On acyclic conjunctive queries and constant delay enumeration

Guillaume Bagan[1], Arnaud Durand[2], and Etienne Grandjean[1]

[1] GREYC, Université de Caen, CNRS ; Campus 2, F-14032 Caen cedex, France.
gbagan@info.unicaen.fr & grandjean@info.unicaen.fr
[2] Equipe de Logique Mathématique - CNRS UMR 7056, Université Paris 7 - Denis Diderot, 2 place Jussieu, 75251 Paris Cedex 05, France - durand@logique.jussieu.fr

Abstract. We study the enumeration complexity of the natural extension of acyclic conjunctive queries with disequalities. In this language, a number of **NP**-complete problems can be expressed. We first improve a previous result of Papadimitriou and Yannakakis by proving that such queries can be computed in time $c.|\mathcal{M}|.|\varphi(\mathcal{M})|$ where $\mathcal{M}$ is the structure, $\varphi(\mathcal{M})$ is the result set of the query and c is a simple exponential in the size of the formula φ. A consequence of our method is that, in the general case, tuples of such queries can be enumerated with a linear delay between two tuples.
We then introduce a large subclass of acyclic formulas called $\mathbf{CCQ}^{\neq}$ and prove that the tuples of a $\mathbf{CCQ}^{\neq}$ query can be enumerated with a linear time precomputation and a constant delay between consecutive solutions. Moreover, under the hypothesis that the multiplication of two $n \times n$ boolean matrices cannot be done in time $O(n^2)$, this leads to the following dichotomy for acyclic queries: either such a query is in $\mathbf{CCQ}^{\neq}$ or it cannot be enumerated with linear precomputation and constant delay. Furthermore we prove that testing whether an acyclic formula is in $\mathbf{CCQ}^{\neq}$ can be performed in polynomial time.
Finally, the notion of free-connex treewidth of a structure is defined. We show that for each query of free-connex treewidth bounded by some constant k, enumeration of results can be done with $O(|\mathcal{M}|^{k+1})$ precomputation steps and constant delay.

Introduction

From a complexity theoretical point of view, very few is known about enumeration problems and their complexity classes. In the context of logical or database query languages, enumeration amounts to produce one by one all the tuples of the result $\varphi(\mathcal{M})$ of a query $\varphi(\bar{x})$ over a structure $\mathcal{M}$. Producing solutions of a query with a regular delay or on user demand makes sense in that area. Although a lot of results are known about the complexity of query evaluation in the classical sense (from hard cases to islands of tractability) [6, 9, 12, 14, 16], only some recent works have started to investigate complexity issues of enumeration for query languages [8, 10, 1, 11, 2].

A large and well-studied class of queries is the class **ACQ** of acyclic conjunctive queries. A classical result of database theory, proved in [17], is that any acyclic conjunctive query φ can be evaluated in time $O(|\varphi|.|\mathcal{M}|.|\varphi(\mathcal{M})|)$. In [16], an extension $\mathbf{ACQ}^{\neq}$ is introduced in which disequalities between variables are allowed. In full generality, deciding a boolean $\mathbf{ACQ}^{\neq}$ query is **NP**-complete for combined complexity. However, it is shown in [16] that such a query can be evaluated in time $f(|\varphi|).|\mathcal{M}|.|\varphi(\mathcal{M})|.\log^2|\mathcal{M}|$ where f is some exponential function. Many **NP**-complete problems can be expressed by $\mathbf{ACQ}^{\neq}$ formulas: existence of some path of a given length k in a graph, multidimensional matching, color matching, etc.

The main goal of this paper is to revisit the complexity of the evaluation of conjunctive queries with or without disequalities from a dynamical point of view. As in [8, 10, 1, 11, 2], queries are seen as enumeration problems and the complexity is measured in terms of delay between two consecutive solutions. We consider the class of **F-ACQ** (resp $\mathbf{F\text{-}ACQ}^{\neq}$) formulas which are the "equivalent" of acyclic conjunctive formulas in functional structures. First, we prove that an elimination of quantifiers can be done on such formulas. More precisely, using a combinatorial argument, we show that given an $\mathbf{F\text{-}ACQ}^{\neq}$ formula φ and a structure $\mathcal{M}$ one can construct a formula φ' which is a disjunction of quantifier-free formulas of $\mathbf{F\text{-}ACQ}^{\neq}$ and a model $\mathcal{M}'$ such that $\varphi(\mathcal{M}) = \varphi'(\mathcal{M}')$. As a consequence, we prove that any query of $\mathbf{F\text{-}ACQ}^{\neq}$ can be evaluated with a linear time delay. This immediately implies that a boolean $\mathbf{ACQ}^{\neq}$ query can be evaluated in time $f(|\varphi|).|\mathcal{M}|$ and a general $\mathbf{ACQ}^{\neq}$ query can be evaluated in time $f(|\varphi|).|\mathcal{M}|.|\varphi(\mathcal{M})|$ where f is a simple exponential function which depends on the number of variables and the number of disequalities of the formula.

Constant-Delay$_{lin}$ introduced in [10] is the class of enumeration problems which are solved by algorithms that run in two phases : the first phase performs a precomputation in linear time and the second phase, using the precomputation, enumerates all the solutions with a constant delay between two consecutive ones. This class is regarded as the minimal (nontrivial) robust class for the complexity of enumeration. A natural question addressed is this paper is the following: is there a large subclass of $\mathbf{ACQ}^{\neq}$ queries whose enumeration belongs to Constant-Delay$_{lin}$? We introduce the class $\mathbf{CCQ}^{\neq}$ of formulas called *free-connex acyclic* formulas and defined by a connectivity condition between the free variables (in particular, it contains the quantifier-free acyclic formulas with disequalities). We show that such queries can be evaluated with linear precomputation and constant delay. Furthermore, it is proved that testing whether a formula is free-connex acyclic can be performed in polynomial time. This result is optimal in some sense because we prove the following dichotomy theorem: under the assumption that any two $n \times n$ boolean matrices *cannot* be multiplied in time $O(n^2)$ (a very reasonable conjecture), each acyclic conjunctive query (with or without disequalities) which is *simple* [3] either is free-connex acyclic or cannot be enumerated with linear precomputation and constant delay. Finally, we intro-

[3] This means that we assume that each relation symbol occurs once only in the conjunction (a slight restriction)

duce the notions of *free-connex tree-decomposition* and *free-connex treewidth*. We show that for any fixed query of free-connex treewidth bounded by some constant k, enumeration of solutions can be done with $O(|D|^{k+1} + |\mathcal{M}|)$ precomputation steps and constant delay (D is the domain of $\mathcal{M}$).

