A glimpse on constant delay enumeration

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 describe here several scenarios when this is possible.

1998 ACM Subject Classification F.4 MATHEMATICAL LOGIC AND FORMAL LANGUAGES

Keywords and phrases Enumeration, constant delay, logic

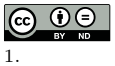
Digital Object Identifier 10.4230/LIPIcs.xxx.yyy.p

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, where the answer to the query is either a "yes" and a "no") and expressed as a function of the sizes of q and $\mathcal{D}$. In this case we know that the model checking problem for first-order queries is PSpace-complete, for conjunctive queries it is NP-complete and that for acyclic conjunctive queries it can be done in polynomial time. 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, for boolean queries, the model checking problem can be solved in time linear in the size of the database. However, as shown in [4], 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 [45].

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 practical 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.

The first part of the paper, Section 3, is devoted to conjunctive queries. We will see how acyclicity plays here a crucial role.

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 classes of databases with bounded degree, bounded treewidth, bounded expansion and low degree.

In Section 5 we will see that, in the bounded treewidth case, one can even enumerate monadic second-order queries 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. We will not survey those results here, the interested reader is referred to [41].

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.

2 Preliminaries

2.1 Database as finite relational structures, queries

In this paper a database is a finite relational structure. All interesting examples can be found over graphs or colored graphs. Hence the reader can safely replace relational structure with graph while reading this paper.

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}. 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* 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) and so on. As usual, $|q|$ denotes the size of q.

Given a query language $\mathcal{L}$, the *model checking problem for* $\mathcal{L}$ 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 finite structures. In this case we speak of *the model checking problem for* $\mathcal{L}$ *over* $\mathcal{C}$.

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* (respectively, *quasi-linear* or *constant time*) if it outputs the solution within $f(k)n$ steps (respectively, $f(k)n \log n$ steps or $f(k)$ steps), for some function f

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 structure comes with a linear order on the domain. If not, we use the one induced by the encoding of the structure as an input to the RAM. 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 $O(m \log m)$ [28]. 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 [24].

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 for $\mathcal{L}$ where the input is a database $\mathcal{D}$ and a sentence $q \in \mathcal{L}$, 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-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. 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 [39]. 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 [39].

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 linear time, 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 access to the output of the preprocessing phase and can use extra memory whose size depends only on q.[1]

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 function 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 $\mathcal{D}$, 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

[1] In the literature one can sometimes find a more liberal definition only requiring constant time 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. It turns out that most of the existing enumeration algorithms do satisfy the extra constraint on memory.

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 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 $f(\bar{v}) \neq g(\bar{v})$. 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 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.

▶ Remark. *Notice that if the enumeration problem for $\mathcal{L}$ over $\mathcal{C}$ is in CD∘Lin, then all answers can be output in time $O(\|\mathcal{D}\| + |q(D)|)$ and the first output is computed in time linear in $\|\mathcal{D}\|$. In particular the model checking problem for $\mathcal{L}$ over $\mathcal{C}$ is in* FPT. *Hence 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 CD∘Lin, unless* W[1] =FPT.

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 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.

▶ Remark. *One difficulty for obtaining constant delay enumeration algorithms is that the class CD∘Lin is not known to be closed under boolean operations. Closure under disjunction is difficult because of 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 CD∘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 CD∘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 CD∘Lin algorithm for $q \vee q'$ over $\mathcal{C}$ [42].

3 Restricting the queries

In this section we consider the evaluation of simple queries over the class of all databases.

3.1 Conjunctive queries

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 $\bar{y}$. A typical example is the distance 2 query of Example 2 in in CQ. Another example is the query returning all triangles in a graph. The model checking problem for CQ is W[1]-complete and we therefore restrict our attention to *acyclic* conjunctive queries (ACQ) that can be evaluated in time $|q| \cdot \|\mathcal{D}\| \cdot |q(\mathcal{D})|$ [45]. We will see that it is very unlikely that constant delay enumeration can be achieved for ACQ. It is only achieved for a subset of ACQ called *free-connex*. We start with the necessary 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* of $q \in$ CQ is a tree T whose nodes are atoms of q and such that

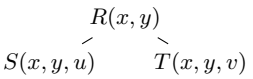
(i) each atom of q is the label of exactly one node of T,

(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 formed by the atoms of q is α-acyclic.

An acyclic conjunctive query $q(\bar{x})$ is said to be *free-connex* if the query $q(\bar{x}) \wedge R(\bar{x})$ where R is a new symbol of appropriate arity, is acyclic.[2] Note that all boolean acyclic query 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 not free-connex as the query $\exists z \;\; S(x,z) \wedge S(z,y) \wedge R(x,y)$ is clearly cyclic.

