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

Arnaud Durand
CNRS and ENS Cachan
www.logique.jussieu.fr/∼durand/

Nicole Schweikardt
Goethe-Universität Frankfurt
www.tks.cs.uni-frankfurt.de/schweika

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

ABSTRACT

A class of relational databases has low degree if for all δ, 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. in time bounded by $n^{1+\varepsilon}$, for all ε. We generalise 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 enumerating the answers to a query can be done with constant delay after a pseudo-linear time preprocessing.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*Query processing*; H.2.3 [**Database Management**]: Languages—*Query languages*

General Terms

Theory; Algorithms

Keywords

query evaluation; enumeration; low degree; algorithm

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 [5, 13] or, more generally, over the class of structures of "bounded expansion" [14]. Similar results have been shown for monadic second-order logic over structures of bounded tree-width [4, 1, 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 [19]).

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 databases of bounded expansion mentioned above.

It has been proved in [11] that over a class of databases of low degree, first-order boolean queries can be checked in pseudo-linear time, i.e., in time bounded by $O(n^{1+\varepsilon})$, for all $\varepsilon > 0$. 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 coloured graphs and quantifier-free queries. With such a tool, we can then focus within each specific task on very simple instances.

To appear in Proc. 33rd ACM SIGACT-SIGMOD-SIGART Sympmosium on Principles of Database Systems (PODS'14)

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 $\mathcal{A}$ amounts to 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 the class of databases of bounded degree, or bounded treewidth, FPT algorithms were derived for databases having bounded expansion [6] (see also [14]). Actually, very recently an FPT algorithm has been obtained for a class of databases known as "nowhere dense", generalising all the previously known results [12]. 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 [16], 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 [11]. They generalise the known enumeration algorithms for databases of bounded degree [5, 13]. However, they differ significantly from those and actually require an extra assumption on our computational model (see Section 2.2).

Organisation. 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 coloured graph. The result is stated in Section 3.2, 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 $R \in \sigma$. 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 $\mathcal{A}$. 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 $\mathcal{A} \models \varphi(\bar{a})$. 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 $\mathcal{A} \models \varphi$ or not.

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 [7, 10].

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.

Our algorithms over RAMs will take as input a query φ of size k and a structure $\mathcal{A}$ of size n. We then say that an algorithm runs in *linear time* (respectively, *constant time*) if it outputs the solution within $f(k)\cdot n$ steps (respectively, $f(k)$ steps), for some function f. We also say than an algorithm runs in *pseudo-linear time* if, for all $\varepsilon \in \mathbb{Q}_{>0}$ it outputs the solution within $f(k,\varepsilon)\cdot n^{1+\varepsilon}$ steps, for some function f.

We make the following important hypothesis on our RAM model. This hypothesis was not necessary in [14, 13, 5] for enumerating queries over classes of structures of bounded degree or bounded expansion, but we need it here for the case of structures of low degree (our proofs make crucial use of it; we don't know, though, whether this hypothesis is actually unavoidable).

If n is the size of the input structure, we assume available a total amount of memory of size $O(n^3)$.[1] Because our algorithms will be linear or pseudo-linear time, they will access only a small fraction of this memory, but it is important that this total memory is available. It turns out that we can assume without loss of generality that this memory is initialised to 0. If this were not the case, it could be achieved by using the so called *lazy array initialisation technique* (cf., e.g., the textbook [18]): During the run of the algorithm, a time-stamp is associated to each memory cell indicating the time of its first initialisation. At the same time we maintain an inverted list indicating for each time-stamp which memory cell was initialised. Then, a memory cell is initialised iff the entry for its associated time-stamp in the inverted list is the memory cell itself.

An important consequence of this assumption is that, modulo linear time preprocessing, we can assume available the adjacency

[1] Actually we need a memory of $O(n^{2+\varepsilon})$ for all $\varepsilon \in \mathbb{Q}_{>0}$

matrix of a graph given by the list of its edges. We do this by scanning all its edges and setting to 1 the corresponding entry in the matrix. As all other entries can be assumed to be 0, we can then test in constant time whether there is an edge between a given pair. We will implement this over graphs of size $O(n^{1+\varepsilon})$.

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 $q \in$ FO and $\mathcal{A} \in C$, can be decomposed into two phases:

- a preprocessing phase that is performed in pseudo-linear time, and
- an enumeration phase that outputs $q(\mathcal{A})$ with no repetition and a delay depending only on q between any two consecutive outputs. The enumeration phase has full access to the output of the preprocessing phase and can use extra memory whose size depends only on q.

Notice that if we can enumerate q with constant delay after a pseudo-linear preprocessing, then all answers can be output in time $f(|q|,\varepsilon) \cdot (\|\mathcal{A}\|^{1+\varepsilon} + |q(\mathcal{A})|)$, for some function f, 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 1. *To illustrate these notions, consider the binary query $q(x,y)$ over coloured 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 and through all red nodes with a constant delay between any two of them. Our extra assumption allows us to 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 colour 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

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 $\mathcal{A}$). 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.

