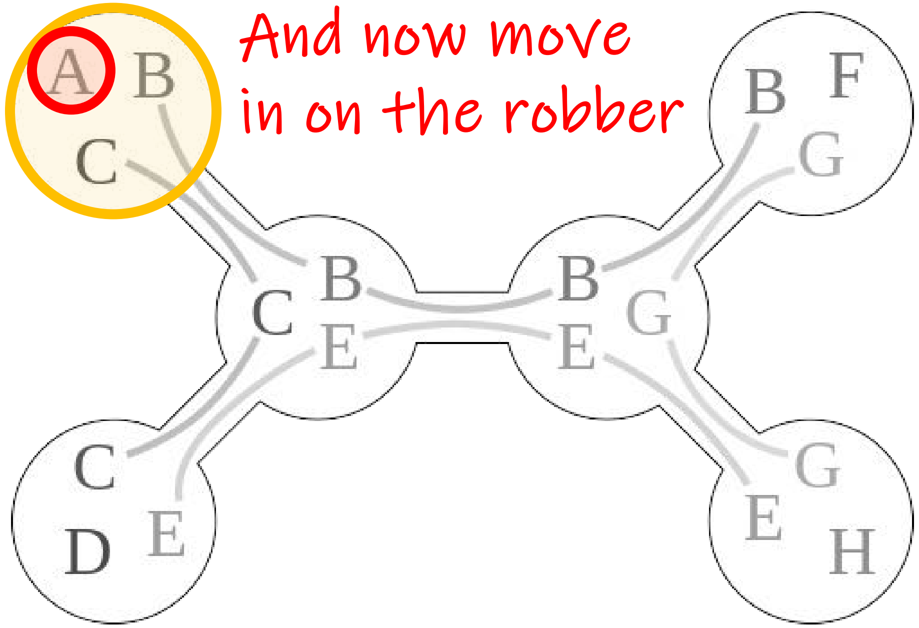
You will need 3 cops

got him!



Tree decomposition

And now move in on the robber

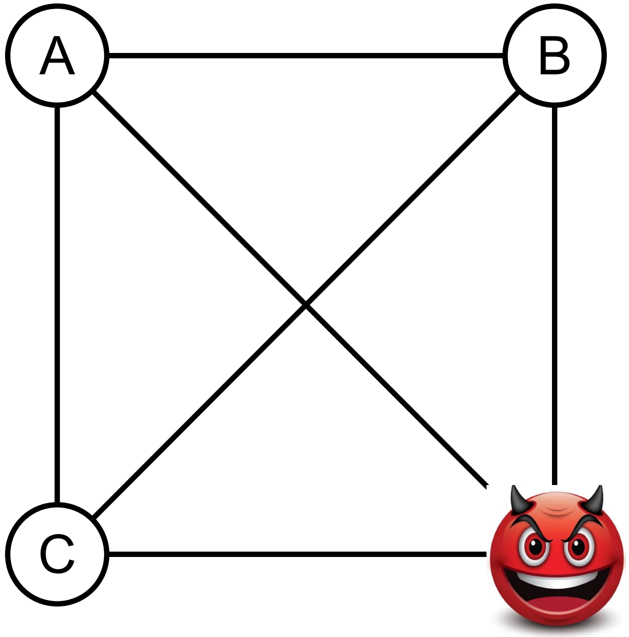


Graph and tree decomposition from: https://en.wikipedia.org/wiki/Tree_decomposition
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Robbers cannot hide from k=? cops on 4-cliques?

4-clique

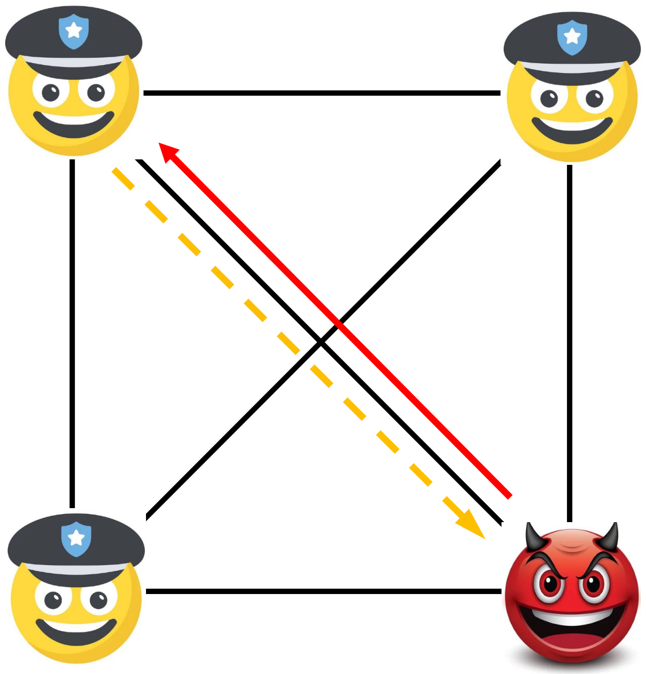
How many cops do we need?



Robbers cannot hide from k=3 cops on graph with treewidth=2

4-clique

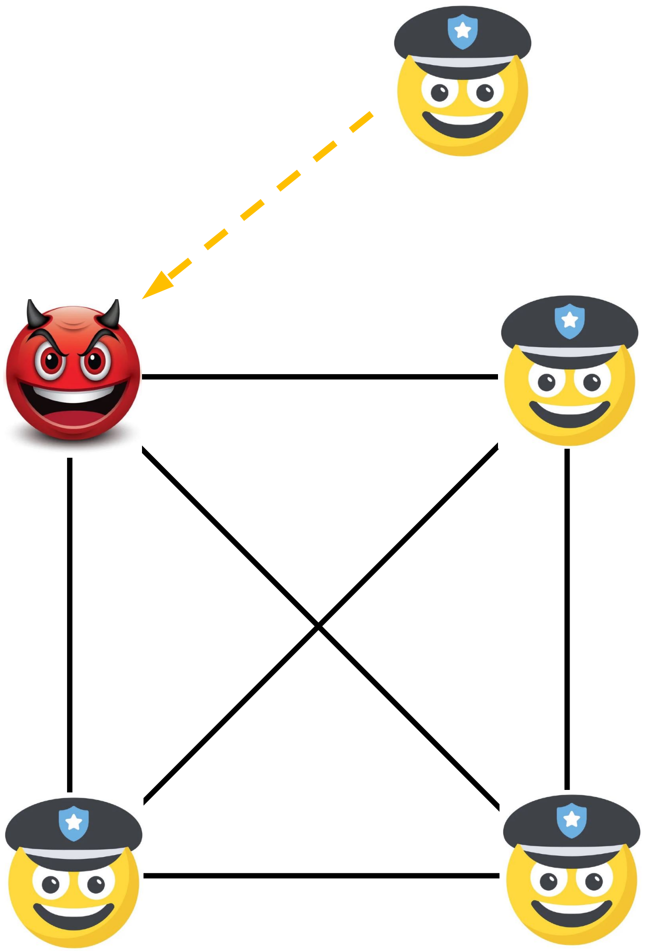
Let's try with 3 cops as before



Robbers cannot hide from k=3 cops on graph with treewidth=2

4-clique

We need 4 cops!



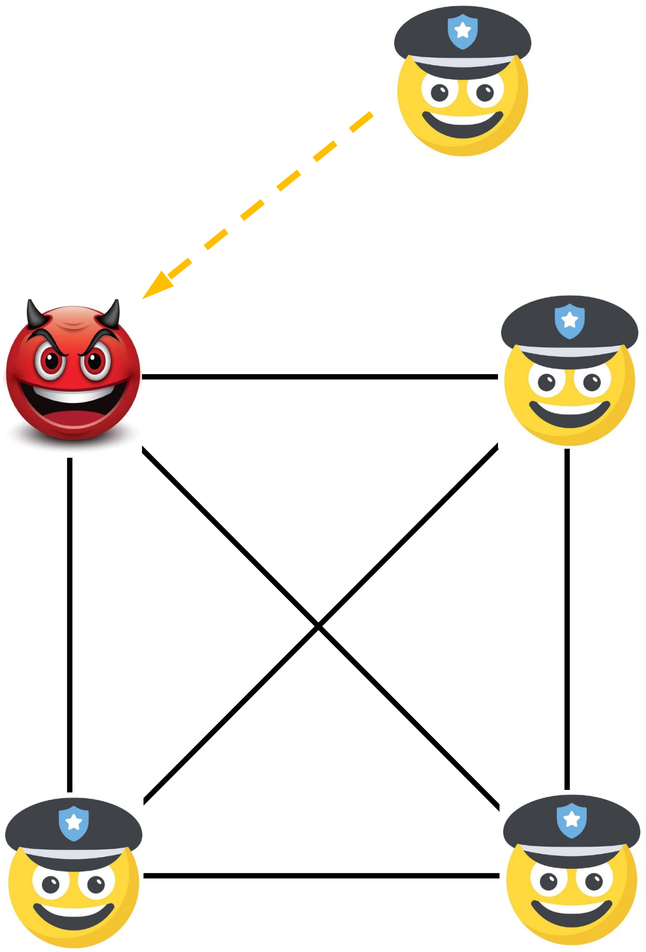
Tree decomposition

?

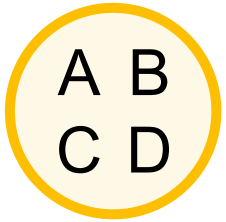
Robbers cannot hide from k=3 cops on graph with treewidth=2

4-clique

We need 4 cops



Tree decomposition



We need treewidth + 1 cops!

Updated 4/9/2024

Topic 3: Efficient query evaluation
Unit 2: Cyclic query evaluation
Lecture 23

Wolfgang Gatterbauer

CS7240 Principles of scalable data management (sp24)

https://northeastern-datalab.github.io/cs7240/sp24/

4/9/2024

Pre-class conversations

- Last class summary
- Project: (P3: today FRI, 3/31)
- Scribes: half through
- Guest speaker on deep theory of set covering this THU 10am
- 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

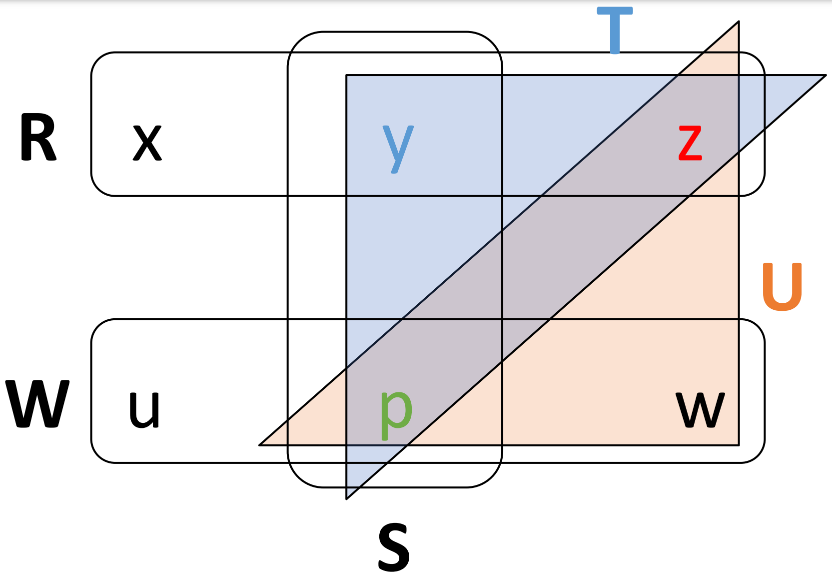
Outline: T3-2: Cyclic conjunctive queries

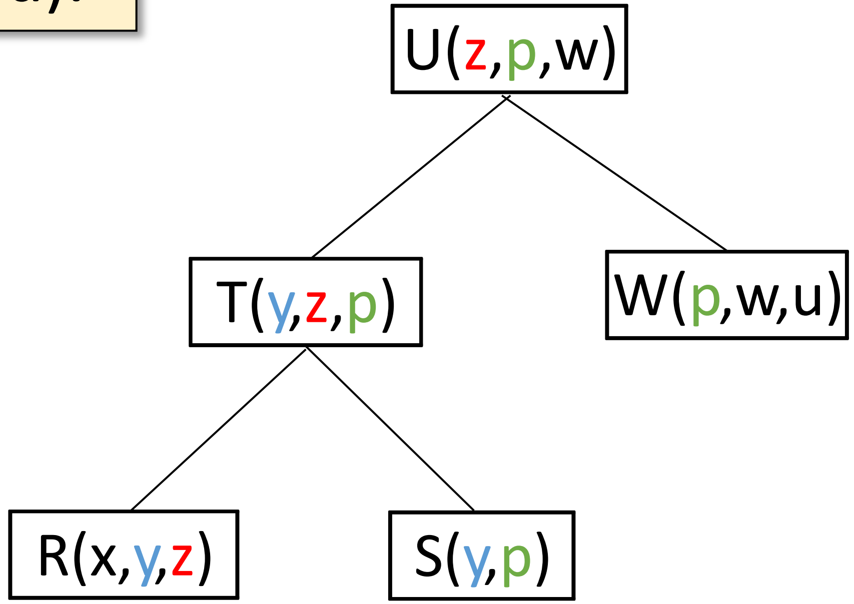
- T3-1: Acyclic conjunctive queries
- T3-2: Cyclic conjunctive queries
  - 2SAT (a detour)
  - Tree decompositions
  - Hypertrees decompositions
  - Duality in Linear programming (a not so quick primer)
  - AGM bound (maximal result size for full CQs) and Worst-case optimal joins for the triangle query
  - Worst-case optimal joins & the 4-cycle
  - Optimal joins & the 4-cycle

Alpha-Acyclic Conjunctive Queries

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

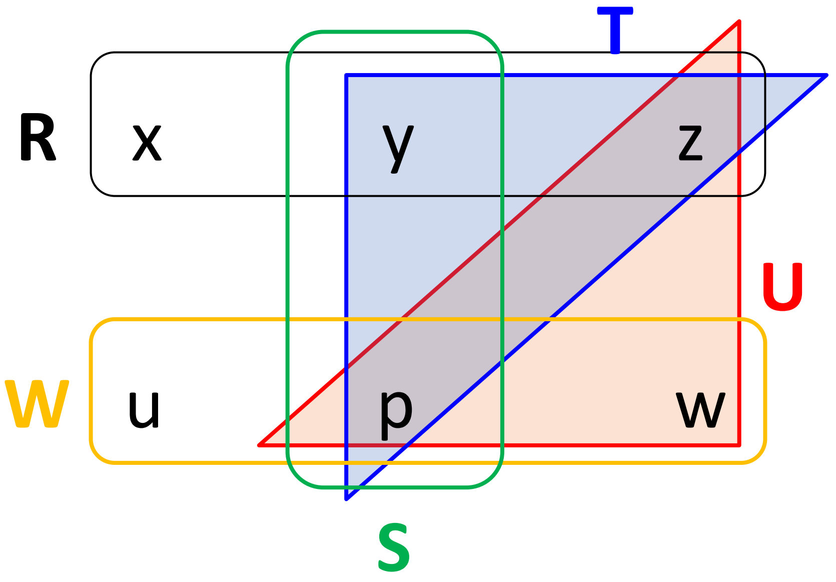
Q :- R(x,y,z), S(y,p), T(y,z,p), U(z,p,w), W(p,w,u).

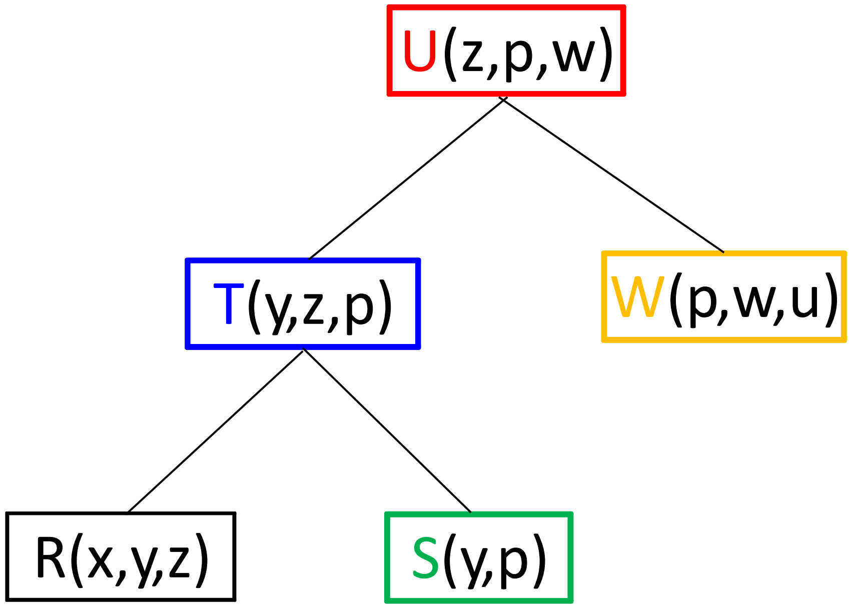




Alpha-Acyclic Conjunctive Queries

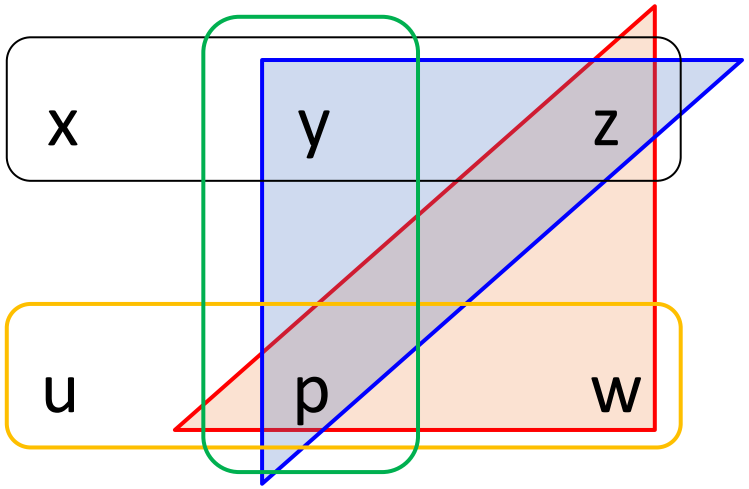
- A join tree for a hypergraph H=(V,E) is a labeled tree T =(N,F,λ) such that:
  - The nodes of T are formed by the hyperedges. In other words, λ: N→E s.t. for each hyperedge e ∈ E of H, there exists n ∈ N such that e = λ(n)
  - For each node u ∈ V of H, the set {n ∈ N | u ∈ λ(n)} induces a connected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

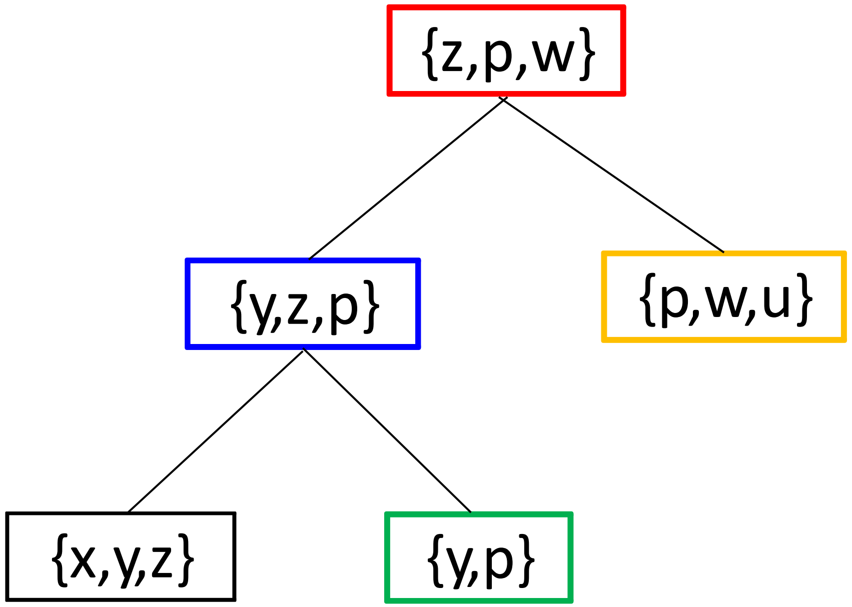




Alpha-Acyclic Conjunctive Queries

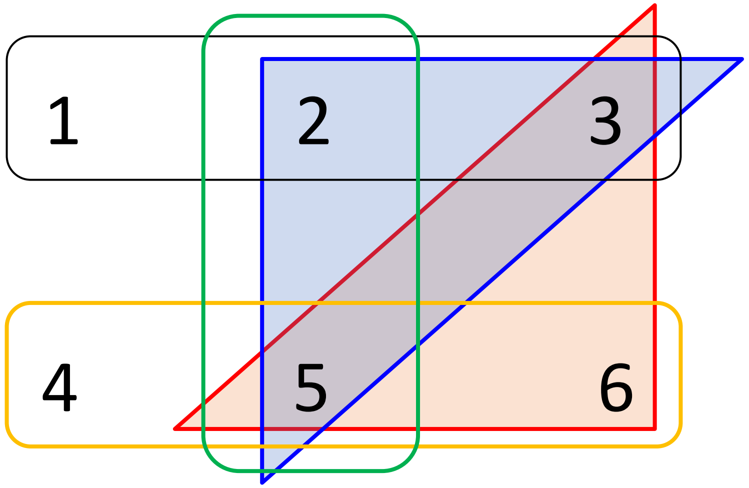
- A join tree for a hypergraph H=(V,E) is a labeled tree T =(N,F,λ) such that:
  - The nodes of T are formed by the hyperedges. In other words, λ: N→E s.t. for each hyperedge e ∈ E of H, there exists n ∈ N such that e = λ(n)
  - For each node u ∈ V of H, the set {n ∈ N | u ∈ λ(n)} induces a connected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

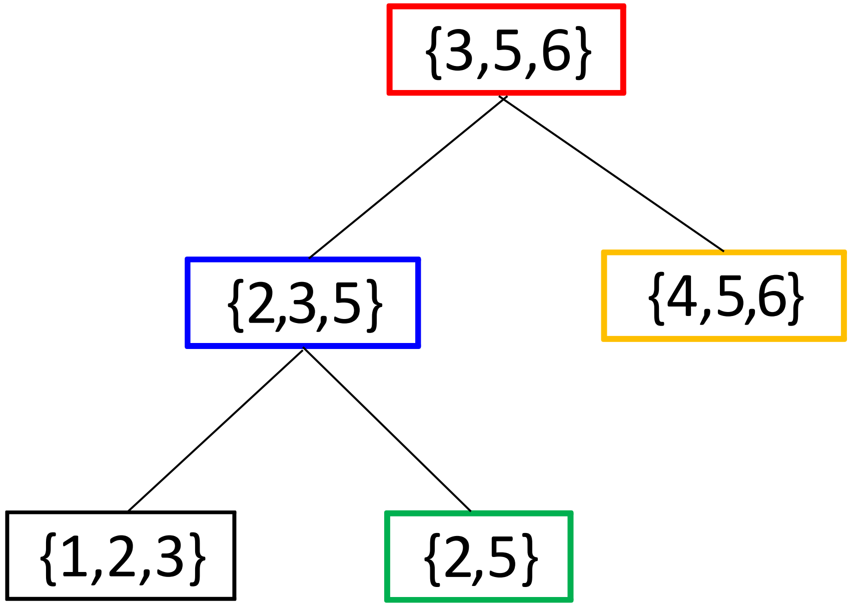




Alpha-Acyclic Conjunctive Queries

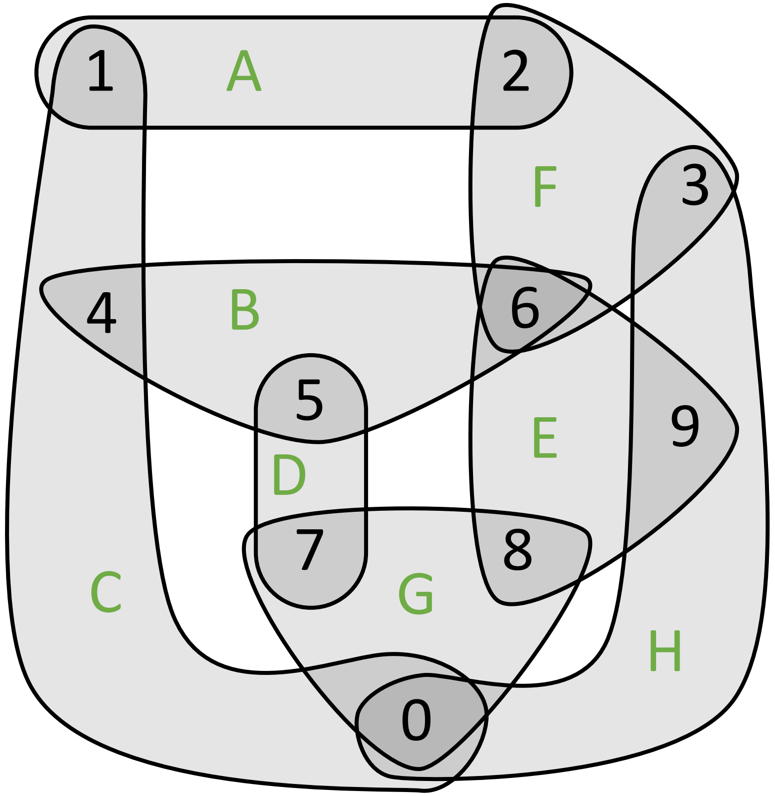
- A join tree for a hypergraph H=(V,E) is a labeled tree T =(N,F,λ) such that:
  - The nodes of T are formed by the hyperedges. In other words, λ: N→E s.t. for each hyperedge e ∈ E of H, there exists n ∈ N such that e = λ(n)
  - For each node u ∈ V of H, the set {n ∈ N | u ∈ λ(n)} induces a connected 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) is still a key structural criterion for efficiency!
- But treewidth does not generalize the notion of hypergraph acyclicity (because acyclic families of hypergraphs may have unbounded treewidth: think of a single relation of high arity ☹)
- What will help is the number of atoms needed to cover sets of variables ☺.
- 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

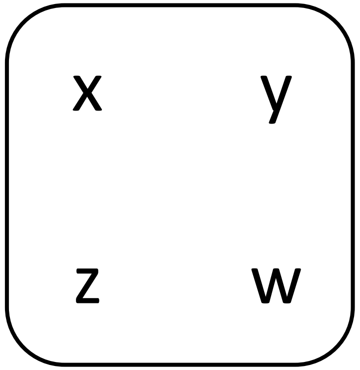
Treewidth based on graphs.

