Fine-Grained Complexity Analysis of Queries: From Decision to Counting and Enumeration

Arnaud Durand

Fine-Grained Complexity Analysis of Queries: From Decision to Counting and Enumeration

Arnaud Durand
arnaud.durand@u-paris.fr
Université de Paris
Paris, France

ABSTRACT

This paper is devoted to a complexity study of various tasks related to query answering such as deciding if a Boolean query is true or not, counting the size of the answer set or enumerating the results. It is a survey of some of the many tools from complexity measures trough algorithmic methods to conditional lower bounds that have been designed in the domain over the last years.

CCS CONCEPTS

• **Theory of computation** → **Database theory**; **Complexity classes**; **Finite Model Theory**.

KEYWORDS

Query evaluation; enumeration algorithm; counting; logic

ACM Reference Format:
Arnaud Durand. 2020. Fine-Grained Complexity Analysis of Queries: From Decision to Counting and Enumeration. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS'20), June 14–19, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3375395.3389130

1 INTRODUCTION

Query answering is a task of major importance in databases that motivates both practical and fundamental research. In particular, a vast literature is devoted to algorithms for efficient query processing and to their complexity. Given a database $\mathbf{D}$ and a query φ, the most basic problem is that of computing the set $\varphi(\mathbf{D})$ of tuples that form the answers when evaluating φ against the database $\mathbf{D}$. When φ is Boolean, the problem reduces only to determine whether φ is true in $\mathbf{D}$.

However, query answering covers a plethora of algorithmic problems for which one has to either find efficient solutions or to understand why it is hard to adapt tractable approaches for providing approximated or partial answers.

One such problem that has received a considerable amount of attention [4, 5, 25, 33–35, 70, 72] is that of counting the number of results (that is, computing the size of $\varphi(\mathbf{D})$). The source of this might lay in the following:

- Extracting numerical and statistical information is obtained by counting and by general aggregations (e.g., summing, averaging etc.)
- It appears as a key tool in probabilistic databases and reasoning.

Another problem is that of enumerating the results of a query (see [14, 76] for surveys on the problem). The answer set of a query may be of huge size and in many applications one either does not need the whole set (such as when interested in top k answers under some criteria) or one can start exploiting the first answers while waiting for the others. Under this angle, it is interesting to obtain algorithms that offer guarantees for the *regularity* of the generating process. These guarantees can be expressed, for example, in terms of delay between two consecutive solutions.

Going further, whether it concerns deciding, counting or enumerating all of these tasks can be handled in contexts where the data is known from the beginning and does not change or, by contrast, when data is updated regularly (see [3, 15, 16, 54, 55]). In that latter context, the objective is then to answer without starting the computation from scratch.

These few examples illustrate how wide the panel of algorithmic tasks for query answering is. However, query languages may be rather expressive and it is well-known that the complexity of simple and popular query problems may be intractable. For example, evaluating a Boolean conjunctive query over an arbitrary database is known to be NP-complete (when both the structure and the query are regarded as input [24]). When a query problem is intractable, one approach to find islands of tractability is by putting structural restrictions either on the class of queries under consideration or on the data and study the effects of these restrictions on the complexity of query answering. This research direction is at the core of many works in the database community.

The objective of this paper is to survey some of the main approaches dedicated to the fine-grained analysis of query problems. The diversity of tasks, from deciding through enumerating to counting, has required along the years the elaboration of a wide spectrum of algorithmic techniques, of structural results for graphs and hypergraphs and of methods for finding lower bounds (see e.g. [1, 2, 4, 8, 11, 21, 22, 28, 31, 32, 36, 37, 42, 44, 59–61, 74, 77]). In a fundamental way, this diversity has also required to think differently what tractability means depending on the contexts. A good example is the notion of constant delay enumeration [11, 32] that guarantees a delay depending only on the query size (which is independent on the underlying database) between two outputs of an enumeration problem. This notion has emerged has a central class in a re-think of what tractability means for enumeration in the context of query answering (while in algorithm design, linear or

polynomial delay were the paradigm of tractability for years). This survey is not intended to be exhaustive but is aimed at illustrating the methods and approaches that have been at the core of fine-grained complexity analysis of queries, in particular for decision problems, counting, and enumerating. Consequently, most of the results will be illustrated through examples.

Structure of the paper. We will survey the complexity of query problems under two main restrictions: on the data set on the one side, and on the structure of the queries on the other side (and sometimes, to some extend, on both).

Data restriction. In Section 3, we examine the complexity of first-order query languages. As it is well-known that this problem is PSPACE-complete, one turns into restricting the class of databases that are considered as input. It has appeared along the years, that the key notion that leads to tractability is that of sparsity of data (see [66]) and the section is devoted to the survey of the main results in that direction for what concerns decision, counting and enumeration. It turns out that unlike in other domains, tractability for decision will often extend to the two other tasks. This part is completed by Section 3.3, where the case of monadic second-order queries is shortly investigated.

Query restriction. Section 4 is devoted to the wide class of conjunctive queries. It is by far the most developed part of the paper. One fragment that is known for a long time to admit efficient algorithms is that of acyclic conjunctive queries. We examine the complexity of this fragment and show that tractability is, this time, subject to different criteria depending on the algorithmic task under consideration. We then study several extensions of this fragments that enrich its expressive power : taking union of acyclic queries, allowing comparisons or disequalities, and allowing negations. The study of this approach based on query restriction is completed in Section 5 where one studies the classification of counting and enumeration complexity for first-order queries based on restricting their prefixes.

In Section 2, we recall the basic definitions about query problems. We also introduce and motivate different complexity measures suitable for deciding, computing, counting or enumerating the result of a query. To make the main technical sections as self-contained as possible, some of the definitions are introduced only later, when needed. A conclusion is given in Section 6.

1.1 References and acknowledgment

Thanks to Florent Capelli, Nofar Carmeli, Liat Peterfreund, Alexandre Vigny for careful reading of parts of this paper.

Material of Section 3 is inspired by the thesis of Alexandre Vigny [80]. Additional surveys can be found also in [76], [14] (on enumeration for query answering and, in particular, constant delay enumeration), in [78] (on enumeration complexity) and in [19] (connexion with knowledge compilation).

2 PRELIMINARIES

We suppose the reader familiar with rudiments of (first order and second order) logic [64], complexity [7, 43], and graph theory [30]. We write $\mathbb{N}$ to denote the set of non-negative integers, $\mathbb{Q}$ to denote the set of rationals, and $\mathbb{Q}_{>0}$ for the set of positive rationals. For $n \in \mathbb{N}$, we set $[n] = \{1, ..., n\}$.

2.1 Databases and queries

A *signature* σ is a finite set of relation R and function f symbols, each of them associated with a fixed *arity* $ar(R), ar(f) \in \mathbb{N}$. The arity of σ, is the maximal arity of its symbols. A *structure* $\mathbf{D}$ over σ, or a σ-structure (σ is omitted when it is clear from the context) consists of a non-empty finite set $\mathrm{Dom}(\mathbf{D})$ (often D, for short) called the *domain* of $\mathbf{D}$, an $ar(R)$-ary relation $R^{\mathbf{D}} \subseteq \mathrm{Dom}(\mathbf{D})^{ar(R)}$ for each relation symbol $R \in \sigma$ and $ar(f)$-ary function $f^{\mathbf{D}} : \mathrm{Dom}(\mathbf{D})^{ar(R)} \to \mathrm{Dom}(\mathbf{D})$ for each function symbol $f \in \sigma$. When σ contains relation symbols only, a σ-structure is said to be relational. In most context below, a *database* is a finite relational structure.

The cardinality of a finite set A is denoted $|A|$. We define the *size* $\|\mathbf{D}\|$ of $\mathbf{D}$ as $\|\mathbf{D}\| = |\sigma| + |\mathrm{Dom}(\mathbf{D})| + \sum_{R \in \sigma} |R^{\mathbf{D}}| \cdot ar(R)$. It corresponds to the size of a reasonable encoding of $\mathbf{D}$. Also, $\|\varphi\|$ denotes the size of φ i.e. the number of symbols necessary to describe φ.

Let $\mathscr{L}$ be a subclass of first-order logic, denoted FO, or second-order logic, denoted SO. We will give the definition of various fragments of FO and SO when needed in the paper.

A $\mathscr{L}(\sigma)$-*query* is a formula in $\mathscr{L}$ of signature σ, for some relational signature σ (σ is omitted when it is clear from the context). For $\varphi \in \mathscr{L}$, we denote by $var(\varphi)$ its set of variables, $free(\varphi)$ its set of free variables and $atom(\varphi)$ its set of atomic formulas. We write $\varphi(\bar{x}, \bar{X})$ to denote a query whose free first order variables are $\bar{x}$, free second order variables are $\bar{X}$. The number of free variables is called the *arity of the query*. A *Boolean query* (i.e. *sentence*) φ is a query of arity 0 i.e. with $free(\varphi) = \emptyset$.

2.2 Query problem(s)

Given a structure $\mathbf{D}$ and a query φ, an *answer* to φ in $\mathbf{D}$ is a tuple $\bar{a}$ and a tuple of sets $\bar{A}$ of elements of $\mathrm{Dom}(\mathbf{D})$ such that $\mathbf{D} \models \varphi(\bar{a}, \bar{A})$. We write $\varphi(\mathbf{D})$ for the set of answers to φ in $\mathbf{D}$, i.e. $\varphi(\mathbf{D}) = \{(\bar{a}, \bar{A}) : \mathbf{D} \models \varphi(\bar{a}, \bar{A})\}$. When φ is a sentence, the answer is a Boolean.

Let $\mathscr{L}$ be a logic and $\mathscr{C}$ be a class of structures. The *query problem* (resp. Boolean query or model checking problem) of $\mathscr{L}$ over $\mathscr{C}$ is the following computational problem: given a (resp. Boolean) query $\varphi \in \mathscr{L}$ and a database $\mathbf{D} \in \mathscr{C}$, compute $\varphi(\mathbf{D})$ (resp. test if $\mathbf{D} \models \varphi$).

For the *counting problem* of $\mathscr{L}$ over $\mathscr{C}$, given a formula (resp. sentence) $\varphi \in \mathscr{L}$ and a database $\mathbf{D} \in \mathscr{C}$, the objective is to compute the number of elements of $\varphi(\mathbf{D})$.

Another important algorithmic task is enumerating solutions. In the context of query problems for a logic $\mathscr{L}$ over a class $\mathscr{C}$, the *enumeration problem* is the following: given $\varphi \in \mathscr{L}$ and a database $\mathbf{D} \in \mathscr{C}$ output the elements of $\varphi(\mathbf{D})$ one by one with no repetition.

When the formula φ is fixed, we denote by $\sharp \cdot \varphi$ and $\textsc{enum} \cdot \varphi$ the respective counting and enumeration problems.

2.3 Model of computation and complexity measures

Recall that P (resp. NP) is the class of problems that can be decided (resp. verified) in polynomial time.

2.3.1 Complexity measures for query problems. Most of the results given in this paper were designed using Random Access Machines

(RAMs) with addition and uniform cost measure as a model of computation. For further details on this model and its use in logic see [43, 50].

The RAM algorithms will take as input a query $\varphi \in \mathscr{L}$ of size k and a database $\mathbf{D} \in \mathscr{C}$ of size n. The complexity of such problems can be evaluated in two well-known contexts:

- *Combined complexity* were both $\|\mathbf{D}\|$ and $\|\varphi\|$ are taken into account for the complexity evaluation,
- *Data complexity* were only $\|\mathbf{D}\|$ is taken into account (i.e. φ being considered as fixed).

Unless otherwise specified, or when precise bounds are given, we will often consider the *data complexity* setting. We then say that an algorithm runs in *polynomial time* (resp. *quasi-linear time*, resp. *linear time*, resp. *constant time*) if it outputs the solution within $f(k)\cdot n^c$ steps for some $c \in \mathbb{N}$ (resp. $f(k)\cdot n\cdot(\log n)^{O(1)}$ steps, resp. $f(k)\cdot n$ steps, resp. $f(k)$ steps), for some computable function f. An algorithm runs in *pseudo-linear time* if, for all $\epsilon \in \mathbb{Q}_{>0}$ it outputs the solution within $f(k,\epsilon)\cdot n^{1+\epsilon}$ steps, for some function f.

Most of the time measures (such as linear time) considered in the paper are rather restrictive so it is necessary to make precise how data is accessed. For example, we assume that the input relational structure comes with a linear order on the domain. If not, we use the one induced by the encoding of the structure as an input to the RAM.

2.3.2 Complexity measures for counting. Let Σ be an alphabet. A predicate $B \in \Sigma^* \times \Sigma^*$ is polynomially balanced if there exists $c \in \mathbb{N}$ such that, for all $x, y \in \Sigma^*$, $(x, y) \in B$ implies that $|y| \leq |x|^c$. Given a polynomially balanced binary predicate $B \in \Sigma^* \times \Sigma^*$, the counting function associated to B, is the function $\sharp\cdot B : \Sigma^* \to \mathbb{N}$ such that, for all $x \in \Sigma^*$:

$$\sharp\cdot B(x) = |\{y : (x, y) \in B\}|.$$

Definition 2.1. $\sharp\cdot$P, or equivalently $\sharp$P, (resp. $\sharp\cdot$NP) the class of counting functions associated to predicates $B \in$ P (resp. $B \in$ NP).

2.3.3 Complexity measures for enumeration. The first complexity measures for enumeration have been formalized in [56]. Algorithms and complexity for enumeration have deserved a lot of attention in the recent years (see the survey [78]).

Formally, the enumeration problem is identical to the classical query problem except that some focus is put on the regularity of the process that leads to the computation of $\varphi(\mathbf{D})$ It is natural that the dynamic of such a process be measured in terms of delay (i.e. maximal time) between any two consecutive outputs of elements of $\varphi(\mathbf{D})$. Due to the important differences in size between data and query, it will also appear meaningful in this context to separate the enumeration process itself to the *preprocessing time* i.e. the time needed to output the first solution and built necessary data structures.

Let $\delta, \mathbf{p} : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ be two functions. We say that the *enumeration problem* of $\mathscr{L}$ over a class $\mathscr{C}$ of structures can be solved with delay δ and after a preprocessing of $\mathbf{p}$, if it can be solved by a RAM algorithm which, on input $\varphi \in \mathscr{L}$ of size k and $\mathbf{D} \in \mathscr{C}$ of size n, can be decomposed into two phases:

- a preprocessing phase that is performed in time $\mathbf{p}(k, n)$, that produces the first solution and
- an enumeration phase that outputs elements of $\varphi(\mathbf{D})$ with no repetition and a delay in time $\delta(k, n)$ between two consecutive outputs.