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 [11]). 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*$(\mathcal{A}) \leqslant |\mathcal{A}|^\delta$.

For example, for every fixed number $c > 0$, the class of all structures of degree at most $(\log n)^c$ is of low degree. 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 class with known low evaluation complexity such as the "nowhere dense" case mentioned in the introduction.

Notice that *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. Therefore all our bounds concerning databases in a class of low degree could be expressed using $|\mathcal{A}|$ instead of $\|\mathcal{A}\|$ 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:

THEOREM 2 (GROHE [11]). *Let C be a class of structures of low degree. There is an algorithm which, on input of a structure $\mathcal{A} \in C$ and a sentence $q \in$ FO, tests in pseudo-linear time whether $\mathcal{A} \models q$.*

REMARK 3. Actually [11] proved a slightly stronger result. Let $k = |q|$. For each $\varepsilon > 0$, the algorithm of [11] runs in time $f(k){\cdot}n^{1+\varepsilon}$ if n is bigger than n_δ, where δ can be computed from k and ε, n_δ is the number given by the fact that C has low degree, and f is a *computable* function that does not depend on ε. If n is smaller than n_δ then the algorithm works in time bounded by $f(k){\cdot}n_\delta^3$. Altogether the algorithm then works in time $g(k,\varepsilon){\cdot}n^{1+\varepsilon}$ for some function g that is computable if the function associating n_δ from δ is computable. In any case it is pseudo-linear according to our definition. For readability we decided to state all our results using the pseudo-linear shorter variant but we actually prove the stronger versions as explained in this remark.

2.4 Our results

We are now ready to state our main results, which essentially lift Theorem 2 to non-boolean queries and to counting, testing, and enumerating their answers.

We start with counting the number of answers to a query.

THEOREM 4. *Let C be a class of structures of low degree. There is an algorithm that, given $\mathcal{A} \in C$ and $\varphi \in$ FO, computes $|\varphi(\mathcal{A})|$ in pseudo-linear time.*

We move to testing whether a given tuple is part of the answers.

THEOREM 5. *Let C be a class of structures of low degree. There is an algorithm that, given $\mathcal{A} \in C$ and $\varphi \in$ FO, computes in pseudo-linear time a data structure such that, on input of any $\bar{a}$, one can then test in constant time whether $\bar{a} \in \varphi(\mathcal{A})$.*

Finally, we consider enumerating the answers to a query.

THEOREM 6. *Let C be a class of structures of low degree. The enumeration problem of* FO *over C can be solved with constant delay after a pseudo-linear preprocessing.*

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 *dist*$^{\mathcal{A}}(a,b)$ for the distance between two nodes a and b of the Gaifman graph of $\mathcal{A}$. For an element $a \in$ *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 $b \in$ *dom*$(\mathcal{A})$ with *dist*$^{\mathcal{A}}(a,b) \leqslant r$. The *$r$-neighbourhood* 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 3.3, 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 γ which is a conjunction of relational atoms and potentially negated *unary* atoms. Note that the query of Example 1 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 8, and in the proofs for our counting and enumeration results in Sections 3.3 and 3.5.

LEMMA 7. *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 $r = k + \ell$.

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-neighbourhood $\mathcal{N}_r^{\mathcal{A}}(a)$ 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) Initialise $S_a := \emptyset$.

(2) Compute $\mathcal{N}_r^{\mathcal{A}}(a)$.

Since $\mathcal{A}$ has degree $\leqslant d$, this neighbourhood's domain contains at most d^{r+1} elements of $dom(\mathcal{A})$. Thus, by using breadth-first search, $\mathcal{N}_r^{\mathcal{A}}(a)$ 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 $\mathcal{N}_r^{\mathcal{A}}(a)$ 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 obtain the following:

PROPOSITION 8. *Let C be a class of structures of low degree. Given a structure $\mathcal{A} \in C$ and a connected conjunctive query q, one can compute $q(\mathcal{A})$ in pseudo-linear time.*

PROOF. We use the algorithm provided in Lemma 7. 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 7 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. □

The method of the proof of Proposition 8 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 *coloured graph* is a finite relational structure over a binary signature.

PROPOSITION 9. *There is an algorithm which, at input of a structure $\mathcal{A}$ and a first-order query $\varphi(\bar{x})$, produces a binary signature τ (containing, among other symbols, a binary relation symbol E), a coloured graph $\mathcal{G}$ of signature τ, an $\mathrm{FO}(\tau)$-formula $\psi(\bar{x})$, and a mapping f 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 $\psi(\mathcal{G})$.*

   Furthermore, on input of any tuple $\bar{v} \in \psi(\mathcal{G})$, the tuple $f^{-1}(\bar{v})$ can be computed in time and space $O(k^2)$.

   Using time $O(n\cdot d^{h(|\varphi|)})$ and space $O(n^2)$, we can furthermore construct a data structure such that, on input of any $\bar{a} \in dom(\mathcal{A})^k$, $f(\bar{a})$ can be computed in time $O(k^2)$.

The proof of Proposition 9 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 *generalised 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 1 is a generalised conjunction).