▶ Theorem 1. [4] The enumeration for free-connex ACQ over the class of all databases is in CD∘Lin.

We note that the multiplicative factors involved in Theorem [4] are polynomial in the query size.

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. In the presence of inequalities, the multiplicative factors are now exponential in the query size.

[2] This is not the initial definition of free-connex as given in [4]. This presentation is from Brault-Baron [9]

It turns out that free-connexity characterizes exactly those self-join free 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 [12]) require more than $n^{2.37}$ steps.

▶ Theorem 2. [4] If boolean matrix multiplication cannot be done in quadratic time then the following are equivalent for self-join free acyclic $q \in$ ACQ:

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.

3.2 Signed conjunctive queries

We are now interested in evaluating signed conjunctive queries (SCQ). Those extends 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 whiles 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 q^+ is α-acyclic. When q^+ is empty it has been shown in [8, 9] that constant delay enumeration can be achieved if q^- is β-*acyclic*. 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 limits case. It also allows for tractable enumeration algorithms.

▶ Theorem 3. [9] The enumeration for signed-acyclic SCQ over the class of all databases can be done with constant delay after a preprocessing time of the form $\|\mathcal{D}\|(\log \|\mathcal{D}\|)^{|q|}$.

The enumeration for signed-acyclic SCQ over the class of all databases can be done with logarithmic delay after a quasi-linear time preprocessing.

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 ACQ case, 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 cannot be avoided [9].

3.3 Guarded First-Order Queries

Guarded first-order formulas (GFO) are defined using the following grammar.

$$\phi ::= \quad R(\bar{x}) \mid x = y \mid \phi \wedge \phi \mid \neg\phi(x) \mid \exists \bar{x}\alpha(\bar{x}\bar{y}) \wedge \phi(\bar{x}\bar{y}) \mid \forall \bar{x}\alpha(\bar{x}\bar{y}) \rightarrow \phi(\bar{x}\bar{y})$$

where R is an arbitrary relation symbol from the schema and $\alpha(\bar{x}\bar{y})$ is an atom containing all variables in $\bar{x}\bar{y}$. See [27] for more details about guarded logics. It has been shown in [5] that the model checking for sentences from GFO could be done in linear time. This can be extended to a constant delay algorithm assuming acyclicity of the *quantifier-free part* of the query.

Indeed consider a subformula γ of the form $\exists\bar{x}\alpha(\bar{x}\bar{y}) \wedge \phi(\bar{x}\bar{y})$ where ϕ is quantifier free. It defines a relation $R_\gamma(\bar{y})$ whose size is linear in the size of the relation occurring in α. A simple argument as the one for $R(\bar{x}\bar{y}) \wedge S(\bar{x})$ explained in Section 2.2 shows that this relation can be computed in linear time.

Therefore, after a linear preprocessing, any GFO query can be transformed into a quantifier free one. Turned into DNF the resulting query is a union of SCQ. By a simple exclusion-inclusion argument the union can be assumed to give disjoint results. Hence it remains to enumerate each disjunct separately. From Theorem 3 this can be done efficiently if each disjunct is signed-acyclic.

This suggest the following definition. Given a GFO query q, it's *quantifier-free part* is the quantifier free query constructed from q by pushing negation down to the atoms and then replacing its maximal subformula $\gamma(\bar{y})$ of the form $\exists\bar{x}\alpha(\bar{x}\bar{y}) \wedge \phi(\bar{x}\bar{y})$ by $R_\gamma(\bar{y})$. It's *normalized quantifier-free part* further transforms the quantifier-free part by putting it into DNF and applying the exclusion-inclusion principle to get disjoint conjunctive formulas.

Let's denote by *signed-acyclic* GFO those queries of GFO whose *normalized quantifier-free part* are such that each conjunct is signed-acyclic. From the previous argument and Theorem 3 the following result follows:

▶ Theorem 4. The enumeration for signed-acyclic GFO over the class of all databases can be done with logarithmic delay after a quasi-linear preprocessing time.

Remark: the same result can probably be obtained with a more natural syntactic fragment of GFO.

4 Restricting the class of structures

In this section we consider first-order queries (FO) and restrict the classes of databases to sparse structures. All these classes are defined over graphs and are generalized to arbitrary relational structures via their Gaifman graphs: Given a class $\mathcal{C}$ of graphs, the associated class $\mathcal{C}$' of databases contains exactly all the databases whose Gaifman graphs are in $\mathcal{C}$.

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. For a graph G we denote by $|G|$ its number of vertices and by $||G||$ its number of edges.

