Constant Delay Enumeration for FO Queries over Databases with Local Bounded Expansion

Luc Segoufin, Alexandre Vigny

Constant delay enumeration for FO queries over databases with local bounded expansion

Luc Segoufin
INRIA and ENS Ulm
Paris

Alexandre Vigny
Université Paris Diderot Paris 7
Paris

ABSTRACT

We consider the evaluation of first-order queries over classes of databases with *local bounded expansion*. This class was introduced by Nešetřil and Ossona de Mendez and generalizes many well known classes of databases, such as bounded degree, bounded tree width or bounded expansion. It is known that over classes of databases with local bounded expansion, first-order sentences can be evaluated in pseudo-linear time (pseudo-linear time means that for all ϵ there exists an algorithm working in time $O(n^{1+\epsilon})$). Here, we investigate other scenarios, where queries are not sentences. We show that first-order queries can be enumerated with constant delay after a pseudo-linear preprocessing over any class of databases having locally bounded expansion. We also show that, in this context, counting the number of solutions can be done in pseudo-linear time.

KEYWORDS

Enumeration, First-Order queries, Local bounded expansion.

1 INTRODUCTION

Query evaluation is a fundamental task in databases and a vast literature is devoted to the complexity of this problem. Given a database $\mathbf{D}$ and a query q the goal is to compute the set $q(\mathbf{D})$ of all solutions for q over $\mathbf{D}$. Unfortunately, the set $q(\mathbf{D})$ might be way bigger than the database itself as the number of solutions could be exponential in the arity of the query. It can therefore be unrealistic to compute all solutions, even for small queries. One could imagine many scenarios to overcome this situation. We could for instance only want to compute the number of solutions or just compute the k most relevant solutions relative to some ranking function.

We consider here the complexity of the enumeration of the set $q(\mathbf{D})$, i.e. generating one by one all the solutions for q over $\mathbf{D}$. In this context two parameters play an important role. The first one is the *preprocessing time*, i.e. the time it takes to produce the first solution. The second one is the *delay*, i.e. the maximum time between the output of any two consecutive solutions. An enumeration algorithm is then said to be *efficient* if these two parameters are small. For the delay, the best we can hope for is constant time: depending only on the query and independent from the size of the database. For the preprocessing time an ideal goal would be linear time: linear in the size of the database with a constant factor depending on the query. When both are achieved we say that the query can be enumerated with constant delay after linear preprocessing.

Constant delay enumeration after linear preprocessing cannot be achieved for all queries. However, for restricted classes of queries and databases several efficient enumeration algorithms have been obtained. This is the case for instance for first-order (FO) queries over databases with bounded degree [3, 11], monadic second-order (MSO) queries over databases with bounded tree-width [2, 13] and FO queries over databases with bounded expansion [12]. Bounded expansion is a large class of databases as it contains in particular all structures excluding at least one minor (planarity, bounded tree-width etc.) and all structures of bounded degree [16].

In some scenarios only pseudo-linear preprocessing time has been achieved. A query can be enumerated with constant delay after a pseudo-linear preprocessing time if for all ϵ there exists an enumeration procedure with constant delay and preprocessing time in $O(\|\mathbf{D}\|^{1+\epsilon})$. This is the case for FO queries over databases with low degree [4].

A special case of enumeration is when the query is boolean. In this case the preprocessing computes the answer to the query. In order to be able to enumerate queries of a given language efficiently, it is therefore necessary to be able to solve the boolean case efficiently.

It has been shown recently that boolean FO queries could be computed in pseudo-linear time over nowhere dense databases [9]. Nowhere dense is an important class of databases generalizing bounded expansion [16]. Among the classes of databases closed under sub-databases, Nowhere dense is the largest possible class enjoying efficient evaluation for FO queries [14].

It's a major open problem to show that over nowhere dense databases the boolean case can be extended to a constant delay enumeration for FO queries of higher arities.

In this paper we make one step towards solving this problem, extending the bounded expansion result to databases having local bounded expansion. Local bounded expansion lies strictly between bounded expansion and nowhere dense. It requires that for all r the class of neighbors of radius r has bounded expansion. It contains for instance all databases having local bounded tree-width, or excluding locally a minor. It strictly extends bounded expansion as there exist classes of local bounded tree-width that do not have bounded expansion [7].

For FO queries over a class of databases with local bounded expansion we provide:

- an enumeration procedure with constant delay after pseudo-linear preprocessing,
- a pseudo-linear time algorithm counting the number of solutions.

Our proof for enumeration follows a classical scheme. Our first ingredient is Gaifman's theorem, decomposing a formula into local ones with distance constraints. In order to evaluate the local formulas we would need to compute local neighborhoods. However this would not be linear as each neighborhood may be of linear size and we have linearly many of them. Our second key ingredient is the result that one can compute in pseudo-linear time a representative "cover" of the database by means of neighborhoods [9].

Because these neighborhoods have bounded expansion we can use the bounded expansion case in order to evaluate the local formulas. It remains to take care of the distance constraints and this is the main technical contribution of this paper.

The paper is organized as follows. We start by giving a new proof of the boolean case in Section 5. We then extend it to constant delay enumeration in Section 6 and to counting in Section 7.

Related work. Our presentation for the model checking, Section 5, uses the same tricks that were used in [7] to lift the model checking from the bounded tree-width case to the local bounded tree-width case. The model checking results presented in Section 5 were already obtained in [5] with a very similar argument. We give the proofs again here for completeness and in order to fix the notations.

An algorithm for counting in linear time the number of solutions for FO queries over classes of databases with "nice" local bounded tree-width was presented in [6]. The restriction "nice" requires that the neighborhood cover can be computed in linear time and that one part of the cover intersects only a constant number of other parts. It is more restrictive than the one we use, given by [9], and is designed to make the counting easy with a simple exclusion/inclusion argument. This argument does not seem to extend to the cover we have and our algorithm for counting, presented in Section 7, is done by induction on the number of free variables.

In [10] a labeling scheme was presented for first-order queries over graphs with "nice" local bounded tree-width. Although constant delay enumeration may be derived from the labeling scheme, this one is computed in polynomial time while we aim for pseudo-linear time. It is unclear whether this result can be generalized to classes of graphs with local bounded expansion using the tools we develop in this paper.

2 PRELIMINARIES

For a positive integer k, $[k]$ denotes the set $\{1, \ldots, k\}$. Thereafter, ϵ will always denote an element of $\mathbb{R}^+$, p, r, s, i, j, and k positive integers and f a function of $\mathbb{N} \to \mathbb{N}$.

Databases and First-Order queries. A relational signature σ is a tuple $(R_1, \ldots, R_s)$ where each R_i is a relational symbol of arity r_i. By database, we mean a finite structure over a relational signature σ, that is a tuple $\mathbf{D} = (D, R_1^{\mathbf{D}}, \ldots, R_s^{\mathbf{D}})$, where D, the *domain* of $\mathbf{D}$, is a finite set and for each i, $R_i^{\mathbf{D}}$ is a subset of D^{r_i}. If $\mathbf{D}$ is a database and $A \subseteq D$ a subset of its universe, we denote by $\mathbf{D}[A]$ the database given by the substructure of $\mathbf{D}$ induced by A. We fix a classical encoding of structures as input, see for example [1]. We denote by $\|\mathbf{D}\|$ the size of (the encoding of) $\mathbf{D}$. Without loss of generality we assume that the domain D comes with a linear order. If not, we arbitrarily choose one, for instance the one induced by the encoding of $\mathbf{D}$. This order induces a lexicographical order among the tuples over D.