EXAMPLE 10. *Before moving to the formal proof of Theorem 4, consider again the query q from Example 1. 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 8 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 4 goes as follows. Using Proposition 9 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 10. The resulting formula is then a connected conjunction that is treated in pseudo-linear time using Proposition 8.

LEMMA 11. *There is an algorithm which, at input of a coloured graph $\mathcal{G}$ and a generalised 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 7 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 7. 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

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

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 9, we can lift this to arbitrary structures and first-order queries:

PROPOSITION 12. *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 9 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 9 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 11 to compute the number $s_i = |\gamma_i(\mathcal{G})|$ and output the value $s = \sum_{i \in I} s_i$.

By Lemma 11 we know that for each $i \in I$ the computation of 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, $\tilde{n} = |dom(\mathcal{G})|$, $\tilde{d} = degree(\mathcal{G})$, and h_0 is some computable function.

By Proposition 9 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 computing $|\varphi(\mathcal{A})| = \sum_{i \in I} s_i$ 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 4 is an immediate consequence of Proposition 12 (following the arguments of the proof of Proposition 8).

3.4 Testing

Here we consider the problem of testing whether a given tuple is a solution to a query. By Proposition 9 it is enough to consider quantifier-free formulas. Those are treated using the lazy array initialisation technique mentioned in Section 2.

PROPOSITION 13. *There is an algorithm which at input of a structure $\mathcal{A}$ and a first-order query $\varphi(\bar{x})$, has a preprocessing phase of time $h(|\varphi|) \cdot n \cdot d^{h(|\varphi|)}$ in which it computes a data structure such that, on input of any $\bar{a} \in dom(\mathcal{A})^k$ for $k = |\bar{x}|$, it can be tested in time $h(|\varphi|)$ whether $\bar{a} \in \varphi(\mathcal{A})$, where h is a computable function, $n = |dom(\mathcal{A})|$, and $d = degree(\mathcal{A})$.*

PROOF. We first use the algorithm of Proposition 9 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, this takes time and space $h(|\varphi|)\cdot n\cdot d^{h(|\varphi|)}$, 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 9 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 $\mathcal{G}$.

To enable checking whether $E(u,u')$ holds in $\mathcal{G}$, we construct the following data structure. W.l.o.g. we assume that $\mathit{dom}(\mathcal{G}) = \{1,\dots,\tilde{n}\}$ for

$$\tilde{n} := |\mathit{dom}(\mathcal{G})| \leqslant \|\mathcal{G}\| \leqslant h(|\varphi|)\cdot n\cdot d^{h(|\varphi|)}.$$

We use an $(\tilde{n}\times\tilde{n})$-array A_E that is initialised to 0. By looping through all edges $E(u,u')$ of $\mathcal{G}$, we then update the array entry $A_E[u,u']$ to 1. This way, using time $O(\|\mathcal{G}\|)$, we ensure that for all nodes u, u' of $\mathcal{G}$ we have $A_E[u,u'] = 1$ if $E(u,u')$ holds in $\mathcal{G}$, and $A_E[u,u'] = 0$ otherwise.

In a similar way, within time $O(\|\mathcal{G}\|)$ we can build, for each unary relation symbol C, a 1-dimensional array A_C of length $\tilde{n}$ such that for all nodes u of $\mathcal{G}$ we have $A_C[u] = 1$ if $C(u)$ holds in $\mathcal{G}$, and $A_C[u] = 0$ otherwise.

All these arrays are constructed in time $O(\|\mathcal{G}\|)$, which is $O(h(|\varphi|)\cdot n\cdot d^{h(|\varphi|)})$. This completes the preprocessing phase.

Note that using the arrays A_E and A_C, testing whether a given tuple $\bar{v} \in \mathit{dom}(\mathcal{G})^k$ belongs to $\psi(\mathcal{G})$ can be done in time $O(|\psi|)$, since each atomic statement of ψ can be checked in constant time by a simple look-up of the according array entry.

Finally, the testing algorithm works as follows. Given a tuple $\bar{a} \in \mathit{dom}(\mathcal{A})^k$, 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 9), and checking whether $\bar{v} \in \psi(\mathcal{G})$ requires time $O(|\psi|)$, which is $O(h(|\varphi|))$. Hence, we meet the required bound for testing. □

Theorem 5 is an immediate consequence of Proposition 13 (following the arguments of the proof of Proposition 8).

3.5 Enumeration

Here we consider the problem of enumerating the solutions to a given query. We first illustrate the proof of Theorem 6 with our running example.

EXAMPLE 14. *Consider again the query q of Example 1. 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 8 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.*

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 6 can be sketched as follows. By Proposition 9 it is enough to consider quantifier-free formulas looking for tuples of nodes that are disconnected and have certain colours. Hence the query q described in Example 1 corresponds to the binary case. For queries of larger arities we proceed by induction on the arity. By induction we can enumerate the answers of the query projecting out the last variable from the initial query. For each tuple $\bar{u}$ obtained by induction, we iterate through all the red nodes that are a potential completion. We then proceed as in Example 14. If the current red node a is not connected to $\bar{u}$, then $\bar{u}a$ forms an answer and we proceed to the next red node. If a is connected to $\bar{u}$ 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 14. The design and computation of this function is the main technical originality of the proof.