Note that, under these conditions, $\varphi(\mathbf{D})$ can be computed in total time:

$$\mathbf{p}(k, n) + |\varphi(\mathbf{D})| \times \delta(k, n).$$

When $\mathbf{p}(k, n)$ and $\delta(k, n)$ are bounded by a polynomial (in n in the data complexity setting; in n and k in combined complexity) the enumeration problem is said to be solvable in *polynomial delay*.

In the database context, where the size of data maybe huge, a delay even linear in the size of $\mathbf{D}$ can hardly be seen as tractable. Consequently, it is crucial to understand if (and when) enumeration can be handled within a delay independent of the data size. In this view, we consider that a query problem can be enumerated with *constant delay after a linear preprocessing*, i.e. is in the class Constant-Delay$_{\text{lin}}$ introduced in [32], if there exists a RAM algorithm which on input φ and $\mathbf{D}$

- performs a preprocessing phase (including the output of the first element of $\varphi(\mathbf{D})$) in time linear in $\|\mathbf{D}\|$ and
- enumerates elements of $\varphi(\mathbf{D})$ with no repetition and a constant delay, i.e. depending on $\|\varphi\|$ only, between two consecutive outputs. The enumeration phase has full access to the output of the preprocessing phase and can use extra memory whose size depends only on $\|\varphi\|$.

2.3.4 Parameterized complexity classes. Let Σ be a finite alphabet. A parameterization of Σ^* is a mapping $\mathsf{k} : \Sigma^* \longrightarrow \mathbb{N}$ that is polynomial time computable. A *parameterized problem* is a pair (Q, k) where $Q \subseteq \Sigma^*$ is a property and k is a parameterization.

Taking into account that the query sizes are usually far smaller than the data sizes, it makes it natural to consider the parameterized version of (below, Boolean) query problems.

p-MC(FO)
Input: A database $\mathbf{D}$ and a Boolean query φ
Parameter: $\|\varphi\|$
Output: Does $\mathbf{D} \models \varphi$

Such an approach applies not only for decision but also naturally for function problems. In this paper, we will express that some algorithmic results can not be improved provided some widely believed complexity hypothesis from parameterized complexity hold. For this we recall briefly three useful measures (see [43]).

Definition 2.2. A parameterized problem (Q, k) is in FPT if there is a computable function $f : \mathbb{N} \to \mathbb{N}$, $c \in \mathbb{N}$ and an algorithm that upon input $x \in \Sigma^*$ of size n decides if $x \in Q$ in time at most:

$$f(\mathsf{k}(x)) \cdot n^c.$$

Intuitively, FPT corresponds to our definition of polynomial time in the data complexity setting.

Definition 2.3. Let Σ, Σ_0 be two finite alphabets. A FPT-reduction from (Q, k) over Σ to (Q_0, k_0) over Σ_0 is a mapping $R : \Sigma^* \to \Sigma_0^*$ such that:

- For all $x \in \Sigma^*$: $x \in Q \iff R(x) \in Q_0$
- R is computable by a FPT algorithm

- There is a computable function $g : \mathbb{N} \to \mathbb{N}$ such that, for all $x \in \Sigma^*$: $\mathrm{k}_0(R(x)) \leq g(\mathrm{k}(x))$

The parameterized clique problem is the following: for a graph $G = (V, E)$ and parameter $k \in \mathbb{N}$, decide whether there is $V' \subseteq V$ of size k such that, for all $x, y \in V'$: $(x, y) \in E$. Then,

- W[1] is the the class of problems FPT-reducible to the parameterized clique problem and,
- AW[$*$] is the class of problems FPT-reducible to p-MC(FO), the model checking of FO.

3 FIRST-ORDER QUERIES AND SPARSITY

We consider here $\mathscr{L}$ = FO and queries $\varphi(\mathbf{x}) \in$ FO with only free first-order variables. It is well known that, given a query $\varphi(\mathbf{x})$ and a database $\mathbf{D}$, $\varphi(\mathbf{D})$ can be computed in time

$$\|\varphi\| \times \|\mathbf{D}\|^h$$

where h is the maximal number of free variables for a sub-formula of φ [64]. Although there could be some variant in the expression of the complexity, under some reasonable complexity assumption (namely that AW[$*$] $\neq$ FPT), there is no hope, for arbitrary databases, to obtain an upper bound where h can be replaced by a constant $c \in \mathbb{N}$ independent of φ. Indeed, as an obvious consequence, such a bound would lead to an algorithmic breakthrough for problem such as testing if a graph has a clique of a given size that can be generically described by:

$$\exists x_1 \ldots \exists x_k \bigwedge_{i<j=1}^{k} x_i \neq x_j \wedge E(x_i, x_j).$$

For arbitrary first-order queries, one can then hope for more efficient bounds only by restricting the databases as input. In this direction, the notion of sparsity has played a key role[66] as we will show.

3.1 The role of sparsity

A first restriction that seems natural to consider is when every elements are in connexion to constantly many others only. Let $\mathbf{D} = \langle D; R_1, \ldots, R_q \rangle$ be a σ-structure for a relational signature $\sigma = \{R_1, \ldots, R_q\}$. For each $i \leq q$, $R_i \subseteq D^{a_i}$. Define the *degree* of an element x in $\mathbf{D}$, denoted $\deg_{\mathbf{D}}(x)$ as the total number of tuples of relations R_i to which x belongs. One defines the degree of a structure as $\deg(\mathbf{D}) = \max_{x \in D}(\deg_{\mathbf{D}}(x))$.

A class $\mathscr{C}$ of structures is of bounded degree if there exists $c \in \mathbb{N}$ such that, for all $\mathbf{D} \in \mathscr{C}$, $\deg(\mathbf{D}) \leq c$. Note that such classes are closed under substructures: if $\mathbf{D}'$ is a substructure of $\mathbf{D}$ and $\mathbf{D}$ is of degree bounded by some $c \in \mathbb{N}$, then so is $\mathbf{D}'$.

By taking advantage of the locality of first-order logic and the fact that in a structure of degree bounded by $c \in \mathbb{N}$, for each element of the domain, the number of elements at distance at most $d \in \mathbb{N}$ is bounded by c^{d+1}, the following result holds.

Theorem 3.1 ([75],[59]). *Let $\mathscr{C}$ be a class of bounded degree structures. Then, the model-checking problem of first-order queries over $\mathscr{C}$ can be decided in time linear in the size of the database i.e. there exists a function f such that, given a sentence $\varphi \in$ FO and $\mathbf{D} \in \mathscr{C}$, testing whether $\mathbf{D} \models \varphi$ can be done in time $f(\|\varphi\|) \cdot \|\mathbf{D}\|$. Moreover, function f is such that:*

$$f(\|\varphi\|) \leq 2^{2^{2^{O(\|\varphi\|)}}}.$$

From [46], it is known that unless AW[$*$] = FPT, the constant can not be lowered to $2^{2^{O(\|\varphi\|)}}$. To which extend could the results from Theorem 3.1 be extended to compute the set $\varphi(\mathbf{D})$? In particular, is it possible to generate the elements of $\varphi(\mathbf{D})$ through a process whose regularity is guaranteed? It turns out that it is indeed the case. Such an investigation leaded to the notion of constant delay enumeration and to the following result(s).

Theorem 3.2 ([32], [59]). *Let $\mathscr{C}$ be a class of bounded structures. Then, there are algorithms that upon input $\varphi \in$ FO and $\mathbf{D} \in \mathscr{C}$:*

- *Output the number of elements of $\varphi(\mathbf{D})$ in time $f(\|\varphi\|) \cdot \|\mathbf{D}\|$.*
- *Enumerate $\varphi(\mathbf{D})$ with constant delay $f(\|\varphi\|)$ after precomputation time $f(\|\varphi\|) \cdot \|\mathbf{D}\|$*

for some function f, such that $f(k) \leq 2^{2^{2^{O(k)}}}$.

Note that this implies that computing $\varphi(\mathbf{D})$ when $\mathbf{D}$ is of bounded degree can be done in total time:

$$f(\|\varphi\|) \cdot (|\varphi(\mathbf{D})| + \|\mathbf{D}\|).$$

The approach of [32] and to some extend some of the works that have followed are based on quantifier elimination methods. They can roughly be described as follows. First, it is convenient to represent bounded degree relations by a collection of partial injective functions. This can be done in different ways. Then, given $\varphi(\mathbf{x}) \in$ FO and a database $\mathbf{D} \in \mathscr{C}$, one constructs:

- a database $\mathbf{D}' \in \mathscr{C}$ in time $O(f(\|\varphi\|) \cdot \|\mathbf{D}\|)$
- and $\varphi'(\mathbf{x}) \in$ FO, quantifier free, in time $O(f(\|\varphi\|))$

for some computable function f such that:

$$\varphi(\mathbf{D}) = \varphi'(\mathbf{D}').$$

Once an equivalent quantifier free has been obtained, enumeration becomes easier.

Example 3.3. Let $\varphi(\mathbf{x}) = \exists y \phi(\mathbf{x}, y)$, on a vocabulary σ of unary function intepreted as partial injective functions. Let $\mathbf{F}$ a σ-structure. We illustrate how to get rid of y. To simplify, let $\varphi(\mathbf{x})$ be as follows

$$\varphi(\mathbf{x}) \equiv \exists y \, [\psi(y) \wedge y \neq f_1(x_1) \wedge \cdots \wedge y \neq f_k(x_k)]$$

where $f_1, ..., f_k \in \sigma$. A term of the form $f_i(x_i) = y$ above, would have permit to replace immediately y by $f_i(x_i)$ everywhere. Let us call $\exists^{h+1}\psi$ the condition:

$$\exists^{h+1}\psi = \exists x_1 \cdots \exists x_{h+1} \bigwedge_{\substack{i,j=1 \\ i \neq j}}^{h+1} x_i \neq x_j \wedge \bigwedge_{i=1}^{h+1} \psi(x_i)$$

Each $\exists^{h+1}\psi$ can be computed in $O(\|\mathbf{F}\|)$. Let $\mathbf{F}'$ be the structure $\mathbf{F}$ enriched with these Boolean informations. Given values for $\mathbf{x}$, suppose $h \leq k$ is the number of distinct values among those of the k terms $f_i(x_i)$ such that $\psi(f_i(x_i))$ is true; then, formula $\varphi(\mathbf{x})$ is true if and only if the number of elements b such that $\mathbf{F} \models \psi(b)$ holds is strictly greater than h. Consequently, $\varphi(\mathbf{F}) = \varphi'(\mathbf{F}')$ where $\varphi'(\mathbf{x})$ is

the (combinatorial involved) quantifier-free formula that look for a possible $h \leq k$ that satisfy the above condition:

$$\varphi'(\mathbf{x}) \equiv \bigvee_{h=0}^{k} \bigvee_{P\subseteq[k], Q\subseteq P, |Q|=h} \psi_P^Q(\mathbf{x}) \wedge \exists^{h+1}\psi$$

with

$$\psi_P^Q(\mathbf{x}) \equiv \bigwedge_{j\in Q} \psi(f_j(x_j)) \wedge \bigwedge_{i\in P} \bigvee_{j\in Q} f_i(x_i) = f_j(x_j) \wedge \bigwedge_{j\in[k]\setminus P} \neg\psi(f_j(x_j))$$

Why is enumeration easier for quantifier-free formulas? Suppose a quantifier-free $\varphi(\mathbf{x}, y)$ is as follows:

$$\varphi(\mathbf{x}, y) \equiv \psi_1(\mathbf{x}) \wedge \psi_2(y) \wedge \bigwedge_{i=1}^{k} y \neq f_i(x_i)$$

$\psi_2(y)$ has one free variable, $\psi_2(\mathbf{F})$ can easily be computer in linear time. By induction enumerating $\varphi(\mathbf{F})$ amounts to enumerate tuples $(\mathbf{a}, b)$ with $\mathbf{a} \in \psi_1(\mathbf{F})$, $b \in \psi_2(\mathbf{F})$ with at most k exceptions for each fixed $\mathbf{a}$: when $\wedge_{i=1}^{k} b \neq f_i(a_i)$ is true. Hence it can be done with constant delay See Algorithm 1 below (if $|\psi_2(\mathbf{F})| < k$ an even simpler tratment can be done).

Algorithm 1 Enumerate $\varphi(\mathbf{F})$ (with the hypothesis that $|\psi_2(\mathbf{F})| \geq k$)

```
for a ∈ ψ1(F) do
    for b ∈ ψ2(F) do
        if D ⊭ ⋁_{i=1}^{k} b = f_i(a_i) then
            output (a, b)
```

The results above were the first of a series that prove linear or pseudo-linear model checking and constant delay enumeration for increasingly larger classes $\mathscr{C}$, closed under substructures: graphs of bounded expansion [39, 60], graphs of locally bounded tree-width [45] (that includes planar and bounded tree-width graph), graphs of locally bounded expansion [77]. Note that numeration results were technically harder and most of the time came significantly later.

All these classes concern structures that can be identified as *sparse*: the density of tuples compared to elements of the domains is somehow restricted. In [67] a notion called *nowhere dense* graphs was introduced as a formalization of classes of sparse graphs. It appears to encompass all known classes of sparse graphs (planar, bounded tree-width, excluding a minor, locally bounded expansion, etc [68]). One way to represent it is through the notion of *r-minor* defined below.

Definition 3.4. Let $G = (V, E)$ be an undirected graph and r be an integer. A graph $H = (V', E')$ is an *r-minor* (also named *shallow minor* or *low depth minor*) of G if:

- $V' \subseteq V$,
- for every a_i in V', there is a set S_i in V such that:
  - $S_i \subseteq N_r^G(a_i)$,
  - for every $i \neq j$ we have $S_i \cap S_j = \emptyset$, and
  - for every $i \neq j$ we have (a_i, a_j) is in E' if and only if in G there is an edge from a node in S_i to a node in S_j.

We note $H \in G\triangledown r$ the fact that H is an r-minor of G. We also note $H \in \mathscr{C}\triangledown r$ the fact that there is a graph G in $\mathscr{C}$ such that H is an r-minor of G.

We give below the basic definition of *nowhere-dense* graphs. It appears to be a very robust notion that can be defined by many other equivalent properties [67].

Definition 3.5 ([67]). Let $\mathscr{C}$ be a class of graphs. $\mathscr{C}$ is *nowhere dense* if and only if for all $r \in \mathbb{N}$ there is a $N_r \in \mathbb{N}$ such that $K_{N_r} \notin \mathscr{C}\triangledown r$. Here, K_n is the clique with n elements i.e. a graph with n vertices and every possible edges.