A *query* is a first-order (FO) formula built from atomic formulas, "$x = y$" and $R_i(x_1, \ldots, x_{r_i})$, and closed under boolean combinations, $\wedge, \vee, \neg$, existential and universal quantifications, $\exists, \forall$. We write $q(\bar{x})$ if $\bar{x}$ are the free variables of q. The length of $\bar{x}$ is called the arity of the query. Queries of arity 0 are called sentences. The size of q is written $|q|$.

We write $\mathbf{D} \models q(\bar{a})$ to denote the fact that $\bar{a}$ is a solution for q over $\mathbf{D}$. We write $q(\mathbf{D})$ to denote the set of tuples $\bar{a}$ such that $\mathbf{D} \models q(\bar{a})$.

Given a database $\mathbf{D}$ and a sentence q, the problem of testing whether $\mathbf{D} \models q$ or not is called *the model checking problem*. It may be restricted to a class C of databases.

Model of computation and complexity. As usual when dealing with linear time, we use Random Access Machines (RAM) with addition and uniform cost measure as a model of computation.

All problems encountered in this paper have two inputs, a database $\mathbf{D}$ and a query q. However they play different roles as $\|\mathbf{D}\|$ is large while $|q|$ is small. We therefore consider the data complexity point of view. We say that a problem is *linear time* if it can be solved in time $O(\|\mathbf{D}\|)$. Here, and in the rest of the paper, the constants hidden behind the "big O" depend on q. We say that a problem is *pseudo-linear time* if, for all ϵ, it can be solved in time $O(\|\mathbf{D}\|^{1+\epsilon})$. In this case the constant factor also depends on ϵ. If a subroutine of a procedure depending on ϵ produces an output of size $O(\|\mathbf{D}\|^{\epsilon})$ we will then say that the output is *pseudo-constant*.

Neighborhoods and bounded expansion. Fix a database $\mathbf{D}$ of domain D. The *Gaifman graph* of $\mathbf{D}$ is the non-directed graph whose set of vertices is D and whose edges are the pairs $\{a, b\}$ such that a and b occur in a tuple of some relation of $\mathbf{D}$. Given two elements a and b of D, the *distance* between a and b is the length of a shortest path between a and b in the Gaifman graph of $\mathbf{D}$. The notion of distance extends to tuples in the usual way.

Given a positive integer r, the r-neighborhood of a in $\mathbf{D}$ is the substructure $\mathcal{N}_r^{\mathbf{D}}(a)$ of $\mathbf{D}$ induced by the elements of D at distance at most r from a, denoted by $N_r^{\mathbf{D}}(a)$. Similarly we define $\mathcal{N}_r^{\mathbf{D}}(\bar{a})$ as the union of the r-neighborhoods of the elements of $\bar{a}$.

Given a graph G with a linear order on its vertices, and two of its vertices a, b, we say that b is weakly r-accessible from a if there exists a path of length at most r between a and b such that b is smaller than all vertices of the path.

A class of graphs C has bounded expansion if for all r, there is a constant N_r, such that for all graphs G of C, there is a linear order on the vertices of G, such that for all vertices a of G, the number of vertices weakly r-accessible from a is bounded by N_r [16]. This is a robust class of graphs with many equivalent definitions [16]. The precise definition will not be important for this paper as we will use this notion via its known algorithmic properties, in particular the fact that constant-delay enumeration algorithms exists for any class of databases with bounded expansion, see Section 3.

It is easy to see that if C has bounded expansion then the class of all subgraphs of all graphs of C also has bounded expansion.

A class C of graphs has *local bounded expansion* if, for any radius r, the class C_r of all subgraphs of all r-neighborhoods of all graphs in C, has bounded expansion [16].

A class C of databases has (local) bounded expansion if the class of their Gaifman graphs has the same property.

Normal form for FO queries. We will make use of Gaifman Normal Form and Gaifman Locality Theorem for FO queries. This is rather classical in this context.

For all r there exists FO queries $\mathrm{dist}_r(y, \bar{x})$ expressing the fact that y is at distance at most r from $\bar{x}$. A query $q(\bar{x})$ is said to be

r-local if all its quantifications are relative to elements at distance at most r from one of its free variables $\bar{x}$. This can be achieved using quantifications of the form $\exists y\ \mathrm{dist}_r(y,\bar{x}) \wedge \cdots$ and $\forall y\ \mathrm{dist}_r(y,\bar{x}) \rightarrow \cdots$.

It is known as Gaifman Normal Form that for any FO query there is an r such that the query is equivalent to a boolean combination of r-local queries and sentences of the form

$$\exists x_1 \dots x_k \Big(\bigwedge_{1 \le i \le j \le k} \mathrm{dist}(x_i, x_j) > 2r \wedge \bigwedge_{1 \le i \le k} \psi(x_i) \Big),$$

where ψ is r-local. A proof can be found for example in [15].

For r-local queries $q(\bar{x})$ it is convenient to refine this normal form in order to know which of the free variables are close together. Any r-local query $q(\overline{x})$ is equivalent to a disjunction of the form:

$$\bigvee_{(\overline{x}_1;\dots;\overline{x}_p) \in P(\overline{x})} \alpha_1(\overline{x}_1) \wedge \dots \wedge \alpha_p(\overline{x}_p) \wedge \tau_r(\overline{x}_1;\dots;\overline{x}_p), \quad (1)$$

where:

- $P(\overline{x})$ is the set of partitions of $\overline{x}$.
- $\alpha_i(\overline{x}_i)$ is r-local.
- $\tau_r(\overline{x}_1;\dots;\overline{x}_p)$ expresses the fact that the distance between $\overline{x}_i$ and $\overline{x}_j$ is bigger than $2r$ and that no refinement of P has this property. We will sometimes refer to τ_r as a *distance type*.

Note that this implies that each α_i is $(r|\overline{x}_i|)$-local around any of its free variables, hence in particular the first one. Notice also that in (1) the disjunction is strict: no two outputs can satisfy two disjuncts. We will use this fact later to restrict our attention to a single disjunct.

Counting and enumeration. The *counting problem* is, given a database $\mathbf{D}$ and a query q, to compute the number of solutions to q over $\mathbf{D}$, i.e. the size of $q(\mathbf{D})$, noted $\#q(\mathbf{D})$.

We will now focus on the *enumeration problem*. An enumeration algorithm for a database $\mathbf{D}$ and a query q is divided into two consecutive phases:

- a preprocessing phase,
- an enumeration phase, outputting one by one and without repetitions the set $q(\mathbf{D})$.

The *preprocessing time* of the enumeration algorithm is the time taken by the preprocessing phase. Its *delay* is the maximum time between any two consecutive outputs.

One can view an enumeration algorithm as a compression algorithm computing a representation of $q(\mathbf{D})$ together with a streaming decompression algorithm. We aim for constant delay and pseudo-linear preprocessing time enumeration algorithms. By this we mean that for all ϵ, there is a preprocessing phase working in time $O(\|D\|^{1+\epsilon})$ and an enumeration phase with constant delay. Note that the multiplicative constants, for both the preprocessing phase and the delay, may depend on q and on ϵ.

All our enumeration procedures will output their tuples in lexicographical order. We will see that this is useful for queries in disjunctive normal form.

