Enumeration of solutions of Conjunctive Queries with self-joins

Projet de recherche encadré M1. Supervised by Luc Segoufin, Nofar Carmeli and David Carral.

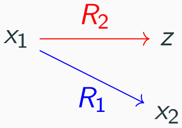
Clément Rouvroy

January 27, 2025

Introduction

Different notations for CQs

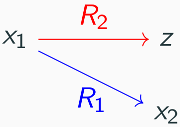
- A Conjunctive Query formally is $q(\vec{x}) : \exists \vec{z}. \bigwedge R_i(\vec{y})$, with $\vec{y} \subseteq \vec{x} \cup \vec{z}$
- Enumerating solutions of q is printing one by one every valuation of $\vec{x}$ that satisfies q without duplicates.
- We restrict to CQ with atoms of arity at most 2. Hence, we see acyclic CQ as a DAG.

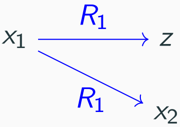


$$\exists z. R_1(x_1, x_2), R_2(x_1, z)$$

Self-join

A CQ q has a self-join if it has two atoms that use the same relational symbol.





Queries without self-joins

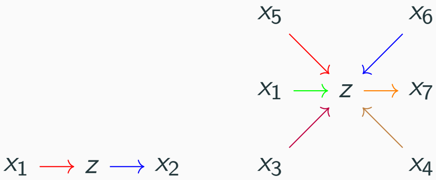
CD ∘ Lin

A CQ q is in CD ∘ Lin if we can enumerate its solutions on a database D with constant delay after a linear time (on $|D|$) preprocessing.

Result

Theorem (Bagan, Durand, and Grandjean 2007)
A CQ without self-join is in CD ∘ Lin $\Leftrightarrow^*$ *acyclic free-connex.*

An acyclic conjuctive query is free-connex ⇔ does not have a free-path.

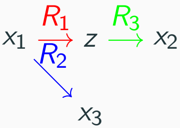


**: Assuming sBMM and sHyperclique.*

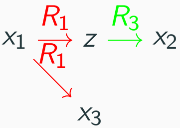
What happens with self-join ?

Example (L3)

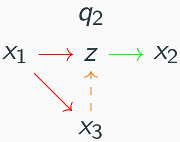
q_1 hard

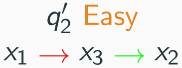


q_2 ?



Example (L3)

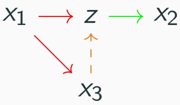




- $(a, b, c) \in q_2'(D) \Rightarrow (a, b, c) \in q_2(D)$
- Printing with constant duplicates is ok. (Carmeli and Segoufin 2022)
- Hence, we will enumerate q_2' to gain information.

