HAL
open science

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

Arnaud Durand, Nicole Schweikardt, Luc Segoufin

To cite this version:

Arnaud Durand, Nicole Schweikardt, Luc Segoufin. Enumerating Answers to First-Order Queries over Databases of Low Degree. Logical Methods in Computer Science, 2022, 18 (2), pp.23. 10.46298/lmcs-18(2:7)2022 . hal-03809756

HAL Id: hal-03809756

https://inria.hal.science/hal-03809756v1

Submitted on 10 Oct 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

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

Arnaud Durand, *IMJ-PRG, Université de Paris, CNRS*
Nicole Schweikardt, *Humboldt-Universität zu Berlin*
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^δ, 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 $\mathcal{A}$ and a first-order query q, it may be not satisfactory enough to express complexity results in terms of the sizes of $\mathcal{A}$ and q as it is often the case. The fact that the solution set $q(\mathcal{A})$ 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 $\mathcal{A}$). 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 [6, 15], over the class of structures of "bounded expansion" [17] and, more generally, over the class of nowhere dense

[0] This is the extended version of the conference contribution [7].

structures [20]. Similar results have been shown for monadic second-order logic over structures of bounded tree-width [4, 1, 16] 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 [23]).

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^δ, 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 [13] that over a class of databases of low degree, first-order boolean queries can be checked in pseudo-linear time, i.e., for all $\varepsilon > 0$ there is an algorithm running in time bounded by $O(n^{1+\varepsilon})$ 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 $\mathcal{A}$ is more general than testing whether q holds on $\mathcal{A}$, 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 $f(q)\cdot\|\mathcal{A}\|^c$ 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 classes of databases of bounded degree, or bounded treewidth, FPT algorithms were derived for classes of databases having bounded expansion [8] (see also [17]). Actually, recently an FPT algorithm has been obtained for classes of databases known as "nowhere dense", generalizing all the previously known results [14]. 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 [18], 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 [13]. They generalize the known enumeration algorithms for databases of bounded degree [6, 15].

This paper is the journal version of [7]. There is an important difference 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 2.1. This makes the technical lemma slightly more complicated to state, but does not affect the general results.

Organization. We fix the basic notation and formulate our main results in Section 2. In Section 3 we present the algorithms for counting, testing, and enumerating answers to first-order queries over classes of structures of low degree. These algorithms rely on a particular preprocessing which transforms a first-order query on a database into a quantifier-free query on a colored graph. The result is stated in Section 3.3, while its proof is presented in Section 4. We conclude in Section 5.

2 Preliminaries and Main Results

We write $\mathbb{N}$ to denote the set of non-negative integers, and we let $\mathbb{N}_{\geqslant 1} := \mathbb{N} \setminus \{0\}$. $\mathbb{Q}$ denotes the set of rationals, and $\mathbb{Q}_{>0}$ is the set of positive rationals.

2.1 Databases and queries

A database is a finite relational structure. A *relational signature* σ is a finite set of relation symbols R, each of them associated with a fixed *arity* $ar(R) \in \mathbb{N}_{\geqslant 1}$. A *relational structure* $\mathcal{A}$ over σ, or a σ-structure (we omit to mention σ when it is clear from the context) consists of a non-empty finite set $dom(\mathcal{A})$ called the *domain* of $\mathcal{A}$, and an $ar(R)$-ary relation $R^{\mathcal{A}} \subseteq dom(\mathcal{A})^{ar(R)}$ for each relation symbol $R \in \sigma$.

The degree of a structure $\mathcal{A}$, denoted $degree(\mathcal{A})$, is the degree of the Gaifman graph associated with $\mathcal{A}$ (i.e., the undirected graph with vertex set $dom(\mathcal{A})$ where there is an edge between two nodes if they both occur in a tuple that belongs to a relation of $\mathcal{A}$). With this definition, in a structure with n domain elements and of degree d, each r-ary relation may have at most $n \cdot (d{+}1)^{r-1}$ tuples.

In the sequel we only consider structures of degree $\geqslant 2$. As structures of degree 1 are quite trivial, this is without loss of generality.

We define the *size* $\|\mathcal{A}\|$ of $\mathcal{A}$ as $\|\mathcal{A}\| = |\sigma| + |dom(\mathcal{A})| + \sum_{R\in\sigma} |R^{\mathcal{A}}|{\cdot}ar(R)$. It corresponds to the size of a reasonable encoding of $\mathcal{A}$. We assume that the input structure $\mathcal{A}$ is presented in a way such that given a relation symbol R in the signature we can directly access the list of tuples in $R^{\mathcal{A}}$ (i.e. without reading the remaining tuples). The cardinality of $\mathcal{A}$, i.e. the cardinality of its domain, is denoted by $|\mathcal{A}|$.

By *query* we mean a formula of FO(σ), the set of all first-order formulas of signature σ, for some relational signature σ (again we omit σ when it is clear from the context). For $\varphi \in$ FO, we write $\varphi(\bar{x})$ to denote a query whose free variables are $\bar{x}$, 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 φ, an *answer* to φ in $\mathcal{A}$ is a tuple $\bar{a}$ of elements of $dom(\mathcal{A})$ such that $\mathcal{A} \models \varphi(\bar{a})$. In the special case where φ is a sentence, it is either true of false in $\mathcal{A}$, and the former is denoted $\mathcal{A} \models \varphi$ and the latter is denoted $\mathcal{A} \not\models \varphi$. We write $\varphi(\mathcal{A})$ for the set of answers to φ in $\mathcal{A}$, i.e. $\varphi(\mathcal{A}) = \{\bar{a} \,:\, \mathcal{A} \models \varphi(\bar{a})\}$. As usual, $|\varphi|$ denotes the size of φ.

Let C be a class of structures. The model checking problem of FO over C is the computational problem of given a **sentence** $\varphi \in$ FO and a database $\mathcal{A} \in C$ to test whether $\mathcal{A} \models \varphi$.

Given a k-ary query φ, we care about "enumerating" $\varphi(\mathcal{A})$ efficiently. Let C be a class of structures. The *enumeration problem of φ over C* is, given a database $\mathcal{A} \in C$, to output the elements of $\varphi(\mathcal{A})$ 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 $\varphi(\mathcal{A})$ 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 [9, 12].

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 φ of size k and a structure $\mathcal{A}$ 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 f 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 stores our functions in a more efficient way. The data structure is a trie of depth $\frac{1}{\varepsilon}$ and of degree n^ε where each pair (key,value) is a tuple $\bar{a}$ in the domain of f and its image $b = f(\bar{a})$. The details can be found in [21].

Theorem 2.1 (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 dom$(f) \subseteq [n]^k$ with:*

- *computation time and storage space $O(|dom(f)| \cdot n^\varepsilon)$,*
- *lookup time only depending on k and ε,*

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

An important consequence of Theorem 2.1 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 2.2. *Let $\mathcal{A}$ be a database over the schema σ. Let $\varepsilon > 0$. One can compute in time $O(d^r n^{1+\varepsilon})$ a data structure such that on input of a tuple $\bar{a}$ and a relation symbol $R \in \sigma$ one can test whether $\mathcal{A} \models R(\bar{a})$ in time $O(1)$, where $n = |dom(\mathcal{A})|$, $d = degree(\mathcal{A})$, and r is a number that only depends on σ.*

Proof. Immediate from Theorem 2.1, as the number of tuples in an r-ary relation of a σ-structure whose Gaifman graph has degree d is at most $(d+1)^{r-1}n$. □

Note that a simple linear time preprocessing would provide a data structure allowing for a test as in Corollary 2.2 in time $O(d)$ (to see how, just think of the special case where $\mathcal{A}$ only contains one binary relation R. In this case, the preprocessing can build an adjacency list representation where upon input of an a we can access in time $O(1)$ the first element of a list of all those b satisfying $R(a, b)$. This list has length at most d. Upon input of a tuple (a, b) we access a's adjacency list and check if it contains b). With the help of the Storing Theorem we get a test in constant time, i.e. depending only on ε and σ 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 $\varepsilon > 0$, $q \in$ FO and $\mathcal{A} \in C$, can be decomposed into two phases:

- a preprocessing phase that is performed in time $f(\varepsilon, |q|) \cdot \|\mathcal{A}\|^{1+\varepsilon}$ for some function f, and
- an enumeration phase that outputs $q(\mathcal{A})$ with no repetition and a delay depending only on q, ε, 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, ε 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 $g(|q|, \varepsilon) \cdot (\|\mathcal{A}\|^{1+\varepsilon} + |q(\mathcal{A})|)$, 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 2.3. *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*

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

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 2.2, we can test in constant time whether there is an edge between any two nodes. The problem with this 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 by a fixed constant, 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 $\mathcal{A}$ of C have degree at most $|\mathcal{A}|^{\delta}$ (see [13]). More formally, C has low degree if for every $\delta \in \mathbb{Q}_{>0}$ there is an $n_\delta \in \mathbb{N}_{\geqslant 1}$ such that all structures $\mathcal{A} \in C$ of cardinality $|\mathcal{A}| \geqslant n_\delta$ have *degree*$(\mathcal{A}) \leqslant |\mathcal{A}|^{\delta}$. If there is a computable function associating n_δ from δ 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 $\mathcal{A} \in C$ 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 classes with known low evaluation complexity such as the "nowhere dense" case mentioned in the introduction.