For the sake of readability, in the reminder of the paper, we only consider classes of graphs. All results and proofs can be easily adapted to the database case using standard techniques.

3 MAIN RESULTS

We will build on several known results over classes of databases with bounded expansion. The first is a linear time model checking algorithm for sentences:

Theorem 3.1 (Dvorak-Kral-Thomas [5])**.** *Let C be a class of graphs with bounded expansion. Then the model checking problem for FO queries over C can be solved is linear time.*

The second one solves the unary query case:

Theorem 3.2 (Dvorak-Kral-Thomas [5])**.** *Let C be a class of graphs with bounded expansion and let $q(x)$ be a query with one free variable. We can compute the set $q(G)$ in linear time.*

For queries with bigger arities, we cannot hope to evaluate their output in linear time anymore. A constant delay enumeration algorithm after linear preprocessing time has been obtained by Kazana and Segoufin in [12]. We present their result using a stronger statement than enumeration that will be useful for us later. Here $\geq$ is the lexicographical order on tuples over the domain. Recall that the constant factor depends on the query.

Theorem 3.3 (Kazana-Segoufin [12])**.** *Let C be a class of graphs with bounded expansion. Then there is an algorithm such that for all graphs G in C, and for any FO query q with arity $k+1$, after a preprocessing in linear time, on input any tuple $\overline{a}$ of length k, the algorithm enumerate lexicographically with constant delay the set of all $b \in G$ such that $G \models q(\overline{a}, b)$. Moreover, if b is given, we can test whether $G \models q(\overline{a}, b)$ in constant time.*

Our first result extends Theorem 3.3 to classes with local bounded expansion, replacing linear preprocessing time with pseudo-linear preprocessing time:

Theorem 3.4. *Let C be a class of graphs with local bounded expansion. Then there is an algorithm such that for all graphs G in C, and for any FO query q with arity $k+1$, after a preprocessing in pseudo-linear time, on input any tuple $\overline{a}$ of length k, the algorithm enumerate lexicographically with constant delay the set of all $b \in G$ such that $G \models q(\overline{a}, b)$. Moreover, if b is given, we can test whether $G \models q(\overline{a}, b)$ in constant time.*

It immediately yields the constant delay enumeration after pseudo-linear preprocessing time.

Corollary 3.1. *The enumeration of first-order query over class of graphs with local bounded expansion can be done with constant delay, after pseudo-linear preprocessing. Moreover the output tuples are given in lexicographical order.*

Our second result shows that counting the number of solutions can be done in pseudo-linear time.

Theorem 3.5. *Let C be a class of graphs with local bounded expansion and $q(\overline{x})$ be a first-order query. Then for all graphs G in C, we can compute $\#q(G)$ in pseudo-linear time.*

Our proof works by induction on the arity of the query. It uses a partition of the database into representative neighborhoods that we describe next. It then combines this partition with the bounded expansion case.

4 NEIGHBORHOOD COVERS AND PARTITIONS

Because of the definition of local bounded expansion it is natural to examine the neighborhoods of our graphs. However, the sum of the sizes of all neighborhoods could be quadratic in the size of the input, which is too big as we aim for pseudo-linear time algorithms. To overcome this we select some representative neighborhoods that cover the entire graph. The result presented here actually works for the more general notion of nowhere dense[1] graphs and is based on [9].

A (r, s)-*neighborhood cover* of a graph G is a set T of bags $U_1, \ldots, U_\omega$ such that:

- $\forall a \in G,\ \exists \lambda \leq \omega \quad N_r^G(a) \subseteq U_\lambda$
- $\forall \lambda \leq \omega,\ \exists a \in G \quad U_\lambda \subseteq N_s^G(a)$

The *size* of the cover T is the sum of the bag sizes: $\|T\| = \sum_{\lambda \leq \omega} \|U_\lambda\|$. Its *degree* is the number $\delta(T) := \max_{a \in G} |\{\lambda \leq \omega \mid a \in U_\lambda\}|$.

Theorem 4.1 (Grohe et al. [9])**.** *Let C be a nowhere-dense class of graphs. Then for all integers s and for all graph G in C, we can compute in pseudo-linear time a $(s, 2s)$-neighborhood cover of G with a pseudo-constant degree. In particular the size of the neighborhood cover is pseudo-linear.*

Let A be a set of vertices of G. The s-*kernel* of A is the set $K_s(A) := \{a \in A \mid N_s^G(a) \subseteq A\}$.

We deduce from a $(s, 2s)$-neighborhood cover of G a partition of the vertices of G as follows:

$$P_\lambda := K_s^G(U_\lambda) \setminus \bigcup_{\mu < \lambda} K_s^G(U_\mu).$$

It follows from the definitions that the P_λ form a partition of the vertices of G. Moreover, modulo an extra linear preprocessing time, given an element a we have access in constant time to the unique λ such that $a \in P_\lambda$. This is a consequence of the following simple lemma.

Lemma 4.1. *For all graphs G, for all sets A of vertices of G, and for all integers s, $K_s^G(A)$ is computable in time $O(s \cdot \|A\|)$.*

Proof. We prove the lemma by induction on s.

- If $s = 1$, let L be a list initialized empty. Then for each element a of A, we go through every neighbor of a. If we find one that is not in A, we add a to L and we go to the following element of A. At the end, we have $K_1^G(A) = A \setminus L$.
- If $s = i + 1$, from $K_{i+1}^G(A) = K_1^G(K_i^G(A))$ we get $K_s^G(A)$ is in time $O(s \cdot \|A\|)$. □

In the following sections, we will often say:
compute $T = \{(U_1, P_1), \ldots, (U_\omega, P_\omega)\}$ that is the $(s, 2s)$-neighborhood cover paired with the s-kernel partition.

The previous observations can be synthesized in the following corollary.

[1]Nowhere dense requires that for all r, the number of weakly r-accessible nodes is pseudo-constant, instead of constant for bounded expansion.

Corollary 4.1. *Let C be a class of graphs with local bounded expansion. Then for all graphs G in C and for all integers s, we can compute in pseudo-linear time a $(s, 2s)$-neighborhood cover with pseudo-constant degree and the associated s-kernel partition.*

If a $(s, 2s)$-neighborhood cover can be computed efficiently on any nowhere dense class of graphs, a key property of the covers that works only for the local bounded expansion case is that all U_λ are in a class of graphs with bounded expansion. This is because each U_λ is included in the $2s$-neighborhood of some point and the latter has bounded expansion by definition. We can therefore enumerate any FO query on each U_λ using Theorem 3.3 in time $O(\|U_\lambda\|)$, for a total time $O(\sum_{\lambda \leq \omega} \|U_\lambda\|)$, that is, pseudo-linear. We will use this property implicitly in the rest of the paper. Note that this does not solve the general case as some solutions may have parts in different U_λ.

5 MODEL CHECKING

Since every class of graphs with local bounded expansion is nowhere dense, we already know that the model checking problem of first-order sentences over graphs with local bounded expansion can be done in pseudo-linear time [9]. Before that, another proof specific to local bounded expansion was given in [5]. In order to illustrate the tools presented in the previous sections, we give a new proof of this result. As in [5], it is based on the ideas of Frick and Grohe for graphs with local bounded tree-width [7].