Nowhere dense seems to be one of the broadest class of graphs that admit good algorithmic properties.

Theorem 3.6 ([53, 74, 80]). *Let $\mathscr{C}$ be a class of nowhere dense graphs. For every* FO *formula φ and $\epsilon > 0$, there is an algorithm and a function f such that upon input of a graph $G \in \mathscr{C}$*

- *Decide whether $G \models \varphi$ in time $f(\|\varphi\|, \epsilon) \cdot \|G\|^{1+\epsilon}$ (when φ is a sentence).*
- *Output the number of solutions of $\varphi(G)$ in time $f(\|\varphi\|, \epsilon) \cdot \|G\|^{1+\epsilon}$.*
- *Enumerate $\varphi(\mathbf{D})$ with delay $f(\|\varphi\|, \epsilon)$ after precomputation time $f(\|\varphi\|, \epsilon) \cdot \|G\|^{1+\epsilon}$.*

In opposition to nowhere dense, one can define the notion of *somewhere dense* as follows: A class of graphs $\mathscr{C}$ is somewhere dense if and only if it is not nowhere dense [67]. Note that none of those two definitions imply closure under subgraphs. However, if a class is nowhere dense (resp. somewhere dense) then its closure by adding every possible subgraph remains nowhere dense (resp. somewhere dense). Combined with the following result, this leads to a complexity dichotomy for classes o structures closed under subgraphs.

Theorem 3.7 ([63]). *The model checking problem over somewhere dense classes of graphs that are closed under subgraphs is* AW[$*$] *complete, and therefore very unlikely to be in* FPT.

Recall that one can hope for an efficient enumeration algorithm for a query problem (in the sense of a constant delay algorithms after fixed polynomial time preprocessing), only when its Boolean counterpart, a.k.a. the model checking problem, admits an FPTalgorithm. Hence the result above also implies a dichotomy algorithm for enumeration tasks.

3.2 The end of the story for FO?

The results above emphasize the role played by sparsity for tractability of first order queries. They also establish a tractability frontier along the notion of somewhere dense structure for classes closed under subgraphs. But what can be said for arbitrary classes of structures?

Let us consider the class $\mathscr{C}$ of graphs of degree $O(\log n)$ where n is the number of vertices. For $k \in \mathbb{N}$, the graph G that contains:

- a clique of size k, and
- 2^k independent nodes,

is in $\mathscr{C}$. Hence, clearly, $\mathscr{C}$ is not closed under substructures: the subgraph of G restricted to its clique of size k, is not of degree bounded by $O(\log k)$. Such a class $\mathscr{C}$ can however be considered as

sparse. One elaborate from this class to the following notion of low degree structure.

Definition 3.8. A class $\mathscr{C}$ of structures has low degree if for every $\epsilon > 0$, there is a rank N in $\mathbb{N}$ such that for every G in $\mathscr{C}$, if $|G| > N$ then $d(G) \leq |G|^{\epsilon}$, where $d(G)$ is the degree of G.

As shown below, such classes of graphs still admit good algorithms.

Theorem 3.9 ([51]). *Let $\mathscr{C}$ be a class of graphs with* low degree. *For every* FO *formula φ and $\epsilon > 0$, there is an algorithm that upon input of a graph G in $\mathscr{C}$ decides whether $G \models \varphi$ in time $O(|G|^{1+\epsilon})$.*

This result uses some ideas that where at the core of Theorem 3.1 i.e. the locality of first-order logic. The difficulty when generalizing this to structures with low degree is that a given node does not have a neighborhood of bounded size but only of small size. This makes harder to obtain a constant delay enumeration (and not only an algorithm whose delay could be $O(n^{\epsilon})$. Fortunately, it is possible to work around this main issue.

Theorem 3.10 ([36]). *Let $\mathscr{C}$ be a class of graphs with low degree. There is an algorithm which, upon input of a structure $\mathbf{D}$ in $\mathscr{C}$ and an* FO *query φ, enumerates the set $\varphi(\mathbf{D})$ with constant-time delay after a pseudo-linear-time preprocessing.*

Beyond the particular case above, some recent tractability results have been obtained by considering first-order interpretations into formerly studied classes (such as bounded degree [47] or bounded expansion [48]).

3.3 Monadic queries

Monadic second-order logic, for short MSO extends first-order logic by allowing quantification over sets. When dealing with MSO queries, it is quite natural to look at graphs with bounded tree-width. The so-called *Courcelle's theorem* below has been one of the most influential tractability result in this area.

Theorem 3.11 (Courcelle's theorem [27]). *The model checking problem for* MSO *queries over classes of structures of bounded tree-width can be solved in linear time.*

Following this first result, other algorithms tasks such as counting [6] have been proved to be tractable over instances of bounded tree-width for MSO queries.

In contrast with first-order logic, the expressive power of MSO can be very high even on some natural classes of sparse (although of unbounded treewidth) structures. Given $m, n \in \mathbb{N}$, a (m, n)-grid is a graph on vertex set $\{1, ..., m\} \times \{1, ..., n\}$ with edge set:

$$\{((i, j), (i', j') : (i = i' \text{ and } |j - j'| = 1) \text{ or } (j = j' \text{ and } |i - i'| = 1))\}.$$

It is easily seen that MSO formulas can be used to describe a n steps, m space bounded computation of a Turing Machine when evaluated over a colored (m, n)-grid graph. Hence, there is few hope to find tractability results beyond treewidth. It is believed that Courcelle's theorem is somewhat optimal [52] (see also [62] for partial result under some complexity hypothesis).

3.3.1 Enumeration. The enumeration problem is also tractable. However with MSO queries we could potentially encounter free set variables i.e. be of the form $\varphi(\mathbf{x}, \mathbf{X})$ where $\mathbf{X}$ is a tuple of free second order variables.

This hardly allows constant delay enumeration since the size of a solution might be as large as the input size. Therefore just writing one solution could require linear time and two consecutive solutions may be "far" from each other. Indeed, consider the database $\mathbf{D}$ over the domain $D = \{1, ..., 2n\}$ with one relation E defined by $\{(a, 1) : a = 1, ..., n\} \cup \{(a, 2) : a = n + 1, ..., 2n\}$ and

$$\varphi(X) = \exists x \forall y \in X\ E(y, x) \wedge \forall y \notin X\ \neg E(y, x)$$

It holds $\varphi(\mathbf{D}) = \{\{1, 2, ..., n\}, \{n + 1, n + 2, ..., 2n\}\}$. The two solutions are disjoints and no algorithms can produce one after the other in constant time. The good measure for the delay is to take into account the *output length.*

The result below extends Theorem 3.11.

Theorem 3.12 ([8, 29]). *Let $\mathscr{C}$ be a class of structures with bounded tree-width, there is an algorithm and a function f which, upon input of a structure $\mathbf{D}$ in $\mathscr{C}$ and an* MSO *query φ, enumerates each solution s of the set $\varphi(\mathbf{D})$ with linear-time preprocessing and delay $f(|\varphi|) \cdot |s|$ i.e. linear in the output size.*

If $\varphi(\mathbf{x})$ contains only free first-order variables then the enumeration can be done with delay $f(|\varphi|)$ i.e in constant delay.

Alternatives proof for the constant delay bound, for queries with free first order variables only has also been found in [61] using quantifier elimination methods. Additional details can be found in two thesis, [9] and [58]. More recently, by using technics from knwoledge compilation, an alternative proof of Theorem 3.12 has also been given in [1].

4 ACYCLIC CONJUNCTIVE QUERIES AND BEYOND

In this section, we focus on tractable fragments of FO queries on arbitrary databases. Recall that conjunctive queries, CQ for short, are queries of the form:

$$\varphi(\mathbf{x}) := \exists \mathbf{y} \bigwedge_i R_i(\mathbf{z}_i)$$

where for every i, R_i is a relational symbol and $\mathbf{z}_i$ is a tuple of variables from $\mathbf{x}$ and $\mathbf{y}$. The combined complexity of CQ is NP-complete and, as such, the fragment is already too expressive. Below, we survey the complexity of a well-known restriction of CQ called acyclic conjunctive queries, ACQ. It turns out that for all algorithmic tasks under consideration tractability results can be found at the price of introducing new methods and measures. We also investigate possible extensions of ACQ and study their effects on the complexity of query evaluation.

Hypergraph of a query. An finite hypergraph $\mathcal{H} = (V, E)$ is a finite set V together with a subset E of the powerset of V, i.e. $E \subseteq \mathscr{P}(V)$. To each query φ, one can associate an hypergraph $\mathcal{H} = (V, E)$ whose vertex set is the set $var(\varphi)$ of variables of φ and hyperedge set is $atom(\varphi)$ the set of atoms of φ.

Queries. A query is said to be *self-join free* if no relation symbol is used more than once. A query is Boolean if it has no free variables that is $free(\varphi) = \emptyset$.

4.1 Acyclic conjunctive queries

A *join tree* of an hypergraph $\mathcal{H} = (V, E)$ is a tree T whose set of nodes is E the hyperedge set of $\mathcal{H}$ and whose edge set is such that:

- for all $v \in V$, $\{e \in E : v \in e\}$ the set of nodes of T in which v occurs is a connected sub-tree of T.

A conjunctive query is said to be α-acyclic, for short acyclic, if its associated hypergraph has a join-tree. The class of acyclic queries is denoted by ACQ. There are several notion of acyclicity for hypergraphs (see [13, 41]), α-acyclicity being the most general and admitting a number of alternative characterizations.

Example 4.1. The (path) query $\varphi_1(x, y, z) := E(x, y) \wedge E(y, z)$ is acyclic. The (triangle) query $\varphi_2(x, y, z) := E(x, y) \wedge E(y, z) \wedge E(z, x)$ is not acyclic. However, the more complex query $\varphi_3(x, y, z) := E(x, y) \wedge E(y, z) \wedge E(z, x) \wedge T(x, y, z)$ is acyclic: It admits a join tree with root $\{x, y, z\}$ that contain as three children $\{x, y\}$, $\{y, z\}$ and $\{z, x\}$.

Although this is a large class of queries, acyclic conjunctive queries are sufficiently restricted to obtained quite efficient algorithms as shown in the well-known result below.

Theorem 4.2 ([81]). *There is an algorithm which, upon input of a database* $\mathbf{D}$ *and* $\varphi(\mathbf{x}) \in$ ACQ*, computes the set* $\varphi(\mathbf{D})$ *in time* $O(\|\varphi\| \cdot \|\mathbf{D}\| \cdot \|\varphi(\mathbf{D})\|)$.

4.1.1 Efficient enumeration : from linear to constant delay. Yannakakis result can be turned into an enumeration process that outputs the solution one after the other with a linear delay as remarked in [11] and shown below.

Theorem 4.3 ([11]). *There is an algorithm which, upon input of a database* $\mathbf{D}$ *and* $\varphi \in$ ACQ*, enumerates the set* $\varphi(\mathbf{D})$ *with linear time preprocessing and linear time delay.*

Proof. It is easy to see that Algorithm 2 outputs solutions with a linear delay between each. One needs Theorem 4.2, to prove linearity of the base case ($p = 1$). The proof follows by induction. □

Algorithm 2 Enumeration of $\varphi(\mathbf{D})$

```
if p = 1 then
    for a ∈ φ(D) do
        output a
else
    let ψ1(x1) ≡ ∃x2 ... ∃xp φ(x1, ... xp)
    for a ∈ ψ(D) do
        let φa ≡ φ(a, x2, ..., xp)
        for b̄ ∈ φa(D) do
            output (a, b̄)
```

However, a delay which is linear in the size of the database delay between two solutions can hardly be considered has a very efficient time bound. It is easily seen, for example, that for quantifier free ACQ the delay can be made constant between two solutions. It turns out that constant delay can be achieved when free variables of a query can be grouped together while preserving acyclicity..

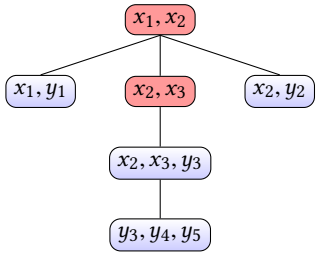

Figure 1: A join tree for a query $\varphi(\mathbf{x}) \equiv \exists \mathbf{y} R(x_1, x_2) \wedge S(x_2, x_3, y_3) \wedge R(x_1, y_1) \wedge T(y_3, y_4, y_5) \wedge S(x_2, y_2)$. A new hyperedge $\{x_2, x_3\} \subseteq \{x_2, x_3, y_3\}$ is introduced that help forming the join tree above (the subtree of red nodes contains free variables only)

Definition 4.4. An acyclic conjunctive query $\varphi(\mathbf{x})$ is *free-connex* if the extended query $\varphi'(\mathbf{x}) := \varphi(\mathbf{x}) \wedge R(\mathbf{x})$ is acyclic, where R is an arbitrary new symbol.

Alternatively, given an acyclic query $\varphi(\mathbf{x})$ with hypergraph $\mathcal{H} = (V, E)$, $\varphi(\mathbf{x})$ is free-connex if the hypergraph $\mathcal{H} = (V, E \cup \{\mathbf{x}\})$ is also acyclic. Note that boolean queries or queries with only one free variable are by definition free-connex. It is not the case for binary queries.

Example 4.5. The query: $\varphi(x, y) = \exists w \exists z\, E(x, w) \wedge E(y, z) \wedge B(z)$. is free-connex because the query $\varphi'(x, y) = \exists w \exists z\, E(x, w) \wedge E(y, z) \wedge B(z) \wedge R(x, y)$ is still acyclic.

The Boolean matrix multiplication query: $\Pi(x, y) = \exists z\, A(x, z) \wedge B(z, y)$. Since $\Pi'(x, y) = \exists z\, A(x, z) \wedge B(z, y) \wedge R(x, y)$ is clearly not acyclic, $\Pi(x, y)$ is not free-connex.

It turns out that free-connexity is a sufficient restriction to put on acyclic conjunctive queries in order to obtained constant delay enumeration. To see this, remark that being free-connex intuitively permits to treat free variables together in an evaluation algorithm. Equivalently, given a free-connex $\varphi(\mathbf{x})$ of hypergraph $\mathcal{H} = (V, E)$, it can be transformed into an equivalent query $\varphi'(\mathbf{x})$ of hypergraph $\mathcal{H}' = (V, E')$ by possibly adding atoms/hyperedges $e' \subseteq e \in E$ such that the associated join tree of $\varphi'(\mathbf{x})$ contains as root a connected subtree made of free variables only (corresponding to a).