We now turn to the technical details that are summarised in the next proposition.

PROPOSITION 15. *There is an algorithm which at input of a structure $\mathcal{A}$ and a first-order query $\varphi(\bar{x})$ enumerates $\varphi(\mathcal{A})$ with delay $h(\varphi)$ after a preprocessing of time $h(\varphi)\cdot n\cdot d^{h(\varphi)}$, where $n = |\mathit{dom}(\mathcal{A})|$, $d = \mathit{degree}(\mathcal{A})$, 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. In case that $k > 0$ we proceed as follows.

We first use the algorithm of Proposition 9 to compute the according coloured graph $\mathcal{G}$ and the quantifier-free formula $\psi(\bar{x})$. This takes time $g(|\varphi|)\cdot n\cdot d^{g(|\varphi|)}$ 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 $\mathit{dom}(\mathcal{G})$ has $\tilde{n}$ elements, where $\tilde{n} \leqslant g(|\varphi|)\cdot n\cdot d^{g(|\varphi|)}$.

From Item 1 of Proposition 9 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 let $\bar{x} = (x_1,\dots,x_k)$. In case that $k = 1$, $\psi(x_1) = \psi_2(x_1)$ is a positive boolean combination of unary atoms. We can use Lemma 7 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. This is time $h(\varphi)\cdot n\cdot d^{h(\varphi)}$, for a computable function h, and can thus be done during the preprocessing phase. In the enumeration phase, we then simply loop through the list of all elements in $\bar{v} \in \psi(\mathcal{G})$, compute $\bar{a} := f^{-1}(\bar{v})$, and output $\bar{a}$. Due to Item 4 of Proposition 9, the delay is $O(k^2)$, and we are done.

The case for $k \geqslant 2$ requires much more elaborate constructions. We let $\bar{x}_{k-1} := (x_1,\dots,x_{k-1})$. To enable enumeration of $\psi(\mathcal{G})$ (and hence also $\varphi(\mathcal{A})$, by translating each result $\bar{v} \in \psi(\mathcal{G})$ to $\bar{a} := f^{-1}(\bar{v})$), 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.

Clearly, we can enumerate $\psi(\mathcal{G})$ by enumerating $\theta_j(\mathcal{G})$ for each $j \in J$. In the following, we therefore restrict attention to the enumeration of $\theta_j(\mathcal{G})$ for a fixed $j \in J$. For 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 the induction hypothesis (since θ' only has $k-1$ free variables), we can enumerate $\theta'(\mathcal{G})$ with delay $h(\theta')$ after a preprocessing phase of time $h(\theta')\cdot\tilde{n}\cdot\tilde{d}^{h(\theta')}$.

Since $P(x_k)$ is a boolean combination of unary atoms on x_k, we can use Lemma 7 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 $\leqslant^P$ to denote the linear ordering of $P(\mathcal{G})$ induced by this list, and we write first^P for the first element in this list, and next^P for the successor function, such that for any node $v \in P(\mathcal{G})$, $\mathit{next}^P(v)$ 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 next^P (i.e., $\mathit{next}(v,v')$ is true in $\hat{\mathcal{G}}$ iff $v' = \mathit{next}^P(v)$). Note that $\hat{\mathcal{G}}$ has degree $\hat{d} = \tilde{d}+2$, which is $\leqslant d^{g(|\varphi|)}$ for a computable function g.

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 $E_1, \dots, E_k$ 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 of Claim 1.

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

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

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 enumeration algorithm we will have to test, given nodes $u, v \in \mathit{dom}(\mathcal{G})$, whether $(u,v) \in E_k(\hat{\mathcal{G}})$. Since E_k is a first-order formula, Theorem 5 implies that, after preprocessing time $g'(|E_k|)\cdot\tilde{n}\cdot\hat{d}^{g'(|E_k|)}$ (for some computable function g'), testing membership in $E_k(\hat{\mathcal{G}})$, for any given $(u,v) \in \mathit{dom}(\mathcal{G})^2$, is possible within time $g'(|E_k|)$.

By our knowledge on the formula E_k and the size of the parameters $\tilde{n}$ and $\hat{d}$ we know that the preprocessing time is bounded by $g''(|\varphi|)\cdot n\cdot d^{g''(|\varphi|)}$, for a suitable computable function g'', and that each membership test can be done in time $g''(|\varphi|)$.

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 $\leqslant^P$ of $P(\mathcal{G})$) element $z \geqslant^P y$ 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

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

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 $z \geqslant^P y$ in the list $P(\mathcal{G})$ and test whether $\big(E(z,v) \vee E(v,z)\big)$ holds for some $v \in V$. To perform the latter test in constant time, we precompute an $(\tilde{n}\times\tilde{n})$-array A_E such that $A_E[z,v] = 1$ if $(z,v) \in E^{\mathcal{G}}$, and $A_E[z,v] = 0$ otherwise, in the same way as in the proof of Theorem 5.

