Constant delay enumeration for conjunctive queries

Luc Segoufin
INRIA and ENS Cachan

ABSTRACT

We survey some of the recent results about enumerating the answers to queries over a database. We focus on the case where the enumeration is performed with a constant delay between any two consecutive solutions, after a linear time preprocessing.

This cannot be always achieved. It requires restricting either the class of queries or the class of databases.

We consider conjunctive queries and describe several scenarios when this is possible.

1. INTRODUCTION

The evaluation of queries is a central problem in database management systems. Given a query q and a database $\mathcal{D}$ the evaluation of q over $\mathcal{D}$ consists in computing the set $q(\mathcal{D})$ of all answers to q on $\mathcal{D}$. The complexity of this problem has been widely studied. However most of the complexity bounds are extrapolated from the boolean case (aka the model checking problem) and expressed as a function of the sizes of q and $\mathcal{D}$. For non boolean queries it may be not satisfactory enough to express complexity results just in terms of the sizes of $\mathcal{D}$ and q. A simple observation shows that the set $q(\mathcal{D})$ may be huge, even larger than the database itself, as it can have a number of elements of the form $\|\mathcal{D}\|^l$, where $\|D\|$ is the size of the database and l the arity of the query. The fact that the solution set $q(\mathcal{D})$ may be of size exponential in the query is intuitively not sufficient to make the problem hard, and alternative complexity measures had to be found for query answering. For instance one could consider output-sensitive complexity measures expressed as a function of the sizes of q, $\mathcal{D}$ but also $q(\mathcal{D})$. In this direction, one way to define tractability is to assume that tuples of the query result can be generated one by one with some regularity, for example by ensuring a fixed delay between two consecutive outputs once a necessary precomputation has been done to construct a suitable index structure.

This approach, that considers query answering as an enumeration problem, has deserved some attention over the last few years. In this vein, the best that one can hope for is constant delay, i.e., the delay depends only on the size of q (but not on the size of $\mathcal{D}$). A number of query evaluation problems have been shown to admit constant delay algorithms, usually preceded by a preprocessing phase that is linear in the size of the database. We survey some of these results in this paper.

This imposes drastic constraints. In particular, the first answer is output after a time linear in the size of the database and once the enumeration starts a new answer is being output regularly at a speed independent from the size of the database. Altogether, the set $q(\mathcal{D})$ is entirely computed in time $f(q)(\|\mathcal{D}\| + |q(\mathcal{D})|)$ for some function f depending only on q and not on $\mathcal{D}$. In particular boolean queries can be evaluated in time linear in the size of the database. However, as shown in [5], the fact that evaluation of boolean queries is easy does not guarantee the existence of such efficient enumeration algorithms in general: under some reasonable complexity assumption, there is no constant delay algorithm with linear preprocessing enumerating the answers of acyclic conjunctive queries, although it is well-known that the model-checking of boolean acyclic queries can be done in linear time [29].

We stress that our study is theoretical. If most of the algorithms we will mention here are linear in the size of the database, the multiplicative factors are often very big, making any implementation difficult. However, we believe that the index structures designed for making these algorithms work are interesting and, with extra assumptions, could possibly be turned into something practical.

In this paper we concentrate on conjunctive queries, possibly with negated atoms. We will see how various forms of acyclicity play here a crucial role. Modulo reasonable complexity assumptions, we are actually able to characterize precisely those acyclic conjunctive queries that can be enumerated with constant delay.

There are many related problems. Typically one could imagine computing the top-ℓ most relevant answers relative to some ranking function or to provide a sampling of $q(\mathcal{D})$ relative to some distribution. One could also imagine computing only the number of solutions $|q(\mathcal{D})|$ or providing an efficient test for whether a given tuple belongs to $q(\mathcal{D})$ or not. It is not clear a priori how these problems are related to constant delay enumeration. However, it turns out that in the scenarios where constant delay enumeration can be achieved, one can often also count the number of solutions in time linear in the size of the database and, after linear time preprocessing on the database, one can test in constant time whether a given tuple is part of the answers set.

This survey is by no means exhaustive. It is only intended to survey the major theoretical results concerning conjunctive queries and enumeration. Hopefully it will convince the reader that this is an important subject for research that still contains many interesting and challenging open problems.