Notice that essentially all the results of this paper are new, to our knowledge, even if we consider only conjunctive queries (without disequalities). The paper is organized as follows. In Section 1, the main definitions about complexity, queries, hypergraphs and enumeration problems are given. Our results about quantifier elimination and enumeration are proved respectively in Sections 2 and 3. The notion of free-connex acyclic formula and the dichotomy theorem are exposed in Section 4. Finally, in Section 5, results about free-connex treewidth queries are presented. Note that by lack of space, many proofs are omitted.

1 Preliminaries

1.1 Problems

In order to handle problems in a general and uniform framework, we introduce the notion of *general problem* which is merely a binary relation $A \subseteq I \times O$ over two sets I and O respectively called *space of instances* (or *input space*) and *space of solutions* (or *output space*). For each input $x \in I$, the set $A(x) = \{y \in O : (x, y) \in A\}$ is assumed to be finite and is called the *set of solutions* of A on x. Let $<$ be a strict linear order on the space of solutions of A. For any input x and $A(x) = \{y_0, \ldots, y_{n-1}\}$ with $y_0 < y_1 < \ldots < y_{n-1}$, we denote by $\text{enum}(A, x, <) = y_0 y_1, \ldots y_{n-1}$ the increasing list of elements of $A(x)$. We are interested in the following problems.

EVAL(A)
Input: An instance $x \in I$
Output: $A(x)$

ENUM($A, <$)
Input: an instance $x \in I$
Output: $\text{enum}(A, x, <)$

In the case the order does not matter, we omit it and write ENUM(A).

1.2 Complexity framework

The computation model used in this paper is the random access machine (RAM) with uniform cost measure and addition and substraction as the basic operations (see [13]). It has read-only input registers (containing the input x), read-write work memory registers and write-only output registers. Moreover, register values and memory addresses are assumed to be bounded by $c|x|$, for some constant c, where $|x|$ is the size of the input x (that is the number of registers it occupies).

An algorithm $\mathcal{A}$ is said to compute the problem ENUM($A, <$) if for any input x, $\mathcal{A}$ computes one by one the elements of the sequence $enum(A, x, <)$ and stops immediately after writing the last one. We define $\text{delay}_j(x) = \text{time}_j(x) - \text{time}_{j-1}(x)$ where $\text{time}_j(x)$ denotes the moment when $\mathcal{A}$ has completed the writing of the j^{th} solution (by convention $\text{time}_0(x) = 0$). We say that $\mathcal{A}$ is a *constant delay* algorithm if for a constant c and any input x, we have $\text{delay}_j(x) \leq c$ and

if furthermore, $\mathcal{A}$ uses space c (i.e. at most c memory registers) during its computation.

Definition 1. *A problem* $\textsc{Enum}(A, <)$ *is computable* with constant delay and linear precomputation*, which is written* $\textsc{Enum}(A, <) \in \textsc{Constant-Delay}_{lin}$ *if there is a function* $r : x \mapsto r(x)$ *(precomputation phase) computable in linear time and a constant delay algorithm* $\mathcal{A}$ *(enumeration phase) which, when applied to* $r(x)$ *for any input* x*, computes* $enum(A, x, <)$*.*

Note that the additional requirement that the enumeration phase uses constant space (uniform cost measure) appears to be a very strong condition. Nevertheless we have checked that it holds for all the constant delay algorithms we know (see [1, 10, 11, 2]). However, this condition is not essential in this paper. We need a very precise notion of reduction for enumeration problems.

Definition 2. *Let* $A \subseteq I_A \times O_A$ *and* $B \subseteq I_B \times O_B$ *be two general problems. An* exact reduction *from* $\textsc{Enum}(A)$ *to* $\textsc{Enum}(B)$ *(resp. from* $\textsc{Enum}(A, <_A)$ *to* $\textsc{Enum}(B, <_B)$*) is a pair* (r, t) *of mappings* $r : I_A \to I_B$ *and* $t : I_B \times O_B \to O_A$ *which satisfy conditions 1 and 2*

1. *For every instance* x *of* A*, the correspondence* $y \mapsto t(r(x), y)$ *is a one-one (resp. strictly increasing) mapping from the set (resp* $<_B$*-ordered set)* $B(r(x))$ *onto the set (resp.* $<_A$*-ordered set)* $A(x)$*.*
2. *The instance* $r(x)$ *of* B *is computable from* x *in time and space* $O(|x|)$ *and the solution* $t(r(x), y)$ *is computable from* $(r(x), y)$ *in constant time and constant space.*

The following results are straightforward consequences of our definitions.

Lemma 3. *Assume there exists an exact reduction from* $\textsc{Enum}(A)$ *to* $\textsc{Enum}(B)$*. Then* $\textsc{Enum}(B) \in \textsc{Constant-Delay}_{lin}$ *implies* $\textsc{Enum}(A) \in \textsc{Constant-Delay}_{lin}$*. The similar result holds for* $\textsc{Enum}(A, <_A)$ *and* $\textsc{Enum}(B, <_B)$*.*

Lemma 4. *Let* $A, B \subseteq I \times O$ *be two general problems over the same input and output spaces. Let* $<$ *be a linear order on* O*. If both problems* $\textsc{Enum}(A, <)$ *and* $\textsc{Enum}(B, <)$ *belong to* $\textsc{Constant-Delay}_{lin}$*, then so does the union problem* $\textsc{Enum}(A \cup B, <)$*.*

1.3 Logical framework

The reader is supposed familiar with first-order logic on finite structures (see [15]). For a signature σ, a (finite) σ-structure consists of a domain D together with an interpretation of each symbol of σ over D (we do not distinguish between a symbol and its interpretation). For each σ-formula (or σ-query) $\varphi(\bar{x})$ with m variables $\bar{x} = (x_1, \ldots, x_m)$ and for each σ-structure $\mathcal{M}$ of domain D, we denote by $\varphi(\mathcal{M})$ the set $\{\bar{a} \in D^m : \mathcal{M} \models \varphi(\bar{a})\}$. Let $<$ be a linear order on D^m. We are interested in the evaluation and enumeration problems associated to $\varphi(\bar{x})$. Note that in most of our problems formula φ is considered as *fixed*.