Notice that $\mathit{degree}(\mathcal{A}) \leqslant |\mathcal{A}|^\delta$ implies that $\|\mathcal{A}\| \leqslant c{\cdot}|\mathcal{A}|^{1+\delta\cdot r}$, where r is the maximal arity of the signature and c is a number only depending on σ.

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 2.4 (Grohe [13])**.** *There is a computable function h such that on input of a structure $\mathcal{A} \in C$ of degree d and a sentence $q \in$ FO, one can test in time $h(|q|){\cdot}|\mathcal{A}|{\cdot}d^{h(|q|)}$ whether $\mathcal{A} \models q$.*

In particular, if C is a class of structures of low degree, then there is a function g such that, given a structure $\mathcal{A} \in C$, a sentence $q \in$ FO, and $\varepsilon > 0$, one can check if $\mathcal{A} \models q$ in time $g(|q|, \varepsilon){\cdot}|\mathcal{A}|^{1+\varepsilon}$. If C is effective then g is computable.

2.4 Our results

We are now ready to state our main results, which essentially lift Theorem 2.4 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 2.5. *Let C be a class of structures of low degree. There is a function g such that, given a structure $\mathcal{A} \in C$, a query $q \in$ FO, and $\varepsilon > 0$, one can compute $|q(\mathcal{A})|$ in time $g(|q|, \varepsilon){\cdot}|\mathcal{A}|^{1+\varepsilon}$. 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 2.6. *Let C be a class of structures of low degree. There is a function g such that, given a structure $\mathcal{A} \in C$, a query $q \in$ FO, and $\varepsilon > 0$, one can compute in time $g(|q|, \varepsilon){\cdot}|\mathcal{A}|^{1+\varepsilon}$ a data structure such that, on input of any $\bar{a}$, one can then test in time $g(|q|, \varepsilon)$ whether $\bar{a} \in q(\mathcal{A})$. 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 2.7. *Let C be a class of structures of low degree. There is a function g such that, given a structure $\mathcal{A} \in C$, a query $q \in$ FO and $\varepsilon > 0$, the enumeration problem of q over $\mathcal{A}$ can be solved with delay $g(|q|, \varepsilon)$ after a preprocessing running in time $g(|q|, \varepsilon){\cdot}|\mathcal{A}|^{1+\varepsilon}$. 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 $\mathcal{A}$ we write $\mathit{dist}^{\mathcal{A}}(a, b)$ for the distance between two nodes a and b of the Gaifman graph of $\mathcal{A}$. For an element $a \in \mathit{dom}(\mathcal{A})$ and a number $r \in \mathbb{N}$, the *r-ball* around a is the set $N_r^{\mathcal{A}}(a)$ of all nodes $b \in \mathit{dom}(\mathcal{A})$ with $\mathit{dist}^{\mathcal{A}}(a, b) \leqslant r$. The *$r$-neighborhood* around a is the induced substructure $\mathcal{N}_r^{\mathcal{A}}(a)$ of $\mathcal{A}$ on $N_r^{\mathcal{A}}(a)$. Note that if $\mathcal{A}$ is of degree $\leqslant d$ for $d \geqslant 2$, then $|N_r^{\mathcal{A}}(a)| \leqslant \sum_{i=0}^{r} d^i < d^{r+1}$.

3 Evaluation algorithms

In this section, we present our algorithms for counting, testing, and enumerating the solutions to a query (see Sections 3.4, 3.5, and 3.6). 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 3.3). However, before presenting these algorithms, we start with very simple cases.

3.1 Computing the neighborhoods

Unsurprisingly, all our algorithms will start by computing the neighborhood $\mathcal{N}_r^{\mathcal{A}}(a)$ for all the elements a of the input structure $\mathcal{A}$ for a suitable constant r depending only on q. We actually do not need to compute $\mathcal{N}_r^{\mathcal{A}}(a)$ but only $\mathcal{N}_r^{\mathcal{A}\downarrow q}(a)$ where $\mathcal{A}\downarrow q$ is the restriction of $\mathcal{A}$ to the relational symbols occurring in q.

The next lemma states that all these neighborhoods can be computed in reasonable time.

Lemma 3.1. *There is an algorithm which, at input of a structure $\mathcal{A}$, a query q and a number r computes $\mathcal{N}_r^{\mathcal{A}\downarrow q}(a)$ for all elements a of the domain of $\mathcal{A}$ in time $O(|q| \cdot n \cdot d^{h(r,|q|)})$, where $n = |dom(\mathcal{A})|$, $d = degree(\mathcal{A})$ and h is a computable function.*

Proof. We first compute the Gaifman graph associated to $\mathcal{A}\downarrow q$. We consider each relation symbol R occurring in q. We scan $R^{\mathcal{A}}$ and for each of its tuples we add the corresponding clique to the Gaifman graph. As each element a of $dom(\mathcal{A})$ can appear at the first component of at most $(d+1)^{ar(R)-1}$ tuples of $R^{\mathcal{A}}$, the total time is $|q| \cdot n \cdot (d+1)^{k-1}$ where k is the maximal arity of the relation symbols occurring in q. This yields a graph with n nodes and degree d corresponding to the Gaifman graph of $\mathcal{A}\downarrow q$. From this graph we easily derive, in time $O(n \cdot d)$, a structure associating to each node $a \in dom(\mathcal{A})$ the set of its immediate neighbors, i.e $N_1^{\mathcal{A}\downarrow q}(a)$.

With r steps of transitive closure computation we get a structure associating to each node $a \in dom(\mathcal{A})$ the set $N_r^{\mathcal{A}\downarrow q}(a)$ of nodes at distance at most r from a. This can be done in time $O(n \cdot d^r)$.

With an extra scan over the database we can derive $\mathcal{N}_r^{\mathcal{A}\downarrow q}(a)$ from $N_r^{\mathcal{A}\downarrow q}(a)$. Altogether we get the desired time bounds. □

3.2 Connected conjunctive queries

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 γ which is a conjunction of relational atoms and potentially negated *unary* atoms. Note that the query of Example 2.3 is not a conjunction as it has a binary negated atom. With each conjunction γ we associate a *query graph* H_γ. This is the undirected graph whose vertices are the variables $x_1, \ldots, x_k$ of γ, and where there is an edge between two vertices x_i and x_j iff γ contains a relational atom in which both x_i and x_j occur. We call the conjunction γ *connected* if its query graph H_γ is connected.

A *connected conjunctive query* is a query $q(\bar{x})$ of the form $\exists \bar{y}\, \gamma(\bar{x}, \bar{y})$, where γ 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 3.3, and in the proofs for our counting and enumeration results in Sections 3.4 and 3.6.

Lemma 3.2. *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 $n = |dom(\mathcal{A})|$, $d = degree(\mathcal{A})$, 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 γ. Let $k = |\bar{x}|$ be the number of free variables of q, let $\ell = |\bar{y}|$, and let $r = k + \ell$.

In view of Lemma 3.1 we can assume that all the neighborhoods $\mathcal{N}^{\mathcal{A}\downarrow q}(a)$ have been computed. In order to simplify the notations, in the rest of the proof we will write $\mathcal{N}_r^{\mathcal{A}}(a)$ instead of $\mathcal{N}_r^{\mathcal{A}\downarrow q}(a)$.

Note that since γ 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 $\mathcal{N}_r^{\mathcal{A}}(a)$ of a in $dom(\mathcal{A})$. Thus, $q(\mathcal{A})$ is the disjoint union of the sets

$$S_a \;:=\; \big\{\, \bar{b} \in q(\mathcal{N}_r^{\mathcal{A}}(a)) \;:\; \text{the first component of } \bar{b} \text{ is } a \,\big\},$$

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

(1) Initialize $S_a := \emptyset$.

