First-order queries on classes of structures with bounded expansion

Wojciech Kazana, Luc Segoufin

First-order queries on classes of structures with bounded expansion

Wojciech Kazana
INRIA and ENS Cachan
Luc Segoufin
INRIA and ENS Cachan

January 31, 2018

Abstract

We consider the evaluation of first-order queries over classes of databases with *bounded expansion*. The notion of bounded expansion is fairly broad and generalizes bounded degree, bounded treewidth and exclusion of at least one minor. It was known that over a class of databases with bounded expansion, first-order sentences could be evaluated in time linear in the size of the database. We give a different proof of this result. Moreover, we show that answers to first-order queries can be enumerated with constant delay after a linear time preprocessing. We also show that counting the number of answers to a query can be done in time linear in the size of the database.

Query evaluation is certainly the most important problem in databases. Given a query q and a database $\mathbf{D}$ it computes the set $q(\mathbf{D})$ of all tuples in the output of q on $\mathbf{D}$. However, the set $q(\mathbf{D})$ may be larger than the database itself as it can have a size of the form n^l where n is the size of the database and l the arity of the query. Therefore, computing entirely $q(\mathbf{D})$ may require too many of the available resources.

There are many solutions to overcome this problem. For instance one could imagine that a small subset of $q(\mathbf{D})$ can be quickly computed and that this subset will be enough for the user needs. Typically one could imagine computing the top-ℓ most relevant answers relative to some ranking function or to provide a sampling of $q(\mathbf{D})$ relative to some distribution. One could also imagine computing only the number of solutions $|q(\mathbf{D})|$ or providing an efficient test for whether a given tuple belongs to $q(\mathbf{D})$ or not.

In this paper we consider a scenario consisting in enumerating $q(\mathbf{D})$ with constant delay. Intuitively, this means that there is a two-phases algorithm working as follows: a preprocessing phase that works in time linear in the size of the database, followed by an enumeration phase outputting one by one all the elements of $q(\mathbf{D})$ with a constant delay between any two consecutive outputs. In particular, the first answer is output after a time linear in the size of the database and once the enumeration starts a new answer is being output regularly at a speed independent from the size of the database. Altogether, the set $q(\mathbf{D})$ is entirely computed in time $f(q)(n + |q(\mathbf{D})|)$ for some function f depending only on q and not on $\mathbf{D}$.

One could also view a constant delay enumeration algorithm as follows. The preprocessing phase computes in linear time an index structure representing the set $q(\mathbf{D})$ in a compact way (of size linear in n). The enumeration algorithm is then a streaming decompression algorithm.

One could also require that the enumeration phase outputs the answers in some given order. Here we will consider the lexicographical order based on a linear order on the domain of the database.

There are many problems related to enumeration. The main one is the model checking problem. This is the case when the query is boolean, i.e. outputs only *true* or *false*. In this case a constant delay enumeration algorithm is a Fixed Parameter Linear (FPL) algorithm for the model checking problem of q, i.e. it works in time $f(q)n$. This is a rather strong constraint as even the model checking problem for conjunctive queries is not FPL (assuming some hypothesis in parametrized complexity) [20]. Hence, in order to obtain constant delay enumeration algorithms, we need to make restrictions on the queries and/or on the databases. Here we consider first-order (FO) queries over classes of structures having "bounded expansion".

The notion of class of graphs with bounded expansion was introduced by Nešetřil and Ossona de Mendez in [17]. Its precise definition can be found in Section 1.2. At this point it is only useful to know

that it contains the class of graphs of bounded degree, the class of graphs of bounded treewidth, the class of planar graphs, and any class of graphs excluding at least one minor. This notion is generalized to classes of structures via their Gaifman graphs or adjacency graphs.

For the class of structures with bounded degree and FO queries the model checking is in FPL [21] and there also are constant delay enumeration algorithms [8, 14]. In the case of structures of bounded treewidth and FO queries (actually even MSO queries with first-order free variables) the model checking is also in FPL [7] and there are constant delay enumeration algorithms [3, 15]. For classes of structures with bounded expansion the model checking problem for FO queries was recently shown to be in FPL [9, 11].

Our results can be summarized as follows. For FO queries and any class of structures with bounded expansion:

- we provide a new proof that the model checking problem can be solved in FPL,
- we show that the set of solutions to a query can be enumerated with constant delay,
- we show that computing the number of solutions can be done in FPL,
- we show that, after a preprocessing in time linear in the size of the database, one can test on input $\bar{a}$ whether $\bar{a} \in q(\mathbf{D})$ in constant time.

Concerning model checking, our method uses a different technique than the previous ones. There are several characterizations of classes having bounded expansion [17]. Among them we find the "low tree depth coloring" and the "transitive fraternal augmentations". The previous methods were based on the low tree depth coloring characterization while ours is based on transitive fraternal augmentations. We argue that the use of transitive fraternal augmentations gives a simpler proof. The reason is that it gives a useful normal form on quantifier-free formulas that will be the core of our algorithms for constant delay enumeration and for counting the number of solutions. As for the previous proofs, we exhibit a quantifier elimination method, also based on our normal form. Our quantifier elimination method results in a quantifier-free formula but over a recoloring of a functional representation of a "fraternal and transitive augmentation" of the initial structure.

Our other algorithms (constant delay enumeration, counting the number of solution or testing whether a tuple is a solution or not) start by eliminating the quantifiers as for the model checking algorithm. The quantifier-free case is already non trivial and require the design and the computation of new index structures. For instance consider the simple query $R(x, y)$. Given a pair (a, b) we would like to test whether (a, b) is a tuple of the database in constant time. In general, index structures can do this with $\log n$ time. We will see that we can do constant time, assuming bounded expansion.

In the presence of a linear order on the domain of the database, our constant delay algorithm can output the answers in the corresponding lexicographical order.

Related work We make use of a functional representation of the initial structures. Without this functional representations we would not be able to eliminate all quantifiers. Indeed, with this functional representation we can talk of a node at distance 2 from x using the quantifier-free term $f(f(x))$, avoiding the existential quantification of the middle point. This idea was already taken in [8] for eliminating first-order quantifiers over structures of bounded degree. Our approach differs from theirs in the fact that in the bounded degree case the functions can be assumed to be permutations (in particular they are invertible) while this is no longer true in our setting, complicating significantly the combinatorics.

Once we have a quantifier-free formula, constant delay enumeration could also be obtained using the characterization of bounded expansion based on low tree depth colorings. Indeed, using this characterization one can easily show that enumerating a quantifier-free formula over structures of bounded expansion amounts in enumerating an MSO query over structures of bounded tree-width and for those known algorithms exist [3, 15]. However, the known enumeration algorithms of MSO over structures of bounded treewidth are rather complicated while our direct approach is fairly simple. Actually, our proof shows that constant delay enumeration of FO queries over structures of bounded treewidth can be done using simpler algorithms than for MSO queries. Moreover, it gives a constant delay algorithm outputting the solutions in lexicographical order. No such algorithms were known for FO queries over structures of bounded treewidth. In the bounded degree case, both enumeration algorithms of [8, 14] output their solutions in lexicographical order.

Similarly, counting the number of solutions of a quantifier-free formula over structures of bounded expansion reduces to counting the number of solutions of a MSO formula over structures of bounded treewidth. This latter problem is known to be in FPL [2]. We give here a direct and simple proof of this fact for FO queries over structures of bounded expansion.

1 Preliminaries

In this paper a database is a finite relational structure. A *relational signature* is a tuple $\sigma = (R_1, \ldots, R_l)$, each R_i being a relation symbol of arity r_i. A *relational structure* over σ is a tuple $\mathbf{D} = \big(D, R_1^{\mathbf{D}}, \ldots, R_l^{\mathbf{D}}\big)$, where D is *the domain* of $\mathbf{D}$ and $R_i^{\mathbf{D}}$ is a subset of D^{r_i}. We will often write R_i instead of $R_i^{\mathbf{D}}$ when $\mathbf{D}$ is clear from the context.

We use a standard notion of size. The *size* of $R_i^{\mathbf{D}}$, denoted $\|R_i^{\mathbf{D}}\|$ is the number of tuples in $R_i^{\mathbf{D}}$ multiplied by the arity r_i. The *size* of the domain of $\mathbf{D}$, denoted $|\mathbf{D}|$, is the number of elements in D. Finally the *size* of $\mathbf{D}$, denoted by $\|\mathbf{D}\|$, is

$$\|\mathbf{D}\| = |\mathbf{D}| + \Sigma_{R_i \in \sigma} \|R_i^{\mathbf{D}}\|.$$

By *query* we mean a formula written in the first-order logic, FO, built from atomic formulas of the form $x = y$ or $R_i(x_1, \ldots, x_{r_i})$ for some relation R_i, and closed under the usual Boolean connectives $(\neg, \vee, \wedge)$ and existential and universal quantifications $(\exists, \forall)$. We write $\phi(\bar{x})$ 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. We use the usual semantics, denoted $\models$, for first-order. Given a structure $\mathbf{D}$ and a query q, an *answer* to q in $\mathbf{D}$ is a tuple $\bar{a}$ of elements of $\mathbf{D}$ such that $\mathbf{D} \models q(\bar{a})$. We write $q(\mathbf{D})$ for the set of answers to q in $\mathbf{D}$, i.e. $q(\mathbf{D}) = \{\bar{a} \mid \mathbf{D} \models q(\bar{a})\}$. As usual, $|q|$ denotes the size of q.

Let $\mathcal{C}$ be a class of structures. The model checking problem for FO over $\mathcal{C}$ is the computational problem of given first-order *sentence* q and a database $\mathbf{D} \in \mathcal{C}$ to test whether $\mathbf{D} \models q$ or not.

We now introduce our running examples.

Example A-1. *The first query has arity 2 and returns pairs of nodes at distance 2 in a graph. The query is of the form* $\exists z E(x, z) \wedge E(z, y)$.

Testing the existence of a solution to this query can be easily done in time linear in the size of the database. For instance one can go trough all nodes of the database and check whether it has non-nill in-degree and out-degree. The degree of each node can be computed in linear time by going through all edges of the database and incrementing the counters associated to its endpoints.

Example B-1. *The second query has arity 3 and returns triples* (x, y, z) *such that* y *is connected to* x *and* z *via an edge but* x *is not connected to* z*. The query is of the form* $E(x, y) \wedge E(y, z) \wedge \neg E(x, z)$.

It is not clear at all how to test the existence of a solution to this query in time linear in the size of the database. The problem is similar to the one of finding a triangle in a graph, for which the best know algorithm has complexity even slightly worse than matrix multiplication [1]. If the degree of the input structure is bounded by a constant d*, we can test the existence of a solution in linear time by the following algorithm. We first go through all edges* (x, y) *of the database and add* y *to a list associated to* x *and* x *to a list associated to* y*. It remains now to go through all nodes* y *of the database, consider all pairs* (x, z) *of nodes in the associated list (the number of such pairs is bounded by* d^2*) and then test whether there is an edge between* x *and* z *(by testing whether* x *is in the list associated to* z*).*

We aim at generalizing this kind of reasoning to structures with bounded expansion.

Given a query q, we care about "enumerating" $q(\mathbf{D})$ efficiently. Let $\mathcal{C}$ be a class of structures. For a query $q(\bar{x})$, *the enumeration problem of* q *over* $\mathcal{C}$ is, given a database $\mathbf{D} \in \mathcal{C}$, to output the elements of $q(\mathbf{D})$ one by one with no repetition. The maximal time between any two consecutive outputs of elements of $q(\mathbf{D})$ is called *the delay*. The definition below requires a constant time delay. We formalize these notions in the forthcoming section.

1.1 Model of computation and enumeration

We use Random Access Machines (RAM) with addition and uniform cost measure as a model of computation. For further details on this model and its use in logic see [8]. 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 database as a word. Whenever we iterate through all nodes of the domain, the iteration is with respect to the initial linear order.

We say that the enumeration problem of q over a class $\mathcal{C}$ of structures is in the class $\text{CD} \circ \text{LIN}$, or equivalently that we can enumerate q over $\mathcal{C}$ with constant delay, if it can be solved by a RAM algorithm which, on input $\mathbf{D} \in \mathcal{C}$, can be decomposed into two phases:

- a precomputation phase that is performed in time $O(\|\mathbf{D}\|)$,
- an enumeration phase that outputs $q(\mathbf{D})$ with no repetition and a constant delay between two consecutive outputs. The enumeration phase has full access to the output of the precomputation phase but can use only a constant total amount of extra memory.