Enumeration in general, and constant delay enumeration in particular, is a well identified subfield of algorithmics, and many non trivial enumeration algorithms exist for problems over graphs (like enumerating all spanning trees, all connected components, all cycles etc...) We will not discuss those results at all here.

2. PRELIMINARIES

2.1 Database and queries

In this paper a database is a finite relational structure. A *relational signature* is a tuple

$$\sigma = (R_1, \cdots, R_l),$$

each R_i being a relational symbol of arity r_i. A *relational structure* over σ is a tuple

$$\mathcal{D} = \left(D, R_1^{\mathcal{D}}, \ldots, R_l^{\mathcal{D}}\right),$$

where D is the *domain* of $\mathcal{D}$ and $R_i^{\mathcal{D}}$ is a subset of D^{r_i}. We define the *size* of $\mathcal{D}$ as

$$\|\mathcal{D}\| = |\sigma| + |D| + \sum_{R_i} |R_i^{\mathcal{D}}| r_i.$$

It corresponds to the size of a reasonable encoding of $\mathcal{D}$. The number of elements in the domain of $\mathcal{D}$ is denoted by $|\mathcal{D}|$.

A *query* takes as input a database of a given signature σ and returns a relation of a fixed arity, *the arity of the query*. A query is *boolean* if its arity is 0. The query is then either true or false on $\mathcal{D}$ and defines a property of $\mathcal{D}$. A query is *unary* if its arity is 1. If q is a query and $\bar{a}$ is in the image of q on $\mathcal{D}$, then we write $\mathcal{D} \models q(\bar{a})$. Finally we set

$$q(\mathcal{D}) = \{\bar{a} \mid \mathcal{D} \models q(\bar{a})\}.$$

Note that the size of $q(\mathcal{D})$ may be exponential in the arity of q. A query language is a class of queries. Typically it is defined as a logical formalism such as CQ (for *conjunctive* queries), FO (for *first-order* queries), MSO (for *monadic second-order* queries) and so on. As usual, $|q|$ denotes the size of q.

2.2 Model of computation

We use Random Access Machines (RAM) with addition and uniform cost measure as a model of computation, cf. [1]. Our algorithms will take as input a query q of size k and a database $\mathcal{D}$ of size n. We then say that an algorithm runs in *linear time* if it ends within $f(k)n$ steps, for some function f. It runs in *quasi-linear* time if it ends within $f(k)n \log n$ steps. It runs in *constant time* if it ends in $f(k)$ steps.

Given an $n \times n$ matrix, and two numbers $i, j \leq n$ the RAM model returns the content to the entry (i, j) of the matrix in constant time. Therefore when given the adjacency matrix of a graph it can test in constant time where two given nodes are adjacent or not. However our databases are encoded by the list of their tuples and we therefore do not have access to the adjacency matrix. Testing whether a tuple belongs to a relation may therefore require more than a constant time.

In the sequel we assume that the input database comes with a linear order on the domain. If not, we use the one induced by the encoding of the database as an input to the RAM. Whenever we iterate through all nodes of the domain, the iteration is with respect to the initial linear order.

An important observation is that the RAM model can sort m elements of size $O(\log m)$ in time $O(m \log m)$ [18]. In particular, we can sort lexicographically the tuples of a relation in linear time. As a consequence, a simple merge-sort algorithm we can compute the relation $\{\bar{x}\bar{y} \mid R(\bar{x}\bar{y}) \wedge S(\bar{x})\}$ in time linear in the sizes of R and S.

2.3 Parametrized complexity

The database $\mathcal{D}$ and the query q play different roles as input of our problems. It is often assumed that $|\mathcal{D}|$ is large while $|q|$ is small. Hence it is useful to distinguish them in the input of the query answering problem. Parametrized complexity is a suitable framework for analyzing such situations.

We only provide here the basics of parametrized complexity needed for understanding this paper. The interested reader is referred to the monograph [17].

We view the problem of boolean query evaluation as a parametrized problem where the input is a database $\mathcal{D}$ and a boolean query q, the parameter is $|q|$ and the problem asks whether $\mathcal{D} \models q$.

