Enumerating Answers to First-Order Queries over Databases of Low Degree

Arnaud Durand, IMJ-PRG, Université de Paris, CNRS
Nicole Schweikardt
Luc Segoufin, INRIA, Laboratoire Cogitamus

Abstract
A class of relational databases has low degree if for all $\delta > 0$, all but finitely many databases in the class have degree at most $n^\delta$, where $n$ is the size of the database. Typical examples are databases of bounded degree or of degree bounded by $\log n$.

It is known that over a class of databases having low degree, first-order boolean queries can be checked in pseudo-linear time, i.e. for all $\varepsilon > 0$ in time bounded by $n^{1+\varepsilon}$. We generalize this result by considering query evaluation.

We show that counting the number of answers to a query can be done in pseudo-linear time and that after a pseudo-linear time preprocessing we can test in constant time whether a given tuple is a solution to a query or enumerate the answers to a query with constant delay.

1 Introduction

Query evaluation is a fundamental task in databases and a vast literature is devoted to the complexity of this problem. However, for more demanding tasks such as producing the whole set of answers or computing aggregates on the query result (such as counting the number of answers), complexity bounds are often simply extrapolated from those for query evaluation; and until recently, few specific methods and tools had been developed to tackle these problems. Given a database $A$ and a first-order query $q$, it may be not satisfactory enough to express complexity results in terms of the sizes of $A$ and $q$ as it is often the case. The fact that the solution set $q(A)$ 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. 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 $A$). Surprisingly, a number of query evaluation problems have been shown to admit constant delay algorithms, usually preceded by a preprocessing phase that is linear or almost linear. This is the case when queries are evaluated over the class of structures of bounded degree [5, 14], over the class of structures of “bounded expansion” [16] and, more generally, over the class of nowhere dense structures [19]. Similar results have been shown for monadic second-order logic over structures of bounded tree-width [4, 11, 15] or for fragments of first-order logic over arbitrary structures [2, 3].
However, as shown in [2], 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 [22]).

In this paper, we investigate the complexity of the enumeration, counting, and testing problems for first-order queries over classes of low degree. A class of relational databases has low degree if for all $\delta > 0$, all sufficiently large databases in the class have degree at most $n^\delta$, where $n$ is the size of the database. Databases of bounded degree or of degree bounded by $(\log n)^c$, for any fixed constant $c$, are examples of low degree classes. However, it turns out to be incomparable with the class of nowhere dense databases mentioned above.

It has been proved in [12] that over a class of databases of low degree, first-order boolean queries can be checked in pseudo-linear time, i.e., for all $\epsilon > 0$ there is an algorithm running in time bounded by $O(n^{1+\epsilon})$ checking the given first-order query. In this paper, we prove that counting the number of answers to a query can be done in pseudo-linear time, and that enumerating the answers to a query can be done with constant delay after a pseudo-linear time preprocessing. We also prove that testing membership of a tuple to a query result can be done in constant time after a pseudo-linear time preprocessing. We adopt a uniform approach to prove all these results by using at the heart of the preprocessing phases a quantifier elimination method that reduces our different problems to their analog but for colored graphs and quantifier-free queries. With such a tool, we can then focus within each specific task on very simple instances.

Over a class of databases of low degree, the difficulty is to handle queries requiring that in all its answers, some of its components are far away from each other. When this is not the case, for instance when in all answers all its components are within short distance from the first component, then the low degree assumption implies that there are only few answers in total and those can be computed in pseudo-linear time. In the difficult case, the number of answers may be exponential in the arity of the query and the naive evaluation algorithm may spend too much time processing tuples with components close to each other. To avoid this situation, we introduce suitable functions that can be precomputed in pseudo-linear time, and that allow us to jump in constant time from a tuple with components close to each other to a correct answer.

Related work. Enumerating the answers to a boolean query $q$ over a database $A$ is more general than testing whether $q$ holds on $A$, a problem also known as the model checking problem. An enumeration algorithm with constant delay after a preprocessing phase taking pseudo-linear time, or even polynomial time, induces a model checking algorithm that is fixed-parameter tractable (FPT), i.e., works in time $f(q)\cdot||A||^c$ for some constant $c$ and some function $f$ depending only on the class of databases. There is a vast literature studying the model checking problem for first-order logic aiming at finding FPT algorithms for larger and larger classes of databases. Starting from the class of databases of bounded degree, or bounded treewidth, FPT algorithms were derived for databases having bounded expansion [7] (see also [16]). Actually, recently an FPT algorithm has been obtained for a class of databases known as “nowhere dense”, generalizing all the previously known results [13]. This last result is in a sense “optimal” as it is known that if a class of databases is closed under substructures and has no FPT model checking algorithm then it is somewhere dense [17], modulo some reasonable complexity hypothesis.

Classes of databases of low degree do not belong to this setting. It is easy to see that they are neither nowhere dense nor closed under substructures (see Section 2.3). Our algorithms build on the known model checking algorithm for low degree databases [12]. They generalize the known enumeration algorithms for databases of bounded degree [5,14].
This paper is the journal version of [6]. There are two main differences with the conference version. In the conference version we needed the extra hypothesis that even though we would use only a memory of pseudo-linear size, a total amount of memory of quadratic size was necessary for our algorithms to work. This extra memory is no longer necessary here thanks to the data structure constructed in Theorem [1]. This makes the technical lemma slightly more complicated to state, but does not affect the general results. The second difference is that we now prove a slightly stronger result than just constant delay enumeration. We prove that, after a preprocessing phase of pseudo-linear time, given an arbitrary tuple we can compute in constant time the smallest solution to the query, relative to the lexicographical order, bigger than the input tuple. Constant delay enumeration follows immediately from this and moreover, the enumeration is done in lexicographical order, which was left open in the conference version.

We write $N$ the set of non-negative integers, and we let $N_\geq := N \setminus \{0\}$. $\mathbb{Q}$ denotes the set of rationals, and $\mathbb{Q}_>\mathbb{n}$ is the set of positive rationals.

## 2 Preliminaries and Main Results

We write $\mathbb{N}$ to denote the set of non-negative integers, and we let $\mathbb{N}_> := \mathbb{N} \setminus \{0\}$. $\mathbb{Q}$ denotes the set of rationals, and $\mathbb{Q}_>\mathbb{n}$ is the set of positive rationals.

### 2.1 Databases and queries

A database is a finite relational structure. A relational signature $\sigma$ is a finite set of relation symbols $R$, each of them associated with a fixed arity $ar(R) \in \mathbb{N}_\geq$. A relational structure $\mathcal{A}$ over $\sigma$, or a $\sigma$-structure (we omit to mention $\sigma$ when it is clear from the context) consists of a non-empty finite set $\text{dom}(\mathcal{A})$ called the domain of $\mathcal{A}$, and an $ar(R)$-ary relation $R^\mathcal{A} \subseteq \text{dom}(\mathcal{A})^{ar(R)}$ for each relation symbol $R \in \sigma$.

The degree of a structure $\mathcal{A}$, denoted $\text{degree}(\mathcal{A})$, is the degree of the Gaifman graph associated with $\mathcal{A}$ (i.e., the undirected graph with vertex set $\text{dom}(\mathcal{A})$ where there is an edge between two nodes if they both occur in a tuple that belongs to a relation of $\mathcal{A}$). In the sequel we only consider structures of degree $\geq 2$. As structures of degree 1 are quite trivial, this is without loss of generality.

We define the size $|\mathcal{A}|$ of $\mathcal{A}$ as $|\mathcal{A}| = |\sigma| + |\text{dom}(\mathcal{A})| + \sum_{R \in \sigma} |R^\mathcal{A}| \cdot ar(R)$. It corresponds to the size of a reasonable encoding of $\mathcal{A}$. The cardinality of $\mathcal{A}$, i.e. the cardinality of its domain, is denoted by $|\mathcal{A}|$.

By query we mean a formula of $\text{FO}(\sigma)$, the set of all first-order formulas of signature $\sigma$, for some relational signature $\sigma$ (again we omit $\sigma$ when it is clear from the context). For $\varphi \in \text{FO}$, we write $\varphi(\vec{x})$ to denote a query whose free variables are $\vec{x}$, and the number of free variables is called the arity of the query. A sentence is a query of arity 0. Given a structure $\mathcal{A}$ and a query $\varphi$, an answer to $\varphi$ in $\mathcal{A}$ is a tuple $\vec{a}$ of elements of $\text{dom}(\mathcal{A})$ such that $\mathcal{A} \models \varphi(\vec{a})$. In the special case where $\varphi$ is a sentence then it is either true of false in $\mathcal{A}$ and the former is denoted $\mathcal{A} \models \varphi$. We write $\varphi(\mathcal{A})$ for the set of answers to $\varphi$ in $\mathcal{A}$, i.e. $\varphi(\mathcal{A}) = \{\vec{a} : \mathcal{A} \models \varphi(\vec{a})\}$. As usual, $|\varphi|$ denotes the size of $\varphi$.

Let $\mathcal{C}$ be a class of structures. The model checking problem of $\text{FO}$ over $\mathcal{C}$ is the computational problem of given a sentence $\varphi \in \text{FO}$ and a database $\mathcal{A} \in \mathcal{C}$ to test whether $\mathcal{A} \models \varphi$ or not.

We write $\mathbb{N}$ the set of non-negative integers, and we let $\mathbb{N}_\geq := \mathbb{N} \setminus \{0\}$. $\mathbb{Q}$ denotes the set of rationals, and $\mathbb{Q}_>\mathbb{n}$ is the set of positive rationals.

## 2 Preliminaries and Main Results

We write $\mathbb{N}$ to denote the set of non-negative integers, and we let $\mathbb{N}_> := \mathbb{N} \setminus \{0\}$. $\mathbb{Q}$ denotes the set of rationals, and $\mathbb{Q}_>\mathbb{n}$ is the set of positive rationals.

### 2.1 Databases and queries

A database is a finite relational structure. A relational signature $\sigma$ is a finite set of relation symbols $R$, each of them associated with a fixed arity $ar(R) \in \mathbb{N}_\geq$. A relational structure $\mathcal{A}$ over $\sigma$, or a $\sigma$-structure (we omit to mention $\sigma$ when it is clear from the context) consists of a non-empty finite set $\text{dom}(\mathcal{A})$ called the domain of $\mathcal{A}$, and an $ar(R)$-ary relation $R^\mathcal{A} \subseteq \text{dom}(\mathcal{A})^{ar(R)}$ for each relation symbol $R \in \sigma$.

The degree of a structure $\mathcal{A}$, denoted $\text{degree}(\mathcal{A})$, is the degree of the Gaifman graph associated with $\mathcal{A}$ (i.e., the undirected graph with vertex set $\text{dom}(\mathcal{A})$ where there is an edge between two nodes if they both occur in a tuple that belongs to a relation of $\mathcal{A}$). In the sequel we only consider structures of degree $\geq 2$. As structures of degree 1 are quite trivial, this is without loss of generality.

We define the size $|\mathcal{A}|$ of $\mathcal{A}$ as $|\mathcal{A}| = |\sigma| + |\text{dom}(\mathcal{A})| + \sum_{R \in \sigma} |R^\mathcal{A}| \cdot ar(R)$. It corresponds to the size of a reasonable encoding of $\mathcal{A}$. The cardinality of $\mathcal{A}$, i.e. the cardinality of its domain, is denoted by $|\mathcal{A}|$.