Notice that if we can enumerate q with constant delay, then all answers can be output in time $O(\|\mathbf{D}\| + |q(\mathbf{D})|)$ and the first output is computed in time linear in $\|\mathbf{D}\|$. In the particular case of boolean queries, the associated model checking problem must be solvable in time linear in $\|\mathbf{D}\|$. Notice also that the total amount of memory used after computing all answers is linear in $\|\mathbf{D}\|$, while a less restrictive definition requiring only a constant time delay between any two outputs may yield in a total amount of memory linear in $\|\mathbf{D}\| + \|q(\mathbf{D})\|$.

Note that we measure the running time complexity as a function of $\|\mathbf{D}\|$. The multiplicative factor will depend on the class $\mathcal{C}$ of database under consideration and, more importantly, on the query q. In our case we will see that the multiplicative factor is non elementary in $|q|$ and that cannot be avoided, see the discussion in the conclusion section.

We may in addition require that the enumeration phase outputs the answers to q using the lexicographical order. We then say that we can enumerate q over $\mathcal{C}$ with constant delay in lexicographical order.

Example A-2. *Over the class of all graphs, we cannot enumerate pairs of nodes at distance 2 with constant delay unless the Boolean Matrix Multiplication problem can be solved in quadratic time [5]. However, over the class of graphs of degree d, there is a simple constant delay enumeration algorithm. During the preprocessing phase, we associate to each node the list of all its neighbors at distance 2. This can be done in time linear in the size of the database as in Example B-1. We then color in blue all nodes having a non empty list and make sure each blue node points to the next blue node (according to the linear order on the domain). This also can be done in time linear in the size of the database and concludes the preprocessing phase. The enumeration phase now goes through all blue nodes x using the pointer structure and, for each of them, outputs all pairs (x, y) where y is in the list associated to x.*

Example B-2. *Over the class of all graphs, the query of this example cannot be enumerated with constant delay because, as mentioned in Example B-1, testing whether there is one solution is already non linear. Over the class of graphs of bounded degree, there is a simple constant delay enumeration algorithm, similar to the one from Example A-2.*

Note that in general constant delay enumeration algorithms are not closed under any boolean operations. For instance it is not because we can enumerate q and q' with constant delay, that we can enumerate $q \vee q'$ with constant delay as enumerating one query after the other would break the "no repetition" requirement. However, if we can enumerate with constant delay in the lexicographical order, then a simple argument that resembles the problem of merging two sorted lists shows closure under union:

Lemma 1. *If both queries $q(\bar{x})$ and $q'(\bar{x})$ can be enumerated in lexicographical order with constant delay then the same is true for $q(\bar{x}) \vee q'(\bar{x})$.*

Proof. The preprocessing phase consists in the preprocessing phases of the enumeration algorithms for q and q'.

The enumeration phase keeps two values, the smallest element from $q(\mathbf{D})$ that was not yet output and similarly the smallest element from $q'(\mathbf{D})$ that was not yet output. It then outputs the smaller of the two values and replaces it in constant time with the next element from the appropriate set using the associated enumeration procedure. In case the elements are equal, the value is output once and both stored values are replaced with their appropriate successors. □

It will follow from our results that the enumeration problem of FO over the class of structures with "bounded expansion" is in CD$\circ$LIN. The notion of bounded expansion was defined in [17] for graphs and then it was generalized to structures via their Gaifman or Adjacency graphs. We start with defining it for graphs.

1.2 Graphs with bounded expansion and augmentation

By default a graph has no orientation on its edges and has colors on its vertices. In an *oriented* graph every edge is an arrow going from the source vertex to its target. We can view a (oriented or not) graph as a relational structure $\mathbf{G} = (V^{\mathbf{G}}, E^{\mathbf{G}}, P_1^{\mathbf{G}}, \ldots, P_l^{\mathbf{G}})$, where $V^{\mathbf{G}}$ is the set of nodes, $E^{\mathbf{G}} \subseteq V^2$ is the set of edges and, for each $1 \leq i \leq l$, $P_i^{\mathbf{G}}$ is a predicate of arity 1. We omit the subscripts when $\mathbf{G}$ is clear from the context. In the nonoriented case, E is symmetric and we denote by $\{u, v\}$ the edge between u and v. In the oriented case we denote by (u, v) the edge from u to v. We will use the notation $\vec{\mathbf{G}}$ when the graph is oriented and $\mathbf{G}$ in the nonoriented case. An *orientation* of a graph $\mathbf{G}$ is any graph $\vec{\mathbf{H}}$ such that $\{u, v\} \in E^{\mathbf{G}}$ implies $(u, v) \in E^{\vec{\mathbf{H}}}$ or $(v, u) \in E^{\vec{\mathbf{H}}}$. The *in-degree* of a node v of $\vec{\mathbf{G}}$ is the number of nodes u such that $(u, v) \in E$. We denote by $\Delta^-(\vec{\mathbf{G}})$ the maximal in-degree of a node of $\vec{\mathbf{G}}$. Among all orientations of a graph $\mathbf{G}$, we choose the following one, which is computable in time linear in $\|\mathbf{G}\|$. We find the first node of minimal degree, orient its edges towards it and repeat this inductively in the induced subgraph obtained by removing this node. The resulting graph, denoted $\vec{\mathbf{G}}_0$, has maximal in-degree which is at most twice the optimal value and that is enouh for our needs.

In [17] several equivalent definitions of bounded expansion were shown. We present here only the one we will use, exploiting the notion of "augmentations".

Let $\vec{\mathbf{G}}$ be an oriented graph. A 1-*transitive fraternal augmentation of* $\vec{\mathbf{G}}$ is any graph $\vec{\mathbf{H}}$ with the same vertex set as $\vec{\mathbf{G}}$ and the same colors of vertices, including all edges of $\vec{\mathbf{G}}$ (with their orientation) and such that for any three vertices x, y, z of $\vec{\mathbf{G}}$ we have the following:

(transitivity) if (x, y) and (y, z) are edges in $\vec{\mathbf{G}}$, then (x, z) is an edge in $\vec{\mathbf{H}}$,

(fraternity) if (x, z) and (y, z) are edges in $\vec{\mathbf{G}}$, then at least one of the edges: (x, y), (y, x) is in $\vec{\mathbf{H}}$,

(strictness) moreover, if $\vec{\mathbf{H}}$ contains an edge that was not present in $\vec{\mathbf{G}}$, then it must have been added by one of the previous two rules.

Note that the notion of 1-transitive fraternal augmentation is not a deterministic operation. Although transitivity induces precise edges, fraternity implies nondeterminism and thus there can possibly be many different 1-transitive fraternal augmentations. We care here about choosing the orientations of the edges resulting from the fraternity rule in order to minimize the maximal in-degree.

Following [18] we fix a deterministic algorithm computing a "good" choice of orientations of the edges induced by the fraternity property. The precise definition of the algorithm is not important for us, it only matters here that the algorithm runs in time linear in the size of the input graph (see Lemma 2 below). With this algorithm fixed, we can now speak of **the** 1-transitive fraternal augmentation of $\vec{\mathbf{G}}$.

Let $\vec{\mathbf{G}}_0$ be an oriented graph. The *transitive fraternal augmentation* of $\vec{\mathbf{G}}_0$ is the sequence $\vec{\mathbf{G}}_0 \subseteq \vec{\mathbf{G}}_1 \subseteq \vec{\mathbf{G}}_2 \subseteq \ldots$ such that for each $i \geq 1$ the graph $\vec{\mathbf{G}}_{i+1}$ is the 1-transitive fraternal augmentation of $\vec{\mathbf{G}}_i$. We will say that $\vec{\mathbf{G}}_i$ is the i-th augmentation of $\vec{\mathbf{G}}_0$. Similarly we denote the *transitive fraternal augmentation* of a nonoriented graph $\mathbf{G}$ by considering its minimal orientation $\vec{\mathbf{G}}_0$.

Definition 1. *[17] Let $\mathcal{C}$ be a class of graphs. $\mathcal{C}$ has bounded expansion if there exists a function $\Gamma_{\mathcal{C}} : \mathbb{N} \to \mathbb{R}$ such that for each graph $\mathbf{G} \in \mathcal{C}$ its transitive fraternal augmentation $\vec{\mathbf{G}}_0 \subseteq \vec{\mathbf{G}}_1 \subseteq \vec{\mathbf{G}}_2 \subseteq \ldots$ of $\mathbf{G}$ is such that for each $i \geq 0$ we have $\Delta^-(\vec{\mathbf{G}}_i) \leq \Gamma_{\mathcal{C}}(i)$.*

Consider for instance a graph of degree d. Notice that the 1-transitive fraternal augmentation introduces an edge between nodes that were at distance at most 2 in the initial graph. Hence, when starting with a graph of degree d, we end up with a graph of degree at most d^2. This observation shows that the class of graphs of degree d has bounded expansion as witnessed by the function $\Gamma(i) = d^{2^i}$. Exhibiting the function Γ for the other examples of classes with bounded expansion mentioned in the introduction: bounded treewidth, planar graphs, graphs excluding at least one minor, requires more work [17].

The following lemma shows that within a class $\mathcal{C}$ of bounded expansion the i-th augmentation of $\mathbf{G} \in \mathcal{C}$ can be computed in linear time, the linear factor depending on i and on $\mathcal{C}$.

Lemma 2. *[18] Let $\mathcal{C}$ be a class of bounded expansion. For each $\boldsymbol{G} \in \mathcal{C}$ and each $i \geq 0$, $\vec{\boldsymbol{G}}_i$ is computable from $\boldsymbol{G}$ in time $O(\|\boldsymbol{G}\|)$.*

A transitive fraternal augmentation introduces new edges in the graphs in a controlled way. We will see that we can use these extra edges in order to eliminate quantifiers in a first-order formula. Lemma 2 shows that this quantifier elimination is harmless for enumeration as it can be done in time linear in the size of the database and can therefore be done during the preprocessing phase.

1.3 Graphs of bounded in-degree as functional structures

Given the definition of bounded expansion it is convenient to work with oriented graphs. These graphs will always be such that the maximal in-degree is bounded by some constant depending on the class of graphs under investigation. It is therefore convenient for us to represent our graphs as functional structures where the functions links the current node with its predecessors. This functional representation turns out to be also useful for eliminating some quantifiers.

A *functional signature* is a tuple $\sigma = (f_1, \ldots, f_l, P_1, \ldots, P_m)$, each f_i being a functional symbol of arity 1 and each P_i being an unary predicate. A *functional structure* over σ is then defined as for relational structures. FO is defined as usual over the functional signature. In particular, it can use atoms of the form $f(f(f(x)))$, which is crucial for the quantifier elimination step of Section 3 as the relational representation would require existential quantification for denoting the same element. A graph $\vec{\mathbf{G}}$ of in-degree l and colored with m colors can be represented as a functional structure $f\vec{\mathbf{G}}$, where the unary predicates encode the various colors and $v = f_i(u)$ if v is the i^{th} element (according to some arbitrary order that will not be relevant in the sequel) such that (v, u) is an edge of $\vec{\mathbf{G}}$. We call such node v the *i^{th} predecessor* of u (where "i^{th} predecessor" should really be viewed as an abbreviation for "the node v such that $f_i(u) = v$" and not as a reference to the chosen order). If we do not care about the i and we only want to say that v is the image of u under some function, we call it a *predecessor* of u. Given a nonoriented graph $\mathbf{G}$ we define $f\vec{\mathbf{G}}$ to be the functional representation of $\vec{\mathbf{G}}_0$ as described above. Note that $f\vec{\mathbf{G}}$ is computable in time linear in $\|\mathbf{G}\|$ and that for each first order query $\phi(\bar{x})$, over the relational signature of graphs, one can easily compute a first order query $\psi(\bar{x})$, over the functional signature, such that $\phi(\mathbf{G}) = \psi(f\vec{\mathbf{G}})$.

Example A-3. *Consider again the query computing nodes at distance 2 in a nonoriented graph. There are four possible ways to orient a path of length 2. With the functional point of view we further need to consider all possible predecessors. Altogether the distance 2 query is now equivalent to:*

$$\bigvee_{f,g} f(g(x)) = y \ \vee \ g(f(y)) = x \ \vee \ f(x) = g(y) \ \vee \exists z \ \ f(z) = x \wedge g(z) = y$$

where the small disjuncts correspond to the four possible orientations and the big one to all possible predecessors, each of them corresponding to a function name, whose number depends on the function $\Gamma_{\mathcal{C}}$.