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 an array, in time $O(\tilde{n}\cdot\hat{d}^{(3k^2)}\cdot g''(|\varphi|)\cdot k^2\cdot\tilde{d})$, which is time $g(|\varphi|)\cdot n\cdot d^{g(|\varphi|)}$ for a suitable computable function g. During the enumeration phase we will use this array such that for given y and V, 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 $g(|\varphi|)\cdot n\cdot d^{g(|\varphi|)}$, 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 enumerating $\theta'(\mathcal{G})$, which is $h(\theta')\cdot\tilde{n}\cdot\tilde{d}^{h(\theta')}$, for the computable function h in the Proposition's statement
   (b) the time for computing $P(\mathcal{G})$, which is $g(|\varphi|)\cdot n\cdot d^{g(|\varphi|)}$, for a computable function g
   (c) the preprocessing time for testing membership in $E_k(\hat{\mathcal{G}})$ and for producing the array A_E, which can be done in time $g(|\varphi|)\cdot n\cdot d^{g(|\varphi|)}$, for a computable function g
   (d) and the time for computing the *skip*-function, which is $g(|\varphi|)\cdot n\cdot d^{g(|\varphi|)}$, 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 $h(\varphi)\cdot n\cdot d^{h(\varphi)}$.

We now turn to the enumeration phase. We will describe how to enumerate $\theta(\mathcal{G})$ with delay $h(\varphi)$. Note that this will immediately lead to the desired enumeration algorithm for $\varphi(\mathcal{G})$ by doing the following: Loop through all $j \in J$ to enumerate $\theta_j(\mathcal{G})$; however, instead of outputting a tuple $\bar{v} \in \theta_j(\mathcal{G})$, compute the tuple $\bar{a} := f^{-1}(\bar{v})$ and output $\bar{a}$.

In the rest of this proof, we will therefore restrict attention to enumerating $\theta(\mathcal{G})$. We first describe the enumeration algorithm, then analyse its running time, and finally prove that it outputs, without repetition, all the tuples in $\theta(\mathcal{G})$.

Algorithm for enumerating $\theta(\mathcal{G})$:

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

Note 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 $h(\varphi)$.

By the induction hypothesis, the execution of line 1 and and each execution of line 7 takes time at most $h(\theta')$. Furthermore, each execution of line 3 takes time $(k{-}1){\cdot}g'(|E_k|)$. 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 $z \neq \textit{void}$, 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 $z \neq \textit{void}$, hence an output tuple is generated).

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

Concerning the correctness of the output, let us first show that every tuple $(\bar{u}, z)$ that is produced as an output, does belong to $\theta(\mathcal{G})$:

Recall that $\theta(\bar{x}) = \phi(\bar{x}_{k-1}) \wedge P(x_k) \wedge \gamma(\bar{x})$. We know that $\bar{u} \in \theta'(\mathcal{G})$, and thus, in particular, $\phi(\bar{u})$ is satisfied.

Furthermore, if the tuple $(\bar{u}, z)$ is output in line 5 (this is the only OUTPUT instruction present in the algorithm), we know that $V = \{v \in \{u_1, \ldots, u_{k-1}\} \,:\, (v, y) \in E_k(\hat{\mathcal{G}})\}$ and $z = \textit{skip}(y, V)$. Hence, z belongs to $P(\mathcal{G})$, and we know that z is not connected by an E-edge to any node in V. By the next claim (Claim 1) we obtain that z is not connected by an E-edge to any node in $\{u_1, \ldots, u_{k-1}\}$, and hence $\gamma(\bar{u}, z)$ is satisfied and the tuple $(\bar{u}, z)$ belongs to $\theta(\mathcal{G})$. This is the key of our enumeration algorithm.

CLAIM 1. *Let U be a set of at most $k{-}1$ nodes of $\mathcal{G}$. Let $y \in P(\mathcal{G})$, let $V := \{v \in U \,:\, (v, y) \in E_k(\hat{\mathcal{G}})\}$, and let $z := \textit{skip}(y, V) \neq \textit{void}$. Then,*

$$z = \min\{w \in P(\mathcal{G}) \,:\, y \leqslant^P w \text{ and } \forall u \in U :\, (u, w) \notin E(\mathcal{G}) \text{ and } (w, u) \notin E(\mathcal{G})\}.$$

PROOF OF CLAIM 1. By definition of $\textit{skip}(y, V)$ we have

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

Since $V \subseteq U$, we thus know that

$$z \leqslant^P \min\{w \in P(\mathcal{G}) \,:\, y \leqslant^P w \text{ and } \forall u \in U :\, (u, w) \notin E(\mathcal{G}) \text{ and } (w, u) \notin E(\mathcal{G})\}.$$

All that remains to be done is to show that for all $u \in U \setminus V$ we have $(u, z) \notin E(\mathcal{G})$ and $(z, u) \notin E(\mathcal{G})$.

In case that $z = y$, this is true because $U \setminus V$ only contains vertices that are *not* connected to y by an E_k-edge, and hence also not connected to y by an E-edge.

