Enumerating with constant delay the answers to a query

Luc Segoufin
INRIA and ENS Cachan
http://pages.saclay.inria.fr/luc.segoufin/

ABSTRACT

We survey recent results about enumerating with constant delay the answers to a query over a database. More precisely, we focus on the case when enumeration can be achieved with a preprocessing running in time linear in the size of the database, followed by an enumeration process outputting the answers one by one with constant time between any consecutive outputs. We survey classes of databases and classes of queries for which this is possible. We also mention related problems such as computing the number of answers or sampling the set of answers.

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 $\mathcal{D}$. 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 n^l, where n is the size of the database and l the arity of the query. It can therefore require too many of the available resources to compute it entirely.

There are many solutions to overcome this problem. For instance one could imagine that a small subset of $q(\mathcal{D})$ can be quickly computed and that this subset will be enough for the user needs. 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.

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

In this paper we will focus on another scenario consisting in enumerating $q(\mathcal{D})$ with constant delay. In a nutshell, this means that there is a two-phase algorithm working as follows: a preprocessing phase that works in time linear in the size of the database, followed by an enumeration phase outputting one by one all the elements of $q(\mathcal{D})$ with a constant delay between any two consecutive outputs.

The existence of a constant delay enumeration algorithm 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 $f(q)(n+|q(\mathcal{D})|)$ for some function f depending only on q and not on $\mathcal{D}$.

Without any assumptions on the database or on the query, most queries cannot be enumerated with constant delay. For instance we will see that the simple acyclic conjunctive query computing the pairs of nodes at distance 2 in a graph cannot be enumerated with constant delay unless some drastic consequence on the asymptotic complexity of the Boolean Matrix Multiplication problem.

The first and main part of the paper is devoted to the study of constant delay algorithms. It is organized with one section per query language. In a second part of the paper we will discuss variations and related problems.

We will start with conjunctive queries, actually acyclic conjunctive queries, in Section 3. We will see that we can characterize precisely those acyclic conjunctive queries that can be enumerated with constant delay.

We will then move on to first-order queries in Section 4. In this case we need to restrict the class of databases. We will see that constant delay algorithms can be obtained over databases with bounded degree, bounded treewidth or, more generally, "bounded expansion".

In Section 6 we will see that, in the bounded treewidth case, one can even enumerate monadic second-order queries with constant delay.

The last case permitting constant delay enumeration that we will mention in Section 5 is XPath over XML documents.

There are many variations of constant delay enumeration that have been considered. For instance one could require that the answers are enumerated relative to a specific order. It turns out that the existence of constant delay enumeration algorithm is very sensitive to the order of outputs. We will also discuss linear or even polynomial delays in Section 7.

We will also discuss the problem of counting the number of solutions and the problem of testing whether a given tuple belongs to the answers set or not. It is not clear a priori how these two 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. We will survey those results in Section 7.

Finally Section 7 concludes with the very few known results about sampling the answers set.

This survey is by no means exhaustive. It is only intended to survey the major theoretical results concerning database querying 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.

Acknowledgment: We are grateful to Cristina Sirangelo, Arnaud Durand, Wojtek Kazana and Yann Strozecki for their valuable comments on earlier version of this paper.

2. PRELIMINARIES

2.1 Database as relational structures, queries

In this paper a database is a finite relational structure.

A *relational signature* is a tuple $\sigma = (R_1, \ldots, R_l)$, 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}. The number of elements in the domain of $\mathcal{D}$ is denoted by $|\mathcal{D}|$.

A *query* is a computable function associating to a database $\mathcal{D}$ a relation over the domain of $\mathcal{D}$. In this paper, 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 a *sentence* 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), XPath and so on.

Given a query language $\mathcal{L}$, the *model checking problem of* $\mathcal{L}$ is the computational problem of given a **sentence** $q \in \mathcal{L}$ and a database $\mathcal{D}$, to test whether $\mathcal{D} \models q$ or not. The database $\mathcal{D}$ is often restricted to a class $\mathcal{C}$ of structures. In this case we speak of *the model checking problem of* $\mathcal{L}$ *over* $\mathcal{C}$.

Given a query language $\mathcal{L}$, the *query answering problem of* $\mathcal{L}$ is the computational problem of, given a query $q \in \mathcal{L}$ and a database $\mathcal{D}$, to compute $q(\mathcal{D})$. As for model checking, it is often restricted to a class $\mathcal{C}$ of structures. In this case we speak of *the query answering problem of* $\mathcal{L}$ *over* $\mathcal{C}$.

The definition of the query answering and model checking problems given above correspond to their *combined complexity*. When the query is no longer part of the input, we speak of their *data complexity*.

2.2 Model of computation

As we will deal with linear time, it is important to make our model of computation precise. We use Random Access Machines (RAM) with addition and uniform cost measure as a model of computation, cf. [2]. Here are the features of this model that are the most relevant for this paper. On an input of size n, a RAM has a certain number of registers, each of them containing $\log n$ bits, and the machine can modify the content of these registers as a Turing Machine would do, but can also use these registers for accessing directly the memory cell pointed by the register or performing numerical operations (typically additions) between them. An access to memory using a register or an addition of two registers is counted as a unit time in a RAM (i.e. counts as constant time).

In our setting, the input will be a relational database. We do not define precisely how databases are encoded as inputs of a RAM machine. This is identical as for Turing Machines and this is described in details in many textbooks, cf. [1]. It only matters here that we can enumerate the domain or a relation of a database in linear time. We fix for now on a reasonable encoding of structures by words over some finite alphabet. The *size* of $\mathcal{D}$, denoted by $\|\mathcal{D}\|$, is the length of the encoding of $\mathcal{D}$.