Example - building the Hash Table

q_2



q_2' Easy

$x_1 \longrightarrow x_3 \longrightarrow x_2$

R

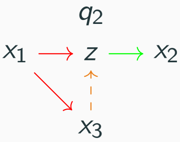
a_1	b_1
a_2	b_2
a_2	b_3

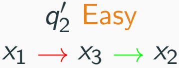
G

b_1	c_1
b_2	c_2
b_3	c_3

H

Example - building the Hash Table





R

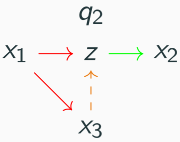
a_1	b_1
a_2	b_2
a_2	b_3

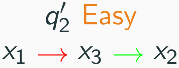
G

b_1	c_1
b_2	c_2
b_3	c_3

H

Example - building the Hash Table





R

a_1	b_1
a_2	b_2
a_2	b_3

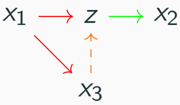
G

b_1	c_1
b_2	c_2
b_3	c_3

$$(a_1, b_1, c_1) \in q_2'(D) \cap q_2(D)$$

H

q_2



q'_2 Easy

$x_1 \longrightarrow x_3 \longrightarrow x_2$

R

a_1	b_1
a_2	b_2
a_2	b_3

G

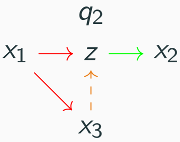
b_1	c_1
b_2	c_2
b_3	c_3

$$(a_1, b_1, c_1) \in q'_2(D) \cap q_2(D)$$

H

a_1	c_1

Example - building the Hash Table



q_2' Easy

$x_1 \longrightarrow x_3 \longrightarrow x_2$

R

a_1	b_1
a_2	b_2
a_2	b_3

G

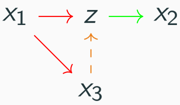
b_1	c_1
b_2	c_2
b_3	c_3

H

a_1	c_1

Example - building the Hash Table

q_2



q_2' Easy

$x_1 \longrightarrow x_3 \longrightarrow x_2$

R

a_1	b_1
a_2	b_2
a_2	b_3

G

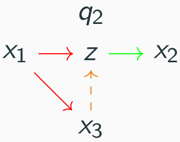
b_1	c_1
b_2	c_2
b_3	c_3

$(a_2, b_2, c_2) \in q_2'(D) \cap q_2(D)$s

H

a_1	c_1

Example - building the Hash Table



q_2' Easy

$x_1 \longrightarrow x_3 \longrightarrow x_2$

R

a_1	b_1
a_2	b_2
a_2	b_3

G

b_1	c_1
b_2	c_2
b_3	c_3

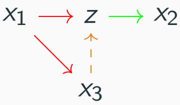
$(a_2, b_2, c_2) \in q_2'(D) \cap q_2(D)$s

H

a_1	c_1

Example - building the Hash Table

q_2



q_2' Easy

$x_1 \longrightarrow x_3 \longrightarrow x_2$

R

a_1	b_1
a_2	b_2
a_2	b_3

G

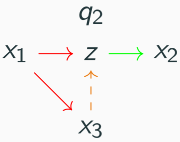
b_1	c_1
b_2	c_2
b_3	c_3

$(a_2, b_2, c_2) \in q_2'(D) \cap q_2(D)$s

H

a_1	c_1
a_2	c_2

Example - building the Hash Table



q_2' Easy

$x_1 \longrightarrow x_3 \longrightarrow x_2$

R

a_1	b_1
a_2	b_2
a_2	b_3

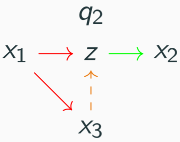
G

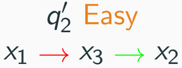
b_1	c_1
b_2	c_2
b_3	c_3

H

a_1	c_1
a_2	c_2
a_2	c_3

Example - building the Hash Table





R

a_1	b_1
a_2	b_2
a_2	b_3

G

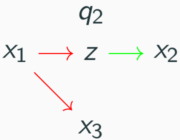
b_1	c_1
b_2	c_2
b_3	c_3

H

a_1	c_1
a_2	c_2
a_2	c_3

Using q_2', we can enumerate some solutions of q_2 and build H that contains all (a_1, a_2) solutions of the free-path.

Example - Using the Hash Table



For all $(a, c) \in H$,

R

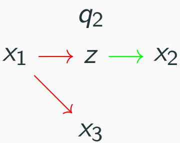
a_1	b_1
a_2	b_2
a_2	b_3

H

a_1	c_1
a_2	c_2
a_2	c_3

Solutions in $q_2(D)$

Example - Using the Hash Table



For all $(a, c) \in H$,
for all $b \in R(a, x_3)$,

R

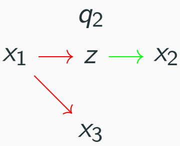
a_1	b_1
a_2	b_2
a_2	b_3

H

a_1	c_1
a_2	c_2
a_2	c_3

Solutions in $q_2(D)$

Example - Using the Hash Table



For all $(a, c) \in H$,
for all $b \in R(a, x_3)$,
output (a, b, c)

R

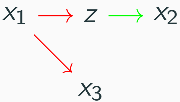
a_1	b_1
a_2	b_2
a_2	b_3

H

a_1	c_1
a_2	c_2
a_2	c_3

Solutions in $q_2(D)$

q_2



For all $(a, c) \in H$,
 for all $b \in R(a, x_3)$,
  output (a, b, c)