(2) Use a brute-force algorithm that enumerates all k-tuples $\bar{b}$ of elements in $\mathcal{N}_r^{\mathcal{A}}(a)$ whose first component is a.

For each such tuple $\bar{b}$, use a brute-force algorithm that checks whether $\mathcal{N}_r^{\mathcal{A}}(a) \models q(\bar{b})$. If so, insert $\bar{b}$ into S_a

Note that the number of considered tuples $\bar{b}$ is $\leqslant d^{(r+1)(k-1)}$. And checking whether $\mathcal{N}_r^{\mathcal{A}}(a) \models q(\bar{b})$ can be done in time $O(|\gamma|{\cdot}d^{(r+1)\ell})$: for this, enumerate all ℓ-tuples $\bar{c}$ of elements in $\mathcal{N}_r^{\mathcal{A}}(a)$ 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 $q(\mathcal{A}) = \bigcup_{a\in A} S_a$ in time $O(n{\cdot}|q|{\cdot}d^{h(|q|)})$, for a computable function h. □

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

Proof. We use the algorithm provided in Lemma 3.2. 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 $m_\delta \in \mathbb{N}_{\geqslant 1}$ such that every structure $\mathcal{A} \in C$ of cardinality $|\mathcal{A}| \geqslant m_\delta$ has $degree(\mathcal{A}) \leqslant |\mathcal{A}|^\delta$.

For a given $\varepsilon > 0$ we let $\delta := \frac{\varepsilon}{h(|q|)}$ and define $n_\varepsilon := m_\delta$. Then, every $\mathcal{A} \in C$ with $|\mathcal{A}| \geqslant n_\varepsilon$ has $degree(\mathcal{A}) \leqslant |\mathcal{A}|^{\varepsilon/h(|q|)}$. Thus, on input of $\mathcal{A}$ and q, the algorithm from Lemma 3.2 has running time $O(|q|{\cdot}|\mathcal{A}|^{1+\varepsilon})$ if $|\mathcal{A}| \geqslant n_\varepsilon$ and takes time bounded by $O(|q|{\cdot}n_\varepsilon^{1+h(q)})$ otherwise. This gives the bounds claimed by the proposition with a computable function g as soon as we can compute n_ε. □

The method of the proof of Proposition 3.3 above will be used several times in the paper.

3.3 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 3.4. *There is an algorithm which, at input of a structure $\mathcal{A}$, a first-order query $\varphi(\bar{x})$, and $\varepsilon > 0$, produces a binary signature τ (containing, among other symbols, a binary relation symbol E), a colored graph $\mathcal{G}$ of signature τ, an* FO(τ)*-formula $\psi(\bar{x})$, a mapping f, and a data structure such that the following is true for $k = |\bar{x}|$, $n = |dom(\mathcal{A})|$, $d = degree(\mathcal{A})$ and h some computable function:*

1. *ψ is quantifier-free. Furthermore, ψ is of the form $(\psi_1 \wedge \psi_2)$, where ψ_1 states that no distinct free variables of ψ are connected by an E-edge, and ψ_2 is a positive boolean combination of unary atoms.*

2. *τ and ψ are computed in time and space $h(|\varphi|)\cdot n\cdot d^{h(|\varphi|)}$.*
   Moreover, $|\tau| \leqslant h(|\varphi|)$ and $|\psi| \leqslant h(|\varphi|)$.

3. *$\mathcal{G}$ is computed in time and space $h(|\varphi|)\cdot n\cdot d^{h(|\varphi|)}$.*
   Moreover, $degree(\mathcal{G}) \leqslant d^{h(|\varphi|)}$.

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

   The data structure representing f can be computed in time and space $h(|\varphi|)\cdot n^{1+\varepsilon}\cdot d^{h(|\varphi|)}$ and can then be used as follows: on input of any tuple $\bar{a} \in dom(\mathcal{A})^k$, the tuple $f(\bar{a})$ can be computed in time $O(k^2)$; and on input of any tuple $\bar{v} \in \psi(\mathcal{G})$, the tuple $f^{-1}(\bar{v})$ can be computed in time $O(k^2)$.

The proof of Proposition 3.4 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.

3.4 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 2.3 is a generalized conjunction).

Example 3.5. *Before moving to the formal proof of Theorem 2.5, consider again the query q from Example 2.3. 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 $q'(x, y)$ returning the set of blue-red nodes that* are *connected. In other words, q' is*

$$B(x) \wedge R(y) \wedge E(x, y).$$

Notice that this query is a connected conjunction. Hence, by Proposition 3.3 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 2.5 goes as follows. Using Proposition 3.4 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 3.5. The resulting formula is then a connected conjunction that is treated in pseudo-linear time using Proposition 3.3.

Lemma 3.6. *There is an algorithm which, at input of a colored graph $\mathcal{G}$ and a generalized conjunction $\gamma(\bar{x})$, computes $|\gamma(\mathcal{G})|$ 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 γ, $n = |dom(\mathcal{G})|$, and $d = degree(\mathcal{G})$.*

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

In case that H_γ is connected, we can use Lemma 3.2 to compute the entire set $\gamma(\mathcal{G})$ in time $O(|\gamma|{\cdot}n{\cdot}d^{h(|\gamma|)})$, for a computable function h. Thus, counting $|\gamma(\mathcal{G})|$ can be done within the same time bound.

In case that γ 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 γ that contain variables in H_i. Note that $\gamma(\bar{x})$ is equivalent to $\bigwedge_{i=1}^{\ell} \gamma_i(\bar{x}_i)$, and

$$|\gamma(\mathcal{G})| \;=\; \prod_{i=1}^{\ell} |\gamma_i(\mathcal{G})|.$$

Since each γ_i is connected, we can compute $|\gamma_i(\mathcal{G})|$ in time $O(|\gamma_i|{\cdot}n{\cdot}d^{h(|\gamma_i|)})$ by using the algorithm of Lemma 3.2. 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 γ be a formula with $m{+}1$ negated binary atoms. Let $\neg R(x, y)$ be a negated binary atom of γ, and let γ_1 be such that

$$\begin{array}{rcl} \gamma & = & \gamma_1 \;\wedge\; \neg R(x,y), \qquad \text{and let} \\ \gamma_2 & := & \gamma_1 \;\wedge\; R(x,y). \end{array}$$

Clearly, $|\gamma(\mathcal{G})| = |\gamma_1(\mathcal{G})| - |\gamma_2(\mathcal{G})|$. Since each of the formulas γ_1 and γ_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 3.4, we can lift this to arbitrary structures and first-order queries:

Proposition 3.7. *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 $n = |dom(\mathcal{A})|$ and $d = degree(\mathcal{A})$.*

Proof. We first use the algorithm of Proposition 3.4 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 $|\psi| \leqslant h(|\varphi|)$. By Proposition 3.4 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 γ_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 γ_i has length at most $|\psi|$, and $|I| \leqslant 2^{|\psi|}$.

Obviously, $|\psi(\mathcal{G})| = \sum_{i\in I} |\gamma_i(\mathcal{G})|$. We now use, for each $i \in I$, the algorithm from Lemma 3.6 to compute the number $s_i = |\gamma_i(\mathcal{G})|$ and output the value $s = \sum_{i\in I} s_i$.

By Lemma 3.6 we know that for each $i \in I$ computing s_i can be done in time $O(2^m{\cdot}|\gamma_i|{\cdot}\tilde{n}{\cdot}\tilde{d}^{h_0(|\gamma_i|)})$, where m is the number of binary atoms in γ, $\tilde{n} = |\mathit{dom}(\mathcal{G})|$, $\tilde{d} = \mathit{degree}(\mathcal{G})$, and h_0 is some computable function.

By Proposition 3.4 we know that $\tilde{n} \leqslant h(|\varphi|){\cdot}n{\cdot}d^{h(|\varphi|)}$ and $\tilde{d} \leqslant d^{h(|\varphi|)}$. Since also $|\gamma_i| \leqslant |\psi| \leqslant h(|\varphi|)$, 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 $|I| \leqslant 2^{|\psi|}$, 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 2.5 is an immediate consequence of Proposition 3.7, following arguments similar with the proof of Proposition 3.3: For a given $\varepsilon > 0$ we let $\delta := \frac{\varepsilon}{h(|\varphi|)}$, where h is the function of Proposition 3.7, and define $n_\varepsilon := m_\delta$. Then, every $\mathcal{A} \in C$ with $|\mathcal{A}| \geqslant n_\varepsilon$ has $\mathit{degree}(\mathcal{A}) \leqslant |\mathcal{A}|^{\varepsilon/h(|\varphi|)}$. Thus, on input of $\mathcal{A}$ and φ, the algorithm from Proposition 3.7 has running time $O(h(|\varphi|){\cdot}|\mathcal{A}|^{1+\varepsilon})$ if $|\mathcal{A}| \geqslant n_\varepsilon$ and takes time bounded by $h(|\varphi|) \cdot n_\varepsilon^{1+h(|\varphi|)}$ otherwise. This gives the bounds claimed by the proposition with a computable function g as soon as we can compute n_ε.