An important observation is that the RAM model can sort m elements of size $O(\log m)$ in time $O(m \log m)$ [26]. In particular, we can sort the domain of $\mathcal{D}$ in time $O(\|\mathcal{D}\|)$, i.e. in linear time. This fact is implicit in many of the results below.

In the sequel when we say that the model checking problem of $\mathcal{L}$ over $\mathcal{C}$ can be solved in linear time, we refer to the data complexity, i.e. we mean that it can be solved in $O(\|\mathcal{D}\|)$, where the big O may depend on q.

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 model checking and query answering problems. 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 [23].

In parametrized complexity, a problem is an input together with a parameter, as a number computable from the input, and a question. A typical example is the parametrized model checking problem where the input is a database $\mathcal{D}$ and a sentence 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, on input of size n and parameter k, it can be solved in time $f(k)n^c$ for some suitable computable function f and constant c.

The idea behind this definition is that for many scenarios, like query answering in databases, it is preferable to have an algorithm working in $2^k n^2$ rather than n^k.

In parametrized complexity there is also a suitable notion of reduction, called FPT-reductions. It is such that FPT is closed under FPT-reductions. 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. In parametrized complexity, completeness relative to a complexity class is always understood to be under FPT-reductions.

An important hard class is denoted W[1]. 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 model checking problem for CQ [36]. It takes as input a database $\mathcal{D}$ and a sentence $q \in$ CQ, as parameter $|q|$, and asks whether $\mathcal{D} \models q$.

Another important hard class is denoted AW[$*$]. It plays in parametrized complexity the role of PSpace in classical complexity. A typical problem which is complete for AW[$*$] is the parametrized model checking problem for FO [36]. It takes as input a database $\mathcal{D}$ and a sentence $q \in$ FO, as parameter $|q|$, and asks whether $\mathcal{D} \models q$.

2.4 The enumeration class CONSTANT-DELAY$_{lin}$

For a query $q(\bar{x})$, the enumeration problem of q is, given a database $\mathcal{D}$, to output the elements of $q(\mathcal{D})$ one by one with no repetitions. The maximal time between any two consecutive outputs of elements of $q(\mathcal{D})$ is called *the delay*. The following definition requires a constant time between any two consecutive outputs.

We say that the enumeration problem of q is in the class CONSTANT-DELAY$_{lin}$ if it can be solved by a RAM algorithm which, on input $\mathcal{D}$, can be decomposed into two steps:

- a precomputation phase that is performed in time $O(\|\mathcal{D}\|)$,
- an enumeration phase that outputs $q(\mathcal{D})$ with no repetitions and a constant delay between two consecutive outputs. The enumeration phase has full access to the output of the precomputation phase but can use only a constant total amount of extra memory[1].

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 $\mathcal{D}$, 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. 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 linear time. 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 $\bar{w} \notin R^{\mathcal{D}}$ such that all tuples between $\bar{u}$ and $\bar{w}$ are in $R^{\mathcal{D}}$ and $\bar{v}'$ is the smallest (relative to the lexicographical order) element $\bar{w} \in R^{\mathcal{D}}$ bigger than $\bar{v}'$. These functions can be computed in linear time 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 enumeration phase is now trivial. 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 $f(\bar{v}) \neq g(\bar{v})$. 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 clearly constant delay as an output is performed at least every other step. All output tuples are clearly not in $R^{\mathcal{D}}$ and the reader can check that all skipped tuples are not 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 a 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 linear time 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.*

[1] In the literature one can sometimes find a more liberal definition only requiring constant delay with no constraints on the memory. Of course this implies that between two consecutive outputs the memory used is constant, but the global memory affected could be linear in the total number of outputs. In our more constrained setting the enumeration algorithm is essentially a finite state automaton running over the index structure produced during the precomputation phase.

EXAMPLE 3. *The schema is now a binary relation R together with unary relations. We consider the class of databases that are colored trees (for instance XML documents). Let $q(x, y)$ be the query returning all pairs (x, y) such that x is blue, y is red and x is an ancestor of y. Again, going through all pairs would yield a non-constant delay. There exists simple labeling scheme, computable in linear time, such as the "interval scheme", allowing to test in constant time whether x is an ancestor of y. But that is not enough as for a blue node x there might be many red nodes y that are not descendant of x. Adding the "document order" to each node during the preprocessing phase, together with a pointer to its nearest red node relative to that order, solves this problem, and yields a constant delay enumeration algorithm.*

REMARK 1. Notice that if the enumeration problem of q is in CONSTANT-DELAY$_{lin}$, then all answers can be output in time $O(\|\mathcal{D}\| + |q(D)|)$ and the first output is computed in time linear in $\|\mathcal{D}\|$. In the particular case of boolean queries, the associated model checking problem must be solvable in time linear in $\|\mathcal{D}\|$.

As shown in the examples above, an enumeration algorithm, on input $\mathcal{D}$, will first build a powerful index structure during the precomputation phase, and then use this index structure for outputting all answers.

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. In all the scenarios described in this paper there were known linear time algorithms for computing $q(D)$ when q is unary. In those scenarios enumeration 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.

Given a query language $\mathcal{L}$ and a class $\mathcal{C}$ of databases, we say that the enumeration of $\mathcal{L}$ over $\mathcal{C}$ is in constant delay if for any $q \in \mathcal{L}$ the enumeration problem of q over $\mathcal{C}$ is in CONSTANT-DELAY$_{lin}$.