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: ?

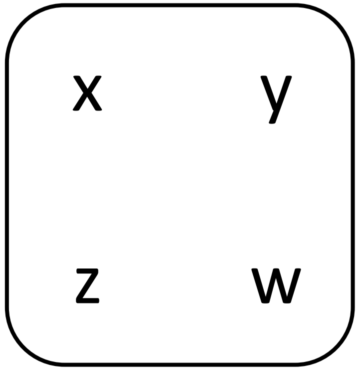
Issues with standard Treewidth (TW) for CQs

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

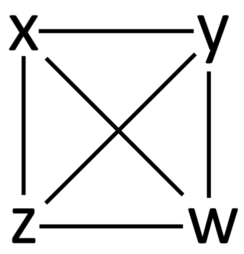
a clique is a graph where where every vertex is connected to every other vertex

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

Hypergraph



Clique graph



Treewidth: ?

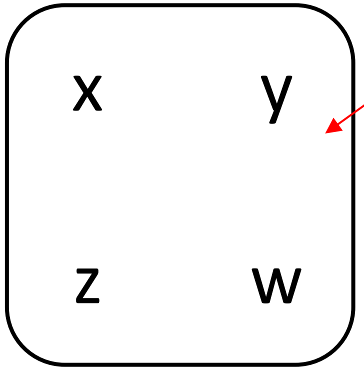
Issues with standard Treewidth (TW) for CQs

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

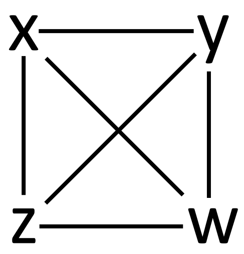
This is actually the best tree decomposition: Nodes of a clique need to appear in the same supernode

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

Hypertree



Clique graph



Treewidth: 3

Resulting complexity bound $O(m^4)$!

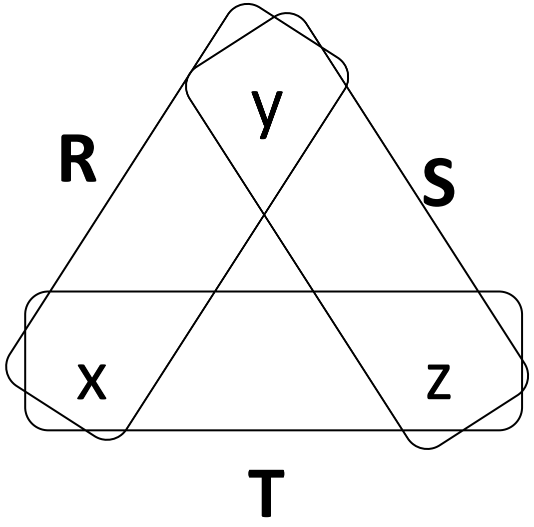
That's a pretty bad bound. We know we can evaluate this query in $O(n)$.

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 ☹

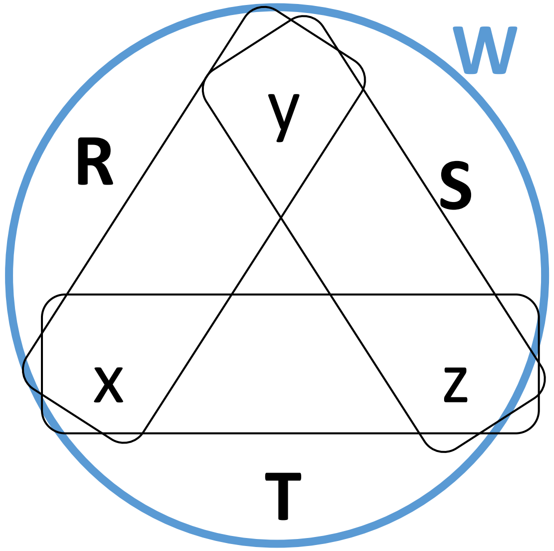
H_1 (Triangle)



Clique graph

?

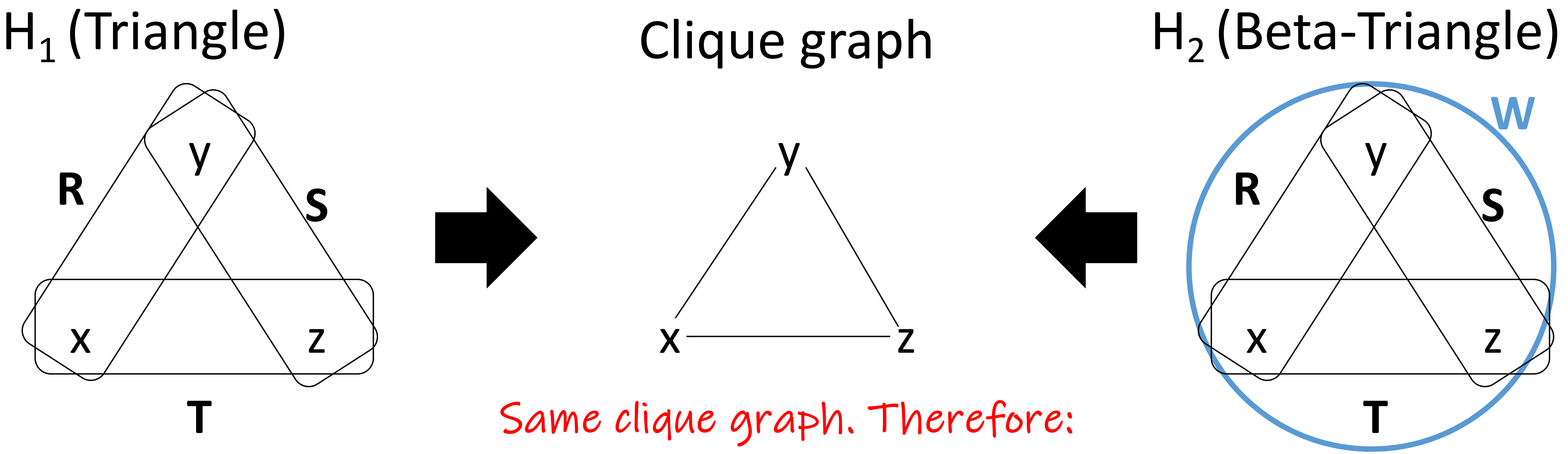
H_2 (Beta-Triangle)



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 ☹



Same clique graph. Therefore:
→ same TW 2.
→ same complexity bound $O(m^3)$

"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

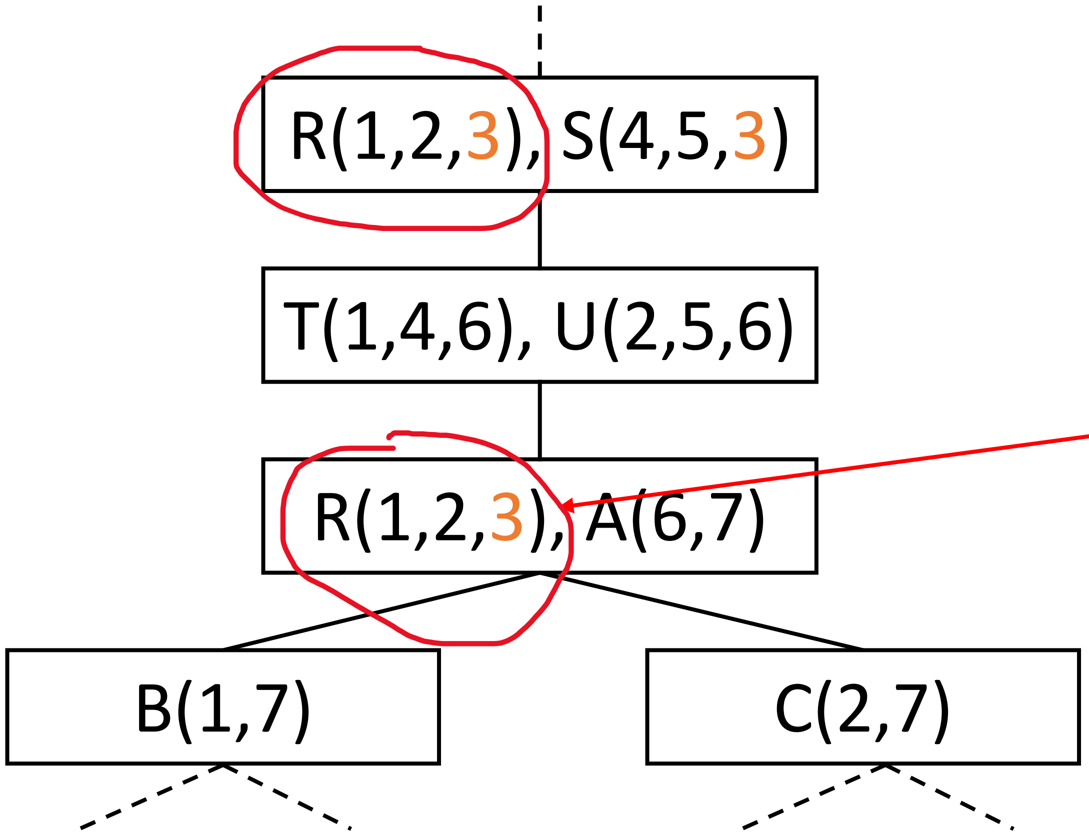
$$\{i \in I \mid A \in X(i)\} \cup \{i \in I \mid A \text{ appears in a subgoal } s \text{ such that } s \in X(i)\}$$

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.

Important Observation 1

Some decomposition



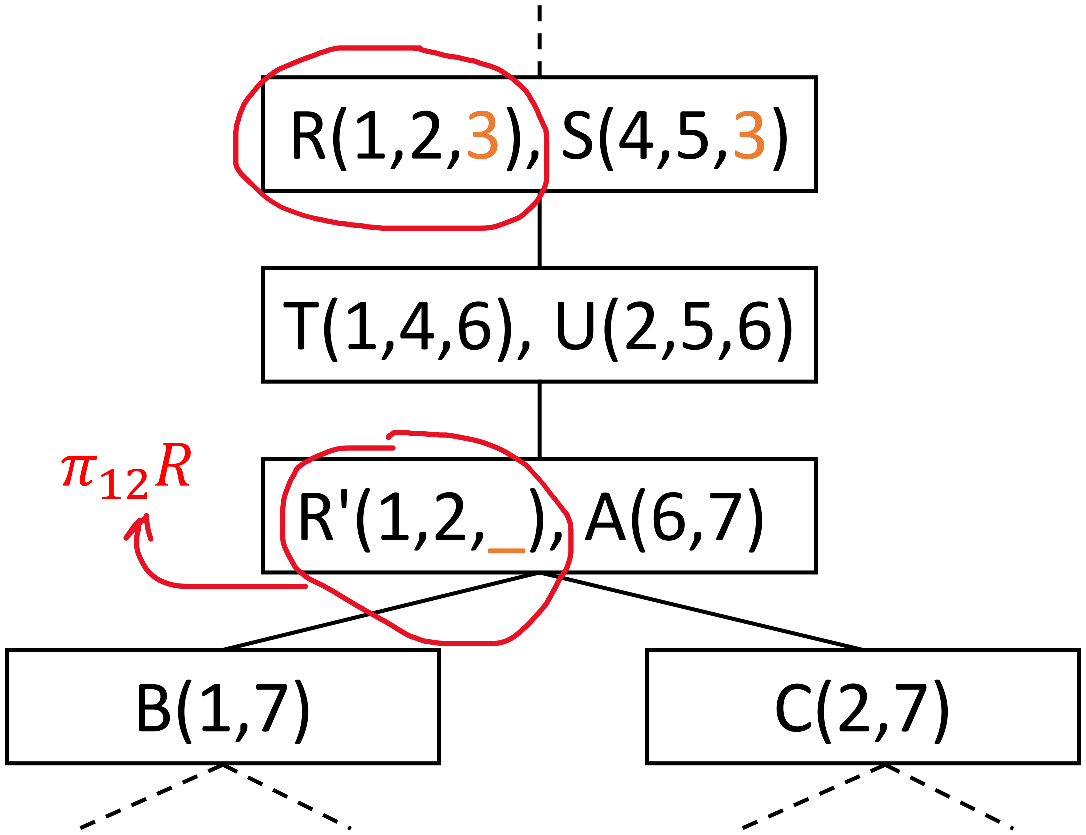
"Query decomposition" as defined by [Chekuri, Rajaraman'97] is too strict about atoms needing to be connected and atoms not allowing projections

This decomposition would not be possible for original "query decomposition" because "3" is not connected.

But what if you project "3" away onto
$R'(1,2) = \pi_{12}R(1,2,3)$

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.

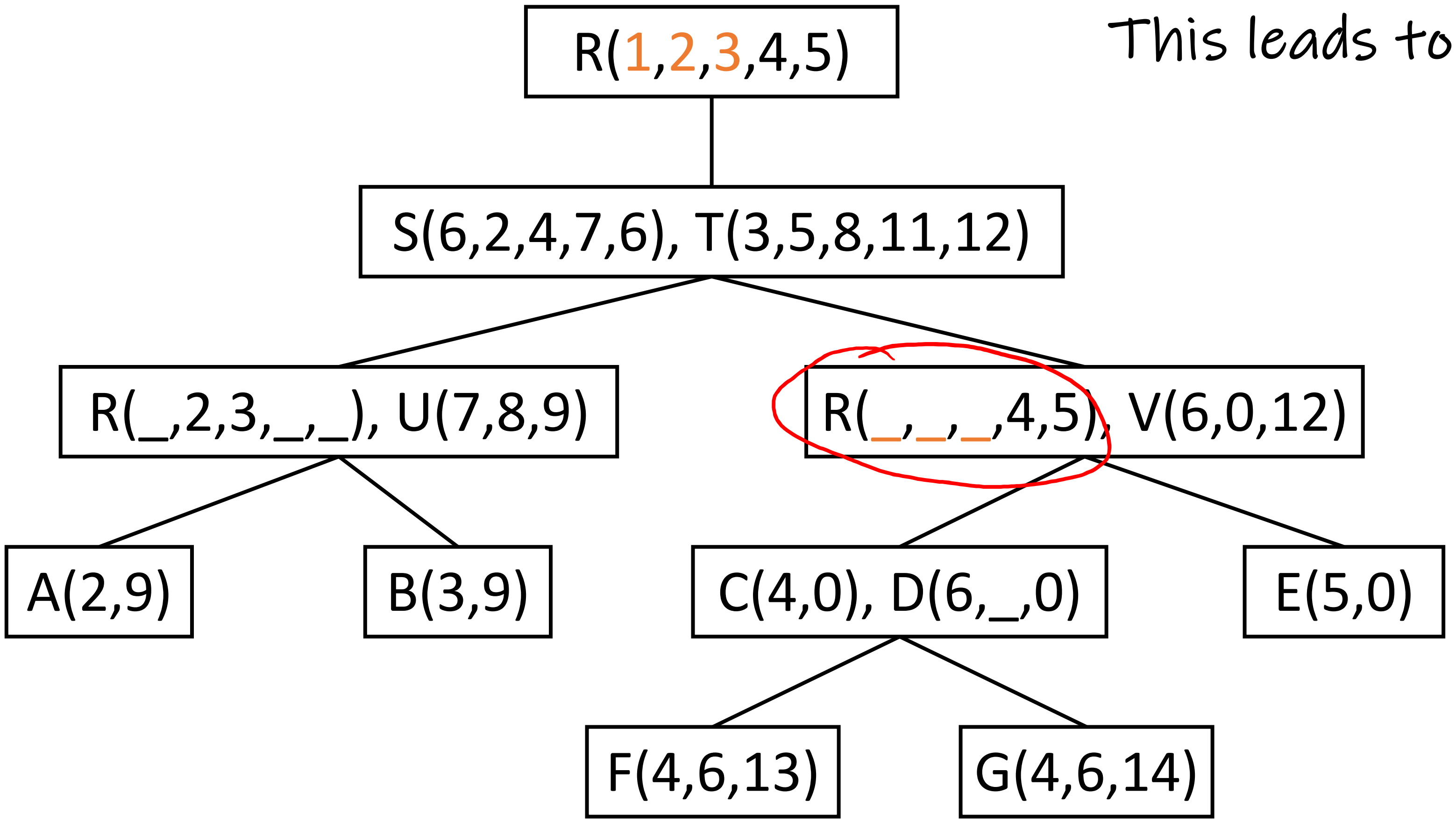
Idea: allow query atoms to be reused partially (with projections) as long as the full atom appears somewhere else.

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

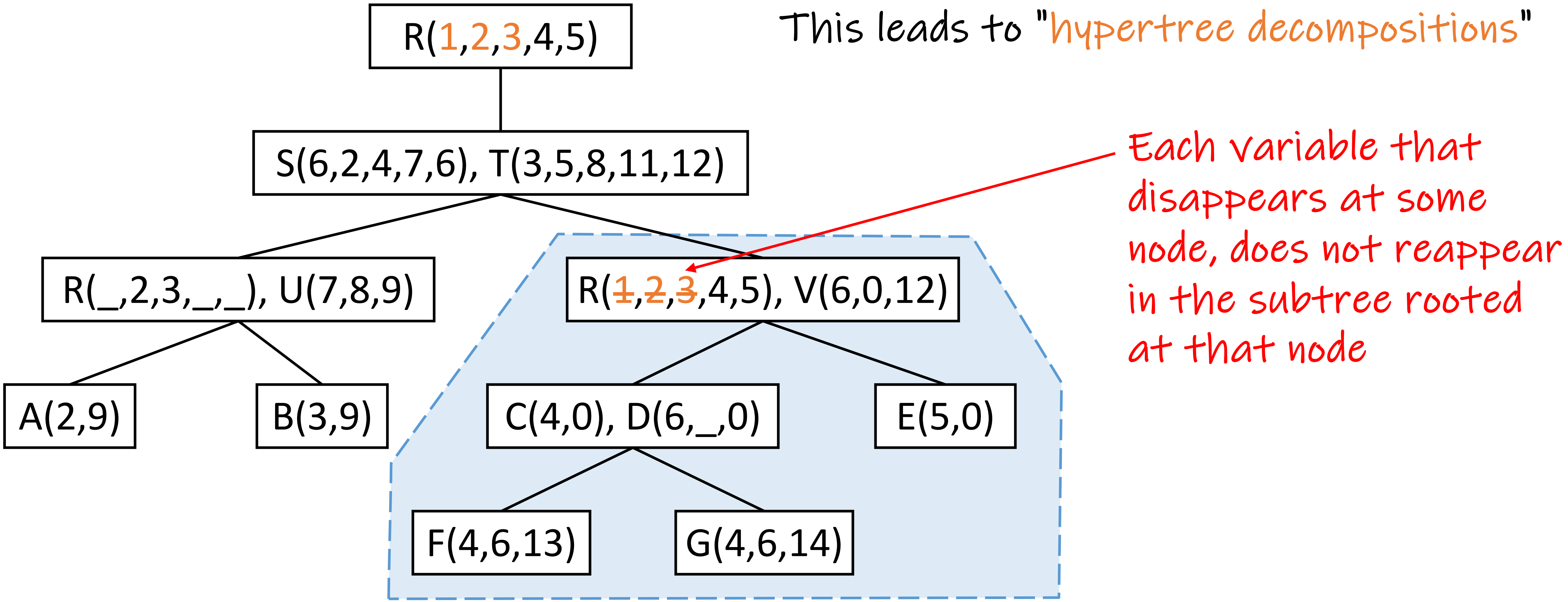
One can avoid NP-hardness of finding a minimal size decomposition by adding an additional syntactic "descendant condition".
This leads to "hypertree decompositions"



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"



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 $p \in vertices(T)$, $var(\lambda(p)) \cap \chi(T_p) \subseteq \chi(p)$.

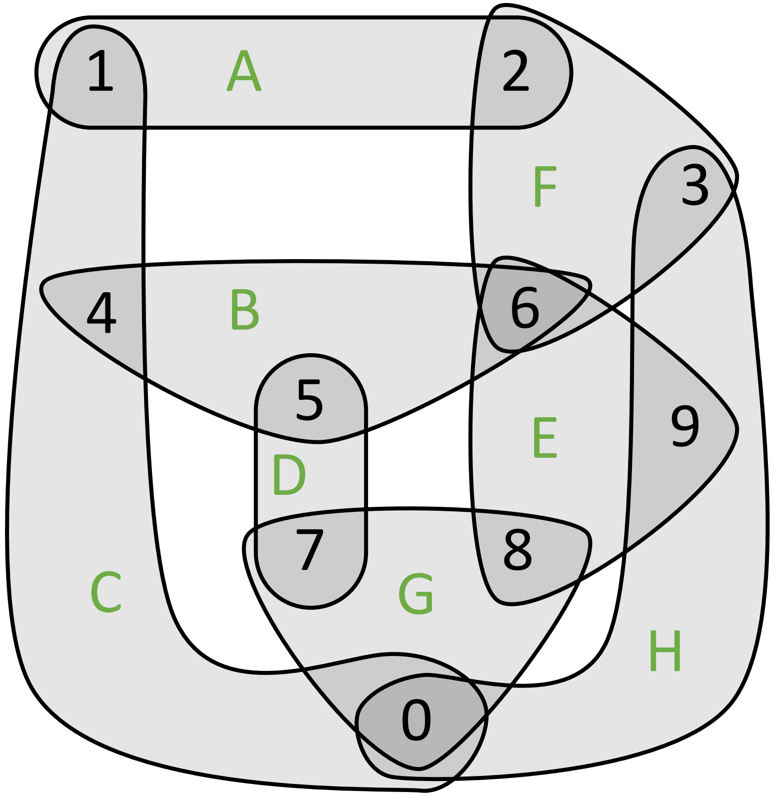
A hypertree decomposition $\langle T, \chi, \lambda \rangle$ of Q is a *complete decomposition* of Q if, for each atom $A \in atoms(Q)$, there exists $p \in vertices(T)$ such that $var(A) \subseteq \chi(p)$ and $A \in \lambda(p)$.

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.

