

Constant-Delay Enumeration of Conjunctive Queries with Self-Joins and Projections

Clément Rouvroy*
University of Zurich
Switzerland

Nofar Carmeli
Inria, LIRMM
France

David Carral
Inria, LIRMM
France

Luc Segoufin
Inria, ENS Paris
France

Abstract

Conjunctive queries are a central formalism for database query processing, and a major goal is to understand when their answers can be enumerated efficiently. For acyclic queries without self-joins, this is characterized by the classical free-connex condition. This picture changes in the presence of self-joins: repeated relation symbols can make some queries tractable even when the usual structural criterion fails.

We study this phenomenon for acyclic conjunctive queries with projections and atoms of arity at most two. We give new sufficient conditions for constant-delay enumeration after linear preprocessing, and we identify structural obstructions that leads to necessary conditions under standard fine-grained assumptions. Our results narrow the gap toward a dichotomy and isolate the remaining cases through the domain-linear delay enumeration class $\text{DomLin} \circ \text{Lin}$, which allows delay linear in the domain size. A padding lemma transfers lower bounds from $\text{DomLin} \circ \text{Lin}$ to $\text{CD} \circ \text{Lin}$, and we introduce the clique-shaped family Q_k to motivate the delay-sensitive synchronization of witnesses problem.

CCS Concepts

• **Theory of computation** → **Database query processing and optimization (theory)**; *Complexity classes*.

Keywords

conjunctive queries, enumeration complexity, constant delay enumeration, self-joins, projections, domain-linear delay

1 Introduction

Conjunctive queries (CQs) are one of the basic languages of database theory [1]. They capture select-project-join queries and many graph-pattern queries, while having a syntax simple enough to support precise structural classifications. This simplicity is deceptive: when both the query and the database are part of the input, evaluating a CQ is NP-complete. Hence efficient evaluation requires either restrictions on the query structure or the data-complexity viewpoint, where the query is fixed and the database is the input.

A CQ may have a number of answers that is polynomial in the size of the input database and exponential in the size of the query, so it is natural to study *enumeration* rather than one-shot evaluation. In this setting, answers are printed one by one, and the complexity is measured by the preprocessing time before the first answer is printed and by the maximum delay between two consecutive outputs. The strongest natural guaranty is linear preprocessing and constant delay, captured by the class $\text{CD} \circ \text{Lin}$. Intuitively, a $\text{CD} \circ \text{Lin}$ algorithm reads and indexes the input database once,

then produces each next answer in constant time. This is optimal up to constant factors: any enumeration algorithm must at least inspect the input and write all output tuples.

Not all conjunctive queries can be evaluated with such strong constraints. For self-join-free queries, a precise picture is known: under some fine-grained hypotheses, a query is in $\text{CD} \circ \text{Lin}$ exactly when it is acyclic and free-connex [2]. Equivalently, the hard cases (i.e. not in $\text{CD} \circ \text{Lin}$) are witnessed by a cycle or, in the acyclic case, by a free-path. However, this classification only applies to queries without self-joins. Queries with self-joins arise naturally in graph-shaped data, where the same edge relation may be reused several times along a pattern.

Berkholz et al. [4] were the first to exhibit a non-acyclic query with self-joins which is in $\text{CD} \circ \text{Lin}$. Carmeli and Segoufin then conducted a more systematic study [7]. In the case of full conjunctive queries (no existential quantifiers) they provide necessary conditions as well as separate sufficient conditions for such queries to be easy. However, they do not provide a complete dichotomy, and concluded with examples for which the precise enumeration complexity is still open. Notice that in the case of full conjunctive queries, all acyclic queries are known to be easy, and therefore all unclassified queries are cyclic. This contrasts with the current work, where all queries considered are acyclic but where we also consider existential quantifications.

Our goal is to make progress toward a dichotomy for acyclic conjunctive queries with self-joins. We focus on queries whose atoms have arity at most two, which already capture graph patterns, while keeping the structural picture readable.

To this end, we develop criteria for proving both tractability and hardness. On the algorithmic side, we give sufficient conditions under which the effect of projections can be repaired using self-joins, yielding constant-delay enumeration after linear preprocessing. On the lower-bound side, we identify structural obstructions that still force hardness, despite the presence of repeated relation symbols.

These two directions do not yet meet in a full dichotomy. The remaining cases are those where the query admits enough self-join structure to avoid the known free-path lower bounds, but where our patching criterion is not strong enough to give a $\text{CD} \circ \text{Lin}$ algorithm. We relate this frontier to the domain-linear class $\text{DomLin} \circ \text{Lin}$, which allows delay linear in the domain size, and use the clique-shaped family Q_k as a representative source of such unresolved cases.

Section 2 gives preliminaries. Section 3 develops the recursive patching criterion; Section 4 proves the BMM and VUTD lower bounds. Section 5 discusses the remaining cases through $\text{DomLin} \circ \text{Lin}$ and Q_k .

*Work done while interning at INRIA, Univ. Montpellier and ENS-PSL.

2 Preliminaries

2.1 Databases

A *relational schema* is a set of relation names together with an arity for each of them. In this work, we always assume an underlying relational schema σ , and we follow standard database-theoretic terminology [1]. A *database* D over σ is a finite relational structure consisting of a finite domain $\text{dom}(D)$ and, for each relation symbol R of σ of arity r , a relation $R^D \subseteq \text{dom}(D)^r$. For $\bar{a} \in R^D$, the expression $R(\bar{a})$ is called a *fact* of D . We write $\text{facts}(D)$ for the set of facts of D , that is,

$$\text{facts}(D) = \{R(\bar{a}) \mid R \in \sigma \text{ and } \bar{a} \in R^D\}.$$

By a slight abuse of notation, we write $R(\bar{a}) \in D$ to mean $R(\bar{a}) \in \text{facts}(D)$, or equivalently $\bar{a} \in R^D$.

If $\bar{y}$ is a tuple of variables and R is a relation name of arity $|\bar{y}|$, then $R(\bar{y})$ is called an *atom*. In this paper, we restrict ourselves to *conjunctive queries* (CQs). We write $\bar{x}$ for the tuple of free variables, $\bar{z}$ for the tuple of quantified variables, and $\psi(\bar{x}, \bar{z})$ for a conjunction of atoms using variables from these tuples. A CQ is then an expression of the form

$$q(\bar{x}) \leftarrow \exists \bar{z}. \psi(\bar{x}, \bar{z}),$$

where ψ is called the *body* of the query. We write $\text{atoms}(q)$ for the set of atoms occurring in the body of q , that is,

$$\text{atoms}(q) = \{R_i(\bar{y}_i) \mid 1 \leq i \leq k\}.$$

When no confusion arises, we slightly abuse notation and write $R(\bar{y}) \in q$ to mean $R(\bar{y}) \in \text{atoms}(q)$. The variables in $\bar{x}$ (resp. $\bar{z}$) are called *free* (resp. *quantified*). A query is *full*, or equivalently *without projection*, if there are no quantified variables; it is *Boolean* if there are no free variables. We write $\text{vars}(q)$ for the set of variables occurring in q and $\text{free}(q)$ for the set of free variables of q . We use $x_1, x_2, x_3, \dots$ for free variables and $z_1, z_2, z_3, \dots$ for quantified variables.

An *assignment* for a CQ q over a database D is a function $\mu : \text{vars}(q) \rightarrow \text{dom}(D)$ such that for every atom $R(\bar{y}) \in q$, the fact $R(\mu(\bar{y}))$ belongs to D . A *solution mapping* to q over D is a function $\mu : \text{free}(q) \rightarrow \text{dom}(D)$ for which there exists an assignment η of q over D such that $\eta|_{\text{free}(q)} = \mu$.

The *output tuple* associated with a solution mapping μ is $\mu(\bar{x})$, where $\bar{x}$ is the tuple of free variables appearing in q . The *evaluation* of q over D , denoted $q(D)$, is the set of all such output tuples:

$$q(D) = \{\mu(\bar{x}) \mid \mu \text{ is a solution mapping to } q \text{ over } D\}.$$

The *enumeration problem* for q over D consists in outputting all tuples of $q(D)$ one by one, without repetition.

For two CQs $q(\bar{x})$ and $q'(\bar{x}')$, a *homomorphism* from q to q' is a function $h : \text{vars}(q) \rightarrow \text{vars}(q')$ such that for every atom $R(\bar{y}) \in q$, there exists an atom $R(\bar{y}') \in q'$ with $h(\bar{y}) = \bar{y}'$. Unless explicitly stated otherwise, homomorphisms are homomorphisms of the bodies of the queries; they are not required to preserve free variables. An *isomorphism* is a bijective homomorphism whose inverse is also a homomorphism.

For a set $S \subseteq \text{vars}(q)$, we write $q|_S$ for the query with the same body as q and with free variables $\text{free}(q) \cap S$. Thus $q|_S$ only changes the projection; it does not delete atoms.

Each CQ can be represented by a hypergraph whose vertices are the variables of the query and whose hyperedges are the sets of

variables occurring in its atoms. We also use the *variable graph* G_q : its vertices are $\text{vars}(q)$, and two variables are adjacent when they occur together in an atom of q . For $S \subseteq \text{vars}(q)$, $G_q[S]$ denotes the subgraph of G_q induced by S .