R

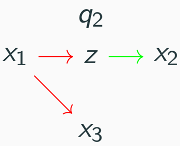
a_1	b_1
a_2	b_2
a_2	b_3

H

a_1	c_1
a_2	c_2
a_2	c_3

Solutions in $q_2(D)$

Example - Using the Hash Table



For all $(a, c) \in H$,
for all $b \in R(a, x_3)$,
output (a, b, c)

R

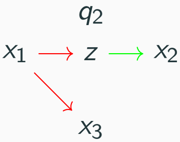
a_1	b_1
a_2	b_2
a_2	b_3

H

a_1	c_1
a_2	c_2
a_2	c_3

Solutions in $q_2(D)$

Example - Using the Hash Table



For all $(a, c) \in H$,
for all $b \in R(a, x_3)$,
output (a, b, c)

R

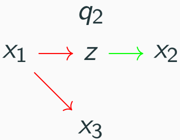
a_1	b_1
a_2	b_2
a_2	b_3

H

a_1	c_1
a_2	c_2
a_2	c_3

Solutions in $q_2(D)$

Example - Using the Hash Table



For all $(a, c) \in H$,
for all $b \in R(a, x_3)$,
output (a, b, c)

R

a_1	b_1
a_2	b_2
a_2	b_3

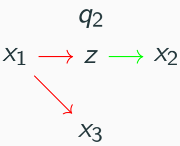
H

a_1	c_1
a_2	c_2
a_2	c_3

Solutions in $q_2(D)$

- (a_1, b_1, c_1)

Example - Using the Hash Table



For all $(a, c) \in H$,
for all $b \in R(a, x_3)$,
output (a, b, c)

R

a_1	b_1
a_2	b_2
a_2	b_3

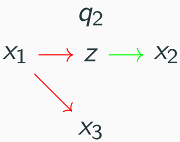
H

a_1	c_1
a_2	c_2
a_2	c_3

Solutions in $q_2(D)$

- (a_1, b_1, c_1)

Example - Using the Hash Table



For all $(a, c) \in H$,
for all $b \in R(a, x_3)$,
output (a, b, c)

R

a_1	b_1
a_2	b_2
a_2	b_3

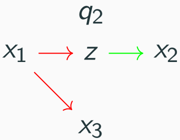
H

a_1	c_1
a_2	c_2
a_2	c_3

Solutions in $q_2(D)$

- (a_1, b_1, c_1)
- (a_2, b_2, c_2)

Example - Using the Hash Table



For all $(a, c) \in H$,
for all $b \in R(a, x_3)$,
output (a, b, c)

R

a_1	b_1
a_2	b_2
a_2	b_3

H

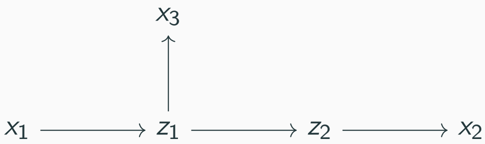
a_1	c_1
a_2	c_2
a_2	c_3

Solutions in $q_2(D)$

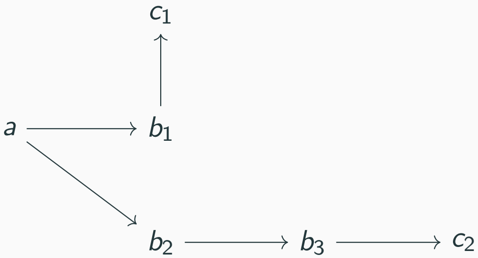
- (a_1, b_1, c_1)
- (a_2, b_2, c_2)
- (a_2, b_3, c_2)
- (a_2, b_2, c_3)
- (a_2, b_3, c_3)

Studying $CD \circ Lin$ for CQs with self-join: Tractability

Multiple Hash-Tables ?