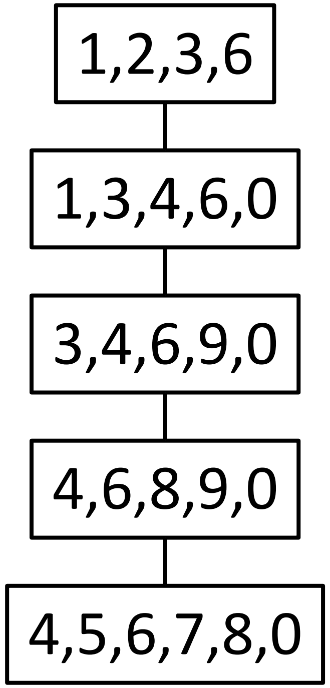
Source: Gottlob, Leone, Scarcello. "Hypertree decompositions and tractable queries." PODS 1999. https://doi.org/10.1145/303976.303979 (Gems of PODS 2016)

Hypertree decomposition: full example

Hypergraph



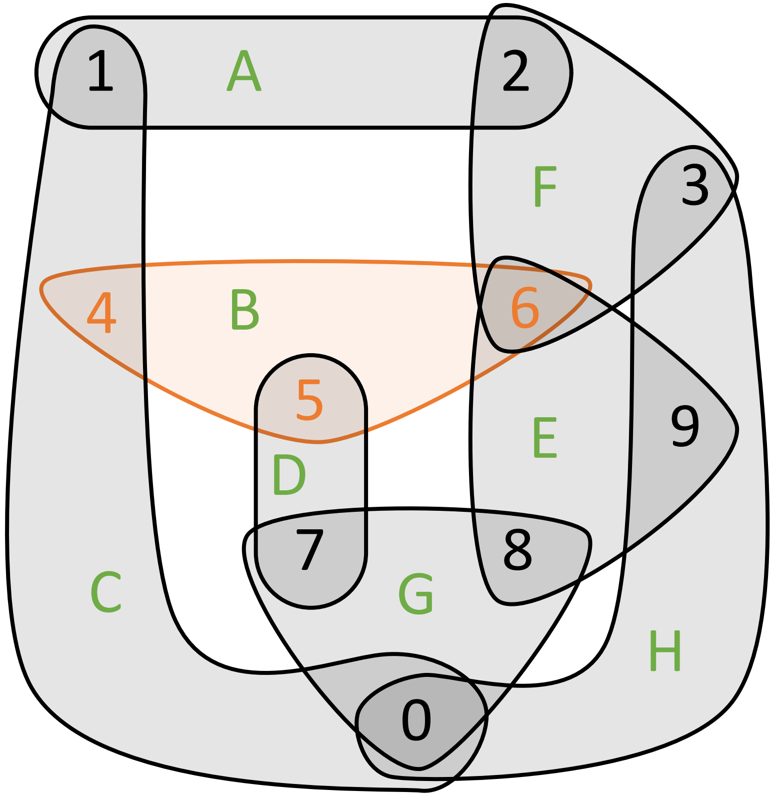
Tree decomposition



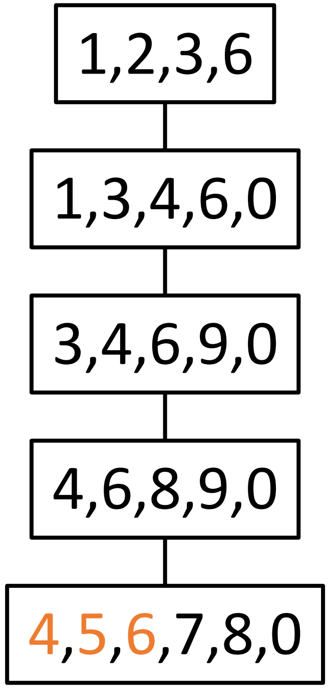
How to check that this is a valid tree decomposition? ?

Hypertree decomposition: full example

Hypergraph



Tree decomposition

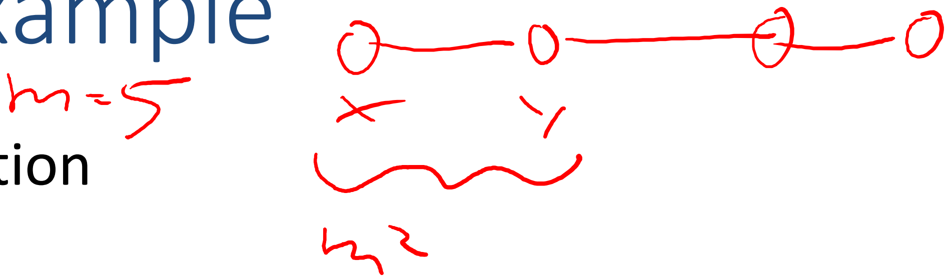


TREE DECOMPOSITION (ALTERNATIVE)

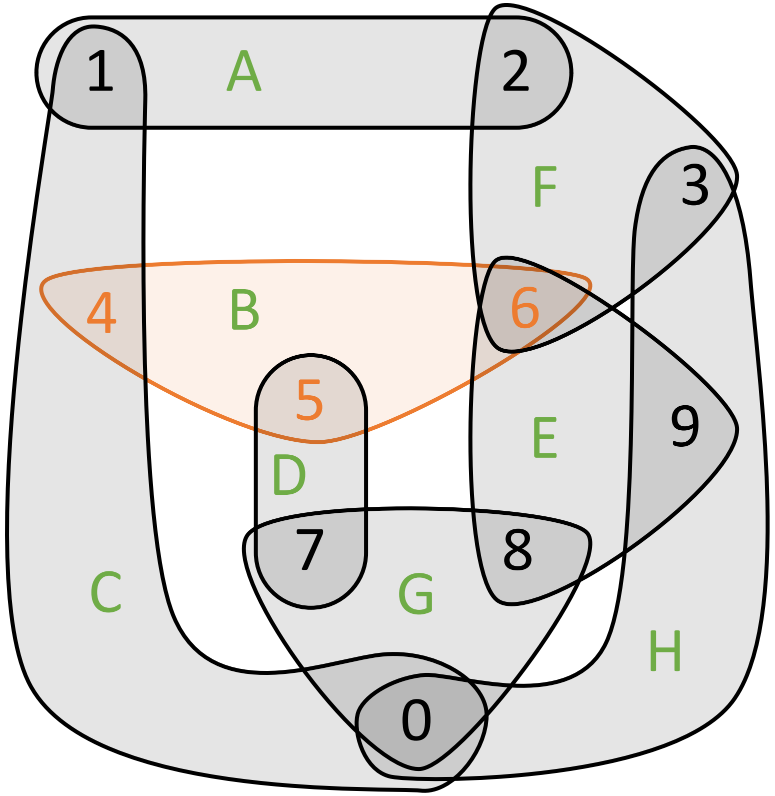
1. Hyperedge coverage: For every hyperedge h of H, there is a vertex in T that contains all its variables
2. Coherence

What is its width ?

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

TREE DECOMPOSITION (ALTERNATIVE)

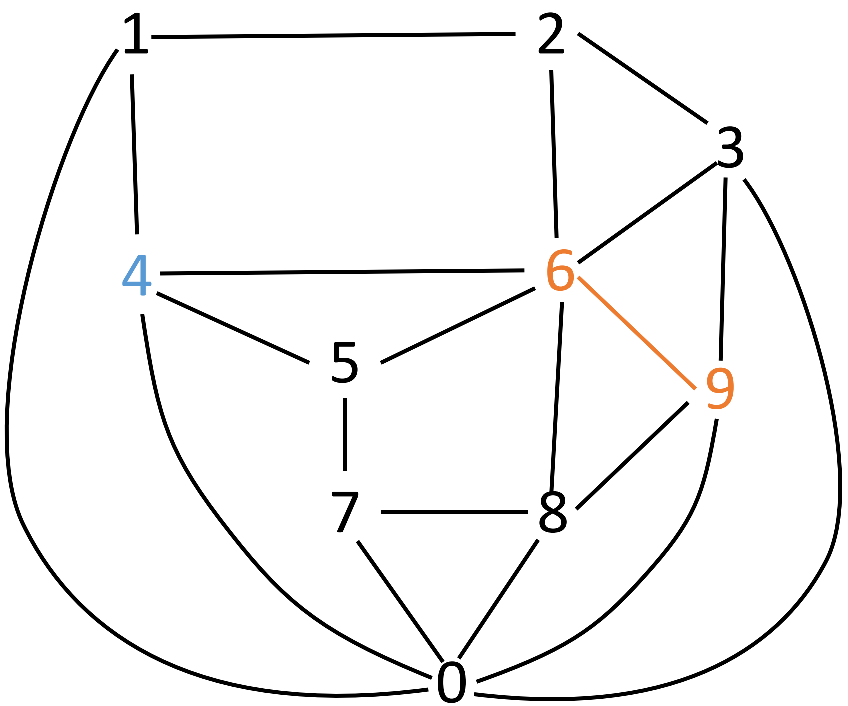
1. Hyperedge coverage: For every hyperedge h of H, there is a vertex in T that contains all its variables
2. Coherence

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

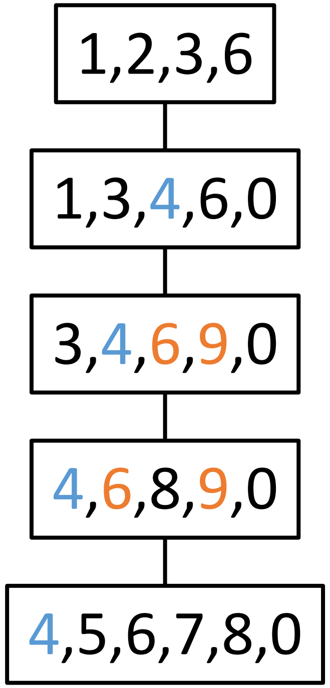
tree width = 5:
= size of largest supernode - 1

Hypertree decomposition: full example

Clique graph of Hypergraph
(also primal or Gaifman graph)



Tree decomposition
(width 5)



TREE DECOMPOSITION

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

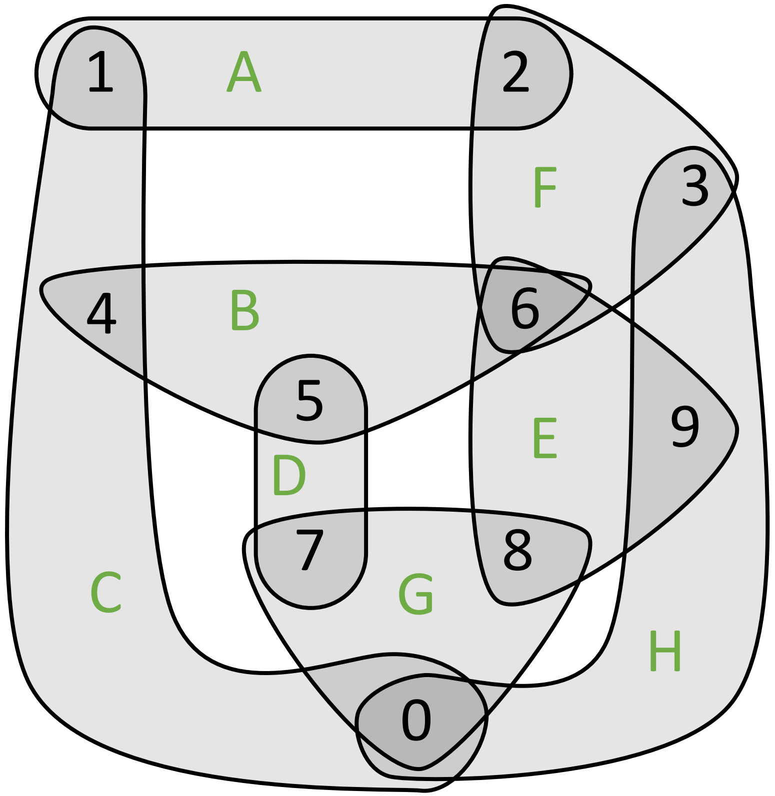
2. Coherence

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}

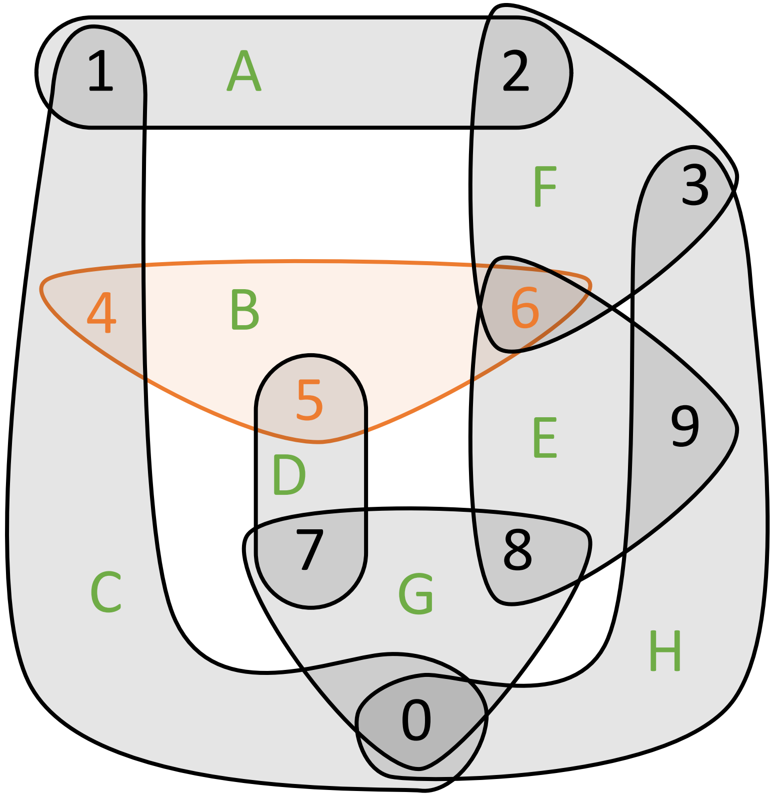
Is this a valid "generalized hypertree decomposition"; where is D?

?

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
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

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}

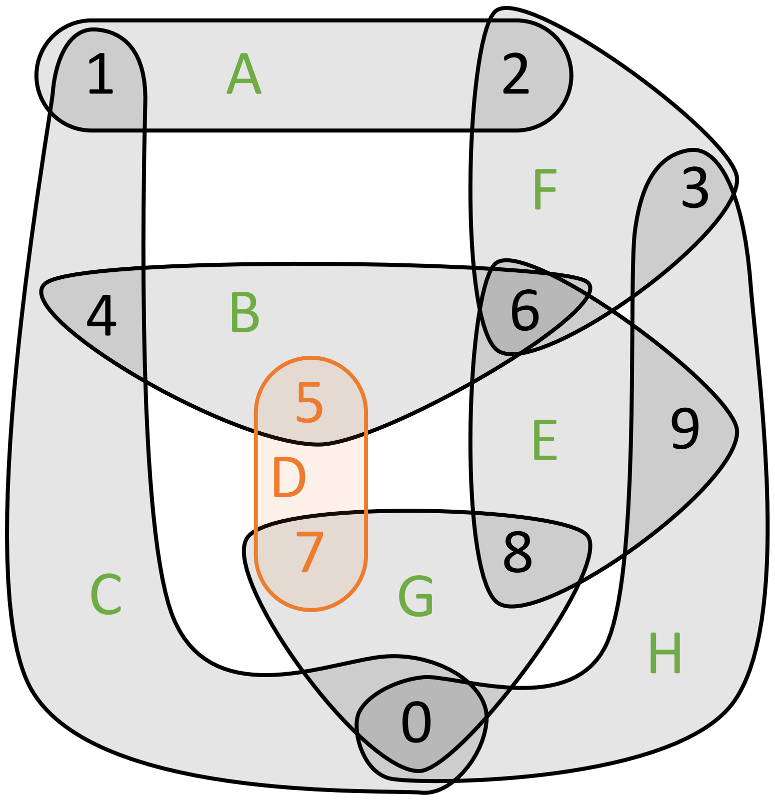
Basically identical to tree decomposition.
Just the width measure is different!

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
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Hypertree decomposition: full example

Final algorithm O(n²) preprocessing (materializing the vertices of the decomposition), then Yannakakis O(r)

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

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

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}

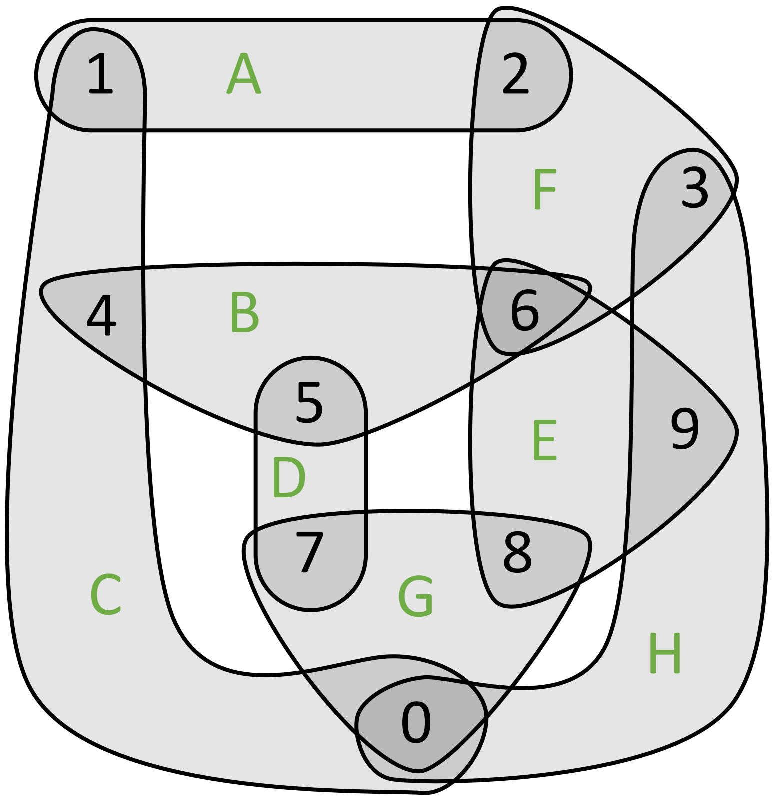
C{~~1~~,4,0}, E{6,8,9}

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

B and G together contain all variables from D

Hypertree decomposition: full example

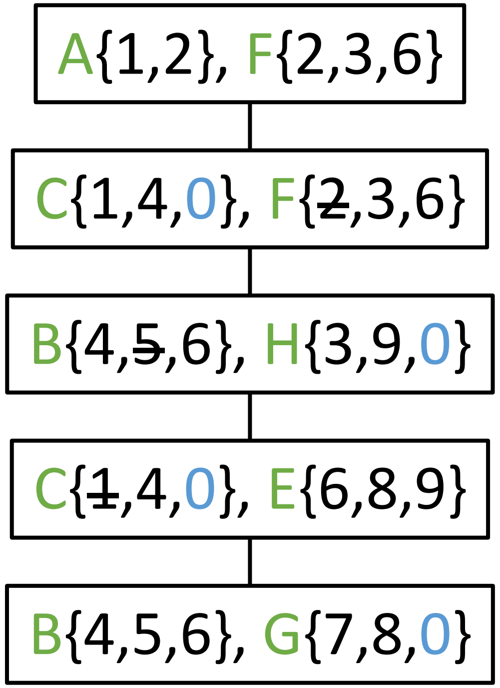
Hypergraph



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)

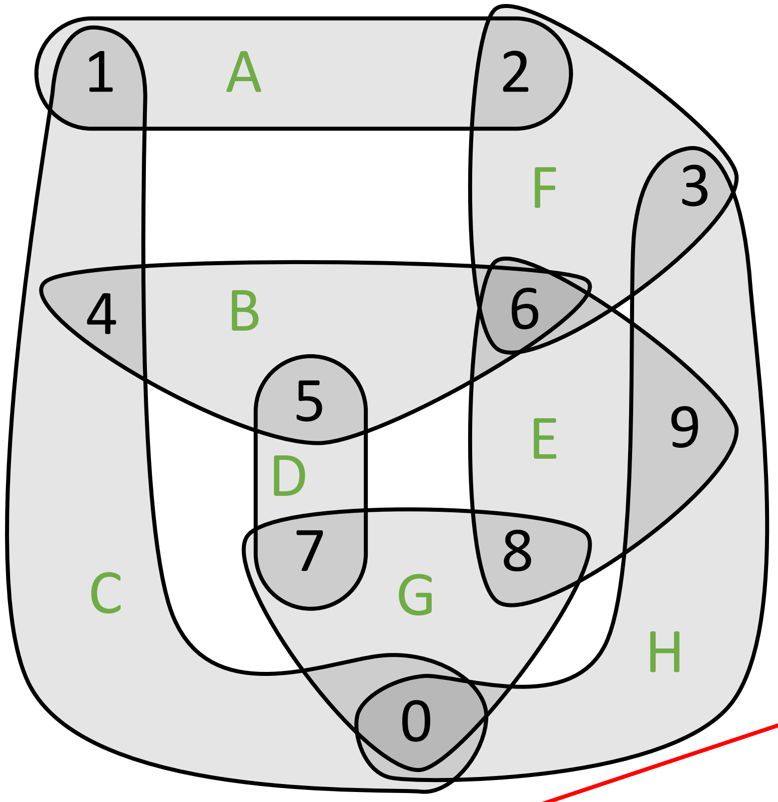


Is this also a valid "hypertree decomposition"?

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

Hypergraph



A condition to limit the search space of valid HD decompositions

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

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}

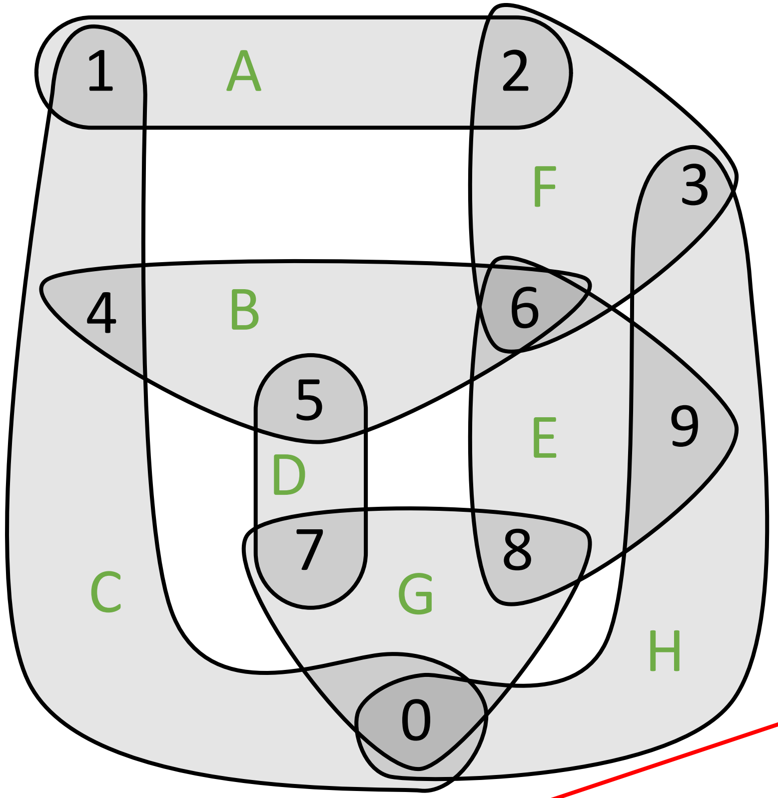
B{4,5,6}, G{7,8,0}

Is this also a valid "hypertree 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