A CQ is *acyclic* if its associated hypergraph is α -acyclic [3]; otherwise, it is *cyclic*. A CQ q is *free-connex* if it is acyclic and the hypergraph obtained by adding one hyperedge containing all variables of $\text{free}(q)$ is still α -acyclic. A *free-path* in a CQ q is a chordless path in its hypergraph whose endpoints are free and whose internal vertices are quantified, i.e. a path of the form $(x_1, z_1, \dots, z_n, x_2)$ with $n \geq 1$. For instance, the query

$$q_1(x_1, x_2) \leftarrow \exists z. R(x_1, z), S(z, x_2)$$

is acyclic but not free-connex: adding the hyperedge $\{x_1, x_2\}$ creates a cycle in the hypergraph, and (x_1, z, x_2) is a free-path. By contrast,

$$q_2(x_1, x_2, z) \leftarrow R(x_1, z), S(z, x_2)$$

is free-connex, because all variables on the path are free.

2.2 Enumeration Complexity

We use the RAM model for all computations and complexity results, and all our complexity analyses are in *data complexity*, i.e. the query size is treated as a constant.

Our main class of interest is $\text{CD} \circ \text{Lin}$ [4]. If a CQ q belongs to $\text{CD} \circ \text{Lin}$, then for every database D , all answers of $q(D)$ can be enumerated after $O(|D|)$ preprocessing time and with $O(1)$ delay, without repetitions. Here, $|D|$ denotes the size of the database while the size of the domain will be denoted by $|\text{dom}(D)|$. An $O(1)$ delay means a delay not depending on D , but possibly depending on the query (data complexity). Notice that if a query belongs to $\text{CD} \circ \text{Lin}$ then its total evaluation running time is linear in the size of the database and in the number of output tuples.

We remark that we do not use the most restrictive definition of $\text{CD} \circ \text{Lin}$. With our definition, the memory used during the enumeration phase could be linear on the number of solutions (hence potentially polynomial in $|D|$ and exponential in the query size). This may not be desirable. Some works use a more conservative definition requiring only extra constant memory during the enumeration phase; see for instance [10].

2.3 Related Work and Scope

To prove hardness results, we use the following fine-grained complexity hypotheses:

- BMM [2, 6]: given two Boolean $n \times n$ matrices A and B , one cannot compute AB in time $O(n^2)$.
- sBMM [7]: given two Boolean matrices A and B , represented by the lists of their non-zero entries, one cannot compute AB in time $O(m)$, where m is the total number of 1-entries in A , B , and AB .
- sHyperclique [5, 7]: for every $k \geq 3$, one cannot decide the existence of a k -hyperclique in a $(k-1)$ -uniform hypergraph with m hyperedges in time $O(m)$.
- VUTD [6]: for every $\alpha \in (0, 1]$, given a tripartite graph with vertex sets V_1, V_2, V_3 such that $|V_2| = n$ and $|V_1| = |V_3| = O(n^\alpha)$, one cannot decide whether there exists a triangle in time $O(n^{1+\alpha})$.

For CQs without self-joins, Bagan et al. [2] established the following dichotomy, later sharpened and reused in subsequent work such as [5]:

THEOREM 2.1 (BAGAN ET AL. [2], BRAULT-BARON [5]). *Assuming sBMM and sHyperclique, a CQ without self-joins is in $CD \circ Lin$ if and only if it is acyclic and free-connex.*

A first indication that this picture does not extend verbatim to self-joins was given by Berkholz, Gerhardt, and Schweikardt [4], who pointed out that cyclicity is not by itself a certificate of hardness in the presence of self-joins. Carmeli and Segoufin [7] further studied how self-joins affect the fine-grained enumeration complexity of conjunctive queries. Their work focuses mainly on full queries, whereas we study how self-joins interact with projections. In the acyclic self-join-free case, non-free-connexity is witnessed by a free-path, and this free-path is the obstruction used in the lower bound of Theorem 2.1 [2]. We ask what remains of this obstruction when the query contains self-joins.

A key tool in constant-delay enumeration is Carmeli et al.’s *Cheater’s lemma* [8]. It allows us to use enumeration procedures that are almost constant-delay, as long as the exceptional delays and duplicate outputs remain tightly controlled.

LEMMA 2.1 (CHEATER’S LEMMA [8]). *Let q be a fixed CQ. Suppose that there is an algorithm that, on every database D , outputs all tuples of $q(D)$, outputs no tuple outside $q(D)$, and outputs each tuple at most constantly many times. If its delay is constant except for a constant number of gaps of length $O(|D|)$, then $q \in CD \circ Lin$.*

We will use this principle repeatedly when combining enumeration procedures.

Another useful tool is Yannakakis’s algorithm [11]. For a fixed acyclic CQ, it runs in linear time and computes a globally consistent set of surviving tuples along a join tree. If the query is also free-connex, the same join-tree enumeration applied after adding the virtual atom on all free variables gives a $CD \circ Lin$ enumeration algorithm [2, 4].

We use the following Yannakakis preprocessing convention throughout the paper. For an acyclic query q , each atom occurrence is first replaced by its own occurrence relation, initialized with the corresponding input relation. This is important with self-joins: reductions are performed on the occurrence relations, not on the original relation symbol globally. We then run the semijoin reductions of Yannakakis on a join tree of the body. After these reductions, every surviving occurrence tuple can be extended to a full satisfying assignment of q , and the reductions preserve $q(D)$.

We also keep the standard navigation data produced by this preprocessing. For each edge between adjacent atom occurrences A and B in the join tree, and for each surviving tuple $\bar{a}$ of A , we can access in constant time the list of surviving tuples of B that agree with $\bar{a}$ on the shared variables. We call these lists *witness pointers*. They let us extend a partial assignment along the join tree without searching the database again. Unless explicitly stated otherwise, all local constructions below are evaluated over these reduced occurrence relations.

In this paper, we restrict ourselves to acyclic CQs with self-joins and projections. We further assume that all atoms have arity at most 2. This restriction simplifies the structural definitions, still

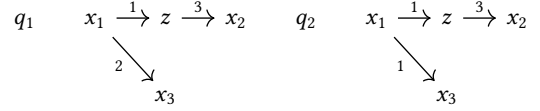


Figure 1: Queries viewed as graph patterns. Distinct arrow labels denote distinct relation symbols, and z is quantified while the other variables are free.

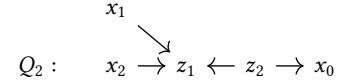


Figure 2: A query showing that enumerating each free-path separately is not enough. It is the case $k = 2$ of the family Q_k discussed in Section 5.

leaves interesting open cases, and already captures many graph patterns.

3 Sufficient condition for Tractability

Throughout this section, we use the Yannakakis preprocessing convention of Section 2: atom occurrences have already been reduced, and witness pointers are available.

We identify the parts of a query with self-joins that can make enumeration hard. If each such part can be materialized with the right delay accounting, the enumeration of the original query reduces to the enumeration of a derived free-connex query. Finally, we give a short patching condition that is sufficient for this materialization, and we explain how we can use homomorphisms of the query to find patches in practice.

3.1 Two motivating examples

Consider the two queries depicted in Figure 1. Both queries have the same variable structure. The only difference is the relation symbol on the side atom: q_1 contains $R_2(x_1, x_3)$, whereas q_2 contains $R_1(x_1, x_3)$. Thus the second query has a self-join, while the first one does not.

In particular, q_1 is self-join-free and has a free-path, so under the assumptions of Theorem 2.1 it cannot be in $CD \circ Lin$. The query q_2 behaves differently: the side branch can be folded onto the hidden witness of the free-path, and this makes the path visible.

EXAMPLE 3.1. *The hard part of q_2 is the free-path x_1, z, x_2 . Consider the query*

$$p(x_1, z, x_2) : \neg R_1(x_1, z) \wedge R_3(z, x_2).$$

This query is acyclic and free-connex. Moreover, each answer (a, c, b) of p is already useful for q_2 : the value c can be used both as the witness z and as the value of the free variable x_3 . Thus enumerating p sees every pair (x_1, x_2) that is connected through the hidden witness z , and it also gives the genuine answer (a, b, c) of q_2 . This is the simple trick that the rest of the section abstracts.

A natural question is whether it is enough to repeat this trick independently for every free-path. The answer is no.

Consider the query Q_2 depicted in Figure 2. It has three free-paths, and for each of them one can build an index similar to the one

suggested by Example 3.1. Nevertheless, as explained in section 5.2, this does not give an enumeration algorithm for the whole query.

3.2 Multi-frontier components

We now define the components that have to be materialized.

DEFINITION 3.2 (MULTI-FRONTIER COMPONENT). *Let q be an acyclic conjunctive query with binary atoms. An existential component of q is a maximal subquery of q that is connected and does not contain any free variable. If E is an existential component, its frontier, denoted $\text{Front}(E)$, is the set of all free variables of q that occur in an atom together with at least one variable of E . We say that E is a multi-frontier component if its frontier contains at least two variables.*

The name only means that such a component can connect at least two free variables through quantified variables. Our multi-frontier components capture the S-components of [9], but in the case of relations of arity at most two. Existential components whose frontier contains zero or one free variable cannot create a free-path. In Q_2 , the variables z_1, z_2 form one existential component, and its frontier is $\{x_0, x_1, x_2\}$. Hence Q_2 has one multi-frontier component.

LEMMA 3.3 (FREE-PATHS LIVE IN MULTI-FRONTIER COMPONENTS). *Let q be an acyclic conjunctive query with binary atoms. The internal variables of every free-path of q belong to a unique multi-frontier component. In particular, distinct multi-frontier components have disjoint quantified variables.*

PROOF. The internal variables of a free-path are quantified and connected to each other, so they belong to a single existential component E . The two endpoints of the path are free variables that occur in atoms together with variables of this component, hence they belong to $\text{Front}(E)$. Thus the frontier of E contains at least two variables, so E is a multi-frontier component. The component E is unique because existential components are maximal connected components and are therefore disjoint. $\square$

3.3 Materializing multi-frontier components

The objective is to build, for every multi-frontier component E , a relation

$$\text{FrontierRel}_E(D)$$

containing exactly the frontier tuples that can be extended through the existential variables of E . Once these relations are available, the enumeration of q is reduced to the enumeration of a query in which every multi-frontier component has been replaced by one atom over its frontier.

