Fast CQ Evaluation

Complexity of **CQ**

Theorem: It holds that:

- BQE(**CQ**) is NP-complete (combined complexity)
- BQE[D](**CQ**) is NP-complete, for a fixed database D (query complexity)
- BQE[Q](**CQ**) is in LOGSPACE, for a fixed query Q ∈ **CQ** (data complexity)

Proof:

(NP-membership) Consider a database D, and a Boolean CQ Q :- body

Guess a substitution h : terms(body) → terms(D)

Verify that h is a match of Q in D, i.e., h(body) ⊆ D

(NP-hardness) Reduction from 3-colorability

(LOGSPACE-membership) Inherited from BQE[Q](**DRC**)

Complexity of **CQ**

Theorem: It holds that:

- BQE(**CQ**) is NP-complete (combined complexity)
- BQE[D](**CQ**) is NP-complete, for a fixed database D (query complexity)
- BQE[Q](**CQ**) is in LOGSPACE, for a fixed query Q ∈ **CQ** (data complexity)

Evaluating a CQ Q over a database D takes time $|D|^{O(|Q|)}$

Minimizing Conjunctive Queries

- Database theory has developed principled methods for optimizing CQs:
  - Find an equivalent CQ with minimal number of atoms (the core)
  - Provides a notion of "true" optimality

Q(x) :- R(x,y), R(x,b), R(a,b), R(u,c), R(u,v), S(a,c,d)

$\{y \mapsto b\}$

Q(x) :- R(x,b), R(a,b), R(u,c), R(u,v), S(a,c,d)

$\{v \mapsto c\}$

Q(x) :- R(x,b), R(a,b), R(u,c), S(a,c,d)

minimal query

Minimizing Conjunctive Queries

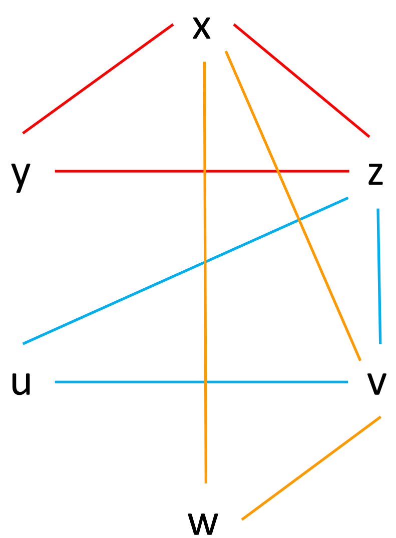
- But, a minimal equivalent CQ might not be easier to evaluate - remains NP-hard

- "Good" classes of CQs for which query evaluation is tractable (*in combined complexity*):
  - Graph-based
  - Hypergraph-based

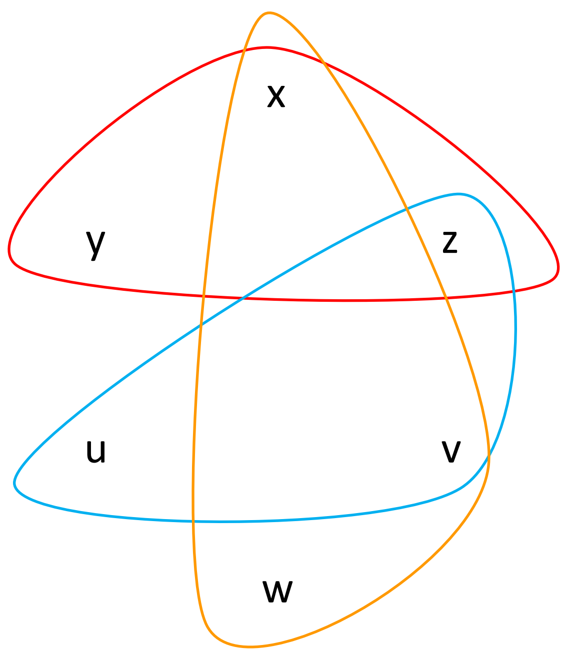
(Hyper)graph of Conjunctive Queries

Q :- R(x,y,z), R(z,u,v), R(v,w,x)

graph of Q - G(Q)



hypergraph of Q - H(Q)



“Good” Classes of Conjunctive Queries

measures how close a graph is to a tree

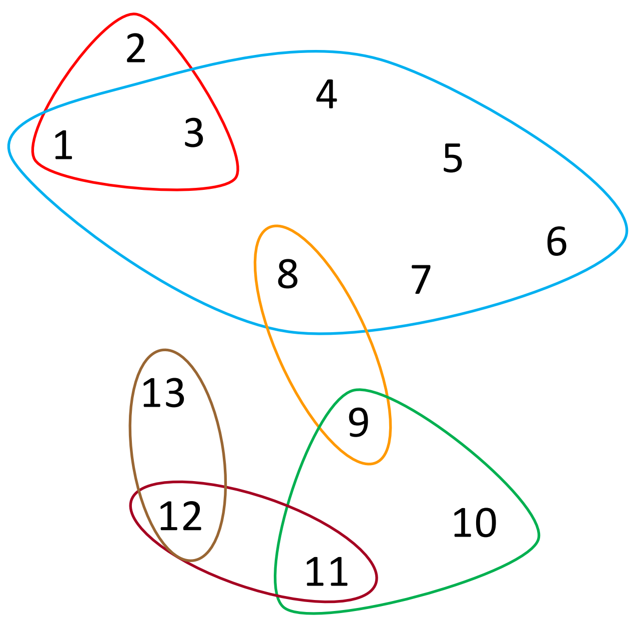
- Graph-based
  - CQs of bounded treewidth - their graph has bounded treewidth