A parametrized problem is Fixed Parameter Tractable, i.e. can be solved in FPT, if it can be solved in time $f(q)\|\mathcal{D}\|^c$ for some suitable computable function f and constant c. The idea behind this definition is that it is often preferable to have an algorithm working in $2^{|q|}\|\mathcal{D}\|^2$ rather than $\|\mathcal{D}\|^{|q|}$.

In parametrized complexity there is also a suitable notion of reduction, called FPT-reduction.
FPT is closed under FPT-reductions and there are some hard classes of parametrized problems, closed under FPT-reductions, containing problems with no known FPT algorithms and that are believed to be different from FPT.

We distinguish two important hard classes denoted W[1] and AW[$*$]. W[1] plays in parametrized complexity the role of NP in classical complexity. A typical problem which is complete for W[1] is the parametrized boolean query evaluation problem for CQ [24]. AW[$*$] plays in parametrized complexity the role of PSpace in classical complexity. A typical problem which is complete for AW[$*$] is the parametrized boolean query evaluation problem for FO [24].

2.4 The enumeration class CD∘Lin

Let $\mathcal{L}$ be a query language and $\mathcal{C}$ be a class of databases. We say that the *enumeration problem* for $\mathcal{L}$ over $\mathcal{C}$ can be solved with *constant delay after linear preprocessing* (is in CD∘Lin), if it can be solved by a RAM algorithm which, on input $q \in \mathcal{L}$ and $\mathcal{D} \in \mathcal{C}$, can be decomposed into two phases:

- a preprocessing phase that is performed in time linear in the size of the database, and
- an enumeration phase that outputs $q(\mathcal{D})$ with no repetition and a delay depending only on q between any two consecutive outputs. The enumeration phase has full read access to the output of the preprocessing phase and can use extra memory whose size depends only on q.

The definition of CD∘Lin requires a preprocessing time linear in $\|\mathcal{D}\|$ and a delay not depending on $\mathcal{D}$. There are hidden multiplicative factors that are functions on the size of the query. These factors may be huge. We will refer to them in the sequel as the *multiplicative factors.*

Before we proceed with the technical presentation of the results, it is worth spending some time with examples.

Example 1. *Consider a database schema containing a binary relational symbol R and the query*

$$q(x, y) := \neg R(x, y).$$

On input database $\mathcal{D}$, the following simple algorithm enumerates $q(\mathcal{D})$:

```
Go through all pairs $(a, b)$;
test if it is a fact of $R^{\mathcal{D}}$;
if so skip this pair;
otherwise output it.
```

However, a simple complexity analysis shows that the delay between any two outputs is not constant. There are two reasons for this. First, arbitrarily long sequences of pairs can be skipped. Second, it is not obvious how to test whether $(a, b) \in R^{\mathcal{D}}$ in constant time (i.e. without going through the whole relation $R^{\mathcal{D}}$). In order to enumerate this query with constant delay it is necessary to perform a preprocessing computing an index structure that can be used for enumeration. This is done as follows.

We first decide on an arbitrary linear order on the domain of $\mathcal{D}$. We then order all $R^{\mathcal{D}}$ according to the lexicographical order. Recall that with the RAM model this can be done in time linear in $\|\mathcal{D}\|$.

We then compute for each tuple $\bar{u}$ of $R^{\mathcal{D}}$ the tuples $\bar{v} = f(\bar{u})$ and $\bar{v}' = g(\bar{u})$ such that $\bar{v}$ is the smallest (relative to the lexicographical order) element not in $R^{\mathcal{D}}$ that is bigger than $\bar{u}$ (hence all tuples between $\bar{u}$ and $\bar{v}$ are in $R^{\mathcal{D}}$) and $\bar{v}'$ is the smallest (relative to the lexicographical order) element in $\in R^{\mathcal{D}}$ that is bigger than $\bar{v}$. These functions can be computed in time linear in $\|\mathcal{D}\|$ by a simple pass on the ordered list of $R^{\mathcal{D}}$ from its last element to the first one.

This concludes the preprocessing phase, the index consists in those precomputed functions. Note that the RAM model is such that once a function h is computed, on input $\bar{u}$, $h(\bar{u})$ is returned in constant time.