We always do this after the usual Yannakakis preprocessing on occurrence relations of q . This keeps only tuples that can participate in some global solution and gives constant-time witness pointers for the remaining acyclic pieces.

For a multi-frontier component E , let $\text{At}_q(E)$ be the set of atom occurrences of q that contain a variable of E . Since E is a maximal connected component of quantified variables and q has binary atoms, every variable occurring in $\text{At}_q(E)$ is either in E or in its frontier. The relevant local assignments are not all assignments satisfying $\text{At}_q(E)$, but only those that occur inside a solution of q . Equivalently, tuples that belong to $q|_{\text{Front}(E)}(D)$.

DEFINITION 3.4 (FRONTIER RELATION). *Let E be a multi-frontier component of q . The frontier relation of E is the relation*

$$\text{FrontierRel}_E(D)$$

defined by

$$\text{FrontierRel}_E(D) = q|_{\text{Front}(E)}(D).$$

Thus FrontierRel_E contains exactly the tuples over the frontier of E that occur in a full solution of q .

Constructing FrontierRel_E must not create a long silent phase before the enumeration starts. The construction may use $O(|D|)$ ordinary preprocessing. If it would require more time, then it must be done during the enumeration phase and therefore computed while producing answers to $q(D)$. For technical reasons, we also store, with each tuple $s \in \text{FrontierRel}_E(D)$, one full answer $\kappa_E(s) \in q(D)$ that witnesses why s was inserted. In view of the Cheater's Lemma, while computing FrontierRel_E we may produce some duplicate answers. This justifies the following definition.

DEFINITION 3.5 (MATERIALIZABLE MULTI-FRONTIER COMPONENT). *Let E be a multi-frontier component of q . We say that E is materializable inside q if the frontier relation FrontierRel_E can be constructed by an algorithm satisfying the following conditions on every database D :*

- (1) *the algorithm is divided into steps and each step is executed in $O(1)$ time*
- (2) *each step may output at most one answer from $q(D)$ and the total number of steps not producing any answer is $O(|D|)$.*
- (3) *for every tuple $s \in \text{FrontierRel}_E(D)$ computed by the algorithm, an answer $\kappa_E(s) \in q(D)$ is also produced, and every answer of $q(D)$ is stored as $\kappa_E(s)$ for at most $O(1)$ tuples s .*

EXAMPLE 3.6. *For the query q_2 of Example 3.1, the unique multi-frontier component is $E = \{z\}$ and its frontier relation is*

$$\text{FrontierRel}_E(D) = \{(a, b) \mid \exists c, R_1(a, c) \wedge R_3(c, b)\}.$$

Enumerating the image $p(x_1, z, x_2)$ from Example 3.1 constructs this relation: each output (a, c, b) inserts (a, b) into FrontierRel_E and stores the answer (a, b, c) of q_2 as $\kappa_E(a, b)$.

PROPOSITION 3.7 (MATERIALIZED MULTI-FRONTIER COMPONENTS REDUCE TO A FREE-CONNEX CQ). *Let q be an acyclic conjunctive query with binary atoms. If every multi-frontier component of q is materializable inside q , then the enumeration of q reduces to the enumeration of an acyclic free-connex query. In particular, $q \in \text{CD} \circ \text{Lin}$.*

PROOF. Work with the reduced occurrence relations provided by the preprocessing convention. For every multi-frontier component E , materialize $\text{FrontierRel}_E(D)$.

Let $\hat{q}$ be obtained from q by removing, for every multi-frontier component E , all atoms that contain a variable of E , and by adding the atom $\text{FrontierRel}_E(D)$.

$\hat{q}$ is acyclic: any cycle using one of the new atoms can be expanded through the connected component that it replaced, yielding a cycle in q . The query $\hat{q}$ is also free-connex. Indeed, if a free-path remained, its internal quantified variables would belong to some existential component touching at least two free variables. This component would be a multi-frontier component, and it would already have been replaced by an atom over its frontier.

The answers of q and $\hat{q}$ are the same on the reduced occurrence relations. One direction is immediate by projection. Conversely, an answer of $\hat{q}$ chooses, for every FrontierRel_E atom, a tuple in $q|_{\text{Front}(E)}(D)$. By definition of this projection, this tuple has a witness on E satisfying $\text{At}_q(E)$. These witnesses have disjoint quantified variables by Lemma 3.3, and they agree with the output tuple on the shared free variables. They can therefore be glued with the remaining atoms of $\hat{q}$ into a satisfying assignment of q .

Since $\hat{q}$ is acyclic and free-connex, it can be enumerated with linear preprocessing and constant delay. The preprocessing of this final enumeration may scan the materialized relations FrontierRel_E . Whenever this preprocessing inspects a tuple $s \in \text{FrontierRel}_E$, we output the stored answer $\kappa_E(s)$ of $q(D)$ before continuing. Since $\hat{q}$ is fixed, each tuple of each FrontierRel_E is inspected only constantly many times, and by materializability each answer of $q(D)$ is stored as $\kappa_E(s)$ for only constantly many tuples s . Thus these intermediate outputs have constant multiplicity and prevent a non-constant delay during the preprocessing of $\hat{q}$. Cheater's lemma then gives $q \in \text{CD} \circ \text{Lin}$. $\square$

3.4 Patching a multi-frontier component

The next definition gives a sufficient condition for materializing a multi-frontier component. The idea is to use another query p as a controlled enumerator for the frontier relation: each answer of p reads a candidate tuple on the frontier of E , and processing this answer must also provide an answer of the original query to output.

DEFINITION 3.8 (PATCH). *Let E be a multi-frontier component of q . A patch for E is a CQ p together with a map*

$$\rho : \text{Front}(E) \rightarrow \text{free}(p).$$

Hence any answer v of p induces a tuple s_v by projecting v onto the coordinates $\rho(\text{Front}(E))$.

We require the following. First, every tuple of $\text{FrontierRel}_E(D)$ must be induced by at least one answer of $p(D)$. Second, membership of s_v in $\text{FrontierRel}_E(D)$ must be testable in $O(1)$ time for each answer v of $p(D)$. Third, every answer v of $p(D)$ must yield in $O(1)$ time an answer

$$\chi_D(v) \in q(D),$$

and every answer of $q(D)$ may be produced in this way only constantly many times.

EXAMPLE 3.9. *For q_2 , take the query p of Example 3.1 and let ρ be the identity on $\{x_1, x_2\}$. Every frontier tuple (a, b) has a witness c , hence is induced by an answer (a, c, b) of p . Conversely, every such answer induces a valid frontier tuple, since x_3 can also take value c . Thus the membership test always accepts, and $\chi_D(a, c, b) = (a, b, c)$ is an answer of q_2 ; this map is injective.*

LEMMA 3.10 (A PATCH MATERIALIZES ITS MULTI-FRONTIER COMPONENT). *Let E be a multi-frontier component of q . If E has a patch p and $p \in \text{CD} \circ \text{Lin}$, then E is materializable inside q .*

PROOF. Enumerate $p(D)$ with constant delay. For every output v , test in constant time whether the induced tuple s_v belongs to $\text{FrontierRel}_E(D)$. If it does, insert it into $\text{FrontierRel}_E(D)$, unless it has already been inserted. In any case, output $\chi_D(v)$.

Soundness follows from the membership test: if a tuple is inserted, then it belongs to $q|_{\text{Front}(E)}(D)$, which is the frontier relation.

Completeness follows from the first requirement in the definition of a patch: every tuple s of the frontier relation is induced by some output of p , accepted by the membership test, and inserted.

For every answer v of $p(D)$, the step that processes v produces the answer $\chi_D(v)$ of $q(D)$. If the membership test accepts and the tuple is inserted into FrontierRel_E for the first time, store $\chi_D(v)$ with this tuple. Since every answer of $q(D)$ is produced from only constantly many answers of $p(D)$, the produced answers and the stored answers have constant multiplicity. The remaining work is the linear preprocessing for enumerating p and the initialization of FrontierRel_E . Hence the multi-frontier component is materializable inside q . $\square$

3.5 Finding patches by folding and deactivation

The definition of patch is semantic. The concrete search space used below is given by homomorphic images of the query, possibly after making some free variables existentially quantified.

DEFINITION 3.11 (CANDIDATE IMAGE). *Let q be an acyclic conjunctive query with binary atoms. A candidate image of q is a query p for which there exist a subquery q' of q and a homomorphism $h : q \rightarrow q'$ such that:*

- (1) p has the same body as q' ;
- (2) $\text{free}(p) \subseteq \{h(x) \mid x \in \text{free}(q)\}$.

If the inclusion is strict, we say that the missing output coordinates have been deactivated.

We call the homomorphism $h : q \rightarrow q'$ a *folding*. Intuitively, it folds a free branch, or an already exposed part of the query, onto the multi-frontier component that we want to patch. This may turn a quantified witness of q into a free variable of the image. Deactivation then keeps the same body but forgets output coordinates that are irrelevant for the current component. This operation is sometimes necessary: in Figure 4, keeping all output coordinates after folding leaves a free-path, while deactivating the irrelevant branch gives a free-connex candidate image.

To avoid checking the semantic definition from scratch, we use the following sufficient condition. A candidate image becomes a patching image when the folding exposes the frontier values, when every candidate answer reads a frontier tuple that is sound for q , and when the deactivated coordinates remain recoverable. The soundness condition is witnessed by a homomorphism $\ell : q \rightarrow q'$: composing any witness of the image with ℓ gives a witness of q with the frontier values read through ρ .