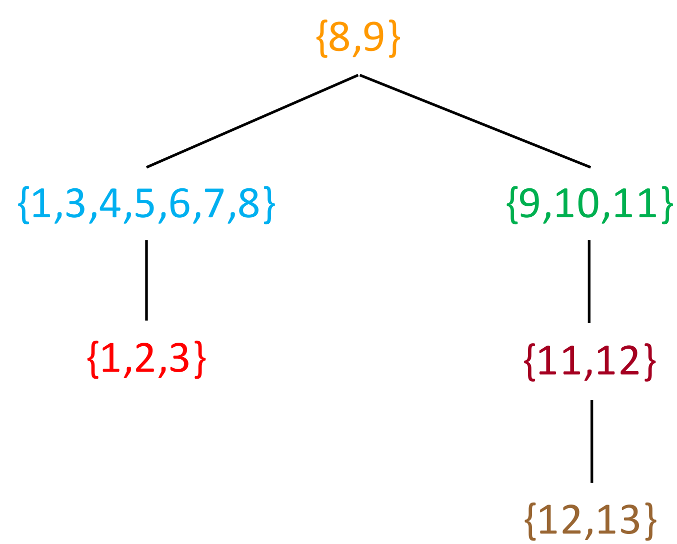
measures how close a hypergraph is to an acyclic one

- Hypergraph-based:
  - CQs of bounded hypertree width - their hypergraph has bounded hypertree width
  - **Acyclic CQs - their hypergraph has hypertree width 1**

Acyclic Hypergraphs

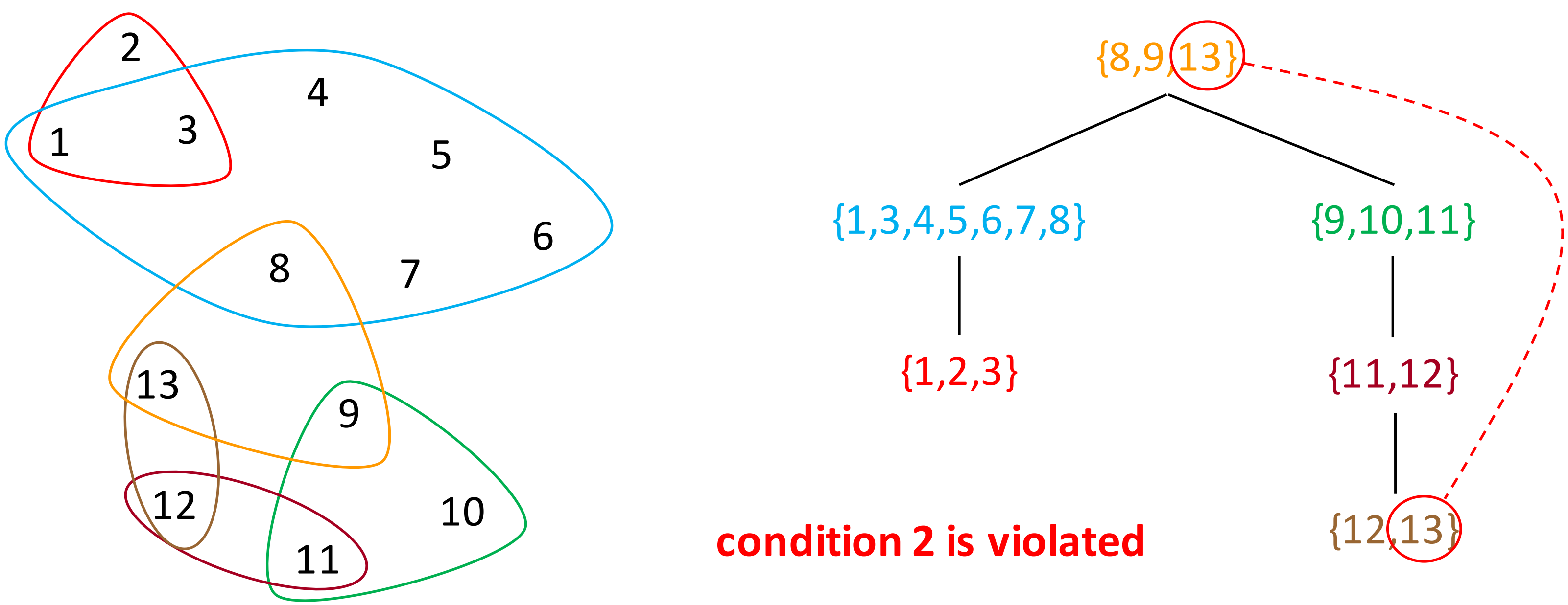
- A join tree of a hypergraph **H** = (V,E) is a labeled tree **T** = (N,F,L), where L : N → E such that:
  1. For each hyperedge e ∈ E of **H**, there exists n ∈ N such that e = L(n)
  2. For each node u ∈ V of **H**, the set {n ∈ N | u ∈ L(n)} induces a connected subtree of **T**





Acyclic Hypergraphs

- A join tree of a hypergraph **H** = (V,E) is a labeled tree **T** = (N,F,L), where L : N → E such that:
  1. For each hyperedge e ∈ E of **H**, there exists n ∈ N such that e = L(n)
  2. For each node u ∈ V of **H**, the set {n ∈ N | u ∈ λ(n)} induces a *connected* subtree of **T**

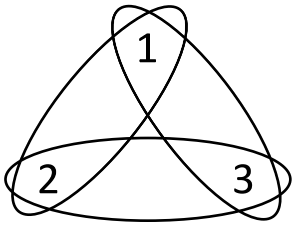


condition 2 is violated

Acyclic Hypergraphs

- A join tree of a hypergraph **H** = (V,E) is a labeled tree **T** = (N,F,L), where L : N → E such that:
    1. For each hyperedge e ∈ E of **H**, there exists n ∈ N such that e = L(n)
    2. For each node u ∈ V of **H**, the set {n ∈ N | u ∈ λ(n)} induces a *connected* subtree of **T**