Using the precomputed functions, the enumeration phase is now simple. We maintain two pairs of elements of $\mathcal{D}$: one is initialized with the smallest pair according to the lexicographical order, the other one with the smallest pair in $R^{\mathcal{D}}$. The second pair will always be pointing to an element of $R^{\mathcal{D}}$. Assuming the current pairs are $\langle \bar{u}, \bar{v} \rangle$, we then do the following until $\bar{u}$ is maximal.

If $\bar{u} = \bar{v}$ then we move to $\langle f(\bar{v}), g(\bar{v}) \rangle$. Note that f and g are such that for all $\bar{v}$, $f(\bar{v}) \neq g(\bar{v})$.

If $\bar{u} \neq \bar{v}$ we output $\bar{u}$ and replace it by its successor in the lexicographical order without changing $\bar{v}$.

This algorithm is constant delay as an output is performed at least every other step and each step can be performed in constant time as all the relevant functions have been precomputed. All output tuples are clearly not in $R^{\mathcal{D}}$ and the reader can check that all skipped tuples are in $R^{\mathcal{D}}$.

Example 2. *Same schema but the query is now computing the pairs of nodes at distance 2:*

$$q(x,y) := \exists z R(x,z) \wedge R(z,y).$$

We will see in Section 3 that it is likely that this query cannot be enumerated with constant delay. However, if we assume that R has degree bounded by d, then for any node u of the graph, at most d^2 nodes v are at distance 2 from u. Moreover, it is easy to see that we can compute in time linear in $\|\mathcal{D}\|$ the function $f(u)$ associating to u the list of its nodes at distance 1. An extra linear pass based on the function f computes the function $g(u)$ associating to u the list of its nodes at distance 2. From there the enumeration algorithm with constant delay is trivial.

Remark 1. Notice that if the enumeration problem for $\mathcal{L}$ over $\mathcal{C}$ is in CD∘Lin, then all answers can be output in time $O(\|\mathcal{D}\| + |q(\mathcal{D})|)$ and the first output is computed in time linear in $\|\mathcal{D}\|$. In particular the evaluation problem for boolean queries of $\mathcal{L}$ is in FPT. Hence unless FPT = W[1] any language $\mathcal{L}$ whose evaluation problem for boolean queries is hard for W[1] cannot be enumerated in CD∘Lin. In particular this holds for CQ and FO.

Notice that if the arity of q is less or equal to 1, then $|q(\mathcal{D})| \leq |\mathcal{D}| \leq \|\mathcal{D}\|$. It is then plausible that the whole set of answers can be computed in time linear in $\|\mathcal{D}\|$. If this is the case then we have a simple constant delay algorithm that precomputes all answers during the precomputation phase and then scans the set of answers and outputs them one by one during the enumeration phase. Hence enumeration often becomes relevant when the arity of q is at least 2. In this case $q(\mathcal{D})$ can be quadratic in $\|\mathcal{D}\|$ and hence can certainly not be computed within the linear time constraint of the precomputation phase. The index structure built during the preprocessing phase is then a non trivial object. One can also view this index structure as a compact (of linear size) representation of the set $q(\mathcal{D})$ (that can be of polynomial size) and the enumeration algorithm as an output streaming decompression algorithm.

3. CONJUNCTIVE QUERIES AND ENUMERATION

In this section we consider the evaluation of conjunctive queries with possibly negated atoms. We start with the case with no negated atoms.

3.1 Conjunctive queries

Recall that a conjunctive query (CQ) is a query of the form

$$q(\bar{x}) := \exists y_1 \cdots \exists y_l \quad \bigwedge_i R_i(\bar{z}_i)$$

where $R_i(\bar{z}_i)$ is an *positive atom* of q, R_i being a relational symbol and $\bar{z}_i$ containing variables from $\bar{x}$ or $\bar{y}$.

A typical example is the distance 2 query of Example 2. Another example is the query returning all triangles in a graph.

As evaluating boolean conjunctive queries is hard for W[1], we restrict our attention to *acyclic* conjunctive queries that can be evaluated in time $|q| \cdot \|\mathcal{D}\| \cdot |q(\mathcal{D})|$ [29]. We will see that it is very unlikely that constant delay enumeration can be done even for acyclic conjunctive queries. It is only achieved for a subset of them called *free-connex*. We start with the necessary definitions.