In case that $z \neq y$, we know that $y <^P z$, and thus $y \leqslant^P z' <^P z$ for the immediate predecessor z' of z, i.e., the node $z' \in P(\mathcal{G})$ with $\textit{next}^P(z') = z$.

For contradiction, assume that for some $u \in U \setminus V$ we have $(u, z) \in E(\mathcal{G})$ or $(z, u) \in E(\mathcal{G})$. Thus, $E'(z, u)$ is satisfied in $\hat{\mathcal{G}}$. Also, $\textit{next}(z', z)$ is satisfied in $\hat{\mathcal{G}}$. Since $y \leqslant^P z' <^P z$ (i.e., z' is skipped by $\textit{skip}(y, V)$), we furthermore know that z' is connected by an E-edge to some node $v \in V$, i.e., $E'(v, z')$ is true in $\hat{\mathcal{G}}$.

Assume now that also $E_{k-1}(v, y)$ is true in $\hat{\mathcal{G}}$. Recalling the definition of the formula E_k, note that we thus have found witnesses showing that $E_k(u, y)$ holds in $\hat{\mathcal{G}}$, i.e., $(u, y) \in E_k(\hat{\mathcal{G}})$. This, however, implies that $u \in V$, contradicting the assumption that $u \in U \setminus V$.

To conclude the proof of Claim 1 it remains to show that $E_{k-1}(v, y)$ is indeed true in $\hat{\mathcal{G}}$, i.e., $(v, y) \in E_{k-1}(\hat{\mathcal{G}})$. To this end, for all $j \leqslant k$, let $V_j := \{v' \in V \,:\, (v', y) \in E_j(\hat{\mathcal{G}})\}$. Clearly, $V_k = V$. By the choice of the formulas E_j we know that $\emptyset \subseteq V_1 \subseteq \cdots \subseteq V_k$. Moreover, it is straightforward to see that the definition of the formulas $E_1, \ldots, E_k$ ensures that the following is true: if $V_j = V_{j+1}$, for some $j < k$, then $V_j = \cdots = V_k$. Thus, there is a $j \leqslant k$ such that

$$(*): \qquad \emptyset \subseteq V_1 \subsetneq \cdots \subsetneq V_j = \cdots = V_k = V.$$

Since $V \subseteq U$, $|U| \leqslant k{-}1$, and $u \in U \setminus V$, we know that $|V| \leqslant k{-}2$. Hence, $(*)$ implies that $j \leqslant k{-}1$. Thus, $V_{k-1} = V$. Since $v \in V$, we therefore obtain that v belongs to V_{k-1}, i.e., $E_{k-1}(v, y)$ is true in $\hat{\mathcal{G}}$. This completes the proof of Claim 1. □

To finish the proof of Proposition 15, we need to verify that every tuple in $\theta(\mathcal{G})$ is eventually output by the algorithm. Let $(u_1, \ldots, u_k)$ be an arbitrary tuple in $\theta(\mathcal{G})$. Then, in particular, for $\bar{u} := (u_1, \ldots, u_{k-1})$, we have that $\bar{u} \in \theta'(\mathcal{G})$, Thus, by the induction hypothesis, the enumeration algorithm for θ' will eventually output the tuple $\bar{u}$. Let $z_1 <^P \cdots <^P z_m$ be an ordered list of all elements such that the enumeration algorithm for $\theta(\mathcal{G})$ outputs the tuple $(\bar{u}, z_i)$ for $i \in \{1, \ldots, m\}$. Clearly, it suffices to show that u_k is one of the z_i's.

Let $U := \{u_1, \ldots, u_{k-1}\}$. Since $(\bar{u}, u_k) \in \theta(\mathcal{G})$ we, in particular, have that $u_k \in P(\mathcal{G})$ and u_k is not connected to any $u \in U$ by an E-edge.

In case that $u_k \leqslant^P z_1$, by construction of the algorithm we know that $z_1 = \textit{skip}(y, V)$ for $y := \textit{first}^P$. By Claim 1, we furthermore know that z_1 is the *minimum* element $w \in P(\mathcal{G})$ with $y \leqslant^P w$, which is not connected to any $u \in U$ by an E-edge. Thus, $z_1 \leqslant^P u_k$, and hence $u_k = z_1$.

In case that $z_{i-1} <^P u_k \leqslant^P z_i$, we know for $y := \textit{next}^P(z_{i-1})$ that $z_i = \textit{skip}(y, V)$. Since $y \leqslant^P u_k$, we obtain from Claim 1 that $z_i \leqslant^P u_k$, and hence $u_k = z_i$.

The case that $z_m <^P u_k$ cannot occur since, by construction of the algorithm, the following is true: Either z_m is the largest element w.r.t. $\leqslant^P$ or for the element $y := \mathit{next}^P(z_m)$, we have $\mathit{skip}(y, V) = \mathit{void}$, whereas according to the definition of the *skip*-function, $\mathit{skip}(y, V)$ would have to be an element $\leqslant^P u_k$. This concludes the proof of Proposition 15

Theorem 6 follows immediately from Proposition 15 (following again the arguments of the proof of Proposition 8).