Hypergraph



A condition to limit the search space of valid HD decompositions

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

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}

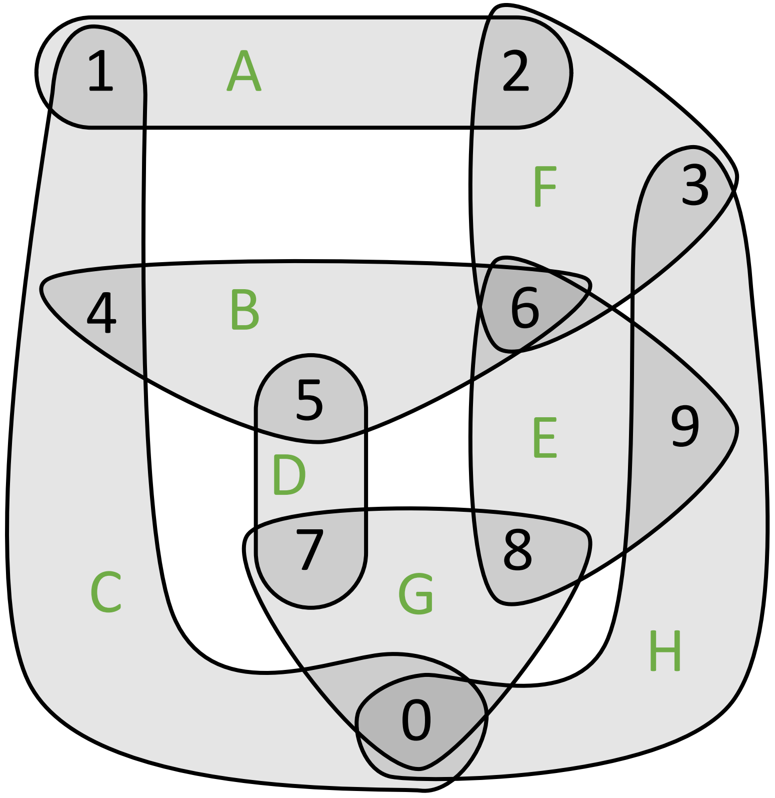
B{4,5,6}, G{7,8,0}

No: "5" got projected away, but reappears below. Also "1" in other direction

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

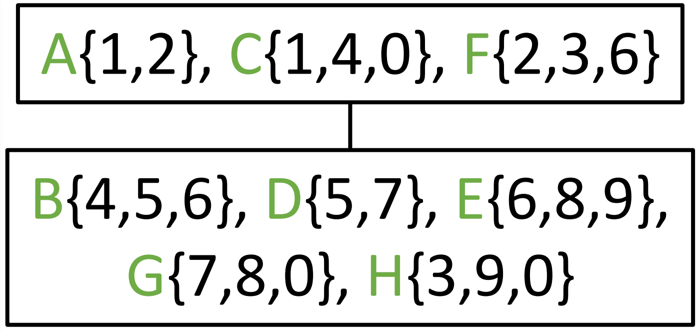
Hypergraph



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

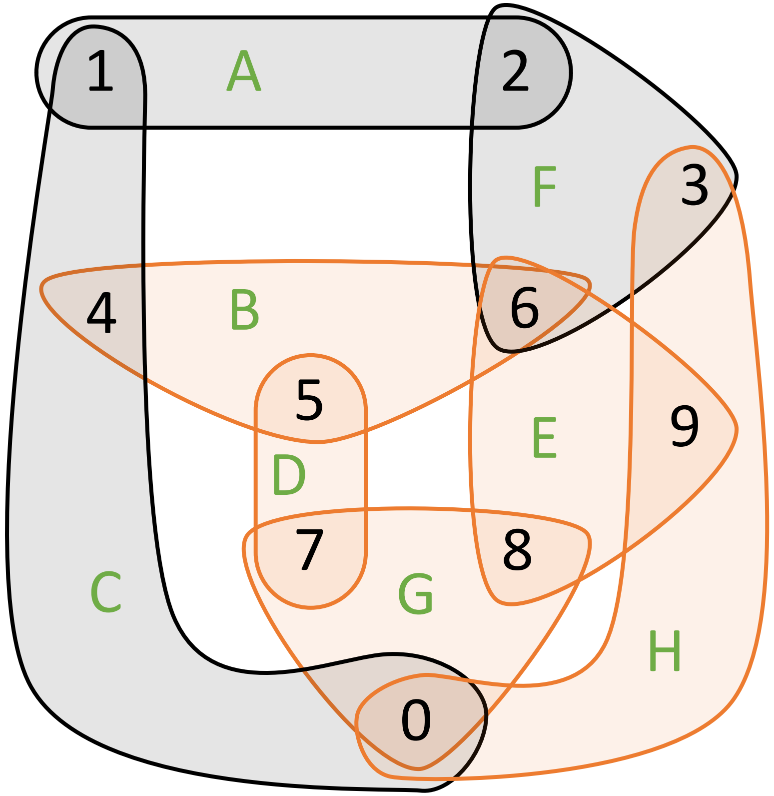
Hypertree decomposition



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

Hypergraph



Hypertree decomposition

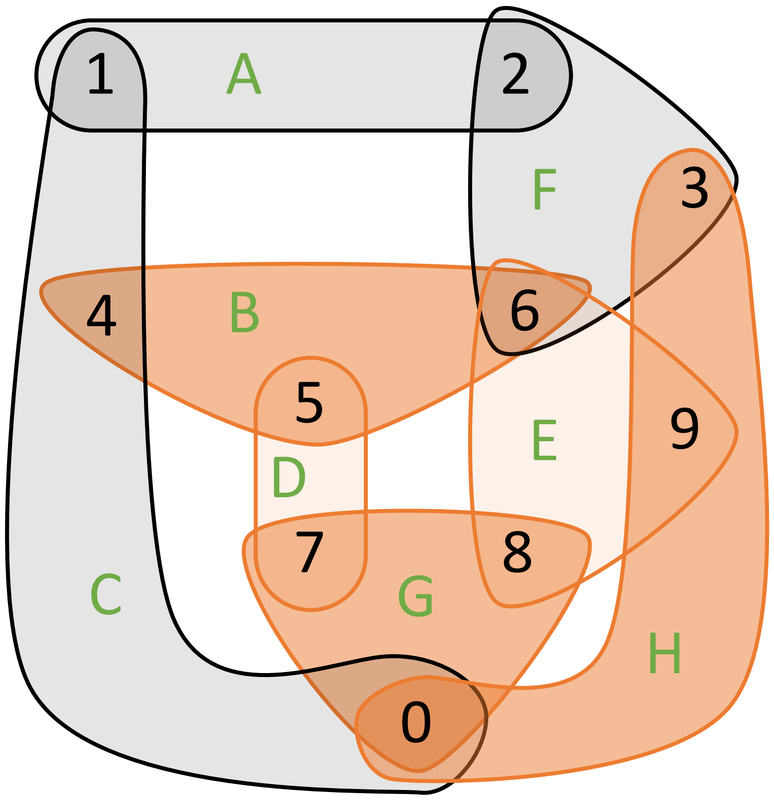
A{1,2}, C{1,4,0}, F{2,3,6}

B{4,5,6}, D{5,7}, E{6,8,9}, G{7,8,0}, H{3,9,0}

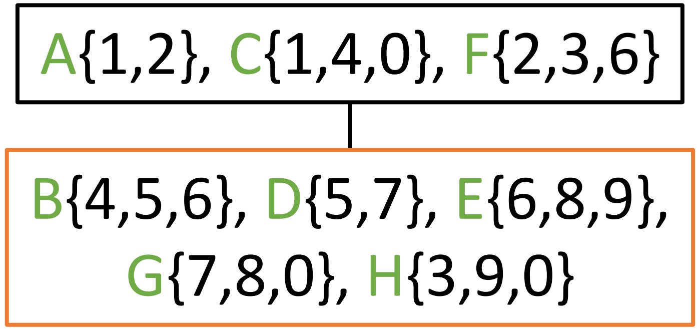
What should be the "width" of this HTD, i.e. what is the complexity of materializing this last supernode ?

Hypertree decomposition: full example

Hypergraph



Hypertree decomposition



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.

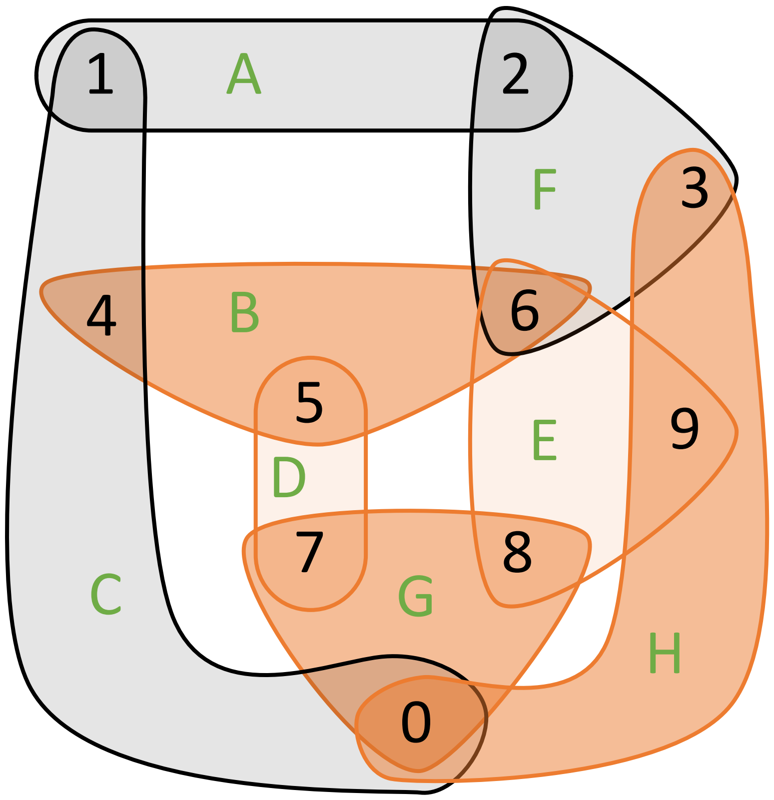
([(B(4,5,6) ⋈ G(7,8,0)) ⋈ H(3,9,0)] ← $O(n^3)$
⋉D(5,7)) ⋉E(6,8,9)

n... maximal size of relations

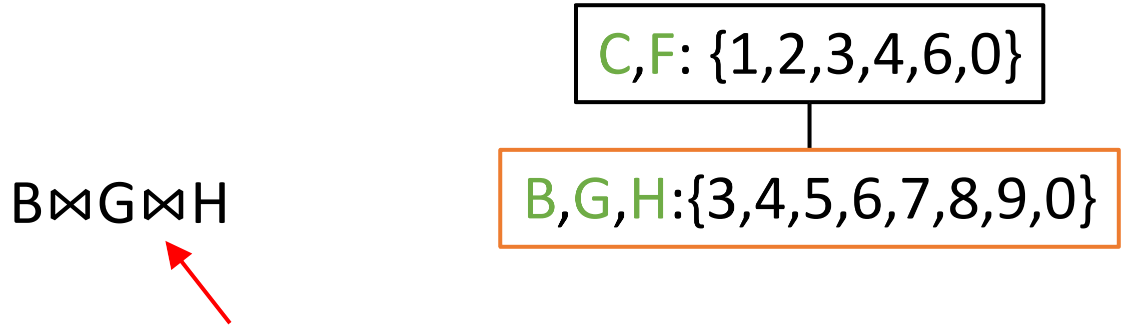
Example adopted from: Markus Krötzsch. "Database theory: Lecture 6: Tree-like Conjunctive Queries." 2016. https://iccl.inf.tu-dresden.de/web/Database_Theory_(SS2016)/en

Hypertree decomposition: full example

Hypergraph



Hypertree decomposition
(width 3)



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*

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

Hypertree Decompositions and friends

Query decomposition
[Chekuri, Rajaraman 1997]

NP-complete to find the optimum

↓ towards tighter bounds
(below is better)

Hypertree Decomposition (HD)
[Gottlob, Leone, Scarcello 1999]

PTIME to find the optimum

↓ towards tighter bounds
(below is better)

Generalized Hypertree Decomposition (GHD)
[Gottlob, Leone, Scarcello 2001]

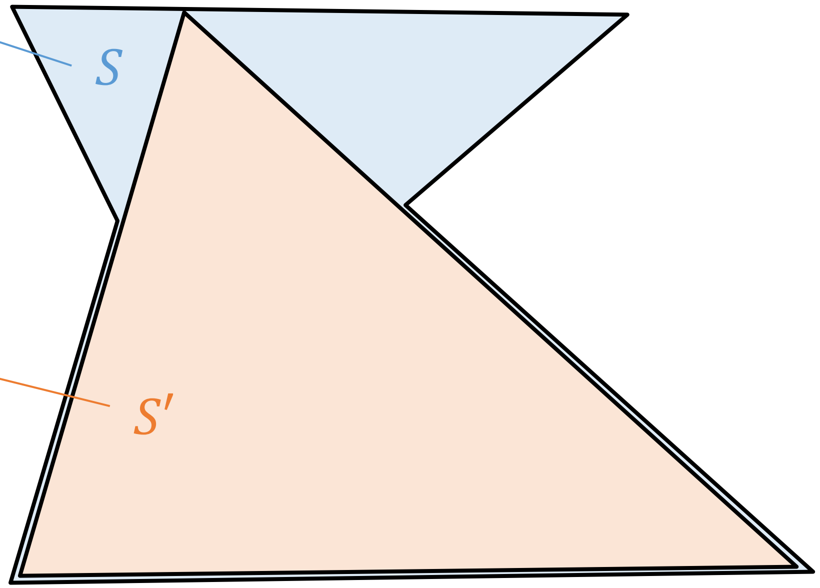
NP-complete to find the optimum

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)
Gottlob, Leone, Scarcello. "Hypertree decompositions and tractable queries." PODS 1999. https://doi.org/10.1145/303976.303979 (Gems of PODS 2016)
Gottlob, Leone, Scarcello. "Hypertree decompositions: a survey." MFCS 2001. https://dl.acm.org/doi/10.5555/645730.668191

Hypertree Decomposition: an unfortunate naming

1. Generalized Hypertree Decomposition (GHD):
explores the whole search space of valid decompositions
(illustrated here with a non-convex search space S in blue)

2. Hypertree Decomposition (HD):
limits the search space in a way that makes it tractable
to find the optimal solution within that limited subspace
(illustrated here with a convex search space $S' \subseteq S$)



Better names would be:

1. Hypertree Decomposition (HD) instead of GHD

2. Restricted Hypertree Decomposition (RHD) instead of HD

Outline: T3-2: Cyclic conjunctive queries

- T3-1: Acyclic conjunctive queries
- T3-2: Cyclic conjunctive queries
  - 2SAT (a detour)
  - Tree decompositions
  - Hypertrees decompositions
  - Duality in Linear programming (a not so quick primer)
  - AGM bound (maximal result size for full CQs) and Worst-case optimal joins for the triangle query
  - Worst-case optimal joins & the 4-cycle
  - Optimal joins & the 4-cycle

Topic Duality in Linear Programming (LP)

- Subtopics
  - Connections between (max) set packing and (min) set covers in graphs
  - Linear Programming (LP) and duality gaps
  - LP relaxations of ILP problems (Integer Linear Programming)
  - Duality b/w independent vertex sets and edge covers

What is "duality"?

Duality

- Duality in linear programming: Intuitively, every Linear Program has a dual problem with the same optimal solution, but the variables in the dual problem correspond to constraints in the primal problem and vice versa.
- But the notion of duality is more general:

DUALITY IN MATHEMATICS AND PHYSICS*

SIR MICHAEL F. ATIYAH

INTRODUCTORY REMARKS

Duality in mathematics is not a theorem, but a "principle". It has a simple origin, it is very powerful and useful, and has a long history going back hundreds of years. Over time it has been adapted and modified and so we can still use it in novel situations. It appears in many subjects in mathematics (geometry, algebra, analysis) and in physics. Fundamentally, duality gives *two different points of view of looking at the same object.* There are many things that have two different points of view and in principle they are all dualities.

https://fme.upc.edu/ca/arxius/butlleti-digital/riemann/071218_conferencia_atiyah-d_article.pdf

The Princeton Companion to Mathematics

III.19 Duality

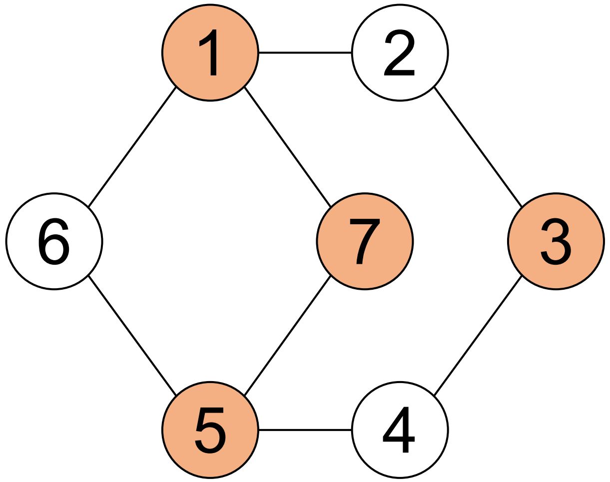
Duality is an important general theme that has manifestations in almost every area of mathematics. Over and over again, it turns out that one can associate with a given mathematical object a related, "dual" object that helps one to understand the properties of the object one started with. Despite the importance of duality in mathematics, there is no single definition that covers all instances of the phenomenon. So let us look at a

https://www.jstor.org/stable/j.ctt7sd01.7

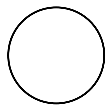
Let's use graphs to explain duality in LP (Linear Programming)

- **(max) Packing** problems: max number of disjoint subsets
  - max set packing: max number of subsets that are pairwise disjoint
  - max independent (vertex) set: max number of vertices not sharing edges
  - max independent edge set = matching: maximum number of edges that don't share any nodes (every vertex can be in max one matching)
- **(min) Coverings** problems: min number of subsets to cover all elements
  - min set cover: min number of subsets to cover the entire domain
  - min vertex cover: min number of vertices to cover all edges
  - min edge cover: min number of edges to cover all vertices
- Some packing problem is the dual problem of some covering problem
  - Min Vertex Cover (VC) is the dual of Max matching (independent edge set)
  - Max Independent Set (IS) is the dual of Min edge cover

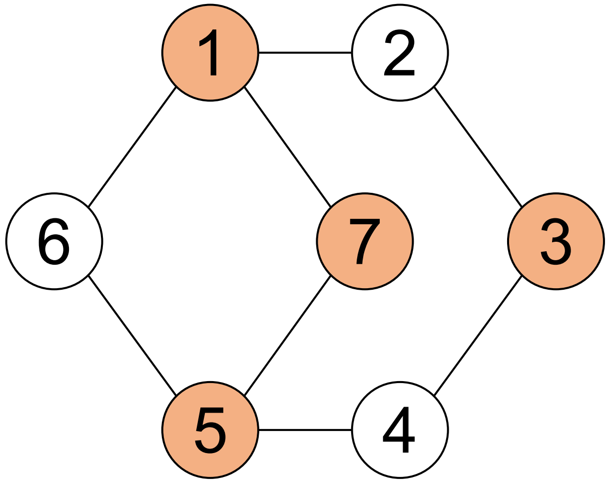
Independent set



Independent set (IS): set of vertices that are not connected (white)

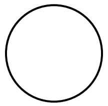


VC vs. Ind set ?



Independent set (IS): set of vertices that are not connected (white)

Vertex cover (VC): set of vertices that covers all edges

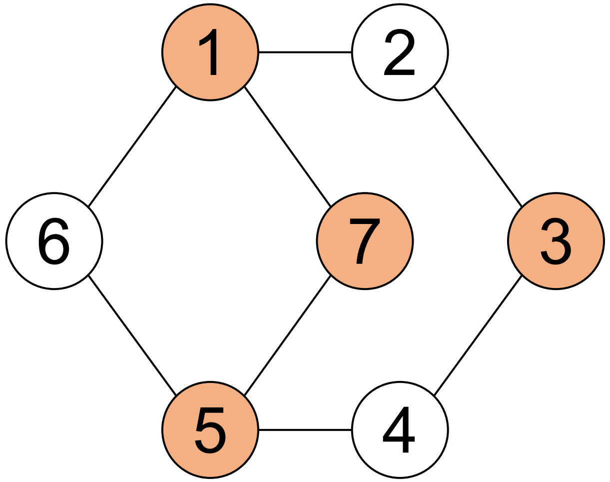


max

Assume you are given an independent set.
How do you find a vertex cover?

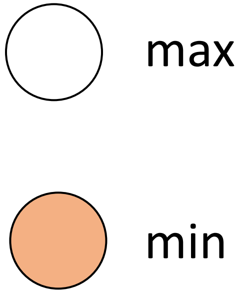
?

VC $=^c$ Ind set

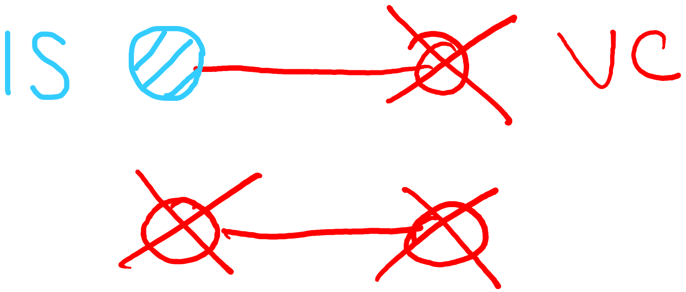


Independent set (IS): set of vertices that are not connected (white)

Vertex cover (VC): set of vertices that covers all edges (orange)

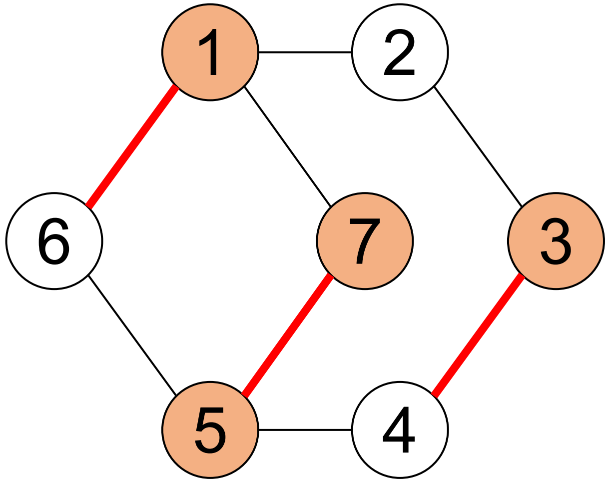


Set S is a **VC** iff the **complement** $V^c = V - S$ is an **IS**



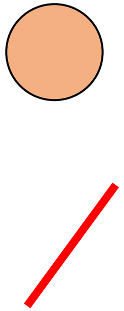
Proof: for each edge at most one vertex is in V^c.
Thus at least one vertex is in Set S.

Matching vs. VC?



Vertex cover (VC): set of vertices that covers all edges (orange)

Matching (Ind edge set): set of edges w/o common vertices (red)



min

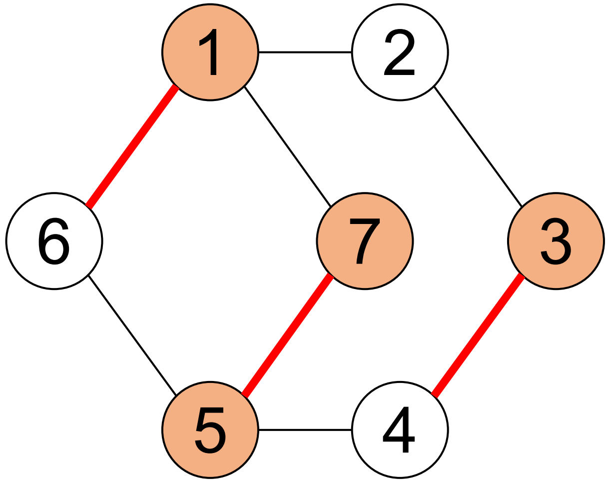
What is a possible connection between VC and matchings

?

Matching ≤ VC

That is called "weak duality"

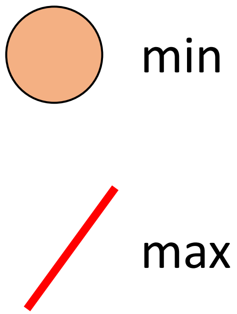
Any feasible solution to the minimization problem is at least as large as any feasible solution to the maximization problem

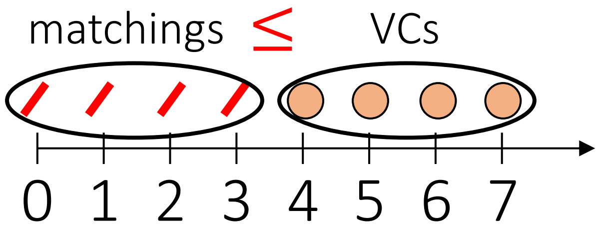


≤

Vertex cover (VC): set of vertices that covers all edges (orange)

Matching (Ind edge set): set of edges w/o common vertices (red)





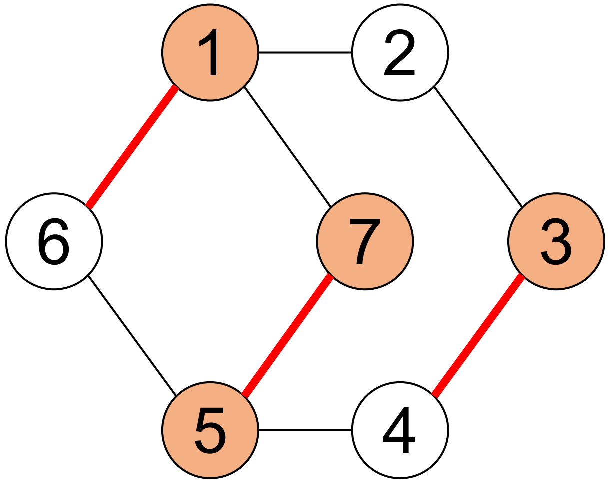
A **VC** needs to cover at least each edge from any matching

Thus, any VC has at least the size of any matching

⇒ **Size of any matching** ≤ any VC

That turns out to be the dual: Max Matching ≤ Min VC

Matching ≤ VC =ᶜ Ind set (summary so far)

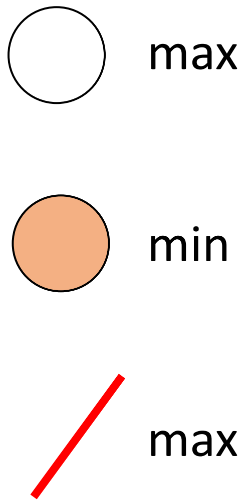


What intuitive problem is missing ?

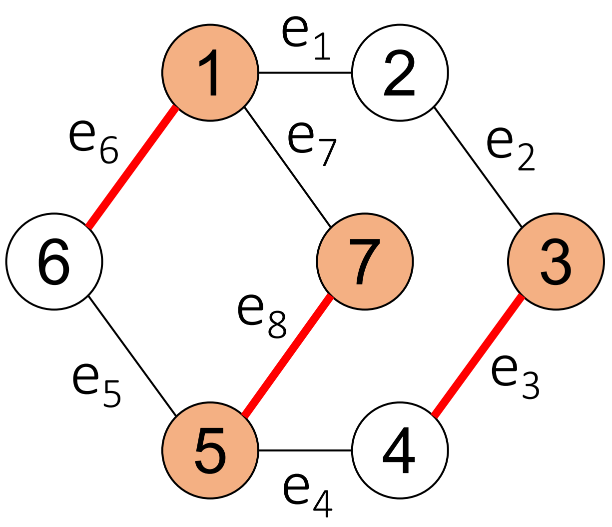
Independent set (IS): set of vertices that are not connected (white)

Vertex cover (VC): set of vertices that covers all edges (orange)

Matching (Ind edge set): set of edges w/o common vertices (red)



Matching ≤ VC =ᶜ Ind set (summary so far)

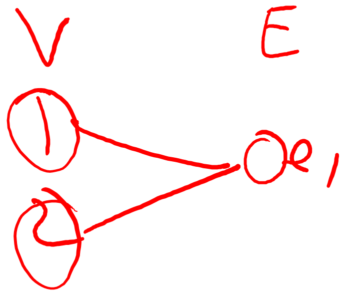


Edges = Sets

Vertices = elements

	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8
1	o					o	o	
2	o	o						
3		o	o					
4			o	o				
5				o	o			o
6					o	o		
7							o	o

Incidence matrix



What intuitive problem is missing ?