REMARK 2. Notice that the definition of the enumeration of $\mathcal{L}$ over $\mathcal{C}$ is in constant delay, does not say anything about whether the enumeration algorithm can be computed from the query $q \in \mathcal{L}$ or not. It only specifies that for each q an enumeration algorithm exists. If the enumeration algorithm can be automatically obtained from q, we then say that the enumeration of $\mathcal{L}$ is *generic*. This is the case for all the scenarios described in this paper and all the practical scenarios of database query languages known to the author. In the generic case, it follows from Remark 1 that having an enumeration problem for $\mathcal{L}$ over $\mathcal{C}$ in CONSTANT-DELAY$_{lin}$ implies that the model checking problem for $\mathcal{L}$ over $\mathcal{C}$ is in FPT. Hence parametrized complexity can be used to show non membership in CONSTANT-DELAY$_{lin}$. For instance if the model checking problem for $\mathcal{L}$ over $\mathcal{C}$ is known to be W[1]-hard, then the enumeration problem for $\mathcal{L}$ over $\mathcal{C}$ cannot be in CONSTANT-DELAY$_{lin}$, unless W[1] =FPT. We will implicitly assume genericity in this paper.

REMARK 3. The class CONSTANT-DELAY$_{lin}$ is not known to be closed under boolean operations. Closure under disjunction is prevented by the requirement that each solution must be output only once. There are two particular cases when closure under disjunction can be obtained. The first one is trivial: It assumes that we have CONSTANT-DELAY$_{lin}$ algorithms for q and q' over a class $\mathcal{C}$ of databases and that, on input $\mathcal{D} \in \mathcal{C}$, both algorithms output the answers relative to the same linear order on all tuples (for instance the lexicographical order). In this case a simple argument that resembles the problem of merging two sorted lists gives a CONSTANT-DELAY$_{lin}$ algorithm for $q \vee q'$ over $\mathcal{C}$. The second case is more subtle. Instead of assuming a linear order on the output tuples, it assumes that after preprocessing in time linear in $\|\mathcal{D}\|$, given a tuple $\bar{a}$, one can test whether $\mathcal{D} \models q(\bar{a})$ in constant time. Then there is a CONSTANT-DELAY$_{lin}$ algorithm for $q \vee q'$ over $\mathcal{C}$ [39].

3. CONJUNCTIVE QUERIES

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

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

where $R_i(\bar{z}_i)$ is an *atom* of q, R_i being a relational symbol and $\bar{z}_i$ containing variables from $\bar{x}$ or $\bar{y}$. As mentioned in Section 2.3, the model checking problem of CQ is W[1]-complete. Therefore, by Remark 2, unless W[1] =FPT, the enumeration problem for CQ cannot be in constant delay. For *acyclic* conjunctive queries (ACQ) however, the model checking problem is known to be in FPT with a linear dependency in the size of the database [42]. Therefore we can hope for constant delay enumeration. We start by recalling the main definitions.

A conjunctice query is said to be *self-join free* if for any two atoms of the query, the associated relational symbols are different.

A *join tree* for a conjunctive query q 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 associated to q is α-acyclic.

The best known algorithm for evaluating a query $q \in$ ACQ runs in time $|q| \cdot \|\mathcal{D}\| \cdot |q(\mathcal{D})|$ [42]. This is not yet of the

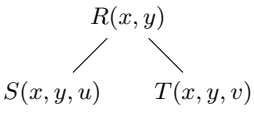
form $f(|q|) \cdot (\|\mathcal{D}\| + |q(\mathcal{D})|)$ implied by any constant delay enumeration algorithm. Actually, we will see that it is very unlikely that constant delay enumeration can be achieved for all queries in ACQ. Constant delay enumeration is only obtained for a subset of ACQ called *free-connex* that we now define.

An acyclic conjunctive query $q(\bar{x})$ is said to be *free-connex* if the query $q \wedge R(\bar{x})$ where R is a new symbol of appropriate arity, is acyclic[2].

Consider for example the following acyclic conjunctive query:

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

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



Notice that for boolean ACQ this is the usual notion of acyclicity. For non boolean queries being free-connex is a stronger requirement. For instance the reader can check that the simple acyclic query

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

is not free-connex as the query $\exists z \;\; S(x, z) \wedge S(z, y) \wedge R(x, y)$ is clearly cyclic.

THEOREM 1. *[7] The enumeration of free-connex* ACQ *over the class of all structures is in* CONSTANT-DELAY$_{lin}$.

The result of Theorem 1 also holds if the queries contain inequalities (ACQ$^{\neq}$). In this case atoms with inequalities are not involved when building the (generalized) join trees. Recall that in general the evaluation of a query $q \in$ ACQ$^{\neq}$ requires $f(|q|) \cdot \|\mathcal{D}\| \cdot |q(\mathcal{D})|$ steps, where f is an exponential function [7].

It turns out that free-connexity characterizes exactly those acyclic queries that can be enumerated in constant delay, assuming the query is self-join free and assuming matrix multiplication cannot be done in quadratic time.

The 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 [13]) requires more than $n^{2.37}$ steps.

THEOREM 2. *[7] If boolean matrix multiplication cannot be done in quadratic time then the following are equivalent for self-join free acyclic queries* $q \in$ ACQ*:*

1. *q is free-connex*
2. *q can be enumerated in* CONSTANT-DELAY$_{lin}$
3. *q can be evaluated in time $O(\|\mathcal{D}\| + |q(\mathcal{D})|)$.*

[2]This is not the initial definition of free-connex as given in [7]. This presentation is from Brault-Baron [11]

Remark: We already mentioned that most of the results described above extends to the case where atoms could also be inequality statements. In the case of *signed conjunctive queries* (SCQ), where atoms could be negated, under a suitable notion of acyclicity (somewhere between α-acyclicity and β-acyclicity) the model checking problem was shown to be $O(n \log n)$, where $n = \|\mathcal{D}\|$ (the constant factor is polynomial in the size of the query) [11]. The same set of acyclic queries can be enumerated with an algorithm having a $n \log n$ preprocessing phase and $\log n$ delay [11].