DEFINITION 3.12 (PATCHING IMAGE FOR A MULTI-FRONTIER COMPONENT). *Let p be a candidate image of q witnessed by $h : q \rightarrow q'$, and let E be a multi-frontier component of q . We say that p is a patching image of q for E if there exist a map $\rho : \text{Front}(E) \rightarrow \text{free}(p)$ and a homomorphism $\ell : q \rightarrow q'$ such that*

$$h(\rho(x)) = x \quad \text{and} \quad \ell(x) = \rho(x) \quad \text{for every } x \in \text{Front}(E).$$

The first equality says that folding a solution of q into q' preserves the frontier tuple that has to be inserted in FrontierRel_E . The second says that every answer of the image reads a frontier tuple that is sound for q . In addition, every connected component of

$$G_{q'}[\text{vars}(q') \setminus \text{free}(p)]$$

that contains a deactivated output coordinate, that is, a variable of $h(\text{free}(q)) \setminus \text{free}(p)$, must be adjacent to at most one variable of $\text{free}(p)$. Thus every forgotten output coordinate lies in a part that can be recovered from one free value of p , or without any free value.

LEMMA 3.13 (PATCHING IMAGES GIVE PATCHES). *Let E be a multi-frontier component of q . If p is a patching image of q for E , then p is a patch for E .*

PROOF. Let $h : q \rightarrow q'$ witness the candidate image, and let ρ and ℓ be given by Lemma 3.12. We apply the preprocessing convention to the acyclic body of q' , obtaining reduced occurrence relations and witness pointers. We also build, for each connected component mentioned in the deactivation condition, an extension index: if the component is adjacent to a free variable u of p , the index stores one assignment for every surviving value of u ; if it has no free neighbour, it stores one arbitrary surviving assignment. Since q' is fixed and acyclic, these indexes are built in linear time and queried in $O(1)$ time.

First, every frontier tuple is induced by some answer of p . Take a tuple $s \in \text{FrontierRel}_E(D)$, and let μ be an assignment of q whose projection on $\text{Front}(E)$ is s . Define an assignment μ' of q' by $\mu'(y) = \mu(h(y))$. Since q' is a subquery of q and h is a homomorphism from q to q' , μ' satisfies the body of q' . Its projection on $\text{free}(p)$ is an answer v of p , and for every $x \in \text{Front}(E)$ we have

$$v(\rho(x)) = \mu'(\rho(x)) = \mu(h(\rho(x))) = \mu(x) = s(x).$$

Second, every induced tuple is sound. Take an answer v of p and any assignment μ' of the body of q' witnessing it. Then $\mu' \circ \ell$ satisfies q . Moreover, for every $x \in \text{Front}(E)$,

$$(\mu' \circ \ell)(x) = \mu'(\rho(x)) = v(\rho(x)).$$

Thus the tuple induced by v belongs to $\text{FrontierRel}_E(D)$. In particular, the membership test for this patching image is the constant-time test that always accepts.

It remains to define the answer map. For a free variable x of q with $h(x) \in \text{free}(p)$, set $\chi_D(v)(x) = v(h(x))$. If $h(x) \notin \text{free}(p)$, use the extension index of the component containing $h(x)$ to choose its value, using the unique adjacent free value when there is one. Since v is an answer of p , these choices extend to some full assignment μ^+ of the body of q' extending v . The composition $\mu^+ \circ h$ satisfies q , so $\chi_D(v) \in q(D)$, and the construction takes $O(1)$ time.

Finally, if two answers of p differ, they differ on some $u \in \text{free}(p)$. By definition of candidate images, $u = h(x)$ for some $x \in \text{free}(q)$, and the answer map assigns x the value of u . Hence distinct answers of p have distinct images under χ_D . $\square$

The semantic definition above is the object used in the materialization proof, while patching images define the finite syntactic space in which patches are searched. No easy query is currently known to require a patch outside this syntactic space.

EXAMPLE 3.14. *In Example 3.1, let q' be the subquery of q_2 whose atoms are $R_1(x_1, z)$ and $R_3(z, x_2)$, and let $h : q_2 \rightarrow q'$ map x_3 to z while fixing x_1, x_2, z . The image*

$$p(x_1, z, x_2) :- R_1(x_1, z) \wedge R_3(z, x_2)$$

is a candidate image. It is a patching image of q_2 for the multi-frontier component $\{z\}$, whose frontier is $\{x_1, x_2\}$, with ρ equal to the identity

on the frontier. The associated answer of q_2 is obtained by using the value of z also for x_3 .

In the semantic definition, a patch may enumerate more candidates than the frontier relation. The membership test removes the candidates that do not correspond to tuples of $q|_{\text{Front}(E)}(D)$, while the map χ_D gives a full answer of $q(D)$ for each processed candidate. Deactivation is the mechanism that keeps only the output coordinates needed for the current component, and the deactivation condition above is what ensures that the forgotten coordinates can still be recovered when an answer of q has to be produced.

3.6 A recursive criterion

A patch may itself be a query with free-paths. Hence the criterion has to be recursive.

DEFINITION 3.15 (PATCHING DEPTH). *Let q be an acyclic conjunctive query with binary atoms. Its patching depth $\delta(q)$ is the least $d \in \mathbb{N}$ such that:*

- (1) *if $d = 0$, then q is free-connex;*
- (2) *if $d > 0$, then for every multi-frontier component E of q , there exists a patching image p_E of q for E such that $\delta(p_E) < d$.*

If no such d exists, we set $\delta(q) = \infty$. We say that q is fully patched if $\delta(q) < \infty$.

PROPOSITION 3.16 (RECURSIVE PATCHING CRITERION). *Let q be an acyclic conjunctive query with binary atoms. If q is fully patched, then $q \in \text{CD} \circ \text{Lin}$.*

PROOF. We proceed by induction on $\delta(q)$. If $\delta(q) = 0$, then q is acyclic and free-connex, hence $q \in \text{CD} \circ \text{Lin}$ by the standard upper bound for acyclic free-connex CQs [2].

Assume that $\delta(q) = d > 0$ and that the claim holds for every query of patching depth smaller than d . Let $E_1, \dots, E_k$ be the multi-frontier components of q . By definition, for every i there exists a patching image p_i of q for E_i such that $\delta(p_i) < d$. By induction hypothesis, each p_i is in $\text{CD} \circ \text{Lin}$. By Lemma 3.13 and Lemma 3.10, every multi-frontier component E_i is materializable inside q . Applying Proposition 3.7 gives $q \in \text{CD} \circ \text{Lin}$. $\square$

COROLLARY 3.1 (ONE-STEP PATCHING). *Let q be an acyclic conjunctive query with binary atoms. Suppose that, for every multi-frontier component E of q , there exists a patching image p_E of q for E such that p_E is free-connex. Then $q \in \text{CD} \circ \text{Lin}$.*

PROOF. Every such p_E has patching depth 0. Hence q has patching depth at most 1, and the result follows from Proposition 3.16. $\square$

3.7 Applications

The criterion isolates three tractable patterns that we use repeatedly: direct patching, recursive patching, and patching by deactivation.

Direct patching. The free-connex image p of Example 3.1 patches the only multi-frontier component of q_2 . Hence $\delta(p) = 0$ and $\delta(q_2) \leq 1$, so $q_2 \in \text{CD} \circ \text{Lin}$ by Proposition 3.16.

Recursive patching without deactivation. Consider the query q_3 and its two patching images in Figure 3. The only multi-frontier component of q_3 is $E = \{z_1, z_2\}$, with frontier $\{x_1, x_2\}$. Folding z_2 onto x_3 yields p_h , which is a patching image of q_3 for this component.

$$\begin{aligned}
q_3 : & \quad x_1 \text{ --- } z_1 \text{ --- } z_2 \text{ --- } x_2 \text{ --- } x_3 \\
p_h : & \quad x_1 \text{ --- } z_1 \text{ --- } x_3 \text{ --- } x_2 \\
p_i : & \quad x_1 \text{ --- } x_2 \text{ --- } x_3
\end{aligned}$$

Figure 3: A depth-2 tractable query. All edges denote atoms of the same binary relation. The image p_h patches the only multi-frontier component of q_3 , and p_i then patches the only multi-frontier component of p_h .

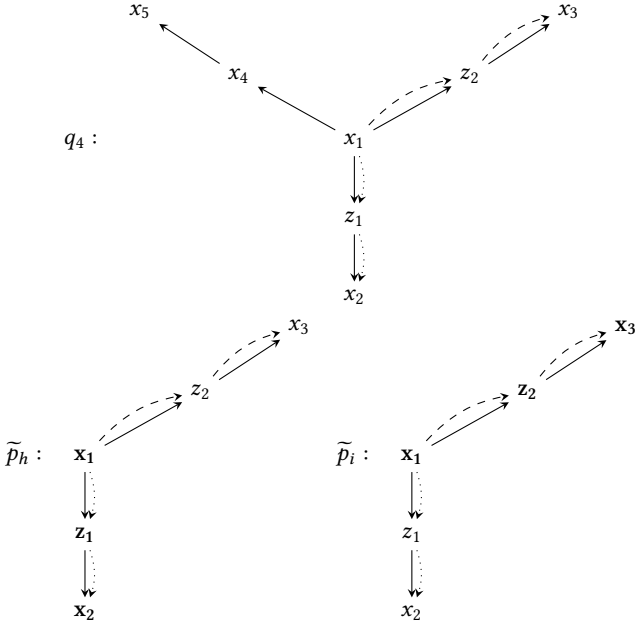


Figure 4: Plain, dashed, and dotted arrows denote three different relation symbols. Bold variables are free.