Independent set (IS): set of vertices that are not connected (white) — max

Vertex cover (VC): set of vertices that covers all edges (orange) — min

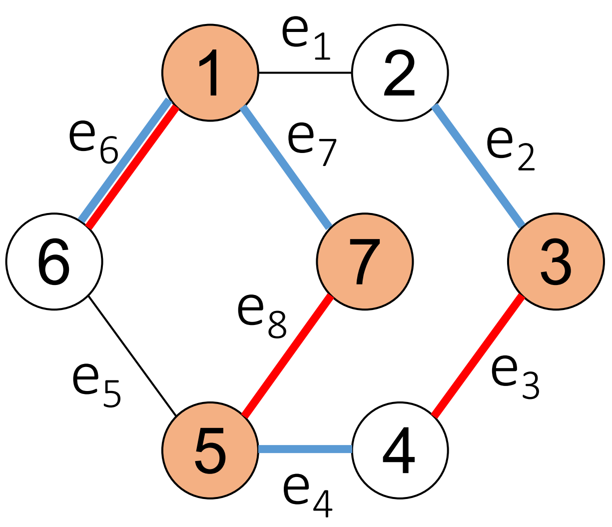
Matching (Ind edge set): set of edges w/o common vertices (red) — max

Cover problems: set of subsets that cover all elements

Packing problems: set of disjoint subsets

Matching ≤ VC =^{c} Ind set vs. Edge cover

What is its connection to IS ?



Edge cover: set of edges that cover all vertices (blue) — min

Independent set (IS): set of vertices that are not connected (white) — max

c

Vertex cover (VC): set of vertices that covers all edges (orange) — min

≤

Matching (Ind edge set): set of edges w/o common vertices (red) — max

Edges = Sets

Vertices = elements

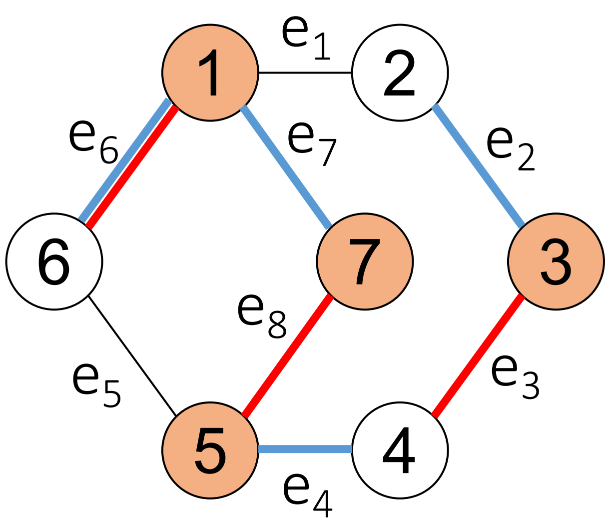
	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8
1	o					o	o	
2	o	o						
3		o	o					
4			o	o				
5				o	o			o
6					o	o		
7							o	o

Incidence matrix

Cover problems: set of subsets that cover all elements (min set cover: min vertex cover, min edge cover)

Packing problems: set of disjoint subsets (max set packing: max ind set, max matching)

Matching ≤ VC =^{c} Ind set ≤ Edge cover



Edges = Sets

Vertices = elements

	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8
1	o					o	o	
2	o	o						
3		o	o					
4			o	o				
5				o	o			o
6					o	o		
7							o	o

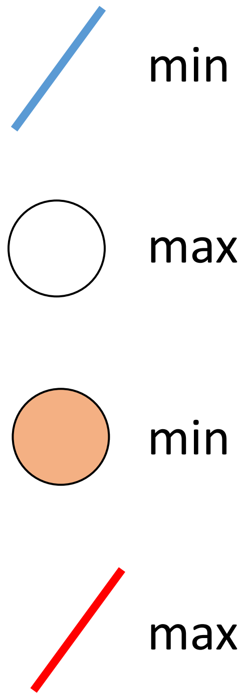
Incidence matrix

Edge cover: set of edges that cover all vertices (blue)

Independent set (IS): set of vertices that are not connected (white)

Vertex cover (VC): set of vertices that covers all edges (orange)

Matching (Ind edge set): set of edges w/o common vertices (red)

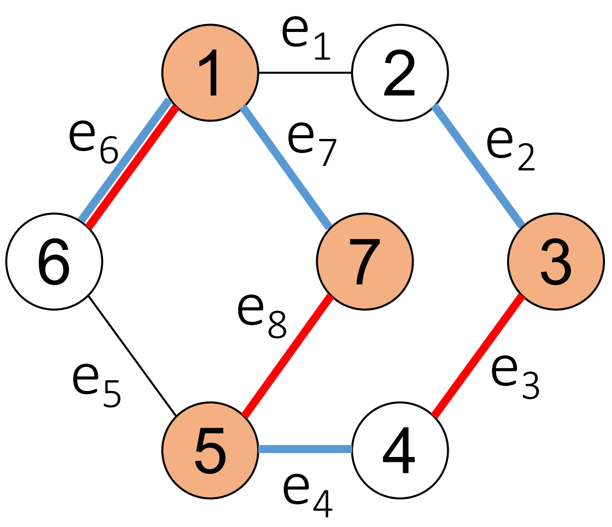


An **edge cover** needs to cover at least each vertex from any IS

Thus, any IS is lower bound to the size of any edge cover

⇒ **Size of min edge cover** ≥ max IS *(duality)*

4 graph problems in the incidence matrix



Edges = Sets

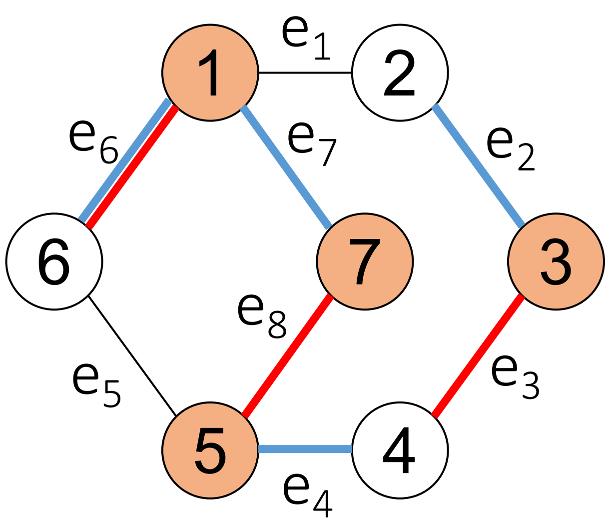
Vertices = elements

	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8
1	o					o	o	
2	o	o						
3		o	o					
4			o	o				
5				o	o			o
6					o	o		
7							o	o

Incidence matrix

	Choose Vertices	Choose Edges
Set **Cover**	?	?
Set **Packing**	?	?

4 graph problems in the incidence matrix

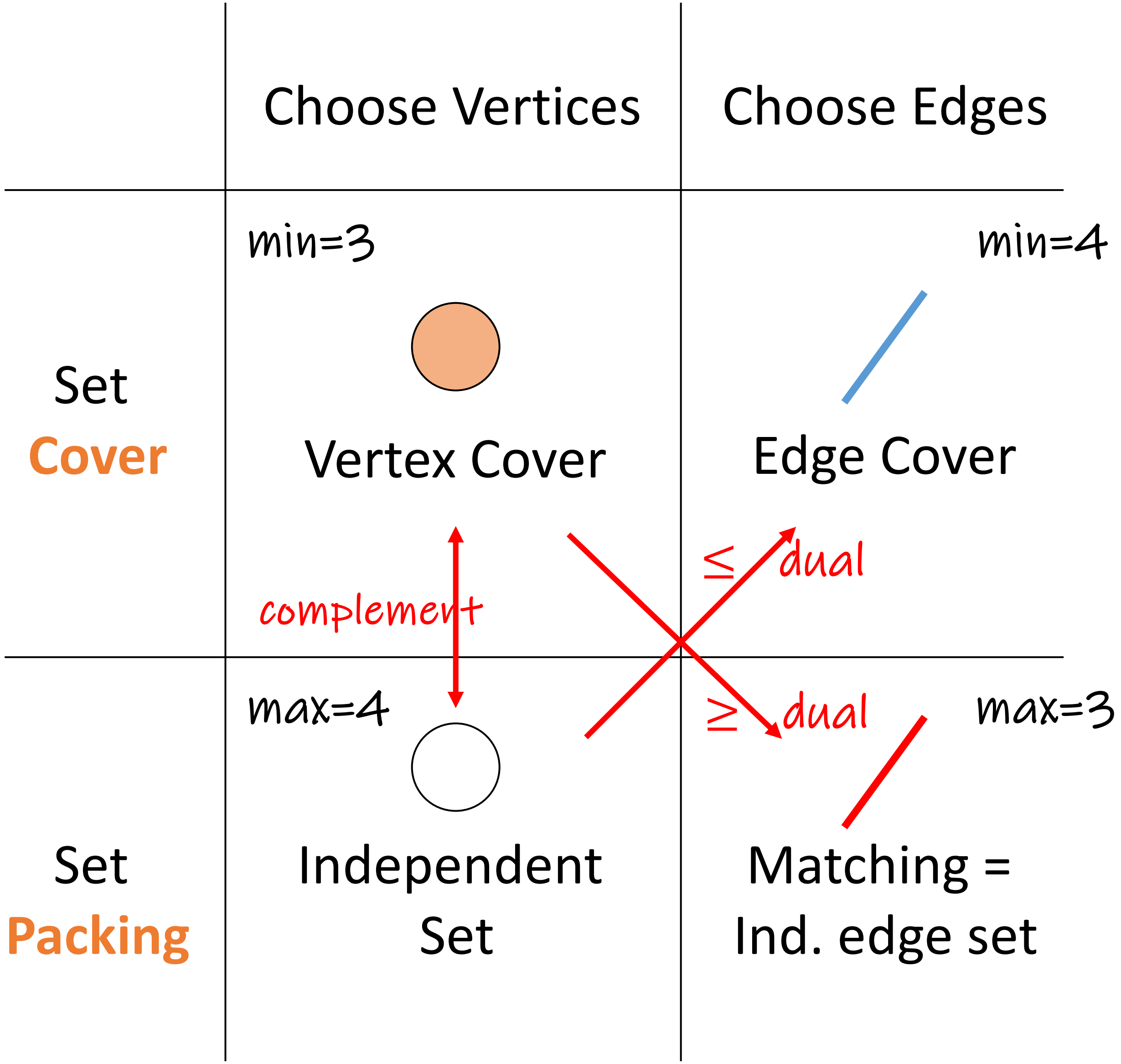


Edges = Sets

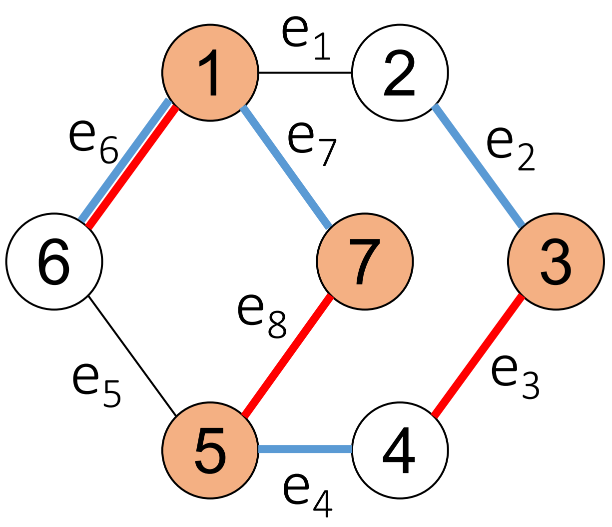
Vertices = elements

	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8
1	o					o	o	
2	o	o						
3		o	o					
4			o	o				
5				o	o			o
6					o	o		
7							o	o

Incidence matrix



4 graph problems in the incidence matrix

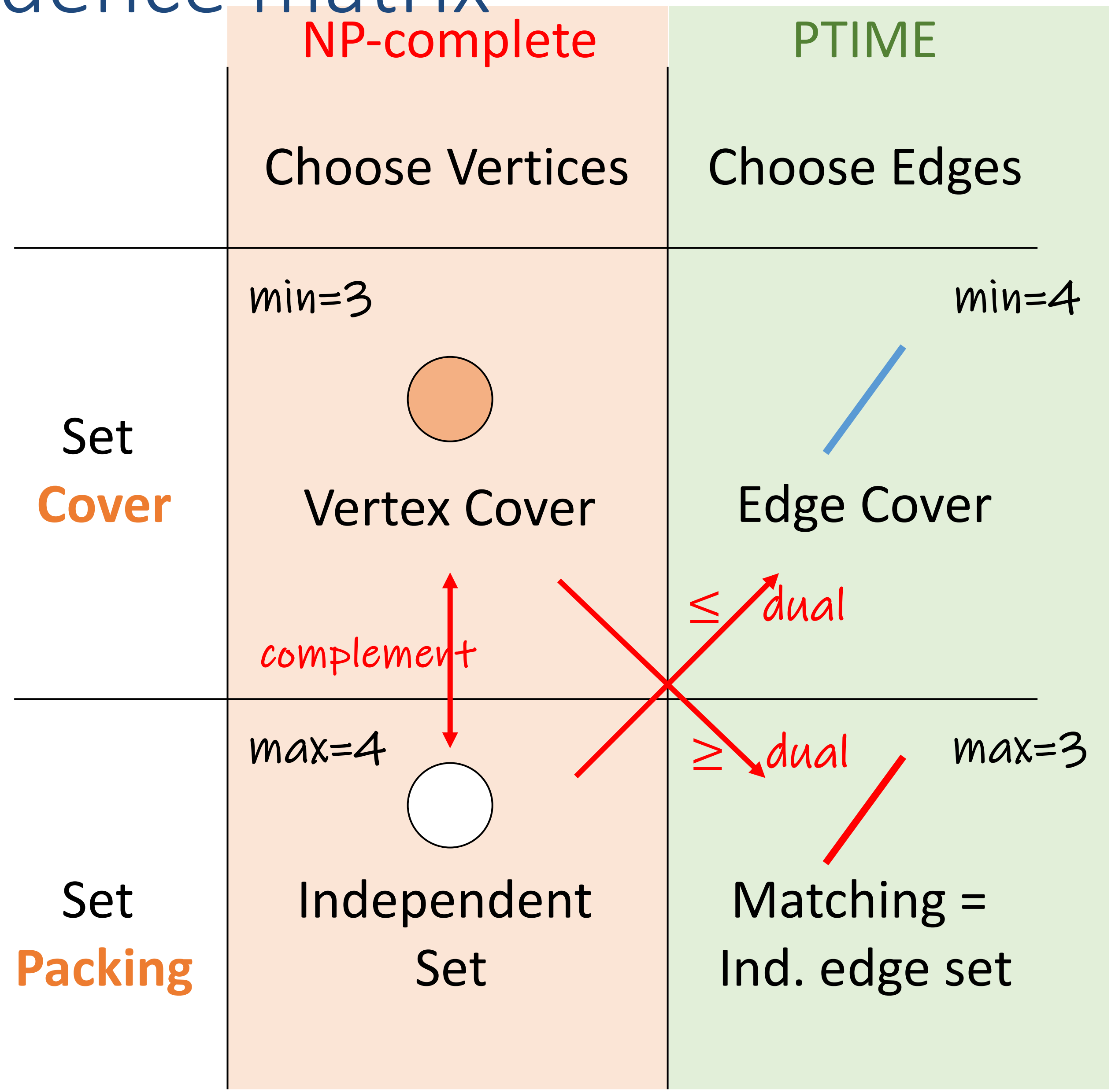


Edges = Sets

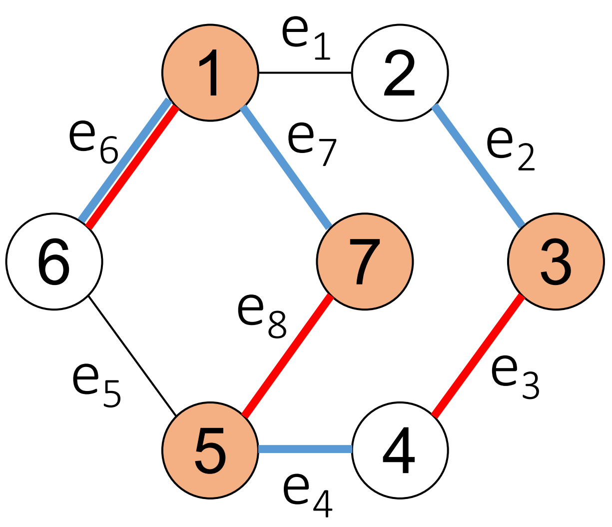
Vertices = elements

	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8
1	o					o	o	
2	o	o						
3		o	o					
4			o	o				
5				o	o			o
6					o	o		
7							o	o

Incidence matrix



4 graph problems in the incidence matrix



Edges = Sets

Vertices = elements

	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8
1	o					o	o	
2	o	o						
3		o	o					
4			o	o				
5				o	o			o
6					o	o		
7							o	o

Incidence matrix

	NP-complete Choose Vertices	PTIME Choose Edges
Set **Cover**	τ=3 (min) Vertex Cover	ρ=4 (min) Edge Cover
Set **Packing**	α=4 (max) Independent Set	μ=3 (max) Matching = Ind. edge set

complement

≤ dual

≥ dual

Same 4 problems for hypergraphs

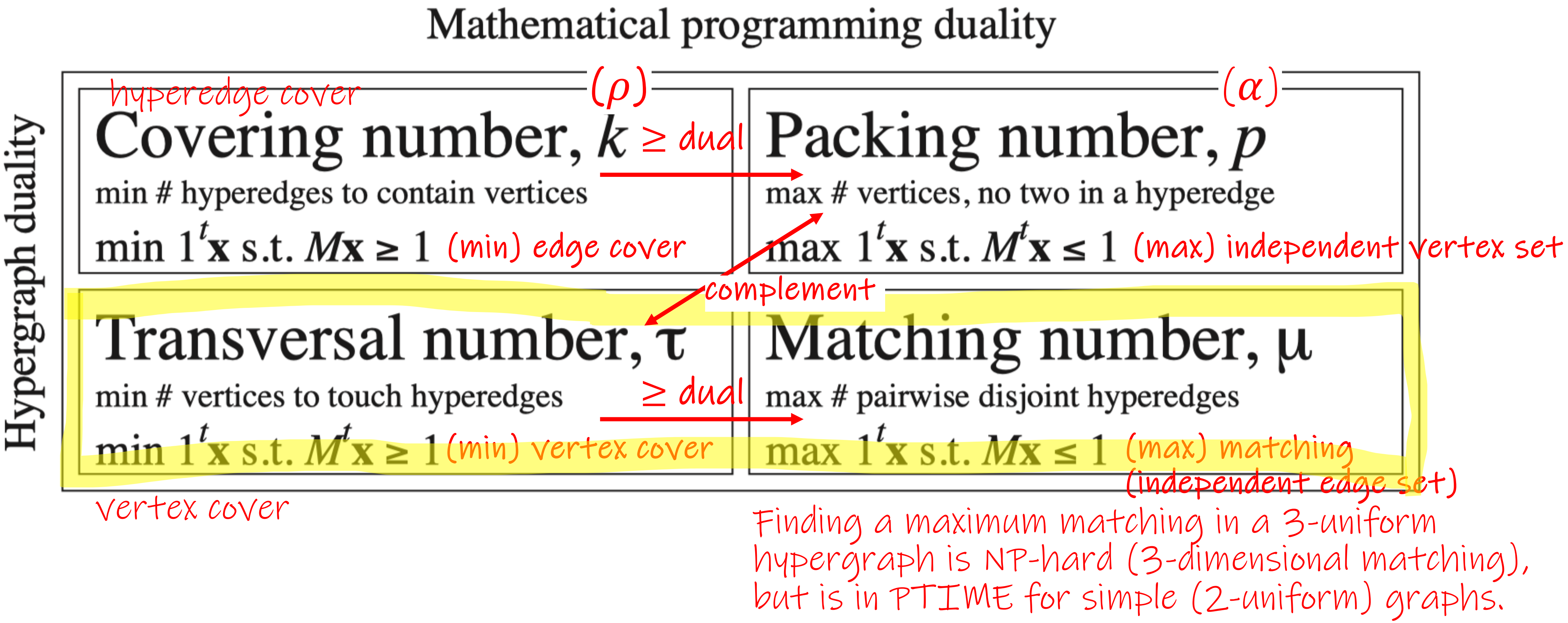
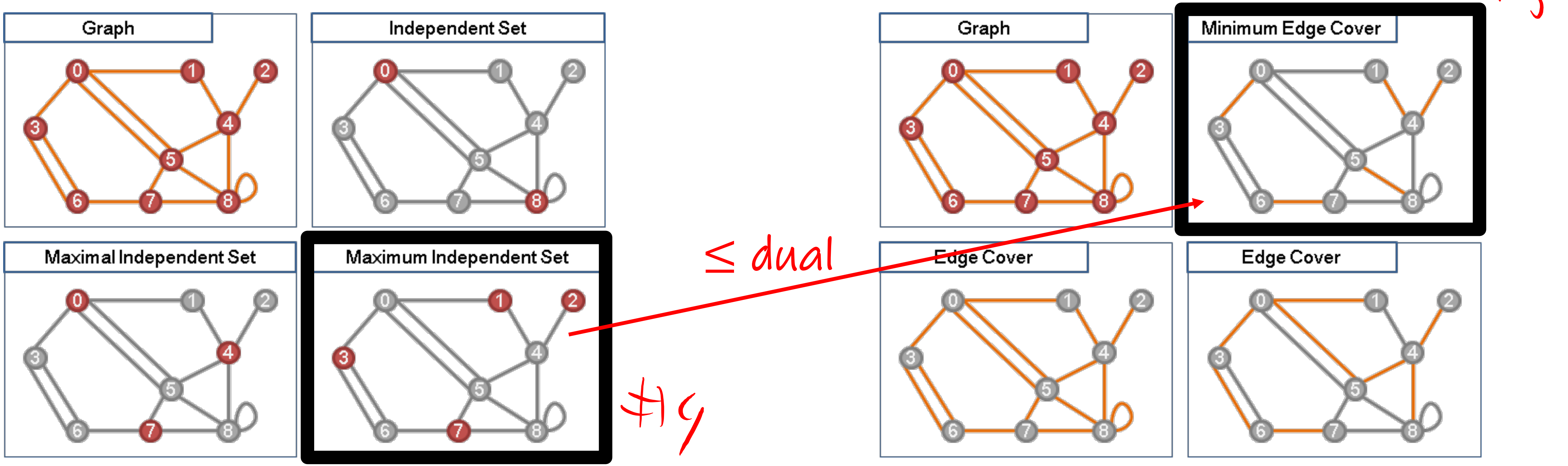


Figure 1.1. The dualities between the covering, packing, transversal, and matching numbers of a hypergraph.

Source: Scheinerman, Ullman. "Fractional Graph Theory: A Rational Approach to the Theory of Graphs", 1997/2008. https://www.ams.jhu.edu/ers/books/fractional-graph-theory-a-rational-approach-to-the-theory-of-graphs/

Background: MAX independent (vertex) set ≤ MIN edge cover

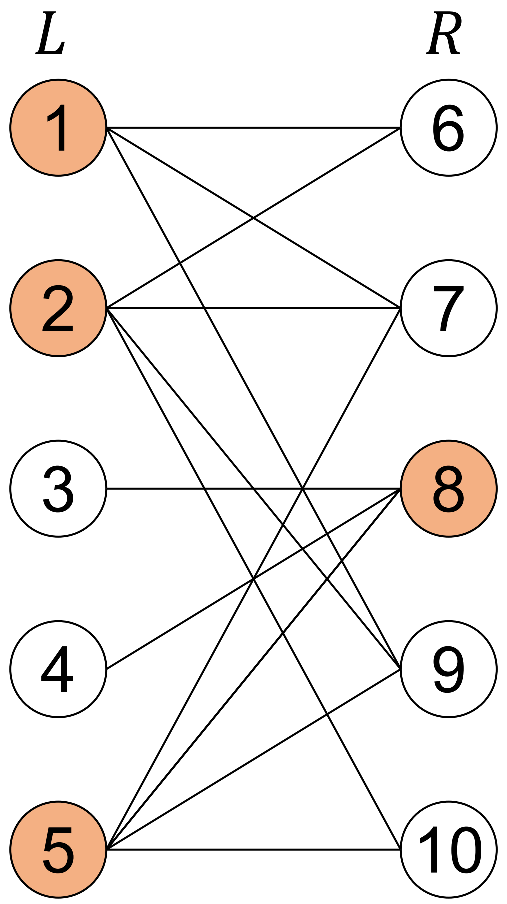


- Assume graph G is connected. Thus, every vertex has at least one edge (unless just one vertex)
- Suppose S is an independent set and E is an edge cover.
- Then for each vertex $v \in S$ there exists at least one edge $e \in E$ incident with v.
- By definition of independent set no two $u, v \in S$, have a common edge in E.
- Therefore $|S| \leq |E|$

Example from: http://www.csie.ntnu.edu.tw/~u91029/Domination.html

Matching ≤ VC: what changes in bipartite graphs?