3.5 Testing

Here we consider the problem of testing whether a given tuple is a solution to a query. By Proposition 3.4 it is enough to consider quantifier-free formulas. Those are treated using the data structure computed by Corollary 2.2.

Proposition 3.8. *There is an algorithm which at input of a structure $\mathcal{A}$, a first-order query $\varphi(\bar{x})$, and an $\varepsilon > 0$ has a preprocessing phase of time $g(|\varphi|,\varepsilon){\cdot}n^{1+\varepsilon}{\cdot}d^{g(|\varphi|,\varepsilon)}$ in which it computes a data structure such that, on input of any $\bar{a} \in \mathit{dom}(\mathcal{A})^k$ for $k = |\bar{x}|$, it can be tested in time $g(|\varphi|,\varepsilon)$ whether $\bar{a} \in \varphi(\mathcal{A})$, where g is a computable function, $n = |\mathit{dom}(\mathcal{A})|$, and $d = \mathit{degree}(\mathcal{A})$.*

Proof. Fix $\varepsilon > 0$.

We first use the algorithm of Proposition 3.4 to compute the graph $\mathcal{G}$, the quantifier-free formula $\psi(\bar{x})$ and the data structure for function f. For some computable function h, all of this is done within time $h(|\varphi|){\cdot}n^{1+\varepsilon}{\cdot}d^{h(|\varphi|)}$, and furthermore, $|\psi| \leqslant h(|\varphi|)$ and $\mathit{degree}(\mathcal{G}) \leqslant d^{h(|\varphi|)}$. Note that $\|\mathcal{G}\| \leqslant h(|\varphi|){\cdot}n{\cdot}d^{h(|\varphi|)}$. By construction, we furthermore know for all $\bar{a} \in \mathit{dom}(\mathcal{A})^k$ that $\bar{a} \in \varphi(\mathcal{A}) \iff f(\bar{a}) \in \psi(\mathcal{G})$.

Recall from Proposition 3.4 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 $\bar{v} \in \mathit{dom}(\mathcal{G})^k$ belongs to $\psi(\mathcal{G})$ can be done easily, provided that one can check whether unary atoms $C(u)$ and binary atoms $E(u,u')$ hold in $\mathcal{G}$ for given nodes u, u' of $\mathcal{G}$.

Let $\tilde{n} = |dom(\mathcal{G})|$ and $\tilde{d} = degree(\mathcal{G})$. Recall that $\tilde{n} \leqslant h(|\varphi|) \cdot n \cdot d^{h(|\varphi|)}$ and $\tilde{d} \leqslant d^{h(|\varphi|)}$. We apply Corollary 2.2 to $\mathcal{G}$ and ε.

This gives an extra preprocessing time of $O(\tilde{d}^{\tilde{r}} \cdot \tilde{n}^{1+\varepsilon})$, where $\tilde{r}$ only depends on τ, i.e., $\tilde{r}$ is bounded by $\tilde{h}(|\varphi|)$ for some computable function $\tilde{h}$. Inserting the known bounds on $\tilde{r}$, $\tilde{n}$, and $\tilde{d}$ shows that this extra preprocessing time is in $O(g(|\varphi|, \varepsilon) \cdot n^{1+\varepsilon} \cdot d^{g(|\varphi|,\varepsilon)})$ for some computable function g.

Finally, the testing algorithm works as follows. Given a tuple $\bar{a} \in dom(\mathcal{A})^k$, we first construct $\bar{v} := f(\bar{a})$ and then check whether $\bar{v} \in \psi(\mathcal{G})$. Building $\bar{v} := f(\bar{a})$ can be done in time $O(k^2)$ (see Proposition 3.4), and by Corollary 2.2 checking whether $\bar{v} \in \psi(\mathcal{G})$ depends only on ψ, ε and σ. Hence, we meet the required bound for testing. □

Theorem 2.6 is an immediate consequence of Proposition 3.8 using the usual argument: For a given query ϕ and an $\varepsilon > 0$ we let $\delta := \frac{\varepsilon}{2g(|\varphi|,\varepsilon/2)}$, where g is the function of Proposition 3.8, and define $n_\varepsilon := m_\delta$. Then, every $\mathcal{A} \in C$ with $|\mathcal{A}| \geqslant n_\varepsilon$ has $degree(\mathcal{A}) \leqslant |\mathcal{A}|^{\varepsilon/2g(|\varphi|,\varepsilon/2)}$. Thus, on input of $\mathcal{A}$, φ and $\varepsilon/2$, the testing algorithm from Proposition 3.8 has preprocessing time $O(g(|\varphi|, \varepsilon/2) \cdot |\mathcal{A}|^{1+\varepsilon})$ if $|\mathcal{A}| \geqslant n_\varepsilon$ and takes time bounded by $g(|\varphi|, \varepsilon/2) \cdot n_\varepsilon^{1+\varepsilon/2+g(|\varphi|,\varepsilon/2)}$ otherwise, and it has testing time $O(g(|\varphi|, \varepsilon/2)$. This gives the bounds claimed by the theorem with a computable function as soon as we can compute n_ε.

3.6 Enumeration

Here we consider the problem of enumerating the solutions to a given query. We first illustrate the proof of Theorem 2.7 with our running example.

Example 3.9. *Consider again the query q of Example 2.3. 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 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 3.3 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 2.7 can be sketched as follows. By Proposition 3.4 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 2.3 corresponds to the binary case. For queries of larger arities we proceed by induction on the arity. If q is given by the formula $\varphi(\bar{x}y)$ we know by induction that, modulo some preprocessing pseudo-linear in the size of the input database D, we can enumerate with constant delay all tuples $\bar{a}$ satisfying $D \models \exists y \varphi(\bar{x}y)$. For each such tuple $\bar{a}$ it remains to enumerate all b such that $D \models \varphi(\bar{a}b)$. We then proceed as in Example 3.9. Starting from an arbitrary node

b of the appropriate color, we iterate the following reasoning. If the current node b is not connected to $\bar{a}$, then $\bar{a}b$ forms an answer and we proceed to the next b. If b is connected to $\bar{a}$ then we need to jump in constant time to the next node of the appropriate color forming a solution. 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 3.9. The design and computation of this function is the main technical originality of the proof. The fact that the database has low degree implies that for each tuple $\bar{a}$ there are few nodes b that are connected to $\bar{a}$. This makes the computation efficient.

The technical details are summarized in the following proposition.

Proposition 3.10. *There is an algorithm which at input of a structure $\mathcal{A}$, a first-order query $\varphi(\bar{x})$, and $\varepsilon > 0$ enumerates $\varphi(\mathcal{A})$ with delay $h(|\varphi|, \varepsilon)$ after a preprocessing of time $h(|\varphi|, \varepsilon) \cdot n^{1+\varepsilon} \cdot d^{h(|\varphi|,\varepsilon)}$, where $n = |dom(\mathcal{A})|$, $d = degree(\mathcal{A})$, and h is a computable function.*

Proof. The proof is by induction on the number $k := |\bar{x}|$ of free variables of φ. In case that $k = 0$, the formula φ is a sentence, and we are done using Theorem 2.4. In case that $k > 0$ we proceed as follows.

We first use the algorithm of Proposition 3.4 to compute the according colored graph $\mathcal{G}$, the quantifier-free formula $\psi(\bar{x})$, and the data structure representing f. This takes time $g(|\varphi|) \cdot n^{1+\varepsilon} \cdot d^{g(|\varphi|)}$ for a computable function g. And we know that $|\psi| \leqslant g(|\varphi|)$, that $\mathcal{G}$ has degree $\tilde{d} \leqslant d^{g(|\varphi|)}$, and that $dom(\mathcal{G})$ has $\tilde{n}$ elements, where $\tilde{n} \leqslant g(|\varphi|) \cdot n \cdot d^{g(|\varphi|)}$.