Example B-3. *Similarly, the reader can verify that the query of Example B-1 is equivalent to:*

$$\bigvee_{f,g} \bigwedge_{h} (h(x) \neq z \wedge h(z) \neq x) \wedge \ \big[(f(x) = y \wedge g(y) = z) \vee (x = f(y) \wedge g(y) = z) \vee (f(x) = y \wedge y = g(z)) \vee (x = f(y) \wedge y = g(z))\big].$$

Augmentation for graphs as functional structures. The notion of 1-transitive fraternal augmentation can be adapted directly to the functional setting. However it will be useful for us to enhance it with extra information. In particular it will be useful to remember at which stage the extra edges are inserted. We do this as follows.

Given a graph $f\vec{\mathbf{G}}$, its 1-transitive fraternal augmentation $f\vec{\mathbf{G}}'$ is constructed as follows. The signature of $f\vec{\mathbf{G}}'$ expands the signature of $f\vec{\mathbf{G}}$ with new function symbols for taking care of the new edges created during the expansion and $f\vec{\mathbf{G}}'$ is then an expansion of $f\vec{\mathbf{G}}$ over this new signature.

For any pair of functions f and g in the signature of $f\vec{\mathbf{G}}$ there is a new function h in the signature of $f\vec{\mathbf{G}}'$ representing the transitive part of the augmentation. It is defined as the composition of f and g, i.e. $h^{f\vec{\mathbf{G}}'} = f^{f\vec{\mathbf{G}}} \circ g^{f\vec{\mathbf{G}}}$

Similarly, for any pair of functions f and g in the signature of $f\vec{\mathbf{G}}$, and any node x in the domain of both $f^{f\vec{\mathbf{G}}}$ and $g^{f\vec{\mathbf{G}}}$ there will be a function h in the new signature representing the fraternity part of the augmentation. I.e h is such that $h^{f\vec{\mathbf{G}}'}(f^{f\vec{\mathbf{G}}'}(x)) = g^{f\vec{\mathbf{G}}}(x)$ or $h^{f\vec{\mathbf{G}}'}(g^{f\vec{\mathbf{G}}'}(x)) = f^{f\vec{\mathbf{G}}'}(x)$.

Given a class $\mathcal{C}$ of bounded expansion, the guarantees that the number of new function symbols needed for the i-th augmentation is bounded by $\Gamma_{\mathcal{C}}(i)$ and does not depend on the graph. Hence a class $\mathcal{C}$ of bounded expansion generates finite functional signatures $\sigma_{\mathcal{C}}(0) \subseteq \sigma_{\mathcal{C}}(1) \subseteq \sigma_{\mathcal{C}}(2) \subseteq \ldots$ such that for any graph $\mathbf{G} \in \mathcal{C}$ and for all i:

1. $f\vec{\mathbf{G}}_i$ is a functional structure over $\sigma_{\mathcal{C}}(i)$ computable in linear time from $\mathbf{G}$,
2. $f\vec{\mathbf{G}}_{i+1}$ is an expansion of $f\vec{\mathbf{G}}_{i+1}$,
3. for every FO query $\phi(\bar{x})$ over $\sigma_{\mathcal{C}}(i)$ and every $j \geq i$ we have that $\phi(f\vec{\mathbf{G}}_i) = \phi(f\vec{\mathbf{G}}_j)$.

We denote by $\alpha_{\mathcal{C}}(i)$ the number of function symbols of $\sigma_{\mathcal{C}}(i)$. Notice that we have $\alpha_{\mathcal{C}}(i) \leq \Sigma_{j \leq i}\Gamma_{\mathcal{C}}(j)$.

We say that a functional signature σ' is a *recoloring* of σ if it extends σ with some extra unary predicates, also denoted as *colors*, while the functional part remains intact. Similarly, a functional structure $f\vec{\mathbf{G}}'$ over σ' is a *recoloring* of $f\vec{\mathbf{G}}$ over σ if σ' is a recoloring of σ and $f\vec{\mathbf{G}}'$ is a σ'-expansion of $f\vec{\mathbf{G}}$. We write *ϕ is over a recoloring of σ* if ϕ is over σ' and σ' is a recoloring of σ. Notice that the definition of bounded expansion is not sensitive to the colors as it depend only on the binary predicates, hence adding arbitrarily many extra colors is harmless.

Given a class $\mathcal{C}$ of graphs, for each $p \geq 0$, we define $\mathcal{C}_p$ to be the class of all recolorings $f\vec{\mathbf{G}}_p'$ of $f\vec{\mathbf{G}}_p$ for some $\mathbf{G} \in \mathcal{C}$. In other words $\mathcal{C}_p$ is the class of functional representations of all recolorings of all p-th augmentations of graphs from $\mathcal{C}$. Note that all graphs from $\mathcal{C}_p$ are recolorings of a structure in $\sigma_{\mathcal{C}}(p)$, hence they use at most $\alpha_{\mathcal{C}}(p)$ function symbols.

From now on we assume that all graphs from $\mathcal{C}$ and all queries are in their functional representation. It follows from the discussion above that this is without loss of generality.

1.4 From structures to graphs

A class of structures is said to have bounded expansion if the set of adjacency graphs of the structures of the class has bounded expansion.

The *adjacency graph* of a relational structure $\mathbf{D}$, denoted by Adjacency($\mathbf{D}$), is a functional structure defined as follows. The set of vertices of Adjacency($\mathbf{D}$) is $D \cup T$ where D is the domain of $\mathbf{D}$ and T is the set of tuples occurring in some relation of $\mathbf{D}$. For each relation R_i in the schema of $\mathbf{D}$, there is a unary symbol P_{R_i} coloring the elements of T belonging to R_i. For each tuple $t = (a_1, \cdots, a_{r_i})$ such that $\mathbf{D} \models R_i(t)$ for some relation R_i of arity r_i, we have an edge $f_j(t) = a_j$ for all $j \leq r_i$.

Observation 1. *It is immediate to see that for every relational structure $\boldsymbol{D}$ we can compute Adjacency($\boldsymbol{D}$) in time $O(\|\boldsymbol{D}\|)$.*

Let $\mathcal{C}$ be a class of relational structures. We say that $\mathcal{C}$ has *bounded expansion* if the class $\mathcal{C}$' of adjacency graphs (seen as graphs) of structures from $\mathcal{C}$ has bounded expansion.

Remark 1. *In the literature, for instance [9, 11], a class $\mathcal{C}$ of relational structures is said to have bounded expansion if the class of their Gaifman graphs has bounded expansion. It is easy to show that if the class of* Gaifman graphs *of structures from $\mathcal{C}$ has bounded expansion then the class of* adjacency graphs *of structures from $\mathcal{C}$ also has bounded expansion. The converse is not true in general. However the converse holds if the schema is fixed, i.e. $\mathcal{C}$ is a class of structures all having the same schema. We refer to [13] for the simple proofs of these facts.*

Let $\Gamma_{\mathcal{C}'}$ be the function given by Definition 1 for $\mathcal{C}$'. The following lemma is immediate. For instance $R(\bar{x})$ is rewritten as $\exists y P_R(y) \wedge \bigwedge_{1 \leq i \leq r} f_i(y) = x_i$.

Lemma 3. *Let $\mathcal{C}$ be a class of relational structures with bounded expansion and let $\mathcal{C}$' be the underlying class of adjacency graphs. Let $\phi(\bar{x}) \in$ FO. In time linear in the size of ϕ we can find a query $\psi(\bar{x})$ over $\sigma_{\mathcal{C}'}(0)$ such that for all $\boldsymbol{D} \in \mathcal{C}$ we have $\phi(\boldsymbol{D}) = \psi(\textit{Adjacency}(\boldsymbol{D}))$.*

As a consequence of Lemma 3 it follows that model checking, enumeration and counting of first-order queries over relational structures reduce to the graph case. Therefore in the rest of the paper we will only concentrate on the graph case (viewed as a functional structure), but the reader should keep in mind that all the results stated over graphs extend to relational structures via this lemma.

2 Normal form for quantifier-free first-order queries

We prove in this section a normal form on quantifier-free first-order formulas. This normal form will be the ground for all our algorithms later on. It says that, modulo performing some extra augmentation steps, a quantifier-free formula has a very simple form.

Fix class $\mathcal{C}$ of graphs with bounded expansion. Recall that we are now implicitly assuming that graphs are represented as functional structures.

A formula is *simple* if it does not contain atoms of the form $f(g(x))$, i.e. it does not contain any compositions of functions. We first observe that, modulo augmentations, any formula can be transformed into a simple one.

Lemma 4. *Let $\psi(\bar{x})$ be a formula over a recoloring of $\sigma_{\mathcal{C}}(p)$. Then, for $q = p + |\psi|$, there is a simple formula $\psi'(\bar{x})$ over a recoloring of $\sigma_{\mathcal{C}}(q)$ such that:*

for all graph $f\vec{\boldsymbol{G}} \in \mathcal{C}_p$ there is a graph $f\vec{\boldsymbol{G}}' \in \mathcal{C}_q$ computable in time linear in $\|f\vec{\boldsymbol{G}}\|$ such that $\psi(f\vec{\boldsymbol{G}}) = \psi'(f\vec{\boldsymbol{G}}')$.

Proof. This is a simple consequence of transitivity. Any composition of two functions in $f\vec{\mathbf{G}}$ represents a transitive pair of edges and becomes an single edge in the 1-augmentation $f\vec{\mathbf{H}}$ of $f\vec{\mathbf{G}}$. Then $y = f(g(x))$ over $f\vec{\mathbf{G}}$ is equivalent to $\bigvee_h y = h(x) \wedge P_{f,g,h}(x)$ over $f\vec{\mathbf{H}}$, where h is one of the new function introduced by the augmentation and the newly introduced color $P_{f,g,h}$ holds for those nodes v, for which the $f(g(v)) = h(v)$. As the nesting of compositions of functions is at most $|\psi|$, the result follows. The linear time computability is immediate from Lemma 2. □

We make one more observation before proving the normal form:

Lemma 5. *Let $f\vec{\boldsymbol{G}} \in \mathcal{C}_p$. Let u be a node of $f\vec{\boldsymbol{G}}$. Let S be all the predecessors of u in $f\vec{\boldsymbol{G}}$ and set $q = p + \Gamma_{\mathcal{C}}(p)$. Let $f\vec{\boldsymbol{G}}' \in \mathcal{C}_q$ be the $(q-p)$-th augmentation of $f\vec{\boldsymbol{G}}$. There exists a linear order $<$ induced on S by $f\vec{\boldsymbol{G}}'$, such that for all $v, v' \in S$, $v < v'$ implies $v' = f(v)$ is an edge of $f\vec{\boldsymbol{G}}'$ for some function f from $\sigma_{\mathcal{C}}(q)$.*

Proof. This is because all nodes of S are fraternal and the size of S is at most $\Gamma_{\mathcal{C}}(p)$. Hence, after one step of augmentation, all nodes of S are pairwise connected and, after at most $\Gamma_{\mathcal{C}}(p) - 1$ further augmentation steps, if there is a directed path from one node u of S to another node v of S, then there is also a directed edge from u to v. By induction on $|S|$ we show that there exists a node $u \in S$ such that for all $v \in S$ there is an edge from v to u. If $|S| = 1$ there is nothing to prove. Otherwise fix $v \in S$ and let $S' = S \setminus \{v\}$. By induction we get a u in S' satisfying the properties. If there is an edge from v to u, u also works for S and we are done. Otherwise there must be an edge from u to v. But then there is a path of length 2 from any node of S' to v. By transitivity this means that there is an edge from any node of S' to v and v is the node we are looking for.

We then set u as the minimal element of our order on S and we repeat this argument with $S \setminus \{u\}$. □

Lemma 5 justifies the following definition. Let p be a number and let $q = p + \Gamma_{\mathcal{C}}(p)$. A p-type $\tau(x)$ is a quantifier-free formula over the signature $\sigma_{\mathcal{C}}(q)$ with one free variable x consisting in the conjunction

of a maximal consistent set of clauses of the form $f(g(x)) = h(x)$ or $\exists y\ y = f(x)$. Given a node u of some graph $f\vec{\mathbf{G}}$ of $\mathcal{C}_p$, its p-type is the set of clauses satisfied by u in the $(q-p)$-th augmentation $f\vec{\mathbf{G}}'$ of $f\vec{\mathbf{G}}$. From Lemma 5 it follows that the p-type of u induces a linear order on its predecessors in $f\vec{\mathbf{G}}$. Indeed the predecessors of u in $f\vec{\mathbf{G}}$ can be deduced from the p-type by looking at the clauses $\exists y\ y = f(x)$ where f is a function symbol from $\sigma_{\mathcal{C}}(p)$ and the linear order can be deduced from the clauses $h(f(x)) = g(x)$. Lemma 5 guarantees that these latter clauses induce a linear order. In the sequel we denote this property as "the p-type τ induces the order $f_1(x) < f_2(x) < \cdots$" and for $i < j$ we refer to the h linking $f_i(x)$ to $f_j(x)$ as $h_{i,j}$.