Figure 1 illustrates this property. Given query $\varphi(\mathbf{x}) \equiv \exists \mathbf{y} R(x_1, x_2) \wedge S(x_2, x_3, y_3) \wedge R(x_1, y_1) \wedge T(y_3, y_4, y_5) \wedge S(x_2, y_2)$, one can introduce a new atom $S'(x_2, x_3)$ such that $\varphi(\mathbf{x})' \equiv \varphi(\mathbf{x}) \wedge S'(x_2, x_3)$ has the described rooted join tree. Given a database $\mathbf{D}$, To enumerate $\varphi(\mathbf{D})$, one first applies the bottom-up Yannakakis algorithm for acyclic conjunctive query as precomputation steps to filter relations by projecting out existentially quantified variables until only the join $R(x_1, x_2) \wedge S'(x_2, x_3)$ remains to be evaluated on the filtered relations. More precisely, one successively computes:

- $S \leftarrow \{(a, b, c) :$ there exists $d, e : S(a, b, c)$ and $T(c, d, e)\}$
- $S' \leftarrow \{(a, b) :$ there exists $c : S(a, b, c)\}$
- $R \leftarrow \{(a, b) :$ there exists $c : R(a, b)$ and $R(a, c)\}$
- $R \leftarrow \{(a, b) :$ there exists $c : R(a, b)$ and $S(b, c)\}$

From here, after sorting R on its second coordinate and S' on its first coordinate, enumerating the results i.e. the (a, b, c) s.t. $(a, b) \in R$ and $(b, c) \in S'$ can be easily done with constant delay. This approach yields to the following result.

Theorem 4.6 ([11]). *There is an algorithm which, upon input of a database* $\mathbf{D}$ *and a free-connex acyclic conjunctive query* φ*, enumerates the set* $\varphi(\mathbf{D})$ *with linear-time preprocessing and constant-time delay.*

4.1.2 Matching lower bounds. In [11], it is also proved that, when looking at acyclic conjunctive queries, free-connex is what is needed to obtain constant delay enumeration, assuming some reasonable algorithmic hypothesis.

The Boolean matrix multiplication is the problem defined as follows: given two $n \times n$ Boolean matrices A and B, compute their product $A \times B$. This is the problem defined by query $\Pi(x, y)$ in Example 4.5. Suppose $\mathbf{D}_{BM}$ is a database of domain $[n]$ with binary relations R_A, R_B such that for all $a, b \in [n]$, $(a, b) \in R_A$ (resp. R_B) iff there is a 1 in line a, column b of matrix A (resp. B). If, on any such $\mathbf{D}_{BM}$, there exists a constant delay algorithm to enumerate $\Pi(\mathbf{D}_{BM})$, then, computing the product $A \times B$ would need only $O(n^2)$ steps.

So far, the best known algorithm for matrix multiplication (based on the Coppersmith-Winograd algorithm [26]) requires than $O(n^\omega)$ with $\omega = 2.37$ steps [49]. We call **Mat-Mul** the widely believed hypothesis that given two such A and B their product can not be computed in quadratic $O(n^2)$ time.

In [11], it is proved, by reduction, that a (self-join free) query that is not free-connex is at least as hard as query $\Pi(x, y)$. Indeed, let $\varphi(\mathbf{x}) \in$ ACQ such that φ is not free-connex. By definition $ar(\varphi) = m \geq 2$. Let $n \in \mathbb{N}$ and $\mathbf{D}_{BM}$ as above. One can construct a database $\mathbf{D}$, on domain $\{0, ..., n-1\} \cup \{\perp\}$ where $\perp$ is new symbol, in time linear in $\|\mathbf{D}_{BM}\|$ such that, up to reordering of variables:

$$\varphi(\mathbf{D}) = \Pi(\mathbf{D}_{BM}) \times \{\perp\}^{m-2}$$

Hence, there is a one-one mapping from $\varphi(\mathbf{D})$ to $\Pi(\mathbf{D}_{BM})$.

Example 4.7. Recall $\Pi(x, y) \equiv \exists z\ A(x, z) \wedge B(z, y)$. Let $\varphi(\mathbf{x}) \equiv E(x_1, x_4) \wedge S(x_1, x_1, x_3) \wedge T(x_3, x_2, x_4)$. Variable x_1, x_2 can play the role of x, y in $\Pi(x, y)$ and x_3 that of z. Database $\mathbf{D}$ contains the relations: $E = \{(a, \perp) : a \in [n]\}$, $S = \{(a, a, b) : (a, b) \in A\}$, $T = \{(b, c, \perp) : (b, c) \in B\}$.

The result below formalizes the above discussion.

Theorem 4.8 ([11]). *Assuming* ***Mat-Mul****, for any self-join free,* $\varphi \in$ ACQ *the following statements are equivalent:*

- φ *is free-connex,*
- *there is an algorithm which, upon an input database* $\mathbf{D}$*, enumerates the set* $\varphi(\mathbf{D})$ *with linear-time preprocessing and constant-time delay,*
- *there is an algorithm which, upon an input database* $\mathbf{D}$*, computes the set* $\varphi(\mathbf{D})$ *in time* $O(\|\mathbf{D}\| + \|\varphi(\mathbf{D})\|)$

The hardness part of the above characterization can be extended beyond acyclic queries. An hypergraph $\mathcal{H}$ is k-uniform if all its hyperedges contain k vertices. An l-hyperclique in a k-uniform hypergraph $\mathcal{H}$ is a set V' of $l > k \geq 2$ vertices, such that every subset of V' of size k forms a hyperedge. Finding such an l-hyperclique is conjectured to require more than $n^{k-o(1)}$ (see [65]). We refer to the hypothesis that finding a k-hyperclique in a $(k-1)$-uniform hypergraph $\mathcal{H}$ can not be done in $O(n^{k-1})$ as **Hyperclique** (for $k = 3$, seeing 2-uniform hypergraphs as graphs, this amounts to find a triangle in a graph in time $O(n^2)$). In [18], it is shown that assuming **Hyperclique**, no cyclic query can be enumerated with linear time preprocessing and constant delay. Combining the results of [11, 18] we get that assuming both **Mat-Mul** and **Hyperclique**, Theorem 4.8 can be generalized to any self-join free $\varphi \in$ CQ.

Theorem 4.9 ([11, 18]). *Assuming* ***Mat-Mul*** *and* ***Hyperclique****, for any self-join free* $\varphi \in$ CQ*:*

- *either* φ *is free-connex and* $\text{enum} \cdot \varphi \in \text{Constant-Delay}_{\text{lin}}$,
- *or* $\text{enum} \cdot \varphi \notin \text{Constant-Delay}_{\text{lin}}$

4.2 Union of conjunctive queries

Definition 4.10. A formula $\varphi(\mathbf{x})$ is a union of conjunctive query, denoted UCQ, if it is of the form

$$\varphi = \varphi_1 \vee \cdots \vee \varphi_k$$

where each $\varphi_i \in$ CQ, $i \leq k$.

Let $\varphi = \varphi_1 \vee \cdots \vee \varphi_k \in$ UCQ. If each $\varphi_i \in$ ACQ is free connex, it can been shown using technics developed in [79] that $\text{enum} \cdot \varphi \in \text{Constant-Delay}_{\text{lin}}$.

However, as remarked in [22], efficient enumeration can be obtain for union of queries even though some of them are not efficiently enumerable. One can easily anticipate the following situation, given two queries $\varphi_1(\mathbf{x})$, $\varphi_2(\mathbf{x})$ and a database $\mathbf{D}$:

- $\varphi_2(\mathbf{D})$ can be enumerated easily (say constant delay)
- $\varphi_1(\mathbf{D})$ can not be enumerated easily (is not free-connex for example)
- But each solution of $\varphi_1(\mathbf{D})$ can be obtained by constant time computation from some solution of $\varphi_2(\mathbf{D})$

In that case enumerating $(\varphi_1 \vee \varphi_2)(\mathbf{D})$ can be enumerated with constant delay. Indeed, let's consider the following example:

$$\begin{aligned} \varphi_1(x, y, w) &= R_1(x, z) \wedge R_2(z, y) \wedge R_3(x, w) \\ \varphi_2(x, y, w) &= R_1(x, y) \wedge R_2(y, w) \end{aligned} \tag{1}$$

Query φ_2 is free-connex while φ_1 is not. However, for any database $\mathbf{D}$, one remark that any solution of $\varphi_1(\mathbf{D})$ can be built from a solution of $\varphi_2(\mathbf{D})$: for a tuple (a, b, c) to be in $\varphi_1(\mathbf{D})$, there should be an element d in the domain such that $(a, d, b) \in \varphi_2(\mathbf{D})$. Hence, to enumerate the union, one can proceed as follows: each time a solution (a, d, b) is produced, one look for all c such that $R_3(a, c)$ is true and enumerate all tuples (a, b, c). No solution of $\varphi_1(\mathbf{D})$ will be missed and since $\varphi_2(\mathbf{D})$ can be enumerated with constant delay, the total delay for the enumeration of the union can be preserved to be constant (though one also has to deal with duplicates also which can be done. See [22]). Such an example shows that charting the tractability frontier for union of conjunctive queries is a challenging task.

In [22] some partial answers to the problem are given. One key concept it that of a query *providing variables* to some other.

Definition 4.11. Let $\varphi_1, \varphi_2 \in$ CQ. Query φ_2 provides a set of variables V_1 to φ_1 if:

- There is a body-homomorphism h from φ_2 to φ_1 i.e. $h : var(\varphi_2) \to var(\varphi_1)$ such that for every atom $R(\mathbf{x}) \in atom(\varphi_2)$, $R(h(\mathbf{x})) \in atom(\varphi_1)$
- $h^{-1}(V_1) \subseteq free(\varphi_2)$
- There is $h^{-1}(V_1) \subseteq S \subseteq free(\varphi_2)$, such that φ_2 is S-connex.

Going back to Equation 1, it is easy to see that φ_2 provides variable set $\{x, z, y\}$ to φ_1.

Let us now consider a query $\varphi_1^+(x, y, w)$ obtained from $\varphi_1(x, y, w)$ adding the variables provided by $\varphi_2(x, y, w)$ in a new atom $P_1(x, z, y)$:

$$\varphi_1^+(x, y, w) = R_1(x, z) \wedge R_2(z, y) \wedge R_3(x, w) \wedge P_1(x, z, y).$$

Obviously $\varphi_1^+(x, y, w)$ is free-connex. The idea behind this example can be formalized. Given $\varphi = \bigvee_{i=1}^{k} \varphi_i \in$ UCQ, with each $\varphi_i(\mathbf{x}) = \bigwedge_{j=1}^{h} R_j(\mathbf{x}_j)$, a *union extension* of $\varphi(\mathbf{x})$ is a syntactic enrichment of the following form:

$$\varphi_i^+(\mathbf{x}) = \bigwedge_{j=1}^{h} R_j(\mathbf{x}_j) \wedge \bigwedge_{j=1}^{s} P_j(\mathbf{v}_j)$$

where each $P_1, ..., P_s$ is a fresh relational symbol and each $\{\mathbf{v}_j\}$ is provided to φ_i by some φ_j (or, more generally, by way of recursion, by a union extension of some φ_j).

Definition 4.12. Given $\varphi = \varphi_1 \vee \cdots \vee \varphi_k \in$ UCQ, query φ is free-connex if each of φ_i, $i = 1, ..., k$ admits a union extension which is free-connex

Building on this, it can be proved that:

Theorem 4.13 ([22]). *Let $\varphi \in$ UCQ. If φ is free-connex then enum$\cdot\varphi \in$ Constant-Delay$_{\text{lin}}$.*

Complete characterizations have been obtained in case φ is the union of *two* intractable CQ or the union of two particular acyclic conjunctive queries (called *Body isomorphic*), see [22]), some lower bound can be proved and it can be shown that free-connexity fully captures tractability. However, proving a full classification for UCQ is an open problem.

4.3 Allowing comparisons and disequalities

For numerical data, it could be interesting to extend ACQ by adding comparisons operators such as $<, \leq, \neq$. In this section, we examine the effect in terms of expressive power of adding such features.

Definition 4.14. Let $\lhd \in \{\leq, <, \neq\}$. A formula $\varphi(\mathbf{x})$ is an acyclic conjunctive query with comparisons in S, i.e. is in ACQ$_{\lhd}$, if it is of the form:

$$\varphi(\mathbf{x}) := \exists \mathbf{y}\ \phi(\mathbf{x}) \wedge \bigwedge_i z_i \lhd z_i'$$

where $\phi(\mathbf{x})$ is an acyclic conjunctive query, each z_i, z_i' are variables among $\mathbf{x}$ and $\mathbf{y}$.

In the definition above, comparisons are not taken into account to measure acyclicity. Interpretation of $<, \leq, \neq$ on a database with domain $[n]$ has its obvious meaning.

In [69], an interesting example is given that emphasizes the gain of expressive power compared to ACQ when considering ACQ$_<$. Let $G = (V, E)$ be a graph with $V = \{0, ..., n-1\}$ (hence implicitly ordered) and $k \in \mathbb{N}$. Let $\mathbf{D}$ be a database with binary relations P and R over a domain D containing all integers $(i+j)n^3 + |i-j|n^2 + bn + i$ for $i, j \in V$ and $b = 0, 1$. Let us denote $[i, j, b]$ such an element of the domain. Let relation P and R be defined as:

- $P([i, j, 0], [i, j, 1])$ iff $(i, j) \in E$, for all $i, j \in V$ (it is also supposed that E has self-loops for each $i \in V$)
- $R([i, j, 1], [i, j', 0])$ for all $i, j, j' \in V$

Being a clique can not be defined directly without introducing a cycle. Instead, one plays with the underlying order and define the following φ over existentially quantified variables x_{ij}, y_{ij} for $1 \leq i, j \leq k$ as:

$$\bigwedge_{1 \leq i, j \leq k} P(x_{ij}, y_{ij}) \wedge \bigwedge_{1 \leq i, j, l = i+1 \leq k} R(y_{ij}, x_{il}) \wedge \bigwedge_{1 \leq i < j \leq k} x_{ij} < x_{ji} < y_{ij}$$

The query φ is clearly acyclic: it consist in k paths of length $2k-1$ connecting each $x_{i1}, y_{i1}, x_{i2}, ..., y_{ik}$. Considered separately, even the graph of comparisons is acyclic. However, it can be shown that:

$$G \text{ has a clique of size } k \text{ iff } \mathbf{D} \models \varphi.$$

One direction is obvious: if v_1, v_2, v_k form a clique in G with $v_1 < v_2 < ... < v_k$. Interpreting x_{ij} by $[v_i, v_j, 0]$ and $y_{ij} = [v_i, v_j, 1]$ the formula is trivially satisfied. For the other direction, one can deduce from the inequalities that, for each $i < j$ if $x_{ij}, y_{ij}, x_{ji}, y_{ji}$ are respectively interpreted by $[v_i, v_j', 0]$, $[v_i, v_j', 1]$, $[v_j, v_i', 0]$ and $[v_j, v_i', 1]$ then $v_i = v_i'$ and $v_j = v_j'$. Then, the satisfaction of each $P(x_{ij}, y_{ij})$ implies that each $(v_i, v_j) \in E$ and the vertices $v_1, ..., v_k$ form a clique. Formally, this helps proving the following result.