To a conjunctive query q, we associate an hypergraph $H(q) = (V,S)$ whose vertices V are the variables of q and whose hyperedges S are the set of variables occurring in a single atom of q, i.e.
$S = \{\{x_1, \cdots, x_p\} \mid R(x_1, \cdots, x_p)$ is an atom of $q\}$.

A *join tree* of $q \in$ CQ is a tree T whose nodes are atoms of q and such that

(i) each atom of q is the label of exactly one node of T,

(ii) for each variable x of q, the set of nodes of T in which x occurs is connected.

A conjunctive query q is said to be *acyclic* if it has a join tree. In graph theoretical terms this is equivalent to saying that the hypergraph $H(q)$ is *α-acyclic.*

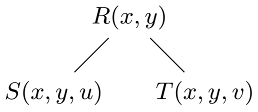
A boolean query associated to a join tree can be evaluated in time linear in $\|\mathcal{D}\|$ using a simple bottom-up traversal of the join tree. If the query is non boolean, the possible valuations of the free variables need to be stored at each step, hence a multiplicative extra factor of $|q(\mathcal{D})|$. The result of [29] follows.

An acyclic conjunctive query $q(\bar{x})$ is said to be *free-connex* if the query $q(\bar{x}) \wedge R(\bar{x})$ is also acyclic, where $\bar{x}$ are the free variables of q and R is a new symbol of appropriate arity[1]. Note that all boolean acyclic queries are free-connex.

For example the acyclic conjunctive query

$$q(x, y) := \exists u, v \;\; S(x, y, u) \wedge T(x, y, v)$$

is free-connex because the following join tree shows acyclicity of the extended query:



However the distance 2 query

$$q(x, y) := \exists z \;\; S(x, z) \wedge S(z, y)$$

is acyclic but not free-connex as the query

$$\exists z \;\; S(x, z) \wedge S(z, y) \wedge R(x, y)$$

is clearly cyclic.

Free-connexity implies the existence of a join tree where all the free variables occur at the root. Hence the bottom-up traversal of the join tree can be performed without having to remember the valuations of the free variables until the last steps. The multiplicative factor of $|q(\mathcal{D})|$ then become an additive factor. With little extra effort this can be turned into a constant delay enumeration algorithm.

Theorem 1. *[5] The enumeration for free-connex acyclic conjunctive queries is in* CD∘Lin.

We stress that the multiplicative factors involved in Theorem 1 are polynomial in the query size.

The result of Theorem 1 also holds if the queries contain inequalities. In this case atoms with inequalities are not involved when building the (generalized) join trees. In the presence of inequalities, the multiplicative factors are now exponential in the query size.

It turns out that free-connexity characterizes exactly those acyclic queries that can be enumerated in constant delay, assuming boolean matrix multiplication cannot be done in quadratic time. Boolean matrix multiplication is the problem of given two $n \times n$ matrices with boolean entries M, N to compute their product MN. The best known algorithms so far (based on the Coppersmith–Winograd algorithm [11]) require more than $n^{2.37}$ steps.

Theorem 2. *[5] If boolean matrix multiplication cannot be done in quadratic time then the following are equivalent for acyclic conjunctive queries q:*

1. *q is free-connex*
2. *q can be enumerated in* CD∘Lin
3. *q can be evaluated in time $O(\|\mathcal{D}\| + |q(\mathcal{D})|)$.*

In particular the distance 2 query cannot be enumerated with constant delay after linear time preprocessing unless boolean matrix multiplication can be done in quadratic time.

With a stronger hypothesis we can even show that acyclicity itself is necessary for having constant delay enumeration. This hypothesis requires that it is not possible to test the existence of a triangle in a hypergraph of n vertices in time $O(n^2)$ and that for any k testing the presence of a k-dimensional tetrahedron cannot be tested in linear time (see [9] for precise definitions).

Theorem 3. *[9] If the above hypothesis is true then the following are equivalent for $q \in$ CQ:*

1. *q is acyclic free-connex*
2. *q can be enumerated in* CD∘Lin

3.2 Signed conjunctive queries

We are now interested in evaluating signed conjunctive queries. Those extend the syntax of conjunctive queries by allowing negated atoms. In other words they are of the form

$$q(\bar{x}) := \exists \bar{y} \quad q^+(\bar{x}\bar{y}) \wedge q^-(\bar{x}\bar{y})$$

where q^+ is a conjunction of positive atoms while q^- is a conjunction of negated atoms.