However, p_h is not yet free-connex, since it still has the free-path (x_1, z_1, x_3) . Folding z_1 onto x_2 then yields p_i , which is free-connex. Therefore $\delta(p_i) = 0$, $\delta(p_h) \leq 1$, and $\delta(q_3) \leq 2$, so $q_3 \in \text{CD} \circ \text{Lin}$. This is exactly why the criterion is recursive rather than one-shot.

A three-branch query that needs deactivation. Consider the query q_4 and the two deactivated patching images shown in Figure 4. It has two multi-frontier components, $\{z_1\}$ and $\{z_2\}$, with frontiers $\{x_1, x_2\}$ and $\{x_1, x_3\}$. The left branch (x_1, x_4, x_5) is the one used for folding. Let h fix every variable except (x_4, x_5) , which it maps to (z_1, x_2) , and let i be defined symmetrically by sending (x_4, x_5) to (z_2, x_3) .

The two extra atoms on the hard branches are essential. They ensure that after folding the free branch onto one hard branch, the other hard branch is still present with its own relation symbol. Consequently, the canonical image given by h still has a free-path on (x_1, z_2, x_3) , while the canonical image given by i still has a free-path on (x_1, z_1, x_2) . Thus the undeactivated images are not free-connex.

Deactivation removes exactly the irrelevant output coordinates. In $\tilde{p}_h$ the free variables are $\{x_1, z_1, x_2\}$, so the branch (x_1, z_2, x_3)

becomes harmless and $\tilde{p}_h$ is a patching image of q_4 for the multi-frontier component $\{z_1\}$. Symmetrically, $\tilde{p}_i$ keeps the free variables $\{x_1, z_2, x_3\}$ and is a patching image of q_4 for $\{z_2\}$. Both queries are acyclic and free-connex, hence of depth 0. Therefore $\delta(q_4) \leq 1$ and Proposition 3.16 yields $q_4 \in \text{CD} \circ \text{Lin}$.

How to read the criterion. Inspect the variable graph to identify multi-frontier components, search for foldings exposing their witnesses, and deactivate irrelevant output coordinates subject to Lemma 3.12. For a fixed query, there are only constantly many homomorphisms to subqueries and choices of deactivation, giving a finite search space for recursive tractability proofs.

Repeated relations help only when they expose witnesses and provide global answers with controlled multiplicity. In q_2 , folding x_3 onto z does this directly. In Q_2 , exposing each pairwise path separately still leaves the need to synchronize a common pair (z_1, z_2) across all branches; see Section 5.

4 Hardness Conditions for self-joins

The previous section established a sufficient condition for tractability. We now turn to lower bounds. We give two hardness conditions, under assumptions of different strength, showing when self-joins do not overcome the obstruction caused by a free-path.

The first condition applies when an entire free-path is *let-alone*, meaning that no free variable outside the path can be folded into it via a homomorphism to a subquery. We show in section 4.2 that any query satisfying this condition is hard assuming the BMM hypothesis.

The second condition applies when a single quantified variable is never the image of a free variable under a homomorphism to a subquery of the query. We show in section 4.3 that any query satisfying this condition is hard assuming the VUTD hypothesis.

4.1 Tagging technique

Our lower bounds are proved using reductions from hard problems (BMM and VUTD) and our reductions use the *tagging* technique of [7] that we discuss in this section. The idea is to replace each database value by a pair $\langle a; \xi \rangle$, where a comes from the hard instance and ξ is a query variable that records which part of the query this value is meant to simulate. The key point is that every solution on such a tagged database can be transformed into a homomorphic image of the query.

LEMMA 4.1 (TAG MAP). *Let q be a conjunctive query and let D^* be a tagged database built so that, for every relation symbol R , each tuple*

$$R(\langle a_1; \xi_1 \rangle, \dots, \langle a_r; \xi_r \rangle) \in D^*$$

is inserted only when $R(\xi_1, \dots, \xi_r)$ is an atom of q . Then every satisfying assignment μ of q on D^* induces a homomorphism

$$\tau_\mu : q \rightarrow q_\mu$$

to a subquery q_μ of q .

PROOF. For every variable $y \in \text{vars}(q)$, write

$$\mu(y) = \langle a_y; \xi_y \rangle$$

and define $\tau_\mu(y) = \xi_y$. Let $R(y_1, \dots, y_r)$ be an atom of q . Since μ is satisfying, the tuple

$$R(\langle a_{y_1}; \xi_{y_1} \rangle, \dots, \langle a_{y_r}; \xi_{y_r} \rangle)$$

belongs to $D^\star$. By the way $D^\star$ was built, this tuple could only have been inserted if

$$R(\xi_{y_1}, \dots, \xi_{y_r})$$

is an atom of q . Hence every atom of q is sent by τ_μ to an atom of q . The atoms reached in this way form a subquery q_μ of q , and τ_μ is a homomorphism from q to q_μ . $\square$

This simple observation is what makes tagging useful in the self-join setting: in a tagged database a variable of a query can only be assigned to an element whose tag is the image of that variable under some automorphism of the query.

4.2 BMM-based hardness

The BMM lower bound captures the case where one free-path can be isolated from the rest of the free variables. If no other free variable can be mapped to a quantified variable of that path, then the path can carry the full matrix-multiplication encoding while all remaining free outputs stay confined to a constant-size dummy domain.

DEFINITION 4.2 (TOUCHED FROM OUTSIDE A SET). *Let q be a conjunctive query, let $C \subseteq \text{vars}(q)$, and let $y \in C$. We say that y is touched from outside C if there exist a free variable $x \in \text{free}(q) \setminus C$ and a homomorphism $h : q \rightarrow q'$ to a subquery q' of q such that $h(x) = y$.*

DEFINITION 4.3 (LET-ALONE FREE-PATH). *Let q be a conjunctive query and let F be a free-path of q . We say that F is let-alone if no variable of F is touched from outside $\text{vars}(F)$.*

For instance, in

$$q_{\text{alone}}(x_1, x_2, u) : \neg \exists z. R(x_1, z) \wedge S(z, x_2) \wedge T(x_1, u),$$

where R, S, T are distinct, the free-path $F = (x_1, z, x_2)$ is let-alone: the only free variable outside F is u , and no homomorphism can send u to x_1, z , or x_2 because the atom $T(x_1, u)$ has no matching copy on the path.

By contrast, in

$$q_{\text{fold}}(x_1, x_2, u) : \neg \exists z. R(x_1, z) \wedge S(z, x_2) \wedge R(x_1, u),$$

the same free-path $F = (x_1, z, x_2)$ is not let-alone. The homomorphism that fixes (x_1, z, x_2) and maps u to z sends the atom $R(x_1, u)$ to the existing atom $R(x_1, z)$. Thus, the outside free variable u touches the quantified variable z of the path. Moreover, this query is easy according to Example 3.1. The following result implies that any query containing a let-alone free-path, like q_{alone} , is hard.

PROPOSITION 4.4 (BMM HARDNESS). *Let q be an acyclic conjunctive query with binary atoms and let*

$$F = (x_1, z_1, \dots, z_\ell, x_2)$$

be a let-alone free-path of q . Then, under BMM, $q \notin \text{CD} \circ \text{Lin}$.

PROOF. Assume toward contradiction that $q \in \text{CD} \circ \text{Lin}$. We reduce Boolean matrix multiplication to the enumeration of q .

Let A and B be two Boolean $n \times n$ matrices. We build a tagged database $D^\star$. Write $v_0 = x_1, v_{\ell+1} = x_2$, and $v_p = z_p$ for $1 \leq p \leq \ell$.

The database $D^\star$ is constructed as follows:

- (1) for every $A[i, k] = 1$, we have in $D^\star$ the elements $\langle i; v_0 \rangle, \langle k; v_1 \rangle \dots \langle k; v_\ell \rangle$ that are related as an isomorphic image of the path $(x_1, z_1, \dots, z_\ell)$ in q .
- (2) for every $B[k, j] = 1$, we have in $D^\star$ the elements $\langle k; v_\ell \rangle$ and $\langle j; v_{\ell+1} \rangle$ that are related as an isomorphic image of the path (z_ℓ, x_2) in q .
- (3) if $R(v_p, y)$ is an atom of q such that $y \notin \text{vars}(F)$ we also have an element $\langle \perp; y \rangle$ in $D^\star$ and for all elements $\langle a; v_p \rangle$ already in $D^\star$ we add the fact $R(\langle a; v_p \rangle, \langle \perp; y \rangle)$.
- (4) likewise for the atoms $R(y, v_p)$ of q .
- (5) finally, for every atom $R(y, y')$ of q such that both y and y' are not in $\text{vars}(F)$ we add the fact $R(\langle \perp; y \rangle, \langle \perp; y' \rangle)$.

Notice that every (i, j) such that $(AB)[i, j] = 1$ corresponds to solution of q as follows: Choose k such that $A[i, k] = B[k, j] = 1$. Map x_1 to $\langle i; x_1 \rangle$, every z_p to $\langle k; z_p \rangle$, x_2 to $\langle j; x_2 \rangle$, and every variable y outside F to $\langle \perp; y \rangle$.

Conversely, let μ be any answer of $q(D^\star)$ such that the x_1 - and x_2 -coordinates are tagged by x_1 and x_2 . By construction, there must exist some k such that the first atom of F witnesses $A[i, k] = 1$ and the last atom witnesses $B[k, j] = 1$, where i and j are the values carried by x_1 and x_2 . Thus $(AB)[i, j] = 1$.

In other words, matrix multiplication is encoded as a subset of the answers of $q(D^\star)$: those such that the free variable x_1 and x_2 are mapped to a pair of elements of the form $(\langle i, x_1 \rangle, \langle j, x_2 \rangle)$. Each such pair could appear many times as q may have more than two free variables. We now show that this is not the case if q is let-alone.