4. SPARSE STRUCTURES

In this section we consider first-order queries (FO) and study classes of databases for which enumeration can be achieved in constant delay.

All the classes of databases considered are defined over graphs and are generalized to arbitrary relational structures via their Gaifman graphs.

A graph is a relational structure $G = (V, E)$, where V is the set of nodes and $E \subseteq V^2$ the set of edges (edges are undirected, so $(x, y) \in E$ implies $(y, x) \in E$). In the case of graphs, we will write $|G|$ to denote the number of nodes of G (i.e. the size of V), while we write $||G||$ to denote the number of edges of G (i.e. the size of E).

The *Gaifman graph* of a relational structure $\mathcal{D}$ is defined as follows: the set of vertices is the domain D of $\mathcal{D}$ and there is an edge (a, b) iff there exists a relation R_i and a tuple $t \in R_i$ such that both a and b occur in t.

Given a class $\mathcal{C}$ of graphs, the associated class $\mathcal{C}$' of databases contains exactly all the databases whose Gaifman graphs are in $\mathcal{C}$.

4.1 Bounded degree

One way to have the model checking problem for FO with a linear time data complexity is to restrict the structures. For instance it is known that FO sentences can be tested in linear time over structures with bounded degree [38].

A graph has *degree* less than d if each node has at most d neighbors. A class of graphs has *bounded degree* if there exists a d such that all graphs in the class have degree less than d.

The linear time model-checking algorithm for FO over structures with bounded degree can be lifted to a enumeration algorithm.

THEOREM 3. *[18, 28] The enumeration of* FO *over a class of structures with bounded degree is in* CONSTANT-DELAY$_{lin}$.

A key property of structures of degree d is that, for a given r, there are only finitely many, up to isomorphism, possible

r-neighborhoods (i.e. including nodes at distance at most r). Given a query $q \in$ FO the Gaifman Locality Theorem tells us that only the r-neighborhood types are relevant, for a suitable value of r depending only on q. One can then show that it is possible to recolor in linear time, hence during the preprocessing phase, each node with its neighborhood type. Based on these colors and the Gaifman Locality Theorem it is then not too difficult to derive an enumeration algorithm [28].

Another key property of structures of degree d is that they can easily be encoded using bijective unary functions. Moreover, for such structures, there exists a quantifier elimination method for FO queries [18]. Once the formula is quantifier free, it is not too difficult to design a constant delay enumeration algorithm. This is the approach taken by [18].

The constants involved in the enumeration algorithms (the constant factor in the preprocessing phase and the constant delay in the enumeration phase) is a tower of exponential depending on $|q|$ in the case of [18] and is triply exponential in $|q|$ in the case of [28]. This latter constant factor cannot be significantly improved: it follows from [25] that a constant factor of only doubly exponential in the size of the formula is not possible unless AW[$*$] =FPT.

4.2 Bounded expansion

The bounded degree case can be generalized to a larger class of structures known as *bounded expansion* and defined in [33].

In order to define structures with bounded expansion we need some terminology from graph theory.

Let $G = (V, E)$ be a graph. For any node $v \in V$ and any $r \in \mathbb{N}$ we denote by $B_r(v)$ the r-ball around v, i.e. the set of nodes of G that are reachable from v by paths of lengths up to r. We say that a graph H is a r-minor of G if the nodes $v_1, \ldots, v_k$ of H are also nodes of G and there exists pairwise non-overlapping sets $S_1, \ldots, S_k$ such that each S_i is a connected set of nodes of G verifying $v_i \in S_i \subseteq B_r(v_i)$ and there is an edge between v_i and v_j in H iff there is an edge in G from a node of S_i to a node of S_j. The set of all r-minors of G is denoted by $G\nabla_r$.

For a graph G the *greatest reduced average density (grad) of G with rank r* is:

$$\nabla_r(G) = \max_{H \in G\nabla_r} \frac{||H||}{|H|}.$$

We say that a class $\mathcal{C}$ of graphs has *bounded expansion* if there exists a computable function $f : \mathbb{N} \to \mathbb{R}$ such that for all graphs $G \in \mathcal{C}$ and for all $r \in \mathbb{N}$ we have:

$$\nabla_r(G) \leq f(r).$$

In [33] a number of definitions equivalent to bounded expansion was shown giving evidence that this class is robust. Many known families of graphs have bounded expansion. We list below some notable examples.

- Class of graphs with bounded degree.
- Class of graphs with bounded treewidth.
- Class of planar graphs.
- Class of graphs excluding at least one minor.

It is easy to see that this definition generalizes the bounded degree case. If a graph G has degree bounded by d, then $||G|| \leq d|G|$. Moreover, if H is a r-minor of G then the degree of H is at most d^r. Hence $\nabla_r(G) \leq d^r$. A similar argument explains why this definition also generalizes the bounded treewidth case. Euler's formula can be used to show that planar graphs have bounded expansion.

The model checking problem of FO over classes of structures with bounded expansion was shown to be linear in [21, 27]. This has been extended to constant delay enumeration:

THEOREM 4. *[29] The enumeration of* FO *over the class of structures with bounded expansion is in* CONSTANT-DELAY$_{lin}$.