4.1 Bounded degree

A class of graphs has *bounded degree* if there exists a d such that all nodes of all graphs in the class have at most d neighbors. It is known that the model checking problem for FO over structures with bounded degree can be solved in linear time [40].

▶ Theorem 5. [18, 33] The enumeration for FO over a class of structures with bounded degree is in CD∘Lin.

The initial proof of [18] is using the fact that structures in a class of graphs of bounded degree can be encoded using finitely bijective unary functions. Moreover, over such structures, there exists a quantifier elimination method for FO formulas [18]. Once the query is quantifier free, it is not too difficult to design for it a constant delay enumeration algorithm.

Another idea is to use the Gaifman Locality Theorem showing that for FO queries only the r-neighborhoods (i.e. substructures of all nodes at distance at most r) occurring in the structures are relevant, for a suitable value of r depending only on the query. In a class of graphs with bounded degree, there are only finitely many such r-neighborhoods and it

is possible to compute them in linear time, hence during the preprocessing phase. The enumeration algorithm follows [33].

The multiplicative factors are a tower of exponential whose height depends on $|q|$ in the case of [18] and are triply exponential in $|q|$ in the case of [33]. This latter multiplicative factor cannot be significantly improved: it follows from [26] that a multiplicative factor 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 [37]. In [37] a number of equivalent characterizations were given for bounded expansion 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.

The model checking problem for FO over classes of structures with bounded expansion can be solved in linear time [21, 30].

▶ Theorem 6. [34] The enumeration for FO over the class of structures with bounded expansion is in CD$\circ$Lin.

This result generalizes the bounded degree case. If structures in a class of bounded degree could be represented using finitely many unary bijections, structures in a class of bounded expansion can be represented using finitely many unary functions of a special kind. A quantifier elimination method is then given over such structures. However solving the quantifier-free case is not immediate.

The multiplicative factors are a tower of exponentials whose height is the quantifier alternation depth of the first-order query. This non-elementary multiplicative factor is unavoidable already on the class of unranked trees, assuming FPT $\neq$ AW[$*$] [26]. In comparison, recall that this factor is triply exponential in the size of the query over bounded degree structures.

4.3 Nowhere dense

It turns out that the notion of bounded expansion can be further generalized. A class $\mathcal{C}$ of graphs is *nowhere dense* if for all r there exists a graph H_r that is not a r-minor of all graphs of $\mathcal{C}$ (a r-minor is a minor where the collapsed balls have radius at most r).

This class was introduced in [38] with a number of equivalent characterizations giving evidence that it is a robust class. It contains all class of graphs of bounded expansion but also any class of graphs that locally excludes a minor or that has local bounded treewidth. We refer to [17, 25] for precise definitions of these notions.

It has recently been claimed that the model checking problem for FO over nowhere dense graphs can be done in quasi-linear time [31].

▶ Open problem 1. Can enumeration for FO over the class of nowhere dense graphs be done with constant delay after a quasi-linear time preprocessing?

If the class of graphs is closed under subgraphs, nowhere dense is the limit for the existence of FPT algorithms.

▶ Theorem 7. [22] If $\mathcal{C}$ is a somewhere dense class of graphs closed under subgraphs, then the model checking problem for FO over this class is W[1]-hard (actually existential formula suffices).

An even stronger result was obtained in [36] assuming that $\mathcal{C}$ is somewhere dense in an "effective way". In this case it is shown that the model-checking for FO is even AW[$*$]-complete.

4.4 Low Degree

For classes of graphs not closed under subgraphs, we can still obtain positive results over a somewhere dense class of graphs.

A class of graphs has low degree if for all δ, all but finitely many graphs in the class have degree at most n^δ, where n is the size of the graph. Typical examples are structures of bounded degree or of degree bounded by $\log n$.

It has been proved in [29] that over a class of structures of low degree, first-order boolean queries can be checked in pseudo-linear time, i.e. in time bounded by $O(n^{1+\epsilon})$, for all $\epsilon > 0$. This can be extended to an efficient enumeration algorithm assuming that sufficient memory is available. The result below assumes that the computation starts with an initial memory of $O(n^3)$ on input of size n. It will use only a small fragment of this memory, as it runs in pseudo-linear time, but for reasons detailed in [19], it requires initially more.

▶ Theorem 8. [19] The enumeration for FO over a class of structures of low degree can be done with constant delay after a pseudo-linear preprocessing time.

5 Structures with bounded treewidth

We have seen in Section 4.2 that structures of bounded treewidth are a special case of structures of bounded expansion. Therefore, over such classes, FO queries can be enumerated with constant delay after a linear time preprocessing. Over structures of bounded treewidth, the enumeration result can be extended to a larger class of queries: MSO queries. It is well known that the associated model checking problem can be solved in linear time by Courcelle's theorem [13].