From Item 1 of Proposition 3.4 we know that the formula $\psi(\bar{x})$ is of the form $(\psi_1 \wedge \psi_2)$, where ψ_1 states that no distinct free variables of ψ are connected by an E-edge and ψ_2 is a positive boolean combination of unary atoms.

We now prove the Proposition in the case of $\mathcal{G}$, i.e. we enumerate $\psi(\mathcal{G})$ and go back to $\mathcal{A}$ using f in constant time by Item 4 of Proposition 3.4. We assume an arbitrary linear order $\leqslant_{\mathcal{G}}$ among the nodes of $\mathcal{G}$.

In case that $k = 1$, $\psi(x_1) = \psi_2(x_1)$ is a positive boolean combination of unary atoms. We can use Lemma 3.2 for each unary atom in order to compute $\psi(\mathcal{G})$ in time $O(|\psi| \cdot \tilde{n} \cdot \tilde{d}^{g(|\psi|)})$ for a computable function g. From this the constant delay enumeration is immediate.

Altogether the preprocessing time is in $h(\varphi) \cdot n^{1+\varepsilon} \cdot d^{h(\varphi)}$, for a computable function h as required. The delay is $O(1)$, and we are done for $k = 1$.

The case $k > 1$ requires much more elaborate constructions.

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

$$\phi_j(\bar{x}_{k-1}) \ \wedge \ P_j(x_k) \ \wedge \ \gamma(\bar{x}), \qquad \text{where}$$

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

$P_j(x_k)$ 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 ψ.

As the θ_j are mutually exclusive, we can enumerate $\psi(\mathcal{G})$ by enumerating for each $j \in J$, $\theta_j(\mathcal{G})$.

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

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

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

By induction hypothesis (since θ' only has $k{-}1$ free variables) we can enumerate $\theta'(\mathcal{G})$ with delay $h(|\theta'|,\varepsilon)$ and preprocessing $h(|\theta'|,\varepsilon){\cdot}\tilde{n}^{1+\varepsilon}{\cdot}\tilde{d}^{h(|\theta'|,\varepsilon)}$

Since $P(x_k)$ is a boolean combination of unary atoms on x_k, we can use Lemma 3.2 to compute $P(\mathcal{G})$ in time $O(|P|{\cdot}\tilde{n}{\cdot}\tilde{d}^{g(|P|)})$ for a computable function g. Afterwards, we have available a list of all nodes v of $\mathcal{G}$ that belong to $P(\mathcal{G})$. In the following, we will write $\leqslant^P_{\mathcal{G}}$ to denote the linear ordering of $P(\mathcal{G})$ induced by $\leqslant_{\mathcal{G}}$ on $P(\mathcal{G})$, and we write $\mathit{first}^P_{\mathcal{G}}$ for the first element in this list, and $\mathit{next}^P_{\mathcal{G}}$ for the successor function, such that for any node $v \in P(\mathcal{G})$, $\mathit{next}^P_{\mathcal{G}}(v)$ is the next node in $P(\mathcal{G})$ in this list (or the value *void*, if v is the last node in the list).

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

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 $(E(x,y) \vee E(y,x))$. The reason for defining these formulas will become clear only later on, in the proof.

$$E_1(u,y) \ := \ E'(u,y), \qquad \text{and}$$

$$E_{i+1}(u,y) \ := \ E_i(u,y) \ \ \vee \ \ \exists z \exists z' \exists v \left(E'(z,u) \wedge \mathit{next}(z',z) \wedge E'(v,z') \wedge E_i(v,y)\right).$$

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

In our algorithm we will have to test, given $i \leqslant k$ and nodes $u,v \in \mathit{dom}(\mathcal{G})$, whether $(u,v) \in E_i(\hat{\mathcal{G}})$. Since E_i is a first-order formula, Proposition 3.8 implies that, after a preprocessing phase using time $g'(|E_i|,\varepsilon){\cdot}\tilde{n}^{1+\varepsilon}{\cdot}\hat{d}^{g'(|E_i|,\varepsilon)}$ (for some computable function g'), testing membership in $E_i(\hat{\mathcal{G}})$, for any given $(u,v) \in \mathit{dom}(\mathcal{G})^2$, is possible within time $g'(|E_i|,\varepsilon)$.

The last step of the precomputation phase computes the function *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 $\leqslant^P_{\mathcal{G}}$ of $P(\mathcal{G})$) element $z \geqslant^P_{\mathcal{G}} y$ 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 $0 \leqslant |V| < k$ and $(v,y) \in E_k(\hat{\mathcal{G}})$ for all $v \in V$, we let

$$\mathit{skip}(y,V) \ := \ \min\{z \in P(\mathcal{G}) \ : \ y \leqslant^P_{\mathcal{G}} z \ \text{ and } \ \forall v \in V : \ (v,z) \notin E'(\mathcal{G})\},$$

respectively, $\mathit{skip}(y,V) := \mathit{void}$ 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 $\hat{d}^{(3k^2)}$ such sets V.

For each set V, $\mathit{skip}(y,V)$ can be computed by running consecutively through all nodes $z \geqslant^P_{\mathcal{G}} y$ in the list $P(\mathcal{G})$ and test whether $E'(z,v)$ holds 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 $|V| \leqslant k$ and each $v \in V$ is of degree at most $\tilde{d}$ in $\mathcal{G}$, the value $\mathit{skip}(y,V)$ can be found in time $O(k^2{\cdot}\tilde{d})$. Therefore, the entire *skip*-function can be computed, and stored in a data structure by Theorem 2.1, in time $O(\tilde{n}^{1+\varepsilon}{\cdot}\hat{d}^{(3k^2)}{\cdot}g''(|\varphi|,\varepsilon))$ for some computable function g''. Later, given y and V as above, the value $\mathit{skip}(y,V)$ can be looked-up within constant time.

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

1. the time to compute ψ and $\mathcal{G}$, which is $g(|\varphi|){\cdot}n^{1+\varepsilon}{\cdot}d^{g(|\varphi|)}$, for a computable function g

2. the time to compute $\bigvee_{j\in J}\theta_j$, 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 $h(|\theta'|,\varepsilon)\cdot\tilde{n}^{1+\varepsilon}\cdot\tilde{d}^{h(|\theta'|,\varepsilon)}$, 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 $i \leqslant k$, the preprocessing time for testing membership in $E_i(\hat{\mathcal{G}})$, which can be done in time $g'(|E_i|,\varepsilon)\cdot\tilde{n}^{1+\varepsilon}\cdot\hat{d}^{g'(|E_i|,\varepsilon)}$, for a computable function g'
   (d) and the time for computing the *skip*-function and to compute the associated data structure, which is $g''(|\varphi|,\varepsilon)\cdot\tilde{n}^{1+\varepsilon}\cdot\tilde{d}^{g''(|\varphi|,\varepsilon)}$, 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 $h(|\varphi|,\varepsilon)\cdot n^{1+\varepsilon}\cdot d^{h(|\varphi|,\varepsilon)}$.

We now turn to the enumeration procedure. As expected we enumerate all tuples $\bar{u} \in \psi(\mathcal{G})$ and return $f^{-1}(\bar{u})$.

In order to enumerate $\psi(\mathcal{G})$ it suffices to enumerate $\theta_j(\mathcal{G})$ for each j. Fix j and let $\theta = \theta_j$ and θ' be as in the preprocessing phase. The enumeration of $\theta(\mathcal{G})$ is done as follows.

1. Let $\bar{u}$ be the first output produced in the enumeration of $\theta'(\mathcal{G})$.
   If $\bar{u} =$ *void* then STOP with output *void*,
   else let $(u_1,\ldots,u_{k-1}) = \bar{u}$ and goto line 2.
2. Let $y :=$ *first*$^P_{\mathcal{G}}$ be the first element in the list $P(\mathcal{G})$.
3. Let $V := \{v \in \{u_1,\ldots,u_{k-1}\} \,:\, (v,y) \in E_k(\hat{\mathcal{G}})\}$.
4. Let $z :=$ *skip*(y,V).
5. If $z \neq$ *void* then OUTPUT $(\bar{u},z)$ and goto line 9.
6. If $z =$ *void* then
   7. Let $\bar{u}'$ be the next output produced in the enumeration of $\theta'(\mathcal{G})$.
   8. If $\bar{u}' =$ *void* then STOP with output *void*,
      else let $\bar{u} := \bar{u}'$ and goto line 2.
9. Let $y :=$ *next*$^P_{\mathcal{G}}(z)$.
10. If $y =$ *void* then goto line 7, else goto line 3.

We prove that the above process enumerates $\theta(\mathcal{G})$ with constant delay.