4. PROOF OF QUANTIFIER ELIMINATION AND NORMAL FORM

This section is devoted to the proof of Proposition 9. 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-neighbourhood 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 [9].

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 2, so it remains to treat formulas that are local around their free variables. By the Feferman-Vaught Theorem (cf., e.g. [17]), 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 9.

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 $\chi' := \mathit{true}$ if $\mathcal{A} \models \chi$ and $\chi' := \mathit{false}$ 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, all this can be done in time and space $O(h(|\varphi|) \cdot n \cdot d^{h(|\varphi|)})$, 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 can easily be computed given $\varphi^G(\bar{x})$.

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-neighbourhood around $\bar{x}$ in $\mathcal{A}$ is the disjoint union of the r-neighbourhoods 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-neighbourhood 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. [17]), we can, for each $P = (P_1, \ldots, P_\ell) \in \mathcal{P}$, compute a decomposition of $\varphi'(\bar{x})$ into r-local formulas $\vartheta_{P,j,t}(\bar{x}_{P_j})$, for $j \in \{1, \ldots, \ell\}$ 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 $\mathcal{A} \models \gamma_{P_j}(\bar{b})$ for $P_j := \{1, \ldots, |b|\}$ and
- for each injective mapping $\iota : \{1, \ldots, |\bar{b}|\} \to \{1, \ldots, k\}$.

Note that the first item ensures that the r-neighbourhood 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 $i \in \{1, \ldots, |\bar{b}|\}$).

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 $\mathcal{A} \models \varrho_P(\bar{a})$. For each $j \leqslant \{1, \ldots, \ell\}$, 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 ι_{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,

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

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 $\bar{y} = (y_1, \dots, y_k)$ 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 $P = (P_1, \dots, P_\ell)$ 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 first statement of item 4 of Proposition 9 are true.

To achieve the second statement of item 4, we use additional binary relation symbols $F_1, \dots, F_k$ which are interpreted in $\mathcal{G}$ as follows: Start by initialising all of them to the empty set. Then, for each $v = v_{(\bar{b},\iota)} \in V$ and each $j \in \{1, \dots, |\bar{b}|\}$, 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 $\tau := \tau_1 \cup \tau_2 \cup \{E, F_1, \dots, F_k\}$.

Using the relations $F_1, \dots, F_k$ of $\mathcal{G}$, in time and space $O(k)$ 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 $i \in \{1, \dots, k\}$ 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 and space $O(k^2)$, thus proving the second statement of item 4.

Step 4: proving item 3.

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

Thus, the set V, along with the relations $C_\perp$, C_ι and $F_1, \dots, F_k$ of $\mathcal{G}$, can be computed as follows: Start by letting $V := \{v_\perp\}$ and initialising all relations to the empty set. Let $C^{\mathcal{G}}_\perp := \{v_\perp\}$. Then, for each $a \in A$, compute the $\hat{r}$-neighbourhood $\mathcal{N}^{\mathcal{A}}_{\hat{r}}(a)$ of a in $\mathcal{A}$, and compute (by a brute-force algorithm), for each $s \in \{1, \dots, k\}$, the set of all s-tuples $\bar{b}$ of elements from this neighbourhood, 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 $P_j = \{1, \dots, s\}$. For each such tuple $\bar{b}$ do the following: For each injective mapping $\iota : \{1, \dots, s\} \to \{1, \dots, k\}$ add to V a new element $v_{(\bar{b},\iota)}$, add this element to the relation $C^{\mathcal{G}}_\iota$, and for each $j \in \{1, \dots, s\}$, 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 $F_1, \dots, F_k$ of $\mathcal{G}$, can be computed in time and space $O(h(|\varphi|){\cdot}n{\cdot}d^{h(|\varphi|)})$, for a computable function h.

For computing the unary relations $C_{P,j,t}$ of $\mathcal{G}$, start by initialising all of them to the empty set. For each $v_{(\bar{b},\iota)} \in V$ do the following: Compute (by using the relations $F_1, \dots, F_k$) the tuple $\bar{b}$ and the mapping ι. Let a be the first component of $\bar{b}$. Compute the $\tilde{r}$-neighbourhood $\mathcal{N}^{\mathcal{A}}_{\tilde{r}}(a)$ of a in $\mathcal{A}$, for $\tilde{r} := \hat{r}{+}r$. For each $P = (P_1, \dots, P_\ell) \in \mathcal{P}$, each $j \in \{1, \dots, \ell\}$ 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 neighbourhood is large enough.)

This way, $\mathcal{G}$'s relations $C_{P,j,t}$ can be computed in time and space $O(h(|\varphi|){\cdot}n{\cdot}d^{h(|\varphi|)})$, 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 initialising this relation to the empty set. For each $v_{(\bar{b},\iota)} \in V$ do the following: Compute (by using the relations $F_1, \dots, F_k$) the tuple $\bar{b}$. Let a be the first component of $\bar{b}$. Compute the r'-neighbourhood $\mathcal{N}^{\mathcal{A}}_{r'}(a)$ 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 $P_j = \{1, \dots, |\bar{c}|\}$. 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 $\hat{\iota} : \{1, \dots, |\bar{c}|\} \to \{1, \dots, k\}$.