When q^- is empty we have seen in the previous section that q can be enumerated with constant delay after a linear preprocessing as soon as $H(q^+)$ is α-acyclic and q^+ free-connex. When q^+ is empty it has been shown in [8, 9] that constant delay enumeration can be achieved if $H(q^-)$ is *β-acyclic* and q^- free-connex. β-acyclicity is the hereditary extension of α-acyclicity. It requires that the hypergraph and all its subhypergraphs are α-acyclic. When neither q^+ nor q^- are empty then a notion of *signed-acyclicity* was introduced in [9]. It yields α-acyclicity and β-acyclicity in the corresponding limit cases. It also allows for tractable enumeration algorithms.

Theorem 4. *[9] The enumeration for free-connex signed-acyclic conjunctive queries can be done with constant delay after a preprocessing time of the form $\|\mathcal{D}\|(\log \|\mathcal{D}\|)^{|q|}$.*

The enumeration for free-connex signed-acyclic conjunctive queries can be done with logarithmic delay after a quasi-linear time preprocessing.

[1] This is not the initial definition of free-connex as given in [5]. This presentation is from Brault-Baron [9]

The multiplicative factors are exponential in the size of the query for the constant delay result but polynomial in the logarithmic delay result. As in the previous section, modulo complexity hypothesis, typically that testing the existence of a triangle cannot be done in $O(n^2 \log n)$ time on a graph of size n, the signed-acyclicity hypothesis and the free-connexity hypothesis cannot be avoided [9].

3.3 Longer delay

We could consider enumeration algorithms allowing for non constant delay.

Delay linear in the size of the database. In this setting, the preprocessing phase remains linear in the size of the database but the delay between any two consecutive outputs is now linear in the size of the database. Notice that linear delay still implies that the associated model checking problem is in FPT, hence CQ cannot be enumerated with linear delay unless W[1] =FPT.

One can then consider restricting the class of databases. A class of databases, called $\underline{\mathrm{X}}$-databases, has been exhibited such that CQ can be enumerated over it with linear delay. We will not define $\underline{\mathrm{X}}$-databases in this note. Typical examples are grids and trees with all XPath axis.

Theorem 5. *[4]. The enumeration for* CQ *over* $\underline{X}$*-structures can be done with linear delay.*

For acyclic conjunctive queries linear delay enumeration can be obtained with no restriction on the databases.

Theorem 6. *[5]. The enumeration for acyclic* CQ *over all databases can be done with linear delay.*

Polynomial delay. One could also allow polynomial precomputation and polynomial delay. This notion is maybe less relevant in the database context. Indeed, the degree of the polynomial could depend on the size of the query and in this case the preprocessing phase can often precompute all solutions. This notion is however relevant when considering first-order queries with free second-order variables. In this case, for Σ_1-queries, polynomial delay enumeration can be achieved [16].

4. NEARBY PROBLEMS

It turns out that the index structures build for enumeration can be used with little modifications for solving several related problems, like counting the number of solutions, or in the presence of an order, directly pointing to the j^{th}-solution. We briefly survey those results here.

Counting the number of solutions.

Given a query q and a database $\mathcal{D}$, the counting problem is to compute $|q(\mathcal{D})|$.

Given a query language $\mathcal{L}$, we say that the counting problem of $\mathcal{L}$ is solvable in time $f(n)$ if for any $q \in \mathcal{L}$ and any database $\mathcal{D}$, $|q(\mathcal{D})|$ can be computed in time $g(q)f(\|\mathcal{D}\|)$ for some computable function g. Note that f does not depend on q. If f is polynomial then the associated parametrized computational problem is in the class FPT.

For conjunctive queries, actually even for acyclic conjunctive queries, counting the number of solutions of a query is a hard problem. Already for acyclic conjunctive queries the combined complexity is $\#P$-complete [25] and only the quantifier free acyclic conjunctive queries can be solved in time linear in $\|\mathcal{D}\|$ [3]. Adding just one quantifier already make already the problem hard [26].