Theorem 4.15 ([69]). *Evaluating queries in* ACQ$_<$ *and* ACQ$_\leq$ *is $W[1]$-complete for both the size of the query and its number of variables as parameters.*

However, allowing disequalities only, i.e. atoms of the form $x \neq y$, leads to a different situation. In [69], it is also shown that evaluating a query $\varphi \in$ ACQ$_\neq$ on a database $\mathbf{D}$ is in FPT with the query size as parameter. More precisely, it can be done in time:

$$f(\|\varphi\|) \cdot \|\varphi(\mathbf{D})\| \cdot \|\mathbf{D}\| \cdot \log^2 \|\mathbf{D}\|.$$

for some function f such that $f(\|\varphi\|) \leq 2^{O(v \log v)} \cdot \|\varphi\|$ where v is the number of variables of φ. Surprisingly, using combinatorial arguments, it can be shown that free connexity is still the criteria for fast enumeration even in the the presence of disequalities. A convenient way to see this is through, again, a mechanism of quantifier elimination. One can at the same time eliminate variables and disequality constraints. Let us illustrate the method.

A database $\mathbf{D}$ on domain D and relations $R_1, \ldots, R_s$ can be seen as a functional structure $\mathbf{F} = \langle F; D, D_1, \ldots, D_s, f_1, \ldots, f_p \rangle$ where $p = \max_{i \leq s} ar(R_i)$ and :

- $D, D_1, \ldots, D_s$ are disjoint unary relations so that the domain F of $\mathbf{F}$ is the disjoint union of $D, D_1, .., D_s$ and $\{\perp\}$ i.e. $F = D \uplus D_1 \uplus \ldots \uplus D_1 \uplus \{\perp\}$
- For each $i = 1, \ldots, s$, D_i is a set of elements representing tuples in R_i, and $\perp$ is an extra element.

- For $1 \leq j \leq p$, f_j is a unary function: $D' \to D \cup \{\perp\}$ such that, for every $t = (t_1, \ldots, t_{a_i}) \in D_i$, we have $f_j(t) = t_j$ for $1 \leq j \leq a_i$ and $a_i = arR_i$, and $f_j(t) = \perp$ otherwise.

Any conjunctive acyclic query with or without disequalities can be transformed into an acyclic conjonctive query (in the graph sense) for this functional representation of data. Indeed, consider the following acyclic queries:

$$\varphi(x, y, z) \equiv \exists t R_1(x, y) \wedge R_2(y, z) \wedge R_3(x, z, t) \wedge R_4(x, y, z, t)$$

$$\phi(x, y, z) \equiv \varphi(x, y, z) \wedge y \neq z \wedge y \neq t \wedge z \neq t$$

Let T be a join tree for φ. For each atomic subformula A_i in φ, one introduce a variable e_i and for each connected pair of vertices (A_i, A_j) in T, one describe the variables A_i and A_j have in common by introducing projection functions $f_1, ..., f_4$. The following functional query $\varphi'(x, y, z)$ can be built.

$$\begin{aligned}\varphi'(x, y, z) \equiv \quad & \exists e_1 e_2 e_3 e_4 \\ & D_1(e_1) \wedge D_2(e_2) \wedge D_3(e_3) \wedge D_4(e_4) \wedge \\ & f_1(e_1) = x \wedge f_2(e_1) = y \wedge f_2(e_2) = z \wedge \\ & f_1(e_1) = f_1(e_4) \wedge f_2(e_1) = f_2(e_4) \wedge \\ & f_1(e_2) = f_2(e_4) \wedge f_2(e_2) = f_3(e_4) \wedge \\ & f_1(e_3) = f_1(e_4) \wedge f_2(e_3) = f_3(e_4) \wedge f_3(e_3) = f_4(e_4)\end{aligned}$$

Similarly, ϕ' above is the result of the transformation of ϕ:

$$\begin{aligned}\phi'(x, y, z) \equiv \quad & \varphi'(x, y, z) \wedge \\ & f_1(e_1) \neq f_2(e_1) \wedge f_2(e_1) \neq f_2(e_2) \wedge f_2(e_2) \neq f_1(e_1)\end{aligned}$$

To each query φ of functional signature σ, one can associate its underlying graph $G = (V, E)$ with $V = var(\varphi)$ and E defined by $(x, y) \in E$ iff there exists an atomic formula $(x) = g(y)$ for $f, g \in \sigma$. A query will be called acyclic if its associated graph is a tree. The graph of formula $\varphi'(x, y, z)$ (and $\phi'(x, y, z)$) above is easily seen to be acyclic. It is easily seen that a conjunctive query is acyclic, its functional translation is.

Inequality ($\leq$, $<$) constraints may force a relative ordering between the possible range of the variables and then permit to express global constraints (recall the definition of the k-clique problem by acyclic queries with inequality). By contrast, disequalities only introduces exception in the possible interpretations: a constraint such as $x \neq y$ only says that among the possibly large interpretation set for x and y, one must choose distinct values. Even repeated for all pairs of variables, this could be handled by combinatorial arguments that carry on the formula only. This idea can be handled through the notion of *cover* of a table below.

Definition 4.16. Let E, F be two finite sets and $\mathbf{f}$ a tuple of k functions s.t. $\mathbf{f} : E \to F^k$. A *cover* $\mathbf{c}$ of a table $(E, \mathbf{f})$ is a tuple $(c_1, \ldots, c_k) \in (F \cup \{\sqcup\})^k$ such that, for all $x \in E$, there exists some $i \leq k$, such that $c_i = f_i(x)$. We denote by $\textsc{covers}(E, \mathbf{f})$ the set of covers of $(E, \mathbf{f})$.

Covers can be compared according to the following definition.

Definition 4.17. A cover $\mathbf{c}'$ is *more general* than a cover $\mathbf{c}$, denoted $\mathbf{c}' \leq \mathbf{c}$ if, for all $i \leq k$, either $c_i = c'_i$ or $c'_i = \sqcup$. A cover $\mathbf{c}$ of a table $(E, \mathbf{f})$ is *minimal* if this table has no more general cover.

Example 4.18. Provided $\mathbf{c}' = (2, 1, \sqcup)$ and $\mathbf{c} = (2, 1, 1)$ are covers of a set E by a triple $\mathbf{f} = (f_1, f_2, f_3)$ then, $\mathbf{c}'$ is more general than $\mathbf{c}$.

The minimal cover set $\textsc{min-covers}(E, \mathbf{f})$ is the set of all minimal covers of of $(E, \mathbf{f})$. A *representative set* of $(E, \mathbf{f})$ is a subset $E' \subseteq E$ such that $\textsc{covers}(E, \mathbf{f}) = \textsc{covers}(E', \mathbf{f})$.

Example 4.19. Let $E = \{a, b, c, d, e\}$, $F = \{1, 2, 3, 4, 5\}$ and $\mathbf{f} = (f_1, f_2, f_3)$ be the following tuple of unary functions over E:

	f_1	f_2	f_3	f_4
a	1	2	4	5
b	1	5	1	5
c	3	2	4	5
d	3	5	3	5
e	5	2	4	5
f	2	2	4	5

The complete cover set of $\mathbf{f}$ over $T(E, \mathbf{f})$ contains 64 tuples, that is the following tuples: $(1, 2, 3, *_4)$, $(1, 5, 4, *_4)$, $(3, 2, 1, *_4)$, $(\sqcup, 5, 4, *_4)$ and $(*_1, *_2, *_3, 5)$ where $*_1 \in \{\sqcup, 1, 2, 3, 5\}$, $*_2 \in \{\sqcup, 2, 5\}$, $*_3 \in \{\sqcup, 1, 3, 4\}$ and $*_4 \in \{\sqcup, 5\}$ (a rough count gives $68 = 2+2+2+2+60$ but 4 of them are counted twice). The (smaller) minimal cover set of $(E, \mathbf{f})$ is of size 4: $\{(1, 2, 3, \sqcup), (3, 2, 1, \sqcup), (\sqcup, 5, 4, \sqcup), (\sqcup, \sqcup, \sqcup, 5)\}$. A representative set is $\{a, b, c, d\}$.

Also, given a k-tuple $\mathbf{c} = (c_1, ..., c_k)$ and $i \in \{1, ..., k\}$, $\mathbf{c}_{-i} = (c_1, .., c_{i-1}, c_{i+1}, ..., c_k)$. Similarly for $\mathbf{f}_{-i}$. Let $a \in E$, and $i \in \{1, ..., k\}$, let's denote $E_i^a = \{x \in E : f_i(x) \neq f_i(a)\}$. Intuitively, E_i^a is what remains to be covered of E by $\mathbf{c}_{-i}$ when $c_i = f_i(a)$ has been chosen is some cover $\mathbf{c}$.

The key remarks rely on the following easily proved combinatorial results. For every table $(E, \mathbf{f})$:

(1) $|\textsc{min-covers}(E, \mathbf{f})| \leq k!$.
Let $E \neq \emptyset$ and $a \in E$. It is easily seen that, for all tuple $\mathbf{c}$: $\mathbf{c} \in \textsc{covers}(E, \mathbf{f})$ iff there exists $i \leq k$, s.t. $c_i = f_i(a)$ and $\mathbf{c}_{-i} \in \textsc{covers}(E_i^a, \mathbf{f}_{-i})$. When $E_i^a = \emptyset$, the cover can be completed by $\sqcup$ to make it minimal. So, the number of minimal covers for $(E, \mathbf{f})$ is bounded by $g(k)$ verifying: $g(0) = 1$ and $g(i) = i \cdot g(i-1)$ for $i \leq k$, hence $g(k) = k!$.

(2) There exists a representative set of cardinality bounded by $O(k!)$.
Easily seen by recursively choosing some $a \in E$ at each step of the process described above.

Let $\mathbf{F}$ be a functional structure. Let $\varphi(\mathbf{x})$ be an acyclic conjunctive query with disequalities over a signature σ containing only unary function symbols. Let T be the associated tree of $\varphi(\mathbf{x})$. Suppose $\varphi(\mathbf{x}) \equiv \exists z \psi(\mathbf{x}, z)$, that z is a leaf of T and y is a variable of $\mathbf{x}$ be the parent of z in T. Query $\varphi(\mathbf{x})$ can be written as follows:

$$\varphi(\mathbf{x}) \equiv \psi_0(\mathbf{x}) \wedge \exists z (P(z) \wedge \bigwedge_{1 \leq i \leq m} g_i(z) = g'_i(y) \wedge \bigwedge_{1 \leq i \leq k} f_i(z) \neq f'_i(\mathbf{x}))$$

where each $g_i, g'_i, f_i, f'_i \in \sigma$, $P(z)$ is a subformula involving only z. Term $f'_i(\mathbf{x})$ stands for $f'_i(x_j)$ for some x_j of $\mathbf{x}$. Query $\varphi(\mathbf{x})$ can be rewritten as:

$$\psi_0(\mathbf{x}) \wedge \exists z \in E(y) \bigwedge_{1 \leq i \leq k} f_i(z) \neq f'_i(\mathbf{x})$$

where $E(y)$ stands as a shortcut for $P \cap \bigcap_{i \leq m} g_i^{-1}(g_i'(y))$. Note that a partition of the domain can be obtained using pairwise distincts $E(y)$, $y \in \mathrm{Dom}(\mathbf{F})$. Then, the query can be expressed as:

$$\psi_0(\mathbf{x}) \wedge \mathbf{f}'(\mathbf{x}) \notin \textsc{cover}(E(y), \mathbf{f}).$$

Considering $E'(y)$ a minimal representative set of $(E(y), \mathbf{f})$, query $\varphi(\mathbf{x})$ can be equivalently seen as:

$$\psi_0(\mathbf{x}) \wedge \mathbf{f}'(\mathbf{x}) \notin \textsc{cover}(E'(y), \mathbf{f})$$

.

But each $E'(y)$ is of size bounded by some $h = O(k!)$. Hence, the database $\mathbf{F}$ can be enriched into a new database $\mathbf{F}'$ by adding a linear number of new information bits describing, for each $a \in F$, the content of the minimal representative set $E'(a)$ and the full description of its minimal cover. More precisely, let E_j, $f_{i,j}$, $i = 1, ..., k$, $j = 1, ..., h$

- $y \in E_j \Leftrightarrow |E'(y)| \geq j$;
- $v_j(y)$ is the j^{th} element of $E'(y)$ if $|E'(y)| \geq j$; otherwise it is an arbitrary value ;
- set $f_{i,j} = f_i \circ v_j$.

This yields a new formula $\varphi'(\bar{x})$ equivalent to $\varphi(\mathbf{x})$

$$\varphi'(\mathbf{x}) \equiv \bigvee_{1 \leq j \leq h} \psi_j(\mathbf{x}) \text{ where}$$

$$\psi_j(\mathbf{x}) \equiv \psi_0(\mathbf{x}) \wedge E_j(y) \wedge \bigwedge_{1 \leq i \leq k} f_{i,j}(y) \neq f_i'(\mathbf{x}).$$

Hence, $\varphi(\mathbf{x})$ can be rewritten into $\varphi'(\mathbf{x})$ a union of acyclic conjunctive queries without existentially quantified variable z such that:

$$\varphi(\mathbf{F}) = \varphi'(\mathbf{F}').$$

Database $\mathbf{F}'$ can be obtained in linear time in $\|\mathbf{F}\|$. By iterating this process on Boolean query with a richer one can obtain a union of acyclic and quantifier free conjunctive queries. More generally, the total time to compute the result of an $\mathrm{ACQ}_{\neq}$ query can be lowered to:

$$f(\|\varphi\|) \cdot \|\varphi(\mathbf{D})\| \cdot \|\mathbf{D}\|.$$

for some function f and the enumeration of the result can be done in linear (in the size of the database) delay. Combining with methods of theorem 4.8, the following holds.