EVAL(φ)
Input: a σ-structure $\mathcal{M}$
Output: $\varphi(\mathcal{M})$

ENUM($\varphi, <$)
Input: a σ-structure $\mathcal{M}$
Output: an $<$-ordered enumeration of $\varphi(\mathcal{M})$

1.4 Hypergraphs

Let us introduce some definitions on hypergraphs. A hypergraph is a pair (V, E) where V is a set of elements, called *vertices*, and E is a set of non-empty subsets of V called *hyperedges*. The *induced subhypergraph* of $H = (V, E)$ on a set of vertices $S \subseteq V$ is the hypergraph $H[S] = (S, \{e \cap S \neq \emptyset : e \in E\})$. A hypergraph is *acyclic* if there is a tree T, called a *tree-structure* of H, whose vertices are the hyperedges of H and having the following *connectivity property*: for each vertex v of H, the set of hyperedges that contain v consists of a subtree (connected subgraph) of T. For two hypergraphs $H_1 = (V_1, E_1)$ and $H_2 = (V_2, E_2)$, let us denote by $H_1 \cup H_2$ the hypergraph $(V_1 \cup V_2, E_1 \cup E_2)$. Given a graph $G = (V, E)$, the neighborhood of x denoted by $N(x)$ is $\{y \in V : \{x, y\} \in E\}$. For more details on hypergraphs (in particular, on paths) see [3].

1.5 Acyclic conjunctive queries

As usual, let **CQ** (resp. $\mathbf{CQ}^{\neq}$) denote the set of *conjunctive queries* (resp *conjunctive queries with disequalities*), i.e. relational σ-formulas of the form $\varphi(\bar{x}) \equiv \exists \bar{y} \psi(\bar{x}, \bar{y})$ where ψ is a conjunction of σ-atoms $R(\bar{v})$ for $\bar{v} \subseteq \{\bar{x}, \bar{y}\}$ (resp σ-atoms and disequalities $u \neq v$ for $u, v \in \{\bar{x}, \bar{y}\}$). A conjunctive query (or conjunctive formula) φ (with or without disequalities) is *acyclic* or **ACQ** (resp. $\mathbf{ACQ}^{\neq}$) if its hypergraph $H_\varphi = (V_\varphi, E_\varphi)$ is acyclic (Recall the set of vertices and the set of hyperedges of H_φ are respectively $V_\varphi = var(\varphi)$ and $E_\varphi = \{\{v_1, \ldots, v_{p_i}\} : R_i(v_1, \ldots, v_{p_i})$ is a σ-atom of $\varphi\}$). A tree-structure of H_φ is said to be a *join-tree* of φ.

Acyclic conjunctive functional queries: An acyclic conjunctive σ-query can be naturally translated in the functional framework, i.e., into a σ'-formula for a signature σ' that consists of unary fuction symbols and unary predicates.

Formula translation: Introduce for each σ-atom $A_i, 1 \leq i \leq q$, of the formula a new variable α_i.

*Translation (1) of a quantifier-free **ACQ** σ-formula*: $\psi(\bar{x}) \equiv \bigwedge_{1 \leq i \leq q} A_i$. Associate to ψ the following quantifier-free functional σ'-formula:

$$\psi'(\alpha_1, \ldots, \alpha_q) \equiv \bigwedge_{1 \leq i \leq q} D_{R_i}(\alpha_i) \wedge \bigwedge_{(A_i, A_j) \in T_\psi} \bigwedge_{\mathrm{var}_h(A_i) = \mathrm{var}_k(A_j)} f_h(\alpha_i) = f_k(\alpha_j) \tag{1}$$

where $\sigma' = \{D_R : R \in \sigma\} \cup \{f_1, \ldots, f_p\}, p = \mathrm{arity}(\sigma) = \max_{R \in \sigma}(\mathrm{arity}(R))$, T_ψ is a join-tree of ψ, $A_i \equiv R_i(v_i^1, \ldots, v_i^h, \ldots, v_i^{p_i})$, p_i is the arity of R_i and $\mathrm{var}_h(A_i)$ denotes the variable v_i^h that occurs at rank h in atom A_i.

Translation (2) of any ***ACQ*** *formula* $\varphi(\bar{x}) \equiv \exists \bar{y} \psi(\bar{x}, \bar{y})$ where ψ is quantifier-free and $\bar{x} = (x_1, \ldots, x_m)$. Associate to φ the following functional σ'-formula:

$$\varphi''(\bar{x}) \equiv \exists \alpha_1 \ldots \exists \alpha_q (\psi'(\bar{\alpha}) \wedge \bigwedge_{1 \leq i \leq m} f_{j_i}(\alpha_{k_i}) = x_i) \tag{2}$$

where ψ' is the above translation (1) of the quantifier-free part ψ of φ and A_{k_i} (represented by variable α_{k_i}) is a chosen atom of φ in which x_i occurs (at rank j_i).

Translation of the structure: Each relational σ-model $\mathcal{M} = (D; R_1, \ldots, R_s)$ is translated into the functional σ'-model $\mathcal{M}' = (D'; D, D_{R_1}, \ldots, D_{R_s}, f_1, \ldots, f_p)$:

- $D, D_{R_1}, \ldots, D_{R_s}$ are disjoint unary relations so that $D' = D \cup D_{R_1} \cup \ldots \cup D_{R_s} \cup \{\perp\}$ and, for each $i = 1, \ldots, s$, we have $D_{R_i} = R_i$, i.e. D_{R_i} is the set of tuples of R_i, and $\perp$ is an extra element.
- Each f_j $(1 \leq j \leq p)$ is a unary function: $D' \to D \cup \{\perp\}$ such that, for every $t = (a_1, \ldots, a_{p_i}) \in D_{R_i}$, we have $f_j(t) = a_j$ for $1 \leq j \leq p_i$ and $f_j(t) = \perp$ otherwise.

Example 5. Translation 1 transforms the quantifier-free **ACQ** formula $\psi(x, y, z) \equiv R(x, y) \wedge S(y, z)$ into the following (quantifier-free) formula: $\psi'(\alpha_1, \alpha_2) \equiv D_R(\alpha_1) \wedge D_S(\alpha_2) \wedge f_2(\alpha_1) = f_1(\alpha_2)$.