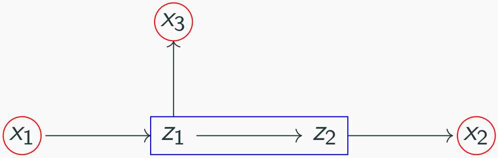


Given H_1 that gives (x_1, x_3) and H_2 that gives (x_1, x_2), we know no way to solve q: how to know if the z_1 are the same ?



Free-tree

We define free-tree to capture intersection of free-pathes.

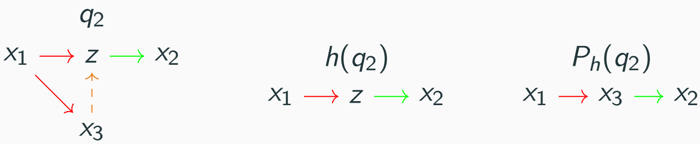


The only hard part of an acyclic CQs with self-join is its free-trees.

Theorem (Fully-patched enumeration, L3 Internship)
If you have an H for each free-trees of a CQ, you can enumerate the CQ.

Preimage: a way to get hash-tables

$P_h(q)$ has the body of $h(q)$. A variable in $P_h(q)$ is free if it is the image of at least one free variable in q_2.

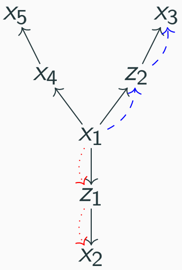


Lemma (Prehomomorphism lemma, L3 Internship)
Each solution of $P_h(q)$ gives a unique solution of q to print. So we can use the solutions of a preimage in a $CD \circ Lin$ algorithm.

If a preimage gives us the value of a free-tree, we say that it patches the free-tree.

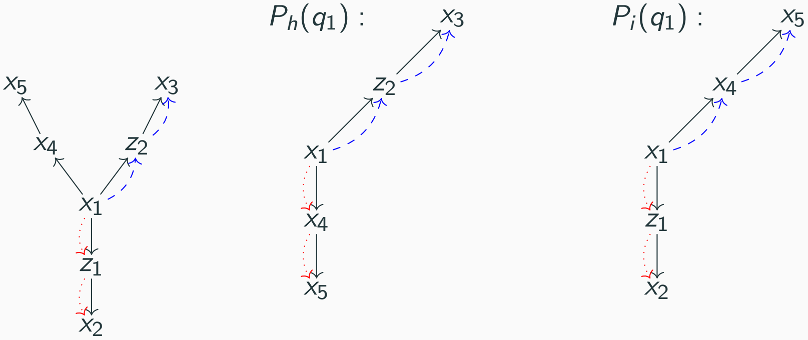
Patching with deactivated preimage

This semester, we have found this example:



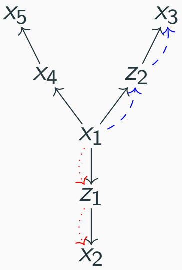
Patching with deactivated preimage

This semester, we have found this example:

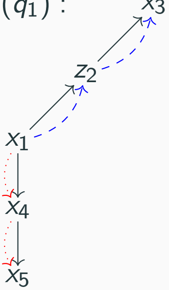


Patching with deactivated preimage

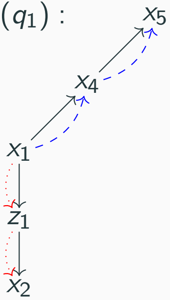
This semester, we have found this example:



$P_h(q_1)$:

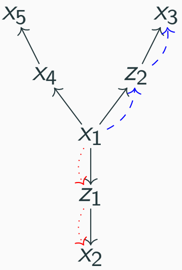


$P_i(q_1)$:

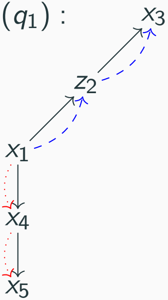


There is no free-connex preimages...