- **Definition:** A hypergraph is acyclic if it has a join tree

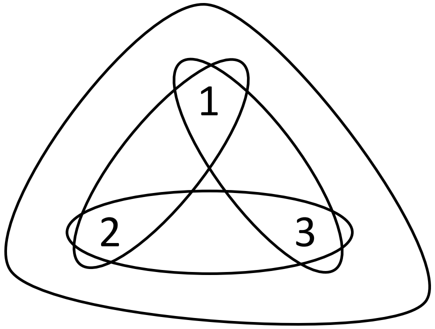


prime example of a cyclic hypergraph

Acyclic Hypergraphs

- A join tree of a hypergraph **H** = (V,E) is a labeled tree **T** = (N,F,L), where L : N → E such that:
    1. For each hyperedge e ∈ E of **H**, there exists n ∈ N such that e = L(n)
    2. For each node u ∈ V of **H**, the set {n ∈ N | u ∈ λ(n)} induces a *connected* subtree of **T**

- **Definition:** A hypergraph is acyclic if it has a join tree



but this is acyclic

Relevant Algorithmic Tasks

ACYCLICITY

Input: a query Q ∈ **CQ**

Question: is Q acyclic? or is H(Q) acyclic?

{Q ∈ **CQ** | H(Q) is acyclic}

BQE(**ACQ**)

Input: a database D, a Boolean query Q ∈ **ACQ**

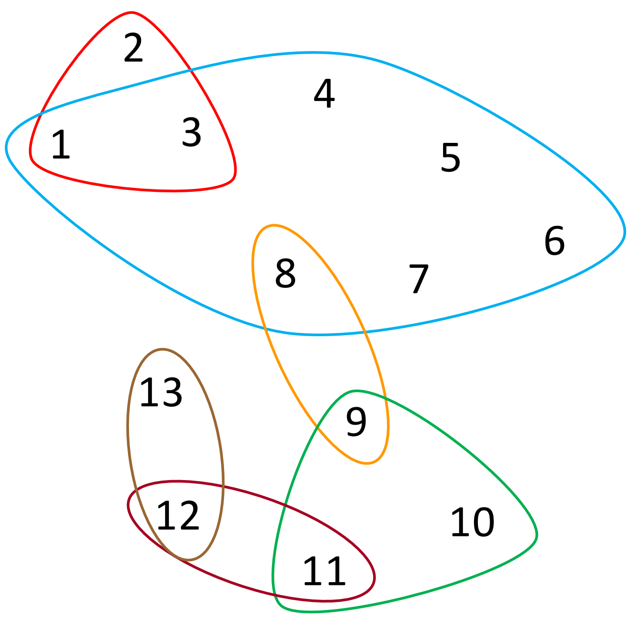
Question: is Q(D) non-empty?

Checking Acyclicity

Via the GYO-reduction (Graham, Yu and Ozsoyoglu)

1. Eliminate nodes occurring in at most one hyperedge

2. Eliminate hyperedges that are empty or contained in other hyperedges



{8,9}

{1,3,4,5,6,7,8}

{9,10,11}

{1,2,3}

{11,12}

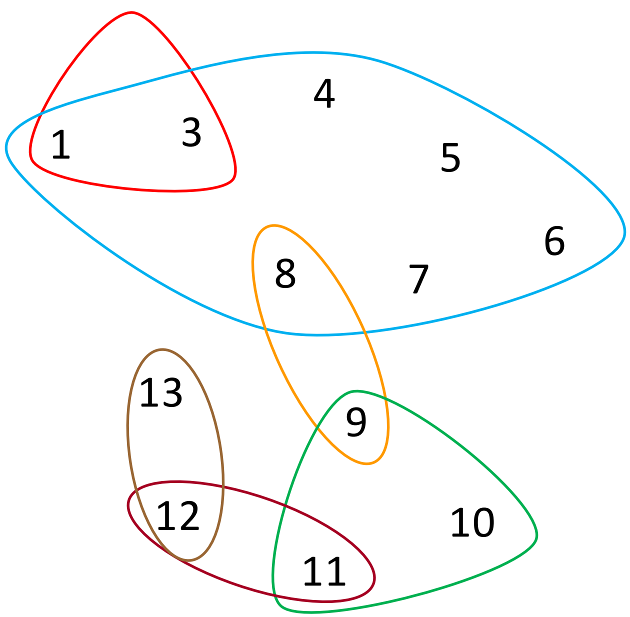
{12,13}

Checking Acyclicity

Via the GYO-reduction (Graham, Yu and Ozsoyoglu)

1. Eliminate nodes occurring in at most one hyperedge

2. Eliminate hyperedges that are empty or contained in other hyperedges



{8,9}

{1,3,4,5,6,7,8} {9,10,11}

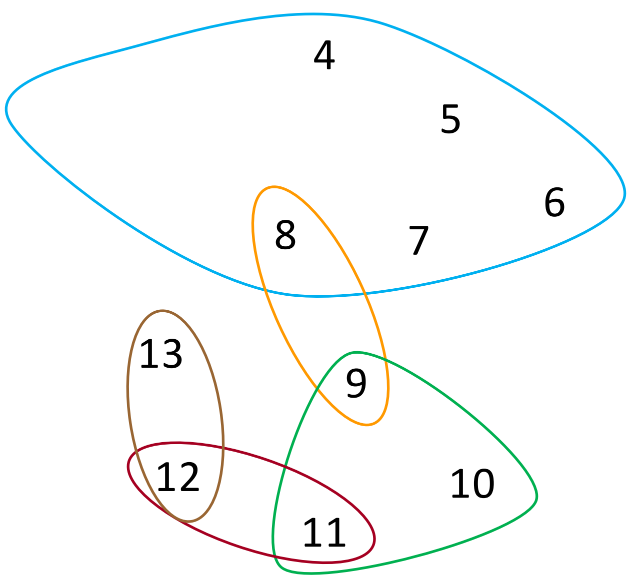
{1,2,3} {11,12}

{12,13}

Checking Acyclicity

Via the GYO-reduction (Graham, Yu and Ozsoyoglu)

1. Eliminate nodes occurring in at most one hyperedge
2. Eliminate hyperedges that are empty or contained in other hyperedges