This result generalizes the bounded degree case and Theorem 3. The index necessary for having a constant delay algorithm is quite complicated and rely on deep graph theoretical constructions. Moreover, the current enumeration algorithm with constant delay has huge hidden constants: a tower of exponentials in the quantifier alternation depth of the first-order query. This non-elementary constant factor is unavoidable already on the class of unranked trees, assuming FPT$\neq$AW[$*$] [25]. In comparison, recall that this factor is triply exponential in the size of the query in the bounded degree case [38, 28].

4.3 Nowhere dense

It turns out that the notion of bounded expansion can be further generalized in an interesting way.

Recall the definition of $G\nabla_r$ as the set of all r-minors of a graph G. Given a class $\mathcal{C}$ of graphs, we denote by $\mathcal{C}\nabla_r$ the set of all r-minors of all graphs of $\mathcal{C}$.

To a class $\mathcal{C}$ of graphs we associate the following number

$$\lim_{r \to \infty} \limsup_{H \in \mathcal{C}\nabla_r} \frac{\log ||H||}{\log |H|} \tag{1}$$

As for any graph H we have $||H|| \leq |H|^2$, the number associated to a class $\mathcal{C}$ as defined in (1) is less than 2. It turns out that this number can take exactly three possible values: $\{0, 1, 2\}$ [34].

Following [34], we say that a class of graphs is *nowhere dense* if its associated number is 0 or 1. It is not difficult to see that this definition extends the class of graphs with bounded expansion: Assume $\mathcal{C}$ has bounded expansion and let $f(r)$ be the function bounding $\nabla_r(G)$ for all $G \in \mathcal{C}$. Then for every graph $H \in \mathcal{C}\nabla_r$ we have $||H|| \leq f(r)|H|$. Hence

$$\frac{\log ||H||}{\log |H|} \leq 1 + \frac{\log f(r)}{\log |H|}$$

and therefore (assuming $\mathcal{C}$ has infinitely many graphs)

$$\limsup_{H \in \mathcal{C}\nabla_r} \frac{\log ||H||}{\log |H|} \leq 1$$

The notion of nowhere dense also encompass any class of graphs that locally excludes a minor or that has local bounded treewidth. We refer to [17, 24] for precise definitions of these notions.

It is a major open problem to know whether model checking of FO over nowhere dense graphs can be done in linear time. Actually, we don't even know whether this can be done in FPT. Recall that by Remark 2 this is a prerequisite for obtaining constant delay enumeration.

OPEN PROBLEM 1. *Is model checking of* FO *over the class of nowhere dense graphs in* FPT*?*

On the upper-bound side, we know that the model checking of *existential first-order formulas* can be done in time $O(n^{1+\epsilon})$, for any positive ϵ, where $n = \|\mathcal{D}\|$ [35]. It is possible to transform this model checking algorithm into an enumeration algorithm whose preprocessing phase is in $O(n^{1+\epsilon})$ and whose delay is in $O(n^{\epsilon})$, for any positive ϵ.

On the lower-bound side, one can show that if a class of graphs is not nowhere dense, we then say it is somewhere dense, then no FPT algorithm exists (unless W[1] =FPT). Hence nowhere dense graphs is the maximal class of graphs where we can hope for FPT algorithms and constant delay enumeration.

THEOREM 5. *[22] If $\mathcal{C}$ is a somewhere dense class of graphs closed under subgraphs, then model checking of* FO *(actually existential formula suffices) is* W[1]*-hard.*

An even stronger result was obtained in [31] assuming that $\mathcal{C}$ is somewhere dense in an "effective way". In this case it is shown that model-checking of FO is already AW[$*$]-complete.

5. DATA TREES AND XPATH

In this section we switch to trees, actually data trees, and the query language XPath.

A data tree is a tree whose every node carries a label from a finite alphabet $\mathbb{A}$ and a datum from some infinite domain, here $\mathbb{N}$. This structure has been considered in the realm of semistructured data, timed automata, program verification, and generally in systems manipulating data values. In particular data trees can model XML documents, see for instance [9].

By XPath we refer to a fragment of XPath 1.0. It is a two-sorted language, with *path* expressions (that we write α, β) and *node* expressions (φ, ψ). Path expressions are binary relations resulting from composing the axis relations and node expressions. The axis relations, denoted AXIS, are the usual CHILD, PARENT, DESCENDANT, ANCESTOR, NEXT-SIBLING, RIGHT-SIBLING, PREVIOUS-SIBLING and LEFT-SIBLING relations. Node expressions are boolean formulas that test a property of a node, like for example, that it has a certain label, or that it has a child labeled a with the same data value as an ancestor labeled b and so on. For comparing data values, we allow any predicate in the set RELOP:= $\{=, \neq, <, >, \leq, \geq\}$.

The syntax of XPath is given below:

$$\alpha, \beta \ :: \ \text{AXIS} \mid [\varphi] \mid \alpha\beta \mid \alpha \cup \beta$$
$$\varphi, \psi \ :: \ \mathbb{A} \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \langle\alpha\rangle \mid \alpha \ \text{RELOP} \ \beta$$

We also consider an extension of XPath allowing the Kleene star on *any* path expression and we denote it by regular XPath. It's semantics over data trees is classical and quite intuitive. It can be found in details for instance in [10]. Typically a path expression $[\varphi]$ selects all the pairs (u, u) such that the node expression φ is true at u. A node expression $\langle\alpha\rangle$ selects all nodes u such that there is a node v with (u, v) selected by the path expression α. Finally a node expression α RELOP β selects all nodes u such that there exist v, w such that (u, v) is selected by the path expression α, (u, w) is selected by the path expression β and the data values of v, w are related accordingly to the predicate in RELOP.