By query we mean a formula of $\text{FO}(\sigma)$, the set of all first-order formulas of signature $\sigma$, for some relational signature $\sigma$ (again we omit $\sigma$ when it is clear from the context). For $\varphi \in \text{FO}$, we write $\varphi(\vec{x})$ to denote a query whose free variables are $\vec{x}$, and the number of free variables is called the arity of the query. A sentence is a query of arity 0. Given a structure $\mathcal{A}$ and a query $\varphi$, an answer to $\varphi$ in $\mathcal{A}$ is a tuple $\vec{a}$ of elements of $\text{dom}(\mathcal{A})$ such that $\mathcal{A} \models \varphi(\vec{a})$. In the special case where $\varphi$ is a sentence then it is either true of false in $\mathcal{A}$ and the former is denoted $\mathcal{A} \models \varphi$. We write $\varphi(\mathcal{A})$ for the set of answers to $\varphi$ in $\mathcal{A}$, i.e. $\varphi(\mathcal{A}) = \{\vec{a} : \mathcal{A} \models \varphi(\vec{a})\}$. As usual, $|\varphi|$ denotes the size of $\varphi$.

Let $\mathcal{C}$ be a class of structures. The model checking problem of $\text{FO}$ over $\mathcal{C}$ is the computational problem of given a sentence $\varphi \in \text{FO}$ and a database $\mathcal{A} \in \mathcal{C}$ to test whether $\mathcal{A} \models \varphi$ or not.
Given a $k$-ary query $\varphi$, we care about “enumerating” $\varphi(A)$ efficiently. Let $C$ be a class of structures. The enumeration problem of $\varphi$ over $C$ is, given a database $A \in C$, to output the elements of $\varphi(A)$ one by one with no repetition. The time needed to output the first solution is called the preprocessing time. The maximal time between any two consecutive outputs of elements of $\varphi(A)$ is called the delay. We are interested here in enumeration algorithms with pseudo-linear preprocessing time and constant delay. We now make these notions formal.

### 2.2 Model of computation and enumeration

We use Random Access Machines (RAMs) with addition and uniform cost measure as a model of computation. For further details on this model and its use in logic see [8, 11]. In the sequel we assume that the input relational structure comes with a linear order on the domain. If not, we use the one induced by the encoding of the structure as an input to the RAM. Whenever we iterate through all nodes of the domain, the iteration is with respect to the initial linear order. The linear order induces a lexicographical order on tuples of elements of the domain.

Our algorithms over RAMs will take as input a query $\varphi$ of size $k$ and a structure $A$ of size $n$. We adopt the data complexity point of view and say that a problem can be solved in linear time (respectively, constant time) if it can be solved by an algorithm outputting the solution within $f(k) \cdot n$ steps (respectively, $f(k)$ steps), for some function $f$. We also say a problem can be solved in pseudo-linear time if, for all $\varepsilon \in \mathbb{Q}_{>0}$, there is an algorithm solving it within $f(k, \varepsilon) \cdot n^{1+\varepsilon}$ steps, for some function $f$.

We will often compute partial $k$-ary functions associating a value to a tuple of nodes of the input graph. Such functions can be easily implemented in the RAM model using $k$-dimensional cubes allowing to retrieve the value of $f$ in constant time. This requires a memory usage of $O(n^k)$ and an initialization process of $O(n^k)$. However our functions will have a domain of size pseudo-linear and can be computed in pseudo-linear time. The following theorem states that we can use the RAM model to build a data structure that store our functions in a more efficient way. It’s proof can be found in [20].

**Theorem 1** (Storing Theorem). For every fixed $n, k \in \mathbb{N}$ and $\varepsilon > 0$, there is a data structure that stores the value of a $k$-ary function $f$ of domain $\text{dom}(f) \subseteq [n]^k$ with:

- Computation time $O(|\text{dom}(f)| \cdot n^\varepsilon)$,
- Lookup time only depending on $k$ and $\varepsilon$.

Here, lookup means that given a tuple $\bar{a} \in [n]^k$, the algorithm either answers $b$ if $\bar{a} \in \text{dom}(f)$ and $f(\bar{a}) = b$, or void otherwise.

An important consequence of Theorem 1 is that, modulo a preprocessing in time pseudo-linear in the size of the database, we can test in constant time whether an input tuple is a fact of the database.

**Corollary 2.** Let $A$ be a database over the schema $\sigma$. Let $\varepsilon > 0$. One can compute in time $O(dn^{1+\varepsilon})$ a data structure such that on input a tuple $\bar{a}$ and a relation symbol $R \in \sigma$ one can test whether $A \models R(\bar{a})$ in time $O(1)$, where $n = |\text{dom}(A)|$ and $d = \text{degree}(A)$.

**Proof.** Immediate from Theorem 1 as the number of tuples in a structure of degree $d$ is at most $dn$. □

Note that a simple linear time preprocessing would provide a data structure allowing for a test as in Corollary 2 in time $O(d)$. The extra effort of Corollary 2 is to get a test in constant time, i.e. depending only in $\varepsilon$ and $\sigma$ and not on $d$. 
We say that the enumeration problem of FO over a class $C$ of structures can be solved with constant delay after a pseudo-linear preprocessing, if it can be solved by a RAM algorithm which, on input $\varepsilon > 0$, $q \in \text{FO}$ and $A \in C$, can be decomposed into two phases:

- a preprocessing phase that is performed in time $f(\varepsilon, |q|) \cdot |A|$ for some function $f$, and
- an enumeration phase that outputs $q(A)$ with no repetition and a delay depending only on $q$, $\varepsilon$ and $C$ 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 $q$, $\varepsilon$ and $C$.

Notice that if we can enumerate $q$ with constant delay after a pseudo-linear preprocessing, then all answers can be output in time $g(|q|, \varepsilon) \cdot (|A|^{1+\varepsilon} + |q(A)|)$, for some function $g$, and the first solution is computed in pseudo-linear time. In the particular case of boolean queries, the associated model checking problem must be solvable in pseudo-linear time.

**Example 3.** To illustrate these notions, consider the binary query $q(x, y)$ over colored graphs computing the pairs of nodes $(x, y)$ such that $x$ is blue, $y$ is red and there is no edge from $x$ to $y$. It can be expressed in FO by

$$B(x) \land R(y) \land \neg E(x, y).$$

A naive algorithm for evaluating $q$ would iterate through all blue nodes, then iterate through all red nodes, check if they are linked by an edge and, if not, output the resulting pair, otherwise try the next pair.

With our RAM model, after a linear preprocessing, we can easily iterate through all blue nodes with a constant delay between any two of them and similarly for red nodes. By Corollary 2, we can test in constant time whether there is an edge between any two nodes. The problem with the above algorithm is that many pairs of appropriate color may be false hits. Hence the delay between two consecutive outputs may be arbitrarily large.

If the degree is assumed to be bounded, then the above algorithm enumerates all answers with constant delay, since the number of false hits for each blue node is bounded by the degree. We will see that for structures of low degree we can modify the algorithm in order to achieve the same result.

**2.3 Classes of structures of low degree**

Intuitively a class $C$ of structures has low degree if for all $\delta > 0$, all but finitely many structures $A$ of $C$ have degree at most $|A|^\delta$ (see [12]). More formally, $C$ has low degree if for every $\delta \in \mathbb{Q}_{> 0}$ there is an $n_\delta \in \mathbb{N}_{> 1}$ such that all structures $A \in C$ of cardinality $|A| \geq n_\delta$ have $\text{degree}(A) \leq |A|^\delta$. If there is a computable function associating $n_\delta$ from $\delta$ then we furthermore say that the class is effective.

For example, for every fixed number $c > 0$, the class of all structures of degree at most $(\log n)^c$ is of low degree and effective. Clearly, an arbitrary class $C$ of structures can be transformed into a class $C'$ of low degree by padding each $A \in C$ with a suitable number of isolated elements (i.e., elements of degree 0). Therefore classes of low degree are usually not closed under taking substructures. In particular if we apply the padding trick to the class of cliques, we obtain a class of low degree that is not in any of the class with known low evaluation complexity such as the “nowhere dense” case mentioned in the introduction.

Notice that $\text{degree}(A) \leq |A|^\delta$ implies that $|A| \leq c \cdot |A|^{1+\delta r}$, where $r$ is the maximal arity of the signature and $c$ is a number only depending on $\sigma$. Therefore, if the signature is fixed, all our bounds concerning databases in a class of low degree could be expressed using $|A|$ instead of $|A|$ modulo a small change of the parameters.
It is known that on classes of graphs of low degree, model checking of first-order sentences can be done in pseudo-linear time. We will actually need the following stronger result:

**Theorem 4 (Grohe [12]).** There is a computable function $h$ such that on input of a structure $A \in C$ of degree $d$ and a sentence $q \in \text{FO}$, one can test in time $h(|q|) \cdot |A| \cdot d^{h(|q|)}$ whether $A \models q$.

### 2.4 Our results

We are now ready to state our main results, which essentially lift Theorem 4 to non-boolean queries and to counting, testing, and enumerating their answers.

Our first result is that we can count the number of answers to a query in pseudo-linear time.

**Theorem 5.** Let $C$ be a class of structures of low degree. There is a function $g$ such that, given a structure $A \in C$, a query $q \in \text{FO}$ and $\varepsilon > 0$, one can compute $|q(A)|$ in time $g(|q|, \varepsilon) \cdot (|A|)^{1+\varepsilon}$. If $C$ is effective then $g$ is computable.

Our second result is that we can test whether a given tuple is part of the answers in constant time after a pseudo-linear time preprocessing.

**Theorem 6.** Let $C$ be a class of structures of low degree. There is a function $g$ such that, given a structure $A \in C$, a query $q \in \text{FO}$ and $\varepsilon > 0$, one can compute in time $g(|q|, \varepsilon) \cdot (|A|)^{1+\varepsilon}$ a data structure such that, on input of any $\bar{a}$, one can then test in time $g(|q|, \varepsilon)$ whether $\bar{a} \in q(A)$. If $C$ is effective then $g$ is computable.

Finally, we show that we can enumerate the answers to a query with constant delay after a pseudo-linear time preprocessing.

**Theorem 7.** Let $C$ be a class of structures of low degree. There is a function $g$ such that given a structure $A \in C$, a query $q \in \text{FO}$ and $\varepsilon > 0$, the enumeration problem of $q$ over $A$ can be solved with delay $g(|q|, \varepsilon)$ after a preprocessing running in time $g(|q|, \varepsilon) \cdot (|A|)^{1+\varepsilon}$. If $C$ is effective then $g$ is computable.

### 2.5 Further notation

We close this section by fixing technical notations that will be used throughout this paper.

For a structure $A$ we write $\text{dist}^A(a, b)$ for the distance between two nodes $a$ and $b$ of the Gaifman graph of $A$. For an element $a \in \text{dom}(A)$ and a number $r \in \mathbb{N}$, the $r$-ball around $a$ is the set $N^A_r(a)$ of all nodes $b \in \text{dom}(A)$ with $\text{dist}^A(a, b) \leq r$. The $r$-neighborhood around $a$ is the induced substructure $N^A_r(a)$ of $A$ on $N^A_r(a)$. Note that if $A$ is of degree $\leq d$ for $d \geq 2$, then $|N^A_r(a)| \leq \sum_{i=0}^r d^i < d^{r+1}$.