{8,9}

{1,3,4,5,6,7,8}

{9,10,11}

{1,2,3}

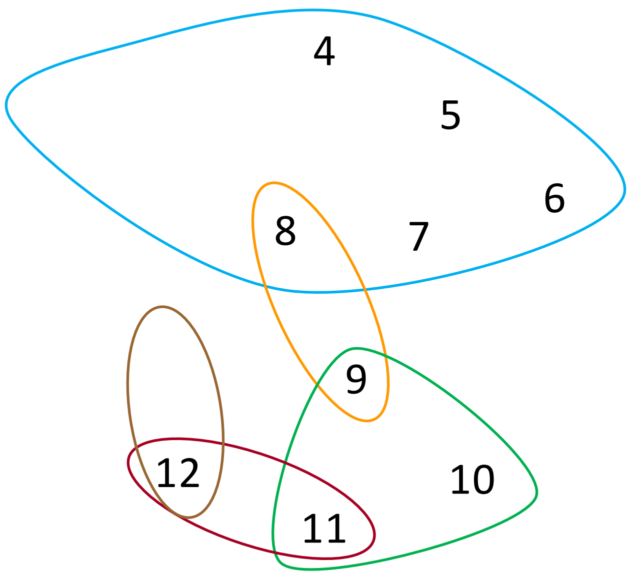
{11,12}

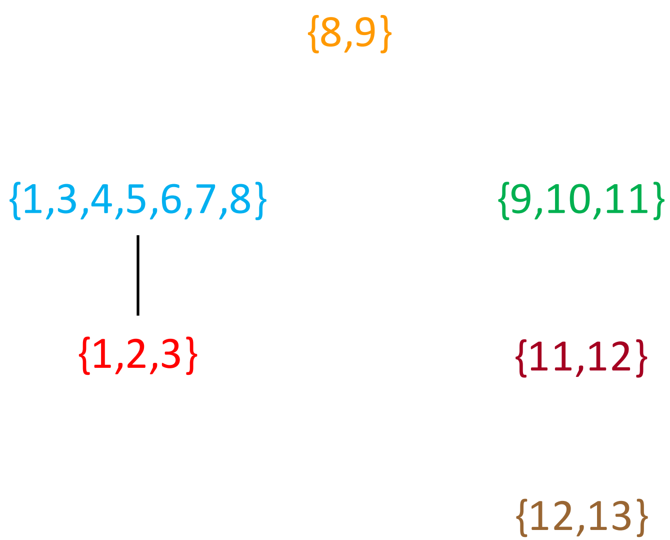
{12,13}

Checking Acyclicity

Via the GYO-reduction (Graham, Yu and Ozsoyoglu)

1. Eliminate nodes occurring in at most one hyperedge
2. Eliminate hyperedges that are empty or contained in other hyperedges



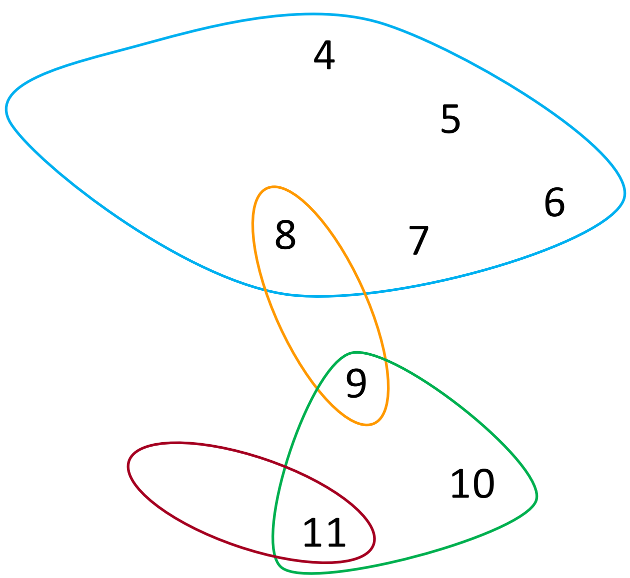


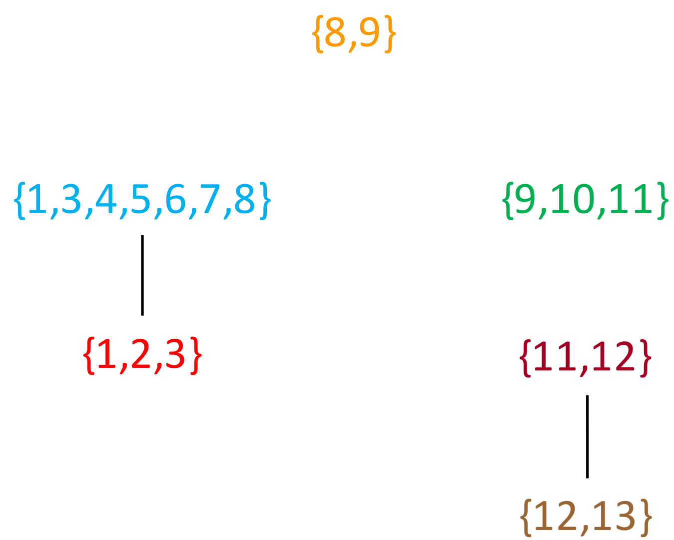
Checking Acyclicity

Via the GYO-reduction (Graham, Yu and Ozsoyoglu)