Example 6. Translation 2 transforms the **ACQ** formula $\varphi(x, z) \equiv \exists y (R(x, y) \wedge S(y, z))$ into the following formula (with the same free variables and the subformula ψ' as above): $\varphi''(x, z) \equiv \exists \alpha_1 \exists \alpha_2 (\psi'(\alpha_1, \alpha_2) \wedge f_1(\alpha_1) = x \wedge f_2(\alpha_2) = z)$.

Remark 7. Our translations (1) and (2) are easily extended to $\mathbf{ACQ}^{\neq}$ formulas.

Let us now introduce the notions of conjunctive and acyclic conjunctive functional queries.

Definition 8. *Let* ***F-CQ*** *(resp.* ***F-CQ***$^{\neq}$*) denote the set of conjunctive functional σ-formulas (for a unary functional signature σ) of the form $\varphi = \exists \bar{x} \psi$ where ψ is a conjunction of unary atoms $U(v)$ and equalities $\tau = \tau'$ (resp. equalities $\tau = \tau'$ and disequalities $\tau \neq \tau'$), each τ, τ' is a functional term of the form $f(v)$ or v, where v is a variable.*

Definition 9. *To each* ***F-CQ*** *or* ***F-CQ***$^{\neq}$ *formula φ, we naturally associate the undirected graph $G_\varphi = (V_\varphi, E_\varphi)$ defined as follows:*

- *$V_\varphi = var(\varphi)$ and*
- *for any pair of distinct variables v, v', set $\{v, v'\} \in E_\varphi$ iff an equality involving both v and v' occurs as a conjunct* [4] *in φ.*

An ***F-CQ*** *(resp.* ***F-CQ***$^{\neq}$*) formula φ is* acyclic *or* ***F-ACQ*** *(resp.* ***F-ACQ***$^{\neq}$*) if its graph G_φ is acyclic.*

[4] Here again, the disequalities of φ are not involved in the definition of G_φ

Without loss of generality we assume that G_φ is also connex and hence is a tree T called a *join-tree* of φ . The following results are easy to prove.

Lemma 10. *1. Preservation of acyclicity: Let φ be a $\boldsymbol{CQ}^{\neq}$ formula. If φ is acyclic then its functional translations 1 (in the case φ is quantifier-free) and 2 are acyclic too.*

2. Linearity of the transformations:
- *For formulas: The sizes of the transformed formulas 1 and 2 are linear in the size of φ.*
- *For structures: Our transformation of a relational σ-structure $\mathcal{M}$ into a functional σ'-structure $\mathcal{M}'$ is computable in time $O(|\mathcal{M}|)$.*

3. Correctness of the reductions:
- *Translation 1: Let ψ' be the quantifier-free translation ($\boldsymbol{F\text{-}ACQ}^{\neq}$) of a (quantifier-free) formula ψ ($\boldsymbol{ACQ}^{\neq}$). There exists an exact reduction from problem* $\textsc{Enum}(\psi)$ *to* $\textsc{Enum}(\psi')$.
- *Translation 2: We have $\varphi(\mathcal{M}) = \varphi''(\mathcal{M}')$.*

2 Quantifier elimination

2.1 Covers and representative sets

In all the definitions and results of this section, E and F are two finite sets and $\bar{f} = (f_1, \ldots, f_k)$ is a tuple of k unary functions from E to F.

Definition 11. *A* cover $\bar{c}$ *of $(E, \bar{f})$ is a tuple $(c_1, \ldots, c_k) \in F^k$ such that $\forall y \in E, \exists i : c_i = f_i(y)$. We denote by* $\textsc{covers}(E, \bar{f})$ *the set of covers of $(E, \bar{f})$.*

Definition 12. *A* representative set *of $(E, \bar{f})$ is a subset $E' \subseteq E$ such that* $\textsc{covers}(E, \bar{f}) = \textsc{covers}(E', \bar{f})$.

Lemma 13. *There is a representative set E' of $(E, \bar{f})$ of cardinality at most $O(k!)$. Such a set E' is called* a small representative set *of $(E, \bar{f})$ and can be computed in time $O(k!|E|)$.*

2.2 Quantifier elimination

Lemma 14. *Let $\varphi(\bar{x})$ be an $\boldsymbol{F\text{-}ACQ}^{\neq}$ σ-formula with a join-tree T and of the form $\varphi(\bar{x}) = \exists z \psi(\bar{x}, z)$ where ψ is quantifier-free and z is a leaf of T. Let k be the number of disequalities in $\varphi(\bar{x})$ involving variable z and let $\mathcal{M}$ be a σ-structure. Then we can compute in time $O(k!.|\varphi|.|\mathcal{M}|)$ a quantifier-free σ'-formula $\varphi'(\bar{x})$ with $\sigma \subseteq \sigma'$ and a σ'-structure $\mathcal{M}'$ that expands $\mathcal{M}$ such that*

- *$\varphi(\mathcal{M}) = \varphi'(\mathcal{M}')$;*
- *$\varphi'(\bar{x})$ is a disjunction of at most $O(k!)$ formulas ψ_i of $\boldsymbol{F\text{-}ACQ}^{\neq}$;*
- *for each disjunct ψ_i of φ', $|\psi_i| \leq |\varphi|$.*

Proof. Formula $\varphi(\bar{x})$ can be put under the form:

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

where $\psi_0(\bar{x})$ is a quantifier-free formula, $y \in \bar{x}$ is the parent of the leaf z in the tree T, $f'_i(\bar{x})$ denotes a term $f'_i(v_i)$ for $v_i \in \bar{x}$ and $P(z)$ is a quantifier-free formula on z. Let $\bar{g}(x) = (g_1(x), \ldots g_m(x))$, $\bar{g'}(x) = (g'_1(x), \ldots g'_m(x))$. One partitions the set $P = P(\mathcal{M})$ according to the values of $\bar{g}$; more precisely, for each $\bar{\alpha} \in \bar{g}(D)$, one defines the set $P_{\bar{\alpha}} = P \cap \bar{g}^{-1}(\bar{\alpha})$ and computes a small representative set of $(P_{\bar{\alpha}}, \bar{f})$ denoted $P'_{\bar{\alpha}}$ (of cardinality at most $h = O(k!)$). This can be done globally in time $\Sigma_{\bar{\alpha}\in\bar{g}(D)} O(k!|P_{\bar{\alpha}}|) = O(k!|P|) = O(k!|\mathcal{M}|)$. Clearly, $\varphi(\bar{x})$ can be rephrased as

$$\psi_0(\bar{x}) \wedge \exists z \in P_{\bar{g'}(y)} \bigwedge_{1\leq i\leq k} f_i(z) \neq f'_i(\bar{x}).$$