Theorem 5.1. *Let C be a class of graphs with local bounded expansion. Given a graph G in C and an FO sentence q, we can decide in pseudo-linear time whether $G \models q$.*

The rest of this section is devoted to the proof of Theorem 5.1. Fix G in C and a FO sentence q. In view of Gaifman Normal Form, we can assume wlog that q is of the form:

$$q := \exists x_1 \ldots x_k \left(\bigwedge_{1 \leq i \leq j \leq k} \operatorname{dist}(x_i, x_j) > 2r \wedge \bigwedge_{1 \leq i \leq k} \psi(x_i) \right),$$

where ψ is r-local for some r.

Our strategy is as follows: we will first compute the set of nodes satisfying ψ and then test whether k of them are far apart from each other. The next lemma takes care of the first step.

Lemma 5.1. *Let C be a class of graphs with local bounded expansion. For all graphs G in C, for all integers r, and for all unary and r-local FO queries ψ, we can compute $\psi(G)$ in pseudo-linear time.*

Proof. Recall that C_s denotes the class of all subgraphs of s-neighborhoods of graphs from C. Fix r, ψ a unary and r-local query and $G \in C$.

We first compute $T = \{(U_1, P_1), \ldots, (U_\omega, P_\omega)\}$, a $(2r, 4r)$-neighborhood cover paired with the $2r$-kernel partition. This can be done in pseudo linear time by Corollary 4.1. We can then view the P_λ as new unary predicates.

For all $\lambda \leq \omega$, we set $\psi_\lambda(x) := \psi(x) \wedge P_\lambda(x)$. Because ψ is r-local, $\psi(G)$ is the disjoint union of all $\psi_\lambda(U_\lambda)$. By definition, $U_\lambda \in C_{4r}$ which has bounded expansion. Consequently, it is possible to compute $\psi_\lambda(U_\lambda)$ in time $O(\|U_\lambda\|)$ by Theorem 3.2. Therefore, we

are able to compute the set $\psi(G)$ in time $O\left(\sum_{\lambda=1}^{\omega} \|U_\lambda\|\right) = O(\|T\|)$ that is pseudo-linear in the size of G. □

Now we want to find k elements far apart in $\psi(G)$. We use a trick found in [7].

Lemma 5.2. *Let C be a class of graphs with local bounded expansion. For all graphs G in C, for all integers r and k, and for all sets A of vertices of G, we can decide in pseudo-linear time whether A contains a subset of k elements that are pairwise at distance more than $2r$.*

Proof. We proceed as follows:

We first compute $T = \{(U_1, P_1), \ldots, (U_\omega, P_\omega)\}$, a $(2r, 4r)$-neighborhood cover paired with the $2r$-kernel partition as in Corollary 4.1.

Let L be a list, initialized as empty.

While A is not empty and $|L| < k$, we select (and remove) an element a in A.

If for all b in L we have: $(b \in P_\lambda \Rightarrow a \notin U_\lambda)$ then we add a in L. Notice that every b belongs to some P_λ, and hence $N_{2r}(b) \subseteq U_\lambda$. If furthermore $a \notin U_\lambda$ then a and b must be at distance more than $2r$.

At the end, we have three different cases:

- 1^{st} case, $|L| = 0$. Then $A = \emptyset$.
- 2^{nd} case, $|L| = k$. Then we are done because all elements of L are far apart from each other by construction.
- 3^{rd} case, $|L| = m$, with $0 < m < k$. Let $L = \{b_1, \cdots, b_m\}$. Notice that $A \subseteq N^G_{4r}(b_1, \ldots, b_m)$. We can see that $H := N^G_{4r}(b_1, \ldots, b_m)$ is in C_{4rm}. Therefore, from Theorem 3.1 it is possible to check in linear time if:

$$H \models \exists\, x_1, \ldots, x_k \bigwedge_{1 \le i \le k} A(x_i) \wedge \bigwedge_{1 \le i < j \le k} \mathrm{dist}(x_i, x_j) > 2r.$$

□

Theorem 5.1 now easily follows from Lemma 5.1 and Lemma 5.2.

6 ENUMERATION

In this section we provide a constant delay enumeration procedure for FO queries over graphs with local bounded expansion. We actually prove a stronger result as stated in Theorem 3.4. Let G be a graph, q a FO query, $k+1$ it's arity, and $\overline{a}$ any tuple from G of length k. We denote by $q(G, \overline{a})$ the set of all $b \in G$ such that $(\overline{a}, b) \in q(G)$.

In the rest of this section we show that for any class C with local bounded expansion, given $G \in C$ and q, after a pseudo-linear time preprocessing, the membership test of $q(G, \overline{a})$ is doable in constant time and the lexicographical enumeration of $q(G, \overline{a})$ is doable with constant delay. Recall that a pseudo-linear preprocessing means that given ϵ there is a preprocessing algorithm working in time $O(\|G\|^{1+\epsilon})$ computing a structure used later for enumeration and membership test. All constants depend on q and ϵ. This proves Theorem 3.4. We proceed by induction on the arity of the query.

Remark 1. Assume q is $q_1 \vee q_2$. Then for all $\overline{a}$ in G, we have that $q(G, \overline{a}) = q_1(G, \overline{a}) \cup q_2(G, \overline{a})$. Therefore if some subprocesses can enumerate lexicographically both $q_1(G, \overline{a})$ and $q_2(G, \overline{a})$, we can easily enumerate $q(G, \overline{a})$ by running the two processes concurrently, pausing the one that is ahead, and output only once a solution produced by both subprocesses. Hence if q is a disjunction of queries, it is enough to prove Theorem 3.4 on each of the disjunct to get the result for q.

Thanks to Gaifman Normal Form, we can assume that the query is a boolean combination of r-local formulas and sentences. By Theorem 5.1 the sentences can be precomputed during the preprocessing phase. We are then left with an r-local query (any boolean combination of r-local queries is a r-local query). Moreover, in view of Remark 1 and Gaifman Theorem for local queries, we can assume without loss of generality that our query q has the form:

$$q(\overline{x}) = \alpha_1(\overline{x}_1) \wedge \ldots \wedge \alpha_p(\overline{x}_p) \wedge \tau_r(\overline{x}_1; \ldots; \overline{x}_p)$$

where the α_i and τ_r satisfy the conditions described in Section 2.

We start with some examples in order to illustrate the difficulty of the task. Assume that the query returns the pairs of blue-red nodes that are sufficiently far apart:

$$q(x, y) := \mathrm{dist}(x, y) > 2r \wedge B(x) \wedge R(y).$$

Given a graph G and a blue node a, we can enumerate $q(G, a)$ as follows.

During the preprocessing phase, thanks to Corollary 4.1, we compute in pseudo-linear time a $(2r, 4r)$-neighborhood cover with its associated partition $\{(U_1, P_1), \ldots, (U_\omega, P_\omega)\}$. Given a, the λ such that $a \in P_\lambda$ can then be obtained in constant time. We will then have two concurrent processes, one will enumerate $q(G, a) \cap U_\lambda$ and the other one $q(G, a) \setminus U_\lambda$.

The easiest one is $q(G, a) \cap U_\lambda$ because it is also equals to $q(U_\lambda, a)$. We can then invoke Theorem 3.3 and enumerate it with constant delay after a preprocessing linear in $\|U_\lambda\|$. As we don't know a, and therefore λ, in advance, we perform that preprocessing for all λ, for a total time linear in $O(\Sigma_\lambda \|U_\lambda\|)$, which is pseudo linear. As the preprocessing in Theorem 3.3 also allow us to test the membership in $q(U_\lambda, a)$ we are done.