1. Eliminate nodes occurring in at most one hyperedge

2. Eliminate hyperedges that are empty or contained in other hyperedges



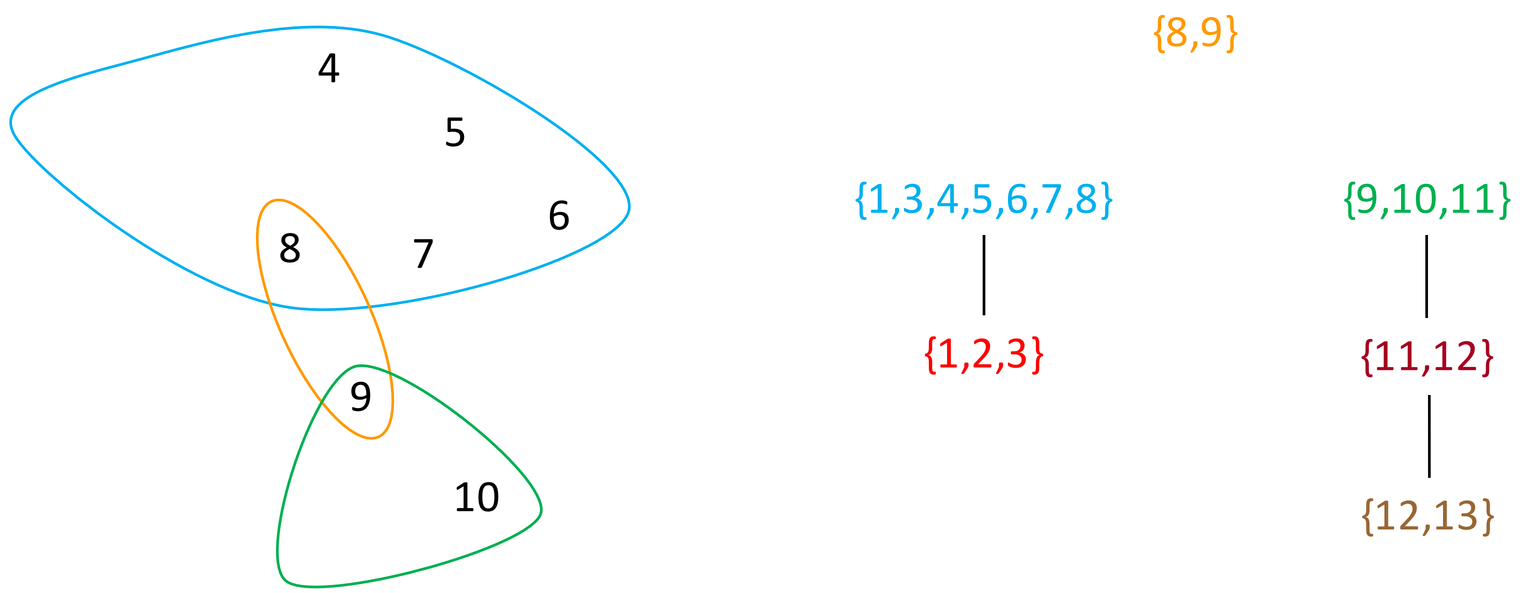


Checking Acyclicity

Via the GYO-reduction (Graham, Yu and Ozsoyoglu)

1. Eliminate nodes occurring in at most one hyperedge

2. Eliminate hyperedges that are empty or contained in other hyperedges

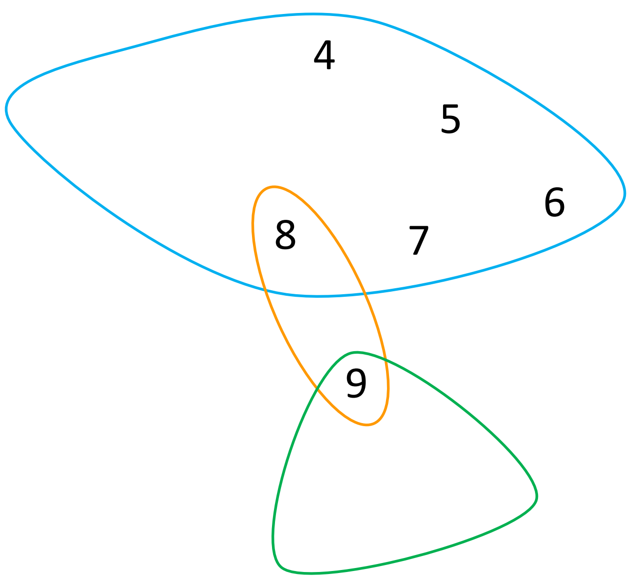


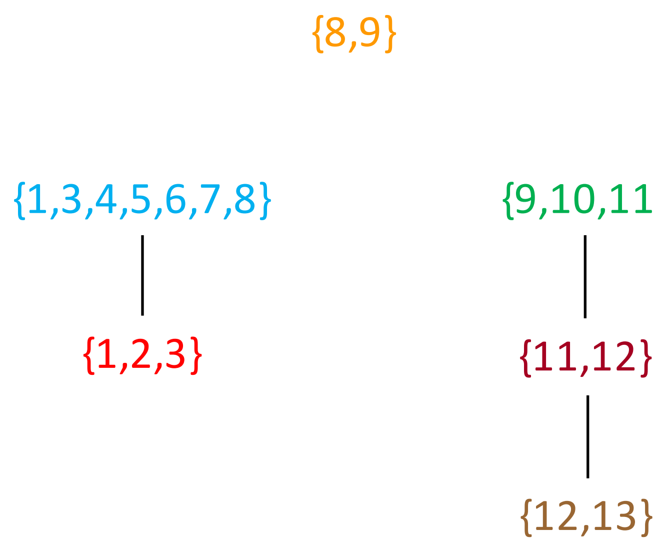
Checking Acyclicity

Via the GYO-reduction (Graham, Yu and Ozsoyoglu)