To see this, notice first that the algorithm never outputs any tuple more than once. Before proving that this algorithm enumerates exactly the tuples in $\theta(\mathcal{G})$, let us first show that it operates with delay at most $h(|\varphi|,\varepsilon)$.

By the induction hypothesis, the execution of line 1 and each execution of line 7 takes time at most $h(|\theta'|, \varepsilon)$. Furthermore, each execution of line 3 takes time $(k-1)\cdot g'(|E_k|, \varepsilon)$. Concerning the remaining lines of the algorithm, each execution can be done in time $O(1)$.

Furthermore, before outputting the first tuple, the algorithm executes at most 5 lines (namely, lines 1–5; note that by our choice of the formula θ' we know that when entering line 5 before outputting the first tuple, it is guaranteed that $z \neq$ *void*, hence an output tuple is generated).

Between outputting two consecutive tuples, the algorithm executes at most 12 lines (the worst case is an execution of lines 9, 10, 3, 4, 5, 6, 7, 8, 2, 3, 4, 5; again, by our choice of the formula θ', at the last execution of line 5 it is guaranteed that $z \neq$ *void*, hence an output tuple is generated).

Therefore, by suitably choosing the function h, we obtain that the algorithm enumerates with delay at most $h(|\varphi|, \varepsilon)$.

We now show that any tuple outputed is a solution.

To see this consider a tuple $\bar{u}z$ outputed at step 5. By construction we have $z \in P(\hat{\mathcal{G}})$ and $\bar{u} \in \phi(\hat{\mathcal{G}})$. Hence in order to show that $\bar{u}z \in \theta(\hat{\mathcal{G}})$, it remains to verify that z is not connected to any of the elements in $\bar{u}$.

By definition of *skip* it is clear that z is not connected to the elements of V. Assume now that $z = y$. Then by definition of V, z is also not connected to all the elements not in V and we are done.

We can therefore assume that $z >^P_{\mathcal{G}} y$. Let z' be the predecessor of z in P (i.e. *next*$(z') = z$, possibly $z' = y$). Assume towards a contradiction that z is connected to an element x of $\bar{u}$. From the remark above, we know that $x \notin V$. As z' was skipped by *skip* this means that $(z', v) \in E'(\mathcal{G})$ for some $v \in V$. Consider $c \in V$. By definition of V we have $(c, y) \in E_k(\hat{\mathcal{G}})$. It turns out that $(c, y) \in E_{k-1}(\hat{\mathcal{G}})$. 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 $(c, y) \in E_{k-1}(\hat{\mathcal{G}})$ and in particular $(v, y) \in E_{k-1}(\hat{\mathcal{G}})$. Altogether, z, z', v witness the fact that $(x, y) \in E_k(\hat{\mathcal{G}})$ contradicting the fact that x is not in V.

It remains to show that all tuples are outputed. This is done by induction. By induction we know that we consider all relevant $\bar{u}$. By definition, the function *skip* skips only elements y such that $\bar{u}y$ is not a solution. Therefore we eventually output all solutions. □

Theorem 2.7 follows immediately from Proposition 3.10 using the usual argument: For a given $\varepsilon > 0$ we let $\delta := \frac{\varepsilon}{2h(|\varphi|,\varepsilon/2)}$, where h is the function of Proposition 3.10, and define $n_\varepsilon := m_\delta$. Then, every $\mathcal{A} \in C$ with $|\mathcal{A}| \geqslant n_\varepsilon$ has *degree*$(\mathcal{A}) \leqslant |\mathcal{A}|^{\varepsilon/2h(|\varphi|,\varepsilon)}$. Thus, on input of $\mathcal{A}$, φ and $\varepsilon/2$, the enumeration algorithm from Proposition 3.8 has preprocessing time $O(h(|\varphi|, \varepsilon/2)\cdot|\mathcal{A}|^{1+\varepsilon})$ if $|\mathcal{A}| \geqslant n_\varepsilon$ and takes time bounded by $h(|\varphi|, \varepsilon/2) \cdot n_\varepsilon^{1+\varepsilon/2+h(|\varphi|,\varepsilon/2)}$ 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_ε.

4 Proof of quantifier elimination and normal form

This section is devoted to the proof of Proposition 3.4. 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 $r \geqslant 0$) if every quantifier is relativized to the r-neighborhood of $\bar{x}$. A

basic-local sentence is of the form

$$\exists y_1 \cdots \exists y_\ell \bigwedge_{1 \leqslant i < j \leqslant \ell} \mathit{dist}(y_i, y_j) > 2r \ \wedge \bigwedge_{i=1}^{\ell} \theta(y_i),$$

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 [11].

The rest of the proof can be sketched as follows. Basic-local sentences can be evaluated on classes of structures of low degree in pseudo-linear time by Theorem 2.4, so it remains to treat formulas that are local around their free variables. By the Feferman-Vaught Theorem (cf., e.g. [19]), 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 3.4.
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 χ occurring in $\varphi^G(\bar{x})$, check whether $\mathcal{A} \models \chi$ and let $\chi' :=$ *true* if $\mathcal{A} \models \chi$ and $\chi' :=$ *false* if $\mathcal{A} \not\models \chi$. Let $\varphi'(\bar{x})$ be the formula obtained from $\varphi^G(\bar{x})$ by replacing every basic-local sentence χ occurring in $\varphi^G(\bar{x})$ with χ'. By using Gaifman's theorem and Theorem 2.4, all this can be done in time $O(h(|\varphi|) \cdot n \cdot d^{h(|\varphi|)})$, for a computable function h.

Clearly, for every $\bar{a} \in \mathit{dom}(\mathcal{A})^k$ we have $\mathcal{A} \models \varphi'(\bar{a})$ iff $\mathcal{A} \models \varphi(\bar{a})$. Note that there is a number $r \geqslant 0$ such that $\varphi'(\bar{x})$ is r-local around $\bar{x}$, and this number is provided as a part of the output of Gaifman's algorithm.

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

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 $1 \leqslant \ell \leqslant k$ such that

- $\emptyset \neq P_j \subseteq \{1, \ldots, k\}$, for every $j \in \{1, \ldots, \ell\}$,
- $P_1 \cup \cdots \cup P_\ell = \{1, \ldots, k\}$,
- $P_j \cap P_{j'} = \emptyset$, 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, $|\mathcal{P}| \leqslant k!$. For each $P = (P_1, \ldots, P_\ell) \in \mathcal{P}$ and each $j \leqslant \ell$ 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(σ)-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 $j \leqslant \ell$. I.e.,
$$\delta_P(\bar{x}) \ := \bigwedge_{1 \leqslant j < j' \leqslant \ell} \ \bigwedge_{(i,i') \in P_j \times P_{j'}} \mathit{dist}(x_i, x_{i'}) > 2r{+}1.$$

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

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

Using the Feferman-Vaught Theorem (see e.g. [19]), we can, for each $P = (P_1, \dots, P_\ell) \in \mathcal{P}$, compute a decomposition of $\varphi'(\bar{x})$ into r-local formulas $\vartheta_{P,j,t}(\bar{x}_{P_j})$, for $j \in \{1, \dots, \ell\}$ and $t \in T_P$, for a suitable finite set T_P, such that the formula $\big(\varrho_P(\bar{x}) \wedge \varphi'(\bar{x})\big)$ is equivalent to

$$\varrho_P(\bar{x}) \ \wedge \ \bigvee_{t\in T_P} \ \big(\ \vartheta_{P,1,t}(\bar{x}_{P_1}) \ \wedge \ \cdots \ \wedge \ \vartheta_{P,\ell,t}(\bar{x}_{P_\ell}) \ \big)$$

which, in turn, is equivalent to $\psi'_P := (\psi'_{P,1} \wedge \psi'_{P,2})$, where $\psi'_{P,1} := \delta_P(\bar{x})$ and

$$\psi'_{P,2} \ := \ \Big(\bigwedge_{j=1}^{\ell} \gamma_{P_j}(\bar{x}_{P_j}) \Big) \ \wedge \ \bigvee_{t\in T_P} \Big(\bigwedge_{j=1}^{\ell} \vartheta_{P,j,t}(\bar{x}_{P_j}) \Big).$$

In summary, $\varphi'(\bar{x})$ is equivalent to $\bigvee_{P\in\mathcal{P}} \psi'_P(\bar{x})$, and for every tuple $\bar{a} \in \mathit{dom}(\mathcal{A})^k$ 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 $\mathcal{G}$, f, and ψ.