Let $y \in \text{free}(q) \setminus \text{vars}(F)$. If $\mu(y)$ is tagged by some variable $z \in \text{vars}(F)$, then Lemma 4.1 would yield a homomorphism witnessing that z is touched from outside $\text{vars}(F)$, contradicting the fact that F is let-alone. Hence, every free variable outside F can only take one of the dummy values $\langle \perp; y' \rangle$, of which there are only constantly many.

It remains to consider the two free variables x_1 and x_2 : they may be assigned values whose tags are not x_1 and x_2 . If they receive a value tagged outside of F it is only a dummy value and this only adds a constant size to the domain of x_1 and x_2 . Now, if they receive a value tagged inside of F , but all values whose tags belong to $\text{vars}(F)$ range over only $O(n)$ possible elements: one value carries one index from the matrix instance.

Therefore $q(D^\star)$ has at most $O(n^2)$ answers. Enumerating all answers of $q(D^\star)$ in $\text{CD} \circ \text{Lin}$ and filtering in $O(1)$ time those whose x_1 - and x_2 -coordinates carry the identity tags computes all 1-entries of AB in $O(n^2)$ time, contradicting BMM. $\square$

4.3 VUTD-based hardness

The previous hardness condition asks for an entire free-path to be let-alone. The VUTD hardness condition is more local: a single quantified variable on a free-path already suffices, provided that no free variable can fold onto it.

DEFINITION 4.5 (TOUCHED VARIABLE). *Let q be a conjunctive query and let $z \in \text{vars}(q)$. We say that z is touched if it is touched from outside $\{z\}$.*

Consider

$$q_{\text{VUTD}}(x_1, x_2, u) : \neg \exists z_1, z_2. R(x_1, z_1) \wedge S(z_1, z_2) \wedge T(z_2, x_2) \wedge T(u, x_2),$$

where R, S, T are distinct relation symbols. The free-path

$$F = (x_1, z_1, z_2, x_2)$$

is not let-alone: the outside free variable u can be folded onto z_2 , since the homomorphism fixing (x_1, z_1, z_2, x_2) and mapping u to z_2 sends $T(u, x_2)$ to $T(z_2, x_2)$. On the other hand, z_1 is not touched: no free variable can be mapped to z_1 via a homomorphism. This is enough to make the query hard as the next proposition shows.

The previous BMM condition is not contained in the local untouchability condition below: a let-alone free-path only prevents touches coming from free variables outside the path, whereas a touched quantified variable may also be touched by one of the free endpoints of the path. For example, in

$$q(x_1, x_2) : \neg \exists z. R(x_1, z) \wedge R(z, x_1) \wedge R(z, x_2) \wedge R(x_2, z).$$

the free-path is let-alone though z is touched by x_2 . We therefore use the VUTD-hardness condition as the union of two cases. The first one is the let-alone free-path condition of Proposition 4.4, which is hard under VUTD because VUTD implies BMM. The second one is the local untouchability condition proved directly below.

PROPOSITION 4.6 (VUTD HARDNESS CONDITION). *Let q be an acyclic conjunctive query with binary atoms. Suppose that one of the following conditions holds:*

- (1) *q has a let-alone free-path;*
- (2) *some quantified variable z occurs in a free-path of q and is not touched.*

Then, under the VUTD hypothesis, $q \notin \text{CD} \circ \text{Lin}$.

PROOF. If q has a let-alone free-path, the result follows from Proposition 4.4 and from the implication $\text{VUTD} \Rightarrow \text{BMM}$ [6]. Hence, assume that some quantified variable z occurs in a free-path of q and is not touched.

Assume towards a contradiction that $q \in \text{CD} \circ \text{Lin}$. Let

$$F = (x_1, \dots, z, \dots, x_2)$$

be a free-path going through z . Since q is acyclic and has binary atoms, its variable graph is a forest once unary atoms and parallel edges are ignored (two edges are said to be parallel if they are over the same variables). Remove z from this graph. Let L be the connected component containing x_1 , let R be the connected component containing x_2 , and let

$$O := \text{vars}(q) \setminus (L \cup R \cup \{z\}).$$

The sets L and R are distinct, and there is no binary atom whose variables lie in two different sets among L, R, O unless one of the two variables is z (otherwise the query is cyclic).

We reduce VUTD to the enumeration of q . Let

$$(V_1, V_2, V_3, E_{1,2}, E_{2,3}, E_{1,3})$$

be a tripartite graph with $|V_2| = n$ and $|V_1| = |V_3| = O(n^\alpha)$. We build a tagged database D^* . The intended meaning is that the valuation of a variable of L has a value from V_1 , the valuation of z has a value from V_2 , a variable of R has a value from V_3 , while a variable of O has a dummy value.

For every variable y , define A_y as

$$A_y := \begin{cases} V_1 & \text{if } y \in L, \\ V_2 & \text{if } y = z, \\ V_3 & \text{if } y \in R, \\ \{\perp\} & \text{if } y \in O. \end{cases}$$

The domain of D^* consists of all pairs $\langle a; y \rangle$ and $\langle \perp; y \rangle$ with $y \in \text{vars}(q)$ and $a \in A_y$.

Unary atoms are copied over these available coordinates: for every atom $U(y)$ of q and every $a \in A_y$, insert $U(\langle a; y \rangle)$.

For every binary atom $R(y_1, y_2)$ of q , insert the tagged tuple

$$R(\langle a_1; y_1 \rangle, \langle a_2; y_2 \rangle)$$

for exactly the following choices of (a_1, a_2) :

- (1) if $y_1, y_2 \in L$, then $a_1 = a_2 = u$ for some $u \in V_1$;
- (2) if $y_1, y_2 \in R$, then $a_1 = a_2 = w$ for some $w \in V_3$;
- (3) if $y_1, y_2 \in O$, then $a_1 = a_2 = \perp$;
- (4) if $y_1 = y_2 = z$, then $a_1 = a_2 = v$ for some $v \in V_2$;
- (5) if $y_1 = z$ and $y_2 \in L$, then $a_1 \in V_2$, $a_2 \in V_1$ and $E_{1,2}(a_1, a_2)$, likewise if $y_2 = z$ and $y_1 \in L$;
- (6) if $y_1 = z$ and $y_2 \in R$, then $a_1 \in V_2$, $a_2 \in V_3$ and $E_{2,3}(a_1, a_2)$, likewise if $y_2 = z$ and $y_1 \in R$;
- (7) if $y_1 = z$ and $y_2 \in O$, then $a_1 \in V_2$ and $a_2 = \perp$, likewise if $y_2 = z$ and $y_1 \in O$;

There are no other binary atoms to consider by the definition of L, R, O . Notice that every inserted tuple keeps the variable names as tags, so Lemma 4.1 applies to this database.

We now decide whether the input graph contains a triangle. First, build a hash table that check in $O(1)$ if a tuple is in $E_{1,3}$. Then enumerate $q(D^*)$. We keep only answers with identity tags, namely answers μ such that for every free variable x

$$\mu(x) = \langle a_x; x \rangle.$$

for some a_x . For each such answer, write $\mu(x_1) = \langle u; x_1 \rangle$ and $\mu(x_2) = \langle w; x_2 \rangle$. If $(u, w) \in E_{1,3}$, we accept; if the enumeration ends without accepting, we reject.

We prove correctness of this decision procedure. Suppose first that (u, v, w) is a triangle. Define an assignment η of q by

$$\eta(y) = \begin{cases} \langle u; y \rangle & \text{if } y \in L, \\ \langle v; z \rangle & \text{if } y = z, \\ \langle w; y \rangle & \text{if } y \in R, \\ \langle \perp; y \rangle & \text{if } y \in O. \end{cases}$$

By construction of D^* it is straightforward to verify that it is a valuation of q . Thus $q(D^*)$ contains an identity-tagged answer whose x_1 - and x_2 -coordinates carry (u, w) , and the procedure accepts because $(u, w) \in E_{1,3}$.

Conversely, suppose that the procedure accepts from some identity-tagged answer μ , and let η be a witnessing assignment. Write

$$\eta(x_1) = \langle u; x_1 \rangle, \quad \eta(z) = \langle v; z \rangle, \quad \eta(x_2) = \langle w; x_2 \rangle.$$

Consider the unique path from x_1 to z in the variable graph. Its vertices, except for z , all belong to L . Along every atom inside L , the construction forces the first coordinate to remain equal, and along the last atom entering z , the construction can only use Item 5. Hence $(u, v) \in E_{1,2}$. The same argument on the unique path from

z to x_2 gives $(v, w) \in E_{2,3}$. Since the procedure accepts only when $(u, w) \in E_{1,3}$, the input graph contains the triangle (u, v, w) .

It remains to bound the running time. Let $k = |\text{free}(q)|$. The database has size $O(n^{1+\alpha})$: the largest families of inserted tuples are indexed by $E_{1,2}$ or by $E_{2,3}$, while all other families have size $O(n)$, $O(n^\alpha)$, or $O(1)$; the membership table for $E_{1,3}$ has size $O(n^{2\alpha}) \subseteq O(n^{1+\alpha})$ for α sufficiently small.