We view formulas of regular XPath as *queries* over data trees. This query is unary and returns a set of nodes if the formula is a node expression and it is binary and returns pairs of nodes if the formula is a path expression.

THEOREM 6. *[10] The enumeration of regular* XPath *over the class of data trees is in* CONSTANT-DELAY$_{lin}$.

It turns out that the constants involved in the algorithm needed for enumerating regular XPath queries are not very big. They are polynomial in the size of the query if this query is in XPath and exponential in the size of the query if this query is in regular XPath.

The index structure constructed during the preprocessing phase is quite involved and builds on the one developed for Proposition 8 of the following section.

6. STRUCTURES WITH BOUNDED TREEWIDTH

Our last scenario concerns MSO queries over structures with bounded treewidth. The associated model checking problem is linear by Courcelle's theorem [14]. The notion of bounded treewidth extends the notion of trees and its precise definition is not important for this paper. The reader is referred to [23, chapter 11] for more details.

Recall that MSO extends FO with the possibility to quantify existentially and universally over monadic second order variables. Those variables range over sets of elements of the input domain. By MSO query we mean here a query of the form $q(\bar{x})$ where q is in MSO and $\bar{x}$ are first-order free variables. The case where $\bar{x}$ can also contain free monadic variables has also been considered in [15, 4] but those cannot be enumerated in CONSTANT-DELAY$_{lin}$ mainly because outputting one solution may require linear time. See Section 7.

However, when restricted to first-order free variables, constant delay enumeration can be achieved. Two different index structures were proposed in the literature. Actually a third one was also proposed in [15], but it requires a precomputation phase of $O(n \log n)$ to build it.

THEOREM 7. *[4, 30] The enumeration of* MSO *queries over the class of structures with bounded treewidth is in* CONSTANT-DELAY$_{lin}$.

The index structures built to prove Theorem 7 is quite complex. The difficulty lies entirely in the tree case. We give below hints on how to solve a special case, based on ideas developed in [30] as this special case is already of independent interest.

Let L be a regular word language over an alphabet $\mathbb{A}$. A typical binary MSO query over trees is the query $q_L(x, y)$ returning the pairs of nodes (u, v) within a tree such that u is an ancestor of v and the labels of the nodes in the path from u to v forms a word in L (this generalizes Example 3).

Given a tree $\mathbf{t}$, there exists an index structure such that, given two nodes u and v one can test in constant time whether $(u, v) \in q_L(\mathbf{t})$ or not. Moreover this index structure can be computed in time linear in $\|\mathbf{t}\|$. This is a nontrivial and powerful result of Colcombet.

PROPOSITION 8. *[12] For any regular language L over an alphabet $\mathbb{A}$ and any $\mathbb{A}$-labeled tree $\boldsymbol{t}$ one can*

- *construct in time $O(\|\boldsymbol{t}\|)$ an index structure such that,*
- *for all nodes u, v of $\boldsymbol{t}$, testing whether $(u, v) \in q_L(\boldsymbol{t})$ can be done in constant time.*

This is a deep result based on algebraic constructions. The constants involved during the construction of the index and during the constant time tests depend on the presentation of L. They are non elementary in L if L is given as an MSO sentence. They are exponential in L if L is given as an automaton, even in the deterministic case. However, there exist cases where these constants are polynomial. In particular this is the case of the *basic automata model* introduced in [10] in order to capture the navigational power of XPath and used in the proof of Theorem 6.

It turns out that the index structure built for proving Proposition 8 has many other interesting consequences. In particular, a normal form for MSO queries over trees.

PROPOSITION 9. *[implicit in[12]] Over trees, every binary* MSO *query $q(x, y)$ is equivalent to a disjunction of queries of the form $\exists \bar{y} \forall \bar{z}\ \theta$, where θ is a disjunction of conjunctions of atomic predicates or* **unary** MSO *queries.*

The index constructed in [30] for enumerating MSO queries over trees builds on Proposition 9. The so called "composition method", or a simple Ehrenfeucht-fraïssé game, shows that any MSO query is equivalent to a boolean combination of binary queries. For binary queries, Proposition 9 applies. The unary MSO subformulas can be precomputed in linear time by Courcelle's theorem and can therefore be considered as new colors. Hence it is enough to consider $\exists \bar{y} \forall \bar{z}$ first-order queries. Those queries being rather simple, an induction on the number of free variables solves the problem, see [30]. It turns out that the constants involved for enumerating MSO queries in Theorem 7 deviates from those of Proposition 8 only by a polynomial factor. Hence their size depends on the presentation of the MSO query as explained above.

7. DISCUSSION

7.1 The impact of order

With the current definition of CONSTANT-DELAY$_{lin}$, there is no constraint on the order in which the solutions 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 solutions of the query to be output in the order of their relevance.

Requiring a specific order when outputting the solutions of 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.

In the presence of a linear order on the database, the enumeration algorithms of Theorem 3 (bounded degree) and Theorem 4 (bounded expansion) can output the solutions in lexicographical order. However, it is not clear how to achieve lexicographical output in the case of MSO over bounded treewidth (Theorem 7).

7.2 Longer delay

Delay linear in the size of the database. We could consider enumeration algorithms allowing for non constant delay. An interesting case is linear delay. 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 structures. A class of structures, called $\underline{\mathrm{X}}$-structures, has been exhibited such that CQ can be enumerated over it with linear delay. We will not define $\underline{\mathrm{X}}$-structures in this note. Typical examples are grids and trees with all XPath axis.