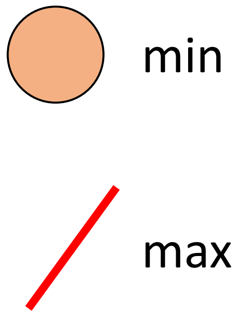
Nodes are partitioned into Left and Right



≤

Vertex cover (VC): set of vertices that covers all edges (orange)

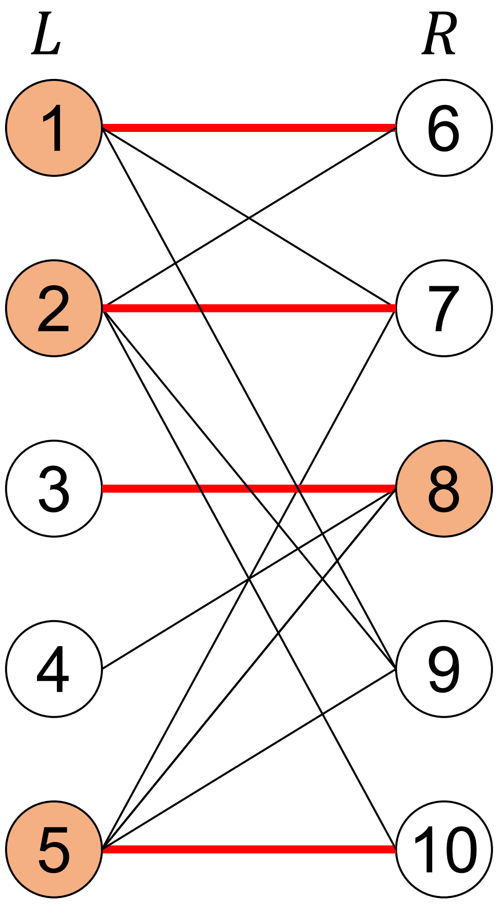
Matching (Ind edge set): set of edges w/o common vertices (red)



A **VC** needs to cover at least each edge from any matching

Thus, min VC at least the size of any matching
⇒ **Size of any matching** ≤ any VC

matching = VC ... in bipartite graphs!



Vertex cover (VC): set of vertices that covers all edges (orange)

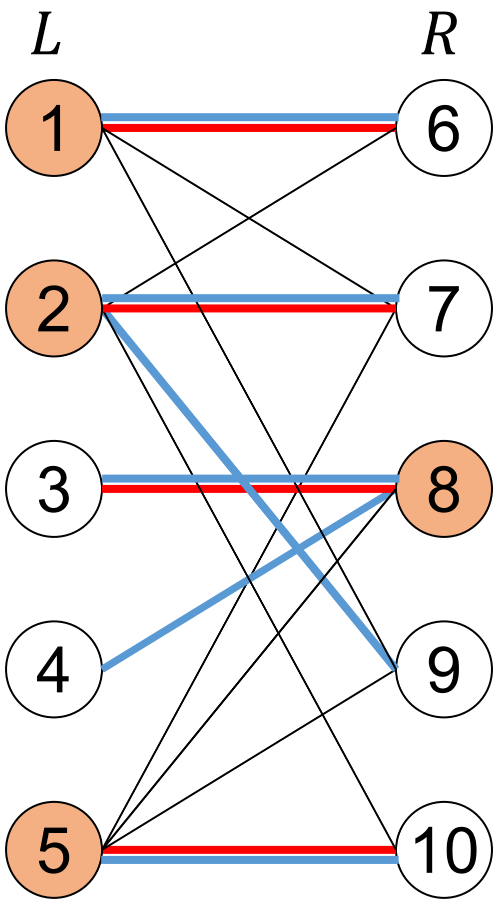
Matching (Ind edge set): set of edges w/o common vertices (red)

min

max

Kőnig-Egeváry theorem for bipartite graphs:
Max matching equivalent **to Min VC**

All for 4 problems become easy in bipartite graphs



PTIME

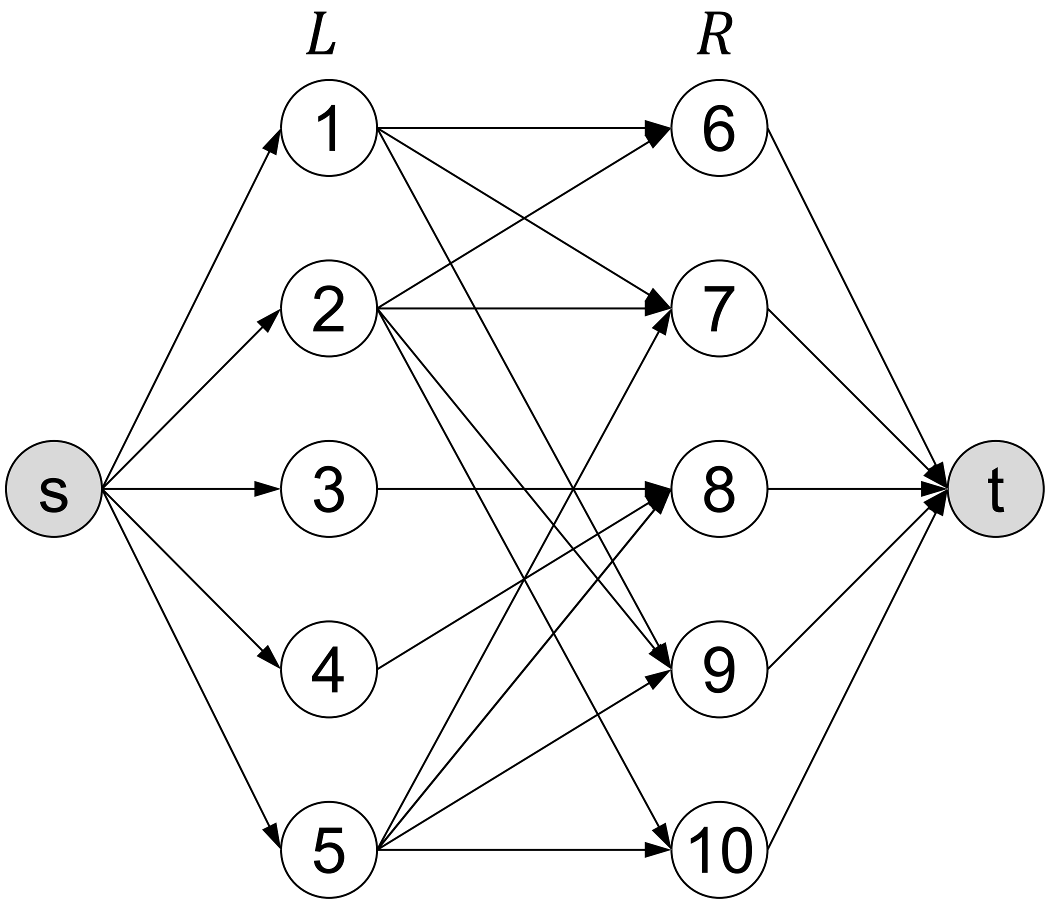
	Choose Vertices	Choose Edges
Set Cover	Vertex Cover	Edge Cover
Set Packing	Independent Set	Matching = Ind. edge set

complement

= dual

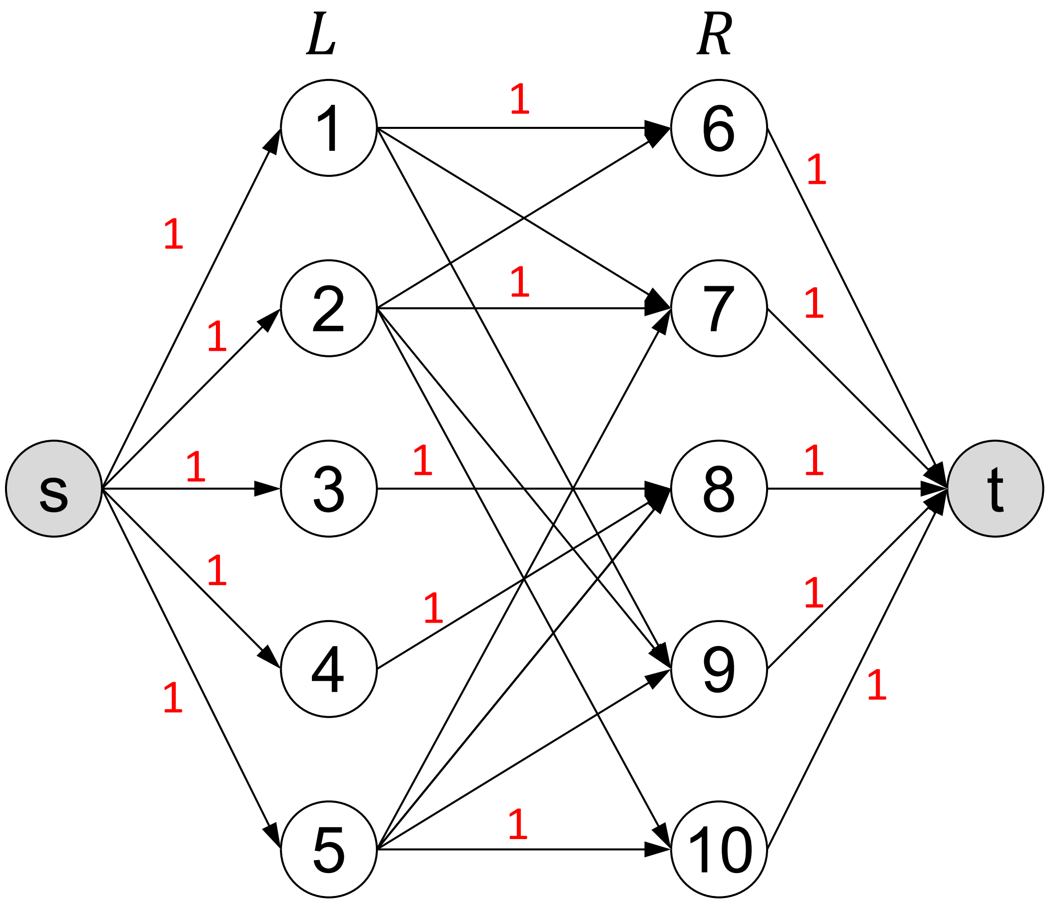
= dual

Cuts and Flows in directed graphs $G = (V, E)$



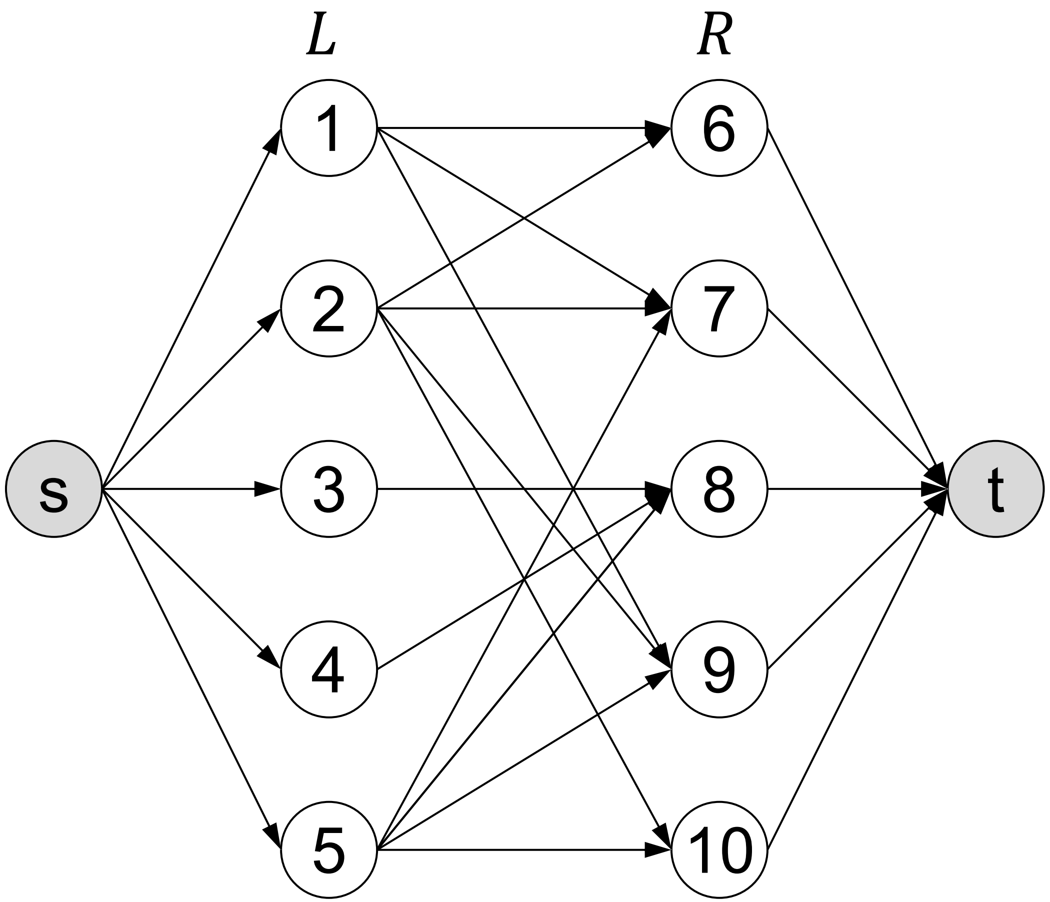
Cuts and Flows in directed graphs $G = (V, E)$

Each edge (u, v) has a capacity c_{uv} which is the max amount of flow that can pass through it.



Cuts and Flows in directed graphs $G = (V, E)$

Each edge (u, v) has a capacity $c_{uv} = 1$ which is the max amount of flow that can pass through it.

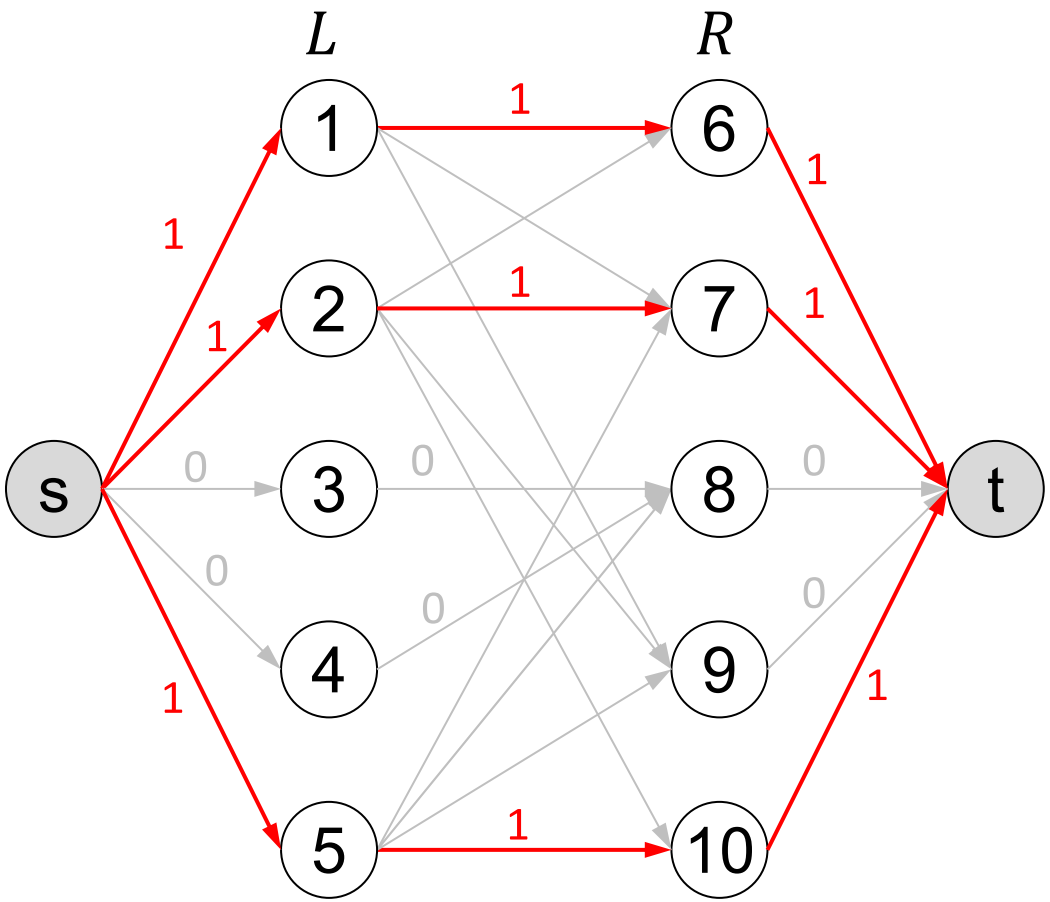


A **flow** is a mapping of edges to flows $f: E \rightarrow \mathbb{R}^+$ s.t. that flows obey their capacities $f_{uv} \leq c_{uv}$ and conservation laws. The **value** $|f|$ of a flow is the amount moved from S to T through the network.

Definitions adapted from: https://en.wikipedia.org/wiki/Max-flow_min-cut_theorem

Cuts and Flows in directed graphs $G = (V, E)$

Each edge (u, v) has a capacity $c_{uv} = 1$ which is the max amount of flow that can pass through it.



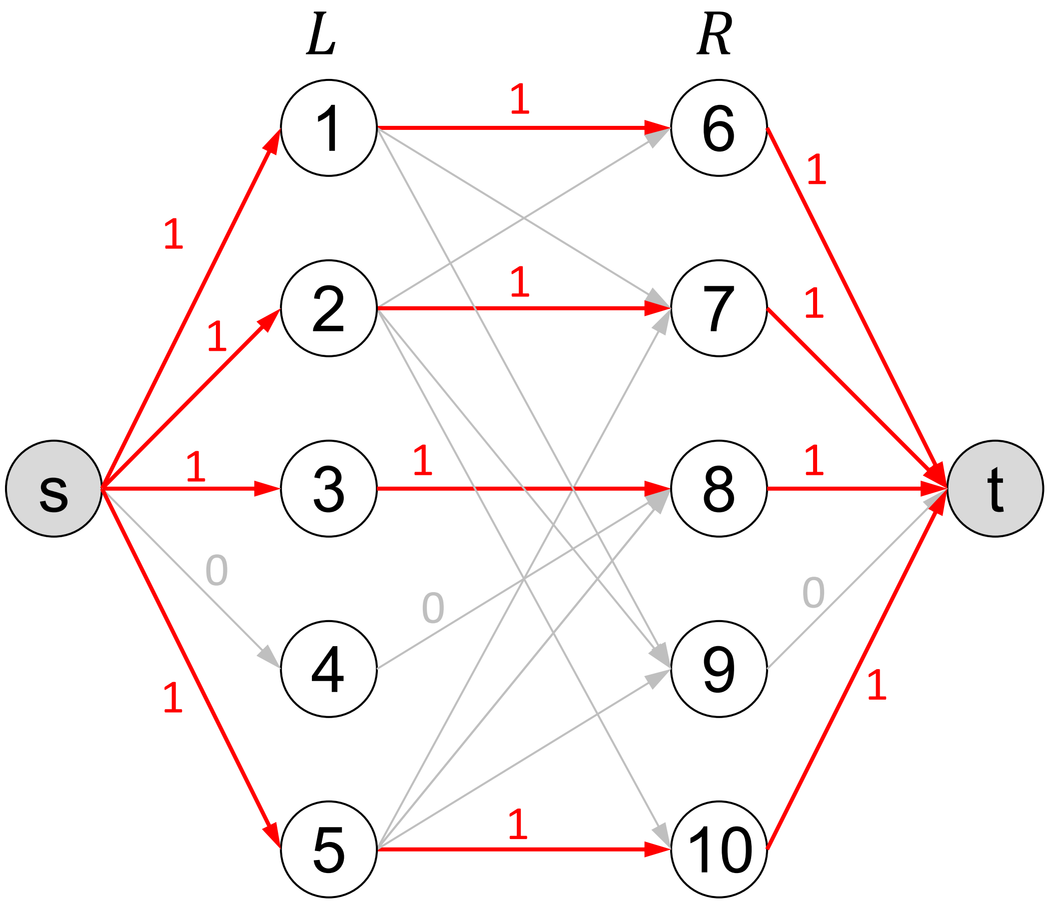
A **flow** is a mapping of edges to flows $f: E \rightarrow \mathbb{R}^+$ s.t. that flows obey their capacities $f_{uv} \leq c_{uv}$ and conservation laws. The **value** $|f|$ of a flow is the amount moved from S to T through the network.

$|f| = 3$

Definitions adapted from: https://en.wikipedia.org/wiki/Max-flow_min-cut_theorem

Cuts and Flows in directed graphs $G = (V, E)$

Each edge (u, v) has a capacity $c_{uv} = 1$ which is the max amount of flow that can pass through it.



A **flow** is a mapping of edges to flows $f: E \rightarrow \mathbb{R}^+$ s.t. that flows obey their capacities $f_{uv} \leq c_{uv}$ and conservation laws. The **value** $|f|$ of a flow is the amount moved from S to T through the network.

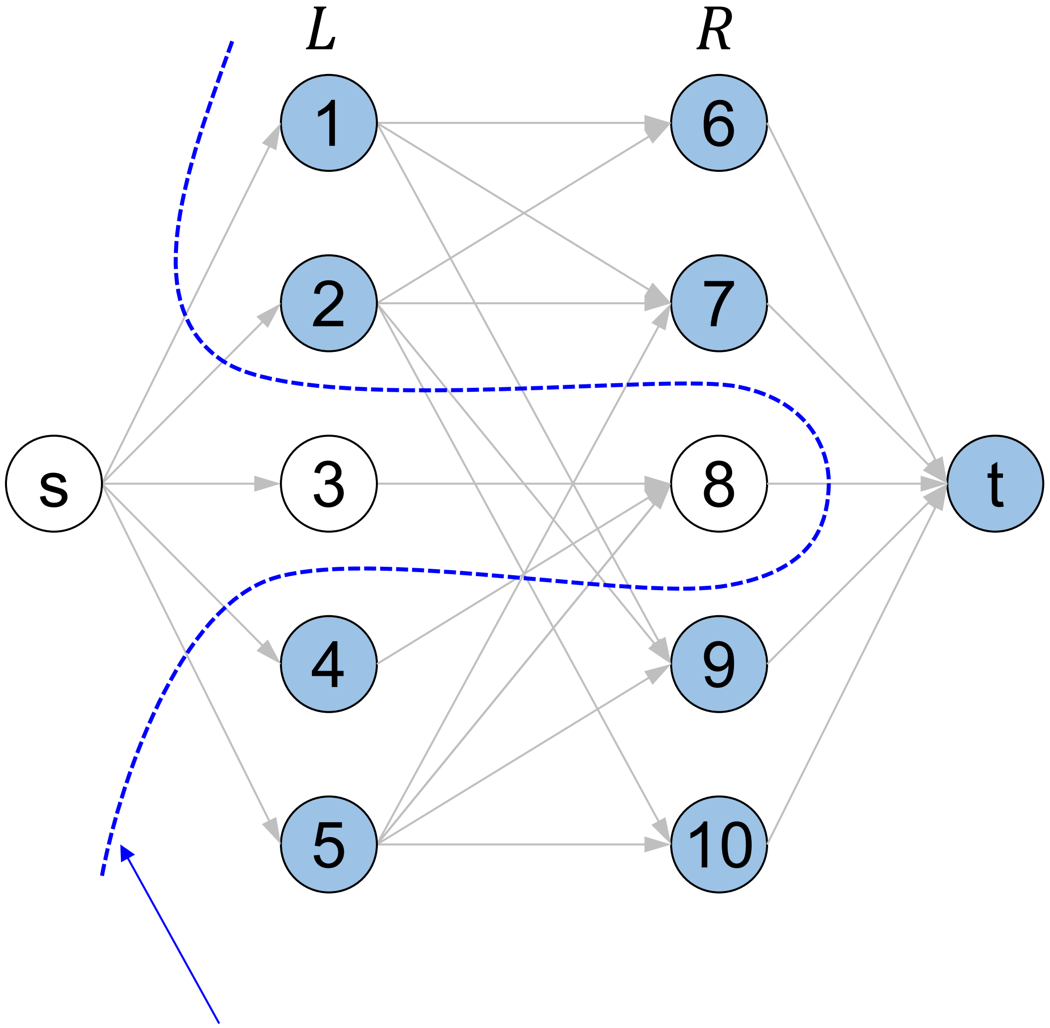
$|f| = 4$

Definitions adapted from: https://en.wikipedia.org/wiki/Max-flow_min-cut_theorem

Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Cuts and Flows in directed graphs $G = (V, E)$

Each edge (u, v) has a capacity $c_{uv} = 1$ which is the max amount of flow that can pass through it.