By definition of the representative sets, it is also equivalent to the following formula on $\mathcal{M}$:

$$\psi_0(\bar{x}) \wedge \exists z \in P'_{\bar{g'}(y)} \bigwedge_{1\leq i\leq k} f_i(z) \neq f'_i(\bar{x}).$$

Let $\mathcal{M}'$ be the σ'-expansion of $\mathcal{M}$ obtained by introducing the unary predicates E_j and the unary functions v_j, for $1 \leq j \leq h$, defined as follows: $y \in E_j \Leftrightarrow |P'_{\bar{g'}(y)}| \geq j$; $v_j(y)$ is the j^{th} element of $P'_{\bar{g'}(y)}$ if $|P'_{\bar{g'}(y)}| \geq j$ and an arbitrary value otherwise. Finally, we obtain $\varphi(\mathcal{M}) = \varphi'(\mathcal{M}')$ for the following σ'-formula φ' of the required form:

$$\varphi'(\bar{x}) \equiv \bigvee_{1\leq j\leq h} (\psi_0(\bar{x}) \wedge E_j(y) \wedge \bigwedge_{1\leq i\leq k} f_i(v_j(y)) \neq f'_i(\bar{x}))$$

Using the result above, one can eliminate one by one all the variables but one in a bottom-up approach and derive the two following results. The iterative treatment of variables is similar in spirit to Yannakakis' algorithm [17].

Lemma 15. *Let $\varphi(x)$ be an **F-ACQ**$^{\neq}$ σ-formula with only one free variable x and $\mathcal{M}$ be a σ-structure. Let l be the number of variables and k be the number of disequalities of $\varphi(x)$. One can compute in time $O(k!^l.|\varphi|.|\mathcal{M}|)$ a σ'-expansion $\mathcal{M}'$ of $\mathcal{M}$ ($\sigma \subseteq \sigma'$) and a σ'-formula φ' which is a disjunction of $O(k!^l)$ **F-ACQ**$^{\neq}$ quantifier-free formulas ψ_i where $|\psi_i| \leq |\psi|$ so that $\varphi(\mathcal{M}) = \varphi'(\mathcal{M}')$.*

Theorem 16. *Let φ be a fixed **F-ACQ**$^{\neq}$ (resp. **ACQ**$^{\neq}$) query. Then* Enum(φ) *can be evaluated with delay $O(|\mathcal{M}|)$.*

In the particular case of boolean queries then, surprisingly, the exponent due to the number of variables l can be avoided.

Proposition 17. *Let φ be an **F-ACQ**$^{\neq}$ σ-formula without free variables and $\mathcal{M}$ be a σ-structure. Testing whether $\mathcal{M} \models \varphi$ can be decided in time $O(k!.|\varphi|.|\mathcal{M}|)$ where k is the number of disequalities of φ*

3 Constant delay enumeration

3.1 Combinatorial tools

In this section, we will need a convenient notion of *two-phase algorithm* for a problem with two inputs : a *static input* x and a *dynamic input* y. It is defined by the following general scheme.

- *Static phase (or precomputation phase)*: it computes some data called $r(x)$ (only depending on x);
- *Dynamic phase*: from the input $(r(x), y)$, it computes the output of the problem.

LeastNonCover
Static input : Two finite ordered sets E and F,
a k-tuple of unary functions $\bar{f} = (f_1, \dots f_k)$ from E to F
Dynamic input : a vector $\bar{u} \in F^k$, an element $x \in E$
Parameter: k
Output: the least element $y \in E$ such that $y \geq x \wedge \bigwedge_{1 \leq i \leq k} f_i(y) \neq u_i$ if such an element exists, and $\perp$ otherwise.

Lemma 18. LeastNonCover *is computable with a static phase of linear time and a dynamic phase of constant time.*

Proof. (sketch) In this proof, E, F and $\bar{f}$ are considered as implicit and we assume that $E = \{1, \dots, |E|\}$. Let x be an element of E, $\bar{u}$ be a k-tuple of elements of F and S be a subset of $[1, k]$. Then define $\mathrm{Find}(x, \bar{u}, S)$ as the least element $y \in E$ if it exists ($\perp$ otherwise) such that $y \geq x \wedge \bigwedge_{i \in S} f_i(y) \neq u_i$. For each list $L = (a_1, \dots, a_l)$ of distinct elements of $[1, k]$, $0 \leq l \leq k$, we define the "pointer function" $\mathrm{Ptr}_L : E \to E \cup \{\perp\}$ useful in the computation of Find as follows

- $\mathrm{Ptr}_\epsilon(x) = x$;
- if $l \geq 1$ then $\mathrm{Ptr}_{a_1,\dots,a_l}(x)$ is the least element $y \in E$ such that $y \geq x \wedge \forall j, 1 \leq j \leq l : f_{a_j}(y) \neq f_{a_j}(\mathrm{Ptr}_{a_1,\dots a_{j-1}}(x))$ if it exists and $\mathrm{Ptr}_{a_1,\dots a_{l-1}}(x) \neq \perp$, and is $\perp$ otherwise.

We assume that during the precomputation phase, we have computed the table PTR where $\mathrm{PTR}_L[x] = \mathrm{Ptr}_L(x)$ for each element $x \in E$ and each list L. We give an algorithm $\mathrm{FIND}(x, \bar{u}, S)$ which computes $\mathrm{Find}(x, \bar{u}, S)$. Clearly, the dynamic phase of LeastNonCover consists in calling FIND with $S = [1, k]$. $\mathrm{FIND}(x, \bar{u}, S)$ consists of the call $\mathrm{FIND\text{-}aux}(x, \bar{u}, S, \epsilon)$.

Finally, Algorithm 2 is given which computes the PTR table for the precomputation phase.

The more elaborated problem below need to be defined.