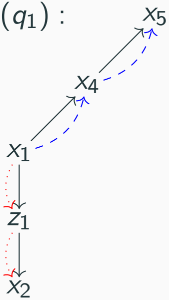
This semester, we have found this example:



$P_h(q_1)$:



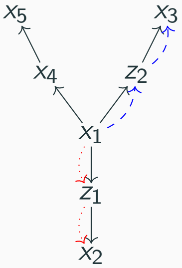
$P_i(q_1)$:



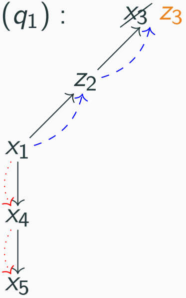
There is no free-connex preimages, but we don't need the hard part in them !

Patching with deactivated preimage

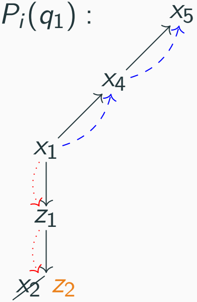
This semester, we have found this example:



$\tilde{P}_h(q_1)$:



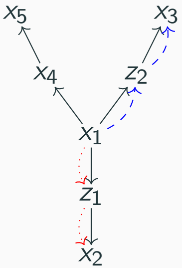
$\tilde{P}_i(q_1)$:



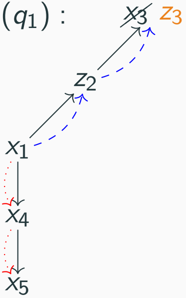
There is no free-connex preimages, but there is free-connex deactivated preimages.

Patching with deactivated preimage

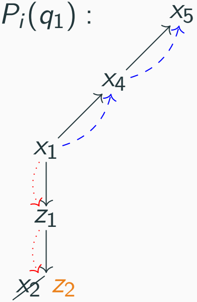
This semester, we have found this example:



$\tilde{P}_h(q_1)$:



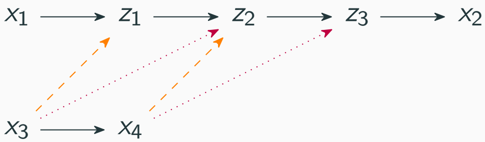
$\tilde{P}_i(q_1)$:



There is no free-connex preimages. Use $\tilde{P}_h(q_1)$ to patch the first free-tree, and $\tilde{P}_i(q_1)$ to patch the second free-tree.

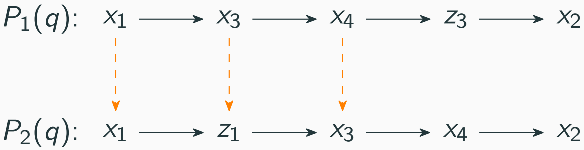
Tractability condition

Theorem (Tractability condition: 1st semester)
A CQ q with self-joins is in $CD \circ Lin$ if there is a set of easy deactivated preimages that can be used to patch all free-trees of q. (recursive algorithm)



Tractability condition

Theorem (Tractability condition: 1st semester)
A CQ q with self-joins is in $CD \circ Lin$ if there is a set of easy deactivated preimages that can be used to patch all free-trees of q. (recursive algorithm)



Tractability condition

Theorem (Tractability condition: 1st semester)
A CQ q with self-joins is in $CD \circ Lin$ if there is a set of easy deactivated preimages that can be used to patch all free-trees of q. (recursive algorithm)

$$P_1(q): \quad x_1 \longrightarrow x_3 \longrightarrow x_4 \longrightarrow z_3 \longrightarrow \cancel{x_2}\ z_2$$

$$P_2(q): \quad x_1 \longrightarrow z_1 \longrightarrow x_3 \longrightarrow x_4 \longrightarrow x_2$$

Then use $P_2(q)$ to patch q.

Hardness

VUTD-Hardness: hypothesis

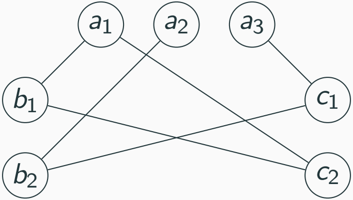


Figure 1: Find a triangle in this unbalanced tripartite graph

Let G be an unbalanced tripartite graph $|V_a| = n$, $V_b = O(n^\alpha)$, $V_c = O(n^\alpha)$, with $\alpha \in [0; 1]$. We can not find a triangle in it in $O(n^{1+\alpha})$.