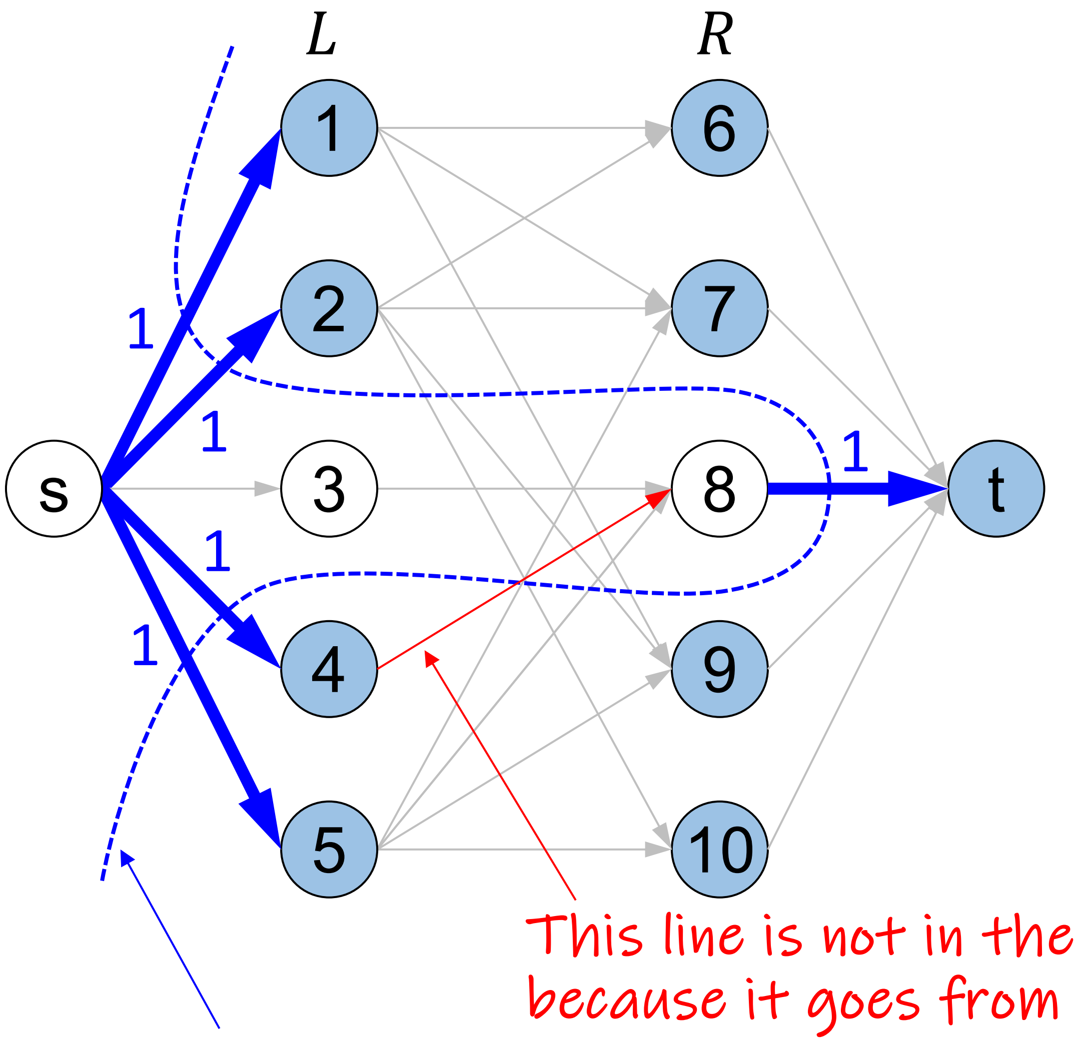
A **flow** is a mapping of edges to flows $f: E \to \mathbb{R}^+$ s.t. that flows obey their capacities $f_{uv} \leq c_{uv}$ and conservation laws. The **value** $|f|$ of a flow is the amount moved from S to T through the network.

An **s-t cut** $C = (S, T)$ is a partition of V s.t. $s \in S$ and $t \in T$. The **cut-set** X_C of a cut C is the set of edges that connect the source part of the cut to the sink part. The **capacity** $c(S, T)$ of an s-t cut is the sum of the capacities of the edges in its cut-set.

Nodes to the left of the dashed line are in S, the rest in T.

Cuts and Flows in directed graphs $G = (V, E)$

Each edge (u, v) has a capacity $c_{uv} = 1$ which is the max amount of flow that can pass through it.



A **flow** is a mapping of edges to flows $f: E \rightarrow \mathbb{R}^+$ s.t. that flows obey their capacities $f_{uv} \leq c_{uv}$ and conservation laws. The **value** $|f|$ of a flow is the amount moved from S to T through the network.

An **s-t cut** $C = (S, T)$ is a partition of V s.t. $s \in S$ and $t \in T$. The **cut-set** X_C of a cut C is the set of edges that connect the source part of the cut to the sink part. The **capacity** $c(S, T)$ of an s-t cut is the sum of the capacities of the edges in its cut-set.

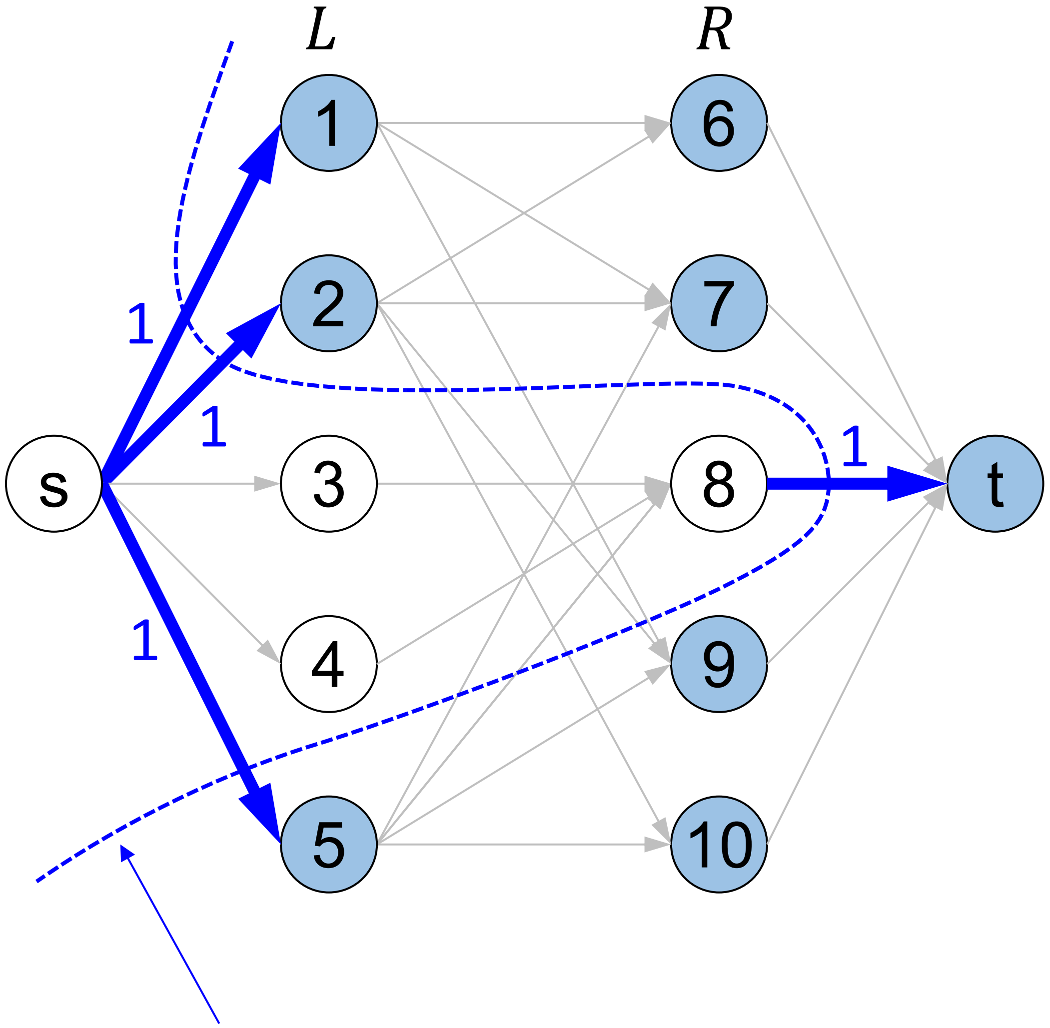
$c(S, T) = 5$

This line is not in the cut-set because it goes from T to S!

Nodes to the left of the dashed line are in S, the rest in T.

Cuts and Flows in directed graphs $G = (V, E)$

Each edge (u, v) has a capacity $c_{uv} = 1$ which is the max amount of flow that can pass through it.



A **flow** is a mapping of edges to flows $f: E \rightarrow \mathbb{R}^+$ s.t. that flows obey their capacities $f_{uv} \leq c_{uv}$ and conservation laws. The **value** $|f|$ of a flow is the amount moved from S to T through the network.

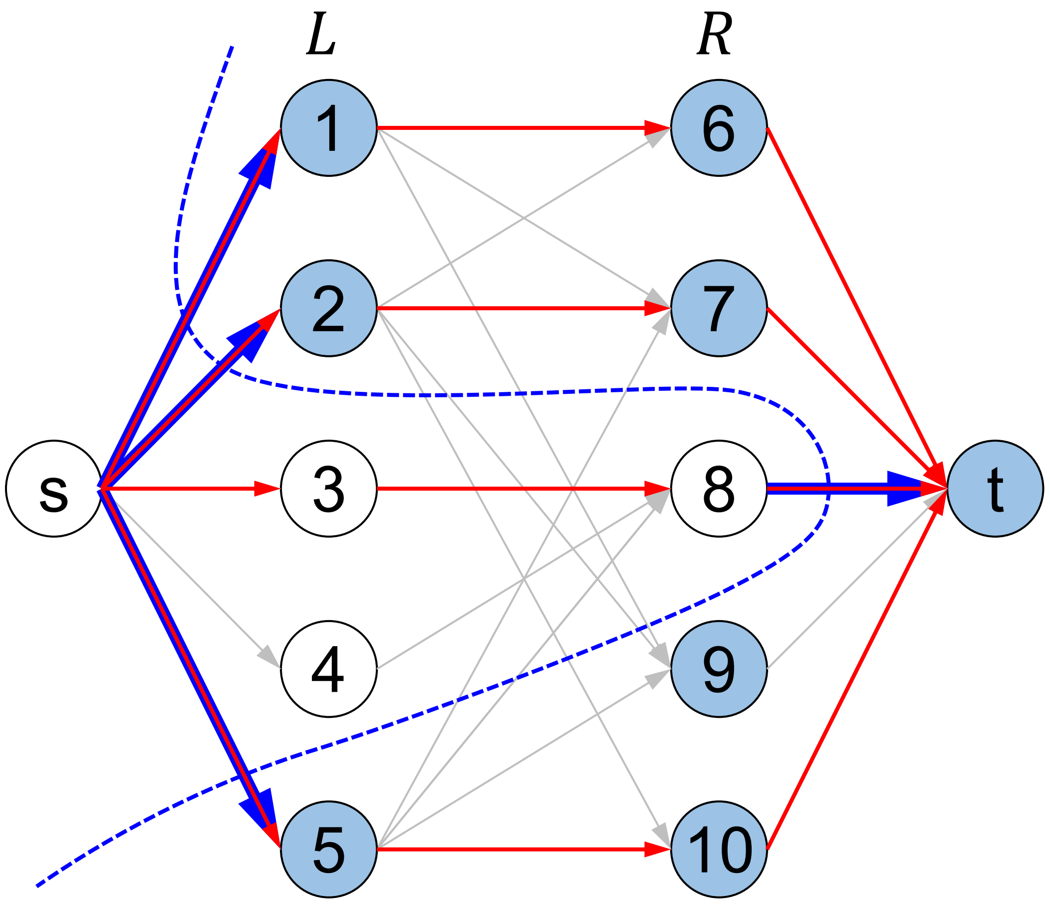
An **s-t cut** $C = (S, T)$ is a partition of V s.t. $s \in S$ and $t \in T$. The **cut-set** X_C of a cut C is the set of edges that connect the source part of the cut to the sink part. The **capacity** $c(S, T)$ of an s-t cut is the sum of the capacities of the edges in its cut-set.

$c(S,T) = 4$

Nodes to the left of the dashed line are in S, the rest in T.

Cuts and Flows in directed graphs $G = (V, E)$

Each edge (u, v) has a capacity $c_{uv} = 1$ which is the max amount of flow that can pass through it.



A **flow** is a mapping of edges to flows $f: E \rightarrow \mathbb{R}^+$ s.t. that flows obey their capacities $f_{uv} \leq c_{uv}$ and conservation laws. The **value** $|f|$ of a flow is the amount moved from S to T through the network.

$|f| = 4$

An **s-t cut** $C = (S, T)$ is a partition of V s.t. $s \in S$ and $t \in T$. The **cut-set** X_C of a cut C is the set of edges that connect the source part of the cut to the sink part. The **capacity** $c(S, T)$ of an s-t cut is the sum of the capacities of the edges in its cut-set.

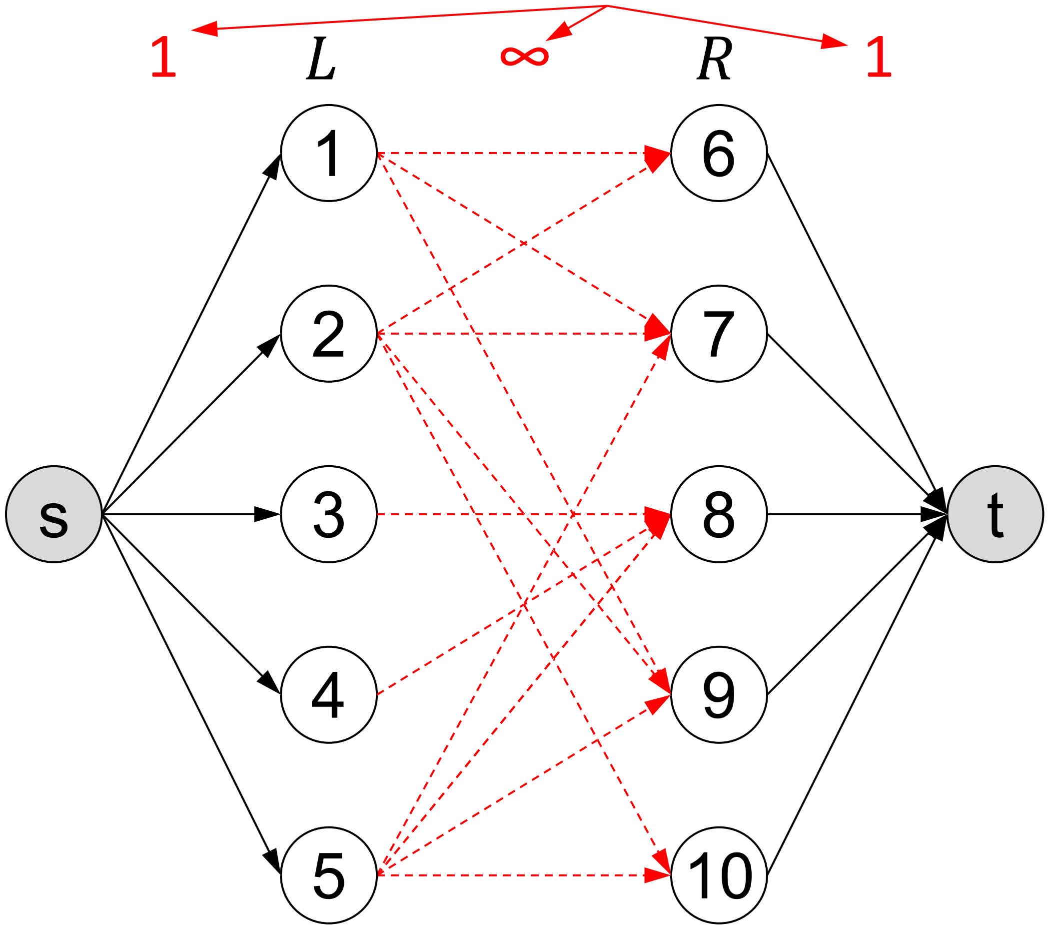
$c(S, T) = 4$

MAX-FLOW MIN-CUT THEOREM.
The maximum value of an s-t flow is equal to the minimum capacity over all s-t cuts.

Definitions adapted from: https://en.wikipedia.org/wiki/Max-flow_min-cut_theorem

Proof Kőnig-Egeváry: outline

Notice the now infinite capacities in the middle:

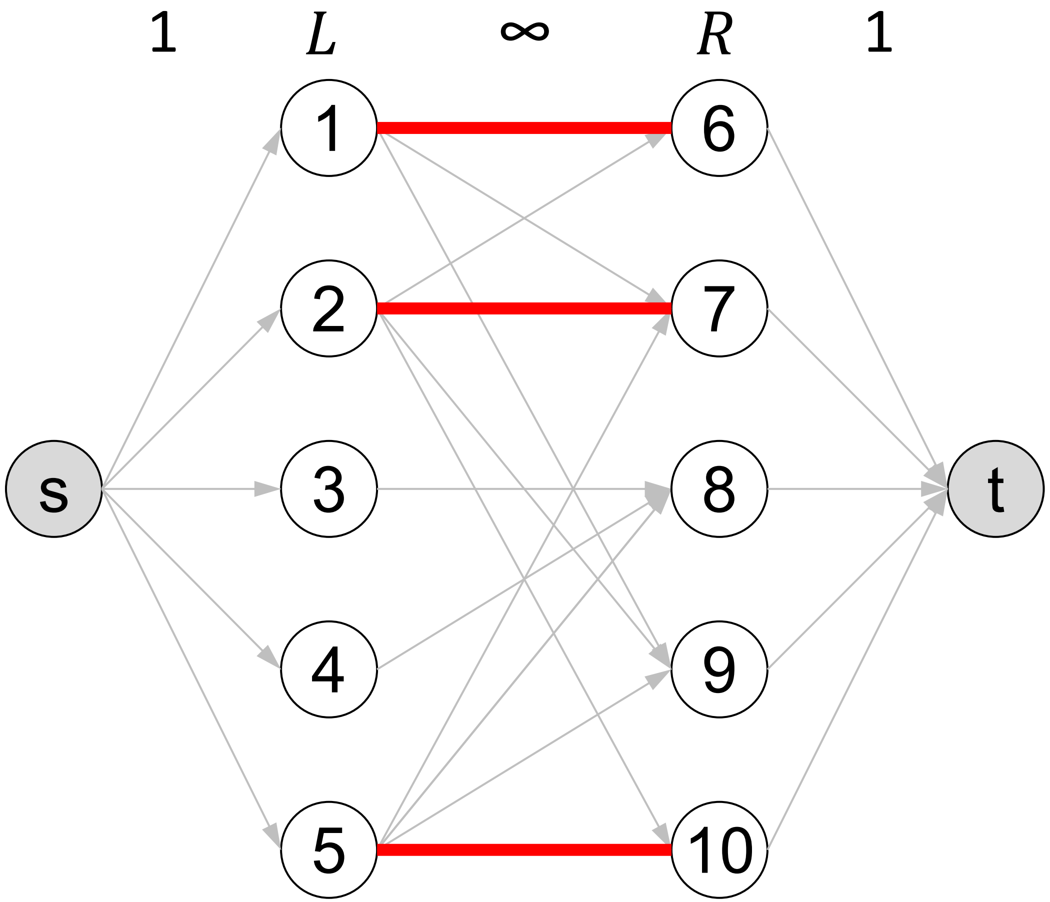


Proof outline:

Consider the flow graph to the left with capacities chosen to avoid a cut between L and R. We will show:

1. every integral flow ⇔ some matching
2. every (finite capacity) cut ⇔ some VC
3. Then we know that max matching = min VC, from the max-flow min-cut theorem