THEOREM 10. *[6]. The enumeration of* CQ *over $\underline{\mathrm{X}}$-structures can be done with linear delay.*

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

THEOREM 11. *[7]. The enumeration of* ACQ *over all structures can be done with linear delay.*

Delay linear in the size of the output. A trivial case when constant delay enumeration cannot be achieved is when the size of one output is too big. This is for instance the case when considering MSO formulas with monadic second-order free variables. Then each answer is a tuple of sets of elements of the domain and can have a size linear in the size of the database. In constant time such an answer can not even be written in the output tape. For such queries it is convenient to allow a delay linear in the size of the output, but still independent from the size of the database[3]. We then speak of an output-linear delay.

The result of Theorem 7 can be generalized to this setting (the preprocessing phase of [15] is actually not linear, but in $O(\|\mathcal{D}\| \log \|\mathcal{D}\|)$) .

THEOREM 12. *[15][4] The enumeration of* MSO *(allowing monadic second-order free predicates) over the class of structures with bounded treewidth can be done with output-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 [20].

7.3 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, the counting problem of q is the computational problem of given a database $\mathcal{D}$ to compute $|q(\mathcal{D})|$. If the enumeration problem of q is in CONSTANT-DELAY$_{lin}$, then the counting problem of q can be done in $O(\|\mathcal{D}\| + |q(\mathcal{D})|)$. In particular it could be exponential in the arity of q, but this is clearly not the most efficient way to do this.

Given a query language $\mathcal{L}$ and a class $\mathcal{C}$ of databases, we say that the counting problem of $\mathcal{L}$ over $\mathcal{C}$ is solvable in time $f(n)$ if for any $q \in \mathcal{L}$ the counting problem of q over $\mathcal{C}$ can be solved in time $f(\|\mathcal{D}\|)$ on input $\mathcal{D}$. Note that f does not depend on q. There is again the genericity issue, i.e. whether the algorithm solving the counting problem for q in time $f(n)$ can be computed from q. We will always assume that this is the case here. In that case, if f is polynomial, then the computational parametrized problem of given a query $q \in \mathcal{L}$ and a database $\mathcal{D} \in \mathcal{C}$ with $|q|$ as a parameter to compute $|q(\mathcal{D})|$ is in the class FPT.

On structures with bounded treewidth counting can be done in linear time for MSO queries. Note that the constant factor is not elementary in the query size.

THEOREM 13. *[3] The counting problem for* MSO *over the class of structures with bounded treewidth can be solved in linear time.*

Counting can also be done in linear time for first-order queries over structures with bounded degree or bounded expansion. For the bounded degree case this was shown in [8]. In the bounded expansion case, which generalizes the bounded degree case, this was shown in [35] for existential formulas by reducing it to Theorem 13. The quantifier elimination methods of [22, 27, 29] then reduces the general case to this case. A direct and simple proof that does not use Theorem 13 was given in [29]. In all cases, the constant factor is not elementary in the query size.

THEOREM 14. *The counting problem for* FO *over the class of structures with bounded expansion can be solved in linear time.*

In the nowhere dense case the problem was solved for existential formulas, but with a quasi linear time. We don't know whether this can be improved.

THEOREM 15. *[35] For any positive ϵ, the counting problem for existential first-order queries over the class of nowhere dense structures can be solved in time $O(n^{1+\epsilon})$.*

With no restrictions on the structures, counting the number of solutions of a query is a hard problem. Already for acyclic conjunctive queries the combined complexity is $\#P$-complete [37] and only the quantifier free ACQ can be solved in time linear in $\|\mathcal{D}\|$ [5].

For this reason, [19] 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 ACQ.

THEOREM 16. *[19] For each number s, the counting problem for* ACQ *with quantified-star size bounded by s over the class of all structures can be solved in time polynomial in both the query and the structure.*

It turns out that this parameter characterizes exactly the class of ACQ having a tractable counting problem. If a class of ACQ does not have a bounded quantified-star size, then its associated counting problem is #W[1]-hard [19]. In particular, it cannot be solved in FPT.

[3]There is actually another approach which consists in having an output tape and only modify the output tape in order to transform the previous solution into the next one. In special cases the delta between two consecutive solutions only affect a constant part of the output and the enumeration can be done with constant delay, see for instance [20].

Testing whether a given tuple is a solution. Given a query q and a database $\mathcal{D}$, after a preprocessing on $\mathcal{D}$, we would like to be able to test efficiently whether a tuple $\bar{a}$ belong to $q(\mathcal{D})$.

Let $\mathcal{C}$ be a class of databases. We say that the membership test for q over $\mathcal{C}$ is constant-time modulo linear preprocessing if for all $\mathcal{D} \in \mathcal{C}$ it is possible to:

- construct in time $O(\|\mathcal{D}\|)$ an index structure such that,
- for all tuple $\bar{u}$ of elements of $\mathcal{D}$, testing whether $\bar{u} \in q(\mathcal{D})$ can be done in constant time.

The following is a simple consequence of Proposition 8 and the composition method:

THEOREM 17. *[12] The membership test for* MSO *queries over structures with bounded treewidth is constant-time modulo linear preprocessing.*

For sparse structures we can obtain the same results but for first-order queries (the bounded degree case was considered in [32]):

THEOREM 18. *[29] The membership test for* FO *over structures with bounded expansion is constant-time modulo linear preprocessing.*

Computing the j^{th} solution. In this section we assume a linear order $<$ on all tuples, for instance the lexicographical order for some linear order on the domain. Given a query q, we say that $\text{Jth}(q, <)$ is solvable in $\text{dyn}(f, g)$ if there is a RAM algorithm such that, on input $\mathcal{D}$ and j, works as follows:

- a precomputation phase, independent of j, and working in time $O(f(\|\mathcal{D}\|))$
- an output phase working in time $O(g(\|\mathcal{D}\|, j))$ and outputting the j^{th} element of $q(\mathcal{D})$ relative to $<$.