Note that for a given p there are only finitely many possible p-types and that each of them can be specified with a conjunctive formula over $\sigma_{\mathcal{C}}(q)$.

We now state the normal form result.

Proposition 1. *Let $\phi(\bar{x}y)$ be a simple quantifier-free query over a recoloring of $\sigma_{\mathcal{C}}(p)$. There exists q that depends only on p and ϕ and a quantifier-free query ψ over a recoloring of $\sigma_{\mathcal{C}}(q)$ that is a disjunction of formulas:*

$$\psi_1(\bar{x}) \wedge \tau(y) \wedge \Delta^{=}(\bar{x}y) \wedge \Delta^{\neq}(\bar{x}y), \tag{1}$$

where $\tau(y)$ contains a p-type of y; $\Delta^{=}(\bar{x}y)$ is either empty or contains one clause of the form $y = f(x_i)$ or one clause of the form $f(y) = g(x_i)$ for some suitable i, f and g; and $\Delta^{\neq}(\bar{x}y)$ contains arbitrarily many clauses of the form $y \neq f(x_i)$ or $f(y) \neq g(x_j)$. Moreover, ψ is such that:

for all $f\vec{\boldsymbol{G}} \in \mathcal{C}_p$ there is a $f\vec{\boldsymbol{G}}' \in \mathcal{C}_q$ computable in time linear in $\|f\vec{\boldsymbol{G}}\|$ with $\phi(f\vec{\boldsymbol{G}}) = \psi(f\vec{\boldsymbol{G}}')$.

Proof. Set q as given by Lemma 5. We first put ϕ into a disjunctive normal form (DNF) and in front of each such disjunct we add a big disjunction over all possible p-types of y (recall that a type can be specified as a conjunctive formula). We deal with each disjunct separately.

Note that each disjunct is a query over $\sigma_{\mathcal{C}}(q)$ of the form:

$$\psi_1(\bar{x}) \wedge \tau(y) \wedge \Delta^{=}(\bar{x}y) \wedge \Delta^{\neq}(\bar{x}y),$$

where all sub-formulas except for $\Delta^{=}$ are as desired. Moreover, $\psi_1(\bar{x})$, $\Delta^{=}(\bar{x}y)$ and $\Delta^{\neq}(\bar{x}y)$ are in fact queries over $\sigma_{\mathcal{C}}(p)$. At this point $\Delta^{=}$ contains arbitrarily many clauses of the form $y = f(x_i)$ or $f(y) = g(x_i)$. If it contains at least one clause of the form $y = f(x_i)$, we can replace each other occurrence of y by $f(x_i)$ and we are done.

Assume now that $\Delta^{=}$ contains several conjuncts of the form $f_i(y) = g(x_k)$. Assume wlog that τ is such that $f_1(y) < f_2(y) < \cdots$, where $f_1(y), f_2(y), \cdots$ are all the predecessors of y from $\sigma_{\mathcal{C}}(p)$. Let i_0 be the smallest index i such that a clause of the form $f_i(y) = g(x_k)$ belongs to $\Delta^{=}$. We have $f_{i_0}(y) = g(x_k)$ in $\Delta^{=}$ and recall that τ specifies for $i < j$ a function $h_{i,j}$ in $\sigma_{\mathcal{C}}(q)$ such that $h_{i,j}(f_i(y)) = f_j(y)$. Then, as y is of type τ, a clause of the form $f_j(y) = h(x_{k'})$ with $i_0 < j$ is equivalent to $h_{i_0,j}(g(x_k)) = h(x_{k'})$. □

Example A-4. *Let us see what Lemma 4 and the normalization algorithm do for $p = 0$ and some of the disjuncts of the query of Example A-3:*

In the case of $f(g(x)) = y$ note that by transitivity, in the augmented graph, this clause is equivalent to one of the form $y = h(x) \wedge P_{f,g,h}(x)$ (this case is handled by Lemma 4).

Consider now $\exists z\ \ f(z) = x \wedge g(z) = y$. It will be convenient to view this query when z plays the role of y in Proposition 1. Notice that in this case it is not in normal form as $\Delta^{=}$ contains two elements. However, the two edges $f(z) = x$ and $g(z) = y$ are fraternal. Hence, after one augmentation step, a new edge is added between x and y and we either have $y = h(x)$ or $x = h(y)$ for some h in the new signature.

Let $\tau_{h,f,g}(z)$ be 0-type stating that $h(f(z)) = g(z)$ and $\tau_{h,g,f}(z)$ be 0-type stating that $h(g(z)) = f(z)$. It is now easy to see that the query $\exists z\ \ f(z) = x \wedge g(z) = y$ is equivalent to

$$\exists z \bigvee_h y = h(x) \wedge \tau_{h,f,g}(z) \wedge f(z) = x \ \ \vee$$
$$x = h(y) \wedge \tau_{h,g,f}(z) \wedge g(z) = y$$

3 Model checking

In this section we show that the model checking problem of FO over a class of structures with bounded expansion can be done in time linear in the size of the structure. This gives a new proof of the result of [9]. Recall that by Lemma 3 it is enough to consider oriented graphs viewed as functional structures.

Theorem 1. *[9] Let $\mathcal{C}$ be a class of graphs with bounded expansion and let ψ be a sentence of* FO*. Then, for all $f\vec{\boldsymbol{G}} \in \mathcal{C}$, testing whether $f\vec{\boldsymbol{G}} \models \psi$ can be done in time $O(\|f\vec{\boldsymbol{G}}\|)$.*

The proof of Theorem 1 is done using a quantifier elimination procedure: given a query $\psi(\bar{x})$ with at least one free variable we can compute a quantifier-free query $\phi(\bar{x})$ that is "equivalent" to ψ. Again, the equivalence should be understood modulo some augmentation steps for a number of augmentation steps depending only on $\mathcal{C}$ and $|\psi|$. When starting with a sentence ψ we end-up with ϕ being a boolean combination of formulas with one variable. Those can be easily tested in linear time in the size of the augmented structure, which in turns can be computed in time linear from the initial structure by Lemma 2. The result follows. We now state precisely the quantifier elimination step:

Proposition 2. *Let $\mathcal{C}$ be a class of graphs with bounded expansion witnessed by the function $\Gamma_{\mathcal{C}}$. Let $\psi(\bar{x}y)$ be a quantifier-free formula over a recoloring of $\sigma_{\mathcal{C}}(p)$. Then one can compute a q and a quantifier-free formula $\phi(\bar{x})$ over a recoloring of $\sigma_{\mathcal{C}}(q)$ such that:*

for all $f\vec{\boldsymbol{G}} \in \mathcal{C}_p$ there is a $f\vec{\boldsymbol{G}}' \in \mathcal{C}_q$ such that:

$$\phi(f\vec{\boldsymbol{G}}') = (\exists y \psi)(f\vec{\boldsymbol{G}})$$

Moreover, $f\vec{\boldsymbol{G}}'$ is computable in time $O(\|f\vec{\boldsymbol{G}}\|)$.

Proof. In view of Lemma 4 we can assume that ψ is simple. We then apply Proposition 1 to ψ and p and obtain a q and an equivalent formula in DNF, where each disjunct has the special form given by (1). As disjunction and existential quantification commute, it is enough to treat each part of the disjunction separately.

We thus assume that $\psi(\bar{x}y)$ is a quantifier-free conjunctive formula over a recoloring of $\sigma_{\mathcal{C}}(q)$ of the form (1):

$$\psi_1(\bar{x}) \wedge \tau(y) \wedge \Delta^{=}(\bar{x}y) \wedge \Delta^{\neq}(\bar{x}y).$$

Let's assume that the p-type τ satisfied by y enforces $f_1(y) < f_2(y) < \cdots$, where $f_1(y), f_2(y), \cdots$ are all the images of y by a function from $\sigma_{\mathcal{C}}(p)$. Moreover, for each $i < j$, τ contains an atom of the form $h_{i,j}(f_i(y)) = f_j(y)$ for some function $h_{i,j} \in \sigma_{\mathcal{C}}(q)$.

We do a case analysis depending on the value of $\Delta^{=}$.

• If $\Delta^{=}$ is $y = g(x_k)$ for some function g and some k, then we replace y with $g(x_k)$ everywhere in $\psi(\bar{x}y)$ resulting in a formula $\phi(\bar{x})$ having obviously the desired properties.

• Assume now that $\Delta^{=}$ is of the form $f(y) = g(x_k)$. Without loss of generality we can assume that f is f_{i_0} and $k = 1$. In other words $\Delta^{=}$ is the constraint $f_{i_0}(y) = g(x_1)$.

The general idea is to limit the quantification on y to a finite set (whose size depends only on $\mathcal{C}$ and ψ), depending only on x_1. We then encode these set using suitable extra colors. To do this, for each node w we first compute a set $\textsc{Witness}(w)$ such that for each tuple $\bar{v}$ we have $f\vec{\mathbf{G}}_q \models \exists y \ \ \psi(\bar{v}y)$ iff $f\vec{\mathbf{G}}_q \models \exists y \in \textsc{Witness}(g(v_1)) \ \ \psi(\bar{v}y)$. Moreover, for all w, $|\textsc{Witness}(w)| \leq N$ where N is a number depending only on p. We then encode these witness sets using suitable extra colors.

The intuition behind the WITNESS set is as follows. Assume first that $\Delta^{\neq}$ is empty. Then we only need to test the existence of y such that $f_{i_0}(y) = g(x_1)$. To do so, we scan through all nodes u, test if $\tau(u)$ holds and if so we add u to $\textsc{Witness}(f_{i_0}(u))$ if this set is empty and do nothing otherwise. Clearly each witness set has size one and the desired properties are verified. It is then enough to color with a new color red all nodes having a non-empty witness set and $\exists y \ \tau(y) \wedge f_{i_0}(y) = g(x_1)$ is then equivalent to $\text{red}(g(x_1))$.

The situation is slightly more complicated if $\Delta^{\neq}$ is not empty. Assume for instance that $\Delta^{\neq}$ contains only constraints of the form $f(y) \neq h(x_k)$. Then the previous procedure does not work because because $\textsc{Witness}(g(x_1))$ may be such that it is equal to $h(x_k)$. However there are only c nodes that we need to avoid, where c depends only on the formula, hence if $\textsc{Witness}(g(x_1))$ contains at least $c + 1$ nodes we

are sure that at least one of them will satisfy all the inequality constraints. We implement this by scanning through all nodes u, test if $\tau(u)$ holds and if so we add u to $\text{WITNESS}(f_{i_0}(u))$ if this set has a size smaller or equal to c do nothing otherwise. The difficulty is to encode this set into the formula. If the witness set is of size $c+1$ one of its element must make all inequalities true hence a new color as before does the job. When the set has a smaller size we need to test each of its elements against the inequalities. For this we introduce a predicates Q_i, and add a node u to Q_i if u has been added as the i^{th} element in $\text{WITNESS}(f_{i_0}(u))$. Once this is done we can check that the i^{th} witness of $g(x_1)$ verify $y \neq h(x_k)$ by testing that $h(x_k)$ is not the $i^t h$ witness of $g(x_1)$, i.e. : $\neg(Q_i(h(x_k)) \wedge f_{i_0}(h(x_k)) = g(x_1))$.

The general case, when $\Delta^{\neq}$ contains also clauses of the form $h_1(y) \neq h_2(x_k)$ is more complex and require an even bigger witness set but this is essentially what we do.

Computation of the Witness function We start by initializing $\text{WITNESS}(v) = \emptyset$ for all v.

We then successively investigate all nodes u of $f\vec{\mathbf{G}}_q$ and do the following. If $f\vec{\mathbf{G}}_q \models \neg\tau(u)$ then we move on to the next u. If $f\vec{\mathbf{G}}_q \models \tau(u)$ then let $u_1, \cdots, u_l$ be the current value of $\text{WITNESS}(f_{i_0}(u))$ - if $\text{WITNESS}(f_{i_0}(u))$ is empty then we add u to this set and move on to the next node of $f\vec{\mathbf{G}}_q$.