For this reason, [14] introduced a new parameter named *quantified-star size*. It measures "how the free variables are spread in the formula" and bounding this parameter yields tractable counting problem for acyclic conjunctive queries.

Theorem 7. *[14] For each number s, the counting problem for acyclic conjunctive queries of quantified-star size bounded by s can be solved in time polynomial in both the size of the query and of the database.*

It turns out that this parameter characterizes exactly the class of acyclic conjunctive queries having a tractable counting problem. If a class of acyclic conjunctive query does not have a bounded quantified-star size, then its associated counting problem is #W[1]-hard [14]. In particular, it cannot be solved in FPT.

It is possible to perform counting efficiently beyond acyclic conjunctive queries. For instance it is known that the boolean case is tractable for CQ having *bounded width* for various notions of width. In order to capture also non boolean queries the notion of *quantified-star size* was extended for various notions of width [13]. Based on this definition for hypertreewidth, the result reads as follows:

Theorem 8. *[13] Let $\mathcal{C}$ be a class of* CQ *of bounded generalized hypertreewidth.*

Assuming W[1] $\neq$ FPT *the following are equivalent:*

1. *The counting problem for queries in $\mathcal{C}$ is solvable in polynomial time*

2. *$\mathcal{C}$ has bounded quantified-star size*

If the schema is not part of the input, or more generally if we assume a bound on the arity of the predicates used in queries of $\mathcal{C}$, they building on [19] we get the following stronger result that also shows that the bounded treewidth hypothesis is necessary:

THEOREM 9. *[13] Let $\mathcal{C}$ be a class of* CQ *using predicates of bounded arity. Assuming* $\mathrm{W}[1] \neq \mathrm{FPT}$ *the following are equivalent:*

1. *The counting problem for queries in $\mathcal{C}$ is solvable in polynomial time*
2. *$\mathcal{C}$ has bounded treewidth and bounded quantified-star size*

5. DISCUSSION

5.1 More expressive queries

In order to be able to enumerate more expressive queries, one need restrictions on the class of databases under consideration. Several restrictions have been investigated, like bounded degree [20, 12], bounded expansion [21] and low degree [15] for FO queries, bounded tree-width [2, 22] for MSO queries and XPath queries over data trees [7]. The interested reader is referred to [27, 28] for a more detailed overview of these results.

5.2 The impact of order

The definition of CD∘LIN presented here does not specify the order in which the answers are output. One could require a specific order, relevant to the context in which the query is evaluated. For instance, if there is a linear order on the domain of the database, one could require that the tuples of the result are output in lexicographical order. Another typical example is when there is a relevance measure associated to each tuple and one would like the answers to the query to be output in the order of their relevance.

Requiring a specific order when outputting the answers to a query may have a dramatic impact on the existence of constant delay algorithms. This is not surprising as the index built during the preprocessing phase is designed for a particular order.

6. CONCLUSIONS

We mentioned several results concerning constant delay enumeration of conjunctive queries. We hope that we succeeded to convince the reader that this is a very interesting topic. We conclude with several research directions.

One could for instance consider relaxing the "no duplicate" constraint during the enumeration phase and enumerate conjunctive queries with the "bag semantics", i.e. each answer occurs as many times as there are valuations witnessing it. This has not been investigated and is clearly relevant for database queries with aggregate predicates.

The situation of the lower bounds mentioned in this paper is not completely satisfactory as they are based on complexity or algorithmics hypothesis. Of course one can construct artificial problems, based on the fact that there exist quadratic but not linear problems, that do not admit constant delay enumeration algorithms. For the queries mentioned in this note, like the distance 2 one, the lower bounds requires an assumption. It is plausible (i.e. there are no known drastic consequences in complexity theory nor in algorithmic) that the non existence of constant delay enumeration algorithms could be proved with no assumptions. We believe this is an interesting and challenging question.

7. REFERENCES

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[2] G. Bagan. MSO Queries on Tree Decomposable Structures Are Computable with Linear Delay. In *Conf. on Computer Science Logic (CSL)*, pages 167–181, 2006.

[3] G. Bagan. *Algorithmes et complexité des problèmes d'énumération pour l'évaluation de requêtes logiques*. PhD thesis, Université de Caen, 2009.