1. Eliminate nodes occurring in at most one hyperedge

2. Eliminate hyperedges that are empty or contained in other hyperedges



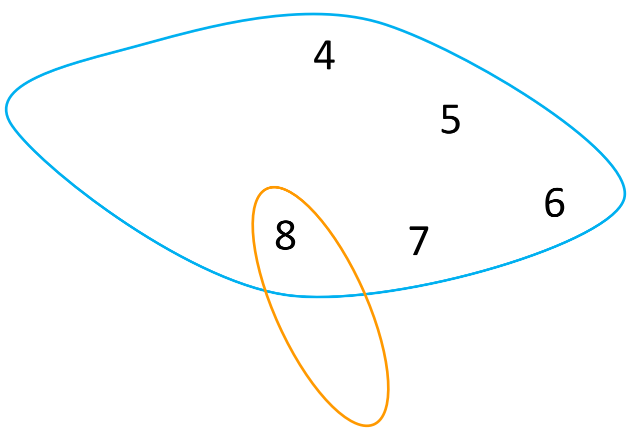


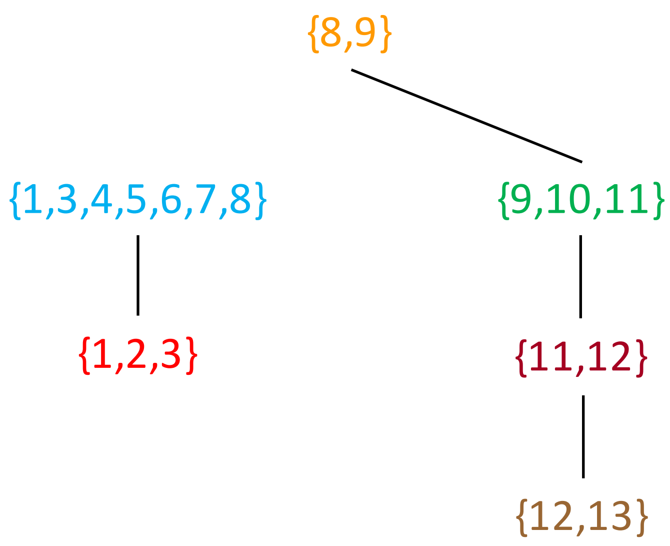
Checking Acyclicity

Via the GYO-reduction (Graham, Yu and Ozsoyoglu)

1. Eliminate nodes occurring in at most one hyperedge

2. Eliminate hyperedges that are empty or contained in other hyperedges



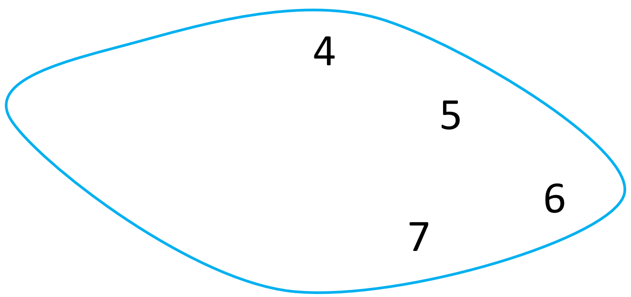


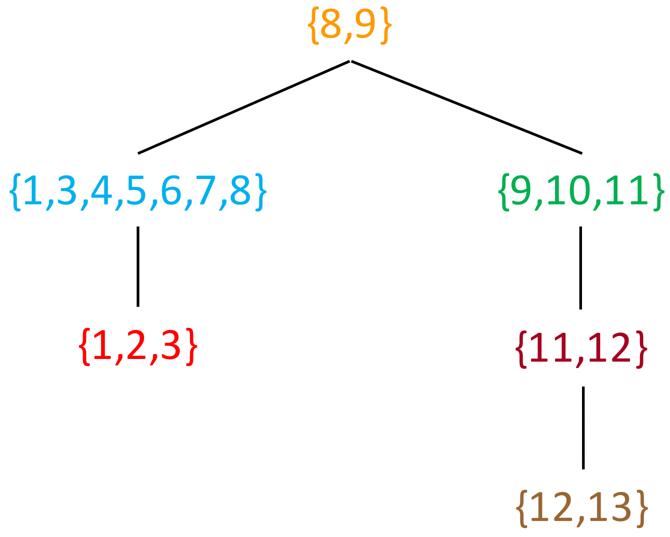
Checking Acyclicity

Via the GYO-reduction (Graham, Yu and Ozsoyoglu)

1. Eliminate nodes occurring in at most one hyperedge

2. Eliminate hyperedges that are empty or contained in other hyperedges



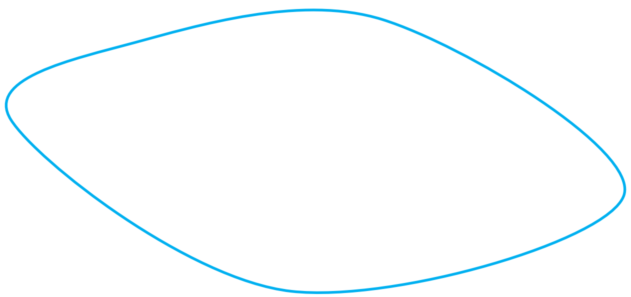


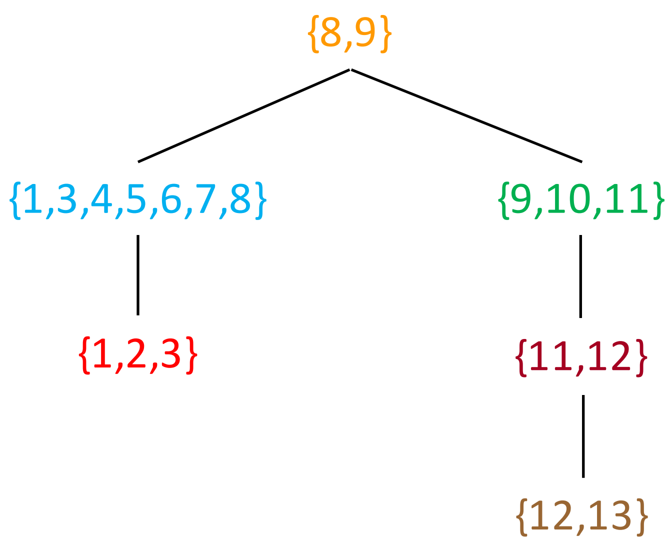
Checking Acyclicity

Via the GYO-reduction (Graham, Yu and Ozsoyoglu)