Theorem 4.20 ([11]). *Assuming **Mat-Mul**, any self-join free query $\varphi \in \mathrm{ACQ}_{\neq}$ can be enumerated with linear-time preprocessing and constant-time delay if and only if it is free-connex.*

Note that the result above implies that for free-connex acyclic queries with inequalities the total time for query answering is

$$f(\|\varphi\|) \cdot (|\varphi(\mathbf{D})| + \|\mathbf{D}\|).$$

Additional extensions for enumeration. Numerous other variants of CQ have been considered from the point of view of enumeration. For example : extensions of CQ with functional dependencies [21], tree like databases with $\underline{\mathrm{X}}$ properties [10] , random acces random order enumeration for [23]. A much thorough review of constant delay enumeration can be found in [14].

4.4 Counting results of ACQ queries

We consider here the counting problem, denoted $\sharp$CQ, associated to conjunctive query answering, that is: given $\varphi \in$ CQ, given a database $\mathbf{D}$, return $|\varphi(\mathbf{D})|$ i.e. the number of solutions of query φ over $\mathbf{D}$. Such task is usually harder than deciding. However, as shown in this section, large islands of tractability can be found and the frontier between tractable and intractable problems be delineated.

As a generalization of #3SAT, it is easy to see that counting solutions to quantifier free conjunctive queries, for short the $\sharp\mathrm{CQ}_0$ problem, which correspond to queries without projections in the database view, is $\sharp$P-complete. The effect of adding existential quantification (i.e. projections) really increases the expressive power: it is proved that counting solutions of general conjunctive queries becomes even harder than $\sharp$P and that $\sharp$CQ is $\sharp\cdot$NP-complete [12].

What is the situation for acyclic queries? We first present a tractability result that also hold in the more general context, introduced below, of weighted counting. Let $\mathbb{F}$ be a field and $\mathbf{D}$ be a finite structure of domain D. A $\mathbb{F}$-weight function for $\mathbf{D}$ is a mapping $w : D \to \mathbb{F}$. If $\mathbf{a}$ is a tuple of elements of D of length k, the weight of $\mathbf{a}$ is

$$w(\mathbf{a}) = \prod_{i=1}^{k} w(a_i).$$

The *weighted* counting problem for CQ, denoted $\sharp_{\mathbb{F}}$CQ, is the following problem: given $\varphi \in$ CQ, given $\mathbf{D}$ and an $\mathbb{F}$-weighted function w, return the sum of the weights of all solutions, i.e., the value of

$$\sum_{\mathbf{a} \in \varphi(\mathbf{D})} w(\mathbf{a}).$$

When φ is acyclic the counting and the weighted counting problems are respectively denoted by $\sharp$ACQ and $\sharp_{\mathbb{F}}$ACQ.

Not too surprisingly, the weighted counting problem associated to projection (i.e. quantifier) free acyclic query $\sharp_{\mathbb{F}}\mathrm{ACQ}_0$ is tractable in a very strong sense (one suppose that, in $\mathbb{F}$, arithmetic operations can be handled in polynomial time).

Theorem 4.21. *The combined complexity of $\sharp_{\mathbb{F}}\mathrm{ACQ}_0$ is polynomial time see [34]). In particular, there is an algorithm that upon input a quantifier free acyclic query φ and a database $\mathbf{D}$, output $|\varphi(\mathbf{D})|$ in time $O(\|\varphi\| \cdot \|\mathbf{D}\|^2)$ (see [70]).*

More surprisingly, the effect of adding even a single existential quantification to acyclic queries results in a huge gain of expressive power. Let $G = (V, E)$ be a bipartite graph $V = A \cup B$, $A = \{a_1, ..., a_n\}, B = \{b_1, ..., b_n\}$. Let's consider the following queries ($a_1, ..., a_n$ can be viewed as constants):

$$\varphi(x_1, ..., x_n) = \bigwedge_{i=1}^{n} E(a_i, x_i), \quad \psi(x_1, ..., x_n) = \exists t \bigwedge_{i=1}^{n} E(a_i, x_i) \wedge E(t, x_i) \tag{2}$$

A careful analysis shows that the number of perfect matchings in G is equal to

$$|\varphi(G)| - |\psi(G)|.$$

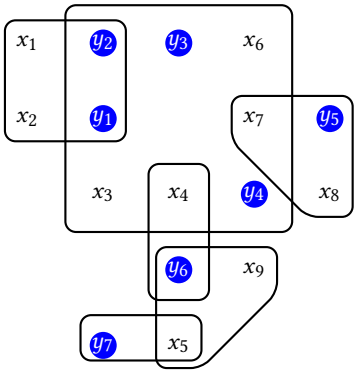

Figure 2: The hypergraph $\mathcal{H}$ of a query $\varphi(\mathbf{y})$ with $S = \mathit{free}(\varphi) = \{y_1, ..., y_7\}$.

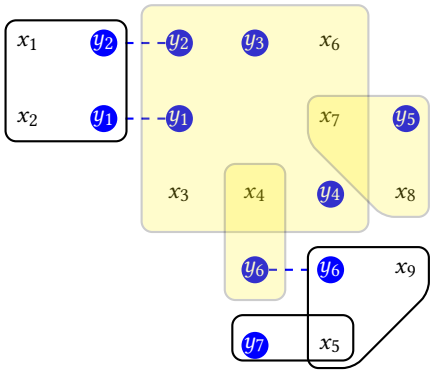

Figure 3: The decomposition of hypergraph $\mathcal{H}$ of Figure 2 into three edge distinct S-components. The central component (in yellow) contains an independent set of size 3 which is here the maximal size among all independent set of S-components

Consequently, the combined complexity for counting ACQ is #P-complete [70] and, as shown by the preceding example, even if the query as only one quantified variable. To summarize:

Theorem 4.22. *The combined complexity of*

- $\sharp\text{CQ}_0$ *and* $\sharp$ACQ *are* $\sharp$P*-complete [70]*
- $\sharp$CQ *is* $\sharp\cdot$NP*-complete [12]*

A particularity of formula $\psi(\mathbf{x})$ in Equation 2 is that the free variables are not connected to each other and the hypergraph (actually, a graph) of $\psi(\mathbf{x})$ forms a star whose center is the unique quantified variable t. Elaborating on that remark, a new parameter, called *quantifier-star size*, was introduced in [34], to measure how free variables are spread in the formula. This will be central to characterize tractability for counting.

Let $\varphi(\mathbf{x})$ be an acyclic conjonctive query, $\mathcal{H} = (V, E)$ its associated hypergraph and $S \subseteq V$ such that $S = \mathit{free}(\varphi)$. We suppose to allege the notation that there is no edges $e \in E$ such that $e \subseteq S$ i.e. fully included in the free variables set S. For $E' \subseteq E$, the hypergraph induced by E' is $\mathcal{H}[E'] = (\bigcup_{e \in E'} e, E')$. For $V' \subseteq V$, $\mathcal{H}[V'] = (V', \{e \cap V' \mid e \in E, e \cap V' \neq \emptyset\})$. For $x, y \in V$, a *path* between x and y is a subset of edges $e_1, ..., e_k \in E$ such that $x \in e_1$, $y \in e_k$, and for all $i \leq k-1$, $e_i \cap e_{i+1} \neq \emptyset$. An independent set I of $\mathcal{H}$ is a subset of V such that for all $x, y \in I$ there is no $e \in E$ such that $x \in E$ and $y \in E$.

A central notion in the following is that of S-component, defined in [11].

Definition 4.23 (S-component). Let $\mathcal{H} = (V, E)$, S be as above. The *S-component* of $e \in E$ is the hypergraph $\mathcal{H}[E']$ where E' is the set of all edges $e' \in E_{\not\subseteq S}$ such that there is a path from $e - S$ to $e' - S$ in $\mathcal{H}[V - S]$.

A subhypergraph $\mathcal{H}'$ of $\mathcal{H}$ is an S-component if there is an edge $e \in E_{\not\subseteq S}$ such that $\mathcal{H}'$ is the S-component of e.

Example 4.24. In Figure 2 an hypergraph $\mathcal{H}$ of an acyclic conjunctive query $\varphi(\mathbf{y})$ is given with $S = \mathit{free}(\varphi) = \{y_1, ..., y_7\}$. It has three S-components are illustrated by Figure 3.

Given $S \subseteq V$, one can decompose $\mathcal{H} = (V, E)$ into disjoint S-components. A measure how S vertices are "spread" into each components through the notion of independence.

Definition 4.25 (S-star size [34]). Let $\mathcal{H} = (V, E)$ be a hypergraph, $S \subseteq V$. and $k \in \mathbb{N}$. The *S-star size* of $\mathcal{H}$ is the maximum size of an independent set of an S-component of $\mathcal{H}$.

Definition 4.26 (quantified-star size [34]). The *quantified star size* of an acyclic conjunctive formula $\varphi(\mathbf{x})$ is the S-star size of the hypergraph $\mathcal{H}$ associated to $\varphi(\mathbf{x})$, where S is the set of free variables in $\phi(\mathbf{x})$.

Note that being of quantified star size 1 is equivalent to being free-connex. A class $\mathscr{C}$ of acyclic conjunctive queries is of bounded quantified star size if there is a $s \in \mathbb{N}$ such that every $\varphi \in \mathscr{C}$ is of quantified-star size bounded by s.

Example 4.27. In Figure 3, the maximal size of an independent set among all S-components is three. For example : $\{y_3, y_5, y_6\}$. Similarly, the quantified star size of formula $\psi(\mathbf{x})$ in Equation 2 is n, the domain size of the database.

To understand why the decomposition into S-components has influence on the complexity, let us make several remarks:

- A query $\varphi(\mathbf{x})$ can be decomposed into $\varphi_1 \wedge \varphi_2 \wedge \cdots \wedge \varphi_m \wedge \psi_0$ where m is the number of S-components of it associated hypergraph and ψ_0 is a subformula containing free variables only.
- Let $i \in \{1, .., n\}$ and suppose subquery φ_i is of maximal quantified-star size $s \in \mathbb{N}$. Then, it is not hard to see there exists $\phi_1, ..., \phi_s$ atomic formulas of φ_i that contains all the free variables $\mathit{free}(\varphi_i)$ (recall s is the size of the maximal independent set in the S-component associated to φ_i).
- Since all free variables of φ_i are packed into at most s atomic formulas, given a database $\mathbf{D}$ and using technics similar to Yannakakis algorithm one can built a new relation $R_i = \varphi_i(\mathbf{D})$ in time $O(\|\mathbf{D}\|^s)$ and replace each φ_i by a new atomic formula ψ_i on free variables only. Let's call $\mathbf{D}'$ the collection of all relations R_i
- Now the query $\psi = \psi_1 \wedge \psi_m \wedge \psi_0$ is acyclic and is such that $\psi(\mathbf{D}') = \varphi(\mathbf{D})$. One conclude using Theorem 4.21.

The arguments above give the basic intuition of the result below. The hardness result can be proved by reduction.

Theorem 4.28 ([34]). *There is an algorithm for the problem* ♯ACQ *that runs in time* $(\|\mathbf{D}\| + \|\varphi\|)^{O(k)}$ *where* k *is the quantified star size of the input query* Φ*. Moreover, if a class of acyclic conjunctive queries does not have a bounded quantifier-star size, then its associated counting problem is* #W[1] *hard. It is therefore not* FPT

The quantified star size of a query φ can be computed in polynomial time. See [25, 35] for further use of this measure for the counting complexity of CQ.

4.5 Allowing negations: from queries to CSP

A natural extension is to allow negations of atoms in conjunctive queries. However such a possibility results in a huge gain of expressive power even in contexts that were previously tractable. The main reason is related to its connexion with the satisfiability problem and the succinctness it permits. Consider, as an example, the Boolean clause:

$$x_1 \vee x_2 \vee x_3 \vee x_4 \vee \neg x_5 \vee \neg x_6$$

Such a clause is satisfied by all possible assignments of $x_1, ..., x_6$ except $(0, 0, 0, 0, 1, 1)$. By allowing negations, we can see such a clause as the query that tests whether $\mathbf{D} = \neg R(x_1, ..., x_6)$ where the domain of $\mathbf{D}$ is $\{0, 1\}$ and $R = \{(0, 0, 0, 0, 1, 1)\}$.

It is then clear that α-acyclic queries can not be tractable. Indeed let $\varphi(\mathbf{x})$ be any conjunctive query. Extending the notion of acyclicity to negative atoms as well, the query $\varphi'(\mathbf{x}) = \varphi(\mathbf{x}) \wedge \neg R(\mathbf{x})$ where R is a new relation symbol is acyclic. Upon a database $\mathbf{D}$, defining $\mathbf{D}'$ as $\mathbf{D}$ extended with the empty relation R, it is clear that: $\varphi(\mathbf{D}) = \varphi'(\mathbf{D}')$.

Tractability fragments have to be searched into a more restricted version of acyclicity for hypergraph: β-acyclicity.

Definition 4.29. A query is β-acyclic if its associated hypergraph $\mathcal{H}$ is α-acyclic and all its subhypergraphs $\mathcal{H}' \subseteq \mathcal{H}$ are also α-acyclic.

Definition 4.30. A negative conjunctive query, NCQ, over a signature σ is a formula of the form

$$\varphi(\mathbf{x}) \equiv \exists \mathbf{y} \bigwedge_i \neg R_i(\mathbf{z}_i)$$

where $\mathbf{z}_i$ is a tuple of variables from $\mathbf{x}$ and $\mathbf{y}$, $R_i \in \sigma$. A NCQ query is β-acyclic if its associated hypergraph is β-acyclic.

We call **Triangle** the hypothesis that the existence of a triangle in a graph of n vertices can not be decided in time $O(n^2 \log n)$. The following result holds.

Theorem 4.31 ([17]). *Assuming* ***Triangle****, an* NCQ *is decidable in quasi-linear time iff it is* β*-acyclic.*

Roughly speaking, an NCQ can be seen as a negative encoding of constraint satisfaction problems (CSP) in the context where the arity of constraints is not fixed (under the simpler form of a SAT problem, each disjunctive clause is represented by a negative atom $\neg R(\mathbf{x})$ for which the associated relation R contains only one element). To prove the quasi-linear time algorithm of Theorem 4.31 two main tools are used:

- the well-known Davis-Putnam resolution procedure that replace two clauses of the form $C_1 \vee x$, $C_2 \vee \neg x$, by their resolvent $C_1 \vee C_2$.
- Hypergraphs that are β-acyclic can be characterized in terms of an an elimination ordering of their vertices (through the notion of nest point [38]). This ordering will be used to drive the Davis-Putnam procedure in the choice of the resolvent variable.

Partial characterizations for the complexity of signed queries, i.e. of queries having both negative and positive atoms are given in [18].