This problem is interesting because it allows to perform a sampling on $q(\mathcal{D})$. Indeed, any random generation of integers can then be used to obtain a good sampling on $q(\mathcal{D})$. If the enumeration problem of q is in $\text{CONSTANT-DELAY}_{lin}$ and the solutions are output in the order of $<$, then $\text{Jth}(q, <)$ is in $\text{dyn}(n, j)$. Note that j could be polynomial in n.

In the bounded degree case we can do better:

THEOREM 19. *[8] For $q \in$ FO, over the classes of structures with bounded degree, $\text{Jth}(q, <)$ is in $\text{dyn}(n, 1)$ for some order $<$ depending on q.*

In the bounded treewidth case we can do the following:

THEOREM 20. *[5] For $q \in$ MSO, over the classes of structures with bounded treewidth, $\text{Jth}(q, <)$ is in $\text{dyn}(n, \log n)$ for some order $<$ depending on q.*

It is not clear whether these two result can be proved for the case where $<$ is the lexicographical order on a linearly ordered database. But even with the specific orders, these results are useful because they allow random sampling of the answers of a given query.

7.4 Other enumeration problems

In this abstract we focused on the problem of enumerating the output of a query on a database. There exist many enumeration algorithms for various kinds of problems like enumerating all the solutions of a SAT instance [16], enumerating monomials of a polynomial [40], enumerating perfect matchings in bipartite graphs [41] and so on. The interested reader is refereed to the thesis [39] for learning more about enumerations outside of the database context.

8. CONCLUSIONS

We have survey many results about constant delay enumeration and related problems. We hope that we succeeded to convince the reader that this is a very interesting topic.

The main open problem is probably the evaluation of first-order queries over nowhere dense structures mentioned in Open Problem 1.

One could also consider relaxing the "no duplicate" constraint and enumerate conjunctive queries with the bag semantic.

We would like to conclude with lower bounds. 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 concrete problems mentioned in this note, the lower bounds have been proved using complexity assumptions, either in parametrized complexity, or for the Boolean Matrix Multiplication problem. But it is also plausible (i.e. there are no known 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.

9. REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison Wesley, 1995.

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

[3] S. Arnborg, J. Lagergren, and D. Seese. Easy Problems for Tree-Decomposable Graphs. *J. of Algorithms*, 12(2):308–340, 1991.

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

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

[6] 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.

[7] 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.

[8] 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.

[9] M. Bojańczyk, A. Muscholl, T. Schwentick, and L. Segoufin. Two-variable logic on data trees and XML reasoning. *J. of the ACM*, 56(3), 2009.

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

[11] J. Brault-Baron. *De la pertinence de l'énumération : complexité en logiques propositionnelle et du premier ordre*. PhD thesis, Université de Caen, 2013.

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

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

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

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

[16] N. Creignou, F. Olive, and J. Schmidt. Enumerating All Solutions of a Boolean CSP by Non-decreasing Weight. In *Theory and Applications of Satisfiability Testing (SAT)*, pages 120–133, 2011.

[17] A. Dawar, M. Grohe, and S. Kreutzer. Locally Excluding a Minor. In *Symp. on Logic in Computer Science (LICS)*, pages 270–279, 2007.

[18] 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.

[19] A. Durand and S. Mengel. On Polynomials Defined by Acyclic Conjunctive Queries and Weighted Counting Problems. *CoRR*, abs/1110.4201, 2011.

[20] 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.

[21] Z. Dvořák, D. Král, and R. Thomas. Deciding First-Order Properties for Sparse Graphs. In *Symp. on Foundations of Computer Science (FOCS)*, pages 133–142, 2010.

[22] Z. Dvořák, D. Král, and R. Thomas. Testing first-order properties for subclasses of sparse graphs. *CoRR*, abs/1109.5036, 2011.

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

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

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

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

[27] M. Grohe and S. Kreutzer. *Model Theoretic Methods in Finite Combinatorics*, chapter Methods for Algorithmic Meta Theorems. American Mathematical Society, 2011.

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

[29] 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.

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

[31] S. Kreutzer and A. Dawar. Parameterized complexity of first-order logic. *Electronic Colloquium on Computational Complexity (ECCC)*, 16:131, 2009.

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

[33] J. Nešetřil and P. O. de Mendez. Grad and classes with bounded expansion I. Decompositions. *Eur. J. Comb.*, 29(3):760–776, 2008.

[34] J. Nešetřil and P. O. de Mendez. On nowhere dense graphs. *European J. of Combinatorics*, 32(4):600–617, 2011.

[35] J. Nešetřil and P. Ossona de Mendez. *Sparsity*. Springer, 2012.

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

[37] 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.

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

[39] Y. Strozecki. *Enumeration complexity and matroid decomposition*. PhD thesis, Université de Paris 7, 2010.

[40] Y. Strozecki. Enumeration of the Monomials of a Polynomial and Related Complexity Classes. In *Intl. Symp. on Mathematical Foundations of Computer Science (MFCS)*, pages 629–640, 2010.

[41] T. Uno. Algorithms for Enumerating All Perfect, Maximum and Maximal Matchings in Bipartite Graphs. In *Intl. Symp. on Algorithms and Computation*, pages 92–101, 1997.

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