For the second set, the first thing to notice is that testing the membership can be done in constant time as it is enough to be a red node and not be in U_λ. The enumeration however is not that easy because we cannot afford to construct for all λ the list of red nodes that are not in U_λ because each such list might has a linear size and there are a linear number of them, leading to a potentially quadratic preprocessing time. We will see in the proof (this is essentially Claim 6.1 and Claim 6.2 bellow) that we can compute in pseudo-linear time a structure allowing us to enumerate $q(G, a) \setminus U_\lambda$ on the fly.

The situation is even worse for higher arities. To see this, assume the query is now

$$q(x_1, x_2, x_3) := B(x_1) \wedge Y(x_2) \wedge R(x_3) \wedge \bigwedge_{1 \le i < j \le 3} \mathrm{dist}(x_i, x_j) > 2r.$$

Given a graph G and a blue node a, a yellow node a' that are both far apart, we want to enumerate $q(G, aa')$.

Let's see what happens when extending the previous reasoning. Given a and a' we derive in constant time λ and λ' such that $a \in P_\lambda$ and $a' \in P_{\lambda'}$. As above we could enumerate with constant delay all red node b outside of U_λ. But if those nodes are certainly far from a some might be close to a'. We can then imagine precomputing the list of all red nodes b outside of $U_\lambda \cup U_{\lambda'}$, but doing so will leads to a cubic preprocessing time. Again, we will see that we can compute in pseudo-linear time a structure allowing us to enumerate

this set. It remains to enumerate the matching solutions within $U_\lambda \cup U_{\lambda'}$. We could use again Theorem 3.3, but we would need a preprocessing linear in $\Sigma_{\lambda,\lambda'}(\|U_\lambda \cup U_{\lambda'}\|)$, which is unfortunately quadratic. To overcome this problem we introduce an intermediate bag V_λ between P_λ and U_λ together with a more complex algorithm based on the positions of a and a' in all the bags we have, that we will describe below.

We now turn to the formal details.

Base case. Assume q is unary. Because q is also r-local, by Lemma 5.1, we can compute $q(G)$ during the preprocessing phase. Once this list has been computed, we just have to read it during the enumeration phase.

Inductive case. Assume now that $q(\bar{x}, y)$ is an r-local query of arity $k+1$. Let $q'(\bar{x})$ be the query $\exists y\, q(\bar{x}, y)$.

We claim that, modulo a pseudo-linear preprocessing, given a tuple $\bar{a}$ we can enumerate with constant delay the (possibly empty) set $q(G, \overline{a})$. Moreover, if an element b is given we can test in constant time whether b is in $q(G, \overline{a})$.

Before proving the claim we show that it implies the constant-delay enumeration of $q(G)$. By induction, $q'(G)$ can be enumerated with constant delay. Then for each $\overline{a}'$ in $q'(G)$ produced this way, we apply the previous claim. The definition of q' ensures that $q(G, \overline{a}')$ is not empty. Therefore at least one tuple will be outputted between two recursive calls, hence the delay of the overall enumeration algorithm is constant. For the membership test, a tuple $(\overline{a}, b)$ is in $q(G)$, if and only if b is in $q(G, \overline{a})$.

In the rest of this section we prove the claim. Recall that $q(\bar{x}, y)$ is of the form:

$$q(\overline{x}, y) = \alpha_1(\overline{x}_1, y) \wedge \ldots \wedge \alpha_p(\overline{x}_p) \wedge \tau_r(\overline{x}_1, y; \ldots; \overline{x}_p).$$

Let $\bar{w} = \overline{x}_2 \cup \ldots \cup \overline{x}_p$. We have:

$$q(\overline{x}, y) = q_1(\overline{x}_1, y) \wedge q_2(\bar{w}) \wedge \tau_r(\overline{x}_1, y; \ldots; \overline{x}_p).$$

We will distinguish two cases, depending whether $\overline{x}_1$ is empty or not. Let k be the arity of q.

Elements far away. We assume here that $\overline{x}_1$ is empty. By Lemma 5.1 we can precompute in pseudo-linear time the set L of nodes satisfying q_1. It remains to compute a structure that given $\overline{a}$ enumerate the elements in L that are at distance $2r$ from $\overline{a}$.

We compute a $(4r, 8r)$-neighborhood cover and the associated $4r$-kernel partition according to Corollary 4.1. We then compute the $2r$-neighborhood V_λ of each P_λ. We now have $T := \{(P_1, V_1, U_1), \ldots, (P_\omega, V_\omega, U_\omega)\}$ such that $N^G_{2r}(P_\lambda) = V_\lambda$ and $N^G_{2r}(V_\lambda) \subseteq U_\lambda$.

At the end, we will have several concurrent enumeration processes. The first one consists of enumerating all b that are in L but not in any V_λ for which there is an a in $\overline{a}$ and $a \in P_\lambda$. Before giving the complete algorithm, we will introduce some tools that will help to enumerate those b.

We define for all vertices b and all sets $I \subseteq \{1, \ldots, \omega\}$ such that $|I| \leq k$ the function.

$$\text{NEXT}(b, I) = \min \left\{ b' \mid b' \geq b \wedge b' \notin \bigcup_{\lambda \in I} V_\lambda \wedge b' \in L \right\}$$

The domain of this function is too big (recall that ω is linear in $\|G\|$) so we cannot compute it. Fortunately, computing only a small part of it will be good enough for our needs. For each vertex b we define by induction the following set $SC(b)$ of elements $I \subseteq \{1, \ldots, \omega\}$ with $|I| \leq k$:

- For all b in G and for all λ with $b \in V_\lambda$, we add $\{\lambda\}$ to $SC(b)$.
- For all b in G, for all I, and for all λ, if $|I| < k$ and $I \in SC(b)$ and $\text{NEXT}(b, I) \in V_\lambda$, then we add $\{I \cup \{\lambda\}\}$ to $SC(b)$.

Our aim is to compute all $\text{NEXT}(b, I)$ for all b and $I \in SC(b)$. We first show that it will be enough to compute in constant time $\text{NEXT}(b, I)$ for all b and I.

Claim 6.1. *Given a vertex b, a set I, and $\text{NEXT}(c, J)$ for all vertices $c > b$ and sets $J \in SC(c)$, then we can compute $\text{NEXT}(b, I)$ in constant time.*

Proof.

- Case 1, $b \in L$ and $b \notin \bigcup_{\lambda \in I} V_\lambda$, then b is $\text{NEXT}(b, I)$.
- Case 2, $b \notin L$ or $b \in \bigcup_{\lambda \in I} V_\lambda$, then let c be the smallest element of L strictly bigger than b. If there is no such c then $\text{NEXT}(b, I) = \text{Null}$, otherwise:
  - Case 2.1, $c \notin \bigcup_{\lambda \in I} V_\lambda$, then c is $\text{NEXT}(b, I)$.
  - Case 2.2, $c \in V_\lambda$ with $\lambda \in I$. Therefore $\{\lambda\} \in SC(c)$. Let J be a maximal (for inclusion) subset of I in $SC(c)$. Since $\{\lambda\} \in SC(c)$, we know that J is non empty. We claim that $\text{NEXT}(c, J) = \text{NEXT}(b, I)$. To see this, assume that $\text{NEXT}(c, J) \in V_\mu$ with $\mu \in I$ hence $|J| < k$, then by definition of $SC(c)$, $J \cup \{\mu\} \in SC(c)$ and J was not maximal. Moreover, by definition of $\text{NEXT}(c, J)$, every point between c and $\text{NEXT}(c, J)$ is either not in L or in one of the V we want to avoid. As all nodes between b and c are not in L, the claim follows. □

We now show that $SC(b)$ is small for all b and that we can compute all of $\text{NEXT}(b, I)$ for all b and $I \in SC(b)$.