Let β_p be $\alpha_{\mathcal{C}}(p)(\alpha_{\mathcal{C}}(p)+1)|\bar{x}|+1$.

Let i be minimal such that there exists j with $f_i(u_j) = f_i(u)$ (notice that $i \leq i_0$). Note that because $f_j(w) = h_{i,j}(f_i(w))$ for all w verifying τ and all $j > i$, this implies that u and u_j agree on each f_j with $j \geq i$ and disagree on each f_j with $j < i$.

Let $S_i = \{f_{i-1}(u_j) \mid f_i(u_j) = f_i(u)\}$, where $f_0(u_j)$ is u_j in the case where $i = 1$. If $|S_i| < \beta_p$ then we add u to $\text{WITNESS}(f_{i_0}(u))$.

Analysis of the Witness function Clearly the algorithm computing the witness function runs in linear time.

Moreover, for each node v, $\text{WITNESS}(v)$ can be represented as the leaves of a tree of depth at most $\alpha_{\mathcal{C}}(p)$ and of width β_p. To see this, notice that all nodes u of $\text{WITNESS}(v)$ are such that $f_{i_0}(u) = v$. Note also that if two nodes u and u' satisfying τ share a predecessor, $f_i(u) = f_i(u')$, then for all $j > i$, u and u' agree f_j as $f_j = h_{i,j} \circ f_i$ for all nodes satisfying τ. The depth of the least common ancestor of two nodes u and u' of $\text{WITNESS}(v)$ is defined as the least i such that u and u' agree on f_i. One can then verify that by construction of $\text{WITNESS}(v)$ the tree has the claimed sizes. Hence the size of $\text{WITNESS}(v)$ is bounded by $\beta_p^{\alpha_{\mathcal{C}}(p)+1}$.

We now show that for each tuple $\bar{v}$ and each node u such that $f\vec{\mathbf{G}}_q \models \psi(\bar{v}u)$ there is a node u' in $\text{WITNESS}(g(v_1))$ such that $f\vec{\mathbf{G}}_q \models \psi(\bar{v}u')$.

To see this assume $f\vec{\mathbf{G}}_q \models \psi(\bar{v}u)$. If $u \in \text{WITNESS}(g(v_1))$ we are done. Otherwise note that $f_{i_0}(u) = g(v_1)$ and that $f\vec{\mathbf{G}}_q \models \tau(u)$. Let i and S_i be as described in the algorithm when investigating u. As u was not added to $\text{WITNESS}(f_{i_0}(u))$, we must have $|S_i| > \beta_p$. Let $u_1, \cdots, u_{\beta_p}$ be the elements of $\text{WITNESS}(g(v_1))$ providing β_p pairwise different values for f_{i-1}. Among these, at most $\alpha_{\mathcal{C}}(p)|\bar{v}|$ of them may be of the form $f_j(v_l)$ for some j and l as each v_l has at most $\alpha_{\mathcal{C}}(p)$ predecessors. Notice that for all $j > i$ and all ℓ, u agree with u_ℓ on f_i and therefore they also agree on f_j for $j > i$ as $f_j = h_{i,j} \circ f_i$ for all nodes satisfying τ. When $j < i$ the values of $f_j(u_\ell)$ and $f_j(u_{\ell'})$ must be different if $\ell \neq \ell'$ as otherwise u_ℓ and $u_{\ell'}$ would also agree on f_{i-1} as $f_{i-1} = h_{j,i-1} \circ h_{i-1}$ for all nodes satisfying τ. Therefore, for each l and each $j < i$ there are at most $\alpha_{\mathcal{C}}(p)$ of the u_ℓ such that $f_j(u_\ell)$ is a predecessor of v_l.

Altogether most $\alpha_{\mathcal{C}}(p)^2|\bar{v}| + \alpha_{\mathcal{C}}(p)|\bar{v}|$ nodes u_ℓ may falsify an inequality constraint. As β_p is strictly bigger than that, one of the u_ℓ is the desired witness.

Recoloring of $f\vec{\mathbf{G}}_q$ Based on WITNESS we recolor $f\vec{\mathbf{G}}_q$ as follows. Let $\gamma_p = (\beta_p + 1)^{\alpha_{\mathcal{C}}(p)+1}$. For each $v \in f\vec{\mathbf{G}}_q$, the i^{th} witness of v is the i^{th} element inserted in $\text{WITNESS}(v)$ by the algorithm.

For each $i \leq \gamma_p$ we introduce a new unary predicate P_i and for each $u \in f\vec{\mathbf{G}}_q$ we set $P_i(u)$ if $\text{WITNESS}(u)$ contains at least i elements.

For each $i \leq \gamma_p$, we introduce a new unary predicate Q_i and for each $v \in f\vec{\mathbf{G}}_q$ we set $Q_i(v)$ if the i^{th} witness of $f_{i_0}(v)$ is v.

For each $i \leq \gamma_p$ and each $h, h' \in \alpha_{\mathcal{C}}(q)$ we introduce a new unary predicate $P_{i,h,h'}$ and for each $v \in f\vec{\mathbf{G}}_q$ we set $P_{i,h,h'}(v)$ if the i^{th} witness of $h(v)$ is an element u with $h'(u) = v$.

We denote by $f\vec{\mathbf{G}}'$ the resulting graph and notice that it can be computed in linear time from $f\vec{\mathbf{G}}$.

Computation of ϕ We now replace $\psi(\bar{x}, y)$ by the following formula:

$$\bigvee_{i \leq \gamma_p} \psi_1(\bar{x}) \wedge \psi^i(\bar{x})$$

where $\psi^i(\bar{x})$ checks that the i^{th} witness of $g(x_1)$ makes the initial formula true. This formula $\psi^i(\bar{x})$ is defined by

$$\begin{aligned} P_i(g(x_1)) \quad \wedge & \bigwedge_{\substack{f_j(y) \neq h(x_k) \in \Delta^{\neq} \\ j < i_0}} \neg\big(h_{j,i_0}(h(x_k)) = g(x_1) \wedge P_{i,h_{j,i_0},f_j}(h(x_k))\big) \\ \wedge & \bigwedge_{\substack{f_j(y) \neq h(x_k) \in \Delta^{\neq} \\ j \geq i_0}} h_{i_0,j}(g(x_1)) \neq h(x_k) \\ \wedge & \bigwedge_{y \neq h(x_k) \in \Delta^{\neq}} \neg\big(f_{i_0}(h(x_k)) = g(x_1) \wedge Q_i(h(x_k))\big) \end{aligned}$$

To see why the rewriting gives the desired result notice that if y is the i^{th} witness of $g(x_1)$, the equality $f_j(y) = h(x_k)$ with $j < i_0$ is equivalent over $f\vec{\mathbf{G}}'$ to $h_{j,i_0}(h(x_k)) = g(x_1) \wedge P_{i,h_{j,i_0},f_j}(h(x_k))$ and the equality $y = h(x_k)$ is equivalent over $f\vec{\mathbf{G}}'$ to $f_{i_0}(h(x_k)) = g(x_1) \wedge Q_i(h(x_k))$. From the definition of p-type, the equality $f_j(y) = h(x_k)$ with $j > i_0$ is equivalent to $h_{i_0,j}(g(x_1)) = h(x_k)$.

• It remains to consider the case when $\Delta^{=}$ is empty. This is a simpler version of the previous case, only this time it is enough to construct a set WITNESS which does not depend on v. It is constructed as in the previous case and verifies: for all tuples $\bar{v}$ of $f\vec{\mathbf{G}}_q$, if $f\vec{\mathbf{G}}_q \models \psi(\bar{v}u)$ for some node u, then there is a node $u' \in$ WITNESS such that $f\vec{\mathbf{G}}_q \models \psi(\bar{v}u')$. Moreover, $|\text{WITNESS}| \leq \gamma_p$. We then argue as in the previous case. □

Example A-5. *Consider one of the quantified formulas as derived by Example A-4:*

$$\exists z \;\; y = h(x) \wedge \tau_{h,f,g}(z) \wedge f(z) = x$$

The resulting quantifier-free query has the form:

$$P(x) \wedge h(x) = y$$

where $P(x)$ is a newly introduced color saying "$\exists z \;\; \tau_{h,f,g}(z) \wedge f(z) = x$". The key point is that this new predicate can be computed in linear time by iterating through all nodes z, testing whether $\tau_{h,f,g}(z)$ is true and, if this is the case, coloring $f(z)$ with color P.

Applying the quantifier elimination process from inside out using Proposition 2 for each step and then applying Lemma 4 to the result yields:

Theorem 2. *Let $\mathcal{C}$ be a class of graphs with bounded expansion. Let $\psi(\bar{x})$ be a query of* FO *over a recoloring of $\sigma_{\mathcal{C}}(0)$ with at least one free variable. Then one can compute a p and a simple quantifier-free formula $\phi(\bar{x})$ over a recoloring of $\sigma_{\mathcal{C}}(p)$ such that:*

$\forall f\vec{\boldsymbol{G}} \in \mathcal{C}$, we can construct in time $O(\|f\vec{\boldsymbol{G}}\|)$ a graph $f\vec{\boldsymbol{G}}' \in \mathcal{C}_p$ such that

$$\phi(f\vec{\boldsymbol{G}}') = \psi(f\vec{\boldsymbol{G}})$$

We will make use of the following useful consequence of Theorem 2:

Corollary 1. *Let $\mathcal{C}$ be a class of graphs with bounded expansion and let $\psi(\bar{x})$ be a formula of* FO *over $\sigma_{\mathcal{C}}(0)$ with at least one free variable. Then, for all $f\vec{\boldsymbol{G}} \in \mathcal{C}$, after a preprocessing in time $O(\|f\vec{\boldsymbol{G}}\|)$, we can test, given $\bar{u}$ as input, whether $f\vec{\boldsymbol{G}} \models \psi(\bar{u})$ in constant time.*

Proof. By Theorem 2 it is enough to consider quantifier-free simple queries. Hence it is enough to consider a query consisting in a single atom of either $P(x)$ or $P(f(x))$ or $x = f(y)$ or $f(x) = g(y)$.

During the preprocessing phase we associate to each node v of the input graph a list $L(v)$ containing all the predicates satisfied by v and all the images of v by a function symbol from the signature. This can be computed in linear time by enumerating all relations of the database and updating the appropriate lists with the corresponding predicate or the corresponding image.

Now, because we use the RAM model, given u we can in constant time recover the list $L(u)$. Using those lists it is immediate to check all atoms of the formula in constant time. □

Theorem 1 is a direct consequence of Theorem 2 and Corollary 1: Starting with a sentence, and applying Theorem 2 for eliminating quantifiers from inside out we end up with a Boolean combination of formulas with one variable. Each such formula can be tested in $O(\|f\vec{\mathbf{G}}\|)$ by iterating through all nodes v of $f\vec{\mathbf{G}}$ and in constant time (using Corollary 1) checking if v can be substituted for the sole existentially quantified variable.

On top of Theorem 1 the following corollary is immediate from Theorem 2 and Corollary 1:

Corollary 2. *Let $\mathcal{C}$ be a class of graphs with bounded expansion and let $\psi(x)$ be a formula of* FO *over $\sigma_{\mathcal{C}}(0)$ with one free variable. Then, for all $f\vec{\boldsymbol{G}} \in \mathcal{C}$, computing the set $\psi(f\vec{\boldsymbol{G}})$ can be done in time $O(\|f\vec{\boldsymbol{G}}\|)$.*

4 Enumeration

In this section we consider first-order formulas with free variables and show that we can enumerate their answers with constant delay over any class with bounded expansion. Moreover, assuming a linear order on the domain of the input structure, we will see that the answers can be output in the lexicographical order. As before we only state the result for graphs, but it immediately extends to arbitrary structures by Lemma 3.

Theorem 3. *Let $\mathcal{C}$ be a class of graphs with bounded expansion and let $\phi(\bar{x})$ be a first-order query. Then the enumeration problem of ϕ over $\mathcal{C}$ is in* CD∘LIN.
Moreover, in the presence of a linear order on the vertices of the input graph, the answers to ϕ can be output in lexicographical order.