Algorithm 1 FIND-aux$(x, \bar{u}, S, L)$

$y \leftarrow \mathrm{PTR}_L[x]$
if $y = \perp \vee \forall i \in S - L : f_i(y) \neq u_i$ **then**
  return y
else
  choose $i \in S - L$ such that $f_i(y) = u_i$
  return FIND-aux$(x, \bar{u}, S, L.i)$
end if

Algorithm 2 Precomputation phase

for x from $|E|$ to 1 **do**
  $\mathrm{PTR}_\epsilon[x] \leftarrow x$
  for l from 1 to k **do**
    for each list $L = (a_1, \ldots, a_l)$ of l distinct elements of $[1, k]$ **do**
      if $x = |E|$ or $\mathrm{PTR}_{a_1,\ldots,a_{l-1}}[x] = \perp$ **then**
        $\mathrm{PTR}_L[x] \leftarrow \perp$
      else
        let $\bar{u} = (u_1, \ldots, u_k)$ such that $\forall j \leq l : u_{a_j} = f_{a_j}(\mathrm{PTR}_{a_1,\ldots,a_{j-1}}[x])$
        $\mathrm{PTR}_L[x] \leftarrow \mathrm{FIND}(x + 1, \bar{u}, \{a_1, \ldots, a_l\})$
      end if
    end for
  end for
end for

EnumNonCover
Static input: Two finite ordered set E and F
a k-tuple of unary functions $\bar{f} = (f_1, \ldots f_k)$ from E to F
Dynamic input: a vector $\bar{u} \in F^k$
Parameter: k
Output: the set $\{x \in E : \bigwedge_{1 \leq i \leq k} f_i(x) \neq u_i\}$ in increasing order

The following lemma is a consequence of Lemma 18

Lemma 19. EnumNonCover *is computable with a static phase in linear time and a dynamic phase with a constant delay.*

3.2 Enumeration of quantifier-free and free-connex acyclic queries

Definition 20. *Let φ be a quantifier-free* ***F-ACQ***$^{\neq}$ *formula and T a join-tree of φ. An elimination order (or T-order) of the variables of φ is an ordered list (may be empty) $x_1, \ldots, x_p$ of its p variables such that (if $p > 0$) x_p is a leaf of T and $x_1, \ldots, x_{p-1}$ is an elimination order of the tree $T - \{x_p\}$.*

In the following, we implicity assume that the variables of any quantifier-free **F-ACQ**$^{\neq}$ formula are numbered in some fixed T-order.

Theorem 21. *Let $\varphi(x_1, \ldots, x_p)$ be a quantifier-free σ-formula in* ***F-ACQ***$^{\neq}$*. Then we have* $\mathrm{Enum}(\varphi, <_{lex}) \in \text{Constant-Delay}_{lin}$ *for the lexicographical order $<_{lex}$.*

Proof. The theorem is proved by induction on the number of variables of φ. Without loss of generality, assume that φ is of the form $\varphi(\bar{x}, y) \equiv \varphi_0(\bar{x}) \wedge \Theta(\bar{x}, y)$ where

$$\Theta(\bar{x}, y) \equiv P(y) \wedge \bigwedge_{1 \leq i \leq m} f_i(y) = g_i(z) \wedge \bigwedge_{1 \leq j \leq k} h_j(y) \neq h'_j(\bar{x}).$$

In this formula, the variables y and $z \in \bar{x} = (x_1, \ldots, x_{p-1})$ are respectively the leaf x_p and its parent in the join-tree T of φ and $h'_j(\bar{x})$ denotes a term $h'_j(v)$ for some variable $v \in \bar{x}$. Trivially, φ is logically equivalent to the following formula $\varphi_1(\bar{x}, y) \equiv \psi(\bar{x}) \wedge \Theta(\bar{x}, y)$ where $\psi(\bar{x}) \equiv \exists y \varphi(\bar{x}, y)$. By Lemma 14, one can compute in linear time, for each σ-structure $\mathcal{M}$, a new σ'-structure $\mathcal{M}'$ (σ'-expansion of $\mathcal{M}$, $\sigma \subseteq \sigma'$) and a quantifier-free disjunction $\psi'(\bar{x}) \equiv \bigvee_{1 \leq i \leq N} \psi_i(\bar{x})$ of $\mathbf{ACQ}^{\neq}$ σ'-formulas ψ_i (each of join-tree $T - \{y\}$) so that we have $\psi(\mathcal{M}) = \psi'(\mathcal{M}')$. This yields $\varphi(\mathcal{M}) = \varphi'(\mathcal{M}')$ for the following quantifier-free formula: $\varphi'(\bar{x}, y) \equiv \psi'(\bar{x}) \wedge \Theta(\bar{x}, y)$. φ' is equivalent to the disjunction $\bigvee_{1 \leq i \leq N} \varphi_i(\bar{x}, y)$ where φ_i is the formula

$$\varphi_i(\bar{x}, y) \equiv \psi_i(\bar{x}) \wedge P(y) \wedge \bar{f}(y) = \bar{g}(z) \wedge \bigwedge_{1 \leq j \leq k} h_j(y) \neq h'_j(\bar{x})$$

By the induction hypothesis, we have for each $i \leq N$, $\textsc{Enum}(\psi_i, <_{lex}) \in \textsc{Constant-Delay}_{lin}$. By Lemma 4, it is sufficient to prove the same complexity result for $\textsc{Enum}(\varphi_i, <_{lex}), 1 \leq i \leq N$. (Notice the essential fact that the linear order $<_{lex}$ is the same for each i.) Here is the required algorithm for $\textsc{Enum}(\varphi_i, <_{lex})$.

Input: A σ'-structure $\mathcal{M}'$

Precomputation phase

- Perform the precomputation phase of $\textsc{Enum}(\psi_i, <_{lex})$
- Set $P = \{y \in D : \mathcal{M}' \models P(y)\}$
- By sorting the elements $y \in P$ according to the values $\bar{f}(y)$, compute a partition of set P into non empty sets $P_{\bar{\alpha}} = \{y \in P : \bar{f}(y) = \alpha\}$
- Compute the set $A = \{\bar{\alpha} \in D^m : P_{\bar{\alpha}} \neq \emptyset\}$
- For each $\bar{\alpha} \in A$, execute the precomputation phase of the algorithm Enum-NonCover on the (static) input $E = P_{\bar{\alpha}}$ and $(h_j)_{j \leq k}$.