VUTD-Hardness: Example of encoding

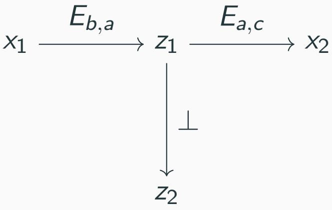


Figure 2: $q(x_1, x_2)$. Atoms labelled with their tagged meaning.

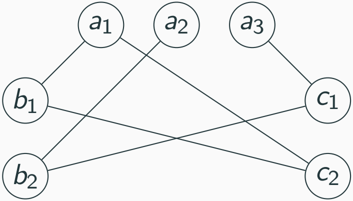


Figure 3: Find a triangle in this unbalanced tripartite graph

VUTD-Hardness: Example of encoding

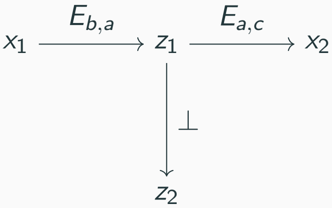


Figure 2: $q(x_1, x_2)$. Atoms labelled with their tagged meaning.

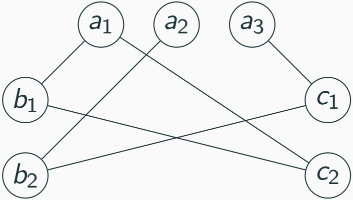


Figure 3: Find a triangle in this unbalanced tripartite graph

- $\mathrm{Dom}(D) = \{\langle x_1; v\rangle \mid v \in V_b\} \cup \{\langle z_1; v\rangle \mid v \in V_a\} \cup \{\langle x_2; v\rangle \mid v \in V_c\} \cup \{\langle z_2; \bot\rangle\}$, we tag the database.
- $\forall (v_b, v_a) \in E_{b,a}.R(\langle x_1; v_b\rangle, \langle z_1; v_a\rangle)$
- $\forall (v_a, v_c) \in E_{a,c}.R(\langle z_1; v_a\rangle, \langle x_2; v_c\rangle)$
- $\forall v_a \in V_a.R(\langle z_1; v_a\rangle, \langle z_2; \bot\rangle)$

VUTD-Hardness: Example of encoding

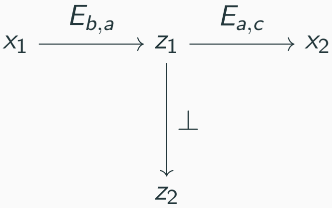


Figure 2: $q(x_1, x_2)$. Atoms labelled with their tagged meaning.

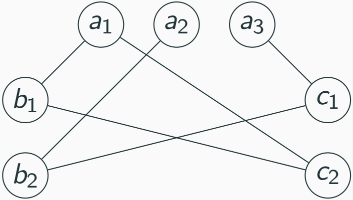


Figure 3: Find a triangle in this unbalanced tripartite graph

- $\langle x_1; a_2 \rangle \to \langle z_1; b_2 \rangle \to \langle x_2; c_1 \rangle$. In $O(1)$, check if $(a_2, c_1) \in E_{b,c}$.
- $\langle x_1; a_1 \rangle \to \langle z_1; b_1 \rangle \to \langle x_2; c_2 \rangle$. In $O(1)$, check if $(a_1, c_2) \in E_{b,c}$.

VUTD-Hardness: Example of encoding

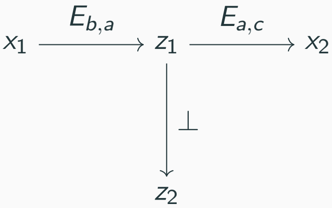


Figure 2: $q(x_1, x_2)$. Atoms labelled with their tagged meaning.

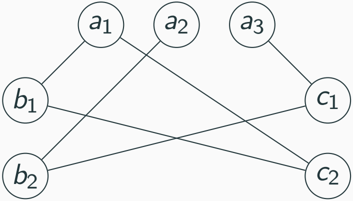


Figure 3: Find a triangle in this unbalanced tripartite graph

If $q \in \text{CD} \circ \text{Lin}$, we can know in $O(n^{2\alpha})$ if there is a triangle or not.