We also need an output-size bound. Fix a free variable x . If some answer assigned to x a value tagged by z , then by Lemma 4.1 there would be a homomorphism from q to a subquery of q sending x to z . This contradicts the assumption that z is not touched. Therefore no free variable can take a value tagged by z in any answer. All remaining tags carry either a value from V_1 , a value from V_3 , or the dummy value $\perp$. Since the query is fixed, each free variable has only $O(n^\alpha)$ possible values across all answers, and therefore

$$|q(D^*)| = O(n^{k\alpha}).$$

Choose $\alpha \in (0, 1]$ with $k\alpha < 1 + \alpha$, for instance $\alpha = 1/(2k)$. By the VUTD hypothesis, no algorithm should decide these instances in time $O(n^{1+\alpha})$. But the assumed $\text{CD} \circ \text{Lin}$ enumeration algorithm for q , followed by the filtering above, runs in time

$$O(|D^*| + |q(D^*)|) = O(n^{1+\alpha} + n^{k\alpha}) = O(n^{1+\alpha}),$$

a contradiction. $\square$

COROLLARY 4.1. *Assuming VUTD, if q is an acyclic conjunctive query with binary atoms and belongs to $\text{CD} \circ \text{Lin}$, then q has no let-alone free-path and every quantified variable that occurs on a free-path of q is touched.*

PROOF. This is exactly the contrapositive of Proposition 4.6. $\square$

5 Open questions and future directions

The frontier left by the previous sections is concrete enough to be illustrated by two explicit patterns: the disconnected padded path P_{disc} , which points to $\text{DomLin} \circ \text{Lin}$, and the family Q_k , where several free branches must synchronize the same hidden pair.

5.1 Motivating $\text{DomLin} \circ \text{Lin}$

Consider the query

$$P(x_1, x_2) : \neg \exists z_1, z_2. R(x_1, z_1) \wedge R(z_1, z_2) \wedge R(z_2, x_2)$$

and its disconnected self-join variant

$$P_{\text{disc}}(x_1, x_2, x_3) : \neg \exists z_1, z_2. R(x_1, z_1) \wedge R(z_1, z_2) \wedge R(z_2, x_2) \\ \wedge \bigwedge_{y \in \{x_1, x_2, x_3, z_1, z_2\}} S(y).$$

This query is not covered by our current criteria: the variable x_3 touches both z_1 and z_2 through the unary atoms, so neither Proposition 4.4 nor Proposition 4.6 applies.

The reason this example is important is that it is essentially a padding of the self-join-free three-edge path. The next lemma is the basic mechanism.

Before giving the lemma, we need to define a new complexity class that, to the best of our knowledge, was not studied in the literature previously:

DEFINITION 5.1 ($\text{DomLin} \circ \text{Lin}$). *We say that a CQ q belongs to $\text{DomLin} \circ \text{Lin}$ if for every database D , all answers of $q(D)$ can be enumerated after $O(|D|)$ preprocessing time and with $O(|\text{dom}(D)|)$ delay, without repetitions.*

This class has a deep link with $\text{CD} \circ \text{Lin}$, and it may be a useful tool to study this latter class:

LEMMA 5.2 (PADDING). *Let $p^\#(\bar{x}, u)$ be obtained from a query $p(\bar{x})$ by adding one fresh free variable u . Then*

$$p \in \text{DomLin} \circ \text{Lin} \iff p^\# \in \text{CD} \circ \text{Lin}.$$

PROOF. Let $N = |\text{dom}(D)|$. In the forward direction, output the N answers $(\bar{a}, b)$ of $p^\#$ associated with the current answer $\bar{a} \in p(D)$ while the $\text{DomLin} \circ \text{Lin}$ enumerator computes the next answer of p in the background. In the backward direction, run the constant-delay enumerator for $p^\#$, project away u , and keep a queue of projections not yet output. We pace the simulation by outputting one queued projection every N simulated outputs. After s simulated outputs, at least $\lfloor s/N \rfloor$ distinct projections have been seen, since each projection appears at most N times. Thus the queue is never empty when queried; after the simulation ends, the remaining queue is flushed with the same spacing. The delay is $O(N)$, and the preprocessing stays linear because $N \leq |D|$. $\square$

PROPOSITION 5.3. *The query P_{disc} belongs to $\text{CD} \circ \text{Lin}$ if and only if P belongs to $\text{DomLin} \circ \text{Lin}$.*

PROOF. Assume first that $P \in \text{DomLin} \circ \text{Lin}$, and let (R, S) be an input database for P_{disc} . Compute in linear time the subrelation

$$R' = R \cap (S \times S).$$

Then every witness of the path of P_{disc} uses only values of S , and the pairs (x_1, x_2) produced by this path are exactly the answers of P on R' . Enumerate these pairs with delay $O(|\text{dom}(R')|) \leq O(|S|)$, and for each pair (a, b) output all triples (a, b, c) with $c \in S$. While these $|S|$ outputs are produced, the next pair can be computed in the background, so the overall delay is constant.

Conversely, given an input database for P , let S be the whole active domain. Then P_{disc} is exactly the padded query $p^\#$ from Lemma 5.2. Hence $P_{\text{disc}} \in \text{CD} \circ \text{Lin}$ implies $P \in \text{DomLin} \circ \text{Lin}$. $\square$

Thus one of the simplest unresolved self-join patterns already collapses to a self-join-free domain-linear question.

The same issue can even be phrased without self-joins.

PROPOSITION 5.4. *Let*

$$P_{\text{sf}}(x_1, x_2) : \neg \exists z_1, z_2. R_1(x_1, z_1) \wedge R_2(z_1, z_2) \wedge R_3(z_2, x_2).$$

Then

$$P \in \text{DomLin} \circ \text{Lin} \iff P_{\text{sf}} \in \text{DomLin} \circ \text{Lin}.$$

PROOF. If $P_{\text{sf}} \in \text{DomLin} \circ \text{Lin}$, then $P \in \text{DomLin} \circ \text{Lin}$ by applying the algorithm to instances where R_1 , R_2 , and R_3 are all interpreted by the same binary relation R .

Conversely, assume that $P \in \text{DomLin} \circ \text{Lin}$, and let (R_1, R_2, R_3) be an input instance for P_{sf} . Build in linear time a tagged relation R

whose domain contains the copies $\langle a; x_1 \rangle$, $\langle a; z_1 \rangle$, $\langle a; z_2 \rangle$, and $\langle a; x_2 \rangle$ for every $a \in \text{dom}(D)$, and whose tuples are:

$$\begin{aligned} (\langle a; x_1 \rangle, \langle b; z_1 \rangle) & \quad \text{for every } (a, b) \in R_1, \\ (\langle a; z_1 \rangle, \langle b; z_2 \rangle) & \quad \text{for every } (a, b) \in R_2, \\ (\langle a; z_2 \rangle, \langle b; x_2 \rangle) & \quad \text{for every } (a, b) \in R_3. \end{aligned}$$

Then solutions of P_{sf} on (R_1, R_2, R_3) are in bijection with the solutions of P on R whose free coordinates carry the tags x_1 and x_2 . The construction increases both the database size and the domain size by only a constant factor, so a $\text{DomLin} \circ \text{Lin}$ algorithm for P yields one for P_{sf} . $\square$

Hence the remaining frontier already contains a genuinely self-join-free question.

From a witness perspective, this is also the first place where $\text{DomLin} \circ \text{Lin}$ differs qualitatively from $\text{CD} \circ \text{Lin}$. For constant delay, scanning all possible middle witnesses between two outputs is already enough to recover the known BMM obstruction. For domain-linear delay, one full scan of the active domain is allowed between two outputs, so the question becomes subtler: *how many layers of hidden witnesses can be skipped before duplicates or long pauses become unavoidable?*

The same example also suggests a refined triangle-based viewpoint. Let G be a tripartite graph with parts V_1, V_2, V_3 such that $|V_1| = |V_2| = n$ and $|V_3| = \Theta(\sqrt{n})$. Use tagged copies X_1, X_2 of V_3 , Z_1 of V_1 , and Z_2 of V_2 . The single relation R contains the tagged edges $X_1 \rightarrow Z_1$, $Z_1 \rightarrow Z_2$, and $Z_2 \rightarrow X_2$, and S contains the whole tagged domain.

PROPOSITION 5.5. *If $P_{\text{disc}} \in \text{CD} \circ \text{Lin}$, then one can detect triangles in tripartite graphs with part sizes $n, n, \Theta(\sqrt{n})$ in time $O(n^2)$.*

PROOF. A triangle $(u, v, w) \in V_3 \times V_1 \times V_2$ yields an answer of P_{disc} with $x_1 = \langle u; X_1 \rangle$ and $x_2 = \langle u; X_2 \rangle$, witnessed by $z_1 = \langle v; Z_1 \rangle$ and $z_2 = \langle w; Z_2 \rangle$. Conversely, since R contains only the tagged edge types $X_1 \rightarrow Z_1$, $Z_1 \rightarrow Z_2$, and $Z_2 \rightarrow X_2$, every answer follows this pattern, and an answer whose x_1 and x_2 have the same underlying V_3 value witnesses a triangle. The database has size $O(n^2)$. Moreover x_1 and x_2 each range over a tagged copy of V_3 , while x_3 ranges over the tagged active domain of size $O(n)$, so the total number of answers is at most

$$O(|V_3|^2 \cdot |\text{dom}(D)|) = O(n^2).$$

Enumerating all answers and checking whether some answer has matching underlying values for x_1 and x_2 therefore decides triangle existence in time $O(n^2)$. $\square$

Note that Proposition 5.5 is only an indication, not a hardness theorem under a standard conjecture.

5.2 Synchronizing one witness

The family Q_k exposes a different obstruction. Here the issue is not an extra disconnected output but the need to synchronize the same hidden pair across several branches:

$$Q_k(x_0, x_1, \dots, x_k) : -\exists z_1, z_2. R(z_2, z_1) \wedge R(z_2, x_0) \wedge \bigwedge_{i=1}^k R(x_i, z_1).$$

The query shown earlier in Figure 2 is Q_2 .