### 3 Evaluation algorithms

In this section, we present our algorithms for counting, testing, and enumerating the solutions to a query (see Sections 3.3, 3.4, and 3.5). They all build on the same preprocessing algorithm which runs in pseudo-linear time and which essentially reduces the input to a quantifier-free query over a suitable signature (see Section 3.2). However, before presenting these algorithms, we start with a very simple case.
3.1 Warming up

As a warm-up for working with classes of structures of low degree, we first consider the simple case of queries which we call connected conjunctive queries, and which are defined as follows.

A conjunction is a query \( \gamma \) which is a conjunction of relational atoms and potentially negated unary atoms. Note that the query of Example 3 is not a conjunction as it has a binary negated atom.

With each conjunction \( \gamma \) we associate a query graph \( H_\gamma \). This is the undirected graph whose vertices are the variables \( x_1, \ldots, x_k \) of \( \gamma \), and where there is an edge between two vertices \( x_i \) and \( x_j \) if \( \gamma \) contains a relational atom in which both \( x_i \) and \( x_j \) occur. We call the conjunction \( \gamma \) connected if its query graph \( H_\gamma \) is connected.

A connected conjunctive query is a query \( q(\bar{x}) \) of the form \( \exists \bar{y} \gamma(\bar{x}, \bar{y}) \), where \( \gamma \) is a connected conjunction in which all variables of \( \bar{x}, \bar{y} \) occur (here, \(|\bar{y}| = 0 \) is allowed).

The next simple lemma implies that over a class of structures of low degree, connected conjunctive queries can be evaluated in pseudo-linear time. It will be used in several places throughout this paper: in the proof of Proposition 9 and in the proofs for our counting and enumeration results in Sections 3.3 and 3.5.

**Lemma 8.** There is an algorithm which, at input of a structure \( \mathcal{A} \) and a connected conjunctive query \( q(\bar{x}) \) computes \( q(\mathcal{A}) \) in time \( O(|q| \cdot n \cdot d^h(|q|)) \), where \( n = |\text{dom}(\mathcal{A})| \), \( d = \text{degree}(\mathcal{A}) \), and \( h \) is a computable function.

**Proof.** Let \( q(\bar{x}) \) be of the form \( \exists \bar{y} \gamma(\bar{x}, \bar{y}) \), for a connected conjunction \( \gamma \). Let \( k = |\bar{x}| \) be the number of free variables of \( q \), let \( \ell = |\bar{y}| \), and let \( r = k + \ell \).

Note that since \( \gamma \) is connected, for every tuple \( \bar{c} \in \gamma(\mathcal{A}) \) the following is true, where \( a \) is the first component of \( \bar{c} \). All components \( c' \) of \( \bar{c} \) belong to the \( r \)-neighborhood \( \mathcal{N}_r^A(a) \) of \( a \) in \( \text{dom}(\mathcal{A}) \). Thus, \( q(\mathcal{A}) \) is the disjoint union of the sets

\[
S_a := \left\{ \bar{b} \in q(\mathcal{N}_r^A(a)) : \text{the first component of } \bar{b} \text{ is } a \right\},
\]

for all \( a \in \text{dom}(\mathcal{A}) \). For each \( a \in \text{dom}(\mathcal{A}) \), the set \( S_a \) can be computed as follows:

1. Initialize \( S_a := \emptyset \).
2. Compute \( \mathcal{N}_r^A(a) \).
   
   Since \( \mathcal{A} \) has degree \( \leq d \), this neighborhood’s domain contains at most \( d^{r+1} \) elements of \( \text{dom}(\mathcal{A}) \). Thus, by using breadth-first search, \( \mathcal{N}_r^A(a) \) can be computed in time \( O(d^h(|q|)) \), for a computable function \( h \).

3. Use a brute-force algorithm that enumerates all \( k \)-tuples \( \bar{b} \) of elements in \( \mathcal{N}_r^A(a) \) whose first component is \( a \).
   
   For each such tuple \( \bar{b} \), use a brute-force algorithm that checks whether \( \mathcal{N}_r^A(a) \models q(\bar{b}) \). If so, insert \( \bar{b} \) into \( S_a \).

   Note that the number of considered tuples \( \bar{b} \) is \( \leq d^{(r+1)(k-1)} \). And checking whether \( \mathcal{N}_r^A(a) \models q(\bar{b}) \) can be done in time \( O(|\gamma| \cdot d^{r+1} \ell) \): for this, enumerate all \( \ell \)-tuples \( \bar{c} \) of elements in \( \mathcal{N}_r^A(a) \) and take time \( O(|\gamma|) \) to check whether \( \gamma(\bar{x}, \bar{y}) \) is satisfied by the tuple \( (\bar{b}, \bar{c}) \).

   Thus, we are done after \( O(|\gamma| \cdot d^{r+2}) \) steps.

In summary, we can compute \( q(\mathcal{A}) = \bigcup_{a \in \mathcal{A}} S_a \) in time \( O(n \cdot |q| \cdot d^h(|q|)) \), for a computable function \( h \). \( \square \)
As an immediate consequence we can compute in pseudo-linear time the answers to a connected conjunctive query over a class of structures of low degree.

**Proposition 9.** Let $C$ be a class of structures of low degree. Given a structure $A \in C$, a connected conjunctive query $q$ and $\varepsilon > 0$, one can compute $q(A)$ in time $q(|q|, \varepsilon) \cdot |A|^{1+\varepsilon}$ for some function $g$ which is computable when $C$ is effective.

**Proof.** We use the algorithm provided in Lemma 8. To see that the running time is as claimed, we use the assumption that $C$ is of low degree: for every $\delta > 0$ there is an $m_\delta \in \mathbb{N}_{>1}$ such that every structure $A \in C$ of cardinality $|A| \geq m_\delta$ has degree $A \leq |A|^\delta$.

For a given $\varepsilon > 0$ we let $\delta := \frac{\varepsilon}{h(|q|)}$ and define $n_\delta := m_\delta$. Then, every $A \in C$ with $|A| \geq n_\delta$ has degree $A \leq |A|^{|h(|q|)|}$. Thus, on input of $A$ and $q$, the algorithm from Lemma 8 has running time $O(|q|\cdot |A|^{1+\varepsilon})$ if $|A| \geq n_\delta$ and takes time bounded by $O(|q| \cdot n_\delta^{1+|h(|q|)|})$ otherwise. This gives the bounds claimed by the proposition with a computable function $g$ as soon as we can compute $n_\delta$. \qed

The method of the proof of Proposition 9 above will be used for several times in the paper.

### 3.2 Quantifier elimination and normal form

In this section, we make precise the quantifier elimination approach that is at the heart of the preprocessing phase of the query evaluation algorithms of our paper.

A signature is binary if all its relation symbols have arity at most 2. A colored graph is a finite relational structure over a binary signature.

**Proposition 10.** There is an algorithm which, at input of a structure $A$, a first-order query $\varphi(x)$ and $\varepsilon > 0$, produces a binary signature $\tau$ (containing, among other symbols, a binary relation symbol $E$), a colored graph $G$ of signature $\tau$, an FO($\tau$)-formula $\psi(x)$, a mapping $f$ and a data structure such that the following is true for $k = |x|$, $n = |\text{dom}(A)|$, $d = \text{degree}(A)$ and $h$ some computable function:

1. $\psi$ is quantifier-free. Furthermore, $\psi$ is of the form $(\psi_1 \land \psi_2)$, where $\psi_1$ states that no distinct free variables of $\psi$ are connected by an E-edge, and $\psi_2$ is a positive boolean combination of unary atoms.

2. $\tau$ and $\psi$ are computed in time $h(|\varphi|) \cdot n^d h(|\varphi|)$.
   Moreover, $|\tau| \leq h(|\varphi|)$ and $|\psi| \leq h(|\varphi|)$.

3. $G$ is computed in time $h(|\varphi|) \cdot n^d h(|\varphi|)$.
   Moreover, degree$(G) \leq d h(|\varphi|)$.

4. $f$ is an injective mapping from dom$(A)^k$ to dom$(G)^k$ such that $f$ is a bijection between $\varphi(A)$ and $\psi(G)$.

The data structure can be computed in time $h(|\varphi|) \cdot n^{1+\varepsilon} \cdot d h(|\varphi|)$ and can be used for, on input of any tuple $\bar{a} \in \text{dom}(A)^k$, computing the tuple $f(\bar{a})$ in time $O(k^2)$ and, on input of any tuple $\bar{v} \in \text{dom}(G)^k$, computing the tuple $f^{-1}(\bar{v})$ in time $O(k^2)$.

The proof of Proposition 10 is long and technical and of a somewhat different nature than the results we now describe. It is postponed to Section 4. However, this proposition is central in the proofs of the results below.
3.3 Counting

Here we consider the problem of counting the number of solutions to a query on low degree structures.

A generalized conjunction is a conjunction of relational atoms and negated relational atoms (hence, also atoms of arity bigger than one may be negated, and the query of Example 3 is a generalized conjunction).

Example 11. Before moving to the formal proof of Theorem 5 consider again the query \( q \) from Example 3. Recall that it computes the pairs of blue-red nodes that are not connected by an edge. To count its number of solutions over a class of structures of low degree we can proceed as follows. We first consider the query \( q' \) returning the set of blue-red nodes that are connected. In other words, \( q' \) is

\[
B(x) \land R(y) \land E(x,y).
\]

Notice that this query is a connected conjunction. Hence, by Proposition 9 its answers can be computed in pseudo-linear time and therefore we can also count its number of solutions in pseudo-linear time. It is also easy to compute in pseudo-linear time the number of pairs of blue-red nodes. The number of answers to \( q \) is then the difference between these two numbers.

The proof sketch for Theorem 5 goes as follows. Using Proposition 10 we can assume modulo a pseudo-linear preprocessing that our formula is quantifier-free and over a binary signature. Each connected component is then treated separately and we return the product of all the results. For each connected component we eliminate the negated symbols one by one using the trick illustrated in Example 11. The resulting formula is then a connected conjunction that is treated in pseudo-linear time using Proposition 9.

Lemma 12. There is an algorithm which, at input of a colored graph \( G \) and a generalized conjunction \( \gamma(\bar{x}) \), computes \( |\gamma(G)| \) in time \( O(2^m \cdot |\gamma| \cdot n \cdot d^h(|\gamma|)) \), where \( h \) is a computable function, \( m \) is the number of negated binary atoms in \( \gamma \), \( n = |\text{dom}(G)| \), and \( d = \text{degree}(G) \).

Proof. By induction on the number \( m \) of negated binary atoms in \( \gamma \). The base case for \( m = 0 \) is obtained as follows. We start by using \( O(|\gamma|) \) steps to compute the query graph \( H_\gamma \) and to compute the connected components of \( H_\gamma \).

In case that \( H_\gamma \) is connected, we can use Lemma 8 to compute the entire set \( \gamma(G) \) in time \( O(|\gamma| \cdot n \cdot d^h(|\gamma|)) \), for a computable function \( h \). Thus, counting \( |\gamma(G)| \) can be done within the same time bound.