1. Eliminate nodes occurring in at most one hyperedge

2. Eliminate hyperedges that are empty or contained in other hyperedges



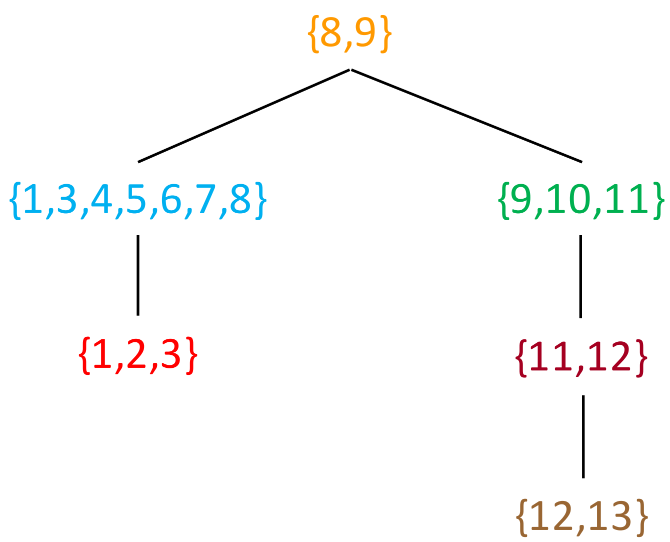


Checking Acyclicity

Via the GYO-reduction (Graham, Yu and Ozsoyoglu)

1. Eliminate nodes occurring in at most one hyperedge

2. Eliminate hyperedges that are empty or contained in other hyperedges



Checking Acyclicity

Via the GYO-reduction (Graham, Yu and Ozsoyoglu)

1. Eliminate nodes occurring in at most one hyperedge
2. Eliminate hyperedges that are empty or contained in other hyperedges

Theorem: A hypergraph **H** is acyclic iff GYO(**H**) = ∅

$\Downarrow$

checking whether **H** is acyclic is feasible in polynomial time, and if it is the case, a join tree can be found in polynomial time

$\Downarrow$

Theorem: ACYCLICITY is in PTIME

Checking Acyclicity

Theorem: ACYCLICITY is in PTIME

NOTE: actually, we can check whether a CQ is acyclic in time O(|Q|)

linear time in the size Q

Evaluating Acyclic CQs

Theorem: BQE(**ACQ**) is in PTIME

NOTE: actually, if H(Q) is acyclic, then Q can be evaluated in time O(|D| · |Q|)

linear time in the size of D and Q

Yannakaki's Algorithm

Dynamic programming algorithm over the join tree

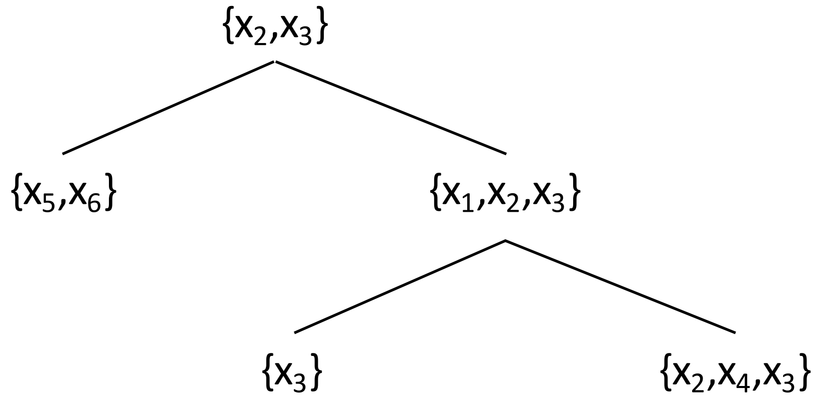
Given a database D, and an acyclic Boolean CQ Q

1. Compute the join tree **T** of H(Q)
2. Assign to each node of **T** the corresponding relation of D
3. Compute semi-joins in a bottom up traversal of **T**
4. Return YES if the resulting relation at the root of **T** is non-empty;

   otherwise, return NO

Yannakaki's Algorithm: Step 1

Q :- $R_1(x_1,x_2,x_3)$, $R_2(x_2,x_3)$, $R_2(x_5,x_6)$, $R_3(x_3)$, $R_4(x_2,x_4,x_3)$



Yannakaki's Algorithm: Step 2

x_2	x_3
c_1	b_2
c_1	b_1
c_4	b_6

x_5	x_6
c_1	b_2
c_1	b_1
c_4	b_6

x_1	x_2	x_3
s_1	c_1	b_1
s_1	c_1	b_2
s_3	c_3	b_1
s_3	c_1	b_4
s_2	c_2	b_3

x_3
b_1
b_2

x_2	x_4	x_3
c_1	a_1	b_1
c_1	a_1	b_2
c_1	a_2	b_2

Yannakaki's Algorithm: Step 3

x_2	x_3
c_1	b_2
c_1	b_1
c_4	b_6

x_5	x_6
c_1	b_2
c_1	b_1
c_4	b_6

x_1	x_2	x_3
s_1	c_1	b_1
s_1	c_1	b_2
s_3	c_3	b_1
$\cancel{s_3}$	$\cancel{c_1}$	$\cancel{b_4}$
s_2	c_2	b_3

x_3
b_1
b_2

x_2	x_4	x_3
c_1	a_1	b_1
c_1	a_1	b_2
c_1	a_2	b_2

Yannakaki's Algorithm: Step 3

x_2	x_3
c_1	b_2
c_1	b_1
c_4	b_6

x_5	x_6
c_1	b_2
c_1	b_1
c_4	b_6

x_1	x_2	x_3
s_1	c_1	b_1
s_1	c_1	b_2
s_3	c_3	b_1
$\cancel{s_3}$	$\cancel{c_1}$	$\cancel{b_4}$
$\cancel{s_2}$	$\cancel{c_2}$	$\cancel{b_3}$