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

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

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

We let f be the function from A^k to V^k defined as follows: For each $\bar{a} \in A^k$ let $P = (P_1, \dots, P_\ell)$ be the unique element in $\mathcal{P}$ such that $\mathcal{A} \models \varrho_P(\bar{a})$. For each $j \leqslant \{1, \dots, \ell\}$, 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 ι_{P_j} be the mapping from $\{1, \dots, |P_j|\}$ to $\{1, \dots, k\}$ such that $\iota(i)$ is the i-th smallest element of P_j, for any $i \in \{1, \dots, |P_j|\}$. Then,

$$f(\bar{a}) \ := \ \big(v_{(\bar{a}_{P_1},\iota_{P_1})}, \dots, v_{(\bar{a}_{P_\ell},\iota_{P_\ell})}, v_\perp, \dots, v_\perp \big),$$

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

We let τ_1 be the signature consisting of a unary relation symbol $C_\perp$, and a unary relation symbol C_ι for each injective mapping $\iota : \{1, \dots, s\} \to \{1, \dots, k\}$ for $s \in \{1, \dots, k\}$.

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

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

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

We let τ_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

$$\psi_1(\bar{y}) \;:=\; \bigwedge_{\substack{1 \leqslant j,j' \leqslant k \\ \text{with } j \neq j'}} \neg E(y_j, y_{j'}).$$

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:

$$\psi_P(\bar{y}) \;:=\; \Big(\bigwedge_{j=1}^{\ell} C_{\iota_{P_j}}(y_j)\Big) \;\wedge\; \Big(\bigwedge_{j=\ell+1}^{k} C_\perp(y_j)\Big) \;\wedge\; \bigvee_{t \in T_P} \Big(\bigwedge_{j=1}^{\ell} C_{P,j,t}(y_j)\Big).$$

It is straightforward to verify that the following is true:

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

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

Finally, we let

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

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

In summary, we now know that items 1 and 2, as well as the non computational part of item 4 of Proposition 3.4 are true.

Later on in *Step 5* we will provide details on how to build a data structure that, upon input of any tuple $\bar{a} \in A^k$, returns the tuple $f(\bar{a})$ within the claimed time bounds.

In order to be able to also compute $f^{-1}(\bar{v})$ upon input of any tuple $\bar{v} \in \psi(\mathcal{G})$, we use additional binary relation symbols $F_1, \ldots, F_k$ which are interpreted in $\mathcal{G}$ as follows: Start by initializing all of them to the empty set. Then, for each $v = v_{(\bar{b},\iota)} \in V$ and each $j \in \{1, \ldots, |\bar{b}|\}$, add to $F^{\mathcal{G}}_{\iota(j)}$ the tuple (v, a), where a is j-th component of $\bar{b}$. This completes the definition of $\mathcal{G}$ and τ, letting $\tau := \tau_1 \cup \tau_2 \cup \{E, F_1, \ldots, F_k\}$.

Using the relations $F_1, \ldots, F_k$ of $\mathcal{G}$, in time $O(k)$ we can, upon input of $v = v_{(\bar{b},\iota)} \in V$ compute the tuple $\bar{b}$ and the mapping ι (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 $\bar{v} \in \psi(\mathcal{G})$, the tuple $f^{-1}(\bar{v}) \in A^k$ can be computed in time $O(k^2)$,

Step 4: Computing $\mathcal{G}$ within the time bounds of Item 3.

First of all, note that for each $v_{(\bar{b},\iota)} \in V$, the tuple $\bar{b}$ is of the form $(b_1, \ldots, b_s) \in A^s$ for some $s \leqslant k$, such that all components of the tuple belong to the $\hat{r}$-neighborhood $\mathcal{N}^{\mathcal{A}}_{\hat{r}}(b_1)$ of b_1 in $\mathcal{A}$, for $\hat{r} := k(2r{+}1)$.

By Lemma 3.1 we can compute in total time $O(|\varphi| \cdot n \cdot d^{h'(|\varphi|)})$ all the neigborhoods $\mathcal{N}^{\mathcal{A}}_{\hat{r}}(a)$ (for all $a \in \mathit{dom}(\mathcal{A})$), where h' is some computable function (recall that $\hat{r}$ depends only on φ). Within the same time bound, we can also compute all the neighborhoods $\mathcal{N}^{\mathcal{A}}_{\tilde{r}}(a)$, $\mathcal{N}^{\mathcal{A}}_{r'}(a)$, and $\mathcal{N}^{\mathcal{A}}_{r'-r}(a)$, for $\tilde{r} := \hat{r}{+}r$ and $r' := r + (2k{+}1)(2r{+}1)$ (later on it will be convenient to have efficient access to all these neighborhoods).

Thus, the set V, along with the relations $C_\perp$, C_ι and $F_1, \ldots, F_k$ of $\mathcal{G}$, can be computed as follows: Start by letting $V := \{v_\perp\}$ and initializing all relations to the empty set. Let $C^{\mathcal{G}}_\perp := \{v_\perp\}$. Then, for each $a \in A$, consider the $\hat{r}$-neighborhood $\mathcal{N}^{\mathcal{A}}_{\hat{r}}(a)$ of a in $\mathcal{A}$, and compute (by a brute-force algorithm), for each $s \in \{1, \ldots, k\}$, the set of all s-tuples $\bar{b}$ of elements from this neighborhood, which satisfy

the following: The first component of $\bar{b}$ is a, and $\mathcal{N}^{\mathcal{A}}_{\hat{r}}(a) \models \gamma_{P_j}(\bar{b})$ for $P_j = \{1,\ldots,s\}$. For each such tuple $\bar{b}$ do the following: For each injective mapping $\iota : \{1,\ldots,s\} \to \{1,\ldots,k\}$ add to V a new element $v_{(\bar{b},\iota)}$, add this element to the relation $C^{\mathcal{G}}_{\iota}$, and for each $j \in \{1,\ldots,s\}$, add to $F^{\mathcal{G}}_{\iota(j)}$ the tuple $(v_{(\bar{b},\iota)}, a)$, where a is the j-th component of $\bar{b}$.

This way, the domain $G = A \cup V$ of $\mathcal{G}$, along with the relations C_ι and $F_1,\ldots,F_k$ of $\mathcal{G}$, 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 $\mathcal{G}$, start by initializing all of them to the empty set. For each $v_{(\bar{b},\iota)} \in V$ do the following: Compute (by using the relations $F_1,\ldots,F_k$) the tuple $\bar{b}$ and the mapping ι. Let a be the first component of $\bar{b}$. Consider the $\tilde{r}$-neighborhood $\mathcal{N}^{\mathcal{A}}_{\tilde{r}}(a)$ of a in $\mathcal{A}$, for $\tilde{r} := \hat{r}+r$. For each $P = (P_1,\ldots,P_\ell) \in \mathcal{P}$, each $j \in \{1,\ldots,\ell\}$ such that $\iota_{P_j} = \iota$, and each $t \in T_P$, check whether $\mathcal{N}^{\mathcal{A}}_{\tilde{r}}(a) \models \theta_{P,t,j}(\bar{b})$. If so, add the element $v_{(\bar{b},\iota)}$ to the relation $C_{P,j,t}$ of $\mathcal{G}$. (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, $\mathcal{G}$'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 $\mathcal{G}$, note that for all tuples $(v_{(\bar{b},\iota)}, v_{(\bar{c},\hat{\iota})}) \in E^{\mathcal{G}}$, we have $\mathit{dist}^{\mathcal{A}}(a, c_j) \leqslant (2k{+}1)(2r{+}1)$, for all components c_j of $\bar{c}$, where a is the first component of $\bar{b}$. Thus, the E-relation of $\mathcal{G}$ can be computed as follows: Start by initializing this relation to the empty set. For each $v_{(\bar{b},\iota)} \in V$ do the following: Compute (by using the relations $F_1,\ldots,F_k$) the tuple $\bar{b}$. Let a be the first component of $\bar{b}$. Consider the r'-neighborhood $\mathcal{N}^{\mathcal{A}}_{r'}(a)$ of a in $\mathcal{A}$, for $r' := r + (2k{+}1)(2r{+}1)$. Use a brute-force algorithm to compute all tuples $\bar{c}$ of elements in $\mathcal{N}^{\mathcal{A}}_{r'-r}(a)$, such that $|\bar{c}| \leqslant k$ and $\mathcal{N}^{\mathcal{A}}_{r'}(a) \models \gamma_{P_j}(\bar{c})$ for $P_j = \{1,\ldots,|\bar{c}|\}$. Check if there are components b' of $\bar{b}$ and c' of $\bar{c}$ such that $\mathit{dist}^{\mathcal{N}^{\mathcal{A}}_{r'}(a)}(b', c') \leqslant 2r{+}1$. If so, add to $E^{\mathcal{G}}$ the tuple $(v_{(\bar{b},\iota)}, v_{(\bar{c},\hat{\iota})})$ for each injective mapping $\hat{\iota} : \{1,\ldots,|\bar{c}|\} \to \{1,\ldots,k\}$.