In case that \( \gamma \) is not connected, let \( H_1, \ldots, H_\ell \) be the connected components. For each \( i \in \{1, \ldots, \ell\} \) let \( \bar{x}_i \) be the tuple obtained from \( \bar{x} \) by removing all variables that do not belong to \( H_i \). Furthermore, let \( \gamma_i(\bar{x}_i) \) be the conjunction of all atoms or negated unary atoms of \( \gamma \) that contain variables in \( H_i \). Note that \( \gamma(\bar{x}) \) is equivalent to \( \bigwedge_{i=1}^\ell \gamma_i(\bar{x}_i) \), and

\[
|\gamma(G)| = \prod_{i=1}^\ell |\gamma_i(G)|.
\]

Since each \( \gamma_i \) is connected, we can compute \( |\gamma_i(G)| \) in time \( O(|\gamma_i| \cdot n \cdot d^h(|\gamma_i|)) \) by using the algorithm of Lemma 8. We do this for each \( i \in \{1, \ldots, \ell\} \) and output the product of the values. In summary, we are done in time \( O(|\gamma| \cdot n \cdot d^h(|\gamma|)) \) for the base case \( m = 0 \).
For the induction step, let $\gamma$ be a formula with $m+1$ negated binary atoms. Let $\neg R(x, y)$ be a negated binary atom of $\gamma$, and let $\gamma_1$ be such that

$$\gamma = \gamma_1 \land \neg R(x, y),$$

and let

$$\gamma_2 := \gamma_1 \land R(x, y).$$

Clearly, $|\gamma(\mathcal{G})| = |\gamma_1(\mathcal{G})| - |\gamma_2(\mathcal{G})|$. Since each of the formulas $\gamma_1$ and $\gamma_2$ has only $m$ negated binary atoms, we can use the induction hypothesis to compute $|\gamma_1(\mathcal{G})|$ and $|\gamma_2(\mathcal{G})|$ each in time $O(2^m \cdot |\gamma| \cdot n \cdot d^h(|\gamma|))$. The total time used for computing $|\gamma(\mathcal{G})|$ is thus $O(2^{m+1} \cdot |\gamma| \cdot n \cdot d^h(|\gamma|))$. 

By using Proposition 10, we can lift this to arbitrary structures and first-order queries:

**Proposition 13.** There is an algorithm which at input of a structure $\mathcal{A}$ and a first-order query $\varphi(\bar{x})$ computes $|\varphi(\mathcal{A})|$ in time $h(|\varphi|) \cdot n \cdot d^h(|\varphi|)$, for a computable function $h$, where $n = |\text{dom}(\mathcal{A})|$ and $d = \text{degree}(\mathcal{A})$.

**Proof.** We first use the algorithm of Proposition 10 to compute the according graph $\mathcal{G}$ and the quantifier-free formula $\psi(\bar{x})$. This takes time $h(|\varphi|) \cdot n \cdot d^h(|\varphi|)$ for a computable function $h$. And we also know that $|\psi| \leq h(|\varphi|)$. By Proposition 10 we know that $|\varphi(\mathcal{A})| = |\psi(\mathcal{G})|$.

Next, we transform $\psi(\bar{x})$ into disjunctive normal form

$$\bigvee_{i \in I} \gamma_i(\bar{x}),$$

such that the conjunctive clauses $\gamma_i$ exclude each other (i.e., for each $\bar{v} \in \psi(\mathcal{G})$ there is exactly one $i \in I$ such that $\bar{v} \in \gamma_i(\mathcal{G})$). Clearly, this can be done in time $O(2^{|\psi|})$. Each $\gamma_i$ has length at most $|\psi|$, and $|I| \leq 2^{|\psi|}$.

Obviously, $|\psi(\mathcal{G})| = \sum_{i \in I} |\gamma_i(\mathcal{G})|$. We now use, for each $i \in I$, the algorithm from Lemma 12 to compute the number $s_i = |\gamma_i(\mathcal{G})|$ and output the value $s = \sum_{i \in I} s_i$.

By Lemma 12, we know that for each $i \in I$ computing $s_i$ can be done in time $O(2^m \cdot |\gamma_i| \cdot \tilde{n} \cdot d^{h_0}(|\gamma_i|))$, where $m$ is the number of binary atoms in $\gamma$, $\tilde{n} = |\text{dom}(\mathcal{G})|$, $\tilde{d} = \text{degree}(\mathcal{G})$, and $h_0$ is some computable function.

By Proposition 10 we know that $\tilde{n} \leq h(|\varphi|) \cdot n \cdot d^h(|\varphi|)$ and $\tilde{d} \leq d^h(|\varphi|)$. Since also $|\gamma_i| \leq |\psi| \leq h(|\varphi|)$, the computation of $s_i$, for each $i \in I$, takes time $h_1(|\varphi|) \cdot n \cdot d^{h_1}(|\varphi|)$, for some computable function $h_1$ (depending on $h$ and $h_0$).

To conclude, since $|I| \leq 2^{|\psi|}$, the total running time for the computation of $|\varphi(\mathcal{A})| = \sum_{i \in I} s_i$ is $h_2(|\varphi|) \cdot n \cdot d^{h_2}(|\varphi|)$, for a suitably chosen computable function $h_2$. Hence, we meet the required bound. 

**Theorem 5** is an immediate consequence of Proposition 13 following arguments similar with the proof of Proposition 4. For a given $\epsilon > 0$ we let $\delta := \frac{\epsilon}{h(|\varphi|)}$, where $h$ is the function of Proposition 13, and define $n_\epsilon := m \delta$. Then, every $\mathcal{A} \in C$ with $|\mathcal{A}| \geq n_\epsilon$ has $\text{degree}(\mathcal{A}) \leq |\mathcal{A}|^{1+\epsilon}$. Thus, on input of $\mathcal{A}$ and $\varphi$, the algorithm from Proposition 13 has running time $O(h(|\varphi|) \cdot |\mathcal{A}|^{1+\epsilon})$ if $|\mathcal{A}| \geq n_\epsilon$ and takes time bounded by $h(|\varphi|) \cdot n_\epsilon^{1+h(|\varphi|)}$ otherwise. This gives the bounds claimed by the proposition with a computable function $g$ as soon as we can compute $n_\epsilon$. 

3.4 Testing

Here we consider the problem of testing whether a given tuple is a solution to a query. By Proposition 10 it is enough to consider quantifier-free formulas. Those are treated using the data structure computed by Corollary 2.

Proposition 14. For all \( \varepsilon > 0 \), there is an algorithm which at input of a structure \( A \) and a first-order query \( \varphi(\bar{x}) \), has a preprocessing phase of time \( g(|\varphi|, \varepsilon) \cdot n^{1+\varepsilon} \cdot d^{h(|\varphi|, \varepsilon)} \) in which it computes a data structure such that, on input of any \( \bar{a} \in \text{dom}(A)^k \) for \( k = |\bar{x}| \), it can be tested in time \( g(|\varphi|, \varepsilon) \) whether \( \bar{a} \in \varphi(A) \), where \( g \) is a computable function, \( n = |\text{dom}(A)| \), and \( d = \text{degree}(A) \).

Proof. Fix \( \varepsilon > 0 \).

We first use the algorithm of Proposition 10 to compute the graph \( G \), the quantifier-free formula \( \psi(\bar{x}) \) and the data structure for function \( f \). For some computable function \( h \), this takes time \( h(|\varphi|) \cdot n \cdot d^{h(|\varphi|)} \), and furthermore, \(|\psi| \leq h(|\varphi|)\) and \( \text{degree}(G) \leq d^{h(|\varphi|)} \). Note that \( |G| \leq h(|\varphi|) \cdot n \cdot d^{h(|\varphi|)} \).

By construction, we furthermore know for all \( \bar{a} \in \text{dom}(A)^k \) that \( \bar{a} \in \varphi(A) \iff f(\bar{a}) \in \psi(G) \).

Recall from Proposition 10 that \( \psi(\bar{x}) \) is a quantifier-free formula built from atoms of the form \( E(y, z) \) and \( C(y) \) for unary relation symbols \( C \). Thus, checking whether a given tuple \( \bar{v} \in \text{dom}(G)^k \) belongs to \( \psi(G) \) can be done easily, provided that one can check whether unary atoms \( C(u) \) and binary atoms \( E(u, u') \) hold in \( G \) for given nodes \( u, u' \) of \( G \).

Let \( \bar{n} = |\text{dom}(G)| \) and \( \bar{d} = \text{degree}(G) \). Recall that \( \bar{n} \leq h(|\varphi|) \cdot n \cdot d^{h(|\varphi|)} \) and \( \bar{d} \leq d^{h(|\varphi|)} \). Apply Corollary 2 to \( G \) and \( \varepsilon \). This gives an extra preprocessing time of \( O(\bar{d} \cdot \bar{n}^{1+\varepsilon}) \) which is \( O(g(|\varphi|, \varepsilon) \cdot n^{1+\varepsilon} \cdot d^{h(|\varphi|, \varepsilon)}) \).

Finally, the testing algorithm works as follows. Given a tuple \( \bar{a} \in \text{dom}(A)^k \), we first construct \( \bar{v} := f(\bar{a}) \) and then check whether \( \bar{v} \in \psi(G) \). Building \( \bar{v} := f(\bar{a}) \) can be done in time \( O(k^2) \) (see Proposition 10), and by Corollary 2 checking whether \( \bar{v} \in \psi(G) \) depends only on \( \psi, \varepsilon \) and \( \sigma \). Hence, we meet the required bound for testing.

Theorem 5 is an immediate consequence of Proposition 14 using the usual argument: For a given \( \varepsilon > 0 \) we let \( \delta := \frac{\varepsilon}{2g(|\varphi|, \varepsilon/2)} \), where \( g \) is the function of Proposition 14 and define \( n_\varepsilon := m_\delta \). Then, every \( A \in \mathcal{C} \) with \( |A| \geq n_\varepsilon \) has \( \text{degree}(A) \leq |A|^{\varepsilon/2g(|\varphi|, \varepsilon)} \). Thus, on input of \( A, \varphi \) and \( \varepsilon/2 \), the testing algorithm from Proposition 14 has preprocessing time \( O(g(|\varphi|, \varepsilon/2) \cdot |A|^{1+\varepsilon}) \) if \( |A| \geq n_\varepsilon \) and takes time bounded by \( g(|\varphi|, \varepsilon/2) \cdot n_\varepsilon^{1+\varepsilon/2+g(|\varphi|, \varepsilon/2)} \) otherwise and testing time \( O(g(|\varphi|, \varepsilon/2)) \). This gives the bounds claimed by the proposition with a computable function as soon as we can compute \( n_\varepsilon \).

3.5 Enumeration

Here we consider the problem of enumerating the solutions to a given query. We first illustrate the proof of Theorem 7 with our running example.