x_3
b_1
b_2

x_2	x_4	x_3
c_1	a_1	b_1
c_1	a_1	b_2
c_1	a_2	b_2

Yannakaki’s Algorithm: Step 3

x_2	x_3
c_1	b_2
c_1	b_1
c_4	b_6

x_5	x_6
c_1	b_2
c_1	b_1
c_4	b_6

x_1	x_2	x_3
s_1	c_1	b_1
s_1	c_1	b_2
~~s_3~~	~~c_3~~	~~b_1~~
~~s_3~~	~~c_1~~	~~b_4~~
~~s_2~~	~~c_2~~	~~b_3~~

x_3
b_1
b_2

x_2	x_4	x_3
c_1	a_1	b_1
c_1	a_1	b_2
c_1	a_2	b_2

Yannakaki's Algorithm: Step 3

x_2	x_3
c_1	b_2
c_1	b_1
$\cancel{c_4}$	$\cancel{b_6}$

x_5	x_6
c_1	b_2
c_1	b_1
c_4	b_6

x_1	x_2	x_3
s_1	c_1	b_1
s_1	c_1	b_2
$\cancel{s_3}$	$\cancel{c_3}$	$\cancel{b_1}$
$\cancel{s_3}$	$\cancel{c_1}$	$\cancel{b_4}$
$\cancel{s_2}$	$\cancel{c_2}$	$\cancel{b_3}$

x_3
b_1
b_2

x_2	x_4	x_3
c_1	a_1	b_1
c_1	a_1	b_2
c_1	a_2	b_2

Yannakaki's Algorithm: Step 4

x_2	x_3
c_1	b_2
c_1	b_1
~~c_4~~	~~b_6~~

YES

x_5	x_6
c_1	b_2
c_1	b_1
c_4	b_6

x_1	x_2	x_3
s_1	c_1	b_1
s_1	c_1	b_2
~~s_3~~	~~c_3~~	~~b_1~~
~~s_3~~	~~c_1~~	~~b_4~~
~~s_2~~	~~c_2~~	~~b_3~~

x_3
b_1
b_2

x_2	x_4	x_3
c_1	a_1	b_1
c_1	a_1	b_2
c_1	a_2	b_2

Acyclic CQs: Recap

ACYCLICITY

Input: a query Q ∈ **CQ**

Question: is Q acyclic? or is H(Q) acyclic?

BQE(**ACQ**)

Input: a database D, a Boolean query Q ∈ **ACQ**

Question: is Q(D) non-empty?

both problems are feasible in linear time

Query Optimization

Replace a given CQ with one that is much faster to execute

or

Replace a given CQ with one that falls in a “good” class of CQs

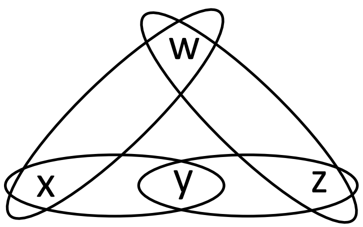
preferably, with an acyclic CQ

since evaluation is in linear time

Semantic Acyclicity

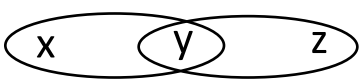
Definition: A CQ Q is semantically acyclic if there exists an acyclic CQ Q′ such that Q ≡ Q′

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



{w ↦ y, z ↦ y}

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



Relevant Algorithmic Tasks

SemACYCLICITY

Input: a query $Q \in$ **CQ**

Question: is there an acyclic CQ Q' such that $Q \equiv Q'$?

$\{Q \in$ **CQ** $\mid Q$ semantically acyclic$\}$

BQE(**SACQ**)

Input: a database D, a Boolean query $Q \in$ **SACQ**

Question: is $Q(D)$ non-empty?

Checking Semantic Acyclicity

Theorem: A CQ Q is semantically acyclic iff its core is acyclic

Theorem: SemACYCLICITY is NP-complete

Proof idea (upper bound):

- If Q is semantically acyclic, then there exists an acyclic CQ Q' such that $|Q'| \leq |Q|$ and $Q \equiv Q'$ (why?)
- Then, we can guess in polynomial time:
  - An acyclic CQ Q' such that $|Q'| \leq |Q|$
  - A mapping $h_1 : \text{terms}(Q) \rightarrow \text{terms}(Q')$
  - A mapping $h_2 : \text{terms}(Q') \rightarrow \text{terms}(Q)$
- And verify in polynomial time that h_1 is a query homomorphism from Q to Q' (i.e., $Q' \subseteq Q$), and h_2 is a query homomorphism from Q' to Q (i.e., $Q \subseteq Q'$)

Evaluating Semantically Acyclic CQs

$$f(|Q|) \quad + \quad O(|D| \cdot |Q|)$$

compute the equivalent acyclic CQ

evaluate the equivalent acyclic CQ

an improvement compare to $|D|^{O(|Q|)}$ for evaluating arbitrary CQs

Theorem: BQE(**SACQ**) is fixed-parameter tractable

Evaluating Semantically Acyclic CQs

Theorem: BQE(**SACQ**) is in PTIME

assuming Q belongs to **SACQ**: Q(D) is non-empty $\Leftrightarrow$ Q $\rightarrow_{\exists 1C}$ D

the duplicator has a winning strategy

for the existential 1-cover game,

which can be checked in polynomial time

Semantically Acyclic CQs: Recap

SemACYCLICITY

Input: a query Q ∈ **CQ**

Question: is there an acyclic CQ Q' such that Q ≡ Q'?

NP-complete - but no database is involved

BQE(**SACQ**)

Input: a database D, a Boolean query Q ∈ **SACQ**

Question: is Q(D) non-empty?

in PTIME (combined complexity)

Recap

- “Good” classes of CQs for which query evaluation is tractable - conditions based on the graph or hypergraph of the CQ

- Acyclic CQs - their hypergraph is acyclic, can be checked in linear time

- Evaluating acyclic CQs in feasible in linear time (Yannakaki’s algorithm)

- Semantic acyclicity - difficult to check, but ensures tractable evaluation