Complexity issues for CSP, in particular their counting versions, based on formula restrictions has deserved a lot of attentions [71, 73]. New measures based on hypergraph decompositions and interesting connexions with domains such as knowledge compilation have been established that have helped understanding where the frontier for tractability is (see [19] for a survey and [20] also for the special case of weighted counting for β-acyclic CSP).

5 PREFIX RESTRICTED QUERIES FOR COUNTING AND ENUMERATION

In this part we study the more unusual setting of FO with second order free variables. More precisely, we consider FO queries of the form $\varphi(\mathbf{x}, \mathbf{X})$ where $\mathbf{x}$ and $\mathbf{X}$ are respectively free first-order and second-order variables. Recalling Fagin's theorem [40], it is clear that a such query language is highly expressive.

One approach to find tractable fragments is to restrict the quantifier prefixes of formulas. For any k, the fragments Σ_k and Π_k of FO are the classes of all formulae in prenex normal form with a quantifier prefix with k alternations starting with an existential or an universal quantifier, respectively. When restricted to relations only as second order variables, we call this fragments $\Sigma_k^{\text{rel}}, \Pi_k^{\text{rel}}$. The well-known examples below shed some light on the expressiveness of queries under these restrictions.

Example 5.1. Let DNF (resp. 3DNF) be the problem of deciding if a propositional formula in disjunctive normal-form (resp. with at most 3 literals per clause) is satisfiable. Let $\sigma_{3\text{DNF}} = (D_0, D_1, D_2, D_3)$. Given a 3DNF-formula φ over variables V, we construct a corresponding σ-structure $\mathcal{A}_\varphi$ with universe V such that for any $x_1, x_2, x_3 \in V$, $D_i(x_1, x_2, x_3)$ holds iff $\bigwedge_{1 \le j \le i} \neg x_j \wedge \bigwedge_{i < j \le 3} x_j$ appears as a disjunct. Now consider the σ-formula $\Phi_0(T)$ below:

$$\begin{aligned}\Phi_0(T) \equiv \exists x \exists y \exists z \quad \big(& \big(D_0(x,y,z) \wedge T(x) \wedge T(y) \wedge T(z)\big) \\ & \vee \big(D_1(x,y,z) \wedge \neg T(x) \wedge T(y) \wedge T(z)\big) \\ & \vee \big(D_2(x,y,z) \wedge \neg T(x) \wedge \neg T(y) \wedge T(z)\big) \\ & \vee \big(D_3(x,y,z) \wedge \neg T(x) \wedge \neg T(y) \wedge \neg T(z)\big)\big)\end{aligned}$$

Formula Φ_1 is in Σ_1^{rel}. Moreover, there is a bijection between the set of relations T such that $\mathcal{A}_\varphi \models \Phi_0(T)$ and the set of satisfying assignments of φ.

To express DNF, one can consider the language with predicate V for variables, D for disjunct (whose union represent the domain) and two predicate $P(d, v)$ (resp. $N(d, v)$) representing the fact that variable v appears positively (resp. negatively) in disjunct D and write the following $\Phi_2 \in \Sigma_2^{\text{rel}}$:

$$\Phi_2 \equiv \exists d \exists v\ D(d) \wedge (P(d,v) \rightarrow T(v)) \wedge (N(d,v) \rightarrow \neg T(v))$$

Example 5.2. The formula $\Psi_0 \in \Sigma_0$ and $\Psi_1 \in \Pi_1^{\text{rel}}$ below express respectively the existence of a 3-clique (on an ordered graph) and of a clique:

$$\Psi_0(\mathbf{v}) \equiv v_1 < v_2 < v_3 \wedge E(v_1,v_2) \wedge E(v_2,v_3) \wedge E(v_3,v_1)$$

$$\Psi_1(T) \equiv \forall v_1 \forall v_2\ T(v_1) \wedge T(v_2) \rightarrow E(v_1,v_2)$$

5.1 Counting

In [72], the prefix restricted approach has been used to characterized counting classes above ♯P. Given a class $\mathscr{L}$ (here a prefix class) we denote by $\sharp\mathscr{L}$, the class of functions

$$\sharp\cdot\varphi : \mathbf{D} \longrightarrow |\varphi(\mathbf{D})| = |\{(\mathbf{a},\mathbf{A}) : \mathbf{D} \models \varphi(\mathbf{a},\mathbf{A})\}|$$

for formulas $\varphi(\mathbf{x},\mathbf{X}) \in \mathscr{L}$. It holds that:

Theorem 5.3 ([72]). *On ordered structures:*

$$\#\Sigma_0^{\text{rel}} = \#\Pi_0^{\text{rel}} \subset \#\Sigma_1^{\text{rel}} \subset \#\Pi_1^{\text{rel}} \subset \#\Sigma_2^{\text{rel}} \subset \#\Pi_2^{\text{rel}} = \#\text{FO}^{\text{rel}} = \sharp\text{P}.$$

Moreover, every function in $\#\Sigma_0^{\text{rel}}$ is computable in polynomial time.

In [5] a quantitative extension of second-order logic, called QSO is defined that allow to use first-order and second-order sum and product as closure operators on top of second-order formulas. The restriction of QSO to first-order and second-order sum, called ΣQSO(FO), provides a hierarchy, $\Sigma\text{QSO}(\Sigma_i^{\text{rel}})$, $\Sigma\text{QSO}(\Pi_i^{\text{rel}})$ for $i \geq 0$ similar (in spirit but different on the lowest classes) to the one of [72]. In [33], an analog of [72] that allows quantifications over functions too is studied. As expected the hierarchy is shorter :

$$\begin{array}{ccccc} \#\Sigma_0 & \subset & \#\text{AC}^0 & \subset & \#\Pi_1 = \#\text{FO} = \sharp\text{P}. \\ & \subset & \#\Sigma_1 & \subset & \end{array} \tag{3}$$

where $\#\text{AC}^0$ is the class of functions that computes the number of accepting proof trees of FO-uniform families of circuit (or similarly that counts the number of skolem functions for existentially quantified variables of a FO formula).

Not surprisingly with counting problems, the classes in the hierarchies become expressive quite fast. For example, already $\#\Sigma_1^{\text{rel}}$ contains ♯P-complete problems (#3DNF in Example 5.1). However, for such functions, it can be worth to relaxe the constraint of exact and certain computations and look for algorithms that provide good approximations with high probability for them. This is done through the concept of randomized approximation that follows.

Definition 5.4. Let $f : \{0,1\}^* \rightarrow \mathbb{N}$ be a counting problem. f is said to have a *fully polynomial-time randomized approximation scheme* (FPRAS), if there is a randomized algorithm M working on inputs (x,ϵ) with $x \in \{0,1\}^*, \epsilon \in \mathbb{Q}$ such that for all such inputs:

- $\text{Pr}\left[|M(x,) - f(x)| > \ \cdot |f(x)|\right] < \frac{1}{4}$, where $M(x,\epsilon)$ is the random variable describing the output of M on input (x,ϵ),
- the running time of M on input (x,ϵ) is bounded by a polynomial in $|x|, \frac{1}{\epsilon}$.

One early and celebrated FPRAS was obtained in [57] for the problem ♯DNF. It served as an inspirational method for proving that $\#\Sigma_1^{\text{rel}}$ (and even some syntactic extension of it) admits a FPRAS. It is also the case for the distinct classes: $\#\Sigma_1$ (from [33]) and $\Sigma\text{QSO}(\Sigma_1^{\text{rel}})$ (from [5]). Finding good approximation algorithms for aggregate functions (and the related problem of random algorithm for uniform generation) has deserved even more attention recently (see for example [4]) although it is not clear yet if classes that admit FPRAS can be characterized.

5.2 Enumeration

One can define similarly enumeration classes related to prefix restricted classes of queries. Given a class $\mathscr{L}$, $\text{enum}\cdot\mathscr{L}$ notes the collection of $\text{enum}\varphi$ problems for $\varphi \in \mathscr{L}$. In [37], a characterization of an hierarchy of enumeration problems is given.

Theorem 5.5 ([37]). *On linearly ordered structures, the following hold:*

$$\textit{enum}\cdot\Sigma_0 \subsetneq \textit{enum}\cdot\Sigma_1 \subsetneq \textit{enum}\cdot\Pi_1 \subsetneq \textit{enum}\cdot\Sigma_2 \subsetneq \textit{enum}\cdot\Pi_2.$$

Moreover:

- $\textit{enum}\cdot\Sigma_0$ *can be enumerated with polynomial time precomputation and constant delay*
- $\textit{enum}\cdot\Sigma_1$ *can be enumerated with polynomial delay*
- $\textit{enum}\cdot\Pi_1$ *can not be enumerated with polynomial delay unless* P = NP.

Since a formula $\varphi(\mathbf{x},\mathbf{X}) \in \Sigma_0$ has second-order free variable the constant delay bound is questionable at first sight. The algortihmic model used is the following: it has an output tape on which the current output is written. During the enumeration phase the algorithm only modifies the content of this tape to transform the previous solution into a new one. In the Σ_0 case, by relating to the well-known problem of Gray code enumeration, one can find an enumeration ordering such that the delta between two consecutive solutions only affects a constant part of the output and thus can be performed in constant time resulting in a procedure with delta-constant delay (see also [4] for additional results on the subject).

6 CONCLUSION

In this paper we have presented a *subjective* panorama on the complexity of query evaluations. We have put the focus on new complexity measures that have been introduced (in particular to deal with enumeration), structural decompositions and parameters that allows to design efficient algorithms and conditional lower bounds.

We have considered various algoritmic task related to query answering such as decision, computation of the answer set, counting, enumeration. Some new framework such as query evaluation under updates (see [3, 15, 16, 54, 55]) for which counting or enumerating make sense, have been ignored in the paper. They have developped so deeply in the last years that they deserve independent surveys.

REFERENCES

[1] Antoine Amarilli, Pierre Bourhis, Louis Jachiet, and Stefan Mengel. 2017. A Circuit-Based Approach to Efficient Enumeration. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland (LIPIcs)*, Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl (Eds.), Vol. 80. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 111:1–111:15. https://doi.org/10.4230/LIPIcs.ICALP.2017.111

[2] Antoine Amarilli, Pierre Bourhis, Stefan Mengel, and Matthias Niewerth. 2019. Constant-Delay Enumeration for Nondeterministic Document Spanners. In *22nd International Conference on Database Theory, ICDT 2019, March 26-28, 2019, Lisbon, Portugal (LIPIcs)*, Pablo Barceló and Marco Calautti (Eds.), Vol. 127. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 22:1–22:19. https://doi.org/10.4230/LIPIcs.ICDT.2019.22

[3] Antoine Amarilli and Benny Kimelfeld. 2019. Uniform Reliability of Self-Join-Free Conjunctive Queries. *CoRR* abs/1908.07093 (2019). arXiv:1908.07093 http://arxiv.org/abs/1908.07093

[4] Marcelo Arenas, Luis Alberto Croquevielle, Rajesh Jayaram, and Cristian Riveros. 2019. Efficient Logspace Classes for Enumeration, Counting, and Uniform Generation. In *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, Dan Suciu, Sebastian Skritek, and Christoph Koch (Eds.). ACM, 59–73. https://doi.org/10.1145/3294052.3319704

[5] Marcelo Arenas, Martin Muñoz, and Cristian Riveros. 2017. Descriptive Complexity for counting complexity classes. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*. IEEE Computer Society, 1–12. https://doi.org/10.1109/LICS.2017.8005150

[6] Stefan Arnborg, Jens Lagergren, and Detlef Seese. 1991. Easy Problems for Tree-Decomposable Graphs. *J. Algorithms* 12, 2 (1991), 308–340.

[7] Sanjeev Arora and Boaz Barak. 2009. *Computational Complexity*. Cambridge University Press.

[8] Guillaume Bagan. 2006. MSO Queries on Tree Decomposable Structures Are Computable with Linear Delay. In *Computer Science Logic (CSL)*. 167–181.

[9] Guillaume Bagan. 2009. *Algorithmes et complexité des problèmes d'énumération pour l'évaluation de requêtes logiques. (Algorithms and complexity of enumeration problems for the evaluation of logical queries)*. Ph.D. Dissertation. University of Caen Normandy, France.

[10] Guillaume Bagan, Arnaud Durand, Emmanuel Filiot, and Olivier Gauwin. 2010. Efficient Enumeration for Conjunctive Queries over X-underbar Structures. In *Computer Science Logic, 24th International Workshop, CSL 2010, 19th Annual Conference of the EACSL, Brno, Czech Republic, August 23-27, 2010. Proceedings (Lecture Notes in Computer Science)*, Anuj Dawar and Helmut Veith (Eds.), Vol. 6247. Springer, 80–94. https://doi.org/10.1007/978-3-642-15205-4_10

[11] Guillaume Bagan, Arnaud Durand, and Etienne Grandjean. 2007. On Acyclic Conjunctive Queries and Constant Delay Enumeration. In *Computer Science Logic (CSL)*. 208–222.

[12] Michael Bauland, Philippe Chapdelaine, Nadia Creignou, Miki Hermann, and Heribert Vollmer. 2004. An Algebraic Approach to the Complexity of Generalized Conjunctive Queries. In *Seventh International Conference on Theory and Applications of Satisfiability Testing, SAT'04, Vancouver, BC, Canada.*

[13] Catriel Beeri, Ronald Fagin, David Maier 0001, and Mihalis Yannakakis. 1983. On the Desirability of Acyclic Database Schemes. *J. ACM* 30, 3 (1983), 479–513.

[14] Christoph Berkholz, Fabian Gerhardt, and Nicole Schweikardt. 2020. Constant delay enumeration for conjunctive queries: a tutorial. *SIGLOG News* 7, 1 (2020), 4–33. https://doi.org/10.1145/3385634.3385636

[15] Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. 2017. Answering Conjunctive Queries under Updates. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2017, Chicago, IL, USA, May 14-19, 2017*, Emanuel Sallinger, Jan Van den Bussche, and Floris Geerts (Eds.). ACM, 303–318. https://doi.org/10.1145/3034786.3034789

[16] Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. 2018. Answering UCQs under Updates and in the Presence of Integrity Constraints. In *21st International Conference on Database Theory, ICDT 2018, March 26-29, 2018, Vienna, Austria (LIPIcs)*, Benny Kimelfeld and Yael Amsterdamer (Eds.), Vol. 98. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 8:1–8:19. https://doi.org/10.4230/LIPIcs.ICDT.2018.8

[17] Johann Brault-Baron. 2012. A Negative Conjunctive Query is Easy if and only if it is Beta-Acyclic. In *Computer Science Logic (CSL)*. 137–151.