Enumeration phase

For each $\bar{a}$ enumerated by $\textsc{Enum}(\psi_i, <_{lex})$ do

- Let $\bar{\alpha} = \bar{g}(c)$ where c is the value of variable z in the tuple $\bar{a}$ (if z is the variable x_l, then $c = a_l$)
- Let $\bar{u} = (h'_j(\bar{a}))_{j \leq k}$
- For each b enumerated by $\textsc{EnumNonCover}(P_{\bar{\alpha}}, (h_j)_{j \leq k}, \bar{u})$, output $(\bar{a}, b)$

Definition 22. *A query of* ***F-ACQ*** *(resp* ***F-ACQ***$^{\neq}$*) is* free-connex acyclic *if φ is acyclic and the set of free variables of φ is a connex subset of the join-tree of φ. We denote by* ***F-CCQ*** *(resp* ***F-CCQ***$^{\neq}$*) the class of free-connex acyclic queries of* ***F-ACQ*** *(resp* ***F-ACQ***$^{\neq}$*).*

Since free variables form a connex part of the tree decomposition, a repeated use of Lemma 14 permits to eliminate one by one all the quantified variables until only the free variables remain. Then, one applies Theorem 21 to obtain the following result.

Theorem 23. *For any φ of* ***F-CCQ***$^{\neq}$, $\textsc{Enum}(\varphi) \in \textsc{Constant-Delay}_{lin}$.

4 A dichotomy result

Definition 24. *Let H be a hypergraph, we say that H' is a P-extension of H if*

- $V(H) = V(H')$,
- $E(H) \subseteq E(H')$,
- $\forall e' \in E(H')\ \exists e \in E(H)\ e' \subseteq e$.

Definition 25. *Let $H = (V, E)$ be a hypergraph and let $S \subseteq V$. We say (T, A) is an S-*connex tree-structure *of H if T is a tree-structure of H, and A is a connex subset of the set of vertices of T such that $\bigcup_{e \in A} e = S$.*
*- H is S-*connex acyclic *if there is an S-connex tree-structure (T, A) of H.*
- H is e-S-connex acyclic *if there is a P-extension H' of H such that H' is S-connex acyclic.*
- A conjunctive formula φ is free-connex acyclic *if the hypergraph associated to φ is S-connex acyclic where S is the set free(φ) of free variables of φ.*
We denote by ***CCQ*** *(resp* ***CCQ***$^{\neq}$*) the class of free-connex acyclic queries φ of* ***CQ*** *(resp* ***CQ***$^{\neq}$*).*

Lemma 26. *Let φ be a formula of* ***CCQ***$^{\neq}$*. Then there exists a formula φ' of* ***F-CCQ***$^{\neq}$ *and an exact reduction from problem* $\textsc{Enum}(\varphi)$ *to* $\textsc{Enum}(\varphi')$

This immediately yields the following Theorem

Theorem 27. *Let φ be a query of* ***CCQ***$^{\neq}$*. Then* $\textsc{Enum}(\varphi) \in \textsc{Constant-Delay}_{lin}$.

We give now another characterization of S-connex acyclic hypergraphs. We need to introduce the notion of S-path.

Definition 28. *Let $H = (V, E)$ be a hypergraph and let $S \subseteq V$. An S-path is a path $(x, a_1, \ldots, a_k, y)$ such that*

- *x and y belong to S,*
- *vertices $a_i, 1 \leq i \leq k$, belong to $V - S$, and*
- *no hyperedge includes both x and y.*

Lemma 29. *Let $H = (V, E)$ be a hypergraph. H is e-S-connex acyclic if and only if H is acyclic and H admits no S-path.*

The previous lemma permits to obtain a polynomial (probably linear) algorithm to recognize **CCQ** formulas.

Lemma 30. *There is a polynomial time algorithm which, given an acyclic hypergraph $H = (V, E)$ and a set $S \subseteq V$ returns a P-extension H' of H and an S-connex tree-structure of H' if H is e-S-connex acyclic, and an S-path of H otherwise.*

We aim to show that the evaluation and enumeration problems of any acyclic query φ which is not **CCQ** are "hard" in some precise sense. For that purpose, we want to exhibit an exact reduction from the multiplication problem of $n \times n$ boolean matrices (a problem that is strongly conjectured to be not computable in $O(n^2)$ time) to $\textsc{Enum}(\varphi)$. We need to encode our matrix problem in the first-order framework. A Two-Matrix structure is a relational σ_{AB}-structure $\mathcal{M} = (D, A, B)$ where $D = [1, n]$, $\sigma_{AB} = \{A, B\}$, and A, B are binary relations. Clearly, the multiplication problem of two $n \times n$ boolean matrices is expressed by the following acyclic σ_{AB}-query: $\Pi(x, y) \equiv \exists z(A(x, z) \wedge B(z, y))$. More precisely, $\textsc{Enum}(\Pi)$ is the problem of enumerating, for each Two-Matrix structure $\mathcal{M} = ([1, n], A, B)$ given as input, the ordered pairs of indices (i, j) where 1 occurs in the matrix product $C = A \times B$, i.e., $C_{i,j} = 1, 1 \leq i, j \leq n$.

Definition 31. *A conjunctive formula is* simple *if each relation symbol appears in at most one atom.*

Remark 32. Given a formula φ of **ACQ** (resp **CCQ**, $\mathbf{ACQ}^{\neq}$, $\mathbf{CCQ}^{\neq}$) and a structure $\mathcal{M}$, one can compute in linear time a simple formula φ' of **ACQ** (resp **CCQ**, $\mathbf{ACQ}^{\neq}$, $\mathbf{CCQ}^{\neq}$) and a structure $\mathcal{M}'$ such that $\varphi(\mathcal{M}) = \varphi'(\mathcal{M}')$.