The proof of Theorem 3 is by induction on the number of free variables of ϕ. The unary case is done by Corollary 2. The inductive case is a simple consequence of the following:

Proposition 3. *Let $\mathcal{C}$ be a class of graphs with bounded expansion and let $\phi(\bar{x}y)$ be a first-order query or arity 2 or more. Let $\boldsymbol{G}$ be a graph of $\mathcal{C}$. Let $<$ be any linear order on the nodes of $\boldsymbol{G}$. After a preprocessing working in time linear in the size of $\boldsymbol{G}$ we can, on input a tuple $\bar{a}$ of nodes of $\boldsymbol{G}$, enumerate with constant delay and in the order given by $<$ all b such that $\boldsymbol{G} \models \phi(\bar{a}b)$ or answer* NILL *if not such b exists.*

Proof. Fix a class $\mathcal{C}$ of graphs with bounded expansion and a query $\phi(\bar{x}y)$ with $k \geq 2$ free variables. Let $f\vec{\mathbf{G}}$ be the functional representation of the input graph and V be its set of vertices. Let $<$ be any order on V.

During the preprocessing phase, we apply Theorem 2 to get a simple quantifier-free query $\varphi(\bar{x}y)$ and a structure $f\vec{\mathbf{G}}' \in \mathcal{C}_p$, for some p that does not depend on $f\vec{\mathbf{G}}$, such that $\varphi(f\vec{\mathbf{G}}') = \phi(f\vec{\mathbf{G}})$ and $f\vec{\mathbf{G}}'$ can be computed in linear time from $f\vec{\mathbf{G}}$.

Furthermore we normalize the resulting simple quantifier-free query using Proposition 1, and obtain an equivalent quantifier-free formula ψ and a structure $f\vec{\mathbf{G}}'' \in \mathcal{C}_q$, where q depends only on p and φ, $f\vec{\mathbf{G}}''$

can be computed in linear time from $f\vec{\mathbf{G}}'$, $\varphi(f\vec{\mathbf{G}}') = \psi(f\vec{\mathbf{G}}'')$ and ψ is a disjunction of formulas of the form (1):

$$\psi_1(\bar{x}) \wedge \tau(y) \wedge \Delta^{=}(\bar{x}y) \wedge \Delta^{\neq}(\bar{x}y),$$

where $\Delta^{=}(\bar{x}y)$ is either empty or contains one clause of the form $y = f(x_i)$ or one clause of the form $f(y) = g(x_i)$ for some suitable i, f and g; and $\Delta^{\neq}(\bar{x}y)$ contains arbitrarily many clauses of the form $y \neq f(x_i)$ or $f(y) \neq g(x_j)$.

In view of Lemma 1 it is enough to treat each disjunct separately. In the sequel we then assume that ψ has the form described in (1). We let $\psi'(y)$ be the formula $\exists\bar{x}\psi(\bar{x}y)$ and $\psi''(\bar{x})$ the formula $\exists y\psi(\bar{x}y)$.

If $\Delta^{=}$ contains an equality of the form $y = f(x_i)$ we then use Corollary 1 and test whether $f\vec{\mathbf{G}}'' \models \psi(\bar{a}f(a_i))$ and we are done as $f(a_i)$ is the only possible solution for $\bar{a}$.

Assume now that $\Delta^{=}$ is either empty or of the form $f(y) = g(x_i)$.

We first precompute the set of possible candidates for y (i.e. those y satisfying ψ') and distribute this set within their images by f. In other words we define a function $L : V \to 2^V$ such that

$$L(w) = \{u \mid w = f(u) \wedge u \in \psi'(f\vec{\mathbf{G}}'')\}.$$

In the specific case where $\Delta^{=}$ is empty we pick an arbitrary node w_0 in $f\vec{\mathbf{G}}''$ and set $L(w_0) = \psi'(f\vec{\mathbf{G}}'')$ and $L(w) = \emptyset$ for $w \neq w_0$. This can be done in linear time by the following procedure. We first use Corollary 2 and compute in linear time the set $\psi'(f\vec{\mathbf{G}}'')$. We next initialize $L(w)$ to $\emptyset$ for each $w \in V$. Then, for each $u \in \psi'(f\vec{\mathbf{G}}'')$, we add u to the set $L(f(u))$.

Let W be the function from V^{k-1} to V such that $W(\bar{v}) = g(v_i)$. In the specific case where $\Delta^{=}$ is empty we set $W(\bar{v}) = w_0$, where w_0 is the node chosen above.

Notice that for each $\bar{v}u$, $f\vec{\mathbf{G}}'' \models \psi(\bar{v}u)$ implies $u \in L(W(\bar{v}))$ and if $u \in L(W(\bar{v}))$ then $\Delta^{=}(\bar{v}u)$ is true. Hence, given $\bar{a}$ it remains to enumerate within $L(W(\bar{a}))$ the nodes b satisfying $\Delta^{\neq}(\bar{a}b)$.

To do this with constant delay, it will be important to jump from an element u of $L(w)$ to the smallest (according to $<$) element $u' \geq u$ of $L(w)$ satisfying the inequality constraints.

For this we define for $S_1, \ldots, S_{\alpha_{\mathcal{C}}(q)} \subseteq V$ we define $\text{NEXT}_{f_1,S_1,\ldots,f_{\alpha_{\mathcal{C}}(q)},S_{\alpha_{\mathcal{C}}(q)}}(u)$ to be the first element $w \geq u$ of $L(f(u))$[1] such that $f_1(w) \notin S_1, \ldots,$ and $f_{\alpha_{\mathcal{C}}(q)}(w) \notin S_{\alpha_{\mathcal{C}}(q)}$. If such w does not exist, the value of $\text{NEXT}_{f_1,S_1,\ldots,f_{\alpha_{\mathcal{C}}(q)},S_{\alpha_{\mathcal{C}}(q)}}(u)$ is NULL. When all S_i are empty, we write $\text{NEXT}_{\emptyset}(u)$ and by the above definitions we always have $\text{NEXT}_{\emptyset}(u) = u$. We denote such functions as *shortcut pointers of* u. The *size* of a shortcut pointer $\text{NEXT}_{f_1,S_1,\ldots,f_{\alpha_{\mathcal{C}}(q)},S_{\alpha_{\mathcal{C}}(q)}}(u)$ is the sum of sizes of the sets S_i.

In order to avoid writing too long expressions containing shortcut pointers, we introduce the following abbreviations:

- $\text{NEXT}_{f_1,S_1,\ldots,f_{\alpha_{\mathcal{C}}(q)},S_{\alpha_{\mathcal{C}}(q)}}(u)$ is denoted with $\text{NEXT}_{\vec{S}}(u)$,
- $\text{NEXT}_{f_1,S_1,\ldots,f_i,S_i\cup\{u_i\},\ldots,f_{\alpha_{\mathcal{C}}(q)},S_{\alpha_{\mathcal{C}}(q)}}(u)$ is denoted with $\text{NEXT}_{\vec{S}[S_i+=\{u_i\}]}(u)$.

Set $\gamma_q = (k-1) \cdot \alpha_{\mathcal{C}}(q)^2$.

Computing all shortcut pointers of size γ_q would take more than linear time. We therefore only compute a subset of those, denoted SC_L, that will be sufficient for our needs. SC_L is defined in an inductive manner. For all u such that $u \in L(f(u))$, $\text{NEXT}_{\emptyset}(u) \in \text{SC}_L$. Moreover, if the shortcut pointer $\text{NULL} \neq \text{NEXT}_{\vec{S}}(u) \in \text{SC}_L$ and has a size smaller than γ_q, then, for each i,$\text{NEXT}_{\vec{S}[S_i+=\{u_i\}]}(u) \in \text{SC}_L$, where $u_i = f_i(\text{NEXT}_{\vec{S}}(u))$. We then say that $\text{NEXT}_{\vec{S}}(u)$ is the *origin* of $\text{NEXT}_{\vec{S}[S_i+=\{u_i\}]}(u)$. Note that SC_L contains all the shortcut pointers of the form $\text{NEXT}_{f_i,\{f_i(u)\}}(u)$ for $u \in L(f(u))$ and these are exactly the shortcut pointers of u of size 1. By $\text{SC}_L(u) \subseteq \text{SC}_L$ we denote the shortcut pointers of u that are in SC_L.

The set SC_L contains only a constant number of shortcut pointers for each node u.

Claim 1. *There exists a constant $\zeta(q,k)$ such that for every node u we have $|\text{SC}_L(u)| \leq \zeta(q,k)$.*

[1] In order to simplify the notations we consider explicitly the case where $\Delta^{=}$ is not empty. If empty then $L(f(u))$ should be replaced by $L(w_0)$.

Proof. The proof is a direct consequence of the recursive definition of $\mathrm{SC}_L(u)$. Fix u. Note that there is exactly 1 shortcut pointer of u of size 0 (namely $\mathrm{NEXT}_\emptyset(u)$) and $\alpha_{\mathcal{C}}(q)$ shortcut pointers of u of size 1. By the definition of SC_L, any shortcut pointer $\mathrm{NEXT}_{\vec{S}}(u)$ can be an origin of up to $\alpha_{\mathcal{C}}(q)$ shortcut pointers of the form $\mathrm{NEXT}_{\vec{S}[S_i+=\{u_i\}]}(u)$, where $u_i = f_i(\mathrm{NEXT}_{\vec{S}}(u))$ and the size of $\mathrm{NEXT}_{\vec{S}[S_i+=\{u_i\}]}(u)$ is the size of $\mathrm{NEXT}_{\vec{S}}(u)$ plus 1. This way we see that $\mathrm{SC}_L(u)$ contains up to $\alpha_{\mathcal{C}}(q)^2$ shortcut pointers of size 2 and, in general, up to $\alpha_{\mathcal{C}}(q)^s$ shortcut pointers of size s. As the maximal size of a computed shortcut pointer is bounded by γ_q, we have $|\mathrm{SC}_L(u)| \leq \sum_{0\leq i\leq \gamma_q} \alpha_{\mathcal{C}}(q)^i$. Both $\alpha_{\mathcal{C}}(q)$ and γ_q depend only on q and k, which concludes the proof. □

Moreover SC_L contains all what we need to know.

Claim 2. *Let $\mathrm{NEXT}_{\vec{S}}(u)$ be a shortcut pointer of size not greater than γ_q. Then there exists $\mathrm{NEXT}_{\vec{S'}}(u) \in \mathrm{SC}_L$ such that $\mathrm{NEXT}_{\vec{S}}(u) = \mathrm{NEXT}_{\vec{S'}}(u)$. Moreover, such $\mathrm{NEXT}_{\vec{S'}}(u)$ can be found in constant time.*

Proof. If $\mathrm{NEXT}_{\vec{S}}(u) \in \mathrm{SC}_L$, then we have nothing to prove. Assume then that $\mathrm{NEXT}_{\vec{S}}(u) \notin \mathrm{SC}_L$. We write $\mathrm{NEXT}_{f_1,S'_1,\ldots,f_{\alpha_{\mathcal{C}}(q)},S'_{\alpha_{\mathcal{C}}(q)}}(u) \preceq \mathrm{NEXT}_{f_1,S_1,\ldots,f_{\alpha_{\mathcal{C}}(q)},S_{\alpha_{\mathcal{C}}(q)}}(u)$ if for each $1 \leq i \leq \alpha_{\mathcal{C}}(q)$ we have $S'_i \subseteq S_i$. Note that for a given u and v the $\preceq$ relation is a partial order on the set of shortcut pointers of u. A trivial observation is that if $\mathrm{NEXT}_{f_1,S'_1,\ldots,f_{\alpha_{\mathcal{C}}(q)},S'_{\alpha_{\mathcal{C}}(q)}}(u) \preceq \mathrm{NEXT}_{f_1,S_1,\ldots,f_{\alpha_{\mathcal{C}}(q)},S_{\alpha_{\mathcal{C}}(q)}}(u)$, then $\mathrm{NEXT}_{f_1,S'_1,\ldots,f_{\alpha_{\mathcal{C}}(q)},S'_{\alpha_{\mathcal{C}}(q)}}(u) \leq \mathrm{NEXT}_{f_1,S_1,\ldots,f_{\alpha_{\mathcal{C}}(q)},S_{\alpha_{\mathcal{C}}(q)}}(u)$.

Let $\mathrm{NEXT}_{\vec{S'}}(u) \in \mathrm{SC}_L$ be a maximal in terms of size shortcut pointer of u such that $\mathrm{NEXT}_{\vec{S'}}(u) \preceq \mathrm{NEXT}_{\vec{S}}(u)$. Such a shortcut pointer always exists as $\mathrm{NEXT}_\emptyset(u) \preceq \mathrm{NEXT}_{\vec{S}}(u)$ and $\mathrm{NEXT}_\emptyset(u) \in \mathrm{SC}_L$. Note that the size of $\mathrm{NEXT}_{\vec{S'}}(u)$ is strictly smaller than the size of $\mathrm{NEXT}_{\vec{S}}(u)$, so it is strictly smaller than γ_q. One can find $\mathrm{NEXT}_{\vec{S'}}(u)$ by exploring all the shortcut pointers of u in $\mathrm{SC}_L(u)$. This can be done in constant time using Claim 1.