[18] Johann Brault-Baron. 2013. *De la pertinence de l'énumération : complexité en logiques propositionnelle et du premier ordre. (The relevance of the list: propositional logic and complexity of the first order)*. Ph.D. Dissertation. University of Caen Normandy, France.

[19] Florent Capelli. 2016. *Structural restrictionsof CNF-formulas: applications to model counting and knowledge compilation*. Ph.D. Dissertation. University of Paris-Diderot, France.

[20] Florent Capelli. 2017. Understanding the complexity of #SAT using knowledge compilation. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*. IEEE Computer Society, 1–10. https://doi.org/10.1109/LICS.2017.8005121

[21] Nofar Carmeli and Markus Kröll. 2018. Enumeration Complexity of Conjunctive Queries with Functional Dependencies. In *21st International Conference on Database Theory, ICDT 2018, March 26-29, 2018, Vienna, Austria (LIPIcs)*, Benny Kimelfeld and Yael Amsterdamer (Eds.), Vol. 98. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 11:1–11:17. https://doi.org/10.4230/LIPIcs.ICDT.2018.11

[22] Nofar Carmeli and Markus Kröll. 2019. On the Enumeration Complexity of Unions of Conjunctive Queries. In *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, Dan Suciu, Sebastian Skritek, and Christoph Koch (Eds.). ACM, 134–148. https://doi.org/10.1145/3294052.3319700

[23] Nofar Carmeli, Shai Zeevi, Christoph Berkholz, Benny Kimelfeld, and Nicole Schweikardt. 2019. Answering (Unions of) Conjunctive Queries using Random Access and Random-Order Enumeration. *CoRR* abs/1912.10704 (2019). arXiv:1912.10704 http://arxiv.org/abs/1912.10704

[24] A K Chandra and P M Merlin. 1977. Optimal Implementation of conjunctive queries in relational databases. In *Proceedings of the 9th Annual ACM Symposium on Theory of Computing*, ACM New York (Ed.). 77–90.

[25] Hubie Chen and Stefan Mengel. 2017. The logic of counting query answers. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*. IEEE Computer Society, 1–12. https://doi.org/10.1109/LICS.2017.8005085

[26] Don Coppersmith and Shmuel Winograd. 1990. Matrix Multiplication via Arithmetic Progressions. *J. Symb. Comput.* 9, 3 (1990), 251–280. https://doi.org/10.1016/S0747-7171(08)80013-2

[27] Bruno Courcelle. 1990. Graph Rewriting: An Algebraic and Logic Approach. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*. Elsevier, 193–242.

[28] Bruno Courcelle. 2009. Linear delay enumeration and monadic second-order logic. *Discrete Applied Mathematics* 157, 12 (2009), 2675–2700.

[29] Bruno Courcelle. 2009. Linear delay enumeration and monadic second-order logic. *Discrete Applied Mathematics* 157, 12 (2009), 2675–2700.

[30] Reinhard Diestel. 2006. *Graph Theory*. Springer Science & Business Media.

[31] Johannes Doleschal, Benny Kimelfeld, Wim Martens, and Liat Peterfreund. 2020. Weight Annotation in Information Extraction. In *23rd International Conference on Database Theory, ICDT 2020, March 30-April 2, 2020, Copenhagen, Denmark (LIPIcs)*, Carsten Lutz and Jean Christoph Jung (Eds.), Vol. 155. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 8:1–8:18. https://doi.org/10.4230/LIPIcs.ICDT.2020.8

[32] Arnaud Durand and Etienne Grandjean. 2007. First-order queries on structures of bounded degree are computable with constant delay. *ACM Trans. Comput. Log.* 8, 4 (2007).

[33] Arnaud Durand, Anselm Haak, Juha Kontinen, and Heribert Vollmer. 2016. Descriptive Complexity of $\#AC^0$ Functions. In *25th EACSL Annual Conference on Computer Science Logic, CSL 2016, August 29 - September 1, 2016, Marseille, France (LIPIcs)*, Jean-Marc Talbot and Laurent Regnier (Eds.), Vol. 62. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 20:1–20:16. https://doi.org/10.4230/LIPIcs.CSL.2016.20

[34] Arnaud Durand and Stefan Mengel. 2014. The complexity of weighted counting for acyclic conjunctive queries. *J. Comput. Syst. Sci.* 80, 1 (2014), 277–296.

[35] Arnaud Durand and Stefan Mengel. 2015. Structural Tractability of Counting of Solutions to Conjunctive Queries. *Theory Comput. Syst.* 57, 4 (2015), 1202–1249. https://doi.org/10.1007/s00224-014-9543-y

[36] Arnaud Durand, Nicole Schweikardt, and Luc Segoufin. 2014. Enumerating answers to first-order queries over databases of low degree. In *Symp. on Principles of Database Systems (PODS)*.

[37] Arnaud Durand and Yann Strozecki. 2011. Enumeration Complexity of Logical Query Problems with Second-order Variables. In *Conf. on Computer Science Logic (CSL)*.

[38] Duris, David. 2012. Some characterizations of γ and β-acyclicity of hypergraphs. *Inform. Process. Lett.* 112, 16 (Aug. 2012).

[39] Zdeněk Dvořák, Daniel Král, and Robin Thomas. 2010. Deciding First-Order Properties for Sparse Graphs. In *Symp. on Foundations Of Computer Science (FOCS)*. 133–142.

[40] R. Fagin. 1974. Generalized first-order spectra and polynomial-time recognizable sets. In *Proceedings Complexity of Computation*, R. M. Karp (Ed.). Vol. 7. SIAM-AMS, 27–41.

[41] R. Fagin. 1983. Degrees of acyclicity for hypergraphs and relational database schemes. 30, 3 (1983), 514–550.

[42] Fernando Florenzano, Cristian Riveros, Martín Ugarte, Stijn Vansummeren, and Domagoj Vrgoc. 2020. Efficient Enumeration Algorithms for Regular Document Spanners. *ACM Trans. Database Syst.* 45, 1 (2020), 3:1–3:42. https://doi.org/10.1145/3351451

[43] Jörg Flum and Martin Grohe. 2006. *Parameterized Complexity Theory*. Springer-Verlag.

[44] Dominik D. Freydenberger, Benny Kimelfeld, and Liat Peterfreund. 2018. Joining Extractions of Regular Expressions. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, Houston, TX, USA, June 10-15, 2018*, Jan Van den Bussche and Marcelo Arenas (Eds.). ACM, 137–149. https://doi.org/10.1145/3196959.3196967

[45] Markus Frick and Martin Grohe. 2001. Deciding first-order properties of locally tree-decomposable structures. *J. ACM* 48, 6 (2001), 1184–1206.

[46] Markus Frick and Martin Grohe. 2004. The complexity of first-order and monadic second-order logic revisited. *Ann. Pure Appl. Logic* 130, 1-3 (2004), 3–31.

[47] Jakub Gajarský, Petr Hlinený, Jan Obdrzálek, Daniel Lokshtanov, and M. S. Ramanujan. 2016. A New Perspective on FO Model Checking of Dense Graph

Classes. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, Martin Grohe, Eric Koskinen, and Natarajan Shankar (Eds.). ACM, 176–184. https://doi.org/10.1145/2933575.2935314

[48] Jakub Gajarský, Stephan Kreutzer, Jaroslav Nesetril, Patrice Ossona de Mendez, Michal Pilipczuk, Sebastian Siebertz, and Szymon Toru'nczyk. 2018. First-Order Interpretations of Bounded Expansion Classes. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic (LIPIcs)*, Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella (Eds.), Vol. 107. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 126:1–126:14. https://doi.org/10.4230/LIPIcs.ICALP.2018.126

[49] Francois Le Gall. 2014. Powers of tensors and fast matrix multiplication. In *Intl. Symp. on Symbolic and Algebraic Computation (ISSAC)*.

[50] Etienne Grandjean and Frédéric Olive. 2004. Graph properties checkable in linear time in the number of vertices. *J. Comput. Syst. Sci.* 68, 3 (2004), 546–597.

[51] Martin Grohe. 2001. Generalized Model-Checking Problems for First-Order Logic. In *Symp. on Theoretical Aspects in Computer Science (STACS)*.

[52] Martin Grohe. 2008. Logic, graphs, and algorithms. In *Logic and Automata*.

[53] Martin Grohe, Stephan Kreutzer, and Sebastian Siebertz. 2017. Deciding First-Order Properties of Nowhere Dense Graphs. *J. ACM* 64, 3 (2017), 17:1–17:32.

[54] Muhammad Idris, Martín Ugarte, and Stijn Vansummeren. 2017. The Dynamic Yannakakis Algorithm: Compact and Efficient Query Processing Under Updates. In *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*, Semih Salihoglu, Wenchao Zhou, Rada Chirkova, Jun Yang, and Dan Suciu (Eds.). ACM, 1259–1274. https://doi.org/10.1145/3035918.3064027

[55] Muhammad Idris, Martín Ugarte, Stijn Vansummeren, Hannes Voigt, and Wolfgang Lehner. 2018. Conjunctive Queries with Inequalities Under Updates. *PVLDB* 11, 7 (2018), 733–745. https://doi.org/10.14778/3192965.3192966

[56] David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. 1988. On Generating All Maximal Independent Sets. *Inf. Process. Lett.* 27, 3 (1988), 119–123. https://doi.org/10.1016/0020-0190(88)90065-8

[57] Richard M. Karp, Michael Luby, and Neal Madras. 1989. Monte-Carlo Approximation Algorithms for Enumeration Problems. *J. Algorithms* 10, 3 (1989), 429–448. https://doi.org/10.1016/0196-6774(89)90038-2

[58] Wojciech Kazana. 2013. *Query evaluation with constant delay. (L'évaluation de requêtes avec un délai constant).* Ph.D. Dissertation. École normale supérieure de Cachan, Paris, France.

[59] Wojciech Kazana and Luc Segoufin. 2011. First-order query evaluation on structures of bounded degree. *Logical Methods in Computer Science (LMCS)* 7, 2 (2011).

[60] Wojciech Kazana and Luc Segoufin. 2013. Enumeration of first-order queries on classes of structures with bounded expansion. In *Principle of Database Systems (PODS)*. 297–308.

[61] Wojciech Kazana and Luc Segoufin. 2013. Enumeration of monadic second-order queries on trees. *ACM Trans. Comput. Log.* 14, 4 (2013). https://doi.org/10.1145/2528928

[62] Stephan Kreutzer. 2009. On the Parameterised Intractability of Monadic Second-Order Logic. In *Conf. on Computer Science Logic (CSL)*.

[63] Stephan Kreutzer and Anuj Dawar. 2009. Parameterized Complexity of First-Order Logic. *Electronic Colloquium on Computational Complexity (ECCC)* 16 (2009), 131. http://eccc.hpi-web.de/report/2009/131

[64] Leonid Libkin. 2004. *Elements of Finite Model Theory.* Springer.

[65] Andrea Lincoln, Virginia Vassilevska Williams, and R. Ryan Williams. 2018. Tight Hardness for Shortest Cycles and Paths in Sparse Graphs. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, Artur Czumaj (Ed.). SIAM, 1236–1252. https://doi.org/10.1137/1.9781611975031.80

[66] Jaroslav Nesetril and Patrice Ossona de Mendez. 2012. *Sparsity - Graphs, Structures, and Algorithms.* Algorithms and combinatorics, Vol. 28. Springer. https://doi.org/10.1007/978-3-642-27875-4

[67] Jaroslav Nešetřil and Patrice Ossona de Mendez. 2010. First order properties on nowhere dense structures. *J. Symb. Log.* 75, 3 (2010), 868–887. https://doi.org/10.2178/jsl/1278682204

[68] Jaroslav Nešetřil and Patrice Ossona de Mendez. 2011. On nowhere dense graphs. *Eur. J. Comb.* 32, 4 (2011), 600–617. https://doi.org/10.1016/j.ejc.2011.01.006

[69] C Papadimitriou and M Yannakakis. 1999. On the complexity of database queries. *J. Comput. System Sci.* 58, 3 (1999), 407–427.

[70] Reinhard Pichler and Sebastian Skritek. 2013. Tractable counting of the answers to conjunctive queries. *J. Comput. Syst. Sci.* 79, 6 (2013), 984–1001.

[71] Sigve Hortemo Sæther, Jan Arne Telle, and Martin Vatshelle. 2015. Solving #SAT and MAXSAT by Dynamic Programming. *J. Artif. Intell. Res.* 54 (2015), 59–82. https://doi.org/10.1613/jair.4831

[72] S. Saluja, K. V. Subrahmanyam, and M. N. Thakur. 1995. Descriptive complexity of #P functions. 50, 3 (1995), 493–505.

[73] Marko Samer and Stefan Szeider. 2010. Algorithms for propositional model counting. *J. Discrete Algorithms* 8, 1 (2010), 50–64. https://www.ac.tuwien.ac.at/files/pub/SamerSzeider10.pdf

[74] Nicole Schweikardt, Luc Segoufin, and Alexandre Vigny. 2018. Enumeration for FO Queries over Nowhere Dense Graphs. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, Houston, TX, USA, June 10-15, 2018*, Jan Van den Bussche and Marcelo Arenas (Eds.). ACM, 151–163. https://doi.org/10.1145/3196959.3196971

[75] Detlef Seese. 1996. Linear Time Computable Problems and First-Order Descriptions. *Mathematical Structures in Computer Science* 6, 6 (1996), 505–526.

[76] Luc Segoufin. 2014. A glimpse on constant delay enumeration (Invited Talk). In *31st International Symposium on Theoretical Aspects of Computer Science (STACS 2014), STACS 2014, March 5-8, 2014, Lyon, France (LIPIcs)*, Ernst W. Mayr and Natacha Portier (Eds.), Vol. 25. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 13–27. https://doi.org/10.4230/LIPIcs.STACS.2014.13

[77] Luc Segoufin and Alexandre Vigny. 2017. Constant Delay Enumeration for FO Queries over Databases with Local Bounded Expansion. In *Intl. Conf. on Database Theory (ICDT)*.

[78] Yann Strozecki. 1988. Enumeration Complexity. *Bulletin of the EATCS, the Algortihmics column* 129 (1988).

[79] Yann Strozecki. 2010. *Enumeration complexity and matroid decomposition.* Ph.D. Dissertation. Université Paris Diderot, Paris, France.

[80] Alexandre Vigny. 2018. *Query enumeration and nowhere dense graphs.* Ph.D. Dissertation. University of Paris-Diderot, France.

[81] Mihalis Yannakakis. 1981. Algorithms for Acyclic Database Schemes. In *Proceeding of VLDB*. 82–94.