Claim 6.2. *For all integers b, $|SC(b)|$ is a pseudo-constant. Moreover, it is possible to compute all $\text{NEXT}(b, I)$ for all vertices b and set $I \in SC(b)$ in pseudo-linear time.*

Proof. We first prove that for all $b \in G$, $|SC(b)|$ is a pseudo-constant. Then we use Claim 6.1 in order to prove that we can compute these pointers by induction.

- By $SC_l(b)$ we denote the subset of $SC(b)$ of elements I with $|I| \leq l$. Let d be the degree of our cover. We have that for all $b \in G$, $|SC_1(b)| = d$. For the same reason, we have that $|SC_{l+1}(b)| = O(d \cdot |SC_l(b)|)$. Therefore, we have that for all $b \in G$, $|SC(b)| = |SC_k(b)| \leq O(d^k)$. Since d is pseudo-constant, $|SC(b)|$ is also pseudo-constant.
- We compute the pointers for b from b_{max} to b_{min} downwards, respectively the biggest and the smallest element of G. Given a b in G, assume we have computed $\text{NEXT}(c, J)$ for all $c > b$ and $J \in SC(c)$. We then compute $\text{NEXT}(b, I)$ for $I \in SC(b)$ using Claim 6.1.
  Here, every pointer was computed in constant time. Since there is only a pseudo-linear number of them, the time required to compute them all is pseudo-linear. □

With those two claims we can enumerate, after a pseudo-linear preprocessing, the set $L \setminus \bigcup_{\lambda \in I} V_\lambda$ for a given I.

We are now ready to conclude the case where $\bar{x}_1$ is empty. The preprocessing phase consists of the following steps:

- 1^{st} step: using the result for queries with lower arity, compute the preprocessing to test whether a tuple is in $q'(G)$. Where $q'(\overline{x}) := \exists y\, q(\overline{x}, y)$.
- 2^{nd} step: compute a $(4r, 8r)$-neighborhood cover and the associated $4r$-kernel partition according to Corollary 4.1.
- 3^{rd} step: compute the $2r$-neighborhood V_λ of each P_λ. We now have $T := \{(P_1, V_1, U_1), \ldots, (P_\omega, V_\omega, U_\omega)\}$ such that $N^G_{2r}(P_\lambda) = V_\lambda$ and $N^G_{2r}(V_\lambda) \subseteq U_\lambda$.
- 4^{th} step: compute $L := q_1(G)$, where $q_1(y) := \exists \bar{x}\ q(\bar{x}, y)$. This can be done in pseudo-linear time by Lemma 5.1.
- 5^{th} step: compute $\text{NEXT}(b, I)$ for all b and $I \in SC(b)$. This can be done in pseudo-linear time by Claim 6.2.
- 6^{th} step: for all $1 \leq l \leq k$, and for all $\lambda \leq \omega$, perform on U_λ the preprocessing phase for the formula:

$$\varphi_{\lambda,l}(x_1, \ldots, x_l, y) := V_\lambda(y) \wedge L(y) \wedge \bigwedge_{i \leq l} \text{dist}(x_i, y) > 2r.$$

  This can be done in time $O(\|U_\lambda\|)$ by Theorem 3.3 because $U_\lambda \in \mathcal{C}_{8r}$.

This is the end of the preprocessing. The total time needed is pseudo-linear.

Now we are given $\overline{a}$, a tuple of k elements of G. Let $\lambda_1, \ldots, \lambda_k$ be such that $a_i \in P_{\lambda_i}$ and $I(\overline{a}) := \{\lambda_1, \ldots, \lambda_k\}$. For all $\lambda \in I(\overline{a})$ we define as $\bar{a}_\lambda$ as the elements of $\overline{a}$ that fall into U_λ and m_λ the number of such elements. We also define $L(\setminus \overline{a}) := L \setminus \bigcup_{\lambda \in I(\overline{a})} V_\lambda$. To prove that the preprocessing allows us to test the membership to $q(G, \overline{a})$ and enumerate it, we show that for all $\bar{a} \in q'(G)$:

$$q(G, \overline{a}) = L(\setminus \overline{a}) \cup \bigcup_{\lambda \in I(\overline{a})} \varphi_{\lambda, m_\lambda}(U_\lambda, \overline{a}_\lambda)$$

In view of Remark 1, it is enough to show this because we can test whether $\overline{a} \in q'(G)$ by step 1, steps 4 and 5 allow us to test the membership to $L(\setminus \overline{a})$ in constant time and enumerate this set with constant delay, and the last step allows us to do the same for every $\varphi_{\lambda, m_\lambda}(U_\lambda, \overline{a}_\lambda)$ for every λ in $I(\overline{a})$.

We now prove the equality by double inclusion. Let b be an element of $q(G, \overline{a})$. It follows that $G \models q'(\overline{a})$ and $G \models q_1(b)$. In particular $b \in L$. We then have two cases. If $b \notin V_\lambda$ for all $\lambda \in I(\overline{a})$ then b is in $L(\setminus \overline{a})$. Otherwise assume $b \in V_\lambda$ for some $\lambda \in I(\overline{a})$. As b is in $(G, \overline{a})$, we have $\text{dist}(a, b) > 2r$ for all a in $\overline{a}$. Therefore, b is in $\varphi_{\lambda, m_\lambda}(U_\lambda, \overline{a}_\lambda)$.

We now prove the other inclusion. Assume that $G \models q'(a)$. We need to show that given b satisfying the right part of the equality, that b is in L and $\text{dist}(a, b) > 2r$ for all a in $\overline{a}$. This is clearly the case when $b \in L(\setminus \overline{a})$. Assume now that b is in $\varphi_{\lambda, m_\lambda}(U_\lambda, \overline{a}_\lambda)$ for some λ in $I(\overline{a})$. By definition of $\varphi_{\lambda, m_\lambda}$, we have that b is in L and b in V_λ. Moreover, for every a in $\overline{a}$ that are in U_λ (recall that this is exactly $\overline{a}_\lambda$), we have $dist(a, b) > 2r$. Using the fact that b is in V_λ, we also have $\text{dist}(a, b) > 2r$ for the a that are not in U_λ. Therefore b is in $q(G, \overline{a})$.

This concludes the correctness of this algorithm and therefore the entire case called *Elements far away*.

Elements nearby. Assume now that $\overline{x}_1$ contains at least one variable. Therefore, there is an $i \leq k$ such that for all tuples $(\overline{a}, b)$ with $G \models q(\overline{a}, b)$, we have that $\text{dist}(a_i, b) < 2r$. We can assume with out loss of generality that $i = 1$. This makes the second case much easier.

The preprocessing phase contains several steps.