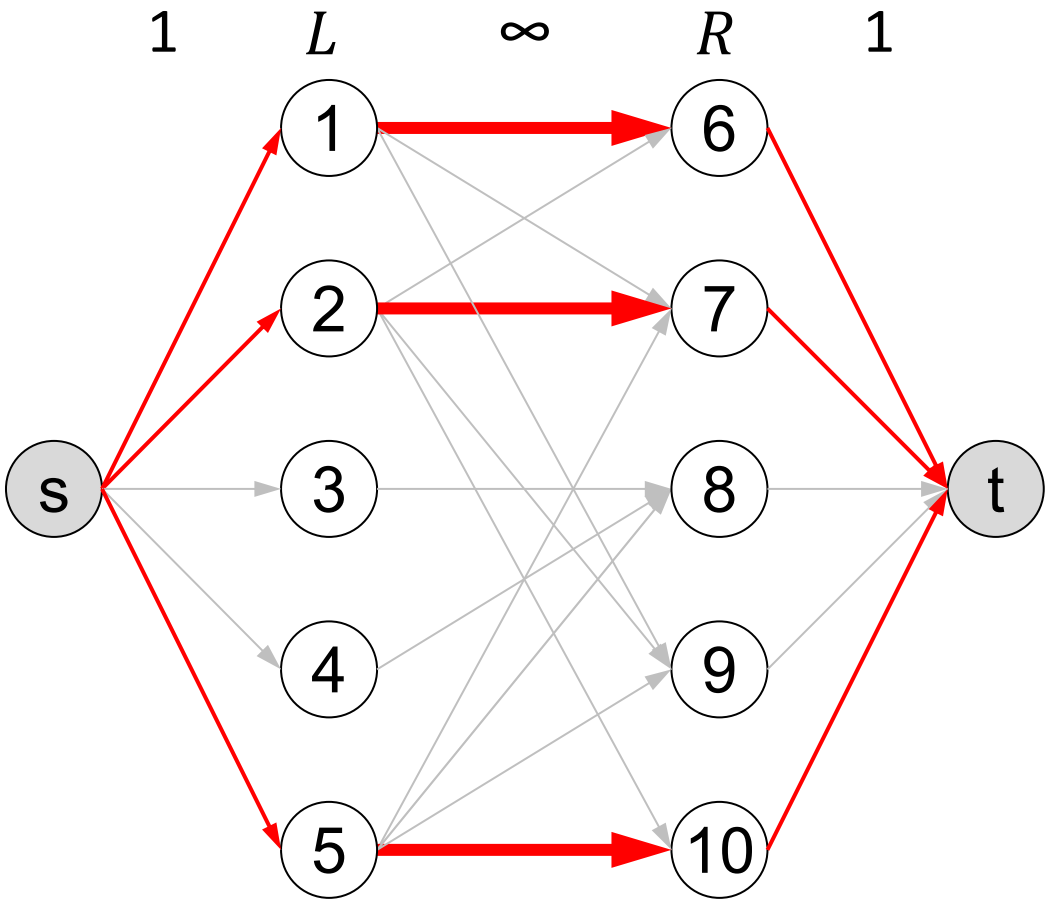
Proof Kőnig-Egeváry 1: matching = flow



1. A matching of size x corresponds to an integral flow of same value.

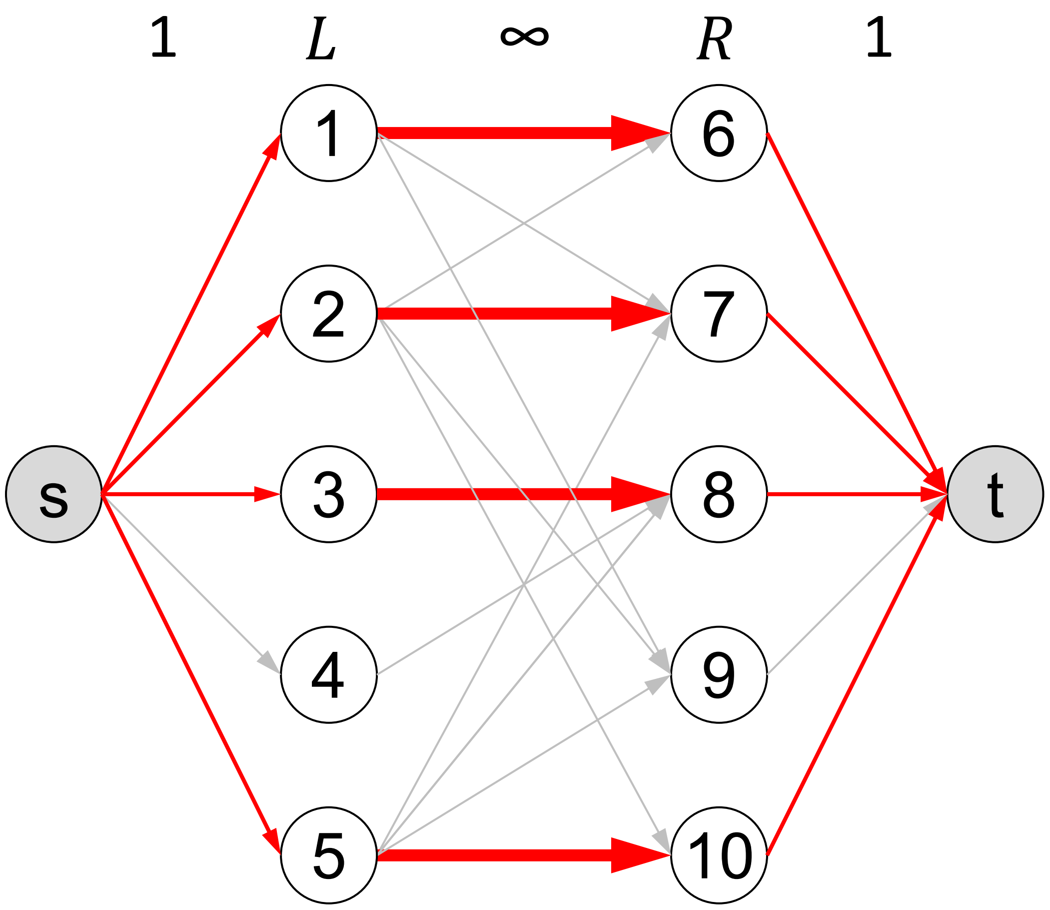
#VC = 5

Proof Kőnig-Egeváry 1: matching = flow



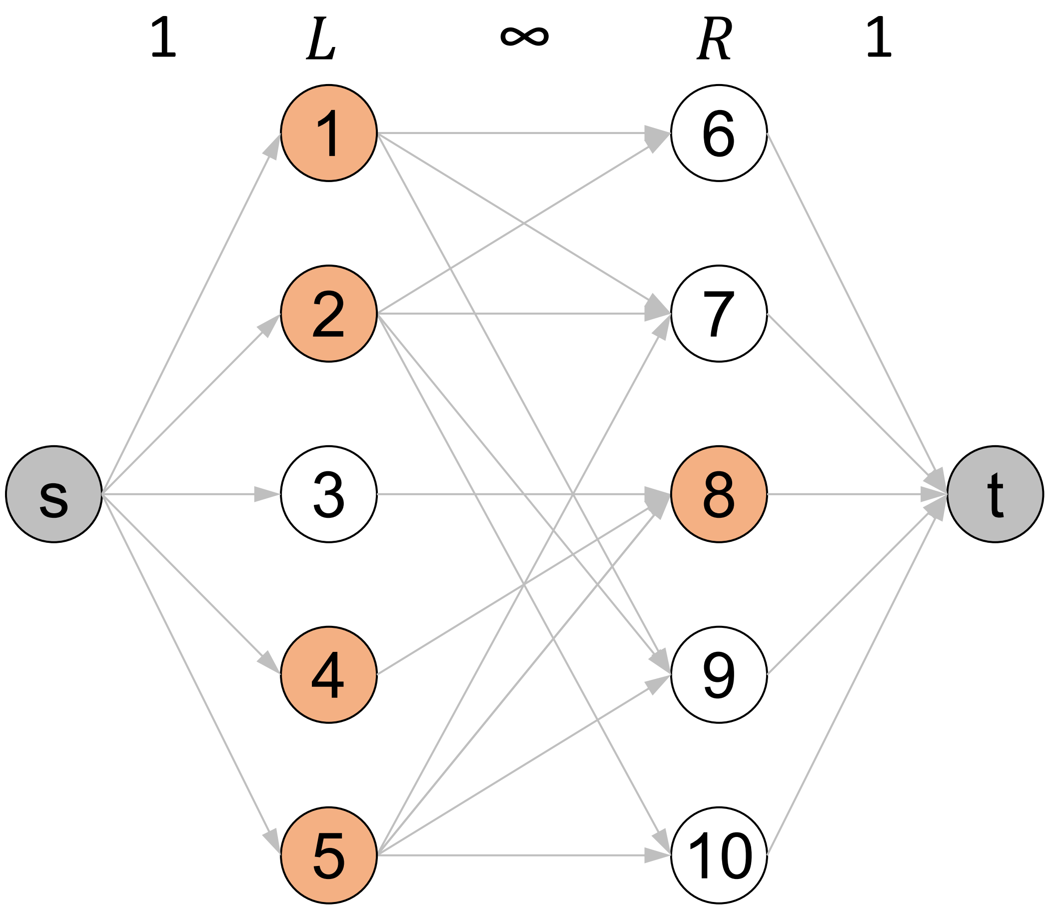
1. A matching of size x corresponds to an integral flow of same value.

Proof Kőnig-Egeváry 1: matching = flow



1. A matching of size x corresponds to an integral flow of same value.

Proof Kőnig-Egeváry 2: VC = cut



1. A matching of size x corresponds to an integral flow of same value.

2. Any VC of size x defines a cut of same capacity.

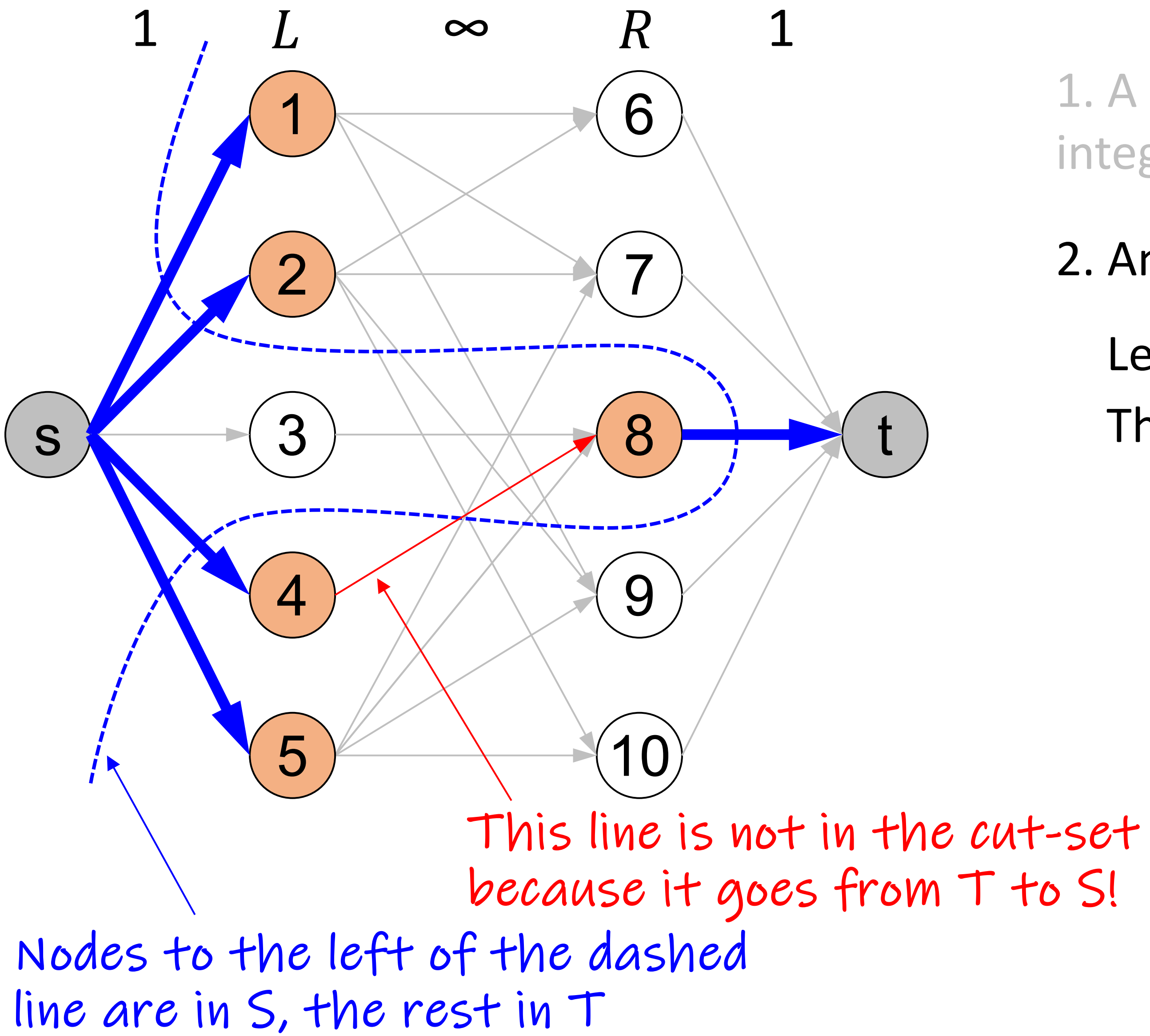
Let C be the VC, $C(L) = C \cap L$, $C(R) = C \cap R$.

Then define: $S \coloneqq \{s\} \cup \big(L - C(L)\big) \cup C(R)$

$T \coloneqq \{t\} \cup \big(R - C(R)\big) \cup C(L)$

#VC = 5

Proof Kőnig-Egeváry 2: VC = cut



1. A matching of size x corresponds to an integral flow of same value.

2. Any VC of size x defines a cut of same capacity.

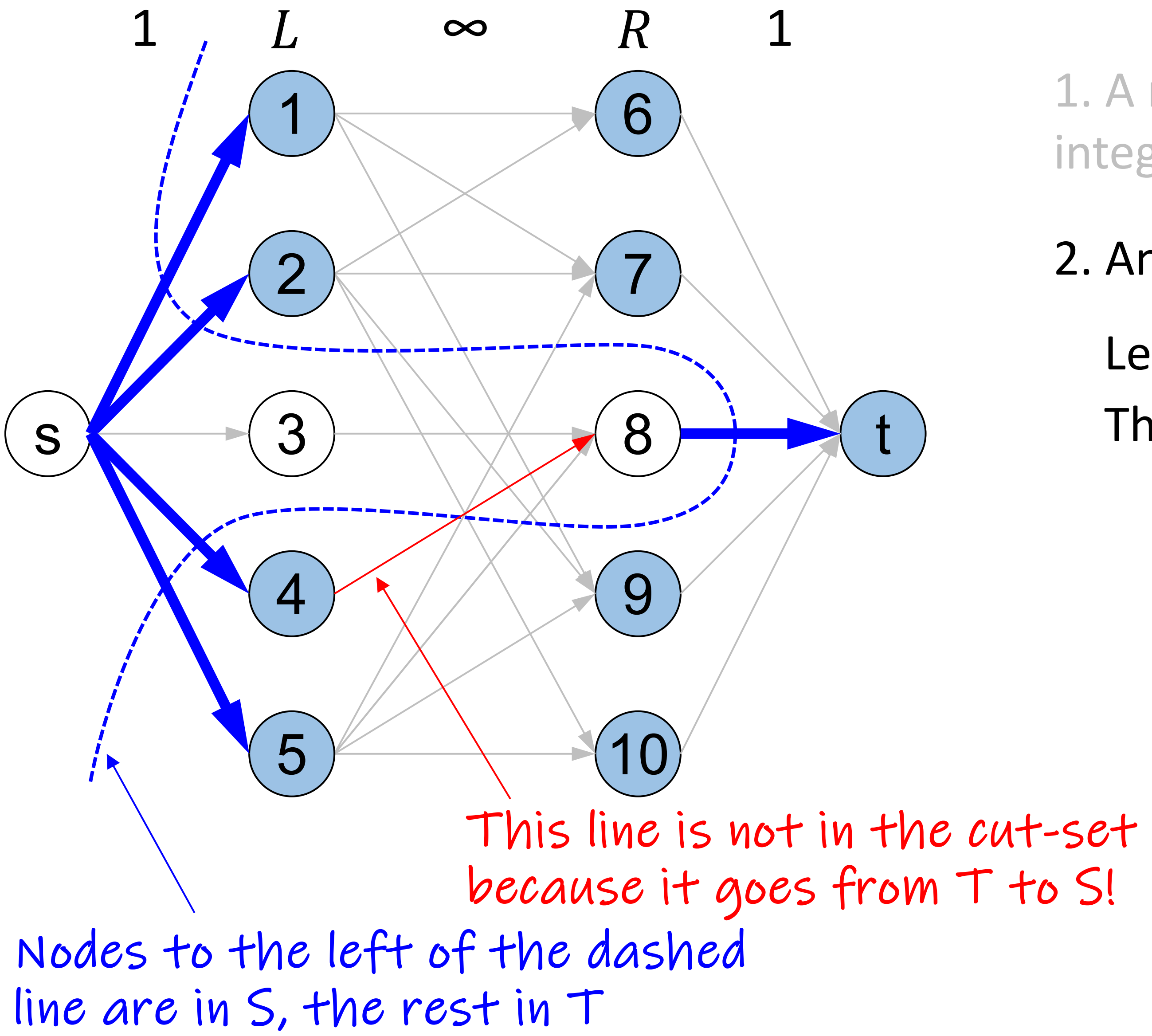
Let C be the VC, $C(L) = C \cap L$, $C(R) = C \cap R$.

Then define: $S \coloneqq \{s\} \cup \big(L - C(L)\big) \cup C(R)$

$T \coloneqq \{t\} \cup \big(R - C(R)\big) \cup C(L)$

#VC = $c(S, T) = 5$

Proof Kőnig-Egeváry 2: VC = cut



1. A matching of size x corresponds to an integral flow of same value.

2. Any VC of size x defines a cut of same capacity.

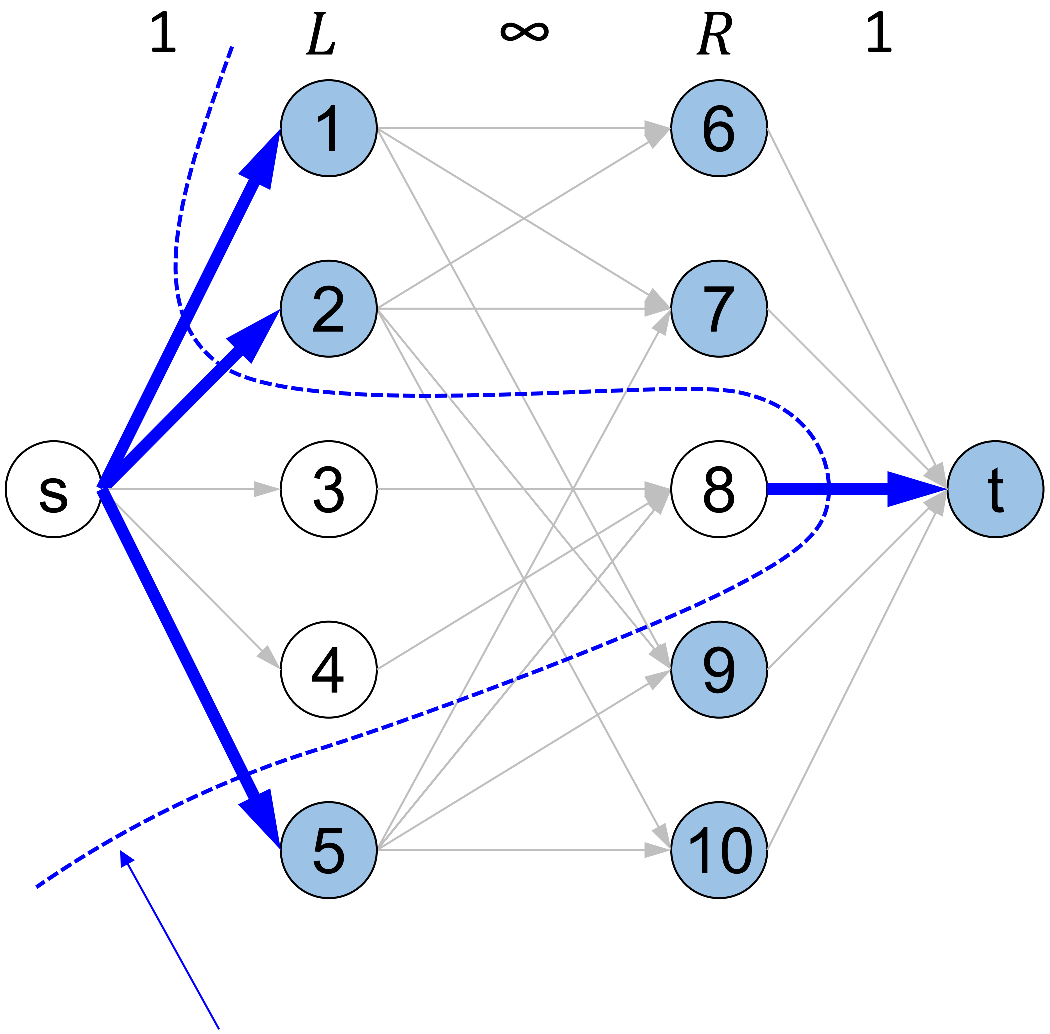
Let C be the VC, $C(L) = C \cap L$, $C(R) = C \cap R$.

Then define: $S \coloneqq \{s\} \cup \big(L - C(L)\big) \cup C(R)$

$T \coloneqq \{t\} \cup \big(R - C(R)\big) \cup C(L)$

#VC = $c(S,T) = 5$

Proof Kőnig-Egeváry 2: VC = cut



Nodes to the left of the dashed line are in S, the rest in T

1. A matching of size x corresponds to an integral flow of same value.

2. Any VC of size x defines a cut of same capacity.

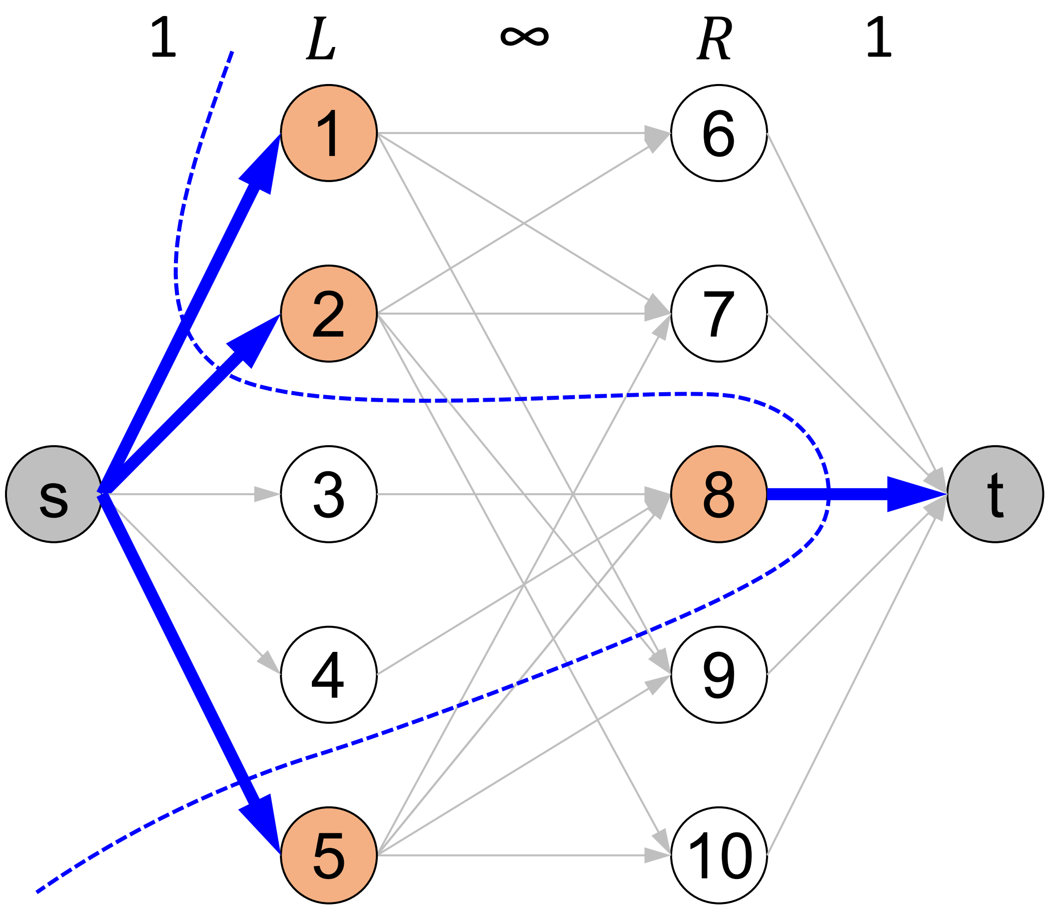
Let C be the VC, $C(L) = C \cap L$, $C(R) = C \cap R$.

Then define: $S \coloneqq \{s\} \cup \big(L - C(L)\big) \cup C(R)$

$T \coloneqq \{t\} \cup \big(R - C(R)\big) \cup C(L)$

$\#VC = c(S,T) = 4$

Proof Kőnig-Egeváry 2: VC = cut



1. A matching of size x corresponds to an integral flow of same value.

2. Any VC of size x defines a cut of same capacity.

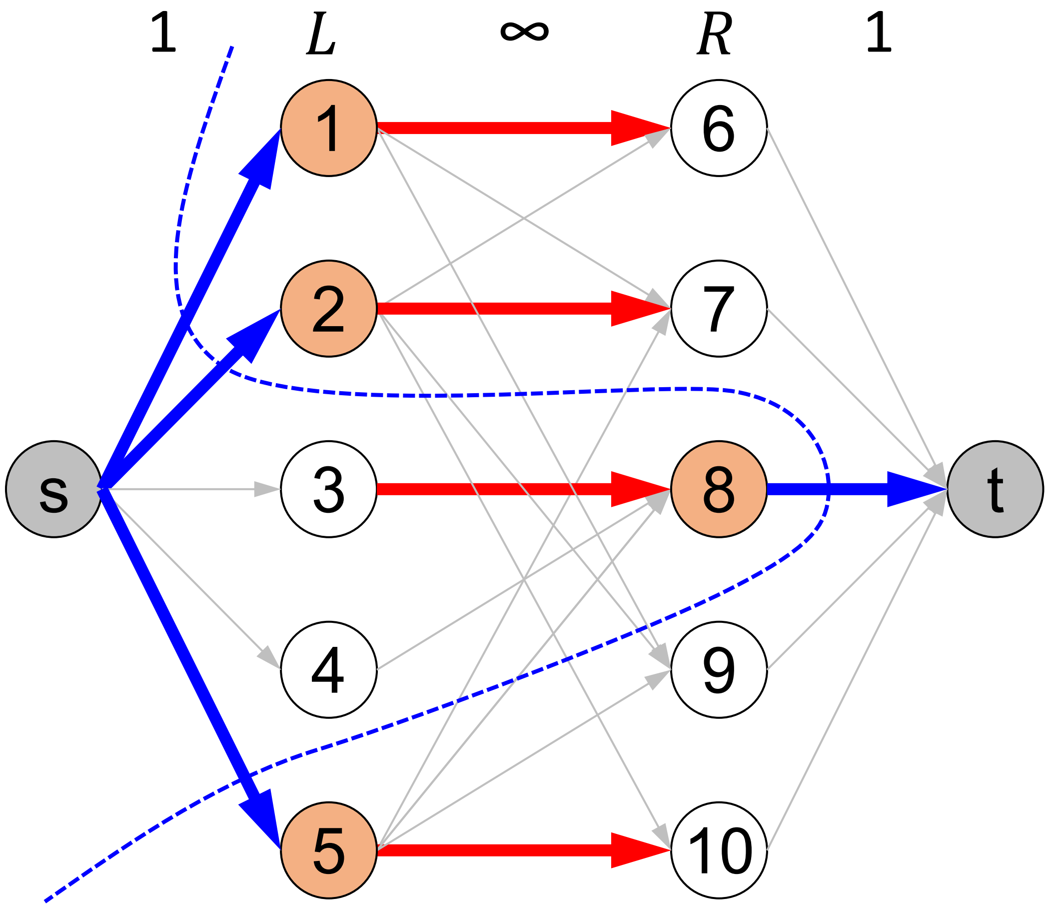
Let C be the VC, $C(L) = C \cap L$, $C(R) = C \cap R$.

Then define: $S \coloneqq \{s\} \cup \big(L - C(L)\big) \cup C(R)$

$T \coloneqq \{t\} \cup \big(R - C(R)\big) \cup C(L)$

$\#VC = c(S, T) = 4$

Proof Kőnig-Egerváry 3: max-flow = min-cut ⇒ max matching = min VC



1. A matching of size x corresponds to an integral flow of same value.

2. Any VC of size x defines a cut of same capacity.

   Let C be the VC, $C(L) = C \cap L$, $C(R) = C \cap R$.

   Then define: $S \coloneqq \{s\} \cup \big(L - C(L)\big) \cup C(R)$

   $T \coloneqq \{t\} \cup \big(R - C(R)\big) \cup C(L)$

3. Since max flow = min cut, therefore also max matching = min VC

#matching = $|f| = 4$
#VC = $c(S,T) = 4$