We now claim that $\mathrm{NEXT}_{\vec{S}}(u) = \mathrm{NEXT}_{\vec{S'}}(u)$.

Let $v = \mathrm{NEXT}_{\vec{S'}}(u)$. We know that $v \leq \mathrm{NEXT}_{\vec{S}}(u)$. Assume now that there would exists $1 \leq i \leq \alpha_{\mathcal{C}}(q)$ such that $f_i(v) \in S_i$. Then we have that $\mathrm{NEXT}_{\vec{S'}[S'_i+=\{u_i\}]}(u) \in \mathrm{SC}_L$, where $u_i = f_i(v)$, and this contradicts the maximality of $\mathrm{NEXT}_{\vec{S'}}(u)$. This means that such an i does not exist and concludes the fact that $\mathrm{NEXT}_{\vec{S}}(u) = \mathrm{NEXT}_{\vec{S'}}(u)$. □

The following claim guarantees that SC_L can be computed in linear time and has therefore a linear size.

Claim 3. *SC_L can be computed in time linear in $\|f\vec{\boldsymbol{G}}''\|$.*

Proof. Fix v and consider $M = L(v)$. SC_L can be constructed in an inductive manner starting from the last node in M and moving backward. Claim 2 plays the key role in constructing each shortcut pointer in constant time, while Claim 1 guarantees that the total size of SC_L is linear in $|M|$. Hence for all nodes u of M, $\mathrm{SC}_L(u)$ can be computed in time linear in M. Hence a total time linear in V as desired.

In linear time we set $\mathrm{NEXT}_\emptyset(u) = u$ for $u \in L(f(u))$.

We do the computation of $\mathrm{NEXT}_{\vec{S}}(u) \in \mathrm{SC}_L$ for $u \in M$ in an inductive manner starting from the last node on M and move backwards. If u is the last node on M then we are already done as all the shortcut pointers of u are NULL. We now assume that u is not last on M and that for all $v > u$ set $\mathrm{SC}_L(v)$ is computed. We show how to compute $\mathrm{SC}_L(u)$.

Consider now $\mathrm{NEXT}_{\vec{S}}(u)$. If $\forall i \;\; f_i(u) \notin S_i$ then we are done, as $\mathrm{NEXT}_{\vec{S}}(u) = u$. Otherwise let v be the successor of u in M. Clearly $v \leq \mathrm{NEXT}_{\vec{S}}(u)$ and $\mathrm{NEXT}_{\vec{S}}(u) = \mathrm{NEXT}_{\vec{S}}(v)$. We can conclude this case $\mathrm{NEXT}_{\vec{S}}(v) = \mathrm{NEXT}_{\vec{S'}}(v)$, where $\mathrm{NEXT}_{\vec{S'}}(v) \in \mathrm{SC}_L(v)$ is the shortcut pointer of v from the application of Claim 2 to $\mathrm{NEXT}_{\vec{S}}(v)$. Claim 2 assures that we can find $\mathrm{NEXT}_{\vec{S'}}(v)$ in constant time and thus $\mathrm{NEXT}_{\vec{S}}(u)$ is computed in constant time. As Claim 1 shows that we only need to consider constantly many shortcut pointers for each u, the whole process takes time $O(|M|)$. □

The computation of SC_L concludes the preprocessing phase and it follows from Claim 3 that it can be done in linear time. We now turn to the enumeration phase.

Assume we are given $\bar{a}$. In view of Corollary 1 we can without loss of generality that $\bar{a}$ is such that $\mathbf{G} \models \psi''(\bar{a})$. If not we simply return NILL and stop here.

By construction we know that all nodes b such that $f\vec{\mathbf{G}}'' \models \psi(\bar{a}b)$ are in $L = L(W(\bar{a}))$. Recall also that all elements $b \in L$ make $\tau(b) \wedge \Delta^{=}(\bar{a}b)$ true. For $1 \leq i \leq \alpha_{\mathcal{C}}(q)$ we set $S_i = \{g(v_j) : g(x_j) \neq f_i(y)$ is a conjunct of $\Delta^{\neq}\}$. Starting with b the first node of the sorted list L, we apply the following procedure:

1. If b is not NULL, let $\text{NEXT}_{\vec{S}'}(b)$ be the shortcut pointer from the application of Claim 2 to $\text{NEXT}_{\vec{S}}(b)$. Set $b' = \text{NEXT}_{\vec{S}'}(b)$. If $b' =$ NULL, stop here.
2. If $f\vec{\mathbf{G}}'' \models \psi(\bar{a}b')$, output b'.
3. Reinitialize b to the successor of b' in L and continue with Step 1.

We now show that the algorithm is correct.

The algorithm clearly outputs only solutions as it tests whether $f\vec{\mathbf{G}}'' \models \psi(\bar{a}b')$ before outputting b'.

By the definition of sets S_i and $\text{NEXT}_{\vec{S}}(b)$, for each $b \leq v < b'$ there is a suitable i and j such that $g(a_j) = f_i(v)$ and $g(x_j) \neq f_i(y)$ is a conjunct of $\Delta^{\neq}$. This way the algorithm does not skip any solutions at Step 1 and so it outputs exactly all solutions.

It remains to show that there is a constant time between any two outputs. Step 1 takes constant time due to Claim 2. From there the algorithm either immediately outputs a solution at Step 2 or jumps to Step 3. In the second case, this means that $f\vec{\mathbf{G}}'' \not\models \psi(\bar{a}b')$, but from the definitions of list L, sets S_i and shortcut pointers $\text{NEXT}_{\vec{S}}(b)$ this can only happen if $\Delta^{\neq}$ is falsified and this is because of an inequality of the form $y \neq g(x_j)$ for some suitable g and j (where g may possibly be identity). Hence $b' = g(a_j)$. As all the elements on L are distinct, the algorithm can skip over Step 2 up to $(k-1) \cdot (\alpha_{\mathcal{C}}(q) + 1)$ times for each tuple $\bar{a}$ (there are up to that many different images of nodes from $\bar{a}$ under $\alpha_{\mathcal{C}}(q)$ different functions). The delay is therefore bounded by $k \cdot (\alpha_{\mathcal{C}}(q) + 1)$ consecutive applications of Claim 2.

As the list L was sorted with respect to the linear order on the domain, it is clear that the enumeration procedure outputs the set of solutions in lexicographical order.

This concludes the proof of the theorem. □

5 Counting

In this section we investigate the problem of counting the number of solutions to a query, i.e. computing $|q(\mathbf{D})|$. As usual we only state and prove our results over graphs but they generalize to arbitrary relational structures via Lemma 3.

Theorem 4. *Let $\mathcal{C}$ be class of graphs with bounded expansion and let $\phi(\bar{x})$ be a first-order formula. Then, for all $f\vec{\boldsymbol{G}} \in \mathcal{C}$, we can compute $|\phi(f\vec{\boldsymbol{G}})|$ in time $O(\|f\vec{\boldsymbol{G}}\|)$.*

Proof. The key idea is to prove a weighted version of the desired result. Assume $\phi(\bar{x})$ has exactly k free variables and for $1 \leq i \leq k$ we have functions $\#_i : V \to \mathbb{N}$. We will compute in time linear in $\|f\vec{\mathbf{G}}\|$ the following number:

$$|\phi(f\vec{\mathbf{G}})|_{\#} := \sum_{\bar{u} \in \phi(f\vec{\mathbf{G}})} \prod_{1 \leq i \leq k} \#_i(u_i).$$

By setting all $\#_i$ to be constant functions with value 1 we get the regular counting problem. Hence Theorem 4 is an immediate consequence of the next lemma.

Lemma 6. *Let $\mathcal{C}$ be class of graphs with bounded expansion and let $\phi(\bar{x})$ be a first-order formula with exactly k free variables.*
For $1 \leq i \leq k$ let $\#_i : V \to \mathbb{N}$ be functions such that for each v the value of $\#_i(v)$ can be computed in constant time.
Then, for all $f\vec{\boldsymbol{G}} \in \mathcal{C}$, we can compute $|\phi(f\vec{\boldsymbol{G}})|_{\#}$ in time $O(\|f\vec{\boldsymbol{G}}\|)$.

Proof. The proof is by induction on the number of free variables.

The case $k = 1$ is trivial: in time linear in $\|f\vec{\mathbf{G}}\|$ we compute $\phi(f\vec{\mathbf{G}})$ using Corollary 2. By hypothesis, for each $v \in \phi(f\vec{\mathbf{G}})$, we can compute the value of $\#_1(v)$ in constant time. Therefore the value

$$|\phi(f\vec{\mathbf{G}})|_{\#} = \sum_{v \in \phi(f\vec{\mathbf{G}})} \#_1(v)$$

can be computed in linear time as desired.

Assume now that $k > 1$ and that $\bar{x}$ and y are the free variables of ϕ, where $|\bar{x}| = k - 1$.

We apply Theorem 2 to get a simple quantifier-free query $\varphi(\bar{x}y)$ and a structure $f\vec{\mathbf{G}}' \in \mathcal{C}_p$, for some p that does not depend on $f\vec{\mathbf{G}}$, such that $\varphi(f\vec{\mathbf{G}}') = \phi(f\vec{\mathbf{G}})$ and $f\vec{\mathbf{G}}'$ can be computed in linear time from $f\vec{\mathbf{G}}$. Note that $|\phi(f\vec{\mathbf{G}})|_{\#} = |\varphi(f\vec{\mathbf{G}}')|_{\#}$, so it is enough to compute the latter value.

We normalize the resulting simple quantifier-free query using Proposition 1, and obtain an equivalent quantifier-free formula ψ and a structure $f\vec{\mathbf{G}}'' \in \mathcal{C}_q$, where q depends only on p and φ, $f\vec{\mathbf{G}}''$ can be computed in linear time from $f\vec{\mathbf{G}}'$, $\varphi(f\vec{\mathbf{G}}') = \psi(f\vec{\mathbf{G}}'')$ and ψ is a disjunction of formulas of the form (1):

$$\psi_1(\bar{x}) \wedge \tau(y) \wedge \Delta^{=}(\bar{x}y) \wedge \Delta^{\neq}(\bar{x}y),$$

where $\Delta^{=}(\bar{x}y)$ is either empty or contains one clause of the form $y = f(x_i)$ or one clause of the form $f(y) = g(x_i)$ for some suitable i, f and g; and $\Delta^{\neq}(\bar{x}y)$ contains arbitrarily many clauses of the form $y \neq f(x_i)$ or $f(y) \neq g(x_j)$. Note that $|\varphi(f\vec{\mathbf{G}}')|_{\#} = |\psi(f\vec{\mathbf{G}}'')|_{\#}$, so it is enough to compute the latter value.

Observe that it is enough to solve the weighted counting problem for each disjunct separately, as we can then combine the results using a simple inclusion-exclusion reasoning (the weighted sum for $q \vee q'$ is obtained by adding the weighted sum for q to the weighted sum for q' and then subtracting the weighted sum for $q \wedge q'$). In the sequel we then assume that ψ has the form described in (1).

The proof now goes by induction on the number of inequalities in $\Delta^{\neq}$. While the inductive step turns out to be fairly easy, the difficult part is the base step of the induction.

We start with proving the inductive step. Let $g(y) \neq f(x_i)$ be an arbitrary inequality from $\Delta^{\neq}$ (where g might possibly be the identity). Let ψ^- be ψ with this inequality removed and $\psi^+ = \psi^- \wedge g(y) = f(x_i)$. Of course ψ and ψ^+ have disjoint sets of solutions and we have:

$$|\psi(f\vec{\mathbf{G}}'')|_{\#} = |\psi^-(f\vec{\mathbf{G}}'')|_{\#} - |\psi^+(f\vec{\mathbf{G}}'')|_{\#}.$$

Note that ψ^- and ψ^+ have one less conjunct in $\Delta^{\neq}$. The problem is that ψ^+ is not of the form (1) as it may now contain two elements in $\Delta^{=}$. However it can be seen that the removal of the extra equality in $\Delta^{=}$ as described in the proof of Proposition 1 does not introduce any new elements in $\Delta^{\neq}$.