Lemma 33. *Let φ be a simple formula which is **ACQ** (resp $\mathbf{ACQ}^{\neq}$) but not **CCQ** (resp not $\mathbf{CCQ}^{\neq}$). Then there exists an exact reduction from the problem $\textsc{Enum}(\Pi)$ to $\textsc{Enum}(\varphi)$.*

Finally, one obtains the following result

Theorem 34. *(Dichotomy Theorem) Let φ be a simple **ACQ** (resp $\mathbf{ACQ}^{\neq}$) query. We have either (1) or (2):*

1. *$\varphi \in$ **CCQ** (resp $\mathbf{CCQ}^{\neq}$) and $\textsc{Enum}(\varphi) \in \textsc{Constant-Delay}_{lin}$ (and $\textsc{Eval}(\varphi)$ is computable in time $O(|\mathcal{M}| + |\varphi(\mathcal{M})|)$)*
2. *There is an exact reduction from $\textsc{Enum}(\Pi)$ to $\textsc{Enum}(\varphi)$ and then $\textsc{Enum}(\varphi) \notin \textsc{Constant-Delay}_{lin}$ (and $\textsc{Eval}(\varphi)$ is not computable in time $O(|\mathcal{M}| + |\varphi(\mathcal{M})|)$) under the hypothesis that the product of two $n \times n$ matrices cannot be computed in time $O(n^2)$.*

5 Free-connex treewidth

In this section, we introduce the notion of free-connex tree-decomposition of conjunctive queries: this is a variant of the tree-decomposition of graphs

Definition 35. *(see [4] for details) A* tree-decomposition *of a graph $G = (V, E)$ is a tree T whose vertices are subsets of V and are called the* bags *of T, so that*

- *T has the connectivity property (see above),*
- *each edge of G is included in some bag of T.*

The width *of a tree-decomposition T is $\max\{|B| : B \text{ is a bag of } T\} - 1$. The* treewidth *of G denoted by $tw(G)$ is the smallest width of any tree-decomposition of G.*

Definition 36. *Let $G = (V, E)$ be a graph and S be a subset of V. A S-*connex tree-decomposition *of G is a tuple (T, A) where T is a tree-decomposition of G and A is a connected subset of $V(T)$ such that $\bigcup_{B \in A} B = S$.*
*Given a graph $G = (V, E)$ and a set $S \subseteq V$, the S-*connex treewidth *of G denoted by $ctw(G, S)$ is the smallest width of any S-connex tree-decomposition of G.*
Let φ be a formula of $\boldsymbol{CQ}$ (resp $\boldsymbol{CQ}^{\neq}$). The free-connex treewidth *of φ is the S-connex treewidth of the Gaifman graph [5] of φ for $S = free(\varphi)$.*

Lemma 37. *Let φ be a $\boldsymbol{CQ}$ (resp $\boldsymbol{CQ}^{\neq}$) formula of free-connex treewidth k over a structure $\mathcal{M}$ of domain D. Then we can compute in time $O(|D|^{k+1} + |\mathcal{M}|)$ a formula of $\boldsymbol{CCQ}$ (resp $\boldsymbol{CCQ}^{\neq}$) φ' (of size only depending on $|\varphi|$) and a model $\mathcal{M}'$ for φ' such that $\varphi(\mathcal{M}) = \varphi'(\mathcal{M}')$.*

By the previous Lemma and Theorem 27, we obtain the following theorem

Theorem 38. *Given a fixed formula φ of $\boldsymbol{CQ}^{\neq}$ of free-connex treewidth k, the problem* Enum(φ) *can be computed with $O(|D|^{k+1} + |\mathcal{M}|)$ precomputation and constant delay.*

We now give a polynomial algorithm to compute the free-connex treewidth of a formula. We need to introduce the notion of completed graph.

Definition 39. *Let G be a graph. The* completed graph *G' of (G, S) is built as follows*

- *$V(G') = V(G)$*
- *$E(G') = E(G) \cup \{\{x, y\} : x, y \in V$ and there is an S-path between x and $y\}$*

Theorem 40. *Let G be a graph, let $S \subseteq V(G)$ and let G' be the completed graph of (G, S). It holds*

1. *$ctw(G, S) = tw(G')$;*
2. *For any fixed k, there is a polynomial time algorithm which returns an S-connex tree-decomposition of G of width at most k if $ctw(G, S) \leq k$ and returns False otherwise.*

Notice that the algorithm of point (2) uses the k-tree decomposition procedure of [5] applied to the completed graph G'.

[5] The Gaifman graph of φ is the graph $G_\varphi = (V, E)$ where $V = \text{var}(\varphi)$ and E is the set of pairs of variables $\{x, y\}$ such that there exists an atom of φ that contains both x and y.

References

1. Guillaume Bagan. MSO queries on tree decomposable structures are computable with linear delay. In *Proc. of the Annual Conference of the European Association for Computer Science Logic (CSL)*, pages 167–181, 2006.
2. Guillaume Bagan, Arnaud Durand, Etienne Grandjean, and Frédéric Olive. Computing the jth element of a first-order query, 2007. submitted.
3. C. Berge. *Graphs and hypergraphs*. Amsterdam, 2nd edition, 1973.
4. Hans L. Bodlaender. A tourist guide through treewidth. *Acta Cybernetica*, 11:1–21, 1993.
5. Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996.
6. Chandra Chekuri and Anand Rajaraman. Conjunctive query containment revisited. *Theor. Comput. Sci.*, 239(2):211–229, 2000.
7. Henry Cohn, Robert D. Kleinberg, Balázs Szegedy, and Christopher Umans. Group-theoretic algorithms for matrix multiplication. In *FOCS*, pages 379–388, 2005.
8. Bruno Courcelle. Linear delay enumeration and monadic second-order logic. *Discrete Applied Maths*, 2006. To appear.
9. Bruno Courcelle and Mohamed Mosbah. Monadic second-order evaluations on tree-decomposable graphs. *Theor. Comput. Sci.*, 109(1&2):49–82, 1993.
10. Arnaud Durand and Etienne Grandjean. First-order queries on structures of bounded degree are computable with constant delay. *Transactions on Computational Logic*, To appear.
11. Arnaud Durand and Frédéric Olive. First-order queries over one unary function. In *Proc. of the Annual Conference of the European Association for Computer Science Logic (CSL)*, pages 334–348, 2006.
12. Jörg Flum, Markus Frick, and Martin Grohe. Query evaluation via tree-decompositions. *J. ACM*, 49(6):716–752, 2002.
13. Etienne Grandjean and Thomas Schwentick. Machine-independent characterizations and complete problems for deterministic linear time. *SIAM J. Comput.*, 32(1):196–230, 2002.
14. Martin Grohe, Thomas Schwentick, and Luc Segoufin. When is the evaluation of conjunctive queries tractable? In *STOC*, pages 657–666, 2001.
15. Leonid Libkin. *Elements of finite model theory*. Springer, 2004.
16. Christos H. Papadimitriou and Mihalis Yannakakis. On the complexity of database queries. *J. Comput. Syst. Sci.*, 58(3):407–427, 1999.
17. M. Yannakakis. Algorithms for acyclic database schemes. pages 82–94, 1981.