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 [14, 2] but those cannot be enumerated in CD$\circ$Lin mainly because outputting one solution may require linear time. See Section 6.

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 [14], but it requires a precomputation phase of $O(n \log n)$ to build it.

▶ Theorem 9. [2, 35] The enumeration for MSO queries over the class of structures with bounded treewidth is in CD$\circ$Lin.

The difficulty of Theorem 9 lies entirely in the tree case. We present the key ingredients of the proof of [35] below as the intermediate results are 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.

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

▶ Proposition 10. [11] For any regular language L over an alphabet $\mathbb{A}$ and any $\mathbb{A}$-labeled tree $\mathbf{t}$ one can

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

The multiplicative factors resulting from the construction of the index and during the constant time tests depend on the presentation of L. They are non elementary if L is given as an MSO sentence. They are exponential if L is given as an automaton, even in the deterministic case (see also [6]). However, there exist cases where these multiplicative factors are polynomial (see for instance [7]).

The index structure built for proving Proposition 10 has the following interesting normal form for MSO queries over trees as a consequence.

▶ Proposition 11. [implicit in [11], see also [10]] 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, ancestor relationships, or **unary** MSO queries.

The index constructed in [35] for enumerating MSO queries over trees builds on Proposition 11. 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 11 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 using the ancestor relationship. Those queries being rather simple, an induction on the number of free variables solves the problem, see [35]. The multiplicative factors of Theorem 9 deviates from those of Proposition 10 only by a polynomial factor. Hence their size depends on the presentation of the MSO query as explained above.

6 Discussion

6.1 The impact of order

With the current definition of CD∘Lin, there is no constraint on 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.

In the presence of a linear order on the database, the enumeration algorithms of Theorem 5 (bounded degree) and Theorem 6 (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 9).

6.2 Longer delay

Delay linear in the size of the database

We could consider enumeration algorithms allowing for non constant delay. We have already seen logarithmic delays in Theorem 3 and Theorem 4. Another 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 12. [3]. The enumeration for 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 13. [4]. The enumeration for 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 9 can be generalized to this setting (the preprocessing phase of [14] is quasi-linear while it is linear in the case of [2]).

▶ Theorem 14. [14][2] The enumeration for 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].

[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].

6.3 Other enumeration problems

In this abstract we focused on the problem of enumerating the output of a query on a database. There exist also interesting enumeration algorithms for enumerating all the solutions of a SAT instance. For 2SAT, this is in CD∘Lin [23], for 3SAT dichotomy results exists [16, 15]. There exists also enumeration algorithms for various kinds of other problems like enumerating monomials of a polynomial [43], enumerating perfect matchings in bipartite graphs [44], independent sets [32] and so on. The interested reader is refereed to the thesis [42, 9] for learning more about enumerations outside of the database context.

7 Conclusions

We mentioned several results about constant delay enumeration. 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 semantics", i.e. each answer occurs as many times as there are valuations witnessing it.

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.

Acknowledgment: We thanks Johann Brault-Baron and Thomas Colcombet for useful comments on earlier versions of this paper.

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, 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.

4 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.

5 D. Berwanger and E. Grädel. Games and model checking for guarded logics. In *Intl. Conf. on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*, pages 70–84, 2001.

6 M. Bojańczyk. Factorization forests. In *Developments in Language Theory (DLT)*, 2009.

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. The factorisation forest theorem. To appear in the handbook "Automata: from Mathematics to Applications".

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

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

13 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.

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

15 N. Creignou and J.-J. Hébrard. On Generating All Solutions of Generalized Satisfiability Problems. *Informatique Théorique et Applications (ITA)*, 31(6):499–511, 1997.

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, N. Schweikardt, and L. Segoufin. Enumerating first-order queries over databases of low degree. submitted.

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 T. Feder. Network flow and 2-satisfiability. *Algorithmica*, 11:291–319, 1994. 10.1007/BF01240738.

24 J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2006.

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

26 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.

27 E. Grädel. On the restraining power of guards. *J. on Symbolic Logic*, 64(4):1719–1742, 1999.

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

29 M. Grohe. Generalized model-checking problems for first-order logic. In *Symp. on Theoretical Aspects in Computer Science (STACS)*, 2001.

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

31 M. Grohe, S. Kreutzer, and S. Siebertz. Deciding first-order properties of nowhere dense graphs. personal communication.

32 D. S. Johnson, C. H. Papadimitriou, and M. Yannakakis. On generating all maximal independent sets. *Inf. Process. Lett.*, 27(3):119–123, 1988.

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

34 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.

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

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

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

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

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

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

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

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

43 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.

44 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.

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