- 1^{st} step: using the result for queries with lower arity, compute the preprocessing to test whether a tuple is in $q'(G)$. Where $q'(\overline{x}) := \exists y\, q(\overline{x}, y)$.
- 2^{nd} step: compute a $(2rk, 4rk)$-neighborhood cover and the associated $2rk$-kernel partition according to Corollary 4.1.
- 3^{rd} step: compute the $2r$ neighborhood V_λ of each P_λ. We now have $T := \{(P_1, V_1, U_1), \ldots, (P_\omega, V_\omega, U_\omega)\}$ such that $N^G_{2r}(P_\lambda) = V_\lambda$ and $N^G_{2r(k-1)}(V_\lambda) \subseteq U_\lambda$.
- 4^{th} step: $\forall 1 \leq l < k$, $\forall \lambda \leq \omega$, perform the preprocessing phase on U_λ of the formula:

$$\varphi_{\lambda,l}(\bar{x}_1, x'_1, \ldots, x'_l, y) := V_\lambda(y) \ \wedge\ q_1(\bar{x}_1, y) \ \wedge \bigwedge_{i<l} \text{dist}(x'_i, y) > 2r.$$

  This can be done in time $O(\|U_\lambda\|)$ by Theorem 3.3 because $U_\lambda \in C_{4kr}$.

This is the end of the preprocessing. The total time needed is pseudo-linear.

Now we are given $\overline{a}$, a tuple of k elements of G. Let λ such that a_1 is in P_λ. We define $\overline{a}_1$ the assignment of the variable in $\overline{x}_1$ excluding b. We also define $\bar{c}_\lambda$ as the elements of $\overline{a}$ that fall into U_λ but that are not in $\overline{a}_1$, and let m_λ be the number of such elements. To prove that the preprocessing allows us to test the membership to $q(G, \overline{a})$ and enumerate it, we will show that for $\overline{a} \in q'(G)$ and λ such that $a_1 \in P_\lambda$, we have:

$$q(G, \overline{a}) = \varphi_{\lambda, m_\lambda}(U_\lambda, \overline{a}_1 \overline{c}_\lambda)$$

As in the *Elements far away* case, proving this equality is enough as the first step of the preprocessing allows us to test that $G \models q'(\overline{a})$ in constant time and the fourth step allows us to to test the membership to $\varphi_{\lambda, m_\lambda}(U_\lambda, \overline{a}_1 \overline{c}_\lambda)$ in constant time and enumerate this set lexicographically with constant delay. Only remains to prove the equality.

Let b be in $q(G, \overline{a})$. Then we have that $G \models q'(\overline{a})$. Moreover, $G \models q_1(b, \overline{a}_1)$ hence $U_\lambda \models q_1(b, \overline{a}_1)$ since q_1 is r-local and U_λ contains the r-neighborhood of $(b, \overline{a}_1)$. We also have that $\text{dist}(b, a) > r$ for all a that is not in $\overline{a}_1$. This includes $\overline{c}_\lambda$. Therefore we have $U_\lambda \models \varphi_{\lambda, m_\lambda}(\overline{a}_1, \overline{c}_\lambda, b)$ and b is in $\varphi_{\lambda, m_\lambda}(U_\lambda, \overline{a}_1 \overline{c}_\lambda)$.

We now prove the other inclusion. Assume that $G \models q'(a)$. Given b in $\varphi_{\lambda, m_\lambda}(U_\lambda, \overline{a}_1 \overline{c}_\lambda)$, we only have to show that $G \models q_1(\overline{a}_1, b)$ and $\text{dist}(a, b) > 2r$ for all a that are in $\overline{a}$ but not in $\overline{a}_1$. As b is in $\varphi_{\lambda, m_\lambda}(U_\lambda, \overline{a}_1 \overline{c}_\lambda)$, we have that $U_\lambda \models q_1(a_1, b)$ hence $G \models q_1(\overline{a}_1, b)$ by locality of q_1. Moreover, for every a in $\overline{a}$ that are in U_λ but not in $\overline{a}_1$, i.e in $a \in \overline{c}_\lambda$, we have by construction $dist(a, b) > 2r$. Using

the fact that b is in V_λ, we also have $\text{dist}(a,b) > 2r$ for the a that are not in U_λ. Therefore b is in $q(G,\overline{a})$ as desired.

This concludes the correctness of this algorithm, the case *Elements nearby* and the proof of Theorem 3.4.

Besides constant delay enumeration, Theorem 3.4 has another interesting immediate corollary. Modulo a pseudo-linear time preprocessing we can test, given a tuple $\overline{a}$ in constant time, whether it belong to $q(G)$ or not:

Corollary 6.1. *Let C be a class of databases with local bounded expansion. Then for all graph G in C, after a pseudo-linear time preprocessing, we can, given a tuple $\overline{a}$, decide in constant time whether it belongs to $q(G)$ or not.*

7 COUNTING

In this section we consider the counting problem which is to compute, given G and q, the size of $q(G)$, denoted by $\#q(G)$. We aim at computing $\#q(G)$ in pseudo-linear time.

Remark 2. Assume q is $q_1 \vee q_2$ and that q_1 and q_2 have no common solution, i.e. the disjunction is strict. Then $\#q(G) = \#q_1(G) + \#q_2(G)$. Hence if q is a strict disjunction of queries, it is enough to prove Theorem 3.5 on each of the disjunct to get the result for q.

Again we will build on the bounded expansion case:

Theorem 7.1 (Kazana, Segoufin [12])**.** *Let C be a class of graphs with bounded expansion and $q(\overline{x})$ be a FO query. Then, for all graphs G in C, we can compute $\#q(G)$ in linear time.*

The rest of the section is dedicated to the proof of Theorem 3.5.

Thanks to Gaifman Normal Form, we can assume that the query is a boolean combination of r-local formulas and sentences. By Theorem 5.1 the sentences can be precomputed during the preprocessing phase. We are then left with a r-local query (any boolean combination of r-local queries is a r-local query). Moreover, in view of Remark 2 and Gaifman Theorem for local queries, we can assume without loss of generality that our query q has the form:

$$q(\overline{x}) = \alpha_1(\overline{x}_1) \wedge \ldots \wedge \alpha_p(\overline{x}_p) \wedge \tau_r(\overline{x}_1;\ldots;\overline{x}_p)$$

where the α_i and τ_r satisfy the conditions described in Section 2.

The proof goes by induction on p, which is the number of connected components of the distance-type τ.

We first give a small example in order to give a hint of how the induction works.

Consider again the query returning the pairs of blue-red nodes that are far apart:

$$q(x,y) := \text{dist}(x,y) > 2r \wedge B(x) \wedge R(y).$$

In this case, there are two connected components. In order to count the number of solutions, we multiply the number of blue nodes by the number of red nodes and we subtract from the result the number of blue-red nodes that are at distance smaller than $2r$. Those three numbers correspond to the number of solutions of three queries with only one connected component in their distance type each, hence we can proceed by induction. This is essentially what we do in the general case.

We now give the details. We start with the base case followed by the inductive case.

Let $G \in C$, and $q(\overline{x}) = \alpha_1(\overline{x}_1) \wedge \ldots \wedge \alpha_p(\overline{x}_p) \wedge \tau_r(\overline{x}_1;\ldots;\overline{x}_p)$

- If $p = 1$. In this case, if $\overline{a} \in q(G)$, then $N_r^G(\overline{a}) \subseteq N_{2rk}^G(a_1)$.
  - 1^{st} step: compute a $(2rk, 4rk)$-neighborhood cover and the associated $2rk$-kernel partition according to Corollary 4.1. We now have:
    $T := \{(P_1,U_1),\ldots,(P_\omega,U_\omega)\}$ such that $N_{2rk}^G(P_\lambda) \subseteq U_\lambda$.
  - 2^{nd} step: for all $\lambda \leq \omega$, let $\varphi_\lambda(\overline{x}) := q(\overline{x}) \wedge x_1 \in P_\lambda$.
    We have that $q(G) = \bigcup_{1 \leq i \leq \omega} \varphi_\lambda(U_\lambda)$. Moreover, the union is disjoint. Therefore:
    $$\#q(G) = \sum_{i=1}^{\omega} \#\varphi_\lambda(U_\lambda)$$
    Since for all λ, $U_\lambda \in C_{4kr}$, we can compute $\#\varphi_\lambda(U_\lambda)$ in time $\|U_\lambda\|$ using Theorem 7.1. Therefore, we can compute $\#q(G)$ in total time $O(\sum_{i=1}^{\omega}(\|U_\lambda\|)) = O(\|T\|)$, that is pseudo-linear in the size of G.