Claim 4. . *There exists a query ψ^+_{NF} such that: its size depends only on the size of ψ^+, ψ^+_{NF} is in the normal form given by* (1)*, it contains an inequality conjunct $h(y) \neq g_1(x_i)$ (where h might possibly be identity) iff ψ^+ also contains such conjunct and $\psi^+_{\text{NF}}(f\vec{\mathbf{G}}'') = \psi^+(f\vec{\mathbf{G}}'')$. Moreover, ψ^+_{NF} can be constructed in time linear in the size of ψ^+.*

Proof. The proof is a simple case analysis of the content of $\Delta^{=}$ of ψ.

If its empty, then ψ^+_{NF} is already in the desired form.

If it contains an atom of the form $y = h_2(x_j)$, then equality $g(y) = f(x_i)$ is equivalent to $g(h_2(x_j)) = f(x_i)$ and we are done.

If it contains an atom of the form $h_3(y) = h_2(x_j)$ and g is identity, then $h_3(y) = h_2(x_j)$ is equivalent to $h_3(f(x_i)) = h_2(x_j)$. If g is not identity, then $\tau(y)$ ensures us that either $g(y)$ determines $h_3(y)$ or vice versa. If we have $h_4(g(y)) = h_3(y)$, then $h_3(y) = h_2(x_j)$ is equivalent to $h_4(f(x_i)) = h_2(x_j)$. The other case is symmetric.

The fact that ψ^+_{NF} does not contain any additional inequalities, that it can be computed in time linear in the size of ψ^+ and that $\psi^+_{\text{NF}}(f\vec{\mathbf{G}}'') = \psi^+(f\vec{\mathbf{G}}'')$ follows from the above construction. □

We can therefore remove the extra element in Δ^+ and assume that ψ^+ has the desired form. We can now use the inductive hypothesis on the size of $\Delta^{\neq}$ to both ψ^- and ψ^+ in order to compute both $|\psi^-(f\vec{\mathbf{G}}'')|_{\#}$ and $|\psi^+(f\vec{\mathbf{G}}'')|_{\#}$ and derive $|\psi(f\vec{\mathbf{G}}'')|_{\#}$.

It remains to show the base of the inner induction. In the following we assume that $\Delta^{\neq}$ is empty. The rest of the proof is a case analysis on the content of $\Delta^{=}$.

Assume first that $\Delta^{=}$ consists of an atom of the form $y = f(x_1)$.

Note that the solutions to ψ are of the form $(\bar{v}f(v_1))$. We have:

$$\begin{aligned}
|\psi(f\vec{\mathbf{G}}'')|_{\#} &= \sum_{(\bar{v}u)\in\psi(f\vec{\mathbf{G}}'')} \left(\#_k(u) \prod_{1\leq i\leq k-1} \#_i(v_i) \right) \\
&= \sum_{(\bar{v}f(v_1))\in\psi(f\vec{\mathbf{G}}'')} \left(\#_k(f(v_1)) \prod_{1\leq i\leq k-1} \#_i(v_i) \right) \\
&= \sum_{(\bar{v}f(v_1))\in\psi(f\vec{\mathbf{G}}'')} \left(\#_1(v_1)\#_k(f(v_1)) \prod_{2\leq i\leq k-1} \#_i(v_i) \right)
\end{aligned}$$

In linear time we now iterate through all nodes w in $f\vec{\mathbf{G}}''$ and set

$$\begin{aligned}
\#'_1(w) &:= \#_1(w) \cdot \#_k(f(w)) \\
\#'_i(w) &:= \#_i(w) \qquad\qquad \text{for } 2 \leq i \leq k-1.
\end{aligned}$$

Let $\vartheta(\bar{x})$ be ψ with all occurrences of y replaced with $f(x_1)$. We then have:

$$\begin{aligned}
|\psi(f\vec{\mathbf{G}}'')|_{\#} &= \sum_{(\bar{v}f(v_1))\in\psi(f\vec{\mathbf{G}}'')} \left(\#'_1(v_1) \prod_{2\leq i\leq k-1} \#'_i(v_i) \right) \\
&= \sum_{\bar{v}\in\vartheta(f\vec{\mathbf{G}}'')} \prod_{1\leq i\leq k-1} \#'_i(v_i) \\
&= |\vartheta(f\vec{\mathbf{G}}'')|_{\#'}
\end{aligned}$$

By induction on the number of free variables, as $\#'_i(w)$ can be computed in constant time for each i and w, we can compute $|\vartheta(f\vec{\mathbf{G}}'')|_{\#'}$ in time linear in $\|f\vec{\mathbf{G}}''\|$ and we are done.

Assume now that $\Delta^{=}$ consists of an atom $g(y) = f(x_1)$. Let $\psi'(y)$ be the formula $\exists\bar{x}\psi(\bar{x}y)$ and $\psi''(\bar{x})$ the formula $\exists y\psi(\bar{x}y)$. We first compute set $\psi'(f\vec{\mathbf{G}}'')$ in linear time using Corollary 2. We now define a function $\#''_k : V \to \mathbb{N}$ as:

$$\#''_k(w) := \sum_{\substack{\{u\in\psi'(f\vec{\mathbf{G}}'')\\ g(u)=w\}}} \#_k(u).$$

Note that this function can be easily computed in linear time by going through all nodes w and adding $\#_k(w)$ to $\#''_k(g(w))$.

Finally we set:

$$\begin{aligned}
\#'_1(w) &:= \#_1(w)\#''_k(f(w)) \\
\#'_i(w) &:= \#_i(w) \qquad\qquad \text{for } 2 \leq i \leq k-1.
\end{aligned}$$

Let $u_1, u_2 \in \psi'(f\vec{\mathbf{G}}'')$ be such that $g(u_1) = g(u_2)$. Because $\Delta^{\neq}$ is empty, observe that $f\vec{\mathbf{G}}'' \models \forall\bar{x}(\psi(\bar{x}u_1) \leftrightarrow \psi(\bar{x}u_2))$. Based on this observation we now group the solutions to ψ according to their last $k-1$ values and get:

$$
\begin{aligned}
|\psi(f\vec{\mathbf{G}}'')|_{\#} &= \sum_{(\bar{v}u)\in\psi(f\vec{\mathbf{G}}'')} \left(\#_k(u) \prod_{1\le i\le k-1} \#_i(v_i) \right) \\
&= \sum_{\bar{v}\in\psi''(f\vec{\mathbf{G}}'')} \sum_{\substack{\{u\in\psi'(f\vec{\mathbf{G}}'')\\ g(u)=f(v_1)\}}} \left(\#_k(u) \prod_{1\le i\le k-1} \#_i(v_i) \right) \\
&= \sum_{\bar{v}\in\psi''(f\vec{\mathbf{G}}'')} \left(\sum_{\substack{\{u\in\psi'(f\vec{\mathbf{G}}'')\\ g(u)=f(v_1)\}}} \#_k(u) \right) \prod_{1\le i\le k-1} \#_i(v_i) \\
&= \sum_{\bar{v}\in\psi''(f\vec{\mathbf{G}}'')} \left(\#''_k(f(v_1)) \prod_{1\le i\le k-1} \#_i(v_i) \right) \\
&= \sum_{\bar{v}\in\psi''(f\vec{\mathbf{G}}'')} \left(\#_1(v_1)\#''_k(f(v_1)) \prod_{2\le i\le k-1} \#'_i(v_i) \right) \\
&= \sum_{\bar{v}\in\psi''(f\vec{\mathbf{G}}'')} \prod_{1\le i\le k-1} \#'_i(v_i) \\
&= |\psi''(f\vec{\mathbf{G}}'')|_{\#'}
\end{aligned}
$$

By induction on the number of free variables, as $\#'_i(w)$ can be computed in constant time for each i and w, we can compute $|\psi''(f\vec{\mathbf{G}}'')|_{\#'}$ and we are done with this case.

The remaining case when $\Delta^{=}$ is empty is handled similarly to the previous one. We then have

$$\psi(\bar{x}y) = \psi_1(\bar{x}) \wedge \tau(y).$$

After setting

$$
\begin{aligned}
\#'_1(w) &:= \#_2(w) \cdot \sum_{u\in\tau(f\vec{\mathbf{G}}'')} \#_1(u) \\
\#'_i(w) &:= \#_{i+1}(w) \qquad\qquad \text{for } 2 \le i \le k-1
\end{aligned}
$$

we see that

$$|\psi(f\vec{\mathbf{G}}'')|_{\#} = |\psi_1(f\vec{\mathbf{G}}'')|_{\#'}$$

and we conclude again by induction on the number of free variables. □

As we said earlier, Theorem 4 is an immediate consequence of Lemma 6. □

6 Conclusions

Queries written in first-order logic can be efficiently processed over the class of structures having bounded expansion. We have seen that over this class the problems investigated in this paper can be computed in time linear in the size of the input structure. The constant factor however is not very good. The approach taken here, as well as the ones of [9, 11], yields a constant factor that is a tower of exponentials whose height depends on the size of the query. This nonelementary constant factor is unavoidable already on the class of unranked trees, assuming FPT≠AW[∗] [10]. In comparison, this factor can be triply exponential in the size of the query in the bounded degree case [21, 14].

It is possible that the results presented here can be generalized to a larger class of structures. In [19] the class of nowhere dense graphs was introduced and it generalizes the notion of bounded expansion.

Recently it has been shown that the model checking problem of first-order logic can be done in nearly linear time (i.e. for any $\epsilon > 0$ it can be done in $O(n^{1+\epsilon})$) over any nowhere dense class of graph [12]. It remains to extend this to constant delay enumeration.

The class of nowhere dense structures seems to be the limit for having good algorithmic properties for first-order logic. Indeed, it is known that the model checking problem of first-order logic over a class of structures that is not nowhere dense cannot be FPT [16] (modulo some complexity assumptions and closure of the class under substructures).

For structures of bounded expansion, an interesting open question is whether a sampling of the solutions can be performed in linear time. For instance: can we compute the j-th solution in constant time after a linear preprocessing? This can be done in the bounded degree case [6] and in the bounded treewidth case [4]. We leave the bounded expansion case for future research.

References

[1] Noga Alon, Raphael Yuster, and Uri Zwick. Color-Coding. *J. ACM*, 42(4):844–856, 1995.

[2] Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy Problems for Tree-Decomposable Graphs. *J. of Algorithms*, 12(2):308–340, 1991.

[3] Guillaume Bagan. MSO Queries on Tree Decomposable Structures Are Computable with Linear Delay. In *Conf. on Computer Science Logic (CSL)*, pages 167–181, 2006.

[4] Guillaume Bagan. *Algorithmes et complexité des problèmes d'énumération pour l'évaluation de requêtes logiques*. PhD thesis, Université de Caen, 2009.

[5] Guillaume Bagan, Arnaud Durand, and Etienne Grandjean. On Acyclic Conjunctive Queries and Constant Delay Enumeration. In *Conf. on Computer Science Logic (CSL)*, pages 208–222, 2007.

[6] Guillaume Bagan, Arnaud Durand, Etienne Grandjean, and Frédéric Olive. Computing the jth solution of a first-order query. *RAIRO Theoretical Informatics and Applications*, 42(1):147–164, 2008.

[7] Bruno Courcelle. Graph Rewriting: An Algebraic and Logic Approach. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, pages 193–242. 1990.

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

[9] Zdeněk Dvořák, Daniel Král, and Robin Thomas. Deciding First-Order Properties for Sparse Graphs. In *Symp. on Foundations of Computer Science (FOCS)*, pages 133–142, 2010.

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

[11] Martin Grohe and Stephan Kreutzer. *Model Theoretic Methods in Finite Combinatorics*, chapter Methods for Algorithmic Meta Theorems. American Mathematical Society, 2011.

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

[13] Wojciech Kazana. *Query evaluation with constant delay. (L'évaluation de requêtes avec un délai constant)*. PhD thesis, École normale supérieure de Cachan, Paris, France, 2013.

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

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

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

[17] Jaroslav Nešetřil and Patrice Ossona de Mendez. Grad and classes with bounded expansion I. Decompositions. *Eur. J. Comb.*, 29(3):760–776, 2008.

[18] Jaroslav Nešetřil and Patrice Ossona de Mendez. Grad and classes with bounded expansion II. Algorithmic aspects. *Eur. J. Comb.*, 29(3):777–791, 2008.

[19] Jaroslav Nešetřil and Patrice Ossona de Mendez. On nowhere dense graphs. *European J. of Combinatorics*, 32(4):600–617, 2011.

[20] Christos H. Papadimitriou and Mihalis Yannakakis. On the Complexity of Database Queries. *J. on Computer and System Sciences (JCSS)*, 58(3):407–427, 1999.

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