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

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

To finish the proof of item 3, we need to give an upper bound on the degree of $\mathcal{G}$. As noted above, $(v_{(\bar{b},\iota)}, v_{(\bar{c},\hat{\iota})}) \in E^{\mathcal{G}}$ implies that $\mathit{dist}^{\mathcal{A}}(a, c_j) \leqslant r'$ for $r' := (2k{+}1)(2r{+}1)$, for all components c_j of $\bar{c}$, where a is the first component of $\bar{b}$. Thus, for each fixed $v_{(\bar{b},\iota)} \in V$, the number of elements $v_{(\bar{c},\hat{\iota})}$ such that $(v_{(\bar{b},\iota)}, v_{(\bar{c},\hat{\iota})}) \in E^{\mathcal{G}}$ is at most

$$k! \cdot \sum_{s=1}^{k} |\mathcal{N}^{\mathcal{A}}_{r'}(a)|^s \quad \leqslant \quad k! \cdot |\mathcal{N}^{\mathcal{A}}_{r'}(a)|^{k+1} \quad \leqslant \quad k! \cdot d^{(r'+1)(k+1)}.$$

Thus, since $E^{\mathcal{G}}$ is symmetric, its degree is $\leqslant 2k! d^{(r'+1)(k+1)}$.

Similarly, for each tuple $(v_{(\bar{b},\iota)}, a) \in F^{\mathcal{G}}_i$ (with $i \in \{1,\ldots,k\}$) we know that a is the $\iota^{-1}(i)$-th component of $\bar{b}$ and each component of $\bar{b}$ belongs to the $\hat{r}$-neighborhood of a in $\mathcal{A}$, for $\hat{r} = k(2r{+}1)$. Thus, for each fixed $a \in A$, the number of elements $v_{(\bar{b},\iota)} \in V$ such that $(v_{(\bar{b},\iota)}, a) \in F^{\mathcal{G}}_i$ is at most $k! \cdot \sum_{s=1}^{k} |\mathcal{N}^{\mathcal{A}}_{\hat{r}}(a)|^s \leqslant k! \cdot d^{(\hat{r}+1)(k+1)}$. In summary, we thus obtain that $\mathcal{G}$ is of degree at most $d^{h(|\varphi|)}$ 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

$$f(\bar{a}) \;:=\; \big(v_{(\bar{a}_{P_1},\iota_{P_1})}, \ldots, v_{(\bar{a}_{P_\ell},\iota_{P_\ell})}, v_\perp, \ldots, v_\perp\big),$$

for the unique partition $P = (P_1, \ldots, P_\ell) \in \mathcal{P}$ such that $\mathcal{A} \models \varrho_P(\bar{a})$. 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 is where we use ε.

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 $\mathit{dist}^{\mathcal{A}}(a_i, a_j) \leqslant 2r{+}1$. This can be done as follows. Let R be the binary relation over $\mathit{dom}(\mathcal{A})$ containing all pairs (a, b) such that $\mathit{dist}^{\mathcal{A}}(a_i, a_j) \leqslant 2r{+}1$. As $\mathcal{A}$ as degree at most d, the size of R is 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 the Storing Theorem (Theorem 2.1), 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 ε whether a given pair is in R or not.

Once H is computed, we can compute its connected components in time depending only on k. Let ℓ 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 $\iota_{P_1}, \ldots, \iota_{P_\ell}$, and the according tuple $f(\bar{a})$. Let ζ_P be the function associating to each pair $(\bar{a}_{P_j}, \iota_{P_j})$ the element $v_{(\bar{a}_{P_j}, \iota_{P_j})}$. The domain of ζ_P is at most $n{\cdot}d^{k(2r+1)+1}$ and ζ_P can be computed in $O(n \cdot d^{k(2r+1)+1})$ by a brute-force algorithm. Hence, using Theorem 2.1 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 ζ_P in time depending only on ε.

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 $\bar{v} \in \mathit{dom}(\mathcal{G})$ in time $O(k^2)$.

This concludes the proof of proof of Proposition 3.4. □

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 (this is already the case for Theorem 2.4 and we build upon this result, this is also a consequence of Gaifman Normal Form which derives a new formula of non-elementary size [5]).

In the bounded degree case the constants are triply exponential in the query size [15]. In the (unranked) tree case the constants are provably non-elementary [10] (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 $O(n^\varepsilon)$ upon insertion or deletion of a tuple [22].

It would also be interesting to know whether we can enumerate the answers to a query using the *lexicographical* order (as it is the case over structures of bounded expansion [17]).

References

[1] Guillaume Bagan. MSO queries on tree decomposable structures are computable with linear delay. In *Proc. of Computer Science Logic (CSL'06)*, pages 167–181, 2006.

[2] Guillaume Bagan, Arnaud Durand, and Etienne Grandjean. On acyclic conjunctive queries and constant delay enumeration. In *Proc. of Computer Science Logic (CSL'07)*, pages 208–222, 2007.

[3] Johann Brault-Baron. A negative conjunctive query is easy if and only if it is beta-acyclic. In *Proc. of Computer Science Logic (CSL'12)*, 2012.

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

[5] Anuj Dawar, Martin Grohe, Stephan Kreutzer, and Nicole Schweikardt. Model theory makes formulas large. In *Automata, Languages and Programming, 34th International Colloquium, ICALP 2007, Wroclaw, Poland, July 9-13, 2007, Proceedings*, volume 4596 of *Lecture Notes in Computer Science*, pages 913–924. Springer, 2007.

[6] Arnaud Durand and Etienne Grandjean. First-order queries on structures of bounded degree are computable with constant delay. *ACM Transactions on Computational Logic*, 8(4), 2007.

[7] Arnaud Durand, Nicole Schweikardt, and Luc Segoufin. Enumerating answers to first-order queries over databases of low degree. In *Proc. Symp. on Principles of Database Systems (PODS'14)*, 2014.

[8] Zdenek Dvorák, Daniel Král, and Robin Thomas. Testing first-order properties for subclasses of sparse graphs. *J. ACM*, 60(5):36:1–36:24, 2013.

[9] Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Springer-Verlag, 2006.

[10] Markus Frick and Martin Grohe. The complexity of first-order and monadic second-order logic revisited. *Annals of Pure and Applied Logic*, 130(1-3):3–31, 2004.

[11] Haim Gaifman. On local and nonlocal properties. In J. Stern, editor, *Logic Colloquium'81*, pages 105–135. North-Holland, 1982.

[12] Etienne Grandjean and Frédéric Olive. Graph properties checkable in linear time in the number of vertices. *Journal of Computer and System Sciences*, 68(3):546–597, 2004.

[13] Martin Grohe. Generalized model-checking problems for first-order logic. In *Proc. Symp. on Theoretical Aspects of Computer Science (STACS'01)*, 2001.

[14] Martin Grohe, Stephan Kreutzer, and Sebastian Siebertz. Deciding first-order properties of nowhere dense graphs. *J. ACM*, 64(3):17:1–17:32, 2017.

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

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

[17] Wojciech Kazana and Luc Segoufin. Enumeration of first-order queries on classes of structures with bounded expansion. *Logical Methods in Computer Science*, 16(1), 2020.

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

[19] Johann A. Makowsky. Algorithmic uses of the Feferman-Vaught Theorem. *Annals of Pure and Applied Logic*, 126(1-3):159–213, 2004.

[20] Nicole Schweikardt, Luc Segoufin, and Alexandre Vigny. Enumeration for FO queries over nowhere dense graphs. In *Proc. of Symp. on Principles of Database Systems (PODS)*, 2018.

[21] Nicole Schweikardt, Luc Segoufin, and Alexandre Vigny. Enumeration for FO queries over nowhere dense graphs. *Journal version of [20], submitted*, 2020.

[22] Alexandre Vigny. Dynamic query evaluation over structures with low degree. *CoRR*, abs/2010.02982, 2020.

[23] Mihalis Yannakakis. Algorithms for acyclic database schemes. In *Proc. Intl. Conf. on Very Large Data Bases (VLDB'81)*, 1981.