- If $p > 1$. Let $\bar{w} = (\overline{x}_2,\ldots,\overline{x}_p)$. Consider the following three queries:

$$q_1(\overline{x}_1) := \alpha_1(\overline{x}_1) \wedge \tau_r(\overline{x}_1),$$
$$q_2(\bar{w}) := \alpha_2(\overline{x}_2) \wedge \ldots \wedge \alpha_p(\overline{x}_p) \wedge \tau_r(\overline{x}_2;\ldots;\overline{x}_p),$$
$$q_3(\bar{x},\bar{w}) := q_1(\overline{x}_1) \wedge q_2(\bar{w}) \wedge \text{dist}(\overline{x}_1,\bar{w}) \leq 2r.$$

We have that

$$G \models q(\overline{ab}) \iff q_1(\overline{a}) \wedge q_2(\overline{b}) \wedge \text{dist}(\overline{a},\overline{b}) > 2r,$$

hence

$$q(G) = q_1(G) \times q_2(G) \setminus \{\overline{a},\overline{b} \in G \mid q_1(\overline{a}) \wedge q_2(\overline{b}) \wedge \text{dist}(\overline{a},\overline{b}) \leq 2r\},$$

which is

$$q(G) = q_1(G) \times q_2(G) \setminus q_3(G).$$

Since

$$q_3(G) \subseteq q_1(G) \times q_2(G),$$

it follows that

$$\#q(G) = \#q_1(G) \cdot \#q_2(G) - \#q_3(G).$$

It is easy to see that both q_1 and q_2 have less than p connected components in their distance type. Therefore, by the induction assumption we can compute $\#q_1(G)$ and $\#q_2(G)$ in pseudo linear time. We now have to compute $\#q_3(G)$.

We say that $(\overline{x}'_1;\ldots;\overline{x}'_{p'}) \in \Pi(\overline{x}_1;\ldots;\overline{x}_p)$ if and only if:

- $(\overline{x}'_1,\ldots,\overline{x}'_{p'})$ is a partition of $\overline{x}$ with $p' < p$,
- $\overline{x}_1 \subsetneq \overline{x}'_1$,
- $\forall 1 < j \leq p'$, there is a $i > 1$ such that $\overline{x}'_j = \overline{x}_i$.

Basically, $\overline{x}'_1$ is the collapse of $\overline{x}_1$ and at least one of the $\overline{x}_i$. The other $\overline{x}_i$ remain unaltered.

Given $(\overline{x}'_1;\ldots;\overline{x}'_{p'})$, we define:
$\alpha'_1(\overline{x}'_1) = \bigwedge_{i \in I} \alpha_i(\overline{x}_i)$ where $I := \{i \leq p \mid \overline{x}_i \subset \overline{x}'_1\}$,
$\alpha'_j(\overline{x}'_j) = \alpha_i(\overline{x}_i)$ where $\overline{x}_i = \overline{x}'_j \quad \forall 1 < j \leq p'$.
It follows from those definitions that:

$$q_3(\overline{x}) = \bigvee_{(\overline{x}'_1;\ldots;\overline{x}'_{p'})\in\Pi(\overline{x}_1;\ldots;\overline{x}_p)} \alpha'_1(\overline{x}_1)\wedge\ldots\wedge\alpha'_{p'}(\overline{x}'_{p'})\wedge\tau_r(\overline{x}'_1;\ldots;\overline{x}'_{p'}).$$

Moreover these disjunctions are strict. Therefore, with Remark 2:

$$\#q_3(G) = \sum_{(\overline{x}'_1;\ldots;\overline{x}'_{p'})\in\Pi(\overline{x}_1;\ldots;\overline{x}_p)} \#\Big(\alpha'_1(\overline{x}_1)\wedge\ldots\wedge\alpha'_{p'}(\overline{x}'_{p'})\wedge\tau_r(\overline{x}'_1;\ldots;\overline{x}'_{p'})\Big).$$

Since every query present here has less than p connected components in its distance type, we can by induction count the number of solutions for each of them in pseudo-linear time. There is only a constant number of queries involved in this sum, therefore $\#q_3(G)$ is computable in pseudo-linear time.

As $\#q_1(G)$ and $\#q_2(G)$ are already computed, we can compute $\#q(G) = \#q_1(G)\cdot\#q_2(G) - \#q_3(G)$.

The total time needed was pseudo-linear. This concludes the proof.

8 CONCLUSION

We have shown how to efficiently process first-order queries over classes of graphs with locally bounded expansion. We did not explicitly mention the constant factors. These are not very good. Even in the bounded expansion case the constant factor is a tower of exponentials whose height depends on the size of the query. Moreover, an elementary constant factor is not reachable (unless FPT = AW[$*$]) even for unranked trees [8].

The results state the existence of an enumeration procedure for all ϵ. A uniform version of this statement would require that the procedure is computable from ϵ. It is indeed the case if the class of local bounded expansion is "effective", see [16] for the precise definition.

An improvement of our work will be to extend the results for the counting and enumeration problems to nowhere-dense structures. On those structures, the model checking can be done in pseudo-linear time [9]. There is therefore hope to find good algorithms for the other problems. However, this remains future work.

REFERENCES

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] Guillaume Bagan. MSO queries on tree decomposable structures are computable with linear delay. In *Computer Science Logic (CSL'06)*, 2006.

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

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

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

[6] Markus Frick. Generalized model-checking over locally tree-decomposable classes. *Theory Comput. Syst.*, 37(1):157–191, 2004.

[7] Markus Frick and Martin Grohe. Deciding first-order properties of locally tree-decomposable structures. *J. ACM*, 48(6):1184–1206, 2001.

[8] 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.

[9] Martin Grohe, Stephan Kreutzer, and Sebastian Siebertz. Deciding first-order properties of nowhere dense graphs. In *Symp. on Theory of Computing (STOC)*, 2014.

[10] Mamadou Moustapha Kanté. *Graph Structurings: Some Algorithmic Applications. (Structurations de Graphes: Quelques Applications Algorithmiques).* PhD thesis, University of Bordeaux, France, 2008.

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

[12] Wojciech Kazana and Luc Segoufin. Enumeration of first-order queries on classes of structures with bounded expansion. In *Symp. on Principles of Database Systems (PODS)*, 2013.

[13] Wojciech Kazana and Luc Segoufin. Enumeration of monadic second-order queries on trees. *ACM Trans. Comput. Log.*, 14(4), 2013.

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

[15] Leonid Libkin. *Elements of Finite Model Theory.* Springer, 2004.

[16] Jaroslav Nešetřil and Patrice Ossona de Mendez. On nowhere dense graphs. *Eur. J. Comb.*, 32(4):600–617, 2011.