[4] G. Bagan, A. Durand, E. Filiot, and O. Gauwin. Efficient Enumeration for Conjunctive Queries over X-underbar Structures. In *Conf. on Computer Science Logic (CSL)*, pages 80–94, 2010.

[5] G. Bagan, A. Durand, and E. Grandjean. On Acyclic Conjunctive Queries and Constant Delay Enumeration. In *Conf. on Computer Science Logic (CSL)*, pages 208–222, 2007.

[6] G. Bagan, A. Durand, E. Grandjean, and F. Olive. Computing the jth solution of a first-order query. *RAIRO Theoretical Informatics and Applications*, 42(1):147–164, 2008.

[7] M. Bojańczyk and P. Parys. XPath evaluation in linear time. *J. of the ACM*, 58(4), 2011.

[8] J. Brault-Baron. A Negative Conjunctive Query is Easy if and only if it is Beta-Acyclic. In *Conf. on Computer Science Logic (CSL)*, pages 137–151, 2012.

[9] J. Brault-Baron. *De la pertinence de l'énumération : complexité en logiques*

propositionnelle et du premier ordre. PhD thesis, Université de Caen, 2013.

[10] T. Colcombet. A Combinatorial Theorem for Trees. In *Intl. Coll. on Automata, Languages and Programming (ICALP)*, pages 901–912, 2007.

[11] D. Coppersmith and S. Winograd. Matrix Multiplication via Arithmetic Progressions. *J. on Symbolic Computation*, 9(3):251–280, 1990.

[12] A. Durand and E. Grandjean. First-order queries on structures of bounded degree are computable with constant delay. *ACM Trans. on Computational Logic (ToCL)*, 8(4), 2007.

[13] A. Durand and S. Mengel. Structural tractability of counting of solutions to conjunctive queries. In *Intl. Conf. on Database Theory*, pages 81–92, 2013.

[14] A. Durand and S. Mengel. On Polynomials Defined by Acyclic Conjunctive Queries and Weighted Counting Problems. *J. on Computer and System Sciences (JCSS)*, 80(1):277–296, 2014.

[15] A. Durand, N. Schweikardt, and L. Segoufin. Enumerating first-order queries over databases of low degree. In *Symp. on Principles of Database Systems (PODS)*, 2014.

[16] A. Durand and Y. Strozecki. Enumeration Complexity of Logical Query Problems with Second-order Variables. In *Conf. on Computer Science Logic (CSL)*, pages 189–202, 2011.

[17] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2006.

[18] E. Grandjean. Sorting, Linear Time and the Satisfiability Problem. *Annals of Mathematics and Artificial Intelligence*, 16:183–236, 1996.

[19] M. Grohe, T. Schwentick, and L. Segoufin. When is the evaluation of conjunctive queries tractable? In *Symposium on Theory of Computing (STOC)*, pages 657–666, 2001.

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

[21] W. Kazana and L. Segoufin. Enumeration of first-order queries on classes of structures with bounded expansion. *Symp. on Principles of Database Systems (PODS)*, 2013.

[22] W. Kazana and L. Segoufin. Enumeration of monadic second-order queries on trees. *ACM Transactions on Computational Logic*, 14(4), 2013.

[23] S. Lindell. A Normal Form for First-Order Logic over Doubly-Linked Data Structures. *Int. J. Found. Comput. Sci.*, 19(1):205–217, 2008.

[24] C. H. Papadimitriou and M. Yannakakis. On the Complexity of Database Queries. *J. on Computer and System Sciences (JCSS)*, 58(3):407–427, 1999.

[25] R. Pichler and S. Skritek. Tractable Counting of the Answers to Conjunctive Queries. In *Alberto Mendelzon Intl. Workshop on Foundations of Data Management (AMW)*, 2011.

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

[27] L. Segoufin. Enumerating with constant delay the answers to a query. In *Intl. Conf. on Database Theory*, pages 10–20, 2013.

[28] L. Segoufin. A glimpse on constant delay enumeration. In *Intl. Symp. on Theoretical Aspects of Computer Science (STACS)*, pages 13–27, 2014.

[29] M. Yannakakis. Algorithms for Acyclic Database Schemes. In *Intl. Conf. on Very Large Databases (VLDB)*, pages 82–94, 1981.