VUTD-Hard: Condition

Theorem (VUTD-Hardness, 1st Semester)
$(\exists F. \exists z \in F. \forall h \in Endo(q). \forall x \in Free(q). h(x) \neq z) \Rightarrow q \notin CD \circ Lin$

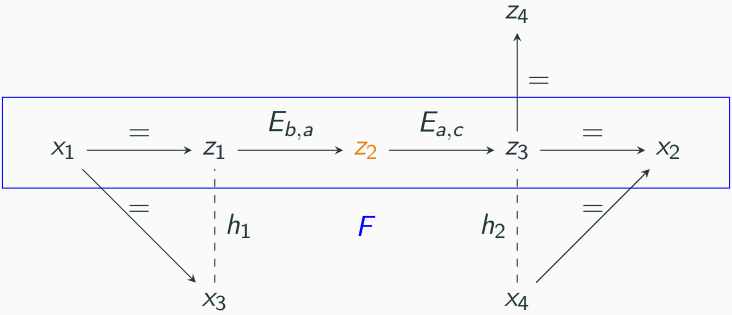
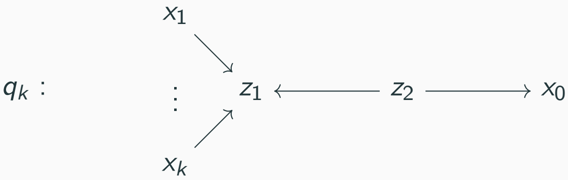


Figure 4: An hard query because of F and z_2. Note that z_4 is not part of the free-path hence you can not encode VUTD with it.

Raised open-questions

k-Clique based hardness



One can find a clique of size $k + 1$ in $O(n^k)$ by enumerating q_{k-1}.

Theorem (Nešetřil and Poljak 1985)
Let $k = 3l + i$ ($(l, i) \in \mathbb{N} \times \{0, 1, 2\}$), and ω is the optimal bound for matrix multiplication ($2 \leq \omega < 2.38$). Let G be a graph with n nodes, we can check if G has a k-clique in $O(n^{\omega l + i})$.

From this, $q_1, \ldots, q_3$ are not in CD $\circ$ Lin. What about $k \geq 4$?

An easy to solve, yet unknown to enumerate query

For any D, using matrix multiplication in $O(|\mathrm{Dom}(D)|^\omega)$, one can solve $q(D)$ in $O(|\mathrm{Dom}(D)|^2 + |\mathrm{Dom}(D)|^\omega + |\mathrm{Dom}(D)|^3)$
$\Rightarrow O(|\mathrm{Dom}(D)|)^3$.

- if it is easy, we need to extend our sufficient condition.
- if it is hard, how to show that it is hard ?

Conclusion

In this work we have:

- Introduced preimages, free-trees.
- Found a sufficient condition.
- Found two necessary conditions.
- Found open-cases that could lead to new enumeration techniques.
- (not in the slide) Built a link between $\mathrm{CD} \circ \mathrm{Lin}$ and $\mathrm{DomLin} \circ \mathrm{Lin}$.

Bibliography

- Bagan, Guillaume, Arnaud Durand, and Etienne Grandjean (2007). **"On Acyclic Conjunctive Queries and Constant Delay Enumeration".** In: *Computer Science Logic*. Ed. by Jacques Duparc and Thomas A. Henzinger. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 208–222. ISBN: 978-3-540-74915-8.
- Carmeli, Nofar and Luc Segoufin (2022). **Conjunctive Queries With Self-Joins, Towards a Fine-Grained Complexity Analysis.** arXiv: 2206.04988 [cs.DB]. URL: https://arxiv.org/abs/2206.04988.
- Nešetřil, Jaroslav and Svatopluk Poljak (1985). **"On the complexity of the subgraph problem".** eng. In: *Commentationes Mathematicae Universitatis Carolinae* 26.2, pp. 415–419. URL: http://eudml.org/doc/17394.