PROPOSITION 5.6 (Q_k REMAINS OUTSIDE THE CURRENT FRAMEWORK). *For every $k \geq 2$, the query Q_k has the following properties:*

- (1) *its unique multi-frontier component is $\{z_1, z_2\}$, with frontier $\{x_0, x_1, \dots, x_k\}$;*
- (2) *no free-path of Q_k is let-alone;*
- (3) *both quantified variables z_1 and z_2 are touched;*
- (4) *if a patching image of Q_k patches this multi-frontier component, then its body is Q_k itself and all free variables remain free.*

Consequently, the tractability criterion of Section 3 can certify Q_k only if Q_k is already free-connex, whereas the hardness obstructions of Section 4 do not apply.

PROOF. The variables z_1 and z_2 form a connected existential component whose frontier is exactly $\{x_0, x_1, \dots, x_k\}$. There is no other existential component, so this is the unique multi-frontier component of Q_k .

No free-path is let-alone. If $F = (x_i, z_1, x_j)$, then the outside free variable x_0 touches z_1 via the homomorphism that sends x_0 to z_1 and fixes z_2 . If $F = (x_i, z_1, z_2, x_0)$, then some other leaf x_j with $j \neq i$ touches z_2 by mapping x_j to z_2 and fixing z_1 . Thus Proposition 4.4 never applies. The same homomorphisms show that both z_1 and z_2 are touched, so Proposition 4.6 does not apply either.

Finally, let p be a patching image that patches this multi-frontier component. By definition, p contains a subquery isomorphic to the whole body of Q_k . Since a proper subquery would have fewer atoms, the underlying body of p must be exactly Q_k . Moreover, every free variable belongs to the frontier of the component, so the patching condition forces all of them to remain free in p . $\square$

PROPOSITION 5.7 (EVIDENCE FROM COMPLETE SUBGRAPHS FOR THE FIRST MEMBERS OF Q_k). *Let $k \in \{2, 3\}$. Under the $(k+2)$ -clique hypothesis, $Q_k \notin \text{CD} \circ \text{Lin}$.*

PROOF. Let

$$S_{k+1}(y_0, \dots, y_k) : -\exists c. \bigwedge_{i=0}^k E_i(y_i, c)$$

be the self-join-free $(k+1)$ -star query. We first note that, under the $(k+2)$ -clique hypothesis, S_{k+1} cannot be answered in time $O(n^{k+1})$ on databases of active-domain size n . Indeed, given an n -vertex graph $G = (V, E)$, interpret every E_i by the symmetric edge relation of G . Then every answer $(a_0, \dots, a_k)$ of S_{k+1} is an ordered $(k+1)$ -tuple of neighbors of some common witness c . Checking in $O(1)$ time whether the vertices $a_0, \dots, a_k$ induce a $(k+1)$ -clique decides whether $\{a_0, \dots, a_k, c\}$ is a $(k+2)$ -clique. Since there are at most n^{k+1} such outputs, an $O(n^{k+1})$ algorithm for S_{k+1} would contradict the $(k+2)$ -clique hypothesis.

Assume now that $Q_k \in \text{CD} \circ \text{Lin}$, and let D be a database for S_{k+1} . We build from D a database D^* over the single relation symbol R on the tagged domain

$$\{\langle a; \xi \rangle \mid a \in \text{dom}(D), \xi \in \{x_0, x_1, \dots, x_k, z_1, z_2\}\}.$$

For every fact $E_1(u, v) \in D$, insert the two tuples

$$R(\langle u; x_1 \rangle, \langle v; z_1 \rangle) \quad \text{and} \quad R(\langle v; z_2 \rangle, \langle u; z_1 \rangle).$$

For every $i \in \{2, \dots, k\}$ and every fact $E_i(u, v) \in D$, insert

$$R(\langle u; x_i \rangle, \langle v; z_1 \rangle).$$

Finally, for every fact $E_0(u, v) \in D$, insert

$$R(\langle v; z_2 \rangle, \langle u; x_0 \rangle).$$

This construction is linear in $|D|$.

If $(u_0, \dots, u_k)$ is an answer of $S_{k+1}(D)$ with witness c , then as-signing

$$\begin{aligned} x_0 &\mapsto \langle u_0; x_0 \rangle, \\ x_i &\mapsto \langle u_i; x_i \rangle \quad (1 \leq i \leq k), \\ z_1, z_2 &\mapsto \langle c; z_1 \rangle, \langle c; z_2 \rangle. \end{aligned}$$

satisfies Q_k on D^* . Hence every answer of $S_{k+1}(D)$ yields an answer of $Q_k(D^*)$ whose free coordinates carry the identity tags.

Conversely, let μ be an answer of $Q_k(D^*)$ whose free coordinates carry the identity tags. Write

$$\begin{aligned} \mu(x_i) &= \langle u_i; x_i \rangle \quad (0 \leq i \leq k), \\ \mu(z_1) &= \langle a; z_1 \rangle, \\ \mu(z_2) &= \langle b; z_2 \rangle. \end{aligned}$$

The atom $R(z_2, z_1)$ can only be witnessed by a tuple of the form

$$R(\langle c; z_2 \rangle, \langle c; z_1 \rangle),$$

so necessarily $a = b = c$. Then the atom $R(x_1, z_1)$ implies $E_1(u_1, c)$, each atom $R(x_i, z_1)$ with $2 \leq i \leq k$ implies $E_i(u_i, c)$, and the atom $R(z_2, x_0)$ implies $E_0(u_0, c)$. Thus $(u_0, \dots, u_k)$ is an answer of $S_{k+1}(D)$.

The filtering is therefore exact: answers of $S_{k+1}(D)$ are in bijection with identity-tagged answers of $Q_k(D^*)$. Since the reduction keeps the active domain within a constant factor of $|\text{dom}(D)|$, the query Q_k has at most $O(|\text{dom}(D)|^{k+1})$ answers on D^* . If Q_k were in $\text{CD} \circ \text{Lin}$, we could enumerate all answers of $Q_k(D^*)$ and keep only the identity-tagged ones, thereby answering $S_{k+1}(D)$ in time

$$O(|D^*| + |Q_k(D^*)|) = O(|D| + |\text{dom}(D)|^{k+1}).$$

Applying this to the graph instances above yields an $O(n^{k+1})$ algorithm for $(k+2)$ -clique, contradicting the $(k+2)$ -clique hypothesis. $\square$

QUESTION 5.8. *Can one classify self-join-free acyclic conjunctive queries that belong to $\text{DomLin} \circ \text{Lin}$? Can one characterize when the projections of all free-paths through a multi-frontier component can be joined without storing their common quantified witnesses?*

6 Conclusion

We studied acyclic conjunctive queries with self-joins, projections, and binary atoms. On the positive side, multi-frontier components, patching images, and deactivation give a recursive $\text{CD} \circ \text{Lin}$ criterion. On the negative side, BMM and VUTD yield complementary hardness obstructions. The remaining frontier is already visible through P_{disc} and the family Q_k , where one must synchronize hidden witnesses across several branches.

References

- [1] Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. *Foundations of Databases*. Addison-Wesley.
- [2] Guillaume Bagan, Arnaud Durand, and Etienne Grandjean. 2007. On Acyclic Conjunctive Queries and Constant Delay Enumeration. In *Computer Science Logic*. Springer, 208–222. doi:10.1007/978-3-540-74915-8_18
- [3] Catriel Beeri, Ronald Fagin, David Maier, and Mihalis Yannakakis. 1983. On the Desirability of Acyclic Database Schemes. *J. ACM* 30, 3 (July 1983), 479–513. doi:10.1145/2402.322389
- [4] Christoph Berkholz, Fabian Gerhardt, and Nicole Schweikardt. 2020. Constant Delay Enumeration for Conjunctive Queries: A Tutorial. *ACM SIGLOG News* 7, 1 (Feb. 2020), 4–33. doi:10.1145/3385634.3385636
- [5] Johann Brault-Baron. 2013. *De la pertinence de l'énumération : complexité en logiques propositionnelle et du premier ordre*. Ph.D. Dissertation. Université de Caen. <https://hal.science/tel-01081392>
- [6] Karl Bringmann and Nofar Carmeli. 2025. Unbalanced Triangle Detection and Enumeration Hardness for Unions of Conjunctive Queries. *Logical Methods in Computer Science* 21, 1, Article 29 (2025), 33 pages. doi:10.46298/lmcs-21(1:29)2025
- [7] Nofar Carmeli and Luc Segoufin. 2023. Conjunctive Queries With Self-Joins, Towards a Fine-Grained Enumeration Complexity Analysis. In *Proceedings of the 42nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS '23)*. Association for Computing Machinery, 277–289. doi:10.1145/3584372.3588667
- [8] Nofar Carmeli, Shai Zeevi, Christoph Berkholz, Benny Kimelfeld, and Nicole Schweikardt. 2019. Answering (Unions of) Conjunctive Queries using Random Access and Random-Order Enumeration. arXiv:1912.10704 [cs.DB] <https://arxiv.org/abs/1912.10704>
- [9] Arnaud Durand and Stefan Mengel. 2011. The Complexity of Weighted Counting for Acyclic Conjunctive Queries. arXiv:1110.4201 [cs.CC]
- [10] Wojciech Kazana. 2013. *Query evaluation with constant delay*. PhD thesis. École normale supérieure de Cachan. <https://tel.archives-ouvertes.fr/tel-00919786> NNT : 2013DENS0030.
- [11] Mihalis Yannakakis. 1981. Algorithms for Acyclic Database Schemes. In *Proceedings of the 7th International Conference on Very Large Data Bases*. IEEE Computer Society, 82–94. <https://dblp.org/rec/conf/vldb/Yannakakis81>