Example 15. Consider again the query \( q \) of Example 3. In order to enumerate \( q \) with constant delay over a class of low degree we proceed as follows. During the preprocessing phase we precompute those blue nodes that contribute to the answer set, i.e., such that there is a red node not connected to it. This is doable in pseudo-linear time because our class has low degree and each blue node is connected to few red nodes. We call green the resulting nodes. We then order the green nodes and the red nodes in order to be able to iterate through them with constant delay. Finally, we compute the binary function \( \text{skip}(x, y) \) associating to each green node \( x \) and red node \( y \) such that \( E(x, y) \) the smallest red node \( y' \) such that \( y < y' \) and \( \neg E(x, y') \), where \( < \) is the order on red nodes precomputed...
above. From Proposition 9 it follows that computing skip can be done in pseudo-linear time. It is crucial here that the domain of skip has pseudo-linear size and this is a consequence of the low degree.

The enumeration phase now goes as follows: We iterate through all green nodes. For each of them we iterate through all red nodes. If there is no edge between them, we output the result and continue with the next red node. If there is an edge, we apply skip to this pair and the process continues with the resulting red node. Note that the new red node immediately yields an answer. Note also that all the red nodes that will not be considered are safely skipped as they are linked to the current green node.

The proof of Theorem 7 can be sketched as follows. We actually show a stronger result, namely that modulo a pseudo-linear preprocessing we can upon input a tuple \( \vec{a} \) compute in constant time the smallest (with respect of the lexicographical order) solution tuple \( \vec{b} \) that is bigger than \( \vec{a} \). This is done as follows. By Proposition 10 it is enough to consider quantifier-free formulas looking for tuples of nodes that are disconnected and have certain colors. Hence the query \( q \) described in Example 3 corresponds to the binary case. For queries of larger arities we proceed by induction on the arity.

Assume we are given \( \vec{a} = \vec{a}'a \). We are looking for the smallest \( b \geq a \) such that \( \vec{a}'b \) is a solution (the case where there is no such \( b \) is taken care by induction by asking for the solution greater than \( \vec{a}' \) in the query projecting out the last element). We then proceed as in Example 3. If the current red node \( a \) is not connected to \( \vec{a} \), then \( \vec{a}a \vec{a} \) forms an answer and we are done. If \( a \) is connected to \( \vec{a} \) then we need to jump in constant time to the next red node that yields an answer. This is done by precomputing a suitable function \( skip \) that depends on the arity of the query and is slightly more complex that the one described in Example 3. The design and computation of this function is the main technical originality of the proof.

The technical details are summarized in the following proposition. Given a query \( \varphi \), a structure \( A \) and a tuple \( \vec{a} \) of elements in the domain of \( A \), we call the next solution of \( \varphi(A) \) bigger than \( \vec{a} \) the smallest tuple \( \vec{b} \in \varphi(A) \) such that \( \vec{b} \geq \vec{a} \), if it exists, and \( \text{void} \) otherwise.

**Proposition 16.** There is an algorithm which at input of a structure \( A \), a first-order query \( \varphi(\vec{x}) \) and \( \varepsilon > 0 \), performs a preprocessing in time \( h(|\varphi|, \varepsilon) \cdot n^{1+\varepsilon} \cdot d^{\delta(|\varphi|)} \) such that afterwards, upon input a tuple \( \vec{a} \), it computes in time \( h(|\varphi|, \varepsilon) \) the next solution of \( \varphi(A) \) bigger than \( \vec{a} \), where \( n = |\text{dom}(A)| \), \( d = \text{degree}(A) \), and \( h \) is a computable function. If there is no such tuple the algorithm returns \( \text{void} \).

**Proof.** The proof is by induction on the number \( k := |\vec{x}| \) of free variables of \( \varphi \). In case that \( k = 0 \), the formula \( \varphi \) is a sentence, and we are done using Theorem 4.

In the case \( k > 0 \).

We first use the algorithm of Proposition 10 to compute the according colored graph \( G \) and the quantifier-free formula \( \psi(\vec{x}) \). This takes time \( g(|\varphi|) \cdot n \cdot d^g(|\varphi|) \) for a computable function \( g \). And we know that \( |\psi| \leq g(|\varphi|) \), that \( G \) has degree \( d \leq d^g(|\varphi|) \), and that \( \text{dom}(G) \) has \( n \) elements, where \( n \leq g(|\varphi|) \cdot n \cdot d^g(|\varphi|) \).

From Item 1 of Proposition 10 we know that the formula \( \psi(\vec{x}) \) is of the form \( (\psi_1 \land \psi_2) \), where \( \psi_1 \) states that no distinct free variables of \( \psi \) are connected by an \( E \)-edge and \( \psi_2 \) is a positive boolean combination of unary atoms.

We now essentially prove the Proposition in the case of \( G \), i.e. given a tuple \( \vec{u} \) of \( \text{dom}(G)^k \) we compute the next element of \( \psi(G) \) that is bigger than \( \vec{u} \). We can go back and forth between \( A \) and \( G \) using \( f \) in constant time by Item 4 of Proposition 11. However we need to be careful about the order on tuples of \( \text{dom}(G)^k \) in order to respect lexicographical order on \( \text{dom}(A)^k \). Hence, for two tuples \( \vec{u} \)
and $\bar{v}$ of $\text{dom}(G)^k$, we write $\bar{u}\leq_G \bar{u}'$ if $f^{-1}(\bar{u}) \leq f^{-1}(\bar{u}')$ and in the sequel when we write “the next element of $\psi(G)$ that is bigger than $\bar{u}$” it is relative to the order $\leq_G$.

In case that $k = 1$, $\psi(x_1) = \psi_2(x_1)$ is a positive boolean combination of unary atoms. We can use Lemma 8 for each unary atom in order to compute $\psi(G)$ in time $O(|\psi| \cdot n \cdot d^h(\psi))$ for a computable function $g$. In time $O(n)$ we can also compute the function $\text{next}$ which associate to any $u \in \text{dom}(G)$ the smallest $v \geq_G u$ such that $v \in \psi(G)$. Altogether the preprocessing time is in $h(\varphi) \cdot n \cdot d^h(\varphi)$, for a computable function $h$ as required. Then upon input $a$ we simply return $f^{-1}(\text{next}(f(a)))$. Due to Item 4 of Proposition 10 the delay is $O(1)$, and we are done for $k = 1$.

The case $k > 1$ requires much more elaborate constructions. The preprocessing phase essentially computes a data structure such that afterwards, upon input a tuple $\bar{u}$, we can obtain in constant time the next solution of $\psi(G)$ bigger than $\bar{u}$.

We let $\bar{x} = (x_1, \ldots, x_k)$ and $\bar{x}_{k-1} := (x_1, \ldots, x_{k-1})$. We first transform $\psi$ into a normal form $\bigvee_{j \in J} \theta_j(\bar{x})$ such that the formulas $\theta_j$ exclude each other (i.e., for each $\bar{v} \in \psi(G)$ there is exactly one $j \in J$ such that $\bar{v} \in \theta_j(G)$), and each $\theta_j(\bar{x})$ is of the form

$$\phi_j(\bar{x}_{k-1}) \land P_j(x_k) \land \gamma(\bar{x}),$$

where

$$\gamma(\bar{x}) := \bigwedge_{i=1}^{k-1} (\neg E(x_i, x_k) \land \neg E(x_k, x_i)),$$

$P_j(x_k)$ is a boolean combination of unary atoms regarding $x_k$, and $\phi_j(\bar{x}_{k-1})$ is a formula with only $k-1$ free variables. Note that the transformation into this normal form can be done easily, using the particularly simple form of the formula $\psi$.

Clearly, upon input $\bar{u}$, it is enough to obtain for each $j \in J$ the next solution of $\theta_j(G)$ bigger than $\bar{u}$ and return the smallest of them.

In the following, we therefore restrict attention to $\theta_j$ for a fixed $j \in J$. For $\theta_j$ we shortly write

$$\theta(\bar{x}) = \phi(\bar{x}_{k-1}) \land P(x_k) \land \gamma(\bar{x}).$$

We let $\theta'(\bar{x}_{k-1}) := \exists x_k \theta(\bar{x})$.

We now perform the preprocessing obtained by induction for $\theta'$ (since $\theta'$ only has $k-1$ free variables) on $G$, that allows us to compute the next solution of $\theta'(G)$ in constant time.

Since $P(\bar{x}_k)$ is a boolean combination of unary atoms on $x_k$, we can use Lemma 8 to compute $P(G)$ in time $O(|P| \cdot n \cdot d^h(\{P\}))$ for a computable function $g$. Afterwards, we have available a list of all nodes $v$ of $G$ that belong to $P(G)$. In the following, we will write $\leq^P_G$ to denote the linear ordering of $P(G)$ induced by this list, and we write $\text{first}^P_G$ for the first element in this list, and $\text{next}^P_G$ for the successor function, such that for any node $v \in P(G)$, $\text{next}^P_G(v)$ is the next node in $P(G)$ in this list (or the value $\text{void}$, if $v$ is the last node in the list).

We extend the signature of $G$ by a unary relation symbol $P$ and a binary relation symbol $\text{next}$, and let $\hat{G}$ be the expansion of $G$ where $P$ is interpreted by the set $P(G)$ and $\text{next}$ is interpreted by the successor function $\text{next}^P_G$ (i.e., $\text{next}(v, v')$ is true in $G$ iff $v' = \text{next}^P_G(v)$). Note that $\hat{G}$ has degree $\bar{d} = \bar{d} + 2$.

We now start the key idea of the proof, i.e., the function that will help us skipping over irrelevant nodes. To this end consider the first-order formulas $E_1, \ldots, E_k$ defined inductively as follows, where $E'(x, y)$ is an abbreviation for $(E(x, y) \lor E(y, x))$. The reason for defining these formulas will become clear only later on, in the proof.

$$E_1(u, y) := E'(u, y),$$

and
\[ E_{i+1}(u, y) := E_i(u, y) \lor \exists z \exists z' \exists v \ (E'(z, u) \land \text{next}(z', z) \land E'(v, z') \land E_i(v, y)). \]

A simple induction shows that for \( E_i(u, y) \) to hold, \( y \) must be at distance \( \leq 3(i-1) + 1 < 3i \) from \( u \).

In algorithm we will have to test, given \( i \geq k \) and nodes \( u, v \in \text{dom}(\mathcal{G}) \), whether \( (u, v) \in E_i(\hat{\mathcal{G}}) \).

Since \( E_i \) is a first-order formula, Theorem 6 implies that, after preprocessing time \( g'(|E_i|, \varepsilon) \cdot n_i^{1+\varepsilon} \cdot d^{\theta(|E_i|)} \) (for some computable function \( g' \)), testing membership in \( E_i(\hat{\mathcal{G}}) \), for any given \( (u, v) \in \text{dom}(\mathcal{G})^2 \), is possible within time \( g'(|E_i|, \varepsilon) \).

The last step of the precomputation phase computes the function \( \text{skip} \) that associates to each node \( y \in P(\mathcal{G}) \) and each set \( V \) of at most \( k-1 \) nodes that are related to \( y \) via \( E_k \), the smallest (according to the order \( \leq \mathcal{P} \) of \( P(\mathcal{G}) \)) element \( z \geq y \) in \( P(\mathcal{G}) \) that is not connected by an \( E \)-edge to any node in \( V \). More precisely: For any node \( y \in P(\mathcal{G}) \) and any set \( V \) with \( 0 \leq |V| < k \) and \( (v, y) \in E_k(\hat{\mathcal{G}}) \) for all \( v \in V \), we let

\[ \text{skip}(y, V) := \min\{ z \in P(\mathcal{G}) : y \leq z \text{ and } \forall v \in V : (v, z) \not\in E'(\mathcal{G}) \}, \]

respectively, \( \text{skip}(y, V) := \text{void} \) if no such \( z \) exists.

Notice that the nodes of \( V \) are related to \( y \) via \( E_k \) and hence are at distance < \( 3k \) from \( y \). Hence for each \( y \), we only need to consider at most \( d^{O(3k^2)} \) such sets \( V \).

For each set \( V \), \( \text{skip}(y, V) \) can be computed by running consecutively through all nodes \( z \geq y \) in the list \( P(\mathcal{G}) \) and test whether \( z \geq y \) for some \( v \in V \). This can be done in constant time as we have done the preprocessing for testing for all of the \( E_i \).

Since \( |V| \leq k \) and each \( v \in V \) is of degree at most \( d \) in \( \mathcal{G} \), the value \( \text{skip}(y, V) \) can be found in time \( O(k^2 \cdot d) \). Therefore, the entire \( \text{skip} \)-function can be computed, and stored in a data structure by Theorem 1 in time \( O(n^{1+\varepsilon} \cdot d^{O(3k^2)} \cdot g''(|\varphi|, \varepsilon)) \) for some computable function \( g'' \). Later, given \( y \) and \( V \) as above, the value \( \text{skip}(y, V) \) can be looked-up within constant time.

We are now done with the precomputation phase. Altogether it took

1. the time to compute \( \psi \) and \( \mathcal{G} \), which is \( g(|\varphi|) \cdot n \cdot d^{g(|\varphi|)} \), for a computable function \( g \)
2. the time to compute \( \lor_{j \in J} \theta_j \), which is \( g(|\varphi|) \), for a computable function \( g \)
3. for each \( j \in J \) and \( \theta := \theta_j \), it took
   a. the preprocessing time for \( \theta'(\mathcal{G}) \),
      which is by induction \( h(|\theta'|, \varepsilon) \cdot n_i^{1+\varepsilon} \cdot \tilde{d}^{h(|\theta'|, \varepsilon)} \), for the computable function \( h \) in the Proposition’s statement
   b. the time for computing \( P(\mathcal{G}) \), which is \( g(|\varphi|) \cdot \tilde{n} \cdot \tilde{d}^{g(|\varphi|)} \), for a computable function \( g \)
   c. for all \( i \leq k \), the preprocessing time for testing membership in \( E_i(\hat{\mathcal{G}}) \), which can be done in time \( g'(|E_i|, \varepsilon) \cdot n_i^{1+\varepsilon} \cdot d^{\theta'(|E_i|, \varepsilon)} \), for a computable function \( g' \)
   d. and the time for computing the \( \text{skip} \)-function and to compute the associated data structure, which is \( g''(|\varphi|, \varepsilon) \cdot n_i^{1+\varepsilon} \cdot \tilde{d}^{g''(|\varphi|, \varepsilon)} \), for a computable function \( g'' \).

It is straightforward to see that, by suitably choosing the computable function \( h \), all the preprocessing steps can be done within time \( h(|\varphi|, \varepsilon) \cdot n_i^{1+\varepsilon} \cdot dh(|\varphi|, \varepsilon) \).

We now turn to computing the next solution.
Let $\vec{a}$ be a tuple of the active domain of $\mathcal{A}$. We first compute $\vec{u} = f(\vec{a})$, which can be done in constant time according to Proposition 10.

Assume $u = \vec{u}' u$ for some $u$ in the domain of $\mathcal{G}$.

We do the following in order to compute $\vec{v}$, as the next tuple of $\psi(\mathcal{G})$ that is bigger than $\vec{u}$:

1. For all $j \in J$, let $v_j$ be the element returned by the following subroutine:

   (a) Let $w$ be the next element in $P_j(\mathcal{G})$ bigger than $u$. If no such $w$ exists, return $void$.

   (b) Let $V$ to be the set of elements $c \in \vec{u}'$ such that $(w, c) \in E_k(\hat{\mathcal{G}})$. This set can be computed in constant time because the preprocessing for the test for $E_k$ has been done. If $V$ is empty then return $w$.

   (c) return $v_j = \text{skip}(w, V)$, which can be retrieved in constant time due to our preprocessing.

2. Let $v$ be the smallest elements among all the $v_j$. If all the $v_j$ are void then set $v$ to void.

3. If $v$ is void then, by induction, we get in constant time the next solution $\vec{u}''$ of $(\exists x \in \theta)(\mathcal{G})$ bigger than $\vec{u}'$. If this is void, then there is no solution anymore and return void. Otherwise, let $u$ be the first element of the domain and start again at Step 1 with $\vec{u} = \vec{u}'' u$. By construction one of the $v_j$ must now be non void.

   If $v$ is not void then return $\vec{u}' v$.

We prove that the above process returns the correct answer. To see this, let $j$ be such that $v = v_j$ and $w$ be the element computed in Step 1a for that $j$. Let $U$ be the elements of $\vec{u}'$ that are not in the set $V$ computed at step 1b. We know that $v \in P(\hat{\mathcal{G}})$ and that there is no edge in $\hat{\mathcal{G}}$ between $v$ and any element of $U$.

If $U$ is all of $\vec{u}'$, then clearly $\vec{u}' v$ is the next answer of $\psi(\mathcal{G})$ bigger than $\vec{u}$.

If $v = w$ this is also clear as by definition, the elements of $U$ are not connected to $w$ in $\hat{\mathcal{G}}$.

Otherwise $v > w$. We claim that it is still the case that there is also no edge between $v$ and the elements of $U$, hence $\vec{u}' v \in \theta_j(\mathcal{G})$.

Let $v'$ be such that $\text{next}(v') = v$ (possibly $v' = v$). Assume towards a contradiction that there is some $x \in U$ connected in $\hat{\mathcal{G}}$ to $v$. As $v'$ was skipped by skip this means that $(v', y) \in E'(\hat{\mathcal{G}})$ for some $y \in V$. Notice now that for all $c \in V$, $(c, w) \in E_k(\hat{\mathcal{G}})$. This is because when one $E_j$ does not produce anything outside of $E_{j-1}$ then all the $E_{j'}$ for $j' > j$ also do not produce anything outside of $E_{j-1}$. Hence, as $|V| < k$, we must have $(c, w) \in E_{k-1}(\hat{\mathcal{G}})$ and in particular $(w, y) \in E_{k-1}(\hat{\mathcal{G}})$.

Altogether, $v, v', y$ witness the fact that $(w, x) \in E_k(\hat{\mathcal{G}})$ contradicting the fact that $x \in U$.

It is also clear that $\vec{u}' v$ is minimal, relative to $\leq_{\mathcal{G}}$ with these properties.

It is now easy to verify that the desired tuple $\vec{b}$ is then simply $f^{-1}(\vec{v})$ which can be computed in constant time.

From Proposition 16 the enumeration easily follows:

**Proposition 17.** There is an algorithm which at input of a structure $\mathcal{A}$ a first-order query $\varphi(\vec{x})$ and $\varepsilon > 0$ enumerates $\varphi(\mathcal{A})$ with delay $h(\varphi, \varepsilon)$ after a preprocessing of time $h(\varphi) \cdot n^{1+\varepsilon} \cdot d^h(\varphi)$, where $n = |\text{dom}(\mathcal{A})|$, $d = \text{degree}(\mathcal{A})$, and $h$ is a computable function.

**Proof.** Immediate from Proposition 16. It is enough to perform the preprocessing of Proposition 16. For the enumeration phase, we start with the first tuple in lexicographical order. Using Proposition 16 we then obtain the first solution in constant time. At any time during the enumeration phase, once we
Step 1: transform \( \varphi \)

Proof of Proposition 10. We first transform \( \varphi \) into a local formula \( \varphi' \) in Gaifman normal form. The latter turns out to have a small answer set that can be precomputed in pseudo-linear time. The remaining time is used to transform an input formula \( \varphi \) where \( \theta \) we let \( \delta := \frac{\varepsilon}{2h(|\varphi|, \varepsilon/2)} \), where \( h \) is the function of Proposition 17 and define \( n_\varepsilon := m_\delta \). Then, every \( A \in C \) with \( |A| \geq n_\varepsilon \) has degree \( \leq |A|^{\varepsilon h(|\varphi|, \varepsilon) \varepsilon} \). Thus, on input of \( A, \varphi \) and \( \varepsilon/2 \), the enumeration algorithm from Proposition 14 has preprocessing time \( O(h(|\varphi|, \varepsilon/2)|A|^{1+\varepsilon}) \) if \( |A| \geq n_\varepsilon \) and takes time bounded by \( h(|\varphi|, \varepsilon) \cdot n_\varepsilon^{1+\varepsilon/2+2h(|\varphi|, \varepsilon)} \) otherwise and delay time \( h(|\varphi|, \varepsilon) \). This gives the bounds claimed by the proposition with a computable function as soon as we can compute \( n_\varepsilon \).

4 Proof of quantifier elimination and normal form

This section is devoted to the proof of Proposition 10. The proof consists of several steps, the first of which relies on a transformation of \( \varphi(\bar{x}) \) into an equivalent formula in Gaifman normal form, i.e., a boolean combination of basic-local sentences and formulas that are local around \( \bar{x} \). A formula \( \Lambda(\bar{x}) \) is \( r\)-local around \( \bar{x} \) (for some \( r \geq 0 \)) if every quantifier is relativized to the \( r \)-neighborhood of \( \bar{x} \). A basic-local sentence is of the form

\[
\exists y_1 \cdots \exists y_\ell \bigwedge_{1 \leq i < j \leq \ell} dist(y_i, y_j) > 2r \land \bigwedge_{i=1}^\ell \theta(y_i),
\]

where \( \theta(y) \) is \( r \)-local around \( y \). By Gaifman’s well-known theorem we obtain an algorithm that transforms an input formula \( \varphi(\bar{x}) \) into an equivalent formula in Gaifman normal form.

The rest of the proof can be sketched as follows. Basic-local sentences can be evaluated on structures of low degree in pseudo-linear time by Theorem 4 so it remains to treat formulas that are local around their free variables. By the Feferman-Vaught Theorem (cf., e.g. [18]), we can further decompose local formulas into formulas that are local around one of their free variables. The latter turns out to have a small answer set that can be precomputed in pseudo-linear time. The remaining time is used to compute the structures useful for reconstructing the initial answers from their components. We now give the details.

Proof of Proposition 10

Step 1: transform \( \varphi(\bar{x}) \) into a local formula \( \varphi'(\bar{x}) \).

We first transform \( \varphi(\bar{x}) \) into an equivalent formula \( \varphi^G(\bar{x}) \) in Gaifman normal form. For each basic-local sentence \( \chi \) occurring in \( \varphi^G(\bar{x}) \), check whether \( A \models \chi \) and let \( \chi' := true \) if \( A \models \chi \) and \( \chi' := false \) if \( A \models \chi \). Let \( \varphi'(\bar{x}) \) be the formula obtained from \( \varphi^G(\bar{x}) \) by replacing every basic-local sentence \( \chi \) occurring in \( \varphi^G(\bar{x}) \) with \( \chi' \). By using Gaifman’s theorem and Theorem 4 all this can be done in time \( O(h(|\varphi|) \cdot n \cdot d^{\varepsilon h(|\varphi|)}) \), for a computable function \( h \).

Clearly, for every \( \bar{a} \in \text{dom}(A)^k \) we have \( A \models \varphi'(\bar{a}) \iff A \models \varphi(\bar{a}) \). Note that there is a number \( r \geq 0 \) such that \( \varphi'(\bar{x}) \) is \( r \)-local around \( \bar{x} \), and this number can easily be computed given \( \varphi^G(\bar{x}) \).

Step 2: transform \( \varphi'(\bar{x}) \) into a disjunction \( \bigvee_{\psi \in P} \psi_{\bar{x}} \).

Let \( \bar{x} = (x_1, \ldots, x_k) \). A partition of the set \( \{1, \ldots, k\} \) is a list \( P = (P_1, \ldots, P_\ell) \) with \( 1 \leq \ell \leq k \) such that

- \( \emptyset \neq P_j \subseteq \{1, \ldots, k\} \), for every \( j \in \{1, \ldots, \ell\} \),
Let $P_1 \cup \cdots \cup P_\ell = \{1, \ldots, k\}$,

- $P_j \cap P_{j'} = \emptyset$, for all $j, j' \in \{1, \ldots, \ell\}$ with $j \neq j'$,
- $\min P_j < \min P_{j+1}$, for all $j \in \{1, \ldots, \ell-1\}$.

Let $\mathcal{P}$ be the set of all partitions of $\{1, \ldots, k\}$. Clearly, $|\mathcal{P}| \leq k!$. For each $P = (P_1, \ldots, P_\ell) \in \mathcal{P}$ and each $j \leq \ell$ let $\bar{x}_{P_j}$ be the tuple obtained from $\bar{x}$ by deleting all those $x_i$ with $i \notin P_j$.

For every partition $P = (P_1, \ldots, P_\ell) \in \mathcal{P}$ let $\varrho_P(\bar{x})$ be an FO($\sigma$)-formula stating that each of the following is true:

1. The $r$-neighborhood around $\bar{x}$ in $\mathcal{A}$ is the disjoint union of the $r$-neighborhoods around $\bar{x}_{P_j}$ for $j \leq \ell$. I.e., $\delta_P(\bar{x}) := 
   \bigwedge_{1 \leq j < j' \leq \ell} \bigwedge_{(i,i') \in P_j \times P_{j'}} \text{dist}(x_i, x_{i'}) > 2r+1$.

2. For each $j \leq \ell$, the $r$-neighborhood around $\bar{x}_{P_j}$ in $\mathcal{A}$ is connected, i.e., satisfies the formula $\gamma_{P_j}(\bar{x}_{P_j}) := 
   \bigvee_{E \subseteq P_j \times P_j \text{ such that the graph } (P_j, E) \text{ is connected}} \bigwedge_{(i,i') \in E} \text{dist}(x_i, x_{i'}) \leq 2r+1$.

Note that the formula $\varrho_P(\bar{x}) := \delta_P(\bar{x}) \land \bigwedge_{j=1}^{\ell} \gamma_{P_j}(\bar{x}_{P_j})$ is $r$-local around $\bar{x}$. Furthermore, $\varphi'(\bar{x})$ obviously is equivalent to the formula $\bigvee_{P \in \mathcal{P}} \left( \varrho_P(\bar{x}) \land \varphi'(\bar{x}) \right)$.

Using the Feferman-Vaught Theorem (see e.g. [18]), we can, for each $P = (P_1, \ldots, P_\ell) \in \mathcal{P}$, compute a decomposition of $\varphi'(\bar{x})$ into $r$-local formulas $\psi_{P,j,t}(\bar{x}_{P_j})$, for $j \in \{1, \ldots, \ell\}$ and $t \in T_P$, for a suitable finite set $T_P$, such that the formula $\left( \varrho_P(\bar{x}) \land \psi'(\bar{x}) \right)$ is equivalent to $\varrho_P(\bar{x}) \land \bigvee_{t \in T_P} \left( \psi_{P,1,t}(\bar{x}_{P_1}) \land \cdots \land \psi_{P,\ell,t}(\bar{x}_{P_\ell}) \right)$ which, in turn, is equivalent to $\psi_P := (\psi_{P,1} \land \psi_{P,2})$, where $\psi_{P,1} := \delta_P(\bar{x})$ and $\psi_{P,2} := \left( \bigwedge_{j=1}^{\ell} \gamma_{P_j}(\bar{x}_{P_j}) \right) \land \bigvee_{t \in T_P} \left( \bigwedge_{j=1}^{\ell} \psi_{P,j,t}(\bar{x}_{P_j}) \right)$.

In summary, $\varphi'(\bar{x})$ is equivalent to $\bigvee_{P \in \mathcal{P}} \psi_P(\bar{x})$, and for every tuple $\bar{a} \in \text{dom}(\mathcal{A})^k$ with $\mathcal{A} \models \varphi'(\bar{a})$, there is exactly one partition $P \in \mathcal{P}$ such that $\mathcal{A} \models \psi_P(\bar{a})$ (since $\mathcal{A} \models \varrho_P(\bar{a})$ is true for only one such $P \in \mathcal{P}$).

Step 3: defining $G$, $f$, and $\psi$.

We define the domain $G$ of $G$ to be the disjoint union of the sets $A$ and $V$, where $A := \text{dom}(\mathcal{A})$, and $V$ consists of a “dummy element” $v_\perp$, and an element $v_{(b,\ell)}$

- for each $\bar{b} \in A^1 \cup \cdots \cup A^k$ such that $\mathcal{A} \models \gamma_P(\bar{b})$ where $P := \{1, \ldots, |\bar{b}|\}$ and
- for each injective mapping $\ell : \{1, \ldots, |\bar{b}|\} \to \{1, \ldots, k\}$. 


Note that the first item ensures that the $r$-neighborhood around $\bar{b}$ in $A$ is connected. The second item ensures that we can view $\varsigma$ as a description telling us that the $i$-th component of $\bar{b}$ shall be viewed as an assignment for the variable $x_i(\bar{a})$ (for each $i \in \{1, \ldots, |\bar{b}|\}$).

We let $f$ be the function from $A^k$ to $V^k$ defined as follows: For each $\bar{a} \in A^k$ let $P = (P_1, \ldots, P_\ell)$ be the unique element in $\mathcal{P}$ such that $A \models \varphi_P(\bar{a})$. For each $j \in \{1, \ldots, \ell\}$, we write $\bar{a}_{P_j}$ for the tuple obtained from $\bar{a}$ by deleting all those $a_i$ with $i \notin P_j$. Furthermore, we let $t_{P_j}$ be the mapping from $\{1, \ldots, |P_j|\}$ to $\{1, \ldots, k\}$ such that for any $i \in \{1, \ldots, |P_j|\}$, $\iota(i)$ is the $i$-th smallest element of $P_j$. Then, 

$$f(\bar{a}) := (v_{(a_{P_1}, t_{P_1})}, \ldots, v_{(a_{P_\ell}, t_{P_\ell})}, v_\perp, \ldots, v_\perp),$$

where the number of $v_\perp$-components is $(k-\ell)$. It is straightforward to see that $f$ is injective.

We let $\tau_1$ be the signature consisting of a unary relation symbol $C_\perp$, and a unary relation symbol $C_i$ for each injective mapping $\iota : \{1, \ldots, s\} \to \{1, \ldots, k\}$ for $s \in \{1, \ldots, k\}$.

In $\mathcal{G}$, the symbol $C_\perp$ is interpreted by the singleton set $\{v_\perp\}$, and each $C_i$ is interpreted by the set of all nodes $v_{(b, i)} \in V$ with $i = \iota$.

We let $E$ be a binary relation symbol which is interpreted in $\mathcal{G}$ by the set of all tuples $(v_{(b, i)}, v_{(c, j)}) \in V^2$ such that there are elements $b' \in A$ in $\bar{b}$ and $c' \in A$ in $\bar{c}$ such that $\text{dist}^A(b', c') \leq 2r+1$.

For each $P = (P_1, \ldots, P_\ell) \in \mathcal{P}$, each $j \in \{1, \ldots, \ell\}$, and each $t \in T_P$ we let $C_{\perp, t}$ be a unary relation symbol which, in $\mathcal{G}$, is interpreted by the set of all nodes $v_{(b, i)} \in V$ such that $i = t$ and $A \models \varphi_{P_j, t}(\hat{b})$.

We let $\tau_2$ be the signature consisting of all the unary relation symbols $C_{P,j,t}$.

We let $\bar{y} = (y_1, \ldots, y_k)$ be a tuple of $k$ distinct variables, and we define $\psi_1(\bar{y})$ to be the FO($E$)-formula 

$$\psi_1(\bar{y}) := \bigwedge_{1 \leq j, j' \leq k, j \neq j'} \neg E(y_j, y_j').$$

For each $P = (P_1, \ldots, P_\ell)$ we let $\psi_P(\bar{y})$ be the FO($\tau_1 \cup \tau_2$)-formula defined as follows:

$$\psi_P(\bar{y}) := \left( \bigwedge_{j=1}^\ell C_{\perp, t}(y_j) \right) \land \left( \bigwedge_{j=\ell+1}^k C_i(y_j) \right) \land \bigvee_{t \in T_P} \left( \bigwedge_{j=1}^\ell C_{P,j,t}(y_j) \right).$$

It is straightforward to verify that the following is true:

1. For every $\bar{a} \in A^k$ with $A \models \psi_P'(\bar{a})$, we have $\mathcal{G} \models (\psi_1 \land \psi_P)(f(\bar{a})).$

2. For every $\bar{v} \in G^k$ with $G \models (\psi_1 \land \psi_P)(\bar{v})$, there is a (unique) tuple $\bar{a} \in A^k$ with $\bar{v} = f(\bar{a})$, and for this tuple we have $A \models \psi_P'(\bar{a})$.

Finally, we let 

$$\psi(\bar{y}) := (\psi_1(\bar{y}) \land \psi_2(\bar{y})) \quad \text{with} \quad \psi_2(\bar{y}) := \bigvee_{P \in \mathcal{P}} \psi_P(\bar{y}).$$

It is straightforward to see that $f$ is a bijection between $\varphi(A)$ and $\psi(G)$.

In summary, we now know that items $\square$ and $\square$ as well as the non computational part of item $\square$ of Proposition $\square$ are true.
Towards the second statement of item \[4\], we use additional binary relation symbols $F_1, \ldots, F_k$ which are interpreted in $G$ as follows: Start by initializing all of them to the empty set. Then, for each $v = v_{(b, c)} \in V$ and each $j \in \{1, \ldots, |b|\}$, add to $F^G_{(j)}$ the tuple $(v, a)$, where $a$ is $j$-th component of $b$. This completes the definition of $G$ and $\tau$, letting $\tau := \tau_1 \cup \tau_2 \cup \{E, F_1, \ldots, F_k\}$.

Using the relations $F_1, \ldots, F_k$ of $G$, in time $O(k)$ we can, upon input of $v = v_{(b, c)} \in V$ compute the tuple $\bar{b}$ and the mapping $\iota$ (for this, just check for all $i \in \{1, \ldots, k\}$ whether node $v$ has an outgoing $F_i$-edge). Using this, it is straightforward to see that upon input of $\bar{v} \in dom(G)$, the tuple $f^{-1}(\bar{v}) \in A^k$ can be computed in time $O(k^2)$.

**Step 4: Computing $G$ within the time bounds of Item $3$**

First of all, note that for each $v_{(b, c)} \in V$, the tuple $\bar{b}$ is of the form $(b_1, \ldots, b_s) \in A^s$ for some $s \leq k$, such that all components of the tuple belong to the $\hat{r}$-neighborhood $N^A_r(b_1)$ of $b_1$ in $A$, for $\hat{r} := k(2r+1)$. Since $A$ has degree $d$, $N^A_r(b_1)$ contains at most $d^s + 1$ elements of $A$. And by using breadth-first search, $N^A_r(b_1)$ can be computed in time $d^O(\hat{r}+1)$.

Thus, the set $V$, along with the relations $C_\perp, C_i$ and $F_1, \ldots, F_k$ of $G$, can be computed as follows: Start by letting $V := \{v_\perp\}$ and initializing all relations to the empty set. Let $C^G_\perp := \{v_\perp\}$. Then, for each $a \in A$, compute the $\hat{r}$-neighborhood $N^A_r(a)$ of $a$ in $A$, and compute (by a brute-force algorithm), for each $s \in \{1, \ldots, k\}$, the set of all $s$-tuples $b$ of elements from this neighborhood, which satisfy the following: The first component of $b$ is $a$, and $N^A_r(a) \models \gamma P_j(b)$ for $P_j = \{1, \ldots, s\}$. For each such tuple $b$ do the following: For each injective mapping $\iota : \{1, \ldots, s\} \rightarrow \{1, \ldots, k\}$ add to $V$ a new element $v_{(b, c)}$, add this element to the relation $C^G_\iota$, and for each $j \in \{1, \ldots, s\}$, add to $F^G_{(j)}$ the tuple $(v_{(b, c)}, a)$, where $a$ is the $j$-th component of $b$.

This way, the domain $G = A \cup V$ of $G$, along with the relations $C_i$ and $F_1, \ldots, F_k$ of $G$, can be computed in time $O(h(|\varphi|) \cdot n \cdot d^h(|\varphi|))$, for a computable function $h$.

For computing the unary relations $C_{P, j, t}$ of $G$, start by initializing all of them to the empty set. For each $v_{(b, c)} \in V$ do the following: Compute (by using the relations $F_1, \ldots, F_k$) the tuple $\bar{b}$ and the mapping $\iota$. Let $a$ be the first component of $\bar{b}$. Compute the $\hat{r}$-neighborhood $N^A_r(a)$ of $a$ in $A$, for $\hat{r} := \hat{r} + r$. For each $P = \{P_1, \ldots, P_{\ell}\} \in P$, each $j \in \{1, \ldots, \ell\}$ such that $P_j = \iota$ and each $t \in T_P$, check whether $N^A_r(a) \models \theta_{P, j, t}(\bar{b})$. If so, add the element $v_{(b, c)}$ to the relation $C_{P, j, t}$ of $G$. (This is correct, since the formula $\theta_{P, j, t}$ is $r$-local around its free variables, and the radius of the neighborhood is large enough.)

This way, $G$’s relations $C_{P, j, t}$ can be computed in time $O(h(|\varphi|) \cdot n \cdot d^h(|\varphi|))$, for a computable function $h$.

To compute the $E$-relation of $G$, note that for all tuples $(v_{(b, c)}, v_{(\bar{c}, \bar{d})}) \in E^G$, we have $dist^A(a, c_j) \leq (2k+1)(2r+1)$, for all components $c_j$ of $\bar{c}$, where $a$ is the first component of $\bar{b}$. Thus, the $E$-relation of $G$ can be computed as follows: Start by initializing this relation to the empty set. For each $v_{(b, c)} \in V$ do the following: Compute (by using the relations $F_1, \ldots, F_k$) the tuple $\bar{b}$. Let $a$ be the first component of $\bar{b}$. Compute the $\hat{r}$-neighborhood $N^A_r(a)$ of $a$ in $A$, for $\hat{r} := r + (2k+1)(2r+1)$. Use a brute-force algorithm to compute all tuples $\bar{c}$ of elements in $N^A_{\hat{r}}(a)$, such that $|\bar{c}| \leq k$ and $N^A_r(a) \models \gamma P_j(\bar{c})$ for $P_j = \{1, \ldots, |\bar{c}|\}$. Check if there are components $b'$ of $\bar{b}$ and $c'$ of $\bar{c}$ such that $dist^A_r(b', c') \leq 2r+1$. If so, add to $E^G$ the tuple $(v_{(b, c)}, v_{(\bar{c}, \bar{d})})$ for each injective mapping $\iota : \{1, \ldots, |\bar{c}|\} \rightarrow \{1, \ldots, k\}$.

This way, the $E$-relation of $G$ can be computed in time $O(h(|\varphi|) \cdot n \cdot d^h(|\varphi|))$, for a computable function $h$.

In summary, we obtain that $G$ is computable from $A$ and $\varphi$ within the desired time bound.

To finish the proof of item 3 we need to give an upper bound on the degree of $G$. As noted above,
\((v(\hat{b},a),v(\hat{c},i)) \in E^G\) implies that \(\text{dist}^A(a,c_j) \leq r'\) for \(r' := (2k+1)(2r+1)\), for all components \(c_j\) of \(\hat{c}\), where \(a\) is the first component of \(\hat{b}\). Thus, for each fixed \(v(\hat{b},a) \in V\), the number of elements \(v(\hat{c},i)\) such that \((v(\hat{b},a),v(\hat{c},i)) \in E^G\) is at most
\[
k! \cdot \sum_{s=1}^{k} |N_i^A(a)|^s \leq k! \cdot |N_i^A(a)|^{k+1} \leq k! \cdot d(r'+1)(k+1).
\]

Thus, since \(E^G\) is symmetric, its degree is \(\leq 2k!d(r'+1)(k+1)\).

Similarly, for each tuple \((v(\hat{b},a),a) \in F_i^G\) (with \(i \in \{1,\ldots,k\}\)) we know that \(a\) is the \(\epsilon^{-1}(i)\)-th component of \(\hat{b}\) and each component of \(\hat{b}\) belongs to the \(\hat{r}\)-neighborhood of \(a\) in \(A\), for \(\hat{r} = k(2r+1)\). Thus, for each fixed \(a \in A\), the number of elements \(v(\hat{b},a) \in V\) such that \((v(\hat{b},a),a) \in F_i^G\) is at most \(k! \cdot \sum_{s=1}^{k} |N_i^A(a)|^s \leq k! \cdot \hat{d}(r'+1)(k+1)\). In summary, we thus obtain that \(G\) is of degree at most \(d^h(|\varepsilon|)\) for a computable function \(h\).

**Step 5: Computing \(f\) within the time bounds of Item 4**

Recall that for \(\bar{a} \in A^k\) we have
\[
f(\bar{a}) := (v(\bar{a}_{P_j},\bar{t}_{P_j}),\ldots,v(\bar{a}_{P_1},\bar{t}_{P_1}),v_{\perp},\ldots,v_{\perp}),
\]
for the unique partition \(P = (P_1,\ldots,P_\ell) \in \mathcal{P}\) such that \(A \models g_P(\bar{a})\). The number of \(v_{\perp}\)-components in \(f(\bar{a})\) is \((k-\ell)\).

We first show how to compute \(f(\bar{a})\) from \(\bar{a}\) in constant time. This where we use \(\varepsilon\).

To compute the partition \(P\) for a given tuple \(\bar{a} = (a_1,\ldots,a_k)\), we can proceed as follows: Construct an undirected graph \(H\) with vertex set \(\{1,\ldots,k\}\), where there is an edge between \(i \neq j\) iff \(\text{dist}^A(a_i,a_j) \leq 2r+1\). This can be done as follows. Let \(R\) be the binary relation over \(\text{dom}(A)\) containing all pairs \((a,b)\) such that \(\text{dist}^A(a_i,a_j) \leq 2r+1\). As \(A\) as degree \(d\) the size of \(R\) if bounded by \(n \cdot d^{2r+2}\) and \(R\) can be computed by a brute force algorithm in time \(O(n \cdot d^{2r+2})\). Hence by Theorem 1 we can compute a data structure in time \(O(n^{1+\varepsilon} \cdot d^{2r+2})\) such that afterwards we can test in time depending only on \(\varepsilon\) whether a given pair is in \(R\) or not.

Once \(H\) is computed we can compute its connected components in time depending only in \(k\). Let \(\ell\) be the number of connected components of \(H\). For each \(j \in \{1,\ldots,\ell\}\) let \(P_j\) be vertex set of the \(j\)-th connected component, such that \(\min P_j < \min P_{j+1}\) for all \(j \in \{1,\ldots,\ell-1\}\). After having constructed the partition \(P = (P_1,\ldots,P_\ell)\), further \(O(k^2)\) steps suffice to construct the tuples \(\bar{a}_{P_1},\ldots,\bar{a}_{P_\ell}\), the mappings \(\bar{t}_{P_1},\ldots,\bar{t}_{P_\ell}\), and the according tuple \(f(\bar{a})\). Let \(\zeta_P\) be the function associating to each pair \((\bar{a}_{P_j},\bar{t}_{P_j})\) the element \(v(\bar{a}_{P_j},\bar{t}_{P_j})\). The domain of \(\zeta_P\) is at most \(n \cdot d^{k(2r+1)+1}\) and \(\zeta_P\) can be computed in \(O(n \cdot d^{k(2r+1)+1})\) by a brute force algorithm, hence using Theorem 1 we can compute in time \(O(n^{1+\varepsilon} \cdot d^{k(2r+1)+1})\) a data structure such that afterwards we can obtain the result of the function \(\zeta_P\) in time depending only on \(\varepsilon\).

Altogether, after the preprocessing, we can compute \(f(\bar{a})\) in time \(O(k^2)\).

Recall that using the relation \(F\), it is straightforward to compute \(f^{-1}(\bar{v})\) upon input of \(\bar{v} \in \text{dom}(G)\) in time \(O(k^2)\).

This concludes the proof of proof of Proposition 10.

5 Conclusion

For classes of databases of low degree, we presented an algorithm which enumerates the answers to first-order queries with constant delay after pseudo-linear preprocessing. An inspection of the proof
shows that the constants involved are non-elementary in the query size. In the bounded degree case the constants are triply exponential in the query size [14]. In the (unranked) tree case the constants are provably non-elementary [9] (modulo some complexity assumption). We do not know what is the situation for classes of low degree.

If the database is updated, for instance if a tuple is deleted or inserted, it would be desirable to be able to update efficiently the data structure that is computed for deriving in constant time counting, testing and enumeration. With the data structure given in this paper it is not clear how to do this without recomputing everything from scratch. However it has been shown recently that there is another data structure, providing the same constant time properties that furthermore can be updated in time $O(n^\epsilon)$ upon insertion or deletion of a tuple [21].

References