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

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

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

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 $i \in \{1, \dots, k\}$) we know that a is the $\iota^{-1}(i)$-th component of $\bar{b}$ and each com-

ponent of $\bar{b}$ belongs to the $\hat{r}$-neighbourhood 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 $(v_{(\bar{b},\iota)}, a) \in F_i^{\mathcal{G}}$ is at most $k! \cdot \sum_{s=1}^{k} |\mathcal{N}_{\hat{r}}^{\mathcal{A}}(a)|^s \leqslant k! \cdot d^{(\hat{r}+1)(k+1)}$. In summary, we thus obtain that $\mathcal{G}$ is of degree at most $d^{h(|\varphi|)}$ for a computable function h.

Step 5: proving the third statement of item 4.

For $\bar{a} \in A^k$ we know that

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

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 $(k-\ell)$. 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 $\mathit{dist}^{\mathcal{A}}(a_i, a_j) \leqslant 2r+1$. Compute the connected components of H. 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 $j \in \{1, \ldots, \ell-1\}$.

Once the edge set of H is constructed, all further steps of this algorithm can be performed in time $O(k^2)$.

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 $f(\bar{a})$. Hence, we can compute $f(\bar{a})$ in time $O(k^2)$, provided that the edge set of H can be computed in time $O(k^2)$.

To enable the fast computation of the edge set of H, we precompute an $(n \times n)$-array D such that for any two elements $a, b \in A = \{1, \ldots, n\}$ we have $D[a,b] = 1$ if $a \neq b$ and $\mathit{dist}^{\mathcal{A}}(a,b) \leqslant 2r+1$, and $D[a,b] = 0$ otherwise. Once D is available, we can compute the edge set of H in time $O(k^2)$ by checking, for all $i, j \in \{1, \ldots, k\}$ whether $D[a_i, a_j] = 1$. To precompute the array D, we can proceed as follows: In the standard RAM-model, when starting the RAM D is initialised to 0 for free (in a model where this initialisation is not for free, we can avoid initialisation by using the *lazy array technique* described in Section 2). For each $a \in A$, we use time $d^{O(2r+1)}$ to compute the $(2r+1)$-neighbourhood of a in $\mathcal{A}$, and for each element $b \neq a$ in this neighbourhood we let $D[a,b] := 1$. In summary, the computation of the array D thus is done in time $O(n \cdot d^{h(|\varphi|)})$ for a computable function h. This completes the proof of Proposition 9. $\square$

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 [13]. In the (unranked) tree case the constants are provably non-elementary [8] (modulo some complexity assumption). We do not know what is the situation for classes of low degree. 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 [14]). Finally it would be interesting to know whether the memory assumption that we use for our RAMs can be avoided.

6. REFERENCES

[1] G. Bagan. MSO queries on tree decomposable structures are computable with linear delay. In *Proc. 20th International Workshop/15th Annual Conference of the EACSL (CSL'06)*, pages 167–181, 2006.

[2] G. Bagan, A. Durand, and E. Grandjean. On acyclic conjunctive queries and constant delay enumeration. In *Proc. 21st International Workshop/16th Annual Conference of the EACSL (CSL'07)*, pages 208–222, 2007.

[3] J. Brault-Baron. A negative conjunctive query is easy if and only if it is beta-acyclic. In *Proc. 26th International Workshop/21st Annual Conference of the EACSL (CSL'12)*, pages 137–151, 2012.

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

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

[6] Z. Dvořák, D. Král, and R. Thomas. Deciding First-Order Properties for Sparse Graphs. In *Proc. 51st Annual IEEE Symposium on Foundations of Computer Science (FOCS'10)*, pages 133–142, 2010.

[7] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer-Verlag, 2006.

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

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

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

[11] M. Grohe. Generalized model-checking problems for first-order logic. In *Proc. 18th Annual Symposium on Theoretical Aspects of Computer Science (STACS'01)*, pages 12–26, 2001.

[12] M. Grohe, S. Kreutzer, and S. Siebertz. Deciding first-order properties of nowhere dense graphs. In *Proc. 46th Symposium on Theory of Computing (STOC'14)*, 2014. Preliminary version: CoRR abs/1311.3899 (2013).

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

[14] W. Kazana and L. Segoufin. Enumeration of first-order queries on classes of structures with bounded expansion. In *Proc. 32nd ACM Symposium on Principles of Database Systems (PODS'13)*, pages 297–308, 2013.

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

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

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

[18] B. M. E. Moret and H. D. Shapiro. *Algorithms from P to NP. Volume 1: Design & Efficiency*. Benjamin/Cummings, 1990.

[19] M. Yannakakis. Algorithms for acyclic database schemes. In *Proc. 7th International Conference on Very Large Data Bases (VLDB'81)*